DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 3

Kwalifikacja E.14 Tworzenie aplikacji internetowych



Podręcznik do nauki zawodu **technik informatyk**

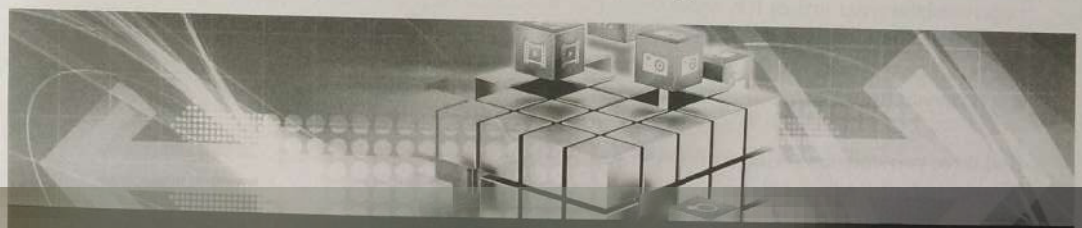
Jolanta Pokorska

Helion Edukacja

DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 3

Kwalifikacja E.14
Tworzenie aplikacji internetowych



Podręcznik do nauki zawodu
technik informatyk

Jolanta Pokorska

Helion Edukacja

Spis treści

Wstęp ... 5

Rozdział 1. Podstawy programowania ... 9
1.1. Podstawowe pojęcia ... 9
1.2. Algorytmy ... 14
1.3. Narzędzia programistyczne ... 25
1.4. Etapy tworzenia programu ... 26
1.5. Dokumentacja programu ... 28
1.6. Istota programowania obiektowego ... 28

Rozdział 2. Aplikacje internetowe ... 37
2.1. Wprowadzenie ... 37
2.2. Narzędzia aplikacji internetowych ... 38
2.3. Struktura aplikacji internetowej ... 39
2.4. Projektowanie aplikacji internetowych ... 42
2.5. Framework — platforma programistyczna ... 44

Rozdział 3. Skrypty po stronie klienta — JavaScript ... 47
3.1. Wprowadzenie ... 47
3.2. Struktura języka JavaScript ... 48
3.3. Składnia języka JavaScript ... 50
3.4. Instrukcje sterujące ... 56
3.5. Funkcje ... 63
3.6. Obiekty ... 70
3.7. Obiekty wbudowane JavaScript ... 75
3.8. Obiekty DOM ... 83
3.9. Obsługa zdarzeń ... 95
3.10. Wykorzystanie skryptów na stronie internetowej ... 99
3.11. Walidacja formularzy ... 106
3.12. Testowanie skryptów ... 115

Rozdział 4. Biblioteka jQuery ... 119
4.1. Opis biblioteki ... 119
4.2. Zdarzenia biblioteki jQuery ... 128

4.3. Zastosowanie biblioteki jQuery na stronie internetowej ... 131
4.4. Walidacja formularzy ... 147

Rozdział 5. Ajax ... 153
5.1. Wprowadzenie ... 153
5.2. Obiekt XMLHttpRequest ... 154
5.3. Żądania Ajax ... 157

Rozdział 6. Serwery WWW ... 163
6.1. Serwer Apache ... 163
6.2. Serwer IIS ... 167

Rozdział 7. Język PHP ... 169
7.1. Wprowadzenie ... 169
7.2. Instalowanie PHP ... 171
7.3. Pakiet XAMPP ... 173
7.4. Struktura języka PHP ... 180
7.5. Składnia języka PHP ... 183
7.6. Instrukcje sterujące ... 198
7.7. Funkcje ... 207
7.8. Funkcje wbudowane ... 215
7.9. Pliki i katalogi ... 229
7.10. Obiekty ... 243
7.11. Formularze ... 259
7.12. Pliki cookies i sesje ... 271
7.13. Sieć ... 287
7.14. Bazy danych w PHP ... 290

Rozdział 8. Technologia .NET ... 311
8.1. Założenia platformy .NET ... 311

Rozdział 9. Systemy zarządzania treścią CMS ... 315
9.1. Joomla! ... 316

Rozdział 10. Testowanie i dokumentowanie aplikacji ... 337
10.1. Testowanie aplikacji ... 337
10.2. Zagrożenia ... 338
10.3. Dokumentowanie aplikacji ... 340

Skorowidz ... 347

Wstęp

Podręcznik *Tworzenie aplikacji internetowych* jest trzecim z grupy podręczników przygotowanych dla kwalifikacji E.14. Tworzenie aplikacji internetowych i baz danych oraz administrowanie bazami.

Zawarte w podręczniku *Tworzenie aplikacji internetowych* treści zostały oparte na podstawie programowej kształcenia w zawodach wprowadzonej rozporządzeniem Ministra Edukacji Narodowej z dnia 7 lutego 2012 r.

Obejmują one zagadnienia teoretyczne prowadzące do uzyskania wymienionych w podstawie programowej efektów kształcenia, projekty różnych zadań oraz ich realizację praktyczną. Tak skonstruowany podręcznik pomaga uczniowi w zdobywaniu wymaganej wiedzy oraz umożliwia mu samodzielne poszerzanie umiejętności.

Podręcznik składa się z dziesięciu rozdziałów. Ich budowa pozwala na realizację treści programowych w sposób wybrany przez nauczyciela.

Rozdział 1. „Podstawy programowania", zawiera omówienie podstawowych zagadnień związanych z programowaniem i algorytmiką. Dotyczy efektów związanych z przestrzeganiem zasad programowania. Efektami tymi są: rozpoznawanie podstawowych pojęć programowania, takich jak kod źródłowy, kompilator, interpreter, konsolidacja; definiowanie etapów tworzenia programu komputerowego; projektowanie struktury programu, analizowanie struktury algorytmu i tworzenie schematów algorytmów; identyfikowanie schematów blokowych, list kroków i drzew decyzyjnych.

Rozdział 2. „Aplikacje internetowe", zawiera omówienie podstawowych zagadnień związanych z użyciem aplikacji internetowych. Dotyczy efektów związanych z tworzeniem oraz wykorzystywaniem aplikacji webowych. Efektami tymi są: rozpoznawanie technologii działających po stronie klienta i serwera; identyfikowanie struktury aplikacji internetowej; rozpoznawanie wzorców projektowych; identyfikowanie narzędzi programistycznych oraz technologii aplikacji internetowych.

Rozdział 3. „Skrypty po stronie klienta — JavaScript", zawiera omówienie zagadnień związanych ze stosowaniem skryptów wykonywanych po stronie klienta przy tworzeniu aplikacji internetowych. Dotyczy efektów związanych ze stosowaniem i tworzeniem własnych funkcji, procedur, obiektów i metod w języku JavaScript oraz testowaniem tworzonych aplikacji i modyfikowaniem ich kodu źródłowego. Efektami tymi są: identyfikowanie operatorów arytmetycznych, bitowych i logicznych oraz rozpoznawanie instrukcji, metod, procedur i obiektów języka JavaScript; stosowanie odpowiednich instrukcji, procedur i metod języka JavaScript do określonych zadań; definiowanie obiektów w języku JavaScript i przypisywanie im wartości; definiowanie własnych procedur i metod; analizowanie poprawności tworzonych obiektów i metod.

Rozdział 4. „Biblioteka jQuery", zawiera omówienie zagadnień związanych z wykorzystaniem biblioteki jQuery do tworzenia własnych aplikacji. Dotyczy efektów związanych ze stosowaniem funkcji, metod i obiektów biblioteki jQuery, tworzeniem własnych funkcji, obiektów i metod z użyciem biblioteki jQuery oraz stosowaniem ich na stronie internetowej. Efektami tymi są: identyfikowanie i stosowanie metod, obiektów i zdarzeń biblioteki jQuery do tworzenia efektów na stronie internetowej oraz do tworzenia i walidacji formularzy.

Rozdział 5. „Ajax", zawiera omówienie zagadnień związanych z wykorzystaniem technologii Ajax do tworzenia aplikacji internetowych realizujących zadania po stronie serwera. Dotyczy efektów związanych ze stosowaniem funkcji, metod i obiektów technologii Ajax. Efektami tymi są: rozpoznawanie zasad funkcjonowania technologii Ajax; tworzenie własnych obiektów XMLHttpRequest; rozpoznawanie i stosowanie żądań wysyłanych do serwera oraz funkcji zwrotnych.

Rozdział 6. „Serwery WWW", zawiera omówienie zagadnień związanych z działaniem serwerów stron internetowych oraz zasadami umieszczania opracowanych aplikacji w internecie. Dotyczy efektów związanych z instalowaniem oraz konfigurowaniem serwerów WWW. Efektami tymi są: instalowanie i konfigurowanie serwera Apache w systemach Linux oraz Windows oraz instalowanie i konfigurowanie serwera IIS w systemie Windows.

Rozdział 7. „Język PHP", zawiera omówienie podstawowych zagadnień związanych z przestrzeganiem zasad programowania oraz stosowaniem skryptów wykonywanych po stronie serwera przy tworzeniu aplikacji internetowych. Dotyczy efektów związanych z wykorzystaniem wbudowanych typów danych, stosowaniem funkcji, procedur, obiektów i metod w języku PHP, z tworzeniem własnych funkcji, procedur, obiektów i metod oraz testowaniem tworzonej aplikacji i modyfikowaniem jej kodu źródłowego. Efektami tymi są: instalowanie i konfigurowanie pakietu XAMPP, identyfikowanie operatorów arytmetycznych, bitowych i logicznych; tworzenie własnych typów danych, rozpoznawanie instrukcji, metod, procedur i obiektów języka PHP; stosowanie instrukcji, procedur i metod języka PHP do określonych zadań; definiowanie obiektów w języku PHP i przypisywanie im wartości; definiowanie własnych procedur i metod; stosowanie operacji na plikach i katalogach, tworzenie i walidacja formularzy, wykorzystywanie plików cookies i sesji do identyfikacji użytkownika oraz bazy danych do przetwarzania dużych ilości informacji gromadzonych na potrzeby strony internetowej, analizowanie poprawności tworzonych aplikacji.

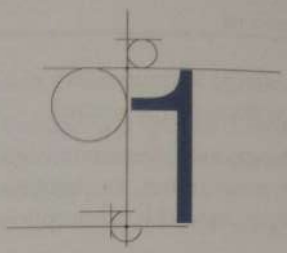
Rozdział 8. „Technologia .NET", zawiera omówienie zagadnień związanych z działaniem platformy .NET oraz narzędziami programistycznymi wykorzystywanymi na tej platformie do tworzenia aplikacji realizujących zadania po stronie serwera. Dotyczy efektów związanych z instalowaniem oraz konfigurowaniem serwerów WWW. Efektami tymi są: identyfikowanie założeń platformy .NET oraz funkcji i możliwości jej stosowania w aplikacjach internetowych realizujących zadania po stronie serwera.

Rozdział 9. „Systemy zarządzania treścią CMS", zawiera omówienie zagadnień związanych z tworzeniem aplikacji internetowych realizujących zadania po stronie serwera

oraz zamieszczaniem opracowanych aplikacji w internecie. Dotyczy efektów związanych ze stosowaniem systemów zarządzania treścią w tworzeniu aplikacji internetowych. Efektami tymi są: tworzenie aplikacji internetowych opartych na szablonach systemu Joomla!.

Rozdział 10. „Testowanie i dokumentowanie aplikacji", zawiera omówienie zagadnień związanych z dokumentowaniem tworzonej aplikacji oraz testowaniem i modyfikowaniem jej kodu źródłowego. Dotyczy efektów związanych z przeprowadzaniem i analizą testów tworzonej aplikacji, analizą zagrożeń oraz dokumentowaniem aplikacji. Efektami tymi są: rozpoznawanie rodzajów testów, identyfikowanie zagrożeń i identyfikowanie sposobów zapobiegania atakom, identyfikowanie sposobów dokumentowania tworzonej aplikacji.

Do książki udostępniamy również za pośrednictwem naszego serwera FTP dodatek, poświęcony językowi C# (*ftp://ftp.helion.pl/przyklady/e14te3.zip*). Zawiera on omówienie podstawowych zagadnień związanych z przestrzeganiem zasad programowania oraz stosowaniem języka C# do tworzenia aplikacji internetowych realizujących zadania po stronie serwera. Dotyczy efektów związanych ze stosowaniem funkcji, procedur, obiektów i metod w języku C#, z tworzeniem własnych funkcji, procedur, obiektów i metod oraz testowaniem tworzonej aplikacji i modyfikowaniem jej kodu źródłowego. Efektami tymi są: identyfikowanie operatorów arytmetycznych, bitowych i logicznych; tworzenie własnych typów danych, rozpoznawanie instrukcji, metod, procedur i obiektów języka C#, definiowanie obiektów i przypisywanie im wartości; definiowanie własnych procedur i metod; wykorzystywanie środowiska programistycznego (edytor, kompilator, debuger), pobieranie informacji z aplikacji internetowych i wczytywanie danych do baz danych oraz stosowanie różnych metod uwierzytelnienia połączeń z bazą danych i analizowanie poprawności tworzonych aplikacji.



Podstawy programowania

1.1. Podstawowe pojęcia

Program jest to zbiór poleceń zapisanych w określonym języku programowania zgodnie z obowiązującymi w tym języku regułami. Programy są tworzone przez programistów podczas procesu programowania.

Programowanie to proces tworzenia i testowania programu. Kod źródłowy programu jest napisany w języku programowania z użyciem określonych reguł.

Język programowania musi posiadać ściśle zdefiniowane reguły syntaktyczne i semantyczne, które opisują, jak należy budować poprawne wyrażenia.

Słowa kluczowe to zarezerwowane słowa, które w danym języku programowania mają ściśle określone znaczenie i mogą zostać użyte tylko zgodnie z ich przeznaczeniem. Słowami kluczowymi są między innymi nazwy poleceń.

Kod źródłowy to ciąg instrukcji i deklaracji zapisany w języku programowania. Opisuje operacje, jakie powinien wykonać komputer. Kod źródłowy programu jest zapisywany w postaci tekstu. Najczęściej składa się z jednego lub kilku plików tekstowych. Kod źródłowy jest przetwarzany za pomocą kompilatora na kod maszynowy lub kod pośredni. Możliwe jest również wykonywanie kodu źródłowego w locie za pomocą programu zwanego interpreterem.

Translator to program służący do tłumaczenia programu zapisanego w języku programowania z postaci źródłowej do postaci wynikowej. Translatory dzielimy na: kompilatory tłumaczące programy zapisane w językach wysokiego poziomu oraz asemblery tłumaczące programy zapisane w językach symbolicznych.

Kompilator to program służący do tłumaczenia kodu napisanego w języku źródłowym na odpowiadający mu kod w języku wynikowym. Najczęściej jest to program do tłumaczenia kodu źródłowego napisanego w wybranym języku programowania na kod maszynowy.

Interpreter to program, który analizuje kod źródłowy instrukcja po instrukcji i każdy przeanalizowany fragment kodu wykonuje na bieżąco.

Moduł to wydzielony fragment programu komputerowego. Podzielenie dużych programów na moduły ułatwia pracę z programem, szczególnie gdy każdą część programu opracowuje inny programista. Każdy moduł jest kompilowany osobno.

Linker (konsolidator) to narzędzie służące do łączenia przekompilowanych modułów w jeden plik wykonywalny. Odpowiada on za poprawne połączenie modułów ze sobą. Dobry linker powinien umożliwiać tworzenie plików wykonywalnych o różnych formatach i dla różnych systemów.

Konsolidacja to proces polegający na połączeniu skompilowanych modułów i utworzeniu pliku wynikowego. Podczas konsolidacji do pliku wynikowego mogą być dołączone biblioteki i inne informacje, dotyczące na przykład formatu pliku wykonywalnego.

Aplikacja to program użytkowy, wykonujący konkretne zadania i oferujący interfejs użytkownika. Określenie *program komputerowy* jest często stosowane zamiennie z określeniem *aplikacja*. Podstawową różnicą między tymi pojęciami jest to, że program komputerowy nie musi oferować żadnego interfejsu użytkownika.

Aplikacja internetowa (ang. *web application*) lub aplikacja webowa to program komputerowy, który pracuje na serwerze i komunikuje się z użytkownikiem poprzez sieć komputerową z wykorzystaniem przeglądarki internetowej. Niektóre aplikacje webowe mogą również działać offline. Na przykład aplikacja gromadzi pewne dane na lokalnym komputerze, a gdy jest możliwy dostęp do internetu, wysyła zgromadzone informacje do serwera i zapisuje je w bazie danych. Funkcję serwera internetowego może pełnić każdy komputer podłączony do internetu, a więc także komputer osobisty. Najczęściej odgrywa on rolę serwera testowego. Zainstalowana na tym samym komputerze aplikacja pracuje offline, wykorzystując wszystkie zalety aplikacji webowych.

Aplet jest to niewielki program komputerowy, którego wykonanie jest możliwe tylko z poziomu innej aplikacji. Aplety najczęściej są pisane w języku Java lub jako komponenty Active-X i są wykorzystywane na stronach internetowych. Ich zadaniem może być wykonywanie prostych czynności, np. uruchomienie animacji, przeprowadzanie obliczeń. W zaawansowanych zastosowaniach mogą być stosowane, np. jako przeglądarkowe wersje demo w przypadku gier pisanych w Javie lub do podpisywania przelewów w bankowości internetowej.

1.1.1. Języki programowania

Język programowania służy do tworzenia programów komputerowych, których zadaniem jest przetwarzanie danych, wykonywanie obliczeń i algorytmów. Może zawierać konstrukcje składniowe do manipulowania strukturami danych oraz zarządzania przepływem sterowania. Niektóre języki programowania posiadają specyfikację swojej składni i semantyki, inne zdefiniowane są jedynie przez oficjalne implementacje.

Elementy języka:

- *Składnia* — zbiór reguł opisujących sposób definiowania struktur danych, rodzaje dostępnych słów kluczowych i symboli oraz zasady, według których symbole mogą być łączone w większe struktury.
- *Semantyka* — zbiór reguł definiujących znaczenie słów kluczowych i symboli oraz ich funkcji w programie.
- *Typy danych* — określają dostępne typy danych, ich właściwości oraz operacje, które mogą być wykonane na wartościach danego typu.

Przykładami używanych obecnie języków programowania są: C, C++, Objective C, C#, Java, Delphi, PHP, Perl, Python, Ruby.

Języki Java, Python i Ruby są popularne, gdyż pozwalają na bardzo szybkie tworzenie aplikacji. Wydajniejsze języki, takie jak C, C++ czy Delphi, posłużyły do napisania programów biurowych typu edytory tekstu czy grafiki.

Języki PHP i Java są popularne w programach korzystających z baz danych. Python bywa wykorzystywany w administrowaniu systemami i na stronach WWW.

Wybór języka programowania może zależeć od przeznaczenia końcowej aplikacji, indywidualnych upodobań lub strategii firmy tworzącej oprogramowanie. Najlepszym rozwiązaniem jest wybór języka programowania najbardziej dostosowanego do rozwiązywanego zadania i istniejącej infrastruktury.

Programy są tworzone i zapisywane w formacie tekstowym. Rozszerzenie pliku zależy od wybranego języka programowania. Identyfikuje ono język, w którym program został napisany, np. *pas* dla języka Pascal, *c* dla języka C, *cpp* dla C++, *java* dla Javy, *cs* dla C#. Niezależnie od rozszerzenia plik zawierający program można otworzyć w dowolnym edytorze tekstu.

1.1.2. Kompilatory

Proces automatycznego tłumaczenia kodu źródłowego zapisanego w języku programowania na kod maszynowy nazywamy **kompilacją**. Jeżeli w wyniku kompilacji zostaną wykryte niezgodności z regułami języka programowania, to kompilacja zostanie przerwana, a użytkownik będzie poinformowany o rodzaju błędu.

W trakcie pracy kompilatory wykonują większość podanych niżej operacji (lub wszystkie):

- wstępne przetwarzanie kodu,
- analiza leksykalna,
- analiza składniowa,
- analiza semantyczna,
- optymalizacja kodu wynikowego,
- generowanie kodu.

Wstępne przetwarzanie kodu

Zazwyczaj przed przekazaniem programu na wejście kompilatora następuje wstępne przetwarzanie kodu źródłowego za pomocą preprocesora. Zadaniem preprocesora jest wyszukanie w kodzie źródłowym specjalnych poleceń i ich wykonanie. W wyniku wykonania tych poleceń następuje zamiana jednego tekstu na inny, dołączenie zawartości innego pliku lub pominięcie fragmentu tekstu. Polecenia preprocesora zwykle zaczynają się znakiem #, na przykład: `#include`, `#define`.

Analiza leksykalna

Wczytywane dane kodu źródłowego muszą mieć określoną składnię. Przed rozpoczęciem przetwarzania danych kompilator musi zanalizować składnię programu. Analiza leksykalna polega na sprawdzeniu, czy nie występują niedozwolone znaki, oraz na podzieleniu tekstu na jednostki leksykalne odseparowane białymi znakami (np. spacja, tabulator, cr) oraz wyróżnieniu słów kluczowych, operatorów i łańcuchów znaków. Analiza leksykalna nazywana jest również analizą liniową lub skanowaniem.

Analiza syntaktyczna (składniowa)

Analizator składniowy (tzw. parser) sprawdza kod źródłowy w celu rozpoznania jego struktury składniowej i ustalenia, czy dane są poprawne składniowo (symbole leksykalne są grupowane w wyrażenia gramatyczne zgodne z gramatyką języka). Analiza składniowa nazywana jest również analizą hierarchiczną.

Analiza semantyczna (znaczeniowa)

W trakcie analizy semantycznej sprawdzana jest poprawność programu na poziomie znaczenia poszczególnych instrukcji oraz programu jako całości. Podczas analizy semantycznej jest sprawdzane, czy program może zostać jednoznacznie skompilowany, ponieważ niektóre konstrukcje, mimo że dopuszczalne przez gramatykę języka, mogą być niepoprawne.

W skład analizy semantycznej wchodzą następujące elementy:

- *kontrola typów* — sprawdzanie poprawności typów (również sprawdzanie, czy identyfikatory zostały zadeklarowane); kontrola typów ma miejsce tylko w językach z silnym typowaniem;
- *kontrola poprawności instrukcji* — sprawdzanie, czy instrukcje i wyrażenia mają sens w kontekście, w którym zostały użyte;
- *kontrola nazw* — sprawdzanie, czy nazwy jednoznacznie identyfikują funkcje, etykiety i inne konstrukcje języka programowania.

Optymalizacja kodu wynikowego

Podczas optymalizacji kodu wynikowego wykonywane są czynności mające na celu przyspieszenie działania programu lub zmniejszenie jego objętości. Kompilator podczas optymalizacji dokonuje analizy przepływu sterowania i przepływu danych oraz

przekształca kod do postaci optymalnej. Podczas przekształceń musi zostać zachowana semantyka programu (przed optymalizacją i po optymalizacji program musi dawać takie same wyniki). Nie zawsze efekty optymalizacji kodu są zadowalające.

Generowanie kodu

W wyniku kompilacji na podstawie kodu źródłowego generowany jest kod języka niskiego poziomu, kod wykonywalny (np. C, C++) lub kod w języku pośrednim (np. Java, C#).

Podsumowanie

Kompilacja kodu źródłowego pozwala na uzyskanie wyższej wydajności programu, ale wygenerowany kod maszynowy jest ściśle powiązany z platformą sprzętową.

Użycie interpretatora pozwala uzyskać większą przenośność programów oraz niezależność od platformy i systemu operacyjnego. Jednak sposób wykonywania kodu źródłowego wpływa negatywnie na jego wydajność.

1.1.3. Języki skryptowe

Skrypt to program napisany w języku skryptowym, który jest wykonywany wewnątrz aplikacji.

Język skryptowy to język programowania służący do wykonywania wyspecjalizowanych czynności. Języki skryptowe są tworzone z myślą o interakcji z użytkownikiem. Często są wykorzystywane do zadań administracyjnych. Języki skryptowe bywają również osadzane w programach w celu zautomatyzowania powtarzających się czynności. Są wykorzystywane do tworzenia dynamicznych stron internetowych. Osadzane w grach komputerowych służą do sterowania przebiegiem gry.

Języki skryptowe mogą służyć do pisania zaawansowanych aplikacji, ale najczęściej są wykorzystywane do szybkiego tworzenia niewielkich skryptów, pozwalających na dynamiczne wyświetlanie stron internetowych lub zapamiętywanie i przetwarzanie wprowadzonych danych. Do popularnych języków skryptowych należą:

- JavaScript,
- PHP,
- ASP,
- Perl.

1.1.4. Biblioteki standardowe

Większość języków programowania udostępnia biblioteki standardowe, które zawierają definicje typowych operacji wykonywanych w programach. Należą do nich najczęściej:

- operacje na ciągach tekstowych,
- operacje na typach danych oraz funkcje do zarządzania nimi,
- obsługa wejścia-wyjścia,

- obsługa plików,
- obsługa wielowątkowości,
- zarządzanie pamięcią.

Biblioteki mogą być statyczne (dołączane do programu na etapie konsolidacji) lub dynamiczne (ładowane przez program na etapie wykonywania).

1.2. Algorytmy

Algorytm to zestaw ściśle określonych czynności, prowadzących do wykonania pewnego zadania. Określa sposób rozwiązania problemu i ma zastosowanie w różnych dziedzinach. Języki programowania to narzędzia, które bardzo dobrze nadają się do zapisu algorytmów. Aby napisać dobry program komputerowy, należy opracować skuteczny algorytm i zdefiniować dla niego odpowiednie struktury danych.

Algorytm przetwarzania danych powinien przy takim samym zbiorze danych wejściowych zwrócić taki sam wynik. Ale stanie się tak tylko w dokładnie takich samych warunkach i przy tych samych danych pomocniczych. Zwykle przy projektowaniu algorytmu zakłada się, że dane wejściowe są poprawne, ale bywają algorytmy, które oprócz przetwarzania danych również weryfikują dane.

W rzeczywistości tak jak nie każdy problem można rozwiązać, tak nie każdą metodę rozwiązania można zapisać przy użyciu algorytmu. Aby problem mógł być rozwiązany za pomocą komputera, musi zostać zapisany w postaci algorytmu. Wynika to z tego, że komputer potrafi rozwiązywać tylko problemy zdefiniowane w postaci jednoznacznych kroków, czyli algorytmu. Jeżeli nie można zdefiniować problemu w postaci algorytmu, nie ma możliwości rozwiązania go z wykorzystaniem komputera.

Zdefiniowany algorytm może zostać zapisany w wybranym języku programowania. Ale ten sam algorytm może zostać zapisany różnie, w zależności od użytego języka programowania.

Zapis algorytmu w wybranym języku programowania nazywamy **implementacją algorytmu**.

1.2.1. Reprezentacja algorytmów

Algorytm opisujący operacje do wykonania może zostać zapisany w różny sposób. Może to być zapis słowny, lista kroków do wykonania, pseudokod, drzewo algorytmu lub schemat blokowy.

Opis słowny

W opisie słownym operacje, które należy wykonać, są zapisywane za pomocą zwykłego tekstu. Ten sposób zapisu algorytmu stosowany jest we wstępnej fazie opisu problemu, gdy trzeba w sposób ogólny przedstawić operacje do wykonania, bez określania szczegółów dotyczących rozwiązania problemu.

Przykład 1.1

Algorytm obliczania pola trójkąta:

Dane wejściowe: a — podstawa trójkąta, h — wysokość trójkąta.

Dane wyjściowe: s — pole trójkąta.

Oblicz pole trójkąta jako iloczyn podstawy trójkąta i wysokości trójkąta podzielony przez 2.

Lista kroków

Na liście kroków każda operacja, którą należy wykonać, jest zapisywana w postaci numerowanego kroku. Lista kroków pozwala dokładnie zdefiniować cały algorytm.

Przykład 1.2

Algorytm obliczania pola trójkąta:

Krok 1. Wczytaj dane wejściowe: a, h.

Krok 2. Oblicz wynik: pole ← 1/2×a×h.

Krok 3. Wyświetl na ekranie wynik: pole.

Krok 4. Zakończ.

Pseudokod

Pseudokod to opis słowny przypominający zapis kroków algorytmu, który może zawierać instrukcje z języka programowania. Tworząc pseudokod, najczęściej używamy słów języka naturalnego do określenia kroków algorytmu (np. *wczytaj dane*, *oblicz wartość*), ale mogą pojawić się w nim elementy zawierające symboliczny opis działań (np. *a* 5, *powtarzaj ... aż do ...*).

Przykład 1.3

Algorytm obliczania pola trójkąta:

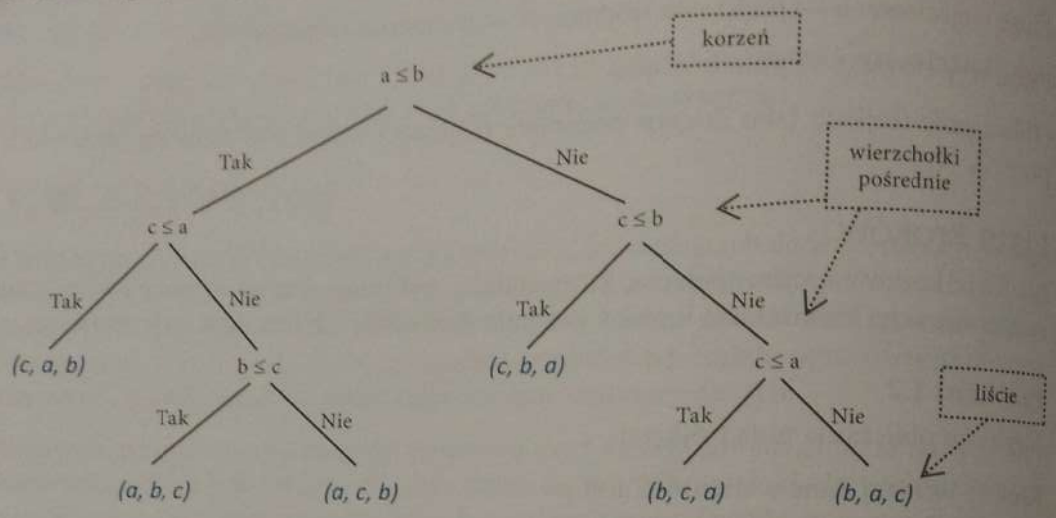
```
INPUT a, h
P = 1/2×a×h
OUTPUT P
EXIT
```

Drzewo algorytmu

Drzewo algorytmu to reprezentacja graficzna algorytmu. W schemacie drzewa wyróżnione są: jeden główny element, tj. korzeń (wierzchołek), który stanowi początek algorytmu, gałęzie (wierzchołki pośrednie), które są reprezentacją wykonywanych operacji, oraz liście (wierzchołki końcowe), które reprezentują otrzymane wyniki. Drzewo algorytmu może zostać przedstawione jako graf.

Przykład 1.4

Drzewo algorytmu porządkowania trzech liczb (rysunek 1.1).



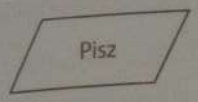
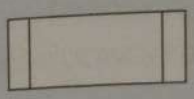
Rysunek 1.1. Przykładowe drzewo algorytmu

Schemat blokowy

W schemacie blokowym operacje, które należy wykonać, są przedstawiane w postaci graficznej z użyciem symboli.

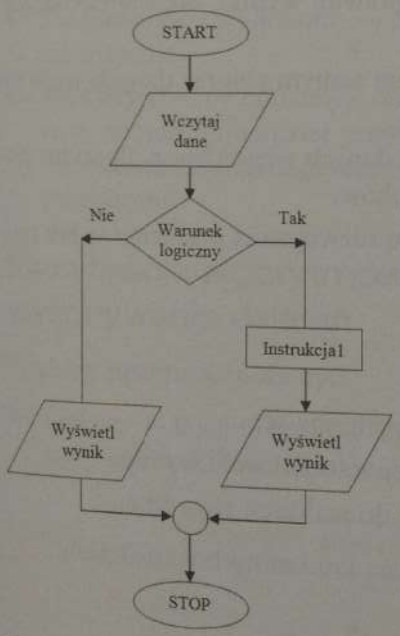
Tabela 1.1. Symbole wykorzystywane do tworzenia schematów blokowych

Symbol	Opis
START	Początek algorytmu, start programu. Od tego miejsca rozpoczyna się wykonywanie operacji.
STOP	Koniec algorytmu, zakończenie programu. W tym miejscu następuje zakończenie wykonywania operacji.
↓	Połączenie między blokami. Wskazuje kolejność wykonywania operacji.
Instrukcja	Wykonanie operacji, blok obliczeniowy. Wewnątrz tego symbolu znajdują się operacje do wykonania.
Czytaj	Wprowadzanie danych. Wewnątrz tego symbolu określamy dane wejściowe, które muszą zostać wczytane.

Symbol	Opis
	Wyprowadzanie danych. Wewnątrz tego symbolu określamy dane wyjściowe, które powinny zostać wyprowadzone jako wynik.
	Warunek logiczny, blok decyzyjny. Umożliwia tworzenie rozgałęzień w algorytmie. Jeżeli warunek jest spełniony, to następuje przejście do gałęzi oznaczonej „Tak", w przeciwnym razie następuje przejście do gałęzi oznaczonej „Nie".
	Proces wstępnie zdefiniowany. Symbol ten oznacza dołączenie podprogramu.
	Łącznik. Odwołanie na stronie. Służy do oznaczenia miejsc łączenia schematu, np. gdyby linie łączące na schemacie musiały się krzyżować.
	Łącznik międzystronicowy. Służy do oznaczenia miejsc łączenia schematu, gdy nie mieści się on na jednej stronie.

Przykład 1.5

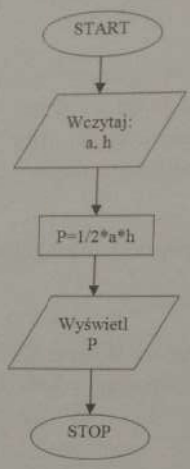
Schemat blokowy (rysunek 1.2).



Rysunek 1.2. Przykład schematu blokowego

Przykład 1.6

Algorytm obliczania pola trójkąta przedstawiony w postaci schematu blokowego (rysunek 1.3):



Rysunek 1.3. Algorytm w postaci schematu blokowego

Badaniem algorytmów zajmuje się algorytmika.

1.2.2. Cechy algorytmów

Podstawowymi cechami algorytmów powinny być:

- *poprawność* — algorytm powinien zwracać poprawne wyniki, odzwierciedlające rzeczywistość;
- *jednoznaczność* — algorytm powinien przy takim samym zbiorze danych wejściowych zwracać takie same wyniki;
- *skończoność* — dla każdego zbioru poprawnych danych wejściowych algorytm powinien zwracać wyniki w skończonej liczbie kroków;
- *efektywność* — algorytm powinien prowadzić do rozwiązania problemu w jak najmniejszej liczbie kroków.

Specyfikacja algorytmu powinna zawierać:

- podanie danych wejściowych;
- określenie wyniku, który algorytm powinien wygenerować;
- określenie warunków, jakie powinny spełniać dane wejściowe i wyniki;
- podanie zmiennych pomocniczych niezbędnych do realizacji algorytmu.

Algorytmy powinny operować na zmiennych, a nie na konkretnych wartościach.

1.2.3. Klasyfikacja algorytmów

Istnieje wiele różnych sposobów podziału algorytmów. Można je dzielić ze względu na:

- metodę konstrukcji algorytmu,
- kolejność wykonywania działań,
- sposób wykonywania operacji,
- obszar zastosowań,
- strukturę danych.

Klasyfikacja algorytmów ze względu na sposób konstruowania algorytmu

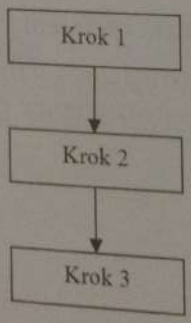
Metody konstruowania algorytmów:

- *Dziel i zwyciężaj* — problem, który należy rozwiązać, jest dzielony na kilka mniejszych, a te znowu są dzielone aż do uzyskania problemów łatwych do rozwiązania. Algorytmy z tej grupy należą do najskuteczniejszych metod rozwiązywania problemów.
- *Programowanie dynamiczne* — metoda podobna do metody *dziel i zwyciężaj*. Problem, który należy rozwiązać, jest dzielony na kilka mniejszych. Wyniki analizy cząstkowych problemów wykorzystuje się do rozwiązania głównego problemu.
- *Metoda zachłanna* — nie jest przeprowadzana dokładna analiza problemu, tylko wybierane jest rozwiązanie, które w danym momencie wydaje się najkorzystniejsze.
- *Poszukiwanie i wyliczanie* — przeszukiwany jest zbiór danych aż do znalezienia rozwiązania.
- *Heurystyka* — na podstawie niepełnych danych tworzony jest algorytm, który działa w sposób najbardziej prawdopodobny. Metody heurystyczne nie zapewniają otrzymania poprawnego rozwiązania; powstałe algorytmy dają zawsze rozwiązania przybliżone.

Klasyfikacja algorytmów ze względu na kolejność wykonywania działań

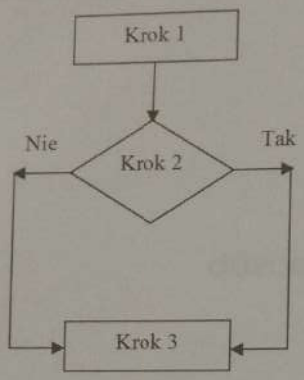
Przebieg algorytmu może być:

- *Liniowy* — kolejne kroki w algorytmie są wykonywane w kolejności, w jakiej zostały zapisane. Żaden krok nie może być pominięty ani powtórzony (rysunek 1.4).



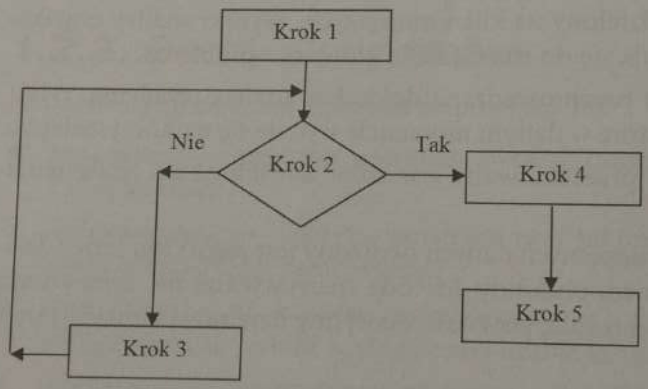
Rysunek 1.4. Algorytm liniowy

- *Warunkowy (z rozgałęzieniem)* — wykonanie poleceń zależy od spełnienia lub niespełnienia określonego warunku (rysunek 1.5).



Rysunek 1.5. Algorytm warunkowy

- *Z pętlą (cykliczny)* — grupa poleceń jest powtarzana wielokrotnie. Liczba powtórzeń może być z góry określona lub grupa poleceń jest powtarzana aż do spełnienia określonego warunku (rysunek 1.6).



Rysunek 1.6. Algorytm iteracyjny

Klasyfikacja algorytmów ze względu na sposób wykonywania operacji

- *Sekwencyjne* — operacje w algorytmie są wykonywane w kolejności, w jakiej zostały opisane.
- *Iteracyjne* — niektóre kroki są powtarzane aż do spełnienia wymaganego warunku.
- *Rekurencyjne* — tworzona jest formuła powtarzająca dane i odwołująca się do niej samej. Zakończenie wywoływania formuły następuje po spełnieniu warunku zakończenia rekurencji.

Klasyfikacja algorytmów ze względu na obszar zastosowań

- *Matematyczne* — wykonują obliczenia numeryczne.
- *Przeszukujące* — przeszukują zbiór danych w celu znalezienia określonego elementu.
- *Porządkujące* — ustawiają elementy zbioru w określonej kolejności.
- *Rekurencyjne* — rozwiązują problemy, które po podzieleniu na mniejsze części stanowią kopię głównego problemu.
- *Szyfrujące* — kodują dane w ten sposób, że ich odczyt jest niemożliwy bez znajomości klucza kodującego.

1.2.4. Implementacja algorytmów

Do najważniejszych technik implementacji algorytmów należą:

- *Proceduralność* — program jest dzielony na fragmenty (procedury) wykonujące ściśle określone operacje. Tworzone programy korzystają ze standardowych procedur wywoływanych podczas pracy programu.
- *Praca sekwencyjna* — procedury są wykonywane według kolejności ich wywołań. W danym momencie może być wykonywana tylko jedna procedura.
- *Praca wielowątkowa* — pozwala na uruchomienie co najmniej dwóch procedur w tym samym czasie. Kolejne procedury wykonywane są sekwencyjnie, lecz kolejność ich wykonania nie jest z góry określona.
- *Praca równoległa* — pozwala na uruchomienie wielu procedur w tym samym czasie. W niektórych algorytmach, na przykład w algorytmach obliczeniowych, trudno wydzielić procedury wykonywane równolegle, ponieważ korzystają one z wyników otrzymanych z wcześniejszych obliczeń.
- *Rekurencja* — procedura lub funkcja wywołuje sama siebie aż do uzyskania ostatecznego wyniku. Ponieważ ciąg wywołań nie może być nieskończony, istotne jest określenie warunku, który kończy rekurencję.
- *Obiektowość* — procedury i dane są definiowane jako klasy obiektów. Algorytmy mają znaczenie drugorzędne. Programy są tworzone jako zbiory klas.
- *Algorytm probabilistyczny* — decyzje dotyczące zachowania się algorytmu są podejmowane w sposób losowy. Działanie programu nie musi być poprawne, ale daje duże prawdopodobieństwo poprawności.

1.2.5. Złożoność obliczeniowa algorytmu

Podstawowymi działaniami przy rozwiązywaniu postawionego problemu jest dobór odpowiedniego algorytmu oraz dobór odpowiedniej struktury danych. Przy doborze algorytmu najważniejszymi aspektami są jego poprawność i złożoność. Złożoność obliczeniowa algorytmu określa ilość zasobów (pamięci, czasu) niezbędnych do rozwiązania problemu.

Określając czas niezbędny do wykonania algorytmu, należy wziąć pod uwagę:

- jakość stworzonego programu,
- szybkość komputera, na którym jest on wykonywany,
- rozmiar danych wejściowych.

Złożoność czasowa algorytmu jest rozpatrywana jako ilość czasu potrzebnego do rozwiązania problemu w zależności od liczby danych wejściowych. Podawanie złożoności obliczeniowej w jednostkach czasu jest jednak nieprecyzyjne, bo wynik zależy również od szybkości działania komputera. Dlatego złożoność obliczeniowa algorytmu najczęściej podawana jest w liczbie wykonanych operacji i jest to największa liczba operacji dominujących wykonywanych w algorytmie dla dowolnego układu danych. Operacja dominująca jest operacją, której wykonanie wpływa bezpośrednio na czas wykonania całego algorytmu. W zależności od zastosowanego algorytmu operacją dominującą może być na przykład w przypadku sortowania porównanie dwóch elementów, w przypadku przeglądania drzewa przejście między jego wierzchołkami, a w przypadku algorytmu tekstowego porównanie dwóch symboli.

Złożoność pamięciowa algorytmu określa wielkość pamięci operacyjnej komputera, która jest potrzebna do przechowywania danych wejściowych, danych pośrednich oraz ostatecznych wyników obliczeń.

Złożoność czasowa oraz złożoność pamięciowa algorytmu często zależą od postaci przetwarzanych danych, dlatego można je przedstawiać jako złożoność:

- *pesymistyczną* (określa zużycie zasobów dla najgorszego przypadku),
- *oczekiwaną* (określa zużycie zasobów dla uśrednionych wszystkich możliwych przypadków lub dla typowych przypadków),
- *optymistyczną* (określa zużycie zasobów dla najkorzystniejszego przypadku).

Do analizy złożoności obliczeniowej algorytmów wykorzystywane bywa drzewo algorytmu. Może ono służyć do określania liczby operacji wykonywanych przez algorytm.

Długość drogi to liczba wierzchołków pośrednich w drodze od korzenia do wybranego wierzchołka końcowego.

Wysokość drzewa to największa długość drogi od korzenia do wierzchołka końcowego.

Przykład 1.7

W przykładzie 1.4 podano drzewo algorytmu porządkowania trzech liczb. Jeżeli podane liczby spełniają nierówności c <= a <= b lub c <= b <= a, to liczba wierzchołków pośrednich wynosi 2. Czyli długość drogi wynosi 2. W pozostałych przypadkach liczba wierzchołków pośrednich wynosi 3. Największa długość od korzenia do wierzchołka końcowego, czyli wysokość drzewa, wynosi 3. Oznacza to największą liczbę operacji wykonywanych dla dowolnego układu danych, czyli złożoność obliczeniową algorytmu. Jest to złożoność pesymistyczna, ponieważ w niektórych przypadkach algorytm będzie wykonywany szybciej.

1.2.6. Przykłady algorytmów

Sortowanie liczb

Jednym z podstawowych zagadnień algorytmicznych jest porządkowanie zbioru danych według określonych jego cech. Szczególnym przypadkiem porządkowania danych jest sortowanie liczb lub słów. Algorytmy sortowania są klasyfikowane ze względu na sposób działania, złożoność lub stabilność. Prostą metodą sortowania jest sortowanie bąbelkowe. Polega ono na porównywaniu dwóch sąsiednich elementów i zamianie ich miejscami, gdy występują w nieprawidłowej kolejności. Sortowanie kończy się, gdy przy kolejnym przejściu nie ma żadnej zmiany kolejności elementów.

Przykład 1.8

Sortowanie bąbelkowe — lista kroków

Dane: n-elementowy ciąg liczb całkowitych

Wynik: uporządkowany malejąco ciąg liczb całkowitych

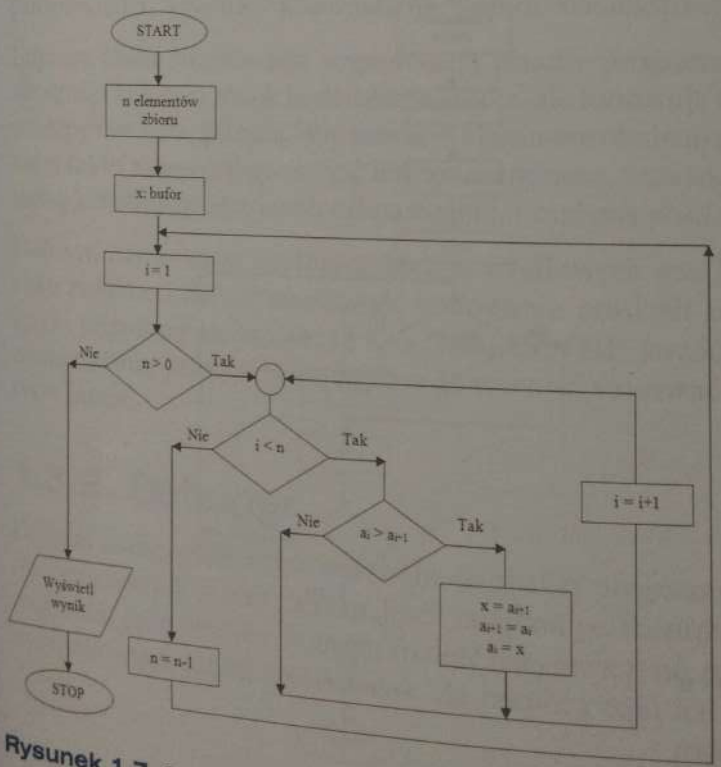
Krok 1. Dla kolejnych elementów a_i, gdzie i = 1, 2, 3, …, n–1, wykonaj krok 2. oraz krok 3.

Krok 2. Sprawdź, czy element a_i jest większy od elementu a_{i+1}.

Krok 3. Jeżeli tak, to zamień elementy miejscami.

Przykład 1.9

Sortowanie bąbelkowe — schemat blokowy (rysunek 1.7).



Rysunek 1.7. Schemat blokowy sortowania bąbelkowego

Znajdowanie najmniejszego lub największego elementu w zbiorze

Algorytm szybkiego wyszukiwania elementu w zbiorze zależy od tego, czy dane zostały uporządkowane, czy zostały zapisane w przypadkowej kolejności. W sytuacji, gdy dane są nieuporządkowane, należy przejrzeć wszystkie elementy, aby znaleźć ten właściwy.

Przykład 1.10

Znajdowanie największego elementu w zbiorze nieuporządkowanym

Dane: n-elementowy zbiór liczb naturalnych

Wynik: max — największa liczba znajdująca się w zbiorze

Krok 1. Przyjmij, że pierwszy element w zbiorze jest największy, czyli max = a_1.

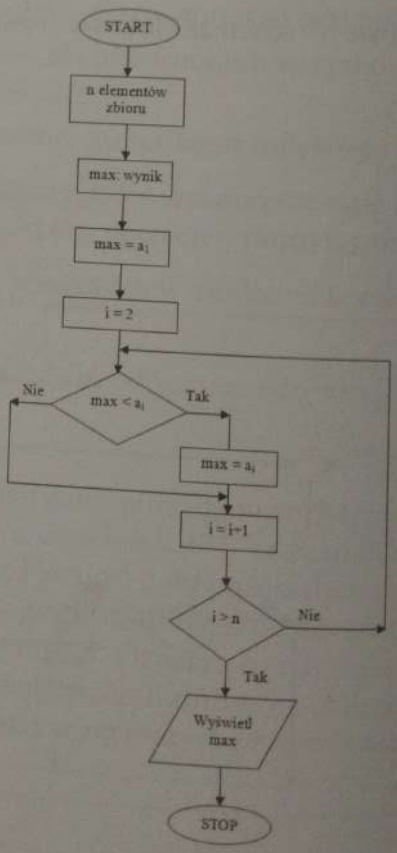
Krok 2. Dla kolejnych elementów a_i, gdzie i = 2, 3, …, n, wykonaj krok 3. oraz krok 4.

Krok 3. Sprawdź, czy max jest mniejsze od a_i.

Krok 4. Jeżeli tak, to dla max przyjmij a_i.

Przykład 1.11

Znajdowanie największego elementu w zbiorze nieuporządkowanym — schemat blokowy (rysunek 1.8).



Rysunek 1.8. Schemat blokowy znajdowania największego elementu w zbiorze nieuporządkowanym

Zadanie 1.1

Utwórz algorytm wyszukiwania najmniejszego elementu w nieuporządkowanym zbiorze danych.

Zadanie 1.2

Zmodyfikuj algorytm wyszukiwania największego elementu w zbiorze, tak aby oprócz największego elementu w zbiorze był wyświetlany indeks określający położenie tego elementu w ciągu danych.

1.3. Narzędzia programistyczne

Narzędzia, które wspomagają tworzenie programów, to:

- edytory,
- debugery,
- zintegrowane środowiska programistyczne (IDE).

1.3.1. Edytory

Do tworzenia kodu źródłowego programu można wykorzystać dowolny edytor tekstu. Najprostszym edytorem tekstu w systemie Windows jest **Notatnik**, w systemie Linux Pico, a w systemie Macintosh **Text**. Podczas tworzenia programu należy kontrolować jego składnię, pilnować konstrukcji językowych. Wyspecjalizowane edytory tekstu mają wbudowane dodatkowe narzędzia, np.: podświetlanie składni, autouzupełnianie słów, podpowiedzi, schematy kolorowania, pomoc kontekstową.

Edytory kodu źródłowego wspomagają pisanie programów. Nie tylko podświetlają składnię i kolorują różne konstrukcje języka, ale formatują tekst, ustalając odpowiednie odstępy i wcięcia, pilnują, aby nawiasy klamrowe otwierający i zamykający znajdowały się w takiej samej odległości od lewego marginesu, wstawiają pustą linię, aby oddzielić jedną konstrukcję językową od następnej, i zapisują pliki w odpowiednim formacie.

Dobrym darmowym edytorem kodów źródłowych jest **Notepad++**, dostępny w języku polskim i dający możliwość kodowania polskich znaków. Obsługuje on takie języki programowania, jak: C, C++, Java, C#, PHP, JavaScript, VB, ASP i inne. Dzięki zastosowaniu pakietu Wine można go również stosować w systemach operacyjnych typu Linux.

1.3.2. Debuger

Debuger to narzędzie wykorzystywane do analizy programu oraz odnajdowania i usuwania znalezionych w nim błędów. Jeżeli program po skompilowaniu i uruchomieniu nie działa tak, jak powinien, i trudno znaleźć tego przyczynę, można skorzystać z programu narzędziowego, jakim jest debuger. Za pomocą tego narzędzia można zlokalizować

instrukcje odpowiedzialne za wadliwe działanie programu. Umożliwia on uruchomienie i śledzenie programu krok po kroku, linia po linii. Jeżeli program jest długi, można zatrzymać jego działanie w określonych miejscach, ustawiając tak zwane pułapki. W miejscu ustawienia pułapki wykonanie programu zostanie zatrzymane, a programista może przeanalizować wykonany fragment kodu. Debugery często udostępniają podgląd zawartości pamięci, wartości zmiennych i wyrażeń. Obecnie debugery są elementami większości środowisk programistycznych.

1.3.3. Zintegrowane środowisko programistyczne (IDE)

Zintegrowane środowisko programistyczne (ang. *Integrated Development Environment*) jest to program lub zbiór programów służących do tworzenia, modyfikowania i testowania oprogramowania. Umożliwia tworzenie aplikacji w określonych językach programowania. Najczęściej udostępnia narzędzia obejmujące edycję i kompilowanie kodu źródłowego oraz narzędzia do tworzenia interfejsu graficznego. Główną zaletą środowiska IDE jest przyspieszenie i usprawnienie pracy nad tworzonym programem. Przykładami środowisk IDE są: pakiet Microsoft Visual Studio, Eclipse, NetBeans, Zend Studio. Pakiety Eclipse i NetBeans to aplikacje, które powstały dla języka Java, ale z czasem dodano do nich wsparcie dla innych języków programowania, na przykład PHP.

1.4. Etapy tworzenia programu

Proces tworzenia programu komputerowego składa się z następujących etapów:

- planowanie,
- tworzenie programu,
- kompilacja,
- konsolidacja,
- testowanie,
- optymalizacja.

1.4.1. Planowanie

Podczas planowania programu komputerowego należy sformułować problem do rozwiązania, określić cele projektu i zadania, które będzie realizował program, oraz odpowiedzieć na pytanie, gdzie będzie on stosowany i na jakiej platformie sprzętowej będzie działał. Jeżeli projekt będzie realizowany w zespole programistów, należy przydzielić zadania poszczególnym członkom zespołu. Sformułowany problem należy zapisać w postaci algorytmu.

1.4.2. Tworzenie programu

Na podstawie opracowanego algorytmu można przystąpić do pisania programu. Zapisywanie programu w postaci poleceń języka programowania wymaga właściwego zorganizowania kodu źródłowego poprzez rozmieszczanie poszczególnych elementów programu zgodnie z zaplanowaną strukturą (np. wcięcia) oraz umieszczanie kolejnych instrukcji w oddzielnych wierszach. Wcięcia ułatwią znalezienie błędnej instrukcji lub zmodyfikowanie fragmentu programu. Gdy kompilator poinformuje o numerze wiersza zawierającego błąd, jego znalezienie będzie łatwiejsze, jeżeli instrukcje zostały zapisane w oddzielnych wierszach.

Tworzony program powinien zostać uzupełniony komentarzami. Pomogą one programiście w czasie pracy nad programem oraz podczas jego modyfikowania i wyszukiwania błędów. Mogą być również wskazówką dla innych członków zespołu pracującego nad projektem.

1.4.3. Kompilacja

W trakcie tworzenia programu w celu przetestowania jego fragmentów należy je skompilować i uruchomić. Kompilacja pozwoli na sprawdzenie poprawności syntaktycznej i semantycznej programu oraz ustali rodzaj i lokalizację błędów.

Błędy kompilacji nie są jedynymi, które mogą wystąpić w programie. Program może zostać skompilowany bez błędów, ale podczas jego wykonywania mogą występować problemy z jego poprawnym działaniem. Są to tak zwane błędy uruchomienia.

W wyniku kompilacji otrzymamy kod wynikowy, który jest samodzielnym programem gotowym do wykonania.

1.4.4. Konsolidacja

W procesie konsolidacji moduły programu, które powstały podczas kompilacji, są łączone w jeden program. Podczas konsolidacji można do programu dołączyć biblioteki lub inne zewnętrzne zasoby.

1.4.5. Testowanie

Testowanie programu to działanie zapewniające jakość oprogramowania. Testowanie programu powinno być prowadzone na różnych etapach pracy, ale powinno rozpocząć się jak najwcześniej. Testowanie na etapie wytwarzania oprogramowania pozwala usunąć błędy składni i znaczenia oraz błędy związane ze strukturą programu. Testowanie w trakcie użytkowania programu sprawdza jego funkcjonalność i poprawność. Jest stosowane w celu weryfikacji oraz walidacji oprogramowania. Weryfikacja powinna sprawdzać, czy wytwarzane oprogramowanie jest zgodne ze specyfikacją. Walidacja sprawdza, czy oprogramowanie jest zgodne z oczekiwaniami użytkownika. Testowanie może dotyczyć sprawdzenia jakości oprogramowania (funkcjonalność, wydajność) oraz zgodności z wymogami formalnymi i bezpieczeństwem.

Przyjmuje się, że w programie wystąpił błąd, gdy:

- oprogramowanie nie wykonuje zadania, które było wymienione w specyfikacji,
- oprogramowanie wykonuje zadanie, którego nie było w specyfikacji,
- oprogramowanie wykonuje zadanie, którego według specyfikacji nie powinno wykonywać,
- oprogramowanie wykonuje zadanie, które było w specyfikacji, ale jego wynik jest niepoprawny.

1.4.6. Optymalizacja kodu wynikowego

Optymalizacja programu to działanie, którego celem jest poprawienie wydajności programu. Powinna prowadzić do przekształcenia kodu źródłowego do postaci, w której będzie on szybciej działał. Optymalizacja poprawia wydajność kodu, ale może utrudnić jego debugowanie, ponieważ utracona zostaje pełna odpowiedniość pomiędzy kodem źródłowym a wykonywanym. Optymalizacja w celu zwiększenia szybkości wykonywania kodu przez procesor może być prowadzona na różnych etapach pracy z programem.

1.5. Dokumentacja programu

Na dokumentację programu składają się dokumentacja techniczna oraz dokumentacja użytkownika. Jest ona tworzona dla określonego programu komputerowego przez twórców tego programu.

Dokumentacja użytkownika to opis programu przeznaczony dla jego użytkownika. Składają się na nią np. pliki pomocy, ogólne informacje o programie i jego sposobie obsługi.

Dokumentacja techniczna jest przeznaczona dla osób, które będą modyfikowały program. Zawiera dokładny opis: metod działania programu, zastosowanych algorytmów, rozmieszczenia i sposobu działania poszczególnych komponentów. Może zawierać fragmenty kodów źródłowych, wykresy, graficzne reprezentacje algorytmów, zrzuty interfejsu użytkownika, diagramy przepływu, opisy UML czy XML. Ze względu na swoją naturę jest ona przeznaczona dla programistów.

1.6. Istota programowania obiektowego

1.6.1. Wprowadzenie

Paradygmat programowania to pewien wzorzec określający sposób przepływu sterowania i sposób wykonania programu komputerowego. Języki programowania korzystają z różnych paradygmatów. Paradygmaty programowania opisują między innymi programowanie:

- strukturalne,
- obiektowe,

- proceduralne,
- funkcyjne,
- uogólnione.

Różnice polegają na różnych strukturach programów oraz na różnym podejściu do problemu.

Najbardziej rozpowszechniony jest paradygmat strukturalny (programowanie strukturalne) oraz obiektowy (programowanie zorientowane obiektowo).

W **programowaniu strukturalnym** następuje dzielenie kodu na bloki (procedury i funkcje). Instrukcje są pobierane ze stosu i są wykonywane z wykorzystaniem pętli i instrukcji warunkowych. Częstymi strukturami są instrukcje warunkowe (`if`, `if...else`), pętle (`while`, `repeat`), instrukcje wyboru (`case`). Występujące w językach strukturalnych instrukcje skoku (`goto`) lub przerwania (`continue`, `switch`) są niezgodne z zasadami strukturalności, ale pojawiają się w językach programowania, ponieważ zwiększają czytelność programu.

Prawie w każdym języku można programować strukturalnie. Jednak niektóre z nich są do tego celu specjalnie przeznaczone (Pascal).

Przykład 1.12

Obliczanie największego wspólnego dzielnika dwóch liczb z wykorzystaniem algorytmu Euklidesa w języku Pascal.

Specyfikacja:

Dane: a > 0, b > 0

```
program nwd;
var a, b, x; integer;
begin
write('Podaj pierwszą liczbę: ');
readln(a);
write('Podaj drugą liczbę: ');
readln(b);
while b > 0 do
   begin
      x := a mod b;
      a := b;
      b := x;
   end;
   write('NWD: ', a);
end.
```

W programowaniu obiektowym (ang. *object-oriented programming*) programy definiowane są za pomocą obiektów. Program napisany zgodnie z zasadami programowania obiektowego składa się ze zbioru obiektów, które komunikują się między sobą, aby wykonać określone zadania.

W programowaniu proceduralnym program dzielony jest na procedury, czyli fragmenty wykonujące ściśle określone operacje. Procedury nie powinny korzystać ze zmiennych globalnych, lecz wszystkie dane powinny być pobierane i przekazywane jako parametry wywołania.

W programowaniu uogólnionym kod programu powstaje bez wcześniejszej znajomości typów danych, na których będzie pracował. Do języków programowania wykorzystujących uogólnienia należą: C++, Java.

Przykład 1.13

Obliczanie największego wspólnego dzielnika dwóch liczb z wykorzystaniem algorytmu Euklidesa w języku C++.

Specyfikacja:

Dane: a > 0, b > 0

```
int a, b, x;
cout << "Podaj pierwszą liczbę: ";
cin >> a;
cout << "Podaj drugą liczbę: ";
cin >> b;
while (b > 0)
{
  x = a % b;
  a = b;
  b = x;
}
cout << "NWD: " << a << "\n";
```

Określony język może wspierać różne paradygmaty programowania. Może np. posiadać elementy programowania proceduralnego, obiektowego i uogólnionego. Wtedy jest to **język hybrydowy**. Można w takim języku programować, stosując się np. wyłącznie do paradygmatów programowania strukturalnego lub wykorzystując elementy wielu paradygmatów. To programista decyduje, w jaki sposób będzie tworzył program.

1.6.2. Programowanie obiektowe

Języki obiektowe w różnym stopniu stosują zasady obiektowości. Popularnymi językami obiektowymi są Java i Python, chociaż pojawiają się w nich elementy proceduralności. Języki C++ i Perl, które pierwotnie były językami programowania strukturalnego, zostały uzupełnione o elementy obiektowości. Niektóre języki, mimo iż nie są językami obiektowymi, pozwalają na stosowanie w ograniczonym zakresie niektórych metod programowania obiektowego (abstrakcyjne typy danych, dziedziczenie, polimorfizm). Językami w pełni obiektowymi są Smalltalk i Ruby, które wymuszają stosowanie metod programowania obiektowego.

Obiektowość występuje zwykle w systemach hybrydowych w połączeniu z programowaniem sieciowym (Java), skryptowym (Perl, Python, Ruby). Systemy czysto obiektowe, jak Smalltalk, nie są powszechnie stosowane.

W programowaniu obiektowym program to zbiór powiązanych obiektów, które na siebie oddziałują.

Obiekt to element, który jest opisywany przez stan (właściwości) i zachowanie (metody, czyli funkcje). Właściwości obiektu opisują jego cechy (np. liczba znaków łańcucha, wymiary okna) lub pozwalają określić jego stan. Są nazywane też polami obiektu, zmiennymi lub zmiennymi składowymi. Metody to funkcje, które wykonują operacje na obiekcie.

Klasa jest specjalną strukturą zawierającą grupę powiązanych ze sobą obiektów. Definiuje ona metody (funkcjonalność) oraz atrybuty wybranych obiektów. Obiekt jest elementem (instancją) danej klasy.

Programowanie obiektowe opiera się na tworzeniu programów, które przedstawiają świat rzeczywisty i relacje w nim zachodzące za pomocą obiektów. Program powinien być tak tworzony, aby jak najlepiej odzwierciedlał opisywany fragment rzeczywistości. Klasa opisuje za pomocą właściwości (zmienne) i metod (funkcje) fragment świata rzeczywistego. Właściwości zawierają informacje o stanie tego fragmentu, metody pozwalają na kontrolę i zmianę jego stanu. Obiekt to wystąpienie (konkretna instancja) danej klasy.

Dobrym zwyczajem jest dzielenie kodu źródłowego na klasy ze względu na funkcje. Np. jedna klasa odpowiada za obsługę błędów, inna za wyświetlanie komunikatów.

Wyróżniamy dwa podejścia do programowania obiektowego. Są to:

- programowanie oparte na klasach,
- programowanie oparte na prototypach.

W programowaniu opartym na klasach definiowane są klasy, a następnie tworzone są obiekty, które są elementem wybranej klasy.

W programowaniu opartym na prototypach nie istnieje pojęcie klasy. Nowe obiekty tworzy się w oparciu o istniejący już obiekt (prototyp), po którym dziedziczone są pola i metody (JavaScript).

Przykład 1.14
Tworzenie klas i obiektów w języku Java.

Klasa `Osoba` przechowuje trzy pola:

```
public class Osoba {
   String imie;
   String nazwisko;
   int wiek;
}
```

Klasa `Dane`, która musi być zawarta w tym samym pakiecie co klasa `Osoba`, tworzy dwa obiekty klasy `Osoba` i odpowiednio inicjuje pola:

```
public class Dane {
   public static void main (String args[]) {
      Osoba osoba1 = new Osoba();
      Osoba osoba2 = new Osoba();

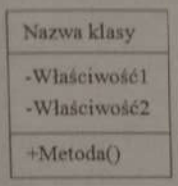
      osoba1.imie  = "Janek";
      osoba1.nazwisko  = "Kowal";
      osoba1.wiek  = 19;

      osoba2.imie  = "Kamil";
      osoba2.nazwisko  = "Nowak";
      osoba2.wiek  = 18;
      System.out.println("Dane osób: ");
     System.out.println(osoba1.imie+"  "+osoba1.nazwisko+" "+osoba1.wiek+"
lat");
       System.out.println(osoba2.imie+"  "+osoba2.nazwisko+" "+osoba2.wiek+"
lat");
   }
```

1.6.3. Modelowanie obiektowe

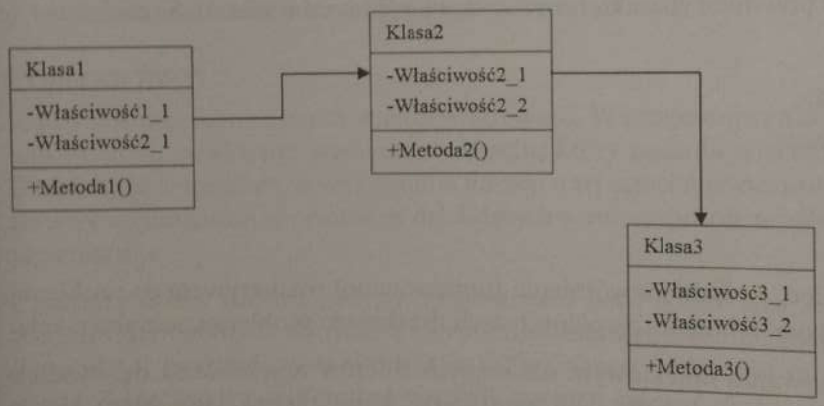
Modelowanie obiektowe jest to opisywanie rzeczywistości za pomocą obiektów. Model jest uproszczeniem rzeczywistości, który pozwala pominąć nieistotne szczegóły i uwypuklić elementy istotne. Model obiektowy tworzony jest zwykle z wykorzystaniem diagramów. Klasa może zostać opisana jako prostokąt podzielony na trzy części (rysunek 1.9):

- górna zawiera nazwę klasy,
- środkowa przedstawia właściwości obiektów,
- dolna przedstawia metody obiektów.



Rysunek 1.9. Struktura klasy

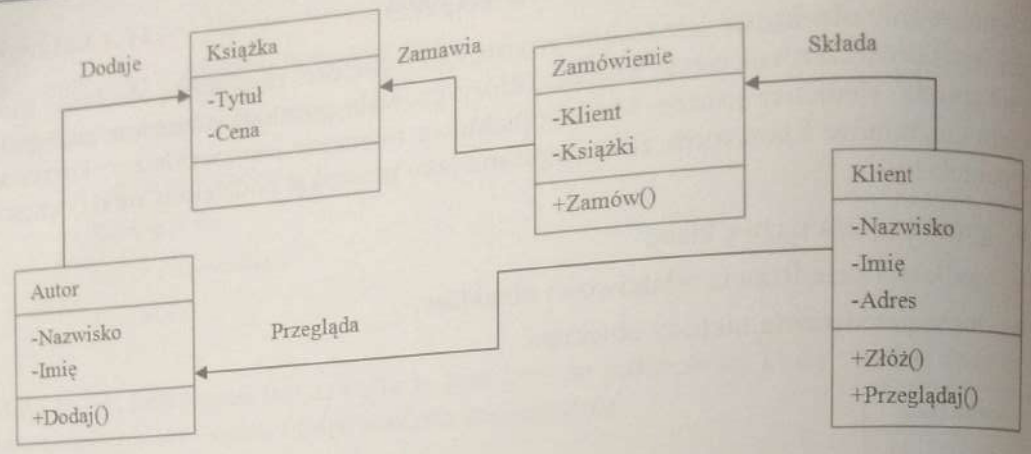
Obiekty komunikują się między sobą za pomocą wywoływania metod danej klasy przez obiekty innej klasy. Aby opisać komunikację między klasami, na diagramie umieszcza się linie łączące klasy (rysunek 1.10).



Rysunek 1.10. Zależności między klasami

Przykład 1.15

Za pomocą diagramów modelowania obiektowego można opisać działanie księgarni internetowej. Możemy powiedzieć, że korzystając z księgarni internetowej, klient zamawia dostępne w niej książki. Może również zapoznać się z informacjami na temat autorów książek. Przy takich założeniach można utworzyć klasy: *Klient*, *Książka*, *Zamówienie*, *Autor*. Dla utworzonych klas można określić właściwości oraz metody (rysunek 1.11).



Rysunek 1.11. Model obiektowy dla księgarni internetowej

Umiejętność tworzenia klas oraz definiowania zależności między nimi jest podstawą programowania obiektowego. Jeżeli struktura fragmentu rzeczywistości zostanie poprawnie zaprojektowana, to będzie stałym elementem, niezależnym od użytego języka programowania.

1.6.4. Cechy obiektowości

Język obiektowy powinien charakteryzować się określonymi cechami. Są to:

- abstrakcja,
- hermetyzacja,
- dziedziczenie,
- polimorfizm.

Abstrakcja

Można powiedzieć, że każde uogólnienie (uproszczenie) rozpatrywanego problemu, które polega na wyodrębnieniu wspólnych cech dla danego problemu, jest abstrakcją.

Jeżeli w programowaniu obiektowym dla różnych zbiorów obiektów da się wyodrębnić wspólne cechy, to można dla nich wprowadzić klasę abstrakcyjną, która będzie udostępniała te cechy. Klasa abstrakcyjna staje się pewnym wzorcem dla innych klas. Sama nie może służyć do tworzenia obiektów, ale można na jej podstawie tworzyć inne klasy (dziedziczenie). Dla klasy abstrakcyjnej można definiować metody i właściwości.

Hermetyzacja

Hermetyzacja, inaczej enkapsulacja, to ukrywanie pewnych właściwości (danych składowych) lub metod obiektów określonej klasy. Dzięki temu obiekty są dostępne tylko w obrębie ich klasy. Tylko metody własne obiektu mogą zmieniać jego stan. Gdy dostęp

do wszystkich właściwości danej klasy jest możliwy tylko przez metody, to taką hermetyzację nazywamy **hermetyzacją pełną**.

W programowaniu obiektowym mechanizm ten jest związany z bezpieczeństwem kodu. Ogranicza on dostęp do części danych składowych oraz metod, które są wykonywane wewnątrz określonej klasy.

Dziedziczenie

Mechanizm dziedziczenia pozwala na tworzenie klas na podstawie innych wcześniej utworzonych klas. Utworzona klasa posiada właściwości i metody zdefiniowane dla niej oraz właściwości i metody klasy, na podstawie której została utworzona, zdefiniowane jako *public* i *protected*. Natomiast niedostępne są właściwości i klasy zdefiniowane jako *private*. Klasa dziedzicząca jest nazywana **klasą pochodną** lub potomną. Klasa, na podstawie której następuje dziedziczenie, nazywana jest **klasą bazową**. Na podstawie jednej klasy bazowej można tworzyć dowolną liczbę klas pochodnych. Zależności między klasami bazowymi i pochodnymi tworzą tzw. hierarchię klas. W wyniku powstają grupy obiektów zwane klasami oraz grupy klas zwane drzewami. Odzwierciedlają one wspólne cechy obiektów. Mechanizm ten ma zastosowanie przy wielokrotnym wykorzystywaniu kodu. Jeżeli dwie klasy są opisane w podobny sposób, to można dla nich utworzyć wspólną klasę bazową, która będzie zawierała identyczne dla obu klas właściwości i metody.

W językach programowania, w których nie występuje pojęcie klasy (np. JavaScript), dziedziczenie zachodzi pomiędzy poszczególnymi obiektami.

Polimorfizm

Słowo **polimorfizm** oznacza wielopostaciowość. W programowaniu obiektowym polimorfizm to możliwość stworzenia obiektu, który posiada więcej niż jedną formę. Oznacza to, że możliwe są wystąpienia funkcji o tej samej nazwie, zawierających różne zestawy argumentów i operatorów działających w różny sposób, w zależności od typów argumentów.

Czasami nie jest istotne, jakiego typu wartości będą przetwarzane. Można napisać program dla ogólnych danych, a ewentualne uszczegółowienie zostawić systemowi (kompilator i system czasu wykonania). Np. inaczej dodawane są liczby całkowite, a inaczej rzeczywiste, ale można obydwie operacje nazwać `dodaj` i oznaczyć symbolem +. Wybór funkcji do wykonania nastąpi w miejscu jej wywołania na podstawie nazwy funkcji oraz typów argumentów.

Gdy uszczegółowienie zostaje podjęte na etapie kompilacji, mamy do czynienia z **polimorfizmem statycznym** (czasu kompilacji). Gdy jest odłożone do momentu wykonywania programu, mamy do czynienia z **polimorfizmem dynamicznym** (czasu wykonania).

PYTANIA KONTROLNE

1. Jak działa *interpreter* kodu źródłowego?
2. Jak działa *aplikacja webowa*?
3. Wymień etapy kompilacji kodu źródłowego.
4. Na czym polega *analiza semantyczna* wykonywana podczas kompilacji kodu źródłowego?
5. Wymień podstawowe cechy języków skryptowych.
6. Co to jest *algorytm*?
7. Wymień podstawowe cechy algorytmów.
8. Jakie elementy powinna zawierać *specyfikacja algorytmu*?
9. Co to jest *złożoność obliczeniowa algorytmu*?
10. Wymień i omów narzędzia wspomagające tworzenie programów.
11. Na czym polega *modelowanie obiektowe*?
12. Wymień podstawowe cechy języków obiektowych.
13. Omów występujący w programowaniu obiektowym mechanizm *dziedziczenia*.
14. Na czym polega *polimorfizm*?

ZADANIA

1. Utwórz algorytm w postaci listy kroków, schematu blokowego i drzewa algorytmu dla równania liniowego a*x+b = 0.
2. Za pomocą diagramów modelowania obiektowego utwórz model działania swojej szkoły. Zaprojektuj klasy: `Uczeń`, `Nauczyciel`, `Klasa`, `Przedmiot`. Określ dla nich odpowiednie metody i właściwości. Każdy uczeń jest przypisany do określonej klasy. Do każdej klasy musi zostać przypisana lista przedmiotów. Nauczyciel uczy wybranych przedmiotów.

2

Aplikacje internetowe

2.1. Wprowadzenie

Aplikacja internetowa to program, który pracuje na serwerze i komunikuje się z użytkownikiem poprzez sieć komputerową z wykorzystaniem przeglądarki internetowej. Przeglądarka internetowa w takim przypadku pełni funkcję interfejsu użytkownika. W aplikacji internetowej zakłada się interakcję z użytkownikiem, korzystanie z baz danych i innych usług, często rozproszonych, umieszczonych na wielu różnych serwerach.

Istotnymi cechami aplikacji internetowej są łatwość i szybkość dotarcia do informacji (dowolny komputer podłączony do internetu lub sieci LAN i przeglądarka stron internetowych) oraz bezpieczeństwo danych (login i hasło, szyfrowanie połączenia oraz system uprawnień).

Zalety aplikacji internetowych to:

- dostępność dla wszystkich zainteresowanych bez ograniczeń związanych z czasem i miejscem,
- praktycznie brak konieczności instalowania dodatkowego oprogramowania; aby korzystać z aplikacji internetowej wystarczy przeglądarka internetowa (czasem tylko z dodatkowymi pluginami),
- brak konieczności zakupu dodatkowego sprzętu komputerowego; aplikacja internetowa może zostać zainstalowana na serwerze dostępnym w internecie,
- łatwość utrzymania i modernizacji, ponieważ zmiany i aktualizacje w aplikacji są wykonywane na serwerze bez udziału użytkowników aplikacji,
- łatwość integracji aplikacji internetowej z innymi usługami, np. stroną internetową,
- niższe koszty wytworzenia, uruchomienia i utrzymania aplikacji internetowej w porównaniu z innymi rozwiązaniami; większość technologii stosowanych do wytworzenia aplikacji internetowych jest bezpłatna.

2.2. Narzędzia aplikacji internetowych

Do prawidłowej pracy aplikacji internetowej potrzebne jest oprogramowanie zwane serwerem WWW (na przykład Apache, IIS) oraz oprogramowanie wspomagające (serwer bazodanowy — na przykład MySQL, MS SQL) i środowisko do tworzenia programów (na przykład PHP, Java, C#). Z serwerem WWW komunikuje się przeglądarka internetowa, a serwer bazodanowy i interpreter umożliwiają udostępnianie dynamicznie tworzonych stron internetowych oraz zarządzanie bazami danych.

Narzędzia stosowane podczas pracy z aplikacją internetową można podzielić na dwie grupy:

- *Narzędzia po stronie klienta* — pozwalają klientowi wykonywać część zadań związanych z obsługą aplikacji. Należą do nich JavaScript, aplety Java, AJAX i inne.
- *Narzędzia po stronie serwera* — pozwalają wykonywać na serwerze zadania służące do obsługi żądań klienta. Są to technologie typu: ASP.NET (Active Server Pages), JavaServer Pages (JSP) i języki PHP, Perl, Coldfusion, Ruby on Rails i wiele innych.

2.2.1. Technologie po stronie klienta

Technologie działające po stronie klienta są związane z rodzajem oprogramowania klienckiego, czyli przeglądarki internetowej i zainstalowanych wtyczek. Najczęściej wspomagają one technologie działające po stronie serwera.

Podstawową technologią stosowaną po stronie klienta jest JavaScript. Jest to język skryptowy, którego kod można wstawić w kod strony HTML w celu uzyskania dynamiki strony. JavaScript jest obecnie wspierany przez wszystkie przeglądarki i nie wymaga instalowania dodatkowego oprogramowania.

Inną technologią, która może być stosowana po stronie klienta, są aplety Java. Aplety te pisane są w języku Java, a do ich wykonania potrzebne jest zainstalowanie w przeglądarce wtyczki Java Virtual Machine. Aplety Java są coraz rzadziej stosowane ze względu na zajmowanie zbyt wielu zasobów komputera i rozwój innych technologii.

Technologia Flash to kolejne narzędzie, które może być stosowane po stronie klienta. Dostarcza ono narzędzi do tworzenia animacji i efektów graficznych. Technologia Flash wymaga instalowania w przeglądarce specjalnej wtyczki Adobe Flash Player.

AJAX (ang. *Asynchronous JavaScript and XML*) to technologia uzupełniająca język JavaScript. Oferuje ona te same działania po stronie klienta co JavaScript. Ale w technologii AJAX komunikacja z serwerem jest realizowana w tle, podczas gdy użytkownik pracuje na stronie WWW.

2.2.2. Technologie po stronie serwera

Technologie działające po stronie serwera powodują, że klient otrzymuje gotową stronę internetową wygenerowaną z uwzględnieniem kryteriów określonych przez użytkownika.

Pierwszą technologią, która zyskała popularność i jest stosowana nadal w niektórych aplikacjach, była technologia CGI (ang. *Common Gateway Interface*). Umożliwiała ona między innymi dynamiczne generowanie dokumentów przed wysłaniem ich do przeglądarki, tworzenie dokumentów w oparciu o dane znajdujące się w bazie, generowanie i przetwarzanie formularzy. Skrypty CGI mogą być programami kompilowanymi do postaci binarnej lub programami interpretowanymi, pisanymi np. w Perlu lub w języku skryptowym powłoki.

Kolejną technologią były filtry ISAPI (ang. *Internet Server API*), których działanie jest sterowane zdarzeniami serwera, a nie żądaniami klienta. Zaletą tej technologii jest szybkość działania oraz większa wydajność w porównaniu z technologią CGI. Jej wadą jest działanie tylko w systemach Windows.

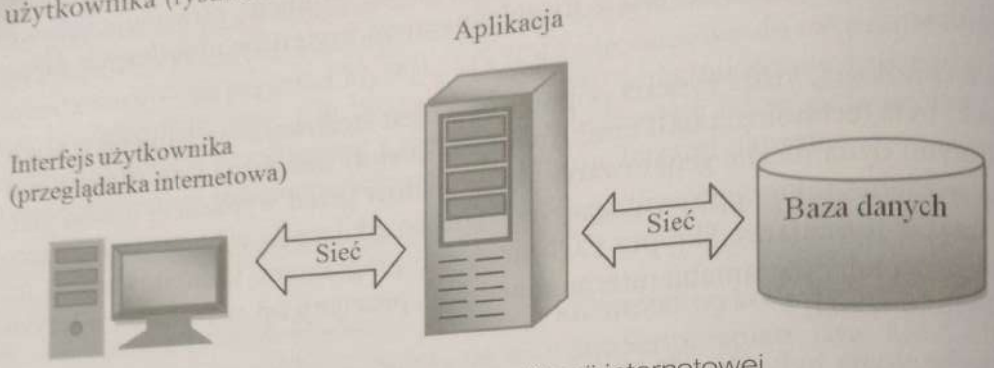
Obecnie popularne stały się technologie, które generują strony WWW poprzez interpretowanie kodu osadzonego wewnątrz strony. Do technologii tego typu należą:

- ASP.NET — jest to bardzo wydajna i nowoczesna technologia opracowana przez firmę Microsoft. Przeznaczona została do tworzenia różnych aplikacji internetowych w formatach HTML, WML oraz XML. Tworzone dokumenty zawierają statyczną treść, ale w ramach kodu statycznego istnieje możliwość dodania kodu dynamicznego.
- JSP (ang. *JavaServer Pages*) — wraz z innymi elementami wchodzącymi w skład platformy Java Platform firmy Sun umożliwia tworzenie dynamicznych dokumentów WWW w formatach HTML, XHTML, DHTML oraz XML z wykorzystaniem języka Java. Jest to rozwiązanie podobne do technologii PHP.
- PHP — wraz z serwerem WWW Apache oraz serwerem baz danych MySQL tworzy wydajną i niezależną od systemu platformę do budowania dynamicznych stron WWW. Platforma ta określana jest jako platforma **AMP**. W systemach Linux znana jest pod nazwą **LAMP**, a w systemie Windows pod nazwą **WAMP**. Istnieje również wieloplatformowa wersja, nazywana **XAMPP**. PHP jest systemem otwartym, dzięki czemu powstają coraz nowsze moduły rozszerzające jego możliwości. PHP jest nie tylko technologią funkcjonującą po stronie serwera. Jest to również język skryptowy, wykorzystywany przy pracy nad aplikacjami uruchamianymi niezależnie od serwera WWW.

2.3. Struktura aplikacji internetowej

Podstawowa architektura aplikacji internetowej zakłada, że na komputerze użytkownika funkcjonuje wyłącznie oprogramowanie obsługi interfejsu użytkownika, natomiast oprogramowanie właściwej aplikacji oraz oprogramowanie dostępu do danych znajdują się na oddzielnych, dedykowanych komputerach. Takie rozwiązanie pozwala

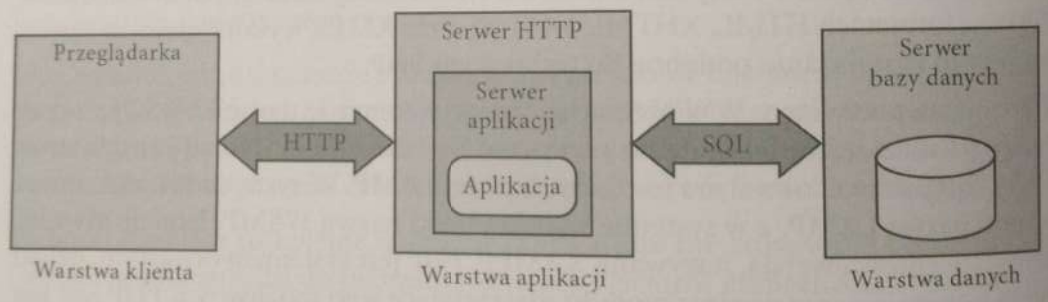
na współbieżny dostęp do danych i aplikacji oraz obniża wymagania dla komputera użytkownika (rysunek 2.1).



Rysunek 2.1. Podstawowa architektura aplikacji internetowej

Rozszerzona architektura aplikacji internetowej zakłada, że w jej skład wchodzą trzy programowe warstwy funkcjonalne:

- **Warstwa klienta** — odpowiedzialna za interfejs użytkownika i interakcję z użytkownikiem, oparta na kliencie HTTP.
- **Warstwa aplikacji** — która w odpowiedzi na żądania klientów tworzy dokumenty dynamiczne. Warstwa ta składa się z serwera HTTP i serwera aplikacji będącego środowiskiem uruchomieniowym dla aplikacji generujących dokumenty dynamiczne.
- **Warstwa danych** — która odpowiada za udostępnianie informacji umieszczanych w dokumentach dynamicznych. Warstwa ta składa się z serwera bazy danych, który odpowiada na żądania w języku SQL (rysunek 2.2).

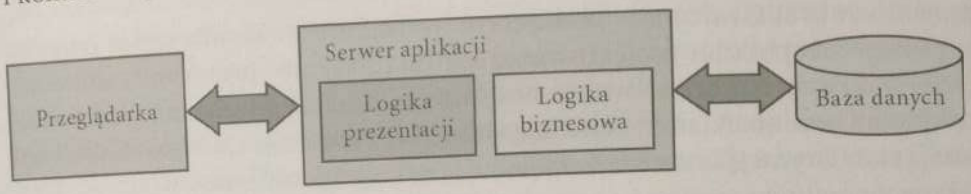


Rysunek 2.2. Programowalne warstwy aplikacji internetowej

Istotnym elementem tej architektury jest serwer aplikacji. Odpowiada on za obsługę komunikacji z warstwą klienta i warstwą danych. Programista, który tworzy aplikacje internetowe, nie musi tworzyć kodu obsługi protokołu HTTP z klientem ani kodu obsługi komunikacji SQL z serwerem bazy danych. Ponadto serwer aplikacji realizuje inne typowe funkcje, np. obsługę transakcji, rejestrację żądań w plikach dziennika, autoryzację dostępu użytkowników do aplikacji.

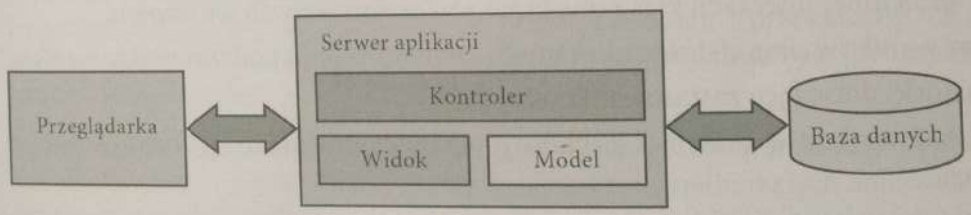
Aplikacje internetowe najczęściej buduje się w ten sposób, że poszczególne funkcje systemowe w celu ich separacji są rozdzielone pomiędzy różne komponenty. Stosowane są dwie metody: architektura 4-warstwowa (*4-tier architecture*) lub architektura *Model-View-Controller*.

Architektura 4-warstwowa nazywa się tak, ponieważ definiuje cztery warstwy aplikacji internetowej. Są to warstwy: klienta, logiki prezentacji, logiki biznesowej i danych. Składniki aplikacji są podzielone na dwie grupy: składniki logiki prezentacji i składniki logiki biznesowej. Zadania składników logiki prezentacji to: przyjmowanie żądań od klientów, wywoływanie funkcji składników logiki biznesowej, generowanie dokumentów dynamicznych i wypełnianie ich danymi przekazywanymi przez składniki logiki biznesowej. Zadania składników logiki biznesowej to realizacja procesów biznesowych i komunikacja z bazą danych (rysunek 2.3).



Rysunek 2.3. Architektura 4-warstwowa

Architektura *Model-View-Controller* (MVC) jest dalszym rozwinięciem architektury 4-warstwowej. Składniki aplikacji internetowej są podzielone na trzy grupy: składniki sterujące — logika sterowania (*controller*), składniki prezentacji — logika prezentacji (*view*) i składniki modelu — logika biznesowa (*model*). Składniki prezentacji tworzą dokumenty dynamiczne i wypełniają je danymi dostarczanymi przez składniki modelu. Składniki modelu odpowiadają za realizację procesów biznesowych i komunikację z bazą danych. Składniki sterujące przyjmują żądania od klientów oraz koordynują ich obsługę (wywoływanie funkcji składników modelu i prezentacji) (rysunek 2.4).



Rysunek 2.4. Architektura Model-View-Controller

Zaletą technologii aplikacji internetowych jest to, że użytkownik korzystający z takiej aplikacji może posługiwać się tylko programem klienta HTTP, który zwykle ma małe wymagania sprzętowe. Dzięki temu dostęp do aplikacji internetowych możliwy jest też za pomocą tabletu czy telefonu komórkowego. Inną zaletą jest wygoda administrowania aplikacjami internetowymi. Są one umieszczane na pojedynczym komputerze (serwerze aplikacji), w związku z tym prace administracyjne dotyczą tylko tego komputera.

Wadą technologii aplikacji internetowych są koszty, jakie należy ponieść, aby wdrożyć ten system (np. konieczność zakupu serwera o dużej mocy obliczeniowej czy zakupu komercyjnego oprogramowania serwera aplikacji). Inną niedogodnością są trudności z wytwarzaniem oprogramowania.

Serwerami aplikacji są np.: JBoss Application Server (serwer aplikacji w Javie na licencji LGPL), IBM WebSphere (serwer aplikacji JEE oferowany przez firmę IBM), platforma .NET Microsoft.

2.4. Projektowanie aplikacji internetowych

2.4.1. Wzorce projektowe

Wzorce projektowe to sprawdzone w praktyce, uniwersalne rozwiązania często pojawiających się problemów podczas projektowania aplikacji. Wzorce projektowe tworzone są najczęściej w oparciu o programowanie obiektowe. Pokazują powiązania i zależności pomiędzy klasami oraz obiektami i ułatwiają tworzenie i modyfikację kodu źródłowego. Wprowadzają standaryzację kodu oraz zwiększają jego wydajność i niezawodność. Wzorzec projektowy powinien zawierać oprócz rozwiązania problemu także dokumentację, która wyjaśni cel, sposób działania i zalety danego rozwiązania.

Dokumentacja wzorca projektowego powinna zawierać:

- unikatową nazwę, która pozwoli odwoływać się do wzorca,
- opis celu, czyli czym należy się kierować podczas wyboru wzorca,
- opis sytuacji, w których wzorzec można stosować,
- graficzną reprezentację wzorca w postaci diagramu klas lub interakcji,
- listę klas i obiektów stosowanych we wzorcu,
- opis wzajemnej interakcji klas i obiektów wykorzystywanych we wzorcu,
- wykaz wyników oraz efektów ubocznych, jakie występują podczas użycia wzorca,
- wskazówki dotyczące zastosowania wzorca,
- przykładowy kod w jednym z języków programowania pokazujący zastosowanie wzorca.

Wzorzec projektowy nie jest gotową implementacją rozwiązania. Przypomina szablon, który może być zastosowany w wielu różnych sytuacjach. Wzorce oparte są na praktycznych rozwiązaniach, które zostały zaimplementowane w wybranym języku obiektowym. Mogą przyspieszyć proces projektowania aplikacji przez wykorzystanie wypróbowanych rozwiązań dla sformułowanego problemu.

2.4.2. Wzorzec projektowy MVC

MVC (ang. *Model-View-Controller*) jest jednym z najczęściej stosowanych wzorców projektowych w aplikacjach internetowych. Głównym założeniem tego wzorca jest podzielenie kodu aplikacji na 3 moduły:

- *Model* — reprezentuje dane (np. pobierane z bazy danych).
- *Widok* — reprezentuje interfejs użytkownika.
- *Kontroler* — reprezentuje logikę sterującą aplikacją.

Model

Model zapewnia zestandaryzowany sposób dostępu do danych. Najczęściej stosowany jest do pobierania i modyfikowania rekordów z bazy danych. Pozostała część aplikacji staje się niezależna od tego skąd i w jaki sposób dane są pobierane. Dane mogą być pobierane z różnych źródeł (np. z bazy lub pliku) i nie jest to istotne z punktu widzenia logiki sterującej aplikacji. Tworząc tę logikę, wywołujemy tylko odpowiednie funkcje modelu i w wyniku dostajemy dane do przetworzenia. Model jest elementem opcjonalnym, gdyż nie zawsze korzystamy z danych pobieranych z bazy czy plików.

Widok

Widok reprezentuje to, co będzie widoczne dla użytkownika. Widok oddzielony od logiki aplikacji pozwala na bezproblemową zmianę oprawy graficznej w dowolnym momencie. Zadaniem kontrolera jest przekazanie danych do widoku. Programista tworzący widok nie musi nic wiedzieć na temat logiki programu. Otrzymuje on tylko dane, które należy sformatować. Widok można zmienić, nie ingerując w kod programu.

Kontroler

Kontroler zawiera najważniejszą część kodu, która steruje aplikacją. Odpowiada m.in. za przetwarzanie danych pobranych za pomocą modelu i przekazanie ich poprzez widok użytkownikowi, a także za wykonywanie odpowiednich akcji w zależności od działań użytkownika.

Zaletą korzystania ze wzorca MVC jest uzyskanie gotowej struktury aplikacji oraz uporządkowanie struktury kodu szczególnie przy tworzeniu dużych aplikacji. Pozwala też na uniknięcie problemów ze znalezieniem określonej funkcjonalności. Umożliwia stosowanie jasnego podziału prac nad projektem. Zmiana bazy danych czy wyglądu strony przestaje być problemem.

Wzorzec MVC często jest traktowany jako wzorzec złożony, który może być rozbijany na prostsze wzorce. Na bazie wzorca MVC powstało również wiele wzorców pochodnych. Obecnie częściej stosowane są wzorce pochodne niż oryginalny wzorzec.

Jedną z pierwszych adaptacji MVC na potrzeby środowiska WWW była platforma Struts przeznaczona dla technologii Java Server Pages.

Firma Microsoft posiada adaptację MVC zbudowaną na bazie technologii ASP.NET w postaci platformy ASP.NET MVC, która umożliwia budowanie aplikacji z uwzględnieniem wyraźnie odseparowanych modeli, kontrolerów oraz widoków. Platforma ta udostępnia dużą liczbę specyficznych rozszerzeń oraz konstrukcji programistycznych.

Istnieje wiele frameworków wspomagających programowanie i udostępniających gotową architekturę opartą na MVC. Należą do nich: Zend Framework, Code Igniter, CakePHP, Symfony.

2.5. Framework — platforma programistyczna

Do tworzenia aplikacji internetowych często wykorzystuje się platformy programistyczne, które definiują strukturę aplikacji, określają mechanizm ich działania oraz dostarczają bibliotek umożliwiających wykonywanie określonych zadań. Framework posiada zdefiniowaną podstawową strukturę aplikacji, czyli elementy, które pozostają niezmienne we wszystkich utworzonych przy jego pomocy aplikacjach. Programista tworzy aplikację, dostosowując poszczególne komponenty do wymagań realizowanego projektu. Ma on również możliwość tworzenia nowych elementów niezbędnych w projektowanej strukturze aplikacji.

Typowe cechy frameworka to:

- *Odwrócenie sterowania* — przepływ sterowania jest narzucany przez framework, a nie przez programistę.
- *Domyślne zachowanie* — framework powinien posiadać domyślną konfigurację, która musi być użyteczna i dawać określony wynik.
- *Rozszerzalność* — programista powinien mieć możliwość rozszerzania poszczególnych komponentów frameworka, jeśli chce je rozbudować o dodatkową funkcjonalność.
- *Zamknięta struktura wewnętrzna* — programista może rozbudowywać framework, ale nie poprzez modyfikację domyślnego kodu.

Tworzenie aplikacji z wykorzystaniem frameworków wymaga od programisty mniejszej ilości kodu do napisania. Aplikacja posiada dobrą wewnętrzną strukturę, jest dobrze zaprojektowana i przetestowana. Ceną za elastyczną budowę jest niższa wydajność tworzonego oprogramowania. Frameworki mogą być stosowane jako szkielety zarówno kompletnych aplikacji, jak i pojedynczych komponentów.

2.5.1. Narzędzia programistyczne

Eclipse IDE

Eclipse to platforma programistyczna przeznaczona do tworzenia i testowania aplikacji internetowych. Sama platforma nie posiada narzędzi do tworzenia aplikacji czy pisania programu, ale zapewnia obsługę wtyczek realizujących określone zadania. Dostępne są

implementacje różnych języków programowania, między innymi w Javie, C/C++ i PHP. Posiada narzędzia do współpracy z serwerami WWW oraz serwerami baz danych. Może pracować z systemami Windows, Linux i Mac OS/X.

Zend Framework

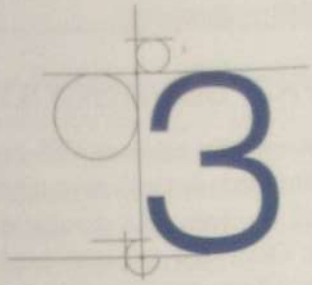
Zend Framework to platforma programistyczna przeznaczona do tworzenia aplikacji internetowych w języku PHP. Zawiera zbiór bibliotek obsługujących podstawowe mechanizmy działania aplikacji oraz biblioteki użytkowe obsługujące np. wysyłanie e-maili, komunikację z innymi aplikacjami internetowymi itp. Posiada również implementację modelu MVC, który może zostać wykorzystany do utworzenia struktury aplikacji.

NetBeans IDE

Jest to kolejny darmowy zestaw narzędzi programistycznych służący do tworzenia aplikacji, w tym również usług Web Services i aplikacji przeznaczonych na urządzenia mobilne. Głównie przeznaczony dla języka Java, ale umożliwia kompilowanie programów napisanych w innych językach, np. JavaScript, C++, Ruby, Python, PHP.

PYTANIA KONTROLNE

1. Wymień podstawowe zalety wykorzystywania aplikacji internetowych.
2. Omów narzędzia stosowane podczas pracy z aplikacją internetową po stronie klienta.
3. Omów narzędzia stosowane podczas pracy z aplikacją internetową po stronie serwera.
4. Wymień programowe warstwy funkcjonalne, które wchodzą w skład rozszerzonej architektury aplikacji internetowej.
5. Jakie zadania ma logika prezentacji w architekturze 4-warstwowej?
6. Z jakich modułów składa się wzorzec projektowy MVC?
7. Omów mechanizm działania frameworków.



Skrypty po stronie klienta — JavaScript

3.1. Wprowadzenie

Techniki służące do dynamicznej zmiany treści, wyglądu lub zachowania się dokumentu HTML umożliwiają interakcję użytkownika ze stroną internetową. W skład dynamicznego HTML wchodzą technologie: DOM, CSS, HTML, SVG, języki skryptowe (JavaScript, VBscript i inne).

DOM (ang. *Document Object Model*) to obiektowy model dokumentu, czyli sposób reprezentacji dokumentów XML i XHTML w postaci modelu obiektowego. Jest on niezależny od platformy i języka programowania. Standard zdefiniowany dla DOM przez organizację W3C opiera się na zespole klas i interfejsów pozwalających na dostęp do struktury dokumentów oraz jej modyfikację przez tworzenie, zmienianie i usuwanie tak zwanych węzłów (ang. *nodes*). Standard W3C definiuje interfejsy DOM tylko dla języków ECMAScript i Java.

SVG (ang. *Scalable Vector Graphics*) — jest to uniwersalny format grafiki wektorowej (statycznej i animowanej) powstały z myślą o zastosowaniu na stronach internetowych. Może być integrowany z językami opartymi na technologii XML. Został opracowany przez organizację W3C na potrzeby internetu. W SVG oprócz standardowych obiektów (prostokąty, elipsy, krzywe) można opisywać efekty specjalne, maski przezroczystości oraz sposób animacji elementów.

Języki skryptowe są to języki interpretowane, zaprojektowane z myślą o interakcji z użytkownikiem. Skrypty są wykonywane wewnątrz określonej aplikacji w odróżnieniu od programów nieskryptowych, które wykonują się niezależnie od innych aplikacji. Język skryptowy działający po stronie przeglądarki jest najważniejszym elementem dynamicznego HTML.

3.2. Struktura języka JavaScript

Po utworzeniu kodu HTML i zdefiniowaniu arkuszy CSS utworzenie kodu kontrolującego zachowanie strony oraz określającego interakcje z użytkownikiem jest kolejnym etapem budowy strony internetowej. Do tego celu stosuje się języki skryptowe. Najpopularniejszym z nich jest JavaScript. Odgrywa on coraz większą rolę w projektowaniu interaktywnych stron internetowych, od prostej manipulacji danymi, przez dynamiczne modelowanie struktury strony, obsługę multimedialnych rozszerzeń, aż po tworzenie odrębnych aplikacji.

Język JavaScript umożliwia tworzenie i umieszczanie w kodzie HTML krótkich programów, które mogą wykonywać różne zadania, np. obsługiwanie zdarzeń, weryfikowanie danych wprowadzanych do formularza, nawigowanie między stronami.

Przy użyciu odpowiednich bibliotek języka JavaScript (np. Prototype, jQuery, MooTools) można tworzyć zaawansowane aplikacje sieciowe o atrakcyjnym interfejsie.

UWAGA

Zbieżność nazwy języka JavaScript i Java może być myląca. Można przyjąć, że poza nazwą i podobieństwem składni oba te języki mają ze sobą niewiele wspólnego.

3.2.1. Opis języka

JavaScript jest obiektowym skryptowym językiem programowania nawiązującym do języka C. Jest językiem interpretowanym, czyli bez konieczności kompilowania można oglądać efekty jego działania. Potrzebna jest tylko przeglądarka internetowa, która obsługuje język JavaScript. Tworzone skrypty zapewniają interaktywność strony internetowej poprzez reagowanie na zdarzenia, budowanie elementów nawigacji oraz sprawdzanie poprawności formularzy.

Typowe zastosowania języka JavaScript to:

- umożliwienie modyfikowania wyglądu bieżącego dokumentu,
- wyświetlanie prostych okien dialogowych,
- kontrola poprawności wypełnienia formularza,
- dostęp do daty i czasu,
- lokalne generowanie dokumentów HTML.

3.2.2. JavaScript w HTML

Kod źródłowy napisany w języku JavaScript może być umieszczony wewnątrz dokumentu HTML między znacznikami `<script>` i `</script>`.

Znaczniki skryptu mogą być wstawiane w dowolnym miejscu dokumentu, ale zalecane jest umieszczanie ich na początku strony, w sekcji `<head>`. W obrębie strony można używać znaczników `<script>` wielokrotnie, na przykład w sekcjach `<head>` i `<body>`. Znacznik `<script>` posiada atrybut `type`, który określa typ MIME języka skryptu. Można mu nadać odpowiednią wartość, definiując język, w którym będzie napisany skrypt (dla JavaScript jest to text/javascript).

```
<script type="text/jezyk">
```

Przykład 3.1

Skrypt umieszczony w dokumencie HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript - Przykładowy program</title>
<script type="text/javascript">
<!--
document.write("Napisz program w JavaScript");
//-->
</script>
</head>
<body>
...
</body>
</html>
```

Dobrą praktyką programistyczną jest wielokrotne wykorzystywanie raz napisanego kodu. Dlatego kod źródłowy skryptu, który będzie wielokrotnie używany, można umieścić w osobnym pliku. Będzie to plik tekstowy, który powinien posiadać rozszerzenie *js*. Kod zapisany w takim pliku nie zawiera znaczników `<script>`. Jeżeli skrypt został zapisany w zewnętrznym pliku, to w kodzie HTML powinna pojawić się informacja w postaci atrybutu `src`. To rozwiązanie, podobnie jak w przypadku zewnętrznych arkuszy CSS, ułatwia zmiany w kodzie strony bez ingerencji w pliki HTML.

Przykład 3.2

Skrypt umieszczony w pliku zewnętrznym o nazwie *plik.js*:

```
<html> <head>
<script type="text/javascript" src="plik.js">
</script>
</head>
```

```
<body>
<p> Skrypt znajduje się w pliku "plik.js".</p>
</body> </html>
```

3.3. Składnia języka JavaScript

Składnia języka jest ściśle określonym zbiorem reguł, których należy przestrzegać. W języku JavaScript stosuje się następujące kategorie pojęć składniowych: wyrażenia, zmienne i literały. Do elementów języka należy również zbiór predefiniowanych obiektów i funkcji (tablice, data, funkcje matematyczne).

3.3.1. Instrukcje

Instrukcje w języku JavaScript mogą być pisane pojedynczo, w jednym wierszu lub w kilku wierszach. Średnik odgrywa rolę ogranicznika instrukcji, ale nie jest wymagany, jeżeli po instrukcji wystąpi znak nowego wiersza. Jeżeli kilka instrukcji zostanie zapisanych w jednym wierszu, muszą być one oddzielone średnikiem.

3.3.2. Wielkość liter

W programie JavaScript wielkość liter przeważnie ma znaczenie. Stosowane są następujące zasady:

- Słowa kluczowe pisane są małymi literami, np. `for`.
- Nazwy obiektów wbudowanych pisane są od dużej litery, pozostałe litery są małe, np. `Date`.
- Nazwy obiektów DOM pisane są małymi literami, ale w nazwach metod tych obiektów dopuszczane są małe i duże litery, np. `toLowerCase`.
- Definiowanym zmiennym, obiektom, funkcjom można nadawać nazwy pisane w dowolny sposób. Oprócz liter w nazwach można stosować cyfry i znak podkreślenia. Nazwa może zaczynać się literą lub znakiem podkreślenia. W nazwie nie można używać słów zarezerwowanych.
- Białe znaki, typu spacja, znak tabulacji itp., są nieistotne.

3.3.3. Komentarze

Komentarze są ignorowane w trakcie przetwarzania kodu. Mogą być umieszczane w dowolnym miejscu kodu. Komentarzem jest każdy wiersz rozpoczynający się od znaków //.

Przykład 3.3

```
// tutaj znajduje się komentarz
```

Komentarz rozpoczynający się od // można także umieszczać po instrukcji.

```
a =b + c // suma wartości
```

Jeżeli komentarz zawiera wiele wierszy, można użyć ograniczników /* (do rozpoczęcia komentarza) i */ (do zakończenia komentarza).

Tak zapisane komentarze dopuszczalne są tylko wewnątrz znaczników <script> i </script>.

3.3.4. Zmienne

Zmienne służą do przechowywania danych i wyników w celu dalszego ich wykorzystania. W języku JavaScript zmienne deklaruje się za pomocą słowa kluczowego var poprzedzającego nazwę zmiennej. Nazwa może zawierać litery, cyfry i znak podkreślenia. Nie może zawierać znaków narodowych. Wielkość liter używanych w nazwach ma znaczenie. Nazwa zmiennej nie może rozpoczynać się od cyfry. Tworzone zmienne nie mają określonego typu. Typ jest przypisywany do zmiennej po nadaniu jej wartości. Typ danych nie jest przypisany do zmiennej na stałe i może ulegać zmianie.

Przykład 3.4

```
var x,y, Nazwa;
var miasto="Warszawa";
var Miasto=5;
```

Przykład 3.5

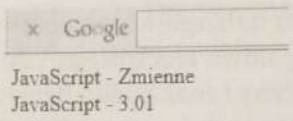
```
<html>
<head>
<title>JavaScript - Deklaracja zmiennych</title>
<script type="text/javascript">
<!--
var x = "JavaScript - ";
var y = "Zmienne";
document.write(x + y + "<br>");

var y = 3.01;
document.write(x + y);

//-->
</script>
</head>
<body>
```

```
...
</body>
</html>
```

W podanym przykładzie zmiennym x i y zostały przypisane ciągi znaków i połączony ciąg znaków został wyświetlony na ekranie. Następnie zmiennej y przypisano wartość liczbową i ponownie połączony ciąg znaków został wyświetlony na ekranie. Zmienna y zmieniła swój typ (rysunek 3.1).



Rysunek 3.1. Rezultat zmiany typu zmiennej y

3.3.5. Skalarne typy danych

W języku JavaScript dopuszczane są następujące typy danych:

- typ liczbowy,
- typ łańcuchowy,
- typ logiczny,
- typ null.

Typ liczbowy

Służy do zapisywania liczb. Można zapisywać liczby w formatach wykładniczym, dziesiętnym, ósemkowym i szesnastkowym. Jeżeli liczba zostanie poprzedzona cyfrą zero (prefiks `0`), to jest traktowana jako wartość ósemkowa (np. `042`). Jeżeli zostanie poprzedzona ciągiem znaków `0x` lub `0X`, to jest traktowana jako wartość szesnastkowa (heksadecymalna) (np. `0x23A`). Wartości liczbowe mogą być zapisywane w notacji wykładniczej (np. `3E-2`). Jeżeli liczba nie jest poprzedzona żadnym znakiem lub jest poprzedzona znakiem +, jest to wartość dodatnia. Jeżeli jest poprzedzona znakiem -, jest to wartość ujemna.

Przykład 3.6

```
var a = 12;
var b = 037;
var c = 0xACB;
var d = 0.12E-2;
```

Typ łańcuchowy string

Zawiera ciągi znaków o dowolnej długości. Ciąg znaków musi być umieszczony w ogranicznikach typu cudzysłów lub apostrof, np. `"Janek"`, `'Nowak'`.

Przykład 3.7

```
var a = "Warszawa";
var b = 'komputer';
```

Typ logiczny boolean

Przyjmuje jedną z dwóch wartości prawda (`true`) lub fałsz (`false`). Wartości typu logicznego są wykorzystywane przy budowaniu wyrażeń logicznych, porównywaniu danych, określaniu, czy wykonywana operacja zakończyła się sukcesem.

Przykład 3.8

```
var k = true;
```

Typ null

Jest to typ specjalny określający wartość pustą (`null`). Nie przechowuje żadnej wartości.

Typ undefned

Jest to typ zawierający zmienne, którym nie została nadana żadna wartość.

3.3.6. Złożone typy danych

W języku JavaScript zostały zdefiniowane również złożone typy danych. Oznacza to, że wewnątrz nich można przechowywać więcej niż jedną wartość. Złożonymi typami danych są:

- typ `object`,
- typ `array`.

Typ object

Służy do reprezentacji obiektów. Wykorzystywane są obiekty wbudowane oraz udostępniane przez przeglądarkę.

Typ array

Typ array jest typem tablicowym, który służy do przechowywania wielu wartości.

3.3.7. Operatory

Operatory w języku JavaScript zostały podzielone na następujące grupy:

- arytmetyczne,
- porównania,
- bitowe,
- logiczne,

- przypisania,
- pozostałe.

Operatory arytmetyczne

Służą do wykonywania operacji arytmetycznych. W tej grupie znajdują się też operatory `inkrementacji` (zwiększania) i `dekrementacji` (zmniejszania). Operatory arytmetyczne są dwuargumentowe, natomiast operatory inkrementacji i dekrementacji są jednoargumentowe (tabela 3.1).

Tabela 3.1. Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	`a + b`
-	odejmowanie	`a - b`
*	mnożenie	`a * b`
/	dzielenie	`a / b`
%	modulo (reszta z dzielenia)	`a % b`
++	inkrementacja	`x++, ++x`
--	dekrementacja	`x--, --x`

Operator inkrementacji powoduje zwiększenie wartości o jeden. Może występować w postaci przedrostkowej (`++x`) lub przyrostkowej (`x++`). Operacja `x++` zwiększa wartość zmiennej po jej wykorzystaniu, natomiast `++x` przed jej wykorzystaniem. Jak widać, oba operatory zwiększają wartość zmiennej, ale nie są równoważne.

Operator dekrementacji działa analogicznie, tylko zamiast zwiększać wartości zmiennych, zmniejsza je.

Przykład 3.9

```
<script type="text/javascript">
<!--
for (var i = 0; i < 10; i++) {
document.write("<br> Ile razy zostanie wykonana pętla? " + i);
}
// -->
</script>
```

Operatory porównania

Operatory porównania porównują argumenty. Ich wynikiem jest wartość logiczna `true` (prawda) lub `false` (fałsz) (tabela 3.2).

Tabela 3.2. Operatory porównania

Operator	Działanie	Przykład
==	Wynik `true`, gdy argumenty są równe.	`a == b`
!=	Wynik `true`, gdy argumenty są różne.	`a != b`
===	Wynik `true`, gdy argumenty są tego samego typu i są równe.	`a === b`
!==	Wynik `true`, gdy argumenty są różne lub są różnych typów.	`a != b`
>	Wynik `true`, gdy argument pierwszy jest większy od drugiego.	`a > b`
<	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego.	`a < b`
>=	Wynik `true`, gdy argument pierwszy jest większy od drugiego lub jest mu równy.	`a >= b`
<=	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego lub jest mu równy.	`a <= b`

Operatory bitowe

Operatory bitowe umożliwiają wykonanie operacji na poszczególnych bitach liczb (tabela 3.3).

Tabela 3.3. Operatory bitowe

Operator	Działanie	Przykład
&	iloczyn bitowy (AND)	`a & b`
|	suma bitowa (OR)	`a | b`
~	negacja bitowa (NOT)	`~a`
^	bitowa różnica symetryczna	`a ^ b`
>>	przesunięcie bitowe w prawo	`a >> n`
<<	przesunięcie bitowe w lewo	`a << n`
>>>	przesunięcie bitowe w prawo z wypełnieniem zerami	`a >>> n`

Operatory logiczne

Operatory logiczne wykonują operacje na argumentach, które posiadają wartość logiczną (`true` lub `false`) (tabela 3.4).

Tabela 3.4. Operatory logiczne

Operator	Działanie	Przykład
`&&`	iloczyn logiczny (AND)	`a && b`
`||`	suma logiczna (OR)	`a || b`
`!`	negacja logiczna (NOT)	`!a`

Wynik iloczynu logicznego przyjmuje wartość `true` tylko wtedy, gdy obydwa argumenty mają wartość `true`. W pozostałych przypadkach wynik przyjmuje wartość `false`.

Wynik sumy logicznej przyjmuje wartość `false` tylko wtedy, gdy obydwa argumenty mają wartość `false`. W pozostałych przypadkach wynik przyjmuje wartość `true`.

Negacja logiczna zmienia wartość argumentu na przeciwną.

Operatory przypisania

Operatory przypisania przypisują wartości argumentom znajdującym się po lewej stronie operatora. Oprócz prostej operacji przypisania pozwalają na połączenie operacji przypisania z inną operacją, np. dodawania.

Zapis `i += 7` oznacza w praktyce to samo co zapis `i = i + 7`. Stosowanie takich skróconych zapisów upraszcza tworzenie bardziej rozbudowanych skryptów. W języku JavaScript istnieje duża grupa operatorów tego typu (tabela 3.5).

Tabela 3.5. Niektóre operatory przypisania

Operator	Przykład	Znaczenie
`=`	`x = y`	x = y
`+=`	`x += y`	x = x + y
`-=`	`x -= y`	x = x – y
`*=`	`x *= y`	x = x * y
`/=`	`x /= y`	x = x / y
`%=`	`x %= y`	x = x % y

3.4. Instrukcje sterujące

Instrukcje sterujące pozwalają zmienić kolejność wykonywania poleceń zapisanych w skrypcie w zależności od spełnienia lub niespełnienia określonych warunków. W języku JavaScript istnieje cała grupa instrukcji sterujących.

3.4.1. Instrukcja warunkowa

Instrukcja warunkowa służy do sprawdzenia w programie warunku i w zależności od tego, czy wynik będzie prawdą, czy fałszem, dalszego wykonywania instrukcji. Do takiego sprawdzania służy instrukcja `if ... else`.

```
If (warunek) {
instrukcje do wykonania, gdy warunek jest prawdziwy
}
else {
instrukcje do wykonania, gdy warunek jest fałszywy
}
```

Przykład 3.10

```
<html>
<head>
<title>JavaScript - instrukcje warunkowe</title>
<script type="text/javascript">
<!--
var a = -5;
var b = 3;
if(a > b){
document.write("Wartość zmiennej a jest większa od wartości zmiennej b.");
}
else{
document.write("Wartość zmiennej a jest równa lub mniejsza od wartości
zmiennej b.");
}
// -->
</script>
</head>
<body>
...
</body>
</html>
```

3.4.2. Instrukcja switch

Instrukcja `switch` jest instrukcją wyboru. Pozwala sprawdzić zestaw warunków i wykonać różne działania w zależności od wyników porównania.

```
switch (wyrażenie) {
   case wartość1:
      instrukcje1;
      break;
   case wartość2:
      instrukcje2;
      break;
   case wartość3:
      instrukcje3;
      break;
   default:
      instrukcje4;
}
```

Działanie instrukcji wygląda następująco: „Sprawdź wartość *wyrażenia* i jeżeli wynikiem jest *wartość1*, wykonaj *instrukcje1* i wyjdź z bloku `switch` (polecenie `break`). Jeżeli wynikiem jest *wartość2*, to wykonaj *instrukcje2* i wyjdź z bloku `switch`. Jeżeli wynikiem jest *wartość3*, to wykonaj *instrukcje3* i wyjdź z bloku `switch`. Jeżeli wartość jest inna, wykonaj *instrukcje4* i zakończ blok `switch`".

Przykład 3.11

```
<script type="text/javascript">
<!--
var a = 20;
var b = 7;
switch(a * b){
case 10:
document.write("Wynik mnożenia wynosi 10");
break;
case 40:
document.write("Wynik mnożenia wynosi 40");
break;
case 100:
document.write("Wynik mnożenia wynosi 100");
```

```
break;
default:

document.write("Nieznany wynik mnożenia");
}
// -->
</script>
```

3.4.3. Pętle

Pętle są używane do wykonywania powtarzających się czynności. W języku JavaScript występują następujące rodzaje pętli: `for`, `while`, `do ... while`.

Pętla for

Pętla typu `for` służy do budowania pętli, gdy został podany licznik wykonań pętli oraz warunek, który musi być spełniony, aby kolejny raz wykonać pętlę. Składnia instrukcji jest następująca:

```
for (wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące) {
blok instrukcji;
}
```

- *wyrażenie początkowe* — inicjuje zmienną, która jest używana jako licznik pętli,
- *wyrażenie warunkowe* — określa warunek, który musi być spełniony, aby pętla została wykonana kolejny raz,
- *wyrażenie modyfikujące* — modyfikuje zmienną, która jest licznikiem.

Pętlę `for` wykorzystuje się zwykle wtedy, gdy znamy liczbę wykonywanych powtórzeń.

Przykład 3.12

```
<script type="text/javascript">
<!--
for (var i = 0; i < 5; i++) {
document.write("Pętla wykonana " + i + " raz/y</br>");
}
// -->
</script>
```

W uproszczonej postaci pętla może zostać pozbawiona wyrażenia modyfikującego.

Przykład 3.13

```
<script type="text/javascript">
<!--
```

```
for (var i = 0; i < 5;) {
document.write("Pętla wykonana " + i + " raz/y</br>");
i++;
}
// -->
</script>
```

W podanym przykładzie zwiększanie licznika zostało przeniesione z pętli do bloku instrukcji. Należy pamiętać o tym, aby średnik występujący po wyrażeniu `i < 5` pozostał w pętli, ponieważ jest on niezbędny do jej prawidłowego działania.

W podobny sposób można postąpić z wyrażeniem początkowym, przenosząc je do bloku przed pętlą.

Przykład 3.14

```
<script type="text/javascript">
<!--
var i = 0;
for (;i < 5;) {
document.write("Pętla wykonana " + i + " raz/y</br>");
i++;
}
// -->
</script>
```

W tym przypadku również średnik występujący przed wyrażeniem `i < 5` powinien pozostać w pętli. Podobnie jak poprzednio, jest on niezbędny do prawidłowego jej działania.

Pętla while

Pętla `while` jest zwykle wykorzystywana wtedy, gdy liczba wykonywanych powtórzeń nie jest znana. Składnia instrukcji jest następująca:

```
while (wyrażenie warunkowe) {
blok instrukcji;
}
```

Blok instrukcji jest wykonywany w pętli, dopóki *wyrażenie warunkowe* jest prawdziwe. Konstrukcja ta znaczy: „Dopóki *wyrażenie warunkowe* jest prawdziwe, wykonuj *instrukcje*".

Przykład 3.15

```
<script type="text/javascript">
<!--
i = 0;
while (i++ < 5) {
document.write("Pętla wykonana " + i + " raz/y</br>");
}
// -->
</script>
```

Pętla do ... while

Pętla `do ... while` jest odmianą pętli `while`. Jej składnia jest następująca:

```
do {
blok instrukcji;
}
while (wyrażenie warunkowe);
```

Konstrukcja ta oznacza: „Wykonuj *instrukcje*, dopóki *wyrażenie warunkowe* jest prawdziwe". W pętli `do ... while` *blok instrukcji* jest wykonywany co najmniej raz, nawet jeżeli warunek zapisany w *wyrażeniu warunkowym* jest fałszywy — ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

Przykład 3.16

```
<script type="text/javascript">
<!--
var i = 1;
do {
document.write("Pętla wykonana" + i + "raz/y</br>");
}
while (i++ <= 5);
// -->
</script>
```

Instrukcja break

Instrukcja `break` jest instrukcją modyfikującą zachowanie się pętli. Służy do przerwania jej wykonywania.

Przykład 3.17

```
<script type="text/javascript">
<!--
var i = 0; k = 5;
do {
j = k * i;
document.write("Wynik " + j + "</br>");
if (j > 30) break;
}
while (i++ < 10);
// -->
</script>
```

Pętla `do ... while` będzie wykonywana, dopóki `i < 10`, ale jeżeli wartość zmiennej `j` obliczanej w pętli przekroczy `30`, nastąpi przerwanie wykonywania powtarzających się instrukcji i wyjście z pętli.

Instrukcja continue

Instrukcja `continue`, podobnie jak `break`, służy do modyfikowania zachowania się pętli. Po jej napotkaniu następuje przerwanie wykonywania bieżącej iteracji i przejście na jej początek.

Przykład 3.18

```
<script type="text/javascript">
<!--
for (var i = 0; i < 30; i++) {
if ((i%3) != 0)
continue;
document.write(i + "; ");
}
// -->
</script>
```

Podany kod wyświetli liczby z zakresu od `0` do `30` podzielne przez 3. Jeżeli wynik dzielenia przez trzy nie jest liczbą całkowitą, następuje przerwanie wykonywania pętli i powrót na jej początek (liczby niepodzielne przez 3 nie będą wyświetlane).

3.5. Funkcje

Funkcja w języku JavaScript to jedno lub więcej poleceń zgrupowanych w całość przy pomocy nawiasów klamrowych { }. Tak zdefiniowana funkcja może zawierać listę argumentów umieszczonych w nawiasach okrągłych ( ) oddzielonych przecinkami — oraz może zwracać wartość.

3.5.1. Definiowanie funkcji

W języku JavaScript można definiować własne funkcje. Definicja funkcji musi zawierać słowo kluczowe `function`, po którym następuje nazwa funkcji, następnie w nawiasach okrągłych powinny zostać wymienione parametry funkcji oddzielone przecinkami. W nawiasach klamrowych jest zapisywana treść funkcji, na której końcu powinna zostać wstawiona instrukcja `return`, określająca zwracaną przez funkcję wartość. Ponieważ funkcja zawsze zwraca jakąś wartość, to jeżeli nie zostanie to podane w jawny sposób z wykorzystaniem instrukcji `return`, zwracana jest automatycznie wartość `undefined`.

Instrukcja `return` powoduje przerwanie wykonywania poleceń funkcji i powrót do miejsca, z którego funkcja została wywołana. Definicja funkcji ma postać:

```
function nazwa_funkcji (argumenty_funkcji) {instrukcje}
```

Można definiować funkcje bezparametrowe:

```
function nazwa_funkcji () {instrukcje}
```

Przykład 3.19

```
function Suma (a,b)
{
var c = a + b;
return c;
}
```

Wywołanie funkcji

Wywołanie funkcji nastąpi po podaniu w kodzie skryptu jej nazwy wraz z parametrami umieszczonymi w nawiasach.

Przykład 3.20

```
<script type="text/javascript">
<!--
function trzy(z)
{
  for (var i = 0; i <= z; i++) {
    if ((i%3) != 0)
```

```
        continue;
        document.write(i + "; ");
    }
    document.write("</br>");
}
trzy(90);
trzy(120);
// -->
</script>
```

W podanym przykładzie została zdefiniowana funkcja `trzy(z)`, która wyświetla ciąg liczb podzielnych przez 3. Argument `z` określa zakres wyświetlania liczb. Funkcja `trzy(z)` została wywołana dwukrotnie, raz z argumentem `z=90`, drugi raz z argumentem `z=120`.

Argumenty

Przy wywołaniu funkcji należy podać wartości jej argumentów. Jeżeli zostaną one pominięte, JavaScript przypisze argumentom wartość `undefined`. Jeżeli otrzyma ich więcej, niż jest wymagane, zignoruje niepotrzebne argumenty. Możliwe jest tworzenie funkcji, które będą miały zmienną liczbę argumentów. Wykorzystywana jest do tego tablica `arguments`, która automatycznie jest tworzona dla każdej funkcji.

Przykład 3.21

```
function lista_arg()
{
return arguments;
}
```

Wywołanie funkcji bez podania parametrów nie zwróci żadnej wartości.

```
lista_arg();
```

Wynik:

```
[ ]
```

Wywołanie funkcji z parametrami zwróci listę wartości.

```
lista_arg(1,4,5,9,"A","koło");
```

Wynik:

```
[1,4,5,9,"A","koło"]
```

Wykorzystując tablicę `arguments`, można zdefiniować funkcję, która będzie sumowała dowolną liczbę parametrów.

Przykład 3.22

```
<script type="text/javascript">
<!--
function suma_dow()
{
var i, wynik = 0;
var l_param = arguments.length;
for (i = 0; i < l_param; i++)
   {
   wynik += arguments[i];
   }
return wynik;
}
// -->
</script>
```

Użyta w definicji funkcji właściwość `arguments.length` zwróci liczbę parametrów podanych podczas wywołania funkcji.

Wywołanie funkcji z różną liczbą parametrów powinno dać zawsze poprawny wynik, np.

```
suma_dow(3,5,7);
```

da wynik 15,

```
suma_dow(1,2,3,4,5,6,7,8,9);
```

da wynik 45.

3.5.2. Zasięg zmiennych

Zasięg zmiennej jest to obszar, w którym można odwoływać się bezpośrednio do zmiennej. Zmienna może być:

- lokalna,
- globalna.

Zmienne **globalne** są widoczne w całym skrypcie. Są to zmienne, które zostały zdefiniowane poza funkcją.

Zmienne **lokalne** mają zasięg lokalny i są definiowane wewnątrz funkcji. Ich zasięg dotyczy tylko funkcji, w której zostały zdefiniowane, i poza nią nie są widoczne.

Jeżeli zmienna zostanie zadeklarowana bez użycia słowa kluczowego `var`, będzie miała zasięg globalny. Powinniśmy starać się ograniczać liczbę zmiennych globalnych i deklarować zmienne słowem kluczowym `var`.

Przykład 3.23

```
<script type="text/javascript">
<!--
function suma_dow()
{
var i;
wynik = 0;
var l_param = arguments.length;
for (i = 0; i < l_param; i++)
  {
  wynik += arguments[i];
  }
return wynik;
}
suma_dow(3,5,7);
document.write("Suma argumentów: " + wynik);
// -->
</script>
```

W podanym przykładzie zmienna `wynik` zadeklarowana wewnątrz funkcji jest zmienną globalną, ponieważ została zadeklarowana bez użycia słowa kluczowego `var`. Może zatem być wykorzystana w dalszej części skryptu.

3.5.3. Funkcje wbudowane

W języku JavaScript istnieje duża grupa funkcji wbudowanych (predefiniowanych). Są to między innymi:

- `parseInt()`
- `parseFloat()`
- `isNaN()`
- `isFinite()`
- `eval()`
- `alert()`

parseInt()

Funkcja pobiera argument dowolnego typu (najczęściej tekstowego) i zamienia go na liczbę. Gdy operacja nie powiedzie się, zwraca wartość `NaN`. Funkcja może pobierać jeszcze drugi argument, określający podstawę liczby (dziesiętna, szesnastkowa, binarna).

Jeżeli ten argument nie zostanie podany, domyślnie podstawa jest dziesiętna. Ale jeżeli pierwszy argument zaczyna się od `"0x"`, podstawa będzie szesnastkowa, natomiast gdy zaczyna się od `"0"`, podstawa będzie ósemkowa.

`NaN` oznacza, że wartość nie jest liczbą.

Przykład 3.24

Funkcja	Wynik
`parseInt("123");`	123
`parseInt("123AB",16);`	74667
`parseInt("123",8);`	83
`parseInt("123AB",8);`	NaN
`parseInt("0377");`	255
`parseInt("0x373");`	883

parseFloat()

Funkcja działa podobnie jak `parseInt()`, ale działa również na ułamkach. Pobiera argument dowolnego typu (najczęściej tekstowego) i zamienia go na liczbę mogącą zawierać część ułamkową. Funkcja poprawnie interpretuje zapisy liczby w postaci wykładniczej.

Przykład 3.25

```
parseFloat("432.37");
```

isNaN()

Funkcja sprawdza, czy wartość podana jako parametr nie jest liczbą. Przy jej pomocy można np. sprawdzić, czy funkcji `parseInt()` udało się z powodzeniem zamienić podaną wartość na liczbę. Funkcja zwraca wartość `true`, gdy argument wejściowy nie jest liczbą, lub `false`, gdy argument jest liczbą.

Przykład 3.26

Funkcja	Wynik
`isNaN(NaN)`	true
`isNaN(567)`	false
`isNaN(37.2)`	false
`isNaN(parseInt("zx23"))`	true

isFinite()

Funkcja sprawdza, czy wartość podana jako parametr to liczba różna od `Infinity` oraz od `NaN`. Gdy argument wejściowy ma wartość `Infinity` lub `NaN`, funkcja zwraca `false`, gdy jest liczbą, zwraca `true`.

Przykład 3.27

Funkcja	Wynik
`isFinite(Infinity)`	`false`
`isFinite(-Infinity)`	`false`
`isFinite(67)`	`true`
`isFinite(2E12)`	`true`

eval()

Funkcja ta pobiera ciąg znaków i uruchamia go jako kod języka JavaScript. Zaleca się nieużywanie tej funkcji.

Przykład 3.28

```
eval( "var x = 5;");
```

alert()

Funkcji `alert()` nie ma w specyfikacji języka, ale można jej używać w środowisku przeglądarki. Służy do wyświetlania komunikatów w oknie dialogowym.

Przykład 3.29

```
<html> <head>
<script type="text/javascript">
function pokaz(){ alert("Uwaga, alarm!");}
</script>
</head> <body>
<input type="button" onclick="pokaz()"
value="Pokaż okno alarmu" />
</body> </html>
```

W podanym przykładzie skrypt został umieszczony wewnątrz kodu HTML. W sekcji `<body>` przy użyciu znacznika `<input type="button">` utworzono przycisk, do którego zostało przypisane zdarzenie `onclick` (przy kliknięciu myszą), i została z nim związana funkcja `pokaz()` zdefiniowana w skrypcie.

Funkcje powinny być definiowane w nagłówku kodu HTML.

W języku JavaScript istnieje predefiniowany obiekt `Function`, za pomocą którego można definiować nowe obiekty będące funkcjami.

```
fun = new Function(lista_parametrów, treść_funkcji)
```

Tworzenie funkcji za pomocą obiektu `Function` jest mniej efektywne niż jej tworzenie instrukcją `function`.

Przykład 3.30

```
var kolor_t = new Function("document.bgColor='antiquewhite' ")
```

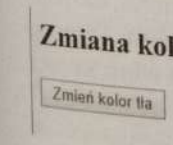
W wyniku zmiennej `kolor_t` została przypisana funkcja ustawiająca kolor tła dokumentu. Wywołanie tak zdefiniowanej funkcji nastąpi przez zmienną `kolor_t`.

Przykład 3.31

```
<html>
<head>
<title>JavaScript - Funkcja</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script LANGUAGE="JavaScript" type="text/javascript">
var kolor_t = new Function("document.bgColor='antiquewhite' ")
document.form1.colorButton.onclick=kolor_t
</script>
</head>
<body>
<h2>Zmiana koloru tła</h2>
<p>
<INPUT NAME="kolorPrzycisku" TYPE="button"
      VALUE="Zmień kolor tła"
      onClick="kolor_t()">
</p>
</body>
</html>
```

Gdy kod zostanie wykonany, po kliknięciu przycisku nastąpi zmiana koloru tła strony (rysunek 3.2).



Rysunek 3.2. Zmiana koloru tła strony

Zadanie 3.1

Zmodyfikuj kod podany w przykładzie 3.31. Zamiast tworzenia zmiennej `kolor_t` zdefiniuj funkcję `zmiana_koloru` i wywołaj ją w kodzie, tak aby uzyskać efekt podobny do pokazanego w przykładzie.

3.6. Obiekty

Język JavaScript jako język zorientowany obiektowo udostępnia wiele wbudowanych obiektów. Daje również możliwość odczytywania ich właściwości oraz korzystania z ich metod. Właściwości obiektu reprezentują jego cechy (np. liczba znaków łańcucha, wymiary okna) lub pozwalają określić jego stan. Nazywane są również polami obiektu, zmiennymi lub zmiennymi składowymi. Metody to funkcje, które wykonują różne operacje na obiekcie.

Do właściwości i metod można się odwołać podobnie jak do zwykłych zmiennych i funkcji, trzeba tylko przed ich nazwą umieścić nazwę obiektu, którego są elementami (np. zmienną, która przechowuje dany obiekt), i kropkę.

Przykład 3.32

```
var napisz="Witaj w szkole";
document.write(napisz.toUpperCase());
document.write(napisz.length);
```

W podanym przykładzie zmienna `napisz` przechowuje obiekt. Funkcja `UpperCase()` jest metodą obiektu, a `napisz.length` jego właściwością. Wynikiem będzie wypisanie tekstu: `"WITAJ W SZKOLE15"`.

Każdy element strony internetowej jest traktowany jako obiekt umieszczony w obiekcie `document`. Obiekty języka JavaScript zawierają informacje opisujące stronę i jej środowisko.

3.6.1. Obiekty przeglądarki

Dla każdej strony internetowej zdefiniowane są następujące obiekty:

- `window`
- `navigator`
- `document`
- `location`
- `history`

window

Opisuje bieżące okno przeglądarki. Jest najważniejszym obiektem w hierarchii, dlatego odwołanie do jego właściwości lub metod nie wymaga podania nazwy obiektu. Tworzony jest automatycznie podczas otwierania okna przeglądarki. Posiada wiele właściwości przydatnych podczas tworzenia strony.

navigator

Pozwala na dostęp do informacji dotyczących przeglądarki. Służy do ustalenia wersji przeglądarki używanej przez użytkownika. Właściwości tego obiektu mogą być tylko odczytywane. Najczęściej używa się go do sprawdzenia, czy przeglądarka jest odpowiednia do zastosowanej wersji języka JavaScript.

Przykład 3.33

```
if((navigator.appName=="Firefox") &&
 (parseInt(navigator.appVersion)>=3))
 { kod skryptu }
else
document.write("Masz niewłaściwą wersję przeglądarki!!!")
```

Właściwość `appName` zawiera nazwę przeglądarki, `appVersion` zawiera numer wersji przeglądarki, funkcja `parseInt()` zamienia dowolny łańcuch na liczbę.

document

Zawiera informacje o bieżącym dokumencie HTML. Poprzez jego właściwości mamy wpływ na elementy strony (np. kolor tła, kolor czcionki). Jego metody umożliwiają wyświetlenie np. tekstu w oknie przeglądarki.

location

Zawiera informacje o bieżącym adresie URL. Jego właściwości odpowiadają kolejnym członom adresu. Ogólna postać lokalizatora URL to:

```
protocol://host:port/path#fragment?query
```

Właściwość `protocol` to łańcuch znakowy określający protokół, zgodnie z którym należy nawiązać łączność z podanym serwerem (np. *http:*, *ftp:*, *file:*), `host` to łańcuch znakowy określający nazwę serwera z bieżącego adresu, `port` to łańcuch znakowy odpowiadający portowi, przez który należy połączyć się z serwerem, `pathname` to łańcuch znakowy określający ścieżkę dostępu do dokumentu na serwerze, `hash` to łańcuch znakowy odpowiadający nazwie zakotwiczenia (przypisanie tu jakiejś wartości spowoduje przewinięcie dokumentu do wskazanego punktu), `search` to człon lokalizatora URL.

Aby zmienić adres strony wyświetlanej w oknie, wystarczy zmienić lokalizator URL.

Przykład 3.34

```
window.location = "https://www.google.pl/search?q=warszawa+lotnisko"
```

history

Zawiera historię stron odwiedzanych w bieżącej sesji. Posiada dwie właściwości:

- `current` — zawiera bieżący lokalizator URL,
- `length` — zawiera długość listy historii.

Dodatkowo strona może zawierać następujące elementy DOM:

- `form` — formularz,
- `anchor` — zakotwiczenie,
- `link` — odsyłacz,
- `image` — obrazek,
- `embed` — dodatek,
- `applet` — aplet Javy,
- `frame` — ramka,
- `area` — mapa graficzna.

Predefiniowane obiekty w JavaScript

- `String` — łańcuch tekstowy. Posiada własność `length` i metody: `slice(`, `split()`.
- `Array` — tablica. Posiada własność `length` i metody: `concat()`, `pop()`, `push(`
- `Date` — data. Posiada metody: `getMonth()`, `getDay()`.
- `Match` — obiekt matematyczny.

3.6.2. Tworzenie obiektów

Aby utworzyć nowy obiekt w języku JavaScript, należy skorzystać z konstrukcji, która zdefiniuje nazwę obiektu oraz pozwoli utworzyć jego właściwości i metody.

Przykład 3.35

```
var obiekt_1 = {
nazwisko: 'Nowacki',
imie: 'Marek',
zawod: 'informatyk',
pokaz: function ()
  {
    document.write(this.nazwisko + ' ' + this.imie)
  }
}
```

Utworzony obiekt posiada trzy właściwości (`nazwisko`, `imie`, `zawod`) i jedną metodę, `pokaz`, która wyświetla nazwisko i imię. Metoda jest funkcją zdefiniowaną w obrębie obiektu. Przy definiowaniu obiektu deklarowane właściwości i funkcje muszą być oddzielone przecinkami. Słowo kluczowe `this` pozwala odwołać się do właściwości lub metod danego obiektu z jego wnętrza. W tym wypadku metoda `pokaz` odwołuje się do właściwości obiektu `obiekt_1`.

Dla istniejących obiektów można deklarować nowe właściwości i metody.

Przykład 3.36

```
var obiekt_1 = {
nazwisko: 'Nowacki',
imie: 'Marek',
zawod: 'informatyk',
pokaz: function ()
  {
    document.write(this.nazwisko + ' ' + this.imie)
  }
}

obiekt_1.wiek = 19;
obiekt_1 wypisz_wiek = function() {alert('Wiek: ' + this.wiek + 'lat')}
obiekt_1.wypisz();
```

Zadanie 3.2

Zdefiniuj w języku JavaScript obiekt opisujący wybraną markę samochodów. Zdefiniuj jego właściwości i metody. Wyświetl w przeglądarce informacje o utworzonym obiekcie.

3.6.3. Tworzenie obiektów z użyciem konstruktora

W języku JavaScript istnieje możliwość tworzenia wielu obiektów posiadających podobne właściwości. W tym celu można posłużyć się konstruktorem obiektu. Konstruktor przypomina zwykłą funkcję.

Przykład 3.37

```
function klient(nazwisko_k,imie_k,zawod_k)
{
 this.nazwisko=nazwisko_k;
 this.imie=imie_k;
 this.zawod=zawod_k;
 wypisz = function ()
  {
    alert(this.nazwisko + ' ' + this.imie)
  }
 }
```

Został utworzony konstruktor o nazwie `klient` z właściwościami `nazwisko`, `imie`, `zawod` oraz metodą `wypisz()`. Właściwościom obiektu zostały przypisane wartości parametrów. Użyte słowo kluczowe `this` odnosi się do aktualnego obiektu i pozwala na przypisanie wartości parametru do odpowiedniego pola tego obiektu.

Właściwości obiektu istnieją w porządku, w jakim zostały zdefiniowane. Można się do nich odwoływać na dwa sposoby:

```
nazwa_obiektu.nazwa_właściwości
```

np. `klient.nazwisko`,

```
nazwa_obiektu["nazwa_właściwości"]
```

np. `klient["nazwisko"]`.

Słowo kluczowe new

Do utworzenia nowego obiektu na podstawie konstruktora stosowane jest słowo kluczowe `new`.

Przykład 3.38

```
var osoba1 = new klient('Kowalski', 'Jan', 'kierowca');
var osoba2 = new klient('Nowak', 'Anna', 'sekretarka');
```

Powstały dwa nowe obiekty `osoba1` i `osoba2` należące do klasy `klient`.

Zadanie 3.3

Sprawdź, jaki rezultat otrzymasz po wykonaniu poniższego kodu:

```
osoba1.wypisz();
osoba2.imie = 'Ewa';
osoba2.wypisz();
```

Właściwość prototype

Innym sposobem deklarowania metod i właściwości dla obiektu jest wykorzystanie właściwości `prototype`.

Przykład 3.39

```
function klient()
{
 this.nazwisko='Bielski';
 this.imie='Paweł';
 }
klient.prototype.pisz_dane = function ()
```

```
{
  document.write(this.nazwisko + ' ' + this.imie)
}
klient.prototype.zawod = 'kierowca';

var osoba1 = new klient();
osoba1.pisz_dane();
```

W definicji konstruktora nie zostały zadeklarowane żadne metody i właściwości. Dopiero po użyciu właściwości `prototype` została dodana metoda `pisz_dane` oraz właściwość `zawod`. Od tej pory każdy nowo tworzony obiekt na podstawie konstruktora *`klient`* będzie posiadał tę dodatkową właściwość i metodę.

Właściwość `prototype` może być również wykorzystana do dodawania dodatkowych metod lub właściwości do istniejących obiektów.

3.7. Obiekty wbudowane JavaScript

3.7.1. Obiekt String

Obiektem zawsze występującym w języku JavaScript jest obiekt `String`. Posiada on jedną właściwość, `length`, określającą długość łańcucha.

Przykład 3.40

```
tekst = "Obiekty języka JavaScript"
dl = tekst.length
```

Zmiennej `dl` przypisana zostanie wartość 25, określająca długość tekstu.

Obiekt `String` posiada dwa typy metod.

Pierwszy typ metod odnosi się do utworzonego łańcucha, np. metoda `substring()`, która zwraca podzbiór tego łańcucha. Jej parametry określają położenie początku i końca podzbioru.

Przykład 3.41

```
x = tekst.substring (15,19)
```

Zostanie zwrócony łańcuch `Java`.

Metodami, które można wykorzystać do zmiany wielkości liter, są: `toUpperCase()` i `toLowerCase()`. Metoda `toUpperCase()` zamienia wszystkie litery ciągu na duże a metoda `toLowerCase()` zamienia wszystkie litery ciągu na małe.

Przykład 3.42

```
x = tekst.toLowerCase()
```

Zostanie zwrócony łańcuch `obiekty języka javascript.`

Przykład 3.43

```
x = tekst.toUpperCase()
```

Zostanie zwrócony łańcuch `OBIEKTY JĘZYKA JAVASCRIPT.`

Do istniejących obiektów można dodawać nowe właściwości i można definiować dla nich nowe metody.

Do obiektu `String` dodamy dodatkową funkcjonalność, dzięki której pierwsza litera łańcucha będzie pisana dużą literą.

Przykład 3.44

```
String.prototype.duzaLitera = function() { return this.charAt(0).
toUpperCase() + this.substr(1);}
```

Metoda `charAt()` zwraca znak z pierwszej pozycji łańcucha znaków. Metoda `substr()` zwraca podzbiór łańcucha znaków. Jako parametr tej metody został podany indeks pierwszego znaku podzbioru. Zadeklarowana metoda została dołączona do obiektu `String` i od tej pory każdy nowy tekst będzie posiadał metodę, która zamieni jego pierwszą literę na dużą.

Przykład 3.45

```
var tx1 = 'komputer';
document.write(tx1.duzaLitera())
```

W wyniku wykonania skryptu wyświetli się tekst `'Komputer'`.

3.7.2. Obiekt Date

Kolejnym obiektem języka JavaScript jest obiekt specjalny `Date`, który służy do przechowywania wartości daty i czasu. Przy jego pomocy można odczytać wartość daty i czasu, można też rozłożyć te wartości na części, odczytując oddzielnie dzień, miesiąc, rok itp. Można również te części niezależnie modyfikować.

Aby odczytać bieżącą datę i czas, należy utworzyć obiekt `Date` bez parametrów.

Przykład 3.46

```
var dat_cz = new Date();
```

Można utworzyć obiekt z określoną liczbą parametrów. Tych parametrów może być od dwóch do siedmiu (rok, miesiąc, dzień, godzina, minuty, sekundy, milisekundy). Taki obiekt będzie zawierał ściśle określoną wartość daty i godziny.

Przykład 3.47

```
var data = new Date(2013,2,27);
```

W języku JavaScript wartości daty i czasu są przechowywane w formacie `timestamp`, czyli jako liczba milisekund, które upłynęły od północy 1 stycznia 1970 roku.

Do konwersji obiektu `Date` na tekst służy kilka funkcji:

- `toString()` — zwraca datę, czas oraz informacje o strefie czasowej w języku angielskim,
- `toLocaleString()` — zwraca datę i czas dla bieżących ustawień regionalnych,
- `toUTCString()` — zwraca datę, czas oraz informacje o strefie czasowej dla formatu UTC (*Universal Coordinated Time*),
- `toGMTString()` — działa jak funkcja `toUTCString()`,
- `toDateString()` — zwraca tylko datę w języku angielskim,
- `toLocaleDateString()` — zwraca tylko datę dla bieżących ustawień regionalnych,
- `toTimeString()` — zwraca tylko czas w języku angielskim,
- `toLocaleTimeString()` — zwraca tylko czas dla bieżących ustawień regionalnych.



UWAGA

Dla funkcji `toString()` i `toLocaleString()` różne przeglądarki zwracają wyniki w różny sposób.

Jednym z najczęściej spotykanych przykładów wykorzystania JavaScript na stronie WWW jest wyświetlanie daty i czasu jako elementu strony internetowej.

Przykład 3.48

```
<html>
<head>
<title>JavaScript - Data i czas</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<body>
<h2>Wyświetlam bieżącą datę i czas</h2>
<p>
<script type="text/javascript">
data_n = new Date();
data_l = data_n.toString();
data_u = data_n.toGMTString();
```

```
data_r = data_n.toLocaleString();
document.write("<b>Czas lokalny:</b> " + data_l + "<br>");
document.write("<b>Czas uniwersalny:</b> " + data_u + "<br>");
document.write("<b>Czas regionalny:</b> " + data_r + "<br>");


</script>
</p>
</body>
</html>
```

Utworzonej zmiennej o nazwie `data_n` przypisany został obiekt `Date`. Zmienne `data_l`, `data_u` i `data_r` zawierają tekstową postać czasu lokalnego, uniwersalnego i regionalnego, odpowiadającego wartości przechowywanej w zmiennej `data_n`. Do wyświetlenia wartości otrzymanych zmiennych została użyta funkcja `document.write`. W rezultacie na stronie wyświetlą się informacje o czasie lokalnym, uniwersalnym i regionalnym (rysunek 3.3).

Wyświetlam bieżącą datę i czas

Czas lokalny: Mon Jul 29 21:11:00 UTC+0200 2013
Czas uniwersalny: Mon, 29 Jul 2013 19:11:00 UTC
Czas regionalny: 29 lipca 2013 21:11:00

Rysunek 3.3. Wyświetlenie na stronie informacji o dacie i czasie

Zadanie 3.4

Przetestuj działanie skryptu z przykładu 3.49 w różnych przeglądarkach.

Zadanie 3.5

Zdefiniuj funkcję, która będzie wyświetlała datę i czas po polsku, na przykład tak jak na rysunku 3.4.

Dzisiejszy dzień to:
Środa, 12 grudnia 2012 roku,
Godzina:21:39:35

Rysunek 3.4. Data i czas wyświetlane na stronie internetowej

Zadanie 3.6

Utwórz stronę internetową, na której data i czas będą wyświetlane w postaci kartki z kalendarza.

3.7.3. Obiekt Array

Tabele służą do przechowywania wielu zmiennych. W języku JavaScript do pracy z tabelami można używać wbudowanego obiektu `Array`. Posiada on metody do manipulowania tablicami zmiennych.

Aby utworzyć nową tablicę, należy zadeklarować obiekt `Array` w postaci:

```
var NazwaTablicy = new Array()
```

lub

```
var NazwaTablicy = []
```

Jeżeli w nawiasach zostanie podana liczba `n`, to zostanie utworzona tablica zawierająca `n` pustych elementów.

Przykład 3.49

```
var Tab1 = new Array(10)
var Tab2 = [15]
```

Tablicę można również tworzyć, wstawiając do niej konkretne wartości.

Przykład 3.50

```
var Tab3 = new Array('Anna', 'Adam', 'Piotr', 'Ewa')
var Tab4 = ['Paweł', 'Marcin', 'Ela']
```

Żeby uzyskać dostęp do elementów tablicy, należy podać numer indeksu danego elementu. Elementy są indeksowane od zera.

Przykład 3.51

```
document.write(Tab3[2])
```

W wyniku wyświetli się wartość `'Piotr'`.

Aby dodać nową wartość do tablicy, należy przypisać tę wartość do odpowiedniego indeksu tablicy.

Przykład 3.52

```
var Tab5 = new Array('kot', pies', 'koń')
Tab5[3] = 'mysz';
Tab5[4] = 'chomik';
document.write(Tab5[0] + 'i' + Tab5[3])
```

W wyniku wyświetli się tekst `'kot i mysz'`.

Dzięki właściwości `length` można określić, z ilu elementów składa się tablica. Jest to bardzo przydatna właściwość, szczególnie gdy chcemy utworzyć pętlę odczytującą wszystkie elementy tablicy.

Przykład 3.53

```
var Tablica_N = new Array('Anna', 'Adam', 'Piotr', 'Ewa', 'Paweł', 'Marcin',
'Ela')
for (i = 0; i<tablica_N.length; i++)
{
document.write(Tablica_N[i] + "<br">);
}
```

W wyniku zostaną wyświetlone po kolei wszystkie elementy tablicy.

Zadanie 3.7

Zmień zapis skryptu podanego w przykładzie 3.54, tak aby wyświetlone zostały elementy tabeli od ostatniego do pierwszego. Pamiętaj, że indeksowanie elementów rozpoczyna się od zera.

Tablice wielowymiarowe

W języku JavaScript można również tworzyć tablice wielowymiarowe. Wtedy element tablicy jest opisywany za pomocą indeksu określającego jego położenie w wierszu i kolumnie.

Przykład 3.54

```
var Tablica_Z = [];
Tablica_Z[0] =  ['Anna', 'Nowak'];
Tablica_Z[1] =  ['Adam', 'Kowal'];
Tablica_Z[2] =  ['Piotr', 'Ogórek'];
Tablica_Z[3] =  ['Ewa', 'Lisowska'];
document.write('imię: ' + Tablica_Z[0][0] + 'nazwisko: ' + Tablica_Z[0]
[1] + "<br>");
document.write('imię: ' + Tablica_Z[1][0] + 'nazwisko: ' + Tablica_Z[1]
[1] + "<br>");
document.write('imię: ' + Tablica_Z[2][0] + 'nazwisko: ' + Tablica_Z[2]
[1] + "<br>");
document.write('imię: ' + Tablica_Z[3][0] + 'nazwisko: ' + Tablica_Z[3][1]);
```

Łączenie elementów tablicy

Za pomocą metody `join()` można łączyć elementy tablicy w jeden tekst. W metodzie tej można opcjonalnie podać parametr, który określi znak oddzielający kolejne elementy tablicy. Jeżeli nie zostanie podana wartość tego parametru, domyślnym znakiem będzie przecinek.

Przykład 3.55

```
var Tablica = new Array('Anna', 'Adam', 'Piotr');
document.write(Tablica.join()  + "<br>");
document.write(Tablica.join(" - ") + "<br>");
```

Odwracanie kolejności elementów tablicy

Za pomocą metody `reverse()` można odwrócić kolejność elementów tablicy.

Przykład 3.56

```
var Tablica = new Array('Anna', 'Adam', 'Piotr');
document.write(Tablica.join()  + "<br>";
Tablica.reverse()
document.write(Tablica.join() + "<br>");
```

Sortowanie

Do sortowania elementów tablicy służy metoda `sort()`.

Przykład 3.57

```
var Tablica = new Array('Paweł', 'Anna', 'Maria', 'Adam', 'Piotr');
Tablica.sort()
document.write(Tablica.join());
```

Domyślnie tablica jest sortowana leksykograficznie. Powoduje to, że liczba 12459 będzie mniejsza od 4567, ponieważ cyfra na pierwszej pozycji jest mniejsza. Aby temu zaradzić, można sortować tablicę według własnych kryteriów. Należy skorzystać z dodatkowego parametru metody `sort()`. Parametrem będzie własna funkcja sortująca. Tworząc taką funkcję, należy pamiętać o trzech zasadach, które muszą być spełnione:

- jeżeli `funkcja(a,b)` zwróci wartość mniejszą od `0`, to wartości `a` zostanie nadany indeks mniejszy od indeksu przyznanego wartości `b`,
- jeżeli `funkcja(a,b)` zwróci wartość równą `0`, to wartości indeksów pozostaną bez zmian,
- jeżeli `funkcja(a,b)` zwróci wartość większą od `0`, to wartości `a` zostanie nadany indeks większy od indeksu przyznanego wartości `b`.

Stosując się do tych zasad, można tworzyć własne metody sortowania w tablicy.

Przykład 3.58

```
function porownaj(a,b)
{
return a - b
}
var Tablica = new Array(27, 100, 10, 450, 1654, 320);
document.write('Bez sortowania: ' + Tablica.join() );

document.write('Sortowanie domyślne: ');
Tablica.sort()
document.write(Tablica.join());

document.write('Sortowanie poprawne: ');
Tablica.sort(porownaj)
document.write(Tablica.join() );
```

W podanym przykładzie została zdefiniowana funkcja `porownaj(a,b)`, która zwróci wartość mniejszą od zera, równą zero lub większą od zera. W zależności od zwróconej wartości elementy tablicy zostaną uporządkowane od wartości najmniejszej do największej.

Zadanie 3.8

Utwórz tablicę, która będzie zawierała elementy zawierające cyfry i litery. Zdefiniuj funkcję, która pozwoli uporządkować elementy tabeli w ten sposób, że najpierw będą umieszczone litery w kolejności alfabetycznej, a następnie cyfry w kolejności od wartości najmniejszej do największej.

Zadanie 3.9

Utwórz tabelę „trzy na trzy".

```
var tablica = new Array(1, 2, 3);
document.write(tablica[1] + "<br>");
tablica[1] = 123;
document.write(tablica[1] + "<br>");
```

Umieść w niej obrazki. Zdefiniuj kod, który spowoduje zmianę wyświetlanych na stronie obrazków co 2 sekundy.

3.8. Obiekty DOM

DOM (ang. *Document Object Model*) to sposób reprezentacji złożonych dokumentów XML i HTML w postaci modelu obiektowego.

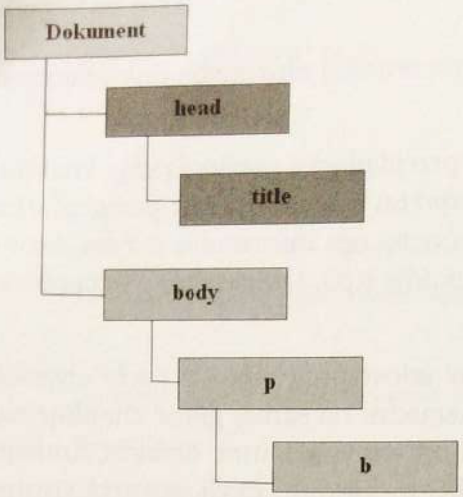
Gdy kod strony jest wczytywany do przeglądarki, przeglądarka zamienia ciąg znaków na stronę internetową. Informacje na temat interpretacji kodu HTML przeglądarka przechowuje w elementach będących obiektami (są to np. informacje o tym, które elementy przedstawić w postaci nagłówków, paragrafów itp.). Obiekty te tworzą obiektowy model dokumentu.

DOM opisuje hierarchię obiektów na stronie oraz udostępnia metody i właściwości, które umożliwiają manipulowanie nimi. W tej hierarchii na samej górze znajduje się okno przeglądarki, czyli obiekt `window`. Zawiera on wszystkie inne obiekty, funkcje i właściwości strony. W oknie znajduje się obiekt `document`, czyli otwarta strona internetowa. W obiekcie `document` znajdują się obiekty strony. W skryptach definiujemy różne działania związane z istniejącymi obiektami, czyli manipulujemy przez skrypty obiektami strony internetowej. Dzięki skryptom można wczytać nową stronę do przeglądarki, zmienić elementy dokumentu, otwierać okna lub modyfikować tekst na stronie. Dzięki DOM język JavaScript staje się narzędziem tworzenia dynamicznych stron internetowych.

Przykład 3.59

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tytuł strony</title>
</head>
<body>
<p>Moja strona <b>internetowa</b></p>
</body>
</html>
```

Podany w przykładzie dokument można rozrysować w postaci drzewa (rysunek 3.5). Na samej górze jest dokument HTML, niżej znajdują się węzły (`nodes`).



Rysunek 3.5. Hierarchia obiektów przykładowej strony internetowej

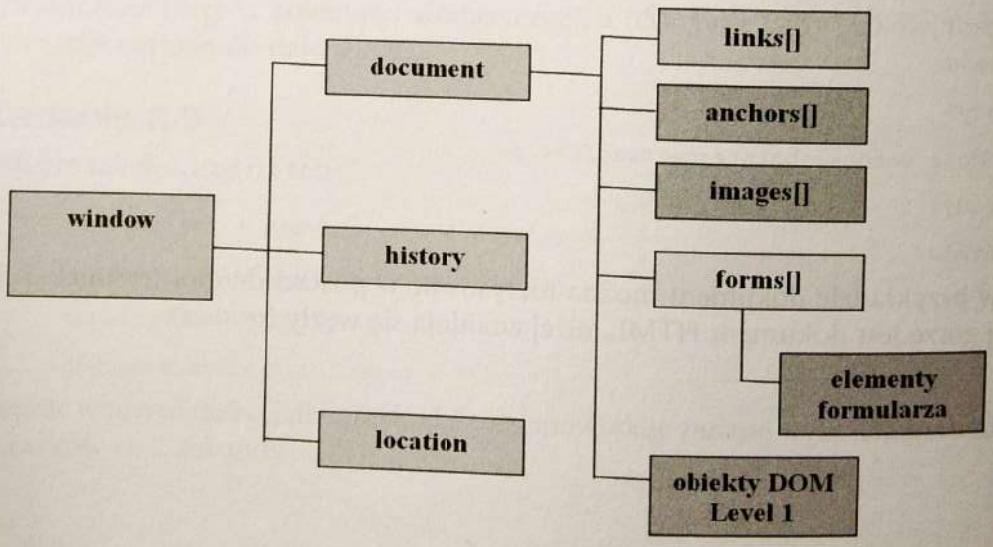
3.8.1. Hierarchia obiektów DOM

Odwołując się do obiektu, należy używać nazw obiektów nadrzędnych oddzielonych kropkami, po których następuje nazwa wybranego obiektu.

Przykład 3.60

```
window.document.link1
```

Wycinek hierarchii DOM z najważniejszymi obiektami strony internetowej został pokazany na rysunku 3.6.



Rysunek 3.6. Hierarchia obiektów DOM

3.8.2. Obiekty przeglądarki

Obiekt window

Obiektem nadrzędnym dla wszystkich obiektów jest obiekt `window`, który zawiera okno przeglądarki. W danej chwili może istnieć wiele obiektów `window`. Każdy z nich reprezentuje otwarte okno przeglądarki. Odwołanie do jego właściwości lub metod nie wymaga podania nazwy obiektu. Tworzony jest automatycznie podczas otwierania okna przeglądarki. Do otwierania nowego okna używamy metody `open()`. Jako parametry metody występują adres URL otwieranej strony oraz nazwa wewnętrzna okna (nie mylić z nazwą wyświetlaną przez przeglądarkę zdefiniowaną metatagiem `<title></title>`):

```
window.open('http://helion.pl','Wydawnictwo');
```

Do określania rozmiaru okna mogą być używane właściwości:

- `window.innerHeight` — wysokość okna przeglądarki,
- `window.innerWidth` — szerokość okna przeglądarki.

Do zamknięcia okna używana jest metoda `close()`.

Obiekt document

Obiekt `document` reprezentuje stronę internetową (dokument HTML). Jest on potomkiem obiektu `window`. Za pomocą polecenia `window.document` można odwołać się do bieżącego dokumentu. Można to zrobić również za pomocą polecenia `document`. Odwołanie nastąpi do bieżącego dokumentu w bieżącym oknie. Jeżeli zostało otwarte kilka okien, to aby określić, do którego dokumentu powinno nastąpić odwołanie, należy podać nazwę okna i nazwę dokumentu.

Informacje o bieżącym dokumencie otrzymamy, odwołując się do właściwości i metod obiektu `document`.

- `document.URL` — zwraca adres URL dokumentu jako ciąg tekstu,
- `document.title` — zwraca tytuł strony zdefiniowany w znaczniku `<title>`,
- `document.lastModified` — zwraca datę ostatniej modyfikacji strony,
- `document.bgColor` — określa kolor tła dokumentu ustawianego atrybutem `bgcolor` znacznika `<body>`,
- `document.fgColor` — określa kolor pierwszego planu dokumentu ustawianego atrybutem `text` znacznika `<body>`,
- `document.linkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `link`,
- `document.alinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `alink`,
- `document.vlinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `vlink`,
- `document.cookie` — ustawia lub odczytuje `cookie` dla dokumentu.

Do odwołania się do elementu strony służą metody `getElementById()` oraz `getElementByTagName()`. Metoda `getElementById()` używana jest, gdy element, do którego się odwołujemy, posiada atrybut `id`, natomiast metodę `getElementByTagName()` wykorzystujemy do pobrania kolekcji zawierającej elementy danego typu.

Przykład 3.61

```
<p id="tekst1">Skrypty języka JavaScript</p>
<p>Dokument HTML</p>
<h2>Model dokumentu DOM</h2>
```

Odwołanie w skrypcie do tak zdefiniowanych elementów może mieć postać:

- `document.getElementById("tekst1")` — odwołanie do akapitu z atrybutem `id="tekst1"`,
- `document.getElementByTagName(p)` — odwołanie do kolekcji akapitów,
- `document.getElementByTagName(p)[0]` — odwołanie do pierwszego akapitu w kolekcji.

Używając metody `getElementById()`, należy pamiętać, że jest to metoda obiektu `document`, dlatego dostęp do niej jest możliwy tylko za pomocą tego obiektu. Odwołanie do elementu strony będzie możliwe tylko wtedy, gdy temu elementowi zostanie nadany atrybut `id`.

Przykład 3.62

```
<input type="button" id="klik" value="Kliknij!"/>
<script type="text/javascript">

var b = document.getElementById("klik");
alert(b.value);

</script>
```

Możliwość odwołania się do elementu strony jest wykorzystywana do zmiany jej zawartości. Zawartość elementu strony można odczytać i zmienić, używając właściwości `innerHTML`. Właściwość tę posiada każdy element strony internetowej. Określa ona wartość przypisaną elementowi. Właściwość `innerHTML` może być użyta tylko razem z metodą `getElementById()` i tylko dla elementów, dla których został zdefiniowany identyfikator (`id`).

Przykład 3.63

```
<html>
<head>
<title>Tekst</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h2>Zmień tekst</h2>
<script type='text/javascript'>

function zmien_Tekst()
{
  document.getElementById('blok').innerHTML = 'Nie ma czasu!';
}
</script>

<p>Czas to pieniądz. <b id='blok'>Nie ma pieniędzy!</b></p>
<input type="button" onclick="zmien_Tekst()" value = 'Zmień tekst'/>

</body>
</html>
```

Po kliknięciu przycisku efektem wykonania kodu będzie zmiana wyświetlanego tekstu (rysunek 3.7).

Zmień tekst

Czas to pieniądz. **Nie ma pieniędzy!**

Zmień tekst

Rysunek 3.7. Możliwość zmiany tekstu wyświetlanego na stronie

Przykład 3.64

W przykładzie 3.49 zostało zdefiniowane wyświetlanie daty i czasu. Zmiana czasu następowała po odświeżeniu strony. Zmodyfikujemy powstały kod tak, aby czas wyświetlany na stronie był zawsze aktualny.

```
<html>
<head>
<title> Mój zegar</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h2>Uciekający czas</h2>
```

```
<div id='zegar' style='width: 100px; border: solid 2px #1d330a;'></div>

<script type='text/javascript'>

var timerID = null;
var timerRunning = false;

function stopclock()
{
    if(timerRunning)
        clearTimeout(timerID)
    timerRunning = false;
}

function startclock()
{
    stopclock();
    showtime();
}

function showtime()
{
  var data_n = new Date();
  var godz = data_n.getHours();
  var min = data_n.getMinutes();
  var sek = data_n.getSeconds();

  var czas = "" + ( godz);
  czas  += ((min < 10) ? ":0" : ":") + min;
  czas  += ((sek < 10) ? ":0" : ":") + sek;

  document.getElementById('zegar').innerHTML = czas;
  timerID = setTimeout("showtime()",1000);
  timerRunning = true;
}
```

```
</script>

<script>startclock();</script>

</script>
</body>
</html>
```

Metoda `getElementById()` odwołuje się do wcześniej zdefiniowanego identyfikatora `'zegar'`. W przykładzie za pomocą właściwości `innerHTML` identyfikatorowi `'zegar'` przypisana została wartość zmiennej `'czas'`, która zawiera bieżący czas. Użyty operator warunkowy *((min < 10) ? ":0" : ":") + min;* ma trzy argumenty: *test* po którym występuje znak `?` oraz *wyrażenie1* i *wyrażenie2* oddzielone znakiem `:`. Jeżeli wartość testu jest *true* to wynikiem jest *wyrażenie1*, w przeciwnym razie wynikiem jest *wyrażenie2*. Wynik interpretacji kodu został pokazany na rysunku 3.8.

Uciekający czas

21:37:01

Rysunek 3.8. Wyświetlanie na stronie aktualnego czasu

Metodą obiektu `document` jest również metoda `document.write()`, która wyświetla na stronie internetowej w oknie dokumentu podany tekst.

Zadanie 3.10

Wykorzystując arkusze CSS oraz skrypty z poprzednich przykładów, zdefiniuj kod HTML, który na stronie internetowej wyświetli bieżący czas w postaci zegara cyfrowego.

Zadanie 3.11

Utwórz i dodaj do dokumentu HTML skrypt, który wyświetli datę ostatniej modyfikacji tego dokumentu.

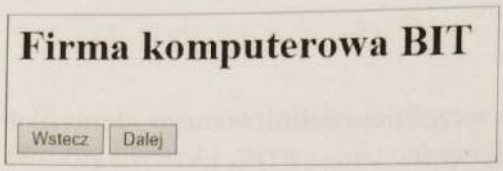
Obiekt history

Drugim obiektem potomnym w stosunku do obiektu `window` jest obiekt `history`. Zawiera on informację o odwiedzanych adresach URL. Posiada zdefiniowane metody, które pozwalają na przejście do wcześniej odwiedzanych stron.

- `history.go()` — otwiera określony adres URL z listy historii. W nawiasach należy podać liczbę dodatnią lub ujemną, określającą, o ile do przodu lub do tyłu należy przemieścić się, aby otworzyć określony adres, np. `history.go(3)`.
- `history.back()` — otwiera poprzedni adres URL z listy historii.
- `history.forward()` — otwiera następny adres URL z listy historii, jeżeli taki istnieje.

Obiekt `history` posiada jedną właściwość, `history.length`, która zawiera informację o długości listy historii.

Wykorzystując metody `back()` i `forward()`, można utworzyć skrypty, które wyświetlą na stronie przyciski *Wstecz* oraz *Dalej*, umożliwiające poruszanie się w przeglądarce po odwiedzanych stronach (rysunek 3.9).



Rysunek 3.9. Wyświetlenie za pomocą metody back() i forward() przycisków Wstecz i Dalej

Przykład 3.65

```
<html>
<head>
<title>Przyciski</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>Firma komputerowa BIT</h1><br>
<input type="button" onclick="history.back()" value="Wstecz">
<input type="button" onclick="history.forward()" value="Dalej">
</body>
</html>
```

Zadanie 3.12

Dla dokumentu HTML utwórz skrypt, dzięki któremu po kliknięciu przycisku *Data modyfikacji* będzie możliwe wyświetlenie daty modyfikacji bieżącej strony internetowej.

Obiekt location

Trzecim obiektem potomnym w stosunku do obiektu `window` jest obiekt `location`. Zawiera on informację o bieżącym adresie dokumentu otwartego w oknie. Za pomocą właściwości tego obiektu można uzyskać pełną informację o adresie URL, można też uzyskać dostęp do jego fragmentów.

- `location.href` — zawiera cały adres URL,
- `location.protocol` — zawiera protokół,
- `location.hostname` — zawiera nazwę hosta,
- `location.port` — zawiera numer portu,

- `location.pathname` — zawiera nazwę pliku ze ścieżką,
- `location.search` — zawiera zapytanie, jeżeli znajduje się ono w adresie,
- `location.hash` — zawiera nazwę kotwicy, jeżeli kotwica występuje w adresie.

Przykład 3.66

```
window.location.href = "http://www.helion.pl"
```

UWAGA

Właściwość `location.href` zawiera ten sam adres co właściwość `document.URL`. Jednak właściwości `document.URL` nie można modyfikować. W celu otwarcia nowej strony należy posługiwać się właściwością `location.href`.

Obiekt `location` posiada dwie metody:

- `location.reload()` — odświeża (ponownie wczytuje) bieżący dokument. Jeżeli dodany zostanie parametr `true`, odświeżanie odbędzie się niezależnie od tego, czy dokument uległ zmianie, czy nie.
- `location.replace()` — zastępuje bieżący adres URL nowym.

Zadanie 3.13

Dla dokumentu HTML utwórz skrypt, który wyświetli nazwę pliku oraz ścieżkę dostępu do bieżącej strony internetowej.

Obiekt link

Obiektem potomnym w stosunku do obiektu `document` jest obiekt `link`. Zawiera on informację o łączu do określonego adresu. Obiekty `link` są zapisane w tablicy `links`. W dokumencie może wystąpić wiele obiektów `link`. Każdy z nich jest zapisany jako oddzielny element tablicy.

Właściwość tablicy `document.links.length` określa liczbę linków na stronie.

Każdy obiekt `link` zapisany w tablicy ma listę właściwości określających adres URL. Są to właściwości takie same jak dla obiektu `location`. Można się do nich odwoływać, podając numer w tablicy i nazwę właściwości.

Przykład 3.67

```
link1 = links[0].href;
```

Obiekt anchor

Kolejnym obiektem potomnym w stosunku do obiektu `document` jest obiekt `anchor`. Reprezentuje on kotwicę w bieżącym dokumencie.

UWAGA

Kotwica określa zdefiniowaną lokalizację w dokumencie HTML, do której można się przenieść.

Podobnie jak łącza, kotwice są zapisywane w tablicy o nazwie `anchors`. Każdy je element jest obiektem `anchor`.

Właściwość tablicy `document.anchors.length` określa liczbę elementów kotwic na stronie.

Obiekt form

Obiektem potomnym w stosunku do obiektu `document` jest również obiekt `form` Zawiera on informacje dotyczące formularzy występujących w dokumencie HTML Obiekty form są zapisane w tablicy `forms`. Ponieważ w dokumencie może wystąpić wiele formularzy, każdy z nich jest zapisany jako oddzielny element tablicy. Do wybranego formularza można odwoływać się przez indeks lub przez nazwę, wpisując w kodzie polecenie `document.forms[0]` lub `document.forms['Form1']`. Lepszą metodą odwołania do formularza jest wykorzystanie metody `getElementById()`, np. `document.getElementById('form1')`.

Przykład 3.68

```
<body>
<form id="form1" name="form1" action="mailto:nauka@gmail.com" method="post">
...
</form>
<script type="text/javascript">
document.forms['form1']
...
</script>
</body>
```

Jeżeli został zastosowany atrybut `name` (jak w przykładzie 3.69), to do formularza można odwołać się również w ten sposób: `document.form1`.

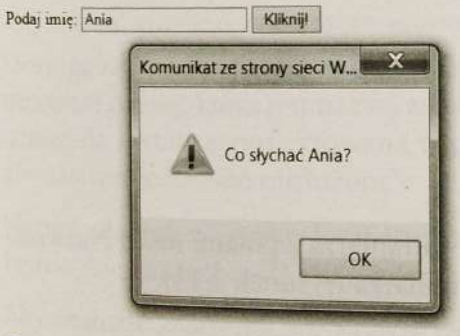
Każdy element formularza jest obiektem, więc posiada właściwości. Jedną z nich jest właściwość `value`, która przechowuje bieżącą wartość elementu.

Przykład 3.69

```
<html>
<head>
<title>Co słychać</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form id="form1" name="form1" action="mailto:nauka@gmail.com" method="post">
Podaj imię: <input type="text" name="imie"/>
<button onclick="Witaj()"> Kliknij!</button>
</form>
<script type="text/javascript">
function Witaj()
{
var imie = document.forms['form1'].imie.value;
alert('Co słychać ' + imie + '?');
}
</script>
</body>
</html>
```

Podany kod definiuje formularz z polem *Imie* i przyciskiem *Kliknij*. Gdy dla przycisku wystąpi zdarzenie `onclick` (kliknięcie przycisku), zostanie uruchomiona funkcja `Witaj()`, która pobierze wartość elementu `imie` (`var imie = document.forms['form1'].imie.value;`) i wyświetli ją w oknie z komunikatem (rysunek 3.10).



Rysunek 3.10. Pobrane z formularza imię zostało wyświetlone w oknie z komunikatem

Elementy formularza tworzą tablicę. Dostęp do nich jest możliwy przez odwołanie się do kolejnych elementów tej tablicy (`elements[i]`). Podobnie jak do całego formularza, do jego elementów można odwoływać się przez indeks lub przez nazwę:

```
document.forms['form1'].elements[0]
document.forms['form1'].elements['nazwa']
document.forms['form1'].nazwa
```

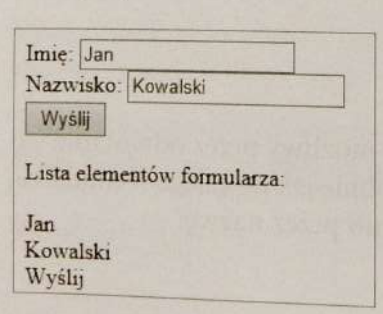
lub za pomocą metody `getElementById()`:

```
document.getElementById('Imie').value
```

Przykład 3.70

```
<html>
<head>
<title>Lista formularza</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form id="form1" action="mailto:nauka@gmail.com" method="post">
Imię: <input type="text" name="imie" value="Jan"/><br />
Nazwisko: <input type="text" name="nazwisko" value="Kowalski" /><br />
<input type="submit" value="Wyślij" />
</form>
<p>Lista elementów formularza:</p>
<script type="text/javascript">
var x=document.getElementById("form1");
for (var i=0;i<x.length;i++){
document.write(x.elements[i].value);
document.write("<br />");
}
</script>
</body>
</html>
```

Wynikiem wykonania kodu będzie wyświetlenie formularza z polami *Imię* i *Nazwisko* oraz przycisku *Wyślij* wraz z listą elementów formularza (rysunek 3.11).



Rysunek 3.11. Wyświetlenie na stronie formularza wraz z listą jego elementów

3.9. Obsługa zdarzeń

Tworzone w języku JavaScript skrypty mogą służyć do obsługi zdarzeń (ang. *event handler*). Są to tak zwane procedury zdarzeniowe. Skrypty takie definiują zachowanie się przeglądarki w przypadku wystąpienia określonego zdarzenia. Większość zdarzeń wywoływana jest przez działania użytkownika (np. kliknięcie przyciskiem myszy). Gdy wystąpi takie zdarzenie, przeglądarka przechodzi do wykonania skryptu związanego z zaistniałym zdarzeniem. Występują również zdarzenia, które nie są inicjowane przez użytkownika (np. zakończenie wczytywania strony).

Przykłady zdarzeń to:

- kliknięcie przez użytkownika myszą,
- przesunięcie myszy nad element,
- wczytanie strony internetowej,
- wczytanie obrazu,
- zmiana zawartości pola wprowadzania,
- naciśnięcie klawisza.

Każdy skrypt opisujący zdarzenie skojarzony jest z określonym obiektem strony internetowej. W kodzie HTML w znaczniku opisującym obiekt musi znaleźć się specyfikacja tego skryptu.

Np. zdarzenie `onMouseOver` zachodzi wówczas, gdy wskaźnik myszy pojawi się nad obiektem (np. przyciskiem). Zdarzenie to może w kodzie HTML zostać opisane w następujący sposób:

```
<img src="przycisk.gif" onMouseOver="highlight();">
```

Obsługa zdarzenia została opisana jako wartość atrybutu (`highlight`), a sam atrybut otrzymał nazwę, która jest nazwą zdarzenia (`onMouseOver`). W tym wypadku obsługa zdarzenia została zaprojektowana w postaci funkcji, czyli wystarczy wywołać funkcję, aby zdarzenie zostało obsłużone.

Nazwy zdarzeń w języku JavaScript zwykle zaczynają się od słowa „on". Kliknięcie będzie mieć nazwę `onClick`, najechanie myszą `onMouseOver` itd.

Aby obsłużyć zdarzenie, nie zawsze trzeba tworzyć funkcję. Jeżeli kod, który ma zostać wywołany, jest pojedynczą instrukcją, można wpisać skrypt bezpośrednio w znaczniku.

Przykład 3.71

```
<html>
<head>
<title>Okno ALERT</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
```

```
<body>
<input type="button" value="Pokaż" onClick="alert('Witaj!');"/>
</body>
</html>
```

Po kliknięciu przycisku wyświetli się okno z komunikatem `'Witaj!'`.

Ogólna postać przypisania zdarzenia:

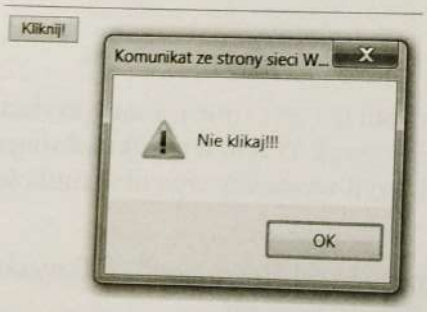
```
element1.zdarzenie = funkcja
```

Przykład 3.72

```
<input type="button" id="klik" value="Kliknij!"/>

<script type="text/javascript">
function pisz()
{
alert('Nie klikaj!!! ');
}
document.getElementById("klik").onClick = pisz
</script>
```

Wynik interpretacji kodu został pokazany na rysunku 3.12.



Rysunek 3.12. Wynik obsługi zdarzenia onClick po kliknięciu przycisku Kliknij!

3.9.1. Zdarzenia myszy

Mysz jest podstawowym narzędziem wykorzystywanym do poruszania się po stronach internetowych, dlatego obsługa zdarzeń związanych z myszą jest jedną z najczęściej stosowanych funkcji.

Do zdarzeń wywołanych działaniem myszy należą:

- `onClick` — występuje po kliknięciu myszą,
- `onDblClick` — występuje po dwukrotnym kliknięciu myszą,

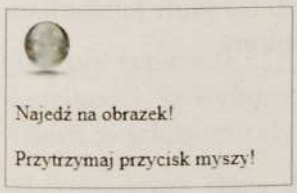
- onMouseDown — występuje, gdy przycisk myszy został wciśnięty,
- onMouseUp — występuje, gdy przycisk myszy został zwolniony,
- onMouseOver — występuje, gdy kursor myszy został umieszczony na elemencie,
- onMouseMove — występuje, gdy kursor myszy został przesunięty wewnątrz elementu,
- onMouseOut — występuje, gdy kursor myszy został przesunięty poza element.

Przykład 3.73

```
<html>
<head>
<title>Zdarzenie myszy</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<a href="http://www.helion.pl" onMouseOver='zmiana.src="Ikona3.png"'
onmouseout='zmiana.src="Ikona1.png"' onMouseDown='zmiana.src="Ikona4.png"'>
<img src="Ikona1.png" name="zmiana" border="0" width="50" height="50" ></a>
<p>Najedź na obrazek!</p>
<p>Przytrzymaj przycisk myszy!</p>

</body>
</html>
```

W przykładzie zostały wykorzystane zdarzenia myszy onMouseOver, onMouseOut i onMouseDown. W efekcie po ustawieniu myszy na pierwszym rysunku ulegnie on zmianie. Wciśnięcie przycisku myszy spowoduje kolejną zmianę rysunku, a przesunięcie myszy poza rysunek przywróci ponowne pokazanie pierwszego rysunku. Kliknięcie rysunku spowoduje przeniesienie do strony *helion.pl* (rysunek 3.13).



Rysunek 3.13. Obsługa zdarzeń myszy

Przykład 3.74

```
<html>
<head>
```

```
<script>
function Ik2()
{
document.getElementById('ikona').src="Ikona2.png";
}
function Ik1()
{
document.getElementById('ikona').src="Ikona1.png";
}
</script>
</head>

<body>
<img id="ikona" onMouseDown="Ik2()" onMouseUp="Ik1()" src="Ikona1.png"
width="100" height="100" />
<p>Kliknij i przytrzymaj!</p>
</body>
</html>
```

W przykładzie zostały wykorzystane zdarzenia myszy `onMouseDown` i `onMouseUp`. W celu odwołania się do elementu `id="ikona"` została wykorzystana metoda `getElementById()`. Po wciśnięciu przycisku myszy nastąpi zmiana grafiki. Po jego zwolnieniu nastąpi powrót do poprzedniej grafiki.

3.9.2. Zdarzenia klawiatury

Zdarzenia związane z klawiaturą są rzadko stosowane. Wykorzystują je tylko aplikacje typu edytor tekstu lub czytnik poczty. Są to trzy zdarzenia:

- `onKeyPress` — klawisz został naciśnięty i zwolniony,
- `onKeyDown` — klawisz został wciśnięty, ale nie został zwolniony,
- `onKeyUp` — klawisz został zwolniony.

3.9.3. Zdarzenia formularza

Z formularzem związane są dwa zdarzenia:

- `onSubmit` — zdarzenie jest generowane, gdy użytkownik wysyła formularz (po kliknięciu przycisku *Submit*). Zdarzenie to może zostać wykorzystane do sprawdzenia, czy formularz został poprawnie wypełniony.
- `onReset` — zdarzenie jest generowane, gdy formularz jest czyszczony z zawartości (po kliknięciu przycisku *Reset*).

3.9.4. Zdarzenia elementów formularza

Oprócz zdarzeń generowanych przez formularz mogą wystąpić zdarzenia generowane przez elementy formularza. Są to:

- `onFocus` — element formularza otrzymał zaznaczenie, czyli wpisywane dane będą trafiały do tego elementu,
- `onBlur` — element stracił zaznaczenie,
- `onSelect` — element został wybrany (wybrany element nie zawsze otrzymuje zaznaczenie),
- `onChange` — zawartość elementu formularza uległa zmianie.

3.9.5. Zdarzenia dokumentu

Do zdarzeń związanych z dokumentem należą:

- `onLoad` — strona została załadowana,
- `onUnload` — strona jest zamykana.

3.10. Wykorzystanie skryptów na stronie internetowej

3.10.1. Kolejność wykonywania skryptów

W kodzie HTML może być umieszczonych wiele różnych skryptów. Mogą to być skrypty zapisane między znacznikami `<script>` i `</script>`, skrypty zapisane w zewnętrznych plikach lub procedury zdarzeniowe. Skrypty będą wykonywane w ściśle określonej kolejności.

Skrypty `<script>`, znajdujące się w sekcji `<head>`, niezależnie od tego, czy są wbudowane w kod, czy też są importowane z zewnętrznych plików, wykonywane są w pierwszej kolejności. Skrypty te nie mogą tworzyć zawartości wyświetlanej na stronie, dlatego najczęściej służą do definiowania funkcji wykorzystywanych w innych skryptach.

Skrypty typu `<script>` znajdujące się w sekcji `<body>` wykonywane są w drugiej kolejności. Są wykonywane podczas wczytywania i wyświetlania strony, w kolejności takiej, w jakiej zostały zdefiniowane w sekcji `<body>`.

Skrypty, które służą do obsługi zdarzeń (procedury zdarzeniowe), są uruchamiane w chwili wystąpienia zdarzenia, na przykład skrypt obsługujący zdarzenie `onLoad` wykonywany jest w chwili rozpoczęcia wczytywania strony. Ponieważ sekcja `<head>` jest wczytywana przed zaistnieniem jakiegokolwiek zdarzenia, funkcje, które zostały zdefiniowane w skryptach znajdujących się w tej sekcji, mogą być używane w procedurach zdarzeniowych.

3.10.2. Animowanie tekstu

Jednym z ciekawszych zastosowań języka JavaScript jest możliwość tworzenia na stronie internetowej animowanej grafiki. Mogą to być obraz zmieniający się w chwili najechania na niego myszą, zmieniające się automatycznie obrazki, animowany baner, pokaz slajdów czy galeria zdjęć.

Baner (przesuwany tekst)

Animowanie tekstu w banerze może polegać na przemieszczaniu tekstu z początkowego położenia z prawej strony okna w kierunku jego lewej krawędzi i na powtórnym wykonaniu animacji po dotarciu tekstu do lewej krawędzi.

Przykład 3.75

```
<html>
<head>
<title>Przepływajacy tekst</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
#napis
{
position: absolute;
background-color: #3ff;
width: 210px;
float: left;
}
</style>

<script type="text/javascript">

function Animacja()
{
var blok = document.getElementById('napis');
if (parseInt(blok.style.left) < 50)
{
  blok.style.left = "250px";
}
else
{
```

```
    blok.style.left = (parseInt(blok.style.left) - 3)+"px";
  }
}

window.setInterval("Animacja()", 100);
</script>

</head>
<body>
<div id="napis" style="left: 300px;">JavaScript - język skryptowy :-)</div>

</body>
</html>
```

W podanym przykładzie w sekcji `<body>` został utworzony blok `div` ze zdefiniowanym identyfikatorem `"napis"`, który będzie animowany. Początkowe położenie bloku `<div>` zostało zdefiniowane z wykorzystaniem parametru `style="left: 300px;"`. Zadaniem skryptu jest przesuwanie bloku o 3 piksele co 0,1 sekundy. Funkcja `Animacja()` sprawdza wartość właściwości `style.left` elementu `id='napis'` i jeżeli wynik jest mniejszy niż `50px`, element jest przenoszony do początkowej pozycji (`blok.style.left = "250px"`). Jeżeli wynik jest większy niż `50px`, to kontynuowane jest odejmowanie od aktualnej wartości `3px`. Metoda `window.setInterval()` wywołuje funkcję `Animacja()` co 0,1 sekundy.

Zadanie 3.14

Wykorzystując podany w przykładzie 3.76 kod oraz język HTML i arkusze stylów, utwórz przesuwający się na stronie internetowej tekst. Dobierz odpowiednio do strony parametry tekstu (kolor, wielkość liter, tło itp.).

Baner (tekst pływający)

Innym sposobem animowania tekstów w banerach reklamowych jest wyświetlanie tekstu w ciągłej pętli.

Przykład 3.76

```
<html>
<head>
<title>Baner reklamowy</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript">
```

```
czas = 100
znak_p = 1
function przewin() {
window.setTimeout('przewin()',czas);

var nap = document.napis.text.value;
document.napis.text.value = nap.substring(znak_p) + nap.substring(0,znak_p);
}
przewin()
//-->
</SCRIPT>
</head>
<body>

<FORM NAME="napis">
<INPUT NAME="text" SIZE=50 VALUE="   Najlepsze strony internetowe   Zostań
z nami  ">
</FORM>

</body>
</html>
```

W podanym przykładzie została wykorzystana metoda `window.setTimeout()`, która wywołuje funkcję po określonej liczbie milisekund. Metoda ta posiada dwa argumenty. Pierwszy z nich to nazwa uruchamianej funkcji, która musi być zapisana w cudzysłowach, drugi to parametr, określający, po jakim czasie powinna zostać uruchomiona funkcja.

Metoda `substring()` zwraca fragment tekstu z łańcucha. Jej parametry określają położenie początku i końca tego tekstu. Pierwszy parametr musi wystąpić, drugi występuje opcjonalnie. Jeżeli ten ostatni nie zostanie podany, koniec fragmentu tekstu jest równoznaczny z końcem łańcucha.

3.10.3. Animowanie grafiki

Zmiana grafiki

Prostym sposobem animowania grafiki jest zmiana obrazka po ustawieniu na nim kursora myszy. Do utworzenia skryptu potrzebne będą dwa pliki z grafiką. Zawartość pierwszego będzie wyświetlana jako element strony. Drugi element graficzny będzie się pojawiał, gdy użytkownik wskaże myszą pierwszy element.

Przykład 3.77

```
<html>
<head>
<title>Zmiana ikony</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p>
<img src="Ikona1.png" alt="" id='ikona' />
</p>

<script type="text/javascript">
   var ikona_n = new Image();
    ikona_n.src = 'Ikona2.png';

   var ikona_p = new Image();
    ikona_p.src = 'Ikona1.png';

    var obraz = document.getElementById('ikona');

    obraz.onMouseOver = function() {
        this.src = ikona_n.src
    }
    obraz.onMouseOut = function() {
        this.src = ikona_p.src
    }
</script>

</body>
</html>
```

W podanym przykładzie zostały zdefiniowane dwa nowe obiekty typu `Image`: `ikona_n` oraz `ikona_p`. Przechowują one odnośniki do zmienianych obrazków (zapisanych w plikach *Ikona1.png* oraz *Ikona2.png*). W wyniku wykonania skryptu po najechaniu kursorem myszy na pierwszy obrazek nastąpi zmiana atrybutu `src` na drugi obrazek. Po odsunięciu kursora myszy atrybut `src` ponownie przyjmie wartość pierwszego obrazka.

W wyniku wykonania polecenia `var obraz = document.getElementById('ikona');` został utworzony obiekt `obraz` zawierający odnośnik do elementu o `id='ikona'`. Kolejnym działaniem jest obsługa zdarzenia `onMouseOver` dla elementu `obraz`. Po wystąpieniu tego zdarzenia nastąpi zmiana atrybutu `src` obiektu (`this.src = ikona_n.src`). Po usunięciu kursora znad obrazka nastąpi ponowna zmiana atrybutu `src` (`this.src = ikona_p.src`). Słowo kluczowe `this` wskazuje na obiekt, który wywołał funkcję.

Zadanie 3.15

Rozbuduj przykład 3.78 tak, aby po kliknięciu obrazu został wyświetlony trzeci element graficzny.

Animowany baner (slider)

Innym rodzajem animowania grafiki jest animowany baner ze zmieniającymi się automatycznie obrazkami. Ten rodzaj animacji najczęściej realizowany jest z użyciem animowanych GIF-ów. Ale podobny efekt osiągniemy, stosując język JavaScript.

W takim skrypcie dobrym rozwiązaniem jest zastosowanie tablicy do przechowywania nazw obrazów. Wielkość tablicy może być zmieniana w zależności od liczby wyświetlanych obrazów. Pojawianie się kolejnych elementów graficznych jest obsługiwane przy użyciu licznika czasu.

Przykład 3.78

W przykładzie wykorzystano sześć zdjęć o takich samych rozmiarach.

```
<html>
<head>
<title>Animowany baner</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
#baner {
width: 300px;
height: 180px;
}
</style>

<script type="text/javascript">
window.onLoad=Zmiana;
var Nr=0;
function Zmiana()
```

```
{
 var obrazy = new Array('z1.jpg', 'z2.jpg', 'z3.jpg', 'z4.jpg', 'z5.jpg',
'z6.jpg'); // dodawanie obrazków do tablicy
 Nr++;
   if (Nr == obrazy.length)
   {
   Nr = 0;
   }
 document.getElementById('baner').src = obrazy[Nr];
 setTimeout(Zmiana,2000);
}
</script>

</head>
<body>
<div >
<img src="z1.jpg" id="baner" alt="Reklama">
</div>
</body>
</html>
```

W podanym przykładzie za pomocą stylów został zdefiniowany rozmiar wyświetlanych obrazów. Pierwsze polecenie skryptu `window.onLoad=zmiana;` obsługuje zdarzenie związane z zakończeniem wczytywania strony i spowoduje wywołanie funkcji `zmiana()`.

UWAGA

Próba odwołania się do elementów, które nie zostały jeszcze wczytane, mogłaby spowodować wystąpienie błędów i nieprawidłowe działanie skryptu.

Zmienna `Nr` to licznik wyświetlanych obrazów. Jej początkowa wartość została ustawiona na wartość `0`. Funkcja `Zmiana()` zawiera definicję tablicy `obrazy`, która przechowuje nazwy plików z wyświetlanymi obrazami. Kod `if (Nr == obrazy.length)` sprawdza, czy licznik obrazów (`Nr`) jest równy liczbie elementów tablicy `obrazy`. Jeżeli tak, to licznik jest zerowany.

Obrazowi wyświetlanemu na stronie został przypisany identyfikator `id='baner'`. Kod `document.getElementById('baner').src = obrazy[Nr]` pobiera adres obrazu

z tablicy `obrazy` z pozycji określonej zmienną `Nr`. Natomiast metoda `setTimeout()` wywołuje funkcję `Zmiana()` co określoną liczbę milisekund.

Zadanie 3.16

Zmodyfikuj kod podany w przykładzie 3.79, tak aby baner był łączem, które po kliknięciu wyświetlanego zdjęcia otworzy podstronę witryny związaną ze zdjęciem.

3.11. Walidacja formularzy

Jednym ze sposobów kontrolowania danych z formularzy jest stosowanie skryptów działających po stronie serwera (PHP, CGI). Innym sposobem jest kontrolowanie formularzy za pomocą skryptów działających po stronie przeglądarki.

Sprawdzanie poprawności wypełnienia formularza po stronie klienta ma kilka zalet. Po pierwsze, informacja o błędzie w wypełnianiu formularza jest zwracana natychmiast, bez konieczności oczekiwania na odpowiedź ze strony serwera. Po drugie, następuje zmniejszenie obciążenia serwera. W praktyce sprawdzanie poprawności wypełnienia formularza za pomocą JavaScript nie zwalnia z konieczności sprawdzania poprawności danych po stronie serwera (walidacja danych po stronie serwera zapewnia bezpieczne działanie całej aplikacji).

Walidacja danych polega na sprawdzeniu, czy wszystkie pola formularza zostały wypełnione oraz czy wprowadzone wartości mają odpowiednią postać.

3.11.1. Sprawdzanie wypełnienia pól formularza

Podstawowym działaniem wykonywanym podczas walidacji formularza jest sprawdzenie, czy pola, które nie powinny pozostać puste, zostały przez użytkownika wypełnione. Jeżeli pozostały puste pola, formularz nie zostanie wysłany.

Przykład 3.79

```
<html>
<head>
<title>Spr_form</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript">
function Sprawdz(form)
{
  if (form.nazw.value =='')
  {
    alert('Pole Nazwisko musi być wypełnione');
    form.nazw.focus();
```

```
    return false;
  }
  if (form.imie.value =='')
  {
    alert('Pole Imię musi być wypełnione');
    form.imie.focus();
    return false;
  }
  if (form.zawod.value =='')
  {
    alert('Pole Zawód musi być wypełnione');
    form.zawod.focus();
    return false;
  }
  return true;
}
</script>

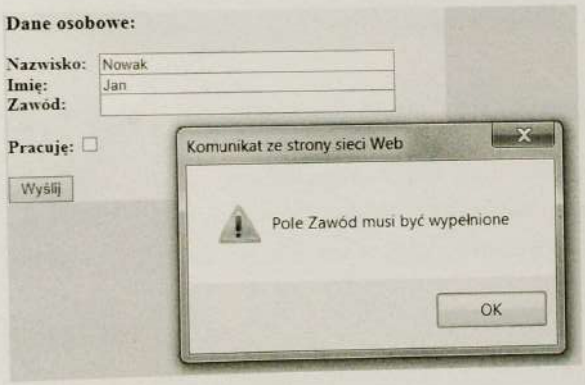
<style type="text/css">
div {
    width: 400px;
    background-color: #dfF;
}
.pole {
    position:absolute;
    left:90px;
}
</style>

</head>
<body bgcolor="#dfdfdf">
<div>
<form enctype="text/plain" action="mailto:nauka@gmail.com" method="post"
onsubmit='return Sprawdz(this);'>
```

```
<b><font size="4"><b>Dane osobowe:</font></b><br><br>
<b>Nazwisko:</b>
<input class="pole" name="nazw" value="" size="40" ><br>
<b>Imię:</b>
<input class="pole" name="imie" value="" size="40" ><br>
<b>Zawód:</b>
<input class="pole" name="zawod" value="" size="40" ><br><br>
<label> Pracuję: <input type="checkbox" name="opcje"  maxlength="1">
</label><br><br>

<input type="submit" value="Wyślij"></form>
</div>
</body>
</html>
```

W podanym przykładzie funkcja `Sprawdz()` zwraca wartość `true` lub `false`. Gdy zwróci `false` (pole nie zostało wypełnione), formularz nie będzie wysłany (rysunek 3.14).



Rysunek 3.14. Walidacja formularza. Formularz nie zostanie wysłany, ponieważ pole Zawód nie zostało wypełnione

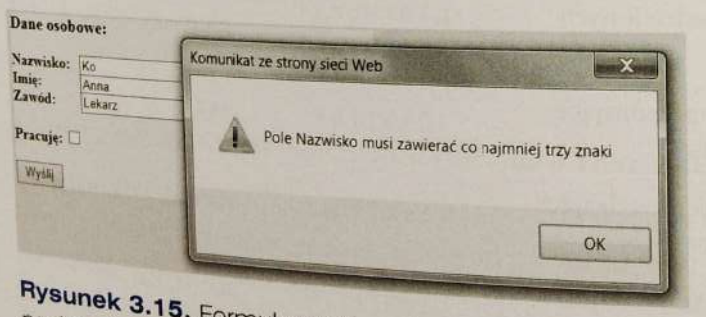
Formularze są wysyłane w formie zakodowanej. Atrybut `(enctype="text/plain")` powoduje, że formularz będzie przysyłany w formie odkodowanej. Tę metodę należy stosować w formularzach pocztowych, co pozwoli na zaoszczędzenie czasu potrzebnego na odszyfrowywanie przysłanych danych. Nie ma ona natomiast większego znaczenia, gdy formularz będzie wykorzystany w skryptach (po stronie klienta lub serwera).

Inny sposób zdefiniowania funkcji `Sprawdz()` z wykorzystaniem metody `getElementById()` został pokazany w przykładzie poniżej (przykład 3.81). Za pomocą funkcji `Sprawdz_d()` można sprawdzić nie tylko to, czy pole zostało wypełnione, ale również długość tekstu wprowadzonego do poszczególnych pól.

ykład 3.80

```
function Sprawdz_d(form)
{
  if (document.getElementById('nazw').value.length <3)
  {
    alert('Pole Nazwisko musi zawierać co najmniej trzy znaki');
    form.nazw.focus();
   return false;
   }
   if (document.getElementById('imie').value.length <2)
   {
     alert('Pole Imię musi zawierać co najmniej dwa znaki');
     form.imie.focus();
    return false;
    }
    if (document.getElementById('zawod').value =='')
    {
      alert('Pole Zawód musi być wypełnione');
      form.zawod.focus();
     return false;
     }
     return true;
   }
```

Wynik zastosowania podanej w przykładzie funkcji został pokazany na rysunku 3.15.



Rysunek 3.15. Formularz nie zostanie wysłany, ponieważ ciąg wpisany w polu Nazwisko jest za krótki

3.11.2. Wyrażenia regularne

Wyrażenia regularne to wzorce opisujące łańcuch symboli. Przy ich użyciu można sprawdzać i modyfikować teksty. Wyrażenia regularne są wykorzystywane do sprawdzania, czy ciągi znaków wprowadzonych do formularza są zgodne z wymaganymi wzorcami.

Język JavaScript ma wbudowane mechanizmy obsługi wyrażeń regularnych. Za ich obsługę odpowiada obiekt `RegExp(wzorzec,flaga)`. Może on być tworzony na dwa sposoby:

```
var nowe_wyr = new RegExp('^[0-9]+[a-z]+$');
var nowe_wyr = /^[0-9]+[a-z]+$/;
```

Podane wyżej przykłady są równoznaczne.

Definiowany `wzorzec` składa się z symboli (znaków specjalnych), które opisują wygląd określonego fragmentu tekstu. Tabela 3.6 zawiera opis wybranych symboli.

Tabela 3.6. Niektóre symbole wzorca

Symbol	Znaczenie	Przykład	Ciągi zgodne z podanym wzorcem
`^`	początek wzorca	`^pa`	pani, pan, parasol
`$`	koniec wzorca	`as$`	las, czas, kompas
`.`	dowolny pojedynczy znak	`.an.a`	banda, fanta, janka
`[...]`	dowolny z wymienionych znaków; w nawiasach można podać kolejne znaki lub wpisać zakres	`[a-z][b-t] naln[ey]`	banalny, finalny, tonalne,
`[^...]`	dowolny z niewymienionych znaków	`kro[^st]`	krowa, kroki, kropy
`|`	jeden z ciągów rozdzielonych znakiem	`pierwszy| 1-szy|1`	pierwszy, 1-szy, 1
`{3}`	dokładnie trzy poprzedzające znaki lub elementy	`[0-9]{3}`	243, 178, 629
`{3,}`	co najmniej trzy poprzedzające znaki lub elementy	`[a-k]{3,}`	abecad, gafa, haha
`{2,5}`	od 2 do 5 poprzedzających znaków lub elementów	`[a-m]{2,4}`	mama, bal, da
`\.`	znak kropki	`[0-9]{3}\. [0-9]{2}`	421.23, 829.45

Jako drugi parametr obiektu `RegExp()` flaga może wystąpić, ale nie musi. Flaga działa na zdefiniowanym wzorcu (tabela 3.7).

Tabela 3.7. Niektóre symbole flagi

Znak flagi	Znaczenie
i	nie jest uwzględniana wielkość liter
g	zwracane są wszystkie pasujące fragmenty

W języku JavaScript dodatkowo zostały zdefiniowane specjalne klasy znaków. Ich symbole mogą być wykorzystane przy definiowaniu wyrażeń regularnych (tabela 3.8)

Tabela 3.8. Klasy znaków

Klasa znaków	Znaczenie
\s	znak spacji, tabulacji lub nowego wiersza
\S	znak nie jest spacją, znakiem tabulacji lub znakiem nowego wiersza
\w	każdy znak, który jest literą, cyfrą i znakiem _
\W	każdy znak, który nie jest literą, cyfrą i znakiem _
\d	każdy znak, który jest cyfrą
\D	każdy znak, który nie jest cyfrą

Wzorzec kodu pocztowego

```
var w_kod = /^[0-9]{2}-[0-9]{3}$/;
```

Wzorce definiowane w języku JavaScript zaczynają się od znaku / i na nim się kończą. Zapis `[0-9]{2}-[0-9]{3}` oznacza, że najpierw powinny wystąpić dwie cyfry. Po nich musi wystąpić znak -, a po nim muszą być trzy cyfry (we wzorcu `[0-9]{3}`).

Inny zapis wzorca dla weryfikacji kodu pocztowego:

```
var w_kod = /^[\d]{2}-[\d]{3}$/;
```

lub

```
var w_kod = new RegExp('^[0-9]{2}[0-9]{3}$');
```

Wzorzec do weryfikacji imienia i nazwiska:

```
var w_nazw = /^[a-zA-Z]{2,}\s+[a-zA-Z]{2,}$/;
```

Znak ^ oznacza, że wzorzec zaczyna się od początku tekstu. Zapis `[a-zA-Z]{2,}` mówi, że ciąg powinien zawierać przynajmniej dwie litery (`imię`). Zapis `\s+` oznacza, że dalej powinny być spacje lub tabulatory (przynajmniej jeden). Kolejny zapis `[a-zA-Z]{2,}` mówi, że następny ciąg to znowu przynajmniej dwie litery (`nazwisko`). Znak `$` oznacza zakończenie wzorca wraz z końcem tekstu.

Inny zapis wzorca dla weryfikacji imienia i nazwiska:

```
var w_nazw = /^[\D]{2,}\s+[\D]{2,}$/;
```

lub:

```
var w_nazw = new RegExp('[a-zA-Z]{2,}\s[a-zA-Z]{2,}');
```

Wzorzec do weryfikacji e-maila

```
var w_mail = /^[0-9a-zA-Z_.-]+@[0-9a-zA-Z.-]+\.[a-zA-Z]{2,3}$/;
```

Znak ^ oznacza, że wzorzec zaczyna się z początkiem tekstu. Zapis `[0-9a-zA-Z_.-]` mówi, że nazwa konta może składać się z dowolnych znaków z zakresu cyfr, liter, znaku podkreślenia _, kropki . i myślnika -. Potem powinien wystąpić znak @. Po tym znaku sprawdzana jest nazwa domeny, która może składać się z dowolnych znaków z zakresu cyfr, liter oraz znaków kropki . i myślnika -. Zapis `\.` oznacza, że kolejnym znakiem musi być kropka, a zapis `[a-zA-Z]{2,3}` mówi, że po kropce musi wystąpić końcowa część nazwy domeny składająca się wyłącznie z liter i jej długość musi wynosić dwa lub trzy znaki. Znak $ oznacza, że wzorzec ma się kończyć wraz z końcem tekstu.

Znając sposoby definiowania wzorców do weryfikacji danych zapisanych w formularzu, można przystąpić do napisania kodu, który będzie weryfikował te dane na stronie internetowej.

Przykład 3.81

```
<html>
<head>
<title>Spr_form (wzorzec)</title>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<script type="text/javascript">
function Spr_wzorzec()
{
    var form = document.getElementById('in_form'),
       wzory = {
      'nazwisko' : /^[a-zA-Z]{2,}\s+[a-zA-Z]{2,}$/i,
      'email' : /^[0-9a-zA-Z_.-]+@[0-9a-zA-Z.-]+\.[a-zA-Z]{2,3}$/i,
        };
  for (var pole in wzory)
  {
    if (form[pole])
    {
      if (!wzory[pole].test(form[pole].value))
      {
        alert('Pole ' + pole + ' ma nieprawidłową wartość');
```

```
        form[pole].style.background = 'yellow';
        return false;
       }
       else
       {
       form[pole].style.background = '';
       }
     }
  }
  alert('Wszystkie pola wypełnione poprawnie!');
  return true;
}
</script>

<style type="text/css">
div {
width: 400px;
background-color: #dfF;
}
.pole {
    position:absolute;
    left:130px;
}
</style>

</head>
<body bgcolor="#dfdfdf">
<div>
<form id="in_form" enctype="text/plain" action="mailto:nauka@gmail.com"
method="post" onsubmit='return Spr_wzorzec();'>

<b><font size="4"><b>Dane osobowe:</font></b><br><br>
<b>Imię i nazwisko:</b>
<input class="pole" name="nazwisko" id="nazwisko" value=""><br>
<b>E-mail:</b>
```

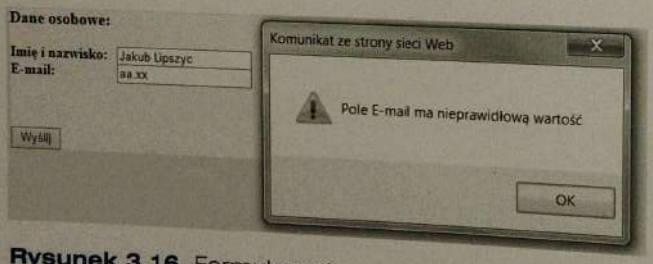
```
<input class="pole" name="email" id="E-mail" value=""><br><br>
<br><br>

<input type="submit" value="Wyślij"></form>
</div>
</body>
</html>
```

W podanym przykładzie formularz składa się z dwóch pól: *Imię i nazwisko* oraz *E-mail*. Po ich wypełnieniu za pomocą funkcji `Spr_wzorzec()` następuje sprawdzenie poprawności wprowadzonych danych.

Funkcja `Spr_wzorzec()` tworzy tablicę asocjacyjną `wzory`. W jej definicji zamiast nawiasów kwadratowych zostały zastosowane nawiasy klamrowe, a zamiast pojedynczych wartości podano pary: klucz i jego wartość (klucz jest oddzielony od wartości dwukropkiem). Kluczami są nazwy pól formularza, a wartościami obiekty wyrażeń regularnych. Dostęp do wszystkich elementów tablicy jest możliwy za pośrednictwem pętli `for (var pole in wzory)`. Jeżeli w formularzu istnieje element odpowiadający elementowi tablicy, to sprawdzane jest, czy jego aktualna wartość jest zgodna z zapisanym w tablicy wyrażeniem regularnym: `if (!wzory[pole].test(form[pole].value))`. Do tego celu została wykorzystana metoda `test()`, która sprawdza, czy wartość w polu formularza (`form[pole].value`) jest zgodna ze zdefiniowanym w tablicy wzorcem (`wzory[pole]`). Jeżeli nie, to wyświetli się komunikat o błędzie w danych i nastąpi powrót do wypełniania formularza. Gdy wszystkie pola zostaną wypełnione prawidłowo, wyświetli się komunikat o poprawności danych, a funkcja przyjmie wartość `true`.

Wynik interpretacji kodu z przykładu 3.83 został pokazany na rysunku 3.16.



Rysunek 3.16. Formularz nie zostanie wysłany, ponieważ pole E-mail jest wypełnione niezgodnie z podanym wzorcem

3.11.3. Zerowanie pól formularza

Czyszczenie wypełnionych pól formularza może być realizowane za pomocą metody `reset()`, która powoduje wyzerowanie wszystkich pól formularza.

Przykład 3.82

```
<html>
<head>
<title>Zerowanie formularza</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script>
function Zeruj()
{
document.getElementById("form1").reset();
}
</script>
</head>
<body>
<p>Wprowadź swoje dane.</p>
<form id="form1">
Imię: <input type="text" name="imie"><br>
Nazwisko: <input type="text" name="nazw"><br><br>
<input type="button" onclick="Zeruj()" value="Wyczyść dane">
</form>
</body>
```

3.12. Testowanie skryptów

Kod tworzonego skryptu jest zwykłym tekstem, który może być dołączony do dokumentu HTML lub być zapisany w osobnym pliku. Do jego tworzenia wystarczy prosty edytor tekstu. Do testowania działania kodu potrzebna jest przeglądarka internetowa — może być dowolna, ale zalecane jest, by była to jej aktualna wersja.

Jeżeli w kodzie skryptu pojawi się błąd, to różne przeglądarki reagują w różny sposób. Najczęściej występuje jedna z sytuacji wymienionych poniżej:

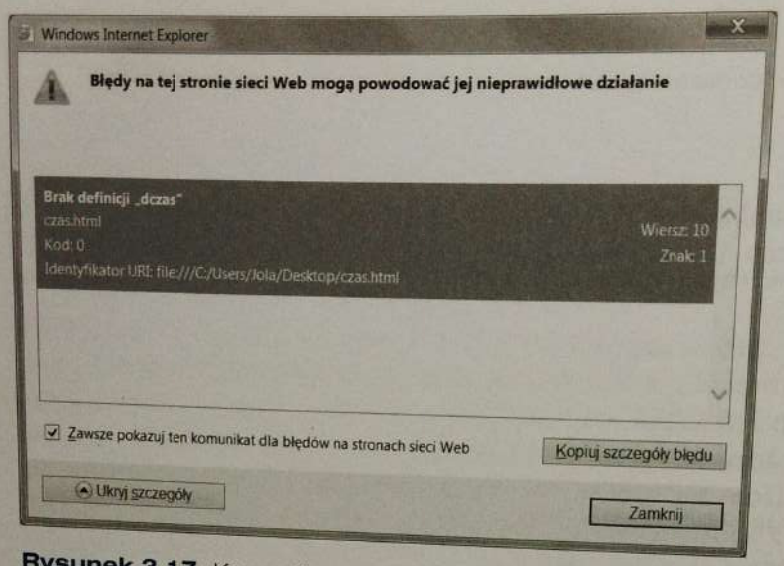
- wyświetli się komunikat o błędzie,
- wykonanie skryptu zostanie przerwane bez żadnej informacji o błędzie.

Jeżeli pojawi się komunikat o błędzie, to można ten błąd poprawić.

Jeżeli taki komunikat się nie pojawi, należy samodzielnie znaleźć przyczynę błędu. Może w tym pomóc odpowiednie ustawienie opcji przeglądarki.

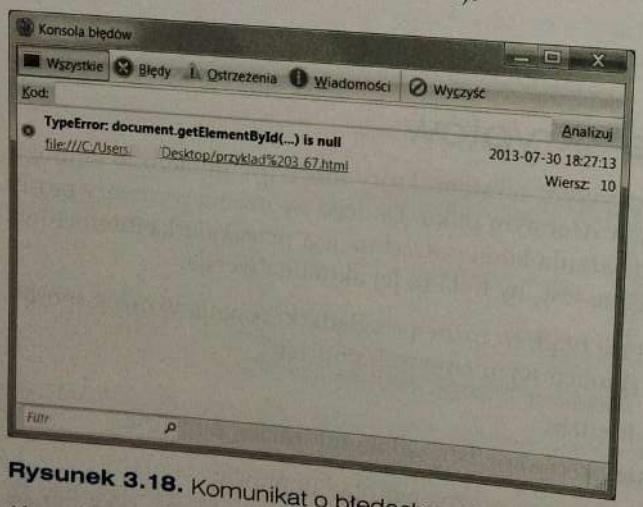
W przeglądarce Internet Explorer trzeba wybrać z menu *Narzędzia/Opcje internetowe*. W otwartym oknie wybrać zakładkę *Zaawansowane* i na liście opcji usunąć zaznaczenie

Wyłącz debugowanie skryptu oraz zaznaczyć *Wyświetl powiadomienie o każdym błędzi skryptu*. Po zatwierdzeniu ustawień podczas testowania skryptów będą wyświetlane komunikaty o występujących błędach (rysunek 3.17).



Rysunek 3.17. Komunikat o błędach w skrypcie — przeglądarka IE

W przeglądarce Firefox lub Netscape należy wybrać z menu *Narzędzia/Dla twórców witryn* i z listy opcję *Konsola błędów*. W otwartym oknie będą wyświetlane komunikaty o występujących błędach (rysunek 3.18).



Rysunek 3.18. Komunikat o błędach w skrypcie — przeglądarka Firefox

Aby wyświetlić konsolę błędów w przeglądarce Opera, należy wybrać w głównym menu *Opera/Strona* zakładkę *Narzędzia deweloperskie* i z listy opcji pozycję *Konsola błędów*. Można również użyć skrótu klawiszowego *Ctrl+Shift+O*.

W przeglądarce Chrome (i pochodnej Chromium) konsola błędów jest zintegrowana z narzędziem zwanym *Inspektorem strony*. Uruchomienie tego dodatku jest bardzo proste — można użyć skrótu klawiszowego *Ctrl+Shift+I* lub kliknąć prawym przyciskiem myszy na stronie, którą chcemy dokładniej przeanalizować, i wybrać *Zbadaj element*. Interesującą nas zakładką jest ta o nazwie *Console*.

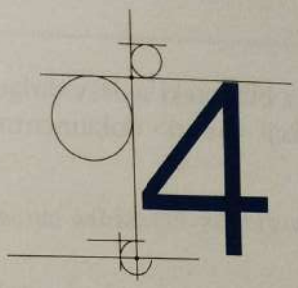
Nie zawsze błąd wykryty przez przeglądarkę jest prawidłowo rozpoznany i zinterpretowany. Również miejsce wystąpienia błędu może być dalekie od faktycznego.

PYTANIA KONTROLNE

1. Gdzie mają zastosowanie języki skryptowe?
2. Które znaczniki pozwalają na wstawienie do dokumentu HTML skryptów napisanych w języku JavaScript?
3. Jak w języku JavaScript deklaruje się zmienne?
4. Wymień podstawowe typy danych dopuszczane w języku JavaScript.
5. W jaki sposób w języku JavaScript definiowane są funkcje?
6. Co oznacza, że zmienna ma zasięg lokalny?
7. Wymień charakterystyczne cechy obiektowości języka JavaScript.
8. Jak działa model obiektowy DOM?
9. Wymień podstawowe obiekty każdej przeglądarki.
10. W jaki sposób obsługiwane są zdarzenia?
11. Na czym polega walidacja formularzy?
12. Do czego wykorzystywane są podczas walidacji danych wyrażenia regularne?

ZADANIE

Utwórz stronę internetową, na której w górnym prawym rogu będą wyświetlane bieżąca data i czas. W nagłówku strony znajdzie się baner reklamowy z tekstem *Witamy na naszej stronie*, który będzie „płynął" od prawej do lewej krawędzi strony. Z lewej strony zostanie utworzone menu pionowe składające się z trzech gotowych ikon (wcześniej przygotowanych w postaci plików graficznych). Najechanie na każdą z ikon kursorem myszy zmieni jej wygląd. Kliknięcie pierwszej ikony spowoduje przeniesienie do strony *www.men.gov.pl*, drugiej — do strony *www.cke.edu.pl*, a trzeciej — do strony *http://helion.pl/*. W swojej pracy wykorzystaj skrypty języka JavaScript.



Biblioteka jQuery

Biblioteki języka JavaScript umożliwiają szybkie i łatwe tworzenie rozszerzeń dla strony internetowej. Do najpopularniejszych należą: jQuery, Prototype, Dojo, Moo Tools, YUI. Są to biblioteki open source. Każda z nich zawiera specyficzny dla siebie zestaw funkcji usprawniających pracę z modelem DOM.

4.1. Opis biblioteki

Biblioteka jQuery to framework zawierający różnorodne funkcje, przy których wykorzystaniu można realizować często spotykane zadania (np. animacje, obsługę zdarzeń, tworzenie efektów wizualnych na stronie internetowej). Jej główną zaletą jest łatwość użycia, niezależność od przeglądarki i małe rozmiary otrzymanego kodu. Biblioteka ułatwia pracę nad stroną internetową i umożliwia tworzenie czytelnego kodu.

4.1.1. Zasady korzystania z biblioteki jQuery

Aby korzystać z biblioteki jQuery, należy pobrać jej plik ze strony *http://www.jquery.com*. Ta metoda jest zalecana w celu programowania i testowania skryptów. Inna metoda to korzystanie z biblioteki za pośrednictwem sieci dostarczania treści. Rozwiązania oparte na udostępnianiu za pośrednictwem sieci dostarczania treści (tzw. hosting) powodują zwiększenie przepustowości danych wysyłanych z witryny. Rozwiązania hostingowe są udostępniane przez firmy o dużych sieciach, takie jak Google, Microsoft. Gdy użytkownik otworzy wybraną stronę na swoim komputerze, biblioteka jQuery zostanie przesłana przez serwer znajdujący się najbliżej niego. Dzięki temu skróci się czas ładowania strony.

Na stronie *http://www.jquery.com* dostępne są dwie wersje biblioteki jQuery. Wersja nieskompresowana przeznaczona jest do programowania i testowania. W celu skrócenia czasu ładowania wersja skompresowana usuwa niepotrzebne komentarze, podziały wierszy, spacje i znaki tabulacji. Zmniejszenie objętości pliku zostało uzyskane kosztem jego czytelności, dlatego ta wersja pliku nie nadaje się do testowania i analizy kodu.

Po wybraniu metody korzystania z biblioteki należy dołączyć ją w kodzie projektowanej strony internetowej. W sekcji <head> dokumentu HTML powinien zostać wstawiony wpis:

```
<script type="text/javascript" src={ścieżka do pliku}/jquery/1.4.2.min.
js></script>
```

Aby zapewnić poprawność przetwarzania strony, arkusze stylów CSS powinny znaleźć się w dokumencie HTML przed biblioteką jQuery. W tym dokumencie powinien zostać zdefiniowany również element doctype. Jego brak może spowodować błędne zachowania przeglądarki lub biblioteka jQuery może zostać niepoprawnie przetworzona.

Sposób dołączania biblioteki jQuery pobranej na lokalny komputer i zapisanej w folderze *js* pod nazwą *jquery.js* został pokazany w przykładzie 4.1.

Przykład 4.1

```
<!DOCTYPE html>
<html>
<head>
<title>Dołączenie biblioteki jQuery</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link href="css/style.css>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
//treść skryptu
</script>
</head>
<body>
...
</body>
</html>
```

Alternatywnie biblioteka jQuery może zostać dołączona u dołu strony przed znacznikiem domykającym </body>.

Sposób dołączania biblioteki jQuery udostępnionej w ramach hostingu przez firmę Google został pokazany w przykładzie 4.2.

Przykład 4.2

```
<link href="css/style.css">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.
js"  type="text/javascript" ></script>
```

```
<script type="text/javascript">
//treść skryptu
</script>
```

Natomiast sposób dołączenia biblioteki jQuery udostępnionej na serwerze jQuery wygląda następująco:

```
<script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
```

Podstawowym sposobem korzystania z biblioteki jQuery jest użycie funkcji `$()`. Wybiera ona określone elementy dokumentu drzewa DOM i pozwala wchodzić w różnego rodzaju interakcje. Zapis `$(document)` to pobranie odnośnika do dokumentu HTML. Druga metoda to użycie zapisu jQuery w postaci: `jQuery(document)`.

Podstawową metodą służącą do obsługi zdarzeń biblioteki jQuery jest metoda `ready()`, która jest wywoływana po załadowaniu drzewa DOM. Jej argumentem jest nazwa funkcji, która zostanie uruchomiona, gdy struktura drzewa DOM otwieranej witryny zostanie załadowana do przeglądarki. Kod w postaci:

```
$(document).ready(function()
{
//tutaj znajdzie się wykonywany kod
});
```

może stanowić szablon tworzonych skryptów. Ten szablon może znaleźć się w dokumencie HTML w treści skryptu, ale lepiej umieścić go w pliku zewnętrznym.

4.1.2. Selektory

Selektory to podstawowe elementy biblioteki jQuery. Wszystkie operacje dotyczące modelu DOM używają selektorów. Przy ich pomocy wybieramy te elementy strony, które będą przetwarzane. Biblioteka jQuery korzysta z selektorów arkuszy CSS i kilku niestandardowych selektorów jQuery. Najczęściej używane selektory CSS zostały przedstawione w tabeli 4.1.

Tabela 4.1. Wybrane selektory CSS

Selektor	Znaczenie
`$('div')`	pobiera wszystkie bloki `div` z dokumentu
`$('.blok')`	pobiera z dokumentu wszystkie elementy, które posiadają klasę `blok`
`$('#id')`	pobiera z dokumentu jeden element o identyfikatorze `id`
`$('p')`	pobiera z dokumentu wszystkie akapity
`$('*')`	pobiera wszystkie elementy modelu DOM lub innego elementu

Ogólna postać użycia selektora wygląda następująco:

```
$('selektor').metoda()
```

Jak widać, selektor składa się z `aliasu` biblioteki jQuery (`jQuery` lub `$`), wybranego elementu modelu DOM zapisanego w cudzysłowach lub apostrofach i ujętego w nawiasy okrągłe oraz metody biblioteki jQuery. Metodą biblioteki może być zarówno definicja stylów, jak i animowanie elementów na stronie.

Prosty przykład pokazuje wykorzystanie biblioteki do ukrywania i pokazywania tekstu po kliknięciu napisu `jQuery`.

Przykład 4.3

Kod HTML:

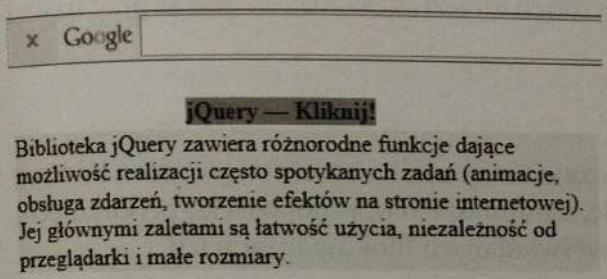
```
<!DOCTYPE HTML> <html>
<head>
<title>JQuery</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moje_skrypty.js"></script>
</head>
<body>
<p id="bibl">jQuery – Kliknij!</p>
<div id="bibl_info" style="display:none;">
Biblioteka jQuery zawiera różnorodne funkcje dające możliwość realizacji
często spotykanych zadań (animacje, obsługa zdarzeń, tworzenie efektów na
stronie internetowej).
Jej głównymi zaletami są łatwość użycia, niezależność od przeglądarki i małe
rozmiary.
 </div>
</body>
</html>
```

Plik ze skryptem *moje_skrypty.js*:

```
$(document).ready(function()
{
  $('#bibl').click(function(){
    $('#bibl_info').toggle();
  });
});
```

W podanym wyżej przykładzie pobrana ze strony biblioteka została zapisana w folderze *js*. W pliku *moje_skrypty.js* został zapisany skrypt utworzony zgodnie z podanym wcześniej szablonem. Po załadowaniu drzewa DOM zostanie wywołana funkcja anonimowa wykonująca polecenia na elementach dokumentu HTML. W wyniku wykonania kodu po kliknięciu napisu *jQuery* wyświetli się opis wybranego hasła, a po ponownym kliknięciu napisu opis zostanie ukryty (rysunek 4.1).



Rysunek 4.1. Po kliknięciu napisu jQuery – Kliknij! został wyświetlony opis tego hasła

W kodzie HTML zostały zdefiniowane identyfikatory dla akapitu `id="bibl"` oraz dla znacznika div `id="bibl_info"`. Zapis funkcji `$('#bibl')` to pobranie elementu o identyfikatorze `bibl` i po wystąpieniu zdarzenia `click` wywołanie funkcji obsługującej to zdarzenie. W wyniku dla elementu o identyfikatorze `bibl_info` zostanie wykonana metoda `toggle()`, która na przemian ukrywa i pokazuje wybrany element. Metodę `toggle()` można uzupełnić o parametr określający szybkość zwijania i rozwijania tekstu. Parametr ten może przyjąć jedną z wartości:

- `slow` — wolne przewijanie:
  ```
  $('#bibl_info').toggle("slow");
  ```
- `normal` — normalna szybkość przewijania,
- `fast` — szybkie przewijanie,
- *`dowolna wartość`* — szybkość przewijania w milisekundach:
  ```
  $('#bibl_info').toggle(3000);
  ```

Zmiana zawartości pliku *moje_skrypty.js* spowoduje zmianę działania strony.

Przykład 4.4

```
$(document).ready(function()
{
  $('*').css('font-family', 'arial, verdana');
});
```

W podanym przykładzie dla wszystkich elementów strony zostały przy użyciu metody `css()` ustawione właściwości `font-family`.

Przykład 4.5

```
$(document).ready(function()
{
  $('div').css('background', 'red');
});
```

W podanym wyżej przykładzie dla wszystkich znaczników `<div>` został przy użyciu metody `css()` zmieniony kolor tła.

Zadanie 4.1

Zaprojektuj na stronie internetowej blok zawierający kilka haseł związanych z budową komputera. Hasła są widoczne na stronie, natomiast ich opis pojawia się po kliknięciu hasła. Zdefiniuj odpowiednio style dla wyświetlanych bloków informacji.

4.1.3. Filtry selektorów jQuery

Przy zastosowaniu selektorów realizujemy wybór określonych elementów modelu DOM. Jeżeli użyjemy filtrów, otrzymamy selektory, które zwracają elementy spełniające określone kryteria.

Filtry jQuery rozpoczynają się od znaku „:", po którym występuje nazwa zastosowanego filtra. Wybrane filtry jQuery zostały przedstawione w tabeli 4.2.

Tabela 4.2. Filtry biblioteki jQuery

Selektor	Znaczenie
`:first`	pierwszy element z wyszukiwanego zbioru
`:last`	ostatni element z wyszukiwanego zbioru
`:eq(indeks)`	wybrany element wyszukiwanego zbioru (indeksowanie rozpoczyna się od 0)
`:odd`	nieparzyste elementy wyszukiwanego zbioru
`:even`	parzyste elementy wyszukiwanego zbioru
`:contains(tekst)`	element zawierający podany tekst
`:hidden`	elementy ukryte oraz pola typu `<input type="hidden">`
`:visible`	elementy widoczne

Jedną z metod ułatwiających odczyt wierszy w tabeli lub elementów listy jest naprzemienne ich kolorowanie. Polega to na kolorowaniu każdego wiersza parzystego lub nieparzystego. Do tego celu mogą zostać użyte filtry `:even` i `:odd`.

Przykład 4.6

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Filt even</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moje_skrypty4.js"></script>
</head>
<body>
<table  width=400>
<tr><td width=100> Produkt</td><td width=250> Opis</td><td width=50> Cena</
td></tr>
<tr><td> Olej</td><td> Olej rzepakowy</td><td> 6 zł</td></tr>
<tr><td> Cukier</td><td> Cukier brązowy</td><td> 4 zł</td></tr>
<tr><td> Mąka</td><td> Mąka krupczatka</td><td> 3 zł</td></tr>
<tr><td> Sól</td><td> Sól warzona</td><td> 2 zł</td></tr>
<tr><td> Masło</td><td> Masło śmietankowe</td><td> 4 zł</td></tr>
<tr><td> Chleb</td><td> Chleb razowy</td><td> 3 zł</td></tr>
</table>
 </div>
</body>
</html>
```

Kod jQuery:

```
$(document).ready(function()
{
  $('tr:even').css('background', '#B0B0B0');
  $('tr:odd').css('background', '#EEE');
});
```

Wynik wykonania kodu został pokazany na rysunku 4.2.

Produkt	Opis	Cena
Olej	Olej rzepakowy	6 zł
Cukier	Cukier brązowy	4 zł
Mąka	Mąka krupczatka	3 zł
Sól	Sól warzona	2 zł
Masło	Masło śmietankowe	4 zł
Chleb	Chleb razowy	3 zł

Rysunek 4.2. Użycie filtrów :even i :odd

Tę samą metodę możemy zastosować również dla listy nieuporządkowanej. Standardem jest wykorzystywanie listy nieuporządkowanej do tworzenia menu na stronę internetową. Rozróżnianie elementów listy poprzez ich naprzemienne kolorowanie może pomóc użytkownikom w poruszaniu się po menu.

Przykład 4.7

Kod HTML:

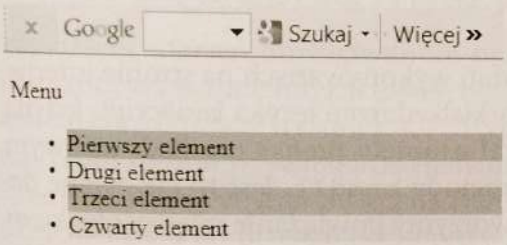
```
<!DOCTYPE HTML>
<html>
<head>
<title>Menu</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
.tlo1{
background: #EEE;
}
.tlo2{
background: #B0B0B0;
}
</style>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moj_skrypt5.js"></script>
</head>
<body>
<p>Menu</p>
<ul id="menu">
   <li> Pierwszy element</li>
   <li> Drugi element</li>
   <li> Trzeci element</li>
```

```
    <li> Czwarty element</li>
  </ul>
  </body>
  </html>
```

Kod jQuery (*moj_skrypt5*):

```
$(document).ready(function()
{
  $("ul li:even").addClass('tlo2');
  $("ul li:odd").addClass('tlo1');
});
```

W kodzie jQuery zostały pobrane wszystkie odnośniki do elementów listy. Elementy parzyste (`"ul li:even"`) zostały za pomocą metody `addClass(tlo2)` przypisane do klasy `tlo2`, a elementy nieparzyste do klasy `tlo1`. Klasy `tlo1` i `tlo2` zostały zdefiniowane w arkuszach stylów CSS. Metoda `addClass()` pozwala na dodanie wybranych selektorów do istniejącej już klasy. W wyniku wykonania kodu elementy parzyste listy zostaną dodane do klasy `tlo2`, a nieparzyste do klasy `tlo1` (rysunek 4.3).



Rysunek 4.3. Naprzemienne kolorowanie elementów menu

Metody związane z używaniem stylów CSS zostały pokazane w tabeli 4.3.

Tabela 4.3. Metody CSS

Nazwa metody	Opis
`.css()`	umożliwia pobranie lub ustawienie właściwości dla dowolnego elementu
`.addClass()`	umożliwia dodanie klasy CSS do dowolnego elementu
`.hasClass()`	umożliwia testowanie, gdy element posiada klasę
`.removeClass()`	umożliwia usunięcie klasy CSS z dowolnego elementu
`.toggleClass()`	umożliwia dodanie i usunięcie klas CSS z dowolnego elementu

Gdyby w dokumencie HTML istniały jeszcze inne listy, to dla wszystkich elementów list zostałyby przypisane klasy `tlo1` lub `tlo2`. Aby tego uniknąć, w skrypcie można dodać informację, że przypisanie do klas dotyczy tylko listy, dla której został zdefiniowany

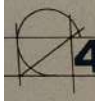
identyfikator `id="menu"`. Natomiast w kodzie HTML musi nastąpić przypisanie tego identyfikatora do wybranej listy.

Przykład 4.8

```
$(document).ready(function()
{
  $("ul#menu li:even").addClass('tlo2');
  $("ul#menu li:odd").addClass('tlo1');
});
```

Zadanie 4.2

Utwórz menu dla strony internetowej, w którym wybrana myszą pozycja menu będzie podświetlana. Wykorzystaj kod z poprzednich przykładów oraz zdefiniowane przez siebie style elementów menu.



4.2. Zdarzenia biblioteki jQuery

4.2.1. Opis zdarzeń

Obsługa zdarzeń to jedno z podstawowych działań wykonywanych na stronie internetowej. Biblioteka jQuery umożliwia użycie wszystkich zdarzeń języka JavaScript. Jedyna różnica polega na tym, że z nazw zdarzeń został usunięty prefiks `on`. Podstawowym sposobem obsługi zdarzeń jest wykorzystanie metody `bind()`. Jest to tak zwane dowiązanie do zdarzenia programu jego obsługi (tworzymy powiązanie między obiektem DOM a zdarzeniem). Metoda `bind()` ma postać:

```
$('selektor').bind(typ_zdarzenia, program_obsługi_zdarzenia)
```

Przykład 4.9

```
$(document).ready(function(){
   $('.blok').bind(click, function(){
alert("Uwaga!");
});
});
```

Istnieje również wersja skrótowa zapisu obsługi zdarzeń.

Przykład 4.10

```
$(document).ready(function(){
   $('.blok').click(function(){
alert("Uwaga!");
```

```
});
});
```

Działanie tak zapisanego kodu jest takie samo jak działanie metody `bind()`. Jego zaletą jest krótszy kod.

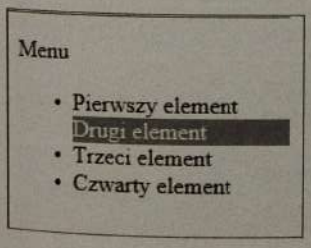
Wybrane zdarzenia biblioteki jQuery zostały przedstawione w tabeli 4.4.

Tabela 4.4. Zdarzenia biblioteki jQuery

Nazwa zdarzenia	Opis
`ready`	występuje po załadowaniu dokumentu HTML
`click`	występuje po kliknięciu przycisku myszy
`mouseout`	występuje po usunięciu kursora myszy z wybranego elementu
`mouseover`	występuje po umieszczeniu kursora myszy w obrębie wybranego elementu
`hover`	występuje po ustawieniu kursora myszy nad wybranym elementem

4.2.2. Zdarzenia myszy

Przy użyciu zdarzeń obsługiwanych przez bibliotekę jQuery można animować menu utworzone na stronie internetowej, np. przez podświetlanie elementu, nad którym pojawi się kursor myszy (rysunek 4.4). Po usunięciu kursora myszy przywrócony zostanie poprzedni sposób wyświetlania elementu. Do tego celu można wykorzystać zdarzenie `hover()`. Po najechaniu myszą na element listy nastąpi dodanie klasy (`addClass()`) do elementu, a po przesunięciu myszy poza element nastąpi jej usunięcie z elementu (`removeClass()`).



Rysunek 4.4. Podświetlanie elementu, nad którym pojawia się kursor myszy

Przed przygotowaniem skryptu, który będzie powodował podświetlenie wybranego elementu, w dokumencie HTML w arkuszu stylów należy zdefiniować nową klasę (np. `li.wybor`). Po najechaniu myszą na element klasa `li.wybor` zostanie do niego dodana, a po odsunięciu myszy zostanie usunięta.

Przykład 4.11

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Menu</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
li.wybor{
background: Gray;
color: #FFF;
}
</style>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moje_skrypty.js"></script>
</head>
<body>
<p>Menu</p>
<ul id="menu">
   <li> Pierwszy element</li>
   <li> Drugi element</li>
   <li> Trzeci element</li>
   <li> Czwarty element</li>
</ul>
</body>
</html>
```

Kod jQuery:

```
$(document).ready(function()
{
  $("ul#menu li").hover(
     function(){
    $(this).addClass('wybor');
     },
    function(){
```

```
        $(this).removeClass('wybor');
        });
    });
```

Pierwsza funkcja w podanym wyżej kodzie dodaje klasę do elementu, druga usuwa ją. Użyte w kodzie słowo kluczowe `this` stanowi odwołanie do bieżącego obiektu (`"ul#menu li"`) wykonywanej funkcji.

4.2.3. Zdarzenia formularza

Biblioteka jQuery oferuje grupę zdarzeń, które są związane z obsługą formularza. Mogą one zostać wykorzystane np. do sprawdzania poprawności wypełnienia formularza, do powiadomienia użytkownika o jakimś działaniu lub do określonego działania, gdy użytkownik opuści pole. Zdarzenia formularza biblioteki jQuery zostały przedstawione w tabeli 4.5.

Tabela 4.5. Zdarzenia formularza

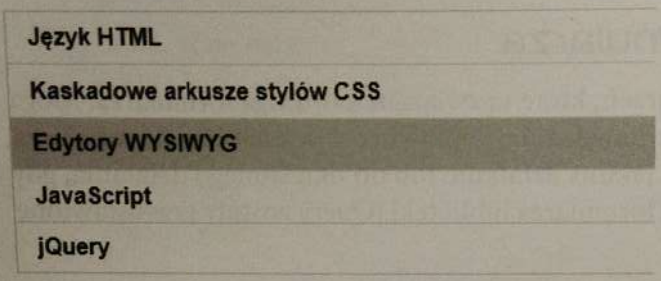
Zdarzenie	Opis
`change()`	występuje w momencie edytowania zawartości pola
`focus()`	występuje w momencie aktywowania pola tekstowego (wybranie pola klawiszem *Tab* lub jego zaznaczenie)
`focusin()`	występuje w momencie aktywowania pola tekstowego znajdującego się wewnątrz elementu
`focusout()`	występuje w momencie dezaktywowania elementu
`blur()`	występuje w momencie dezaktywowania elementu na rzecz innego elementu
`select()`	występuje w momencie zaznaczenia tekstu wewnątrz elementu
`submit()`	występuje w momencie wysłania formularza po kliknięciu przycisku *Submit*
`reset()`	występuje w momencie zresetowania formularza przy użyciu polecenia `input type="reset"`

4.3. Zastosowanie biblioteki jQuery na stronie internetowej

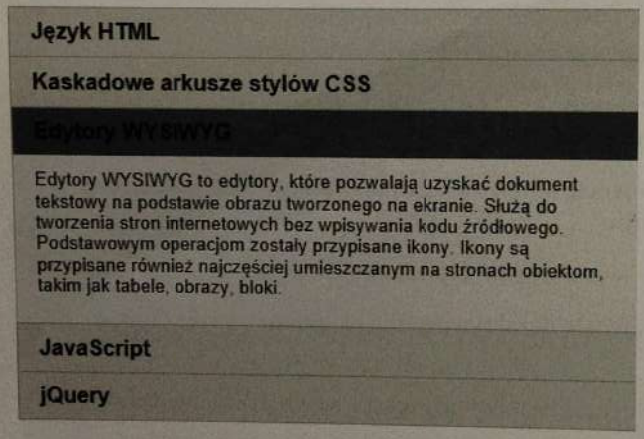
Jednym ze sposobów tworzenia dynamicznych elementów stron internetowych jest wykorzystanie biblioteki jQuery napisanej w języku JavaScript. Biblioteka pozwala na prostą manipulację elementami na stronie, pracę ze zbiorami elementów oraz udostępnia wiele efektów wizualnych.

4.3.1. Pokazywanie i ukrywanie treści

Tak zwana harmonijka to dobry sposób na prezentowanie lub ukrywanie treści na stronie internetowej. Może zostać również wykorzystana jako rozwijane menu nawigacyjne (rysunek 4.5). W tak opracowanym narzędziu hasła dla treści, które będą wyświetlane lub ukrywane, są zorganizowane w postaci menu. Kliknięcie wybranej opcji wyświetla ukrytą treść. Kliknięcie innej opcji ukrywa wyświetlaną treść i wyświetla inną (rysunek 4.6).



Rysunek 4.5. Menu nawigacyjne



Rysunek 4.6. Wyświetlenie treści ukrytej w menu

Przykład 4.12

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Inf rozwijana</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moje_skrypty.js"></script>
```

```
<style type="text/css">
body {
  margin: 10px auto;
  width: 600px;
}
.blok {
  width: 450px;
  border-bottom: solid 1px #b0b0b0;
}
.blok h3 {
  background: # c6d2f2 ;
  padding: 8px 15px;
  margin: 0;
  font: bold 12pt Arial, sans-serif;
  border: solid 1px #b0b0b0;
  border-bottom: none;
  cursor: pointer;
}
.blok h3:hover {
  background-color: # 88a3e8;
}
.blok h3.aktywny {
background-color: #880000;
}
.blok p {
  background: # e1e6f4;
  margin: 0;
  padding: 10px 15px 20px;
  border-left: solid 1px #c4c4c4;
  border-right: solid 1px #c4c4c4;
  font:  10pt Arial, sans-serif;
  display: none;
}
</style>

</head>
```

```
<body>
<div class="blok">
  <h3>Język HTML</h3>
  <p>Podstawowym elementem języka HTML jest znacznik. Znaczniki są poleceniami umieszczonymi w nawiasach ostrych. Informują one przeglądarkę o wyglądzie otwieranej strony oraz o strukturze tekstu umieszczonego na niej. </p>
  <h3>Kaskadowe arkusze stylów CSS</h3>
  <p>Język CSS określa układ graficzny dokumentu HTML: parametry czcionki, wysokość i szerokość obrazków, ich położenie, rodzaj tła itp. Wszystkie polecenia dotyczące formatowania powinny zostać umieszczone w oddzielnym pliku (arkuszu) i być powiązane z elementami zdefiniowanymi w kodzie HTML.</p>
  <h3>Edytory WYSIWYG</h3>
  <p>Edytory WYSIWYG to edytory, które pozwalają uzyskać dokument tekstowy na podstawie obrazu tworzonego na ekranie. Służą do tworzenia stron internetowych bez wpisywania kodu źródłowego. Podstawowym operacjom zostały przypisane ikony. Ikony są przypisane również najczęściej umieszczanym na stronach obiektom, takim jak tabele, obrazy, bloki.</p>
  <h3>JavaScript</h3>
  <p>JavaScript odgrywa coraz większą rolę w projektowaniu interaktywnych stron internetowych. Jego możliwości są bardzo duże, od prostej manipulacji danymi, przez dynamiczne modelowanie struktury strony, obsługę rozszerzeń multimedialnych, aż po tworzenie odrębnych aplikacji.</p>
  <h3>jQuery</h3>
  <p>Biblioteka jQuery to framework zawierający różnorodne funkcje, przy których zastosowaniu można realizować często spotykane zadania (np. animacje, obsługę zdarzeń, tworzenie efektów na stronie internetowej). Jej głównymi zaletami są łatwość użycia, niezależność od przeglądarki i małe rozmiary.</p>
</div>
</body>
</html>
```

Kod jQuery:

```
$(document).ready(function(){

  $(".blok h3").eq(0).addClass("aktywny ");
  $(".blok p").eq(0).show();
```

```
    $(".blok h3").click(function(){
      $(this).next("p").slideToggle("slow")
      .siblings("p:visible").slideUp("slow");
      $(this).toggleClass("aktywny ");
      $(this).siblings("h3").removeClass("aktywny ");
    });

});
```

W podanym wyżej przykładzie kodu HTML zawartość strony zorganizowano w postaci bloku `<div class="blok">`. Opcje menu zostały zdefiniowane w postaci nagłówka `<h3>`, a wyświetlana treść w postaci akapitu `<p>`.

Następnie zdefiniowane zostały style: dla klasy `.blok` przypisanej do bloku `<div>`, dla nagłówka `h3` (`.blok h3`), dla akapitu (`.blok p`) oraz dla nagłówka `h3`, gdy będzie wskazany myszą (`.blok h3:hover`).

Natomiast zapis w skrypcie `$('.blok h3').eg(0).addClass("active")` to pobranie z klasy `.blok` elementu `h3` o indeksie `0` oraz dodanie go do klasy `aktywny`. Metoda `eg(indeks)` wybiera element, który jest zgodny z podanym indeksem. W tym wypadku jest to pierwszy nagłówek `h3`, który po przypisaniu mu klasy `aktywny` zmieni swój wygląd. Instrukcja `$(".blok p").eq(0).show()` pobierze pierwszy element `p` i doda metodę `show()`. Jeżeli element jest ukryty, metoda ta zmienia wartość atrybutu `display` na `"block"`. Jeżeli wybrany element został już wyświetlony, metoda nie będzie powodowała żadnego działania.

Kolejny blok instrukcji odnosi się do sytuacji, gdy po kliknięciu określonej opcji powinien zostać wyświetlony tekst znajdujący się za wybranym nagłówkiem.

Polecenie `$(".blok h3").click(function()` mówi, że po kliknięciu wybranego nagłówka powinna zostać wywołana funkcja, która będzie realizowała wyświetlenie sąsiadującego z nim akapitu: `$(this).next("p").slideToggle("slow").siblings("p:visible").slideUp("slow")`. Metoda `next()` pobiera następny po wskazanym pasujący element typu rodzeństwo: `siblings()`. Metoda `slideToggle("slow")` wyświetla lub ukrywa wybrany element. Parametr `"slow"` określa szybkość ukrywania lub pokazywania elementu.

W wyniku działania skryptu dla wskazanego elementu (`h3`) zostanie pobrany następny element `p` (`"p:visible"`), który się wyświetli. Widoczny siostrzany (`siblings`) element `p` zostanie ukryty (`slideUp`). Do wybranego elementu `h3` zostanie dodana klasa *aktywny*: `$(this).toggleClass("aktywny")`, a siostrzany element `h3` utraci klasę *aktywny*.

4.3.2. Proste animacje

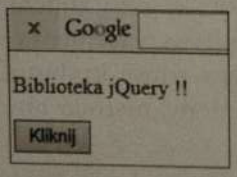
Strony internetowe z pokazami slajdów, animowanymi menu lub innymi animacjami coraz częściej są tworzone w języku JavaScript. Biblioteka jQuery korzysta z efektów dostępnych w JavaScript, a konfigurowanie ich przy użyciu jQuery jest dużo prostsze.

Do pokazywania i ukrywania elementów często wykorzystuje się metody `show()` oraz `hide()`. Można je wywoływać z parametrem określającym czas trwania akcji. Parametrem może być wartość `'slow'`, `'normal'` lub `'fast'`.

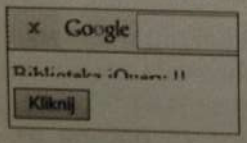
Kolejna metoda to `slideToggle()`. Jest ona odpowiednikiem metod `show()` oraz `hide()`. Automatycznie sprawdza, czy wybrany element jest ukryty, czy nie, i wyświetla go lub ukrywa.

Następne metody to `fadeIn()` oraz `fadeOut()`. Działają prawie identycznie jak `show()` oraz `hide()`. Różnica polega na tym, że ukazują i ukrywają element, zmieniając stopniowo jego przezroczystość.

Metody `slideDown()` oraz `slideUp()` też pokazują lub ukrywają elementy, ale efektem animacji jest zsuwanie i rozsuwanie elementu. Przykład wykorzystania tych metod do animowania treści strony został pokazany poniżej, a wynik interpretacji kodu widać na rysunkach 4.7 i 4.8.



Rysunek 4.7. Pokazanie wszystkich elementów



Rysunek 4.8. Ukrywanie napisu

Przykład 4.13

```
<head>
<title>Zjazd</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script>
$(document).ready(function()
```

```
{
$("button").click(function(){
  $("#p1").css("color","red")
    .slideUp(2000)
    .slideDown(2000);
 });
});
</script>
</head>

<body>
<p id="p1">Biblioteka jQuery !!</p>
<button>Kliknij</button>
</body>
```

Zadanie 4.3

Przeanalizuj kod podany w przykładzie 4.13. Utwórz podobną animację na swojej stronie internetowej.

4.3.3. Animacje zaawansowane

Zmiana rozmiaru elementu

Najbardziej zaawansowaną metodą służącą do animowania elementów strony internetowej jest metoda `animate()`. Metoda ta działa na właściwościach stylów CSS, ale tylko tych, które przyjmują wartości liczbowe (np. `margin`, `padding`, `top`). Pierwszy parametr metody to tablica asocjacyjna z właściwościami stylów, czyli animowane właściwości. Drugi to czas wykonywania animacji (parametr opcjonalny). Określany jest wartościami `'slow'`, `'normal'`, `'fast'` lub wartościami podanymi w milisekundach. Następny parametr `easing` (opcjonalny) określa sposób przejścia animacji (liniowy — `'linear'` — lub wahadłowy — `'swing'`). Domyślnie animacje stosują przejście liniowe. Ostatni parametr to funkcja zwrotna (opcjonalny), czyli funkcja wywoływana po zakończeniu animacji.

Przykład 4.14

```
<head>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>

<script type="text/javascript">
```

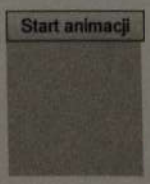
```
$(document).ready(function(){
$("button").click(function(){
$("#blok").animate({
width: "500px", height: "500",
}, 1500 );
 });
 });
</script>

<style type="text/css">
#blok {
  width: 100px; height: 100px;
  background: #c6d2f2;
  position: absolute;
 }
</style>
</head>

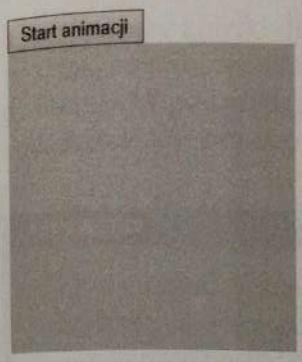
<body>
  <button>Start animacji</button>
<div id="blok">
</div>
</body>
```

W podanym wyżej przykładzie funkcja `animate()` została wywołana dla elementu `<div>` o identyfikatorze `id='blok'`. Zmieniającymi się parametrami są wysokość i szerokość elementu. Czas trwania animacji to 1500 milisekund.

Po kliknięciu przycisku *Start animacji* wynikiem wykonania kodu będzie zmiana rozmiaru bloku o identyfikatorze `id='blok'` (rysunki 4.9 i 4.10).



Rysunek 4.9. Początkowy rozmiar bloku



Rysunek 4.10. Zmieniony rozmiar bloku

Zmiana rozmiaru i położenia elementu

Domyślnie każdy element dokumentu HTML ma określoną pozycję, która nie może być zmieniona. Aby móc nią manipulować, należy najpierw ustawić właściwość `position` na wartość `relative`, `fixed` lub `absolute`. Jednocześnie można manipulować wieloma właściwościami elementu.

Przykład 4.15

```
<head>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript">

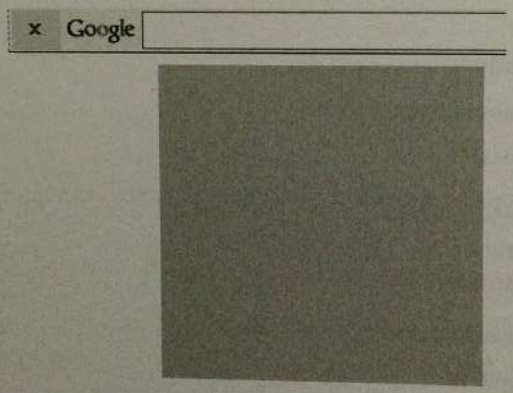
$(document).ready(function(){
$("#blok").mouseover(function(){
  $(this).animate({
  width: "+=300px", height: "+=300px", left: "250px", opacity: "0.5",
  }, 1500 );
  });
$("#blok").mouseout(function(){
  $(this).animate({
  width: "100px", height: "100px", left: "10px", opacity: "1",
  }, 1500 );
  });
 });
</script>

<style type="text/css">
```

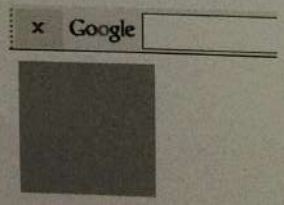
```
#blok {
  width: 100px; height: 100px;
  background: #c6d2f2;
  position: absolute;
}
</style>
</head>

<body>
<div id="blok">
</div>
</body>
```

W podanym przykładzie wielkość bloku zmienia się w zależności od tego, czy kursor myszy znajduje się nad nim. Metoda `animate()` została zadeklarowana dwukrotnie i zostanie wykonana w zależności od występującego zdarzenia. Gdy najedziemy kursorem myszy na element, zmieni on swoje położenie oraz rozmiar (rysunek 4.11). Po odsunięciu kursora myszy powróci do pierwotnego położenia i rozmiaru (rysunek 4.12). Przy definiowaniu właściwości wykorzystana została możliwość ustawienia wartości względnych (`width: "+=300px", height: "+=300px"`). Przy podawaniu wartości względnych przed podaną wartością należy umieścić += lub -=.



Rysunek 4.11. Zmiana rozmiaru i położenia bloku



Rysunek 4.12. Powrót do pierwotnego położenia i rozmiaru

jQuery tworzy kolejki dla animacji. Dlatego jeżeli w kodzie występuje wiele wywołań tej samej metody, zostanie utworzona kolejka wywołań. Metody uruchamiane są jedna po drugiej. W podanym przykładzie kilkakrotne najechanie kursorem myszy na obiekt i odsunięcie kursora spowoduje wielokrotne powtórzenie animacji.

UWAGA

Z użyciem metody `animate()` można przetwarzać większość właściwości CSS. Należy pamiętać, że nazwy właściwości nie mogą zawierać znaku - (łącznik), nazwa taka powinna mieć postać np. `paddingleft`, a nie `padding-left`.

Zadanie 4.4

Utwórz animacje wybranych elementów swojej strony internetowej. W pracy wykorzystaj metodę `animate()` oraz zmianę atrybutów stylów CSS.

Wyświetlanie podpowiedzi do treści

Wyświetlanie podpowiedzi w postaci dymków informacyjnych nad odnośnikami lub elementami formularza pozwala na łatwiejsze poruszanie się w obrębie strony. Podpowiedzi mogą być instrukcjami, jak dalej postępować, mogą wyjaśniać pewne terminy lub zawierać istotne informacje.

Przykład 4.16

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Dymki</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript" src="js/moje_skrypty.js"></script>
<style type="text/css">
body {
  background: url(sejm.jpg) no-repeat;
  margin: 10px auto;
  width: 570px;
  font: 75%/120% Arial, sans-serif;
}
.menu {
```

```
    margin: 100px 0 0;
    padding: 0;
    list-style: none;
  }
  .menu li {
    padding: 0;
    margin: 0 2px;
    float: left;
    position: relative;
    text-align: center;
  }
  .menu a {
    padding: 14px 10px;
    display: block;
    color: #000000;
    width: 144px;
    text-decoration: none;
    font-weight: bold;
    background: url(menu.png) no-repeat center;

  }
  .menu li p {
    background: url(opis.png) no-repeat;
    width: 180px;
    height: 54px;
    position: absolute;
    top: -85px;
    left: -10px;
    text-align: center;
    padding: 10px 10px 10px;
    font-style: normal;
    z-index: 2;
    display: none;
  }
  </style>
  </head>
```

```
<body>
<ul class="menu">
  <li> <a href="http://www.sejm.gov.pl/">Sejm Rzeczypospolitej Polskiej</a>
   <p>Wszelka władza społeczności ludzkiej początek swój bierze z woli
narodu</p>
  </li>
  <li> <a href="http://www.men.gov.pl/">Ministerstwo Edukacji Narodowej</a>
   <p>Takie będą Rzeczypospolite, jakie ich młodzieży chowanie</p>
   </li>
   <li> <a href="http://www.cke.edu.pl/">Centralna Komisja Egzaminacyjna</a>
    <p>Edukacja skuteczna, przyjazna i nowoczesna</p>
    </li>
 </ul>
 </body>
 </html>
```

Kod jQuery:

```
$(document).ready(function(){

  $(".menu a").hover(function() {
    $(this).next("p").animate({opacity: "show", top: "-75"}, "slow");
  }, function() {
    $(this).next("p").animate({opacity: "hide", top: "-85"}, "fast");
  });
});
```

Na stronie internetowej umieszczono kilka elementów menu (rysunek 4.13). Po wybraniu elementu pojawi się okienko z podpowiedzią (rysunek 4.14).



Rysunek 4.13. Menu umieszczone na stronie internetowej



Rysunek 4.14. Wyświetlenie podpowiedzi do menu

Działanie kodu jQuery jest następujące: po wskazaniu myszą elementu menu `$(".mena").hover` zostanie wywołana funkcja. Funkcja ta wybierze następny po wskazanym element `p` i uruchomi dla niego metodę `animate()`. Metoda ta wyświetli zawartość wybranego elementu `p` w miejscu przesuniętym w górę w stosunku do jego pierwotnego położenia i ukryje wcześniej wyświetlany element `p`.

Zadanie 4.5

Wykorzystując przykład 4.16 i poznane narzędzia projektowania stron internetowych, zaprojektuj dla swojej strony internetowej menu z podpowiedziami w postaci dymków pojawiających się po ustawieniu myszy nad wybraną jego opcją.

4.3.4. Pokaz zdjęć

Z wykorzystaniem biblioteki jQuery można w bardzo prosty sposób tworzyć animowane pokazy slajdów. Najczęściej takie pokazy zawierają galerię miniatur zdjęć; wybrane przez użytkownika zdjęcie jest wyświetlane w pełnych rozmiarach. Sposobów prezentowania zdjęć jest wiele. Na rysunku 4.15 został przedstawiony jeden z nich. W dolnej części znajduje się galeria miniatur obrazów, w górnej wyświetlany jest aktualnie wybrany obraz.

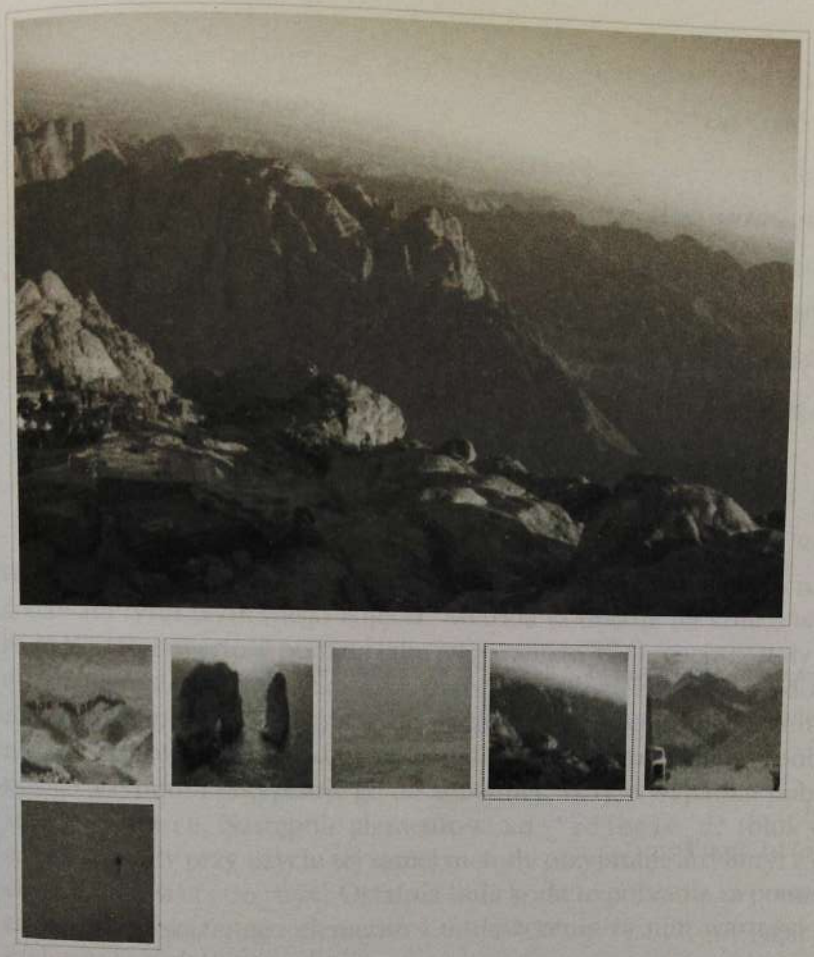
Do utworzenia podobnego pokazu potrzebne są miniatury zdjęć oraz zdjęcia w oryginalnej wielkości. Do ich przygotowania można wykorzystać program Web Album Generator, który generuje miniatury zdjęć oraz tworzy zdjęcia o określonych rozmiarach, najczęściej mniejszych od zdjęć oryginalnych.

Po przygotowaniu zdjęć istotnym elementem jest takie zdefiniowanie stylów CSS, aby cały pokaz dobrze prezentował się na stronie internetowej.

Przykład 4.17

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Pokaz zdjęć (Zdjęcie 4)



Rysunek 4.15. Pokaz zdjęć na stronie internetowej

```
<title>Pokaz zdjęć</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $("h2").append('<em></em>');
  $(".galeria a").click(function(){
    var do_zdj = $(this).attr("href");
    var do_tyt = $(this).attr("title");
    $("#zdjecie_d").attr({ src: do_zdj, alt: do_tyt });
    $("h2 em").html(" (" + do_tyt + ")"); return false;
  });
```

```
});
</script>

<style type="text/css">
body {
  margin: 20px auto;
  padding: 0;
  width: 580px;
  font: 75%/120% Arial, Helvetica, sans-serif;
}
h2 {
  font: bold 190%/100% Arial, Helvetica, sans-serif;
  margin: 0 0 .2em;
}
h2 em {
  font: normal 80%/100% Arial, Helvetica, sans-serif;
  color: #999999;
}
#zdjecie_d {
  border: solid 1px #ccc;
  width: 550px;
  height: 400px;
  padding: 5px;
}
.galeria img {
  border: solid 1px #ccc;
  width: 100px;
  height: 100px;
  padding: 4px;
}
.galeria img:hover {
  border-color: #FF9900;
}
</style>
</head>
```

```
<body>
<h2>Pokaz zdjęć</h2>
<p><img id="zdjecie_d" src="zdj/zdj1.jpg" alt="Duże zdjęcie" /></p>
<p class="galeria">
  <a href="zdj/zdj1.jpg" title="Zdjęcie 1"><img src="zdj/zdj1_s.jpg" /></a>
  <a href="zdj/zdj2.jpg" title="Zdjęcie 2"><img src="zdj/zdj2_s.jpg" /></a>
  <a href="zdj/zdj3.jpg" title="Zdjęcie 3"><img src="zdj/zdj3_s.jpg" /></a>
  <a href="zdj/zdj4.jpg" title="Zdjęcie 4"><img src="zdj/zdj4_s.jpg" /></a>
  <a href="zdj/zdj5.jpg" title="Zdjęcie 5"><img src="zdj/zdj5_s.jpg" /></a>
  <a href="zdj/zdj6.jpg" title="Zdjęcie 6"><img src="zdj/zdj6_s.jpg" /></a>
</p>
</body>
</html>
```

W skrypcie jQuery została użyta metoda `append()`, która do nagłówka `h2` będzie dodawała tytuł wybranego zdjęcia. Metoda ta dodaje na końcu wybranego elementu kodu HTML zawartość podaną jako jej parametr. W skrypcie zostały również utworzone dwie zmienne. Pierwsza, `do_zdj`, przechowuje ścieżkę dostępu do pliku z wybranym zdjęciem. Druga, `do_tyt`, przechowuje tytuł zdjęcia umieszczony w odnośniku do niego. Zastosowana metoda `attr()` dla wybranego elementu pobiera wartość wskazanego atrybutu i przypisuje go do zmiennej. W tym wypadku pobranymi atrybutami są `href` i `title`. Następnie elementowi `id='zdjecie_d'` (blok wyświetlający duże zdjęcie) zostały przy użyciu tej samej metody przypisane atrybuty: `src` równy zmiennej `do_zdj` oraz `alt=do_tyt`. Ostatnia linia kodu to pobranie za pomocą metody `html()` zawartości wskazanego elementu i umieszczenie za nim wartości zmiennej `do_tyt`, czyli tytułu wybranego zdjęcia.

4.4. Walidacja formularzy

Dla tworzonych na stronie internetowej formularzy powinna być stosowana walidacja. Walidacja pozwoli sprawdzić, czy użytkownik wypełnił wymagane pola formularza oraz czy wpisał prawidłowo dane. Tworzenie walidacji formularza z wykorzystaniem języka JavaScript zostało przedstawione w podrozdziale 3.10. Stanie się ono łatwiejsze po zastosowaniu biblioteki jQuery oraz wtyczki Validation opracowanej dla tej biblioteki. Plik wtyczki można pobrać ze strony *http://bassistance.de/jquery-plugins/jquery-plugin-validation/*. Po jego rozpakowaniu do walidacji formularza potrzebne będą tylko pliki: *jquery.validate.js*, zawierający walidator, oraz *messages_pl.js*, zawierający standardowe komunikaty walidacji zapisane w języku polskim. Pliki te kopiujemy i umieszczamy w dowolnym folderze z kodem strony. Np. w folderze *js* zawierającym bibliotekę jQuery tworzymy folder *validate* i w nim umieszczamy potrzebne pliki.

Informację o tym, że w kodzie HTML będzie wykorzystywana pobrana wtyczka, umieszczamy w sekcji <head> dokumentu, najlepiej po deklaracji pobrania biblioteki jQuery. Następnie dodajemy skrypt uruchamiający wtyczkę dla tworzonego formularza.

Przykład 4.18

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Walidacja formularza</title>

<script type="text/javascript" src="js/jquery-1.8.3.js"></script>

<script src="js/validate/jquery.validate.js" type="text/javascript"></
script>

<script src="js/validate/messages_pl.js" type="text/javascript"></script>

<script type="text/javascript">
  jQuery(document).ready(function(){
  jQuery("#formularz").validate();
  });
</script>

<style type="text/css">
body{
font-family: Arial, Helvetica, san-serif; font-size: 12px; font-weight:
normal;
}
#formularz {
width: 450px;
background-color: #cfdfdf;
}
</style>
</head>
```

```
<body bgcolor="#dfdfdf">
<form id="formularz" action="mailto:nauka@gmail.com" method="post">
<fieldset>
<b> <font size=4>Dane osobowe</font></b></br></br>

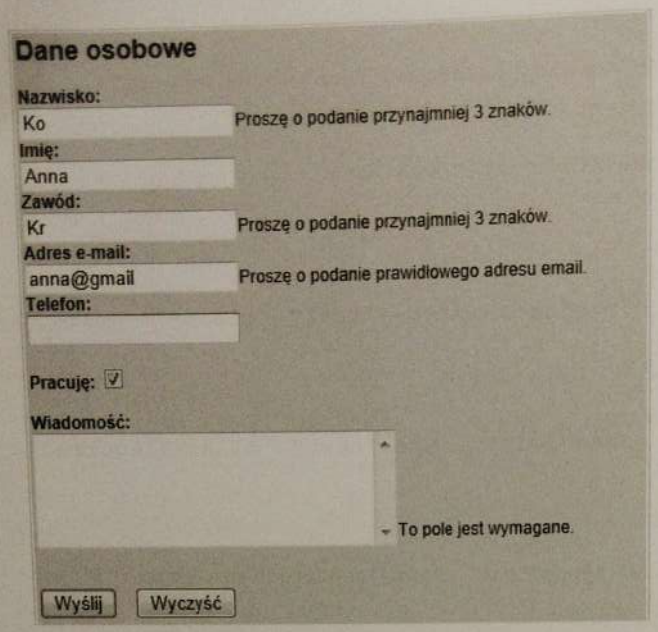
<b>Nazwisko:</b>
<div><input id="nazw" name="nazw" type="text" class="required"
minlength="3"/></div>
<b>Imię:</b>
<div><input id="imie" name="imie" type="text" class="required"
minlength="2"/></div>
<b>Zawód:</b>
<div><input id="zaw" name="zaw" type="text" class="required" minlength="3"/></
div>
<b>Adres e-mail:</b>
<div><input id="email" name="email" type="text" class="required email"/></
div>

<b>Telefon:</b>
<div><input id="tel" name="tel" type="text" class="number" minlength="9"/></
div><br>
<label><b> Pracuję: </b><input type="checkbox" name="opcje" id="opcje"
maxlength="1">
 </label><br><br>

<b>Wiadomość:</b><br>
<div><textarea id="inf" cols="30" name="inf" rows="5" class="required"
minlength="5"></textarea></div><br><br>

<input class="submit" type="submit" value="Wyślij"/>   
<input type="reset" value="Wyczyść"/>
</fieldset>
</form>
</body>
</html>
```

Efekt działania powyższego kodu pokazany jest na rysunku 4.16. Komunikaty pojawiają się automatycznie po błędnym wypełnieniu pól formularza.



Rysunek 4.16. Walidacja formularza

W podanym przykładzie w pierwszej linii kodu po znaczniku `<title>` zostało zapisane dołączenie biblioteki jQuery, następnie dołączenie wtyczki Validate, w kolejnej dołączenie pliku z komunikatami w języku polskim. Następne linie kodu to skrypt uruchamiający wtyczkę dla formularza o zdefiniowanym identyfikatorze `id='formularz'`. Sekcja `<body>` zawiera polecenia utworzenia formularza, który zostanie poddany walidacji. Pola formularza `nazwisko` i `zawód` są obowiązkowe i muszą zawierać przynajmniej trzy znaki. Pole `imię` jest również obowiązkowe i musi zawierać przynajmniej dwa znaki. Pole `e-mail` jest też obowiązkowe i powinno zostać sprawdzone pod kątem poprawnego adresu e-mail. W przypadku kolejnego pola, `telefon`, powinno zostać sprawdzone, czy zawiera cyfry i czy nie jest ich mniej niż dziewięć. Ostatnie pole, `wiadomość`, może zawierać dowolną informację, ale nie krótszą niż pięć znaków.

Weryfikacja formularza przy użyciu wtyczki Validation polega na wywołaniu funkcji dostępnych w walidatorze. Prawidłowość danych jest sprawdzana automatycznie i potwierdzana odpowiednim komunikatem.

W kodzie wtyczki dla pól, których zawartość jest wymagana, została zdefiniowana klasa `required` (`class="required"`). Jeżeli jest wymagana minimalna liczba wprowadzonych znaków, jest stosowany atrybut `minlength` (np. `minlength="2"`). Do weryfikacji poprawności adresu e-mail jest stosowana klasa `email` (`class="email"`), a do weryfikacji numeru telefonu klasa `number` (`class="number"`).

Zadanie 4.6

Wykorzystując kod z przykładu 4.18, utwórz na swojej stronie internetowej formularz rejestracyjny z podstawowymi danymi użytkownika. Przy użyciu wtyczki *Validation* przeprowadź walidację formularza. Wykorzystując kod HTML i arkusze CSS, popraw wygląd utworzonego formularza.

PYTANIA KONTROLNE

1. Jakie zadania można realizować na stronie za pomocą biblioteki jQuery?
2. Podaj metody dołączania biblioteki jQuery do skryptu.
3. Co może być metodą biblioteki jQuery?
4. Jaką rolę odgrywają filtry jQuery?
5. Do jakich celów można w bibliotece jQuery wykorzystać obsługę zdarzeń?
6. Jak działa metoda `animate()`?
7. Do czego służy wtyczka `Validation`?

ZADANIE

Zbuduj stronę internetową zawierającą informacje na temat komputerów. W lewej części strony utwórz menu zawierające trzy pozycje: *Programy*, *Sprzęt*, *Systemy*. Wybrana myszą pozycja menu powinna zostać podświetlona. Kliknięcie wybranej pozycji menu spowoduje wyświetlenie w wybrany przez Ciebie sposób opisu zaznaczonego hasła. Po wyborze innej pozycji menu treść hasła zostanie ukryta. W głównej części strony umieść formularz rejestracyjny dla osób odwiedzających stronę. Powinien on zawierać następujące pola: *Nazwisko*, *Imię*, *Adres*, *Wykształcenie*, *Wiek*, *Adres e-mail*. Opracuj walidację formularza, która sprawdzi, czy użytkownik wypełnił wymagane pola oraz wpisał prawidłowo do niego dane.

5 Ajax

5.1. Wprowadzenie

Technologia Ajax (ang. *Asynchronous JavaScript and XML*) to połączenie języka JavaScript i XML. Jest to technologia tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez ponownego wczytania całego dokumentu w sposób asynchroniczny. Pozwala ona na wyeliminowanie największej wady aplikacji internetowych i klasycznych witryn, którą była konieczność ponownego wczytywania strony po każdej zmianie jej zawartości. W technologii Ajax proces przeładowania danych odbywa się w tle, w sposób niezauważalny dla użytkownika. Technologia Ajax pozwala na tworzenie szybkich i dynamicznych stron internetowych. Przykładami aplikacji opartych na technologii Ajax są: Google Maps, Gmail, YouTube, Facebook. Ajax nie jest nowym językiem programowania. To nowy sposób wykorzystania istniejących standardów.

Technologia Ajax jest oparta na standardach internetowych i jej podstawowymi elementami są:

- `XMLHttpRequest` — jest to obiekt umożliwiający asynchroniczne przesyłanie danych do serwera.
- `JavaScript/DOM` — dowolny język skryptowy funkcjonujący po stronie użytkownika (może to być JavaScript albo inny język skryptowy, np. VBScript). Jest on wykorzystywany do wyświetlenia informacji i tworzenia interakcji z użytkownikiem.
- `XML` — to język znaczników używany jako format przesyłania informacji. Dane często przekazywane są w innym formacie i wtedy odbierane są jako tekst. Mogą to być fragmenty HTML, fragmenty kodu JavaScript lub inne.

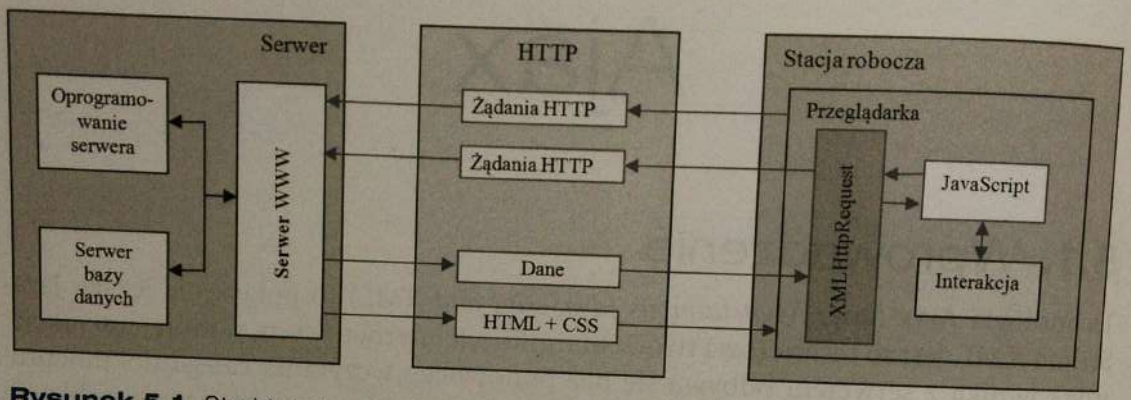
W praktyce wykorzystywane są jeszcze skrypty działające po stronie serwera i współpracujące z bazą danych.

Wadą rozwiązań opartych na technologii Ajax jest ograniczenie dostępu do treści strony dla tych użytkowników, którzy mają zablokowane skrypty lub korzystają z czytników ekranowych. Problemy te występują tylko wtedy, gdy nie stosowano technik tworzenia nieinwazyjnego kodu JavaScript.

5.2. Obiekt XMLHttpRequest

Obiekt `XMLHttpRequest` jest obiektem języka JavaScript służącym do pobierania danych z użyciem protokołu HTTP. Za jego pomocą można przesyłać dane w wielu formatach, np. jako zwykły tekst, jako dokument XML lub JSON.

Struktura komunikacji między serwerem a użytkownikiem z użyciem obiektu `XMLHttpRequest` została pokazana na rysunku 5.1.



Rysunek 5.1. Struktura komunikacji między serwerem a użytkownikiem w technologii Ajax

5.2.1. Tworzenie obiektu XMLHttpRequest

Do wysyłania lub pobierania informacji z serwera język JavaScript potrzebował formularza. Po kliknięciu przycisku *Wyślij* (*Submit*) informacje zapisane w formularzu były przesyłane na serwer i należało poczekać na odpowiedź z serwera. Po jej otrzymaniu następowało przeładowanie strony, w wyniku czego otrzymywaliśmy aktualną jej zawartość. Strona była wyświetlana na nowo za każdym razem, gdy użytkownik wysyłał formularz. Wykorzystując technologię Ajax, język JavaScript może bezpośrednio komunikować się z serwerem za pośrednictwem obiektu `XMLHttpRequest`. Dzięki temu obiektowi strona internetowa może wysyłać żądania oraz otrzymywać odpowiedzi z serwera, wykonując te czynności w tle.

Wszystkie współczesne przeglądarki posiadają wbudowaną obsługę obiektu `XMLHttpRequest`. Starsze wersje przeglądarki Internet Explorer implementowały obiekt `XMLHttpRequest` jako obiekt `ActiveX`.

Procedura tworzenia obiektu `XMLHttpRequest` zalecana przez organizację W3C wygląda następująco:

```
function spr_obiekt()
{
var xmlhttp;
if (window.XMLHttpRequest)
```

```
  {
//Dla przeglądarek IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {
//Dla IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
}
```

W podanym kodzie sprawdzane jest, czy przeglądarka obsługuje obiekt `XMLHttpRequest` (`window.XMLHttpRequest`). Jeżeli tak, to tworzony jest taki obiekt i zostaje on przypisany do zmiennej `xmlhttp`. W przeciwnym razie tworzony jest obiekt `ActiveX`, który zostaje przypisany do zmiennej `xmlhttp`.

Podany kod może stać się szablonem używanym za każdym razem, gdy będzie wymagany obiekt `XMLHttpRequest`.

5.2.2. Właściwości obiektu XMLHttpRequest

onreadystatechange

Właściwość ta jest wywoływana za każdym razem, gdy zmieni się stan `readyState`. Przechowuje ona funkcję, która może przetworzyć odpowiedź otrzymaną z serwera. Do jej ustawienia stosowany jest następujący kod:

```
xmlhttp.onreadystatechange=function()
{
//tutaj wstawiamy kod obsługi
}
```

readyState

Właściwość ta przechowuje status odpowiedzi serwera. Za każdym razem, gdy zmienia się wartość właściwości `readyState`, wykonywana jest funkcja przypisana do właściwości `onreadystatechange`. Właściwość `readyState` może przyjmować jedną z wartości od 0 do 4:

- 0 — uninitialized (żądanie nie zostało zainicjowane),
- 1 — loading (połączenie z serwerem ustanowione),
- 2 — loaded (żądanie wysłane),
- 3 — interactive (żądanie jest przetwarzane — odbieranie danych),
- 4 — complete (żądanie zostało zakończone).

Stan właściwości `readyState==4` oznacza, że serwer zakończył obsługę żądania, ale nie znaczy to, że dane zostały dostarczone.

Sprawdzenie, czy serwer zakończył obsługę żądania, może mieć postać:

```
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4)
{
//można pobrać dane przesłane z serwera
}
}
```

status

Właściwość ta zawiera informację o stanie żądania. Informacja ta jest przechowywana w postaci kodu zgodnego ze specyfikacją HTTP. Niektóre z wartości kodu odpowiedzi serwera zostały podane niżej:

- `200`: *OK* (żądanie zakończone sukcesem),
- `400`: *Bad Request* (nieprawidłowe żądanie),
- `401`: *Unauthorized* (nieautoryzowane),
- `403`: *Forbidden* (dostęp zabroniony),
- `404`: *Not Found* (dokument nie został znaleziony),
- `500`: *Internal Server Error* (wewnętrzny błąd serwera).

Gdy właściwość `readyState` jest równa `4` i właściwość `status` jest równa `200` żądanie zakończyło się sukcesem.

responseText

Właściwość ta przechowuje dane otrzymane z serwera.

Przykład 5.1

```
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("blok").innerHTML=xmlhttp.responseText;
    }
  }
```

responseXML

Właściwość ta przechowuje dane otrzymane z serwera. Jest używana, gdy odpowiedź z serwera jest typu XML i będzie analizowana jako obiekt XML.

5.3. Żądania Ajax

5.3.1. Opcje żądania Ajax

Podczas wysyłania żądania do serwera mogą zostać ustawione następujące opcje:

- `cache` — umożliwia włączenie buforowania dla wysyłanych żądań. Domyślnie ustawiona na `true`. Natomiast dla typów danych `script` i `jsonp` ustawiana na `false`.
- `contentType` — pozwala na ustawienie typu danych wysyłanych do serwera. Domyślna wartość to `application/x-www-form-urlencoded`.
- `data` — to obiekt lub łańcuch danych zawierający dane przesyłane do serwera.
- `dataType` — określa typ danych zwracanych przez serwer.
- `timeout` — służy do określenia czasu oczekiwania na odpowiedź (w milisekundach), po którym skrypt zwróci błąd związany z przekroczeniem czasu odpowiedzi.

5.3.2. Wysyłanie żądania do serwera

W celu wysłania żądania do serwera korzystamy z dwóch metod obiektu `XMLHttpRequest`: `open()` i `send()`.

Metoda open()

Metoda `open()` przygotowuje żądanie, czyli konfiguruje jego parametry. Posiada trzy argumenty i ma postać:

```
open(type, url, async)
```

Argument `type` decyduje o tym, którą metodą protokołu HTTP będą wysyłane żądania. Do wyboru są metody: `POST` i `GET`. Można użyć dowolnej innej metody, jeżeli serwer ją obsługuje.

Argument `url` określa adres URL skryptu po stronie serwera. W praktyce może to być plik lokalny, np. *index.php* lub *default.aspx*. Może to być również strona w określonej lokalizacji. Format pliku może być dowolny.

Ze względu na bezpieczeństwo przesyłania danych nie można podać adresu strony znajdującej się w domenie zewnętrznej.

Argument `async` ma postać flagi i mówi o tym, czy żądanie powinno zostać wykonane w sposób asynchroniczny. Domyślnie ustawiana jest wartość `true`. Oznacza to, że wszystkie żądania są obsługiwane asynchronicznie.

W przypadku transmisji asynchronicznej nie ma potrzeby tworzenia funkcji dla właściwości `onreadystatechange`. Wystarczy po `send()` umieścić odpowiedni kod.

Przykład 5.3

```
xmlhttp.open("GET","info.asp",true);
xmlhttp.send();
document.getElementById("blok").innerHTML=xmlhttp.responseText;
```

Metoda send()

Metoda `send()` wysyła żądanie do serwera. Ma jeden argument określający dane dołączone do żądania. Jest on wykorzystywany tylko w przypadku metody `POST`. Dla metody `GET` przyjmuje wartość `null` i wtedy metoda najczęściej ma postać `send()`. Czasami spotyka się zapis `send(null)`.

Wysłanie żądania do serwera może mieć postać:

```
xmlhttp.open("GET", "info.asp",true);
xmlhttp.send();
```

Metody protokołu HTTP (GET, POST)

Metody protokołu HTTP określają sposób (metodę) przekazania informacji do wskazanego skryptu lub pod wskazany adres. Najczęściej przyjmują jedną z dwóch wartości: `POST` lub `GET`.

W metodzie `GET` dane są przekazywane za pomocą parametru `url`. Metoda ta powinna być stosowana, gdy pobierane z serwera dane nie ulegają zmianie.

W metodzie `POST` dane są dołączane na końcu żądania. Metoda ta powinna być stosowana, gdy znajdujące się na serwerze dane mogą być zmieniane przez użytkownika.

Częściej korzysta się z metody `GET`, ponieważ jest prostsza i szybsza od metody `POST`. Jednak są sytuacje, w których należy zastosować metodę `POST`. Należy jej użyć, gdy:

- do serwera wysyłane są duże ilości danych (metoda `POST` nie ma ograniczenia rozmiaru danych),
- wysyłane dane mogą zawierać informacje poufne (wtedy metoda `POST` jest bezpieczniejsza i bardziej niezawodna).

5.3.3. Funkcje zwrotne

Funkcje zwrotne służą do określenia działań, które będą wykonywane po otrzymaniu odpowiedzi z serwera.

Funkcja error

Funkcja ta jest wywoływana, gdy w komunikacji między skryptem i serwerem wystąpi błąd (np. przerwanie połączenia z serwerem). Jeżeli skrypt nie będzie mógł wysłać i odebrać informacji, to może wywołać funkcję `error()`. Ma ona trzy parametry:

```
error(XMLHttpRequest, textStatus, errorThrown)
```

Funkcja success

Funkcja ta jest wywoływana, gdy wysłane do serwera żądanie zakończy się sukcesem i zostaną przez serwer zwrócone dane. Dane mogą mieć postać określoną za pomocą opcji `dataType` i mogą być zwykłym tekstem, ale może to być również obiekt zawierający XML lub łańcuch JSON. Funkcja `success()` ma postać:

```
success(data,  textStatus, XMLHttpRequest)
```

Aby odebrać dane zwracane z serwera, należy zdefiniować funkcję, która zostanie wywołana po zakończeniu transmisji. Powinna ona zostać przypisana do właściwości `onreadystatechange` obiektu `XMLHttpRequest`, np. w postaci:

```
xmlhttp.onreadystatechange=function()
{
//tutaj wstawiamy kod funkcji
}
```

W treści funkcji powinno zostać sprawdzone, czy nadeszła odpowiedź z serwera i czy żądanie zostało poprawnie przetworzone przez serwer (na przykład po właściwości `status`).

Przykład 5.4

```
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("blok").innerHTML=xmlhttp.responseText;
    }
  }
```

Przykład 5.5

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function sprawdz()
```

```
{
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
 else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("blok").innerHTML=xmlhttp.responseText;
    }
  }
 xmlhttp.open("GET","inf.txt",true);
 xmlhttp.send();
 }
 </script>
 </head>
 <body>

 <h2>AJAX</h2>
 <button type="button" onclick="sprawdz()">'Wyświetl dane'</button>
 <div id="blok"></div>

 </body>
 </html>
```

W podanym przykładzie `xmlhttp.onreadystatechange=function()` to funkcja nasłuchująca, która będzie wywoływana po każdej zmianie statusu obsługi żądania i będzie sprawdzała stan żądania. Jeżeli właściwość `readyState==4` i `status==200`, to żądanie zostało obsłużone prawidłowo. Odpowiedź serwera w podanym przykładzie jest łańcuchem znaków dostępnym przez właściwość `responseText`.

5.3.4. Formaty przesyłanych danych

Zwykle należy przekazywać informacje na temat typu danych, które otrzymamy z serwera w odpowiedzi na wysłane żądanie. W niektórych przypadkach typ danych jest określany przez nazwę użytej metody. Najczęściej występujące formaty przesyłanych danych to `text`, `xml`, `script`, `JSON`. W technologii Ajax dane powinny być kodowane za pomocą UTF-8.

- `text` — po wybraniu formatu `text` dane są przesyłane w postaci prostych łańcuchów.
- `xml` — format ten pozwala na przesyłanie danych w zdefiniowanym przez użytkownika schemacie XML.
- `script` — format ten służy do dodawania na stronę nowego skryptu.
- `JSON` (ang. *Java Script Object Notation*) — jest formatem tekstowym niezależnym od języka programowania. Dane przesyłane w tym formacie mogą być łańcuchami, tablicami lub obiektami. Dane są pobierane z serwera jako tekst, a następnie przekształcane w obiekt.

PYTANIA KONTROLNE

1. Do czego służy technologia Ajax?
2. Jakie zadania w technologii Ajax wykonuje obiekt `XMLHttpRequest`?
3. Wymień podstawowe właściwości obiektu `XMLHttpRequest`.
4. Jakie metody są wykorzystywane podczas wysyłania żądania do serwera?

6

Serwery WWW

Serwery WWW (ang. *web server*) to programy, których zadaniem jest obsługa żądań protokołu komunikacyjnego HTTP. Przeważnie serwery WWW instalowane są na dedykowanych komputerach spełniających rolę serwerów internetowych. Można również instalować serwer WWW na stacji roboczej (lokalnie) w celu np. testowania tworzonych aplikacji. Aby pobrać określoną stronę internetową, klient łączy się z serwerem stron WWW za pośrednictwem przeglądarki internetowej. W celu udostępnienia dynamicznie utworzonych stron internetowych oraz danych przechowywanych w bazach danych serwer WWW może korzystać z dodatkowego oprogramowania, na przykład z PHP (i za jego pomocą z MySQL).

Podstawowe funkcje realizowane przez serwer WWW to:

- przyjęcie połączenia TCP,
- dekodowanie żądań protokołu HTML,
- realizacja żądania (przesłanie zasobu, wykonanie programu),
- obudowanie dokumentu nagłówkami HTTP,
- wysłanie danych do klienta.

Najpopularniejszym serwerem WWW jest Apache, który obsługuje około 60% dostępnych stron internetowych. Drugie miejsce zajmuje IIS firmy Microsoft z wynikiem około 15% dostępnych stron internetowych.

6.1. Serwer Apache

Apache jest najpopularniejszym i bezpłatnym serwerem WWW dostępnym dla wielu systemów operacyjnych (Linux, Windows, Mac OS). Razem z interpreterem języka skryptowego PHP i bazą danych MySQL stanowi jedno z najpopularniejszych środowisk — tak zwaną platformę AMP.

Najważniejsze cechy serwera Apache to:

- wielowątkowość,
- skalowalność,

- bezpieczeństwo,
- kontrola dostępu (uwierzytelnianie),
- uruchamianie witryn internetowych generowanych dynamicznie,
- niezależność od oprogramowania i sprzętu,
- dostępność jako open source.

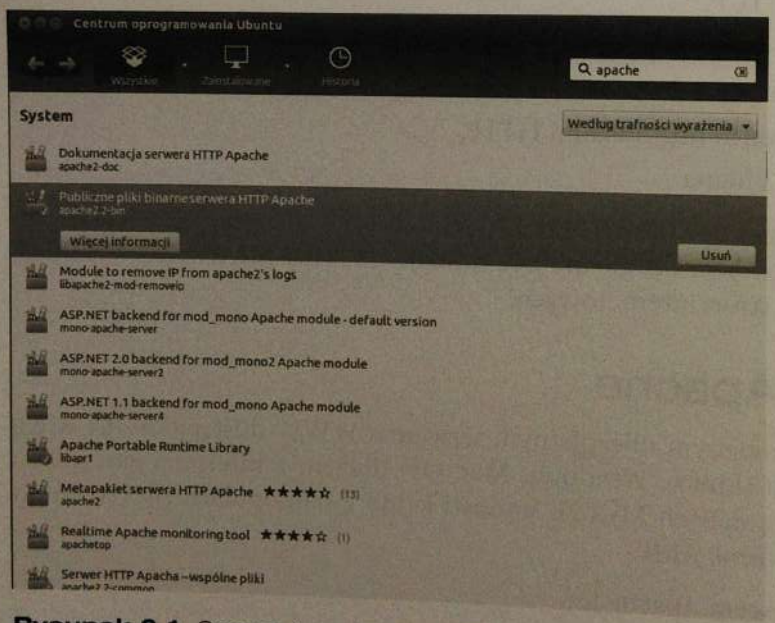
Serwer Apache jest podzielony na wiele modułów. Podczas instalowania serwera można wybrać tylko potrzebne moduły lub zainstalować wszystkie.

6.1.1. Instalowanie serwera w systemach Linux

W większości dystrybucji Linuksa potrzebne do zainstalowania serwera narzędzia są dostępne razem z wersją instalacyjną systemu. Najczęściej są one również instalowane podczas instalowania systemu. Poniższy przykład instalacji dotyczy systemu Ubuntu 12.04.2. Trzeba pamiętać, że dla innych dystrybucji sposób instalacji może się mniej lub bardziej różnić.

Instalowanie serwera

Aby sprawdzić, czy serwer Apache został zainstalowany w systemie, należy uruchomić *Centrum oprogramowania Ubuntu*, wybrać *System* i w polu wyszukiwania wpisać *apache*. W wynikach wyszukiwania sprawdzamy stan pakietu *Publiczne pliki binarne serwera HTTP Apache*.



Rysunek 6.1. Serwer Apache został zainstalowany w systemie Ubuntu

Jeśli pakiet jest zaznaczony, jak na rysunku 6.1, serwer WWW jest już zainstalowany. Jeśli zamiast przycisku *Usuń* jest przycisk *Instaluj* (co się rzadko zdarza, ponieważ Apache zazwyczaj jest domyślnie instalowany w większości dystrybucji), trzeba pakiet doinstalować.

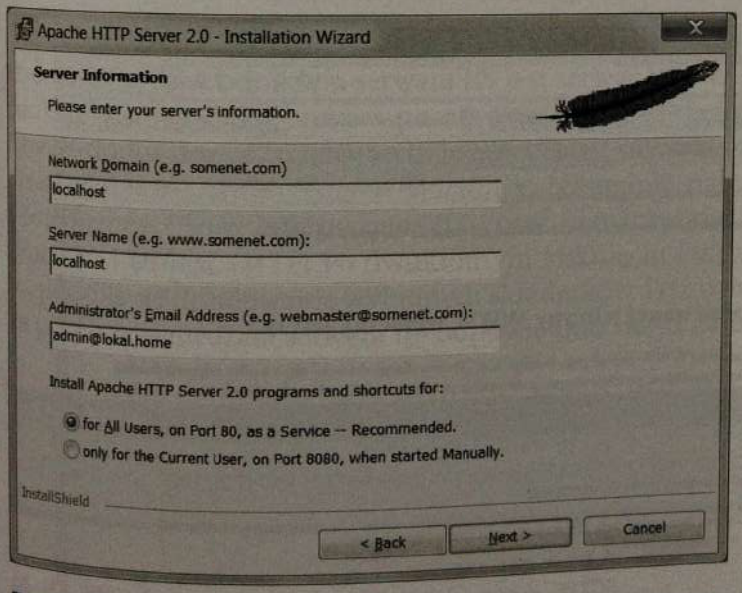
Jeżeli serwer został zainstalowany lokalnie, w celu sprawdzenia jego działania należy uruchomić dowolną przeglądarkę internetową i w polu adresu wpisać adres: *http://localhost/* lub *http://127.0.0.1/*. W oknie przeglądarki pojawi się ekran powitalny z testową stroną serwera WWW.

6.1.2. Instalowanie serwera w systemie Windows

Aby zainstalować serwer Apache w systemie Windows, należy pobrać jego wersję instalacyjną ze strony *http://www.apache.org*. Po wybraniu opcji *Download* należy odszukać najnowszy plik instalacyjny dla systemu Windows (np. *httpd-2.0.64-win32-x86-no_ssl_msi*) i pobrać go.

Instalowanie serwera

Instalacja serwera Apache rozpoczyna się od uruchomienia pobranego pliku. Przez proces instalacji poprowadzi instalator Windows. Należy zaakceptować licencję i wybrać katalog docelowy instalacji. Okno konfiguracyjne umożliwia zmianę nazwy domeny i serwera. Można pozostawić domyślne ustawienia lub gdy serwer ma działać lokalnie, wpisać jako adres hosta i domenę *localhost* (rysunek 6.2). Opcja *for All Users* powinna pozostać zaznaczona, ponieważ serwer Apache będzie działał na porcie 80.

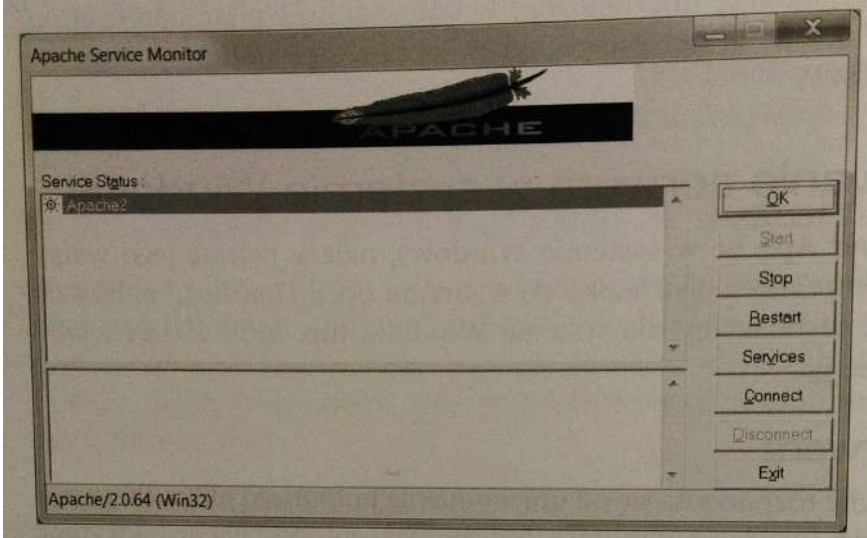


Rysunek 6.2. Wstępne ustawienia konfiguracyjne serwera Apache

W kolejnych oknach należy wybrać typ instalacji (*Typical*) oraz ścieżkę instalacji. Po kliknięciu przycisku *Install* serwer zostanie zainstalowany.

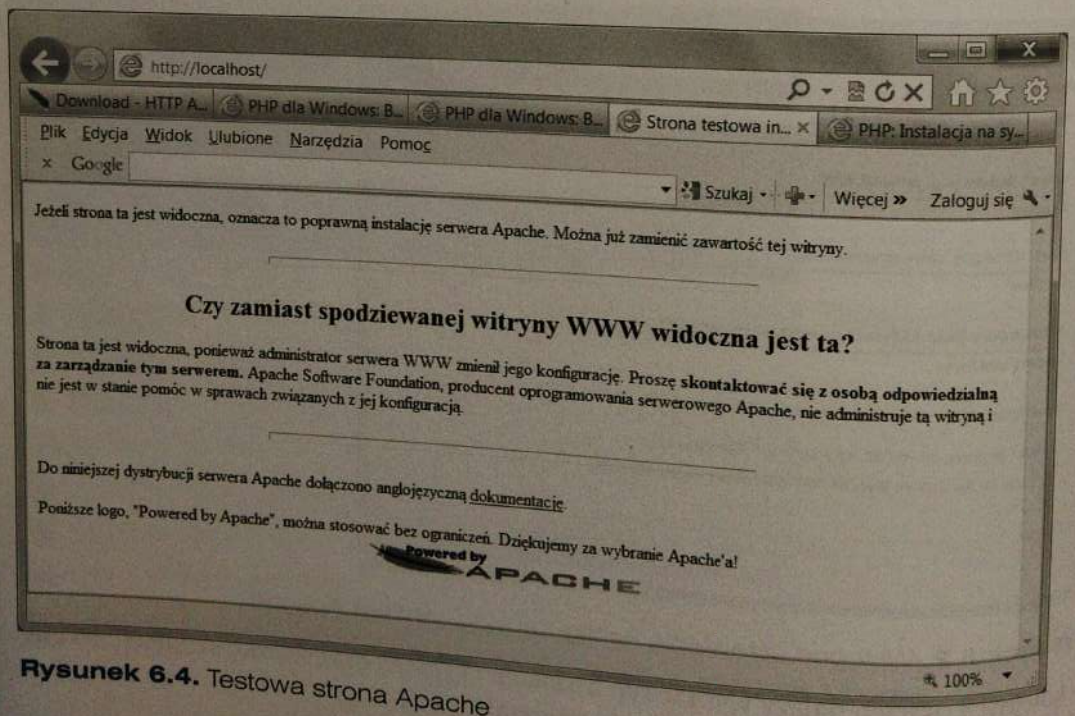
Uruchomienie serwera

Przy każdym uruchomieniu systemu serwer Apache będzie uruchamiany automatycznie. Na pasku zadań znajduje się ikona serwera informująca o jego stanie. Gdy klikniemy ją prawym przyciskiem myszy, uruchomiony zostanie *Apache Service Monitor*, który pozwala na zatrzymanie i ponowne uruchomienie serwera (rysunek 6.3).



Rysunek 6.3. Apache Service Monitor

Aby sprawdzić działanie serwera, należy uruchomić dowolną przeglądarkę internetową i w polu adresu wpisać adres: *http://localhost/* lub *http://127.0.0.1/*. W oknie przeglądarki pojawi się ekran powitalny z testową stroną serwera WWW (rysunek 6.4).



Rysunek 6.4. Testowa strona Apache

6.2. Serwer IIS

Serwer IIS (ang. *Internet Information Services*) jest komercyjnym serwerem WWW dostępnym dla systemów Windows Server i Windows. Wraz z serwerem baz danych MS SQL i środowiskiem programistycznym Visual Web Developer tworzy platformę WEB.PI.

Na serwerze IIS oparta została platforma ASP.NET (ASP — ang. *Active Server Pages*). Jest to technologia firmy Microsoft służąca do tworzenia dynamicznych stron WWW wykonywanych po stronie serwera.

Podstawowe cechy serwera IIS:

- serwer oraz dostępne na nim witryny internetowe działają w dwóch odseparowanych obszarach. W związku z tym źle działająca witryna nie wpływa na pracę serwera,
- witryny internetowe są izolowane i działają niezależnie od serwera IIS oraz od siebie na wzajem.

6.2.1. Instalowanie serwera IIS

Do instalowania serwera IIS można wykorzystać Kreator konfigurowania serwera. Aby uruchomić kreator, trzeba wybrać *Start/Zarządzanie tym serwerem/Dodaj lub usuń rolę/Dalej*, na liście ról zaznaczyć *Serwer aplikacji* (IIS, ASP.NET) i kliknąć przycisk *Dalej*. W następnym oknie, *Opcje serwera aplikacji*, można wybrać dodatkowe składniki serwera, które powinny zostać zainstalowane. Są to rozszerzenia serwera WWW oraz ASP.NET. ASP.NET to składnik, który powinien zostać zainstalowany, ponieważ upraszcza tworzenie dynamicznych witryn internetowych. Po kliknięciu przycisku *Dalej* rozpocznie się instalowanie serwera.

Podstawowym składnikiem serwera IIS jest serwer WWW, ale może on również pełnić funkcję serwera FTP i serwera pocztowego. Funkcje te nie są instalowane domyślnie. Aby je doinstalować, trzeba wybrać *Start/Panel sterowania/Dodaj lub usuń programy/Dodaj/Usuń składniki systemu Windows*. Następnie należy wybrać *Serwer aplikacji* i *Internetowe Usługi Informacyjne* (IIS) oraz zaznaczyć opcje *Usługa FTP* (*File Transfer Protocol*) i *Usługa SMTP*. Po dwukrotnym kliknięciu *OK*, a następnie przycisku *Dalej* rozpocznie się instalowanie wybranych składników. Po zakończeniu instalowania należy za pomocą menedżera serwera IIS odpowiednio skonfigurować usługi FTP oraz SMTP.

UWAGA

Powyższy opis dotyczy systemów Windows z rodziny Server (czyli 2003, 2008, 2012 itp.).

6.2.2. Menedżer serwera IIS

Do zarządzania serwerem IIS służy Menedżer serwera IIS, który może zostać uruchomiony poprzez panel *Narzędzia administracyjne*. Gdy jest uruchomiony, w folderze z nazwą

komputera dostępne są podfoldery *Pule aplikacji*, *Witryny sieci Web* oraz *Rozszerzenia usługi sieci Web*. Jeżeli zostały zainstalowane usługi FTP i SMTP, znajdują się w nim również foldery *Witryny FTP* i *Domyślny serwer wirtualny SMTP*.

Aby utworzyć nową witrynę, należy wskazać pozycję *Witryny sieci Web* i z menu *Akcja* wybrać *Nowy/Witryna sieci Web*, wpisać nazwę strony, np. *Moja strona*, i kliknąć przycisk *Dalej*. Adres IP można pozostawić (*Wszystkie nieprzypisane*). Numer portu dla usługi WWW to numer 80 (na serwerze działa domyślna witryna sieci Web, która nasłuchuje na porcie 80). W kolejnym oknie trzeba zdefiniować ścieżkę do katalogu głównego witryny i zatwierdzić domyślnie zaznaczoną opcję *Zezwalaj na anonimowy dostęp do witryny sieci Web* oraz domyślne uprawnienia *Odczyt* i *Uruchamianie skryptów*. Po kliknięciu przycisku *Dalej* kreator zakończy pracę.

Utworzoną witrynę można przypisać do wybranej puli aplikacji. W tym celu należy w folderze *Witryny sieci Web* zaznaczyć nazwę witryny i wybrać z menu *Akcja* opcję *Właściwości*. Następnie na karcie *Katalog macierzysty* trzeba zdefiniować *Nazwę aplikacji*, ustawić *Uprawnienia wykonywania* i wybrać *Pulę aplikacji*. Jeżeli witryna będzie umieszczona w nowej puli, należy tę pulę wcześniej utworzyć. W tym celu trzeba wybrać folder *Pule aplikacji*, następnie z menu *Akcja* wybrać *Nowy/Pula aplikacji* i zaakceptować domyślne ustawienia (można również skopiować ustawienia z innej puli aplikacji).

6.2.3. Szyfrowanie i certyfikaty

Na jakość pracy serwera ma wpływ zabezpieczenie samego serwera oraz zapewnienie poufności przesyłanych informacji. Za pomocą karty *Zabezpieczenia katalogów* można skonfigurować zabezpieczenia korzystania z usług serwera, takie jak: kontrola dostępu, określenie metody uwierzytelniania, ograniczenie zakresu adresów IP komputerów, z których można nawiązywać połączenia z serwerem, bezpieczne połączenia SSL, nadanie serwerowi certyfikatu.

PYTANIA KONTROLNE

1. Wymień podstawowe funkcje realizowane przez serwer WWW.
2. Wymień popularne serwery WWW.
3. Omów podstawowe cechy serwera Apache.
4. Omów podstawowe cechy serwera IIS.

7

Język PHP

7.1. Wprowadzenie

PHP (ang. *Hypertext Preprocessor*) to skryptowy język programowania najczęściej wykorzystywany do tworzenia aplikacji internetowych. Skrypty pisane w PHP są umieszczane w kodzie HTML i wykonywane po stronie serwera. Klient otrzymuje tylko wynik wykonania skryptu.

Oprogramowanie PHP umożliwia przetwarzanie danych z formularzy, dynamiczne generowanie zawartości stron internetowych, wysyłanie i odbieranie cookies. Pozwala na dynamiczne tworzenie obrazów, dokumentów PDF, animacji Flash. Obsługuje wiele protokołów sieciowych, w tym SMTP, POP3, IMAP, NNTP. Umożliwia wykorzystanie do komunikacji sieciowej gniazd. Współpracuje z większością dostępnych systemów operacyjnych, w tym z różnymi wersjami Linuksa, systemem Windows i Mac OS oraz różnymi systemami baz danych. Obsługuje wiele serwerów HTTP (Apache, IIS).

PHP jest rozpowszechniany na zasadach open source, w związku z tym dostępny jest jego pełny kod źródłowy. Istnieją również liczne rozszerzenia i narzędzia, z których można korzystać, zwiększając jego możliwości.

Wyróżnia się trzy główne obszary, w których używany jest język PHP. Są to:

- Skrypty po stronie serwera — jest to główne zastosowanie PHP. Aby tworzyć skrypty potrzebny jest parser PHP, serwer WWW oraz przeglądarka internetowa.
- Skrypty wywoływane z wiersza poleceń — skrypty są tworzone z wykorzystaniem parsera PHP i wywoływane w linii poleceń. Mogą być wykorzystane również do prostego przetwarzania tekstu.
- Aplikacje po stronie klienta — zaawansowane funkcje PHP umożliwiające pisanie aplikacji desktopowych.

Kod zapisany w skryptach PHP jest przetwarzany za pomocą silnika Zend. Nadzoruje on wykonywanie wszystkich operacji dotyczących przetwarzania kodu PHP. PHP integruje się z serwerem WWW jako jego moduł lub skrypt CGI. Serwerem WWW najczęściej jest serwer Apache lub serwer IIS.

7.1.1. Opis języka

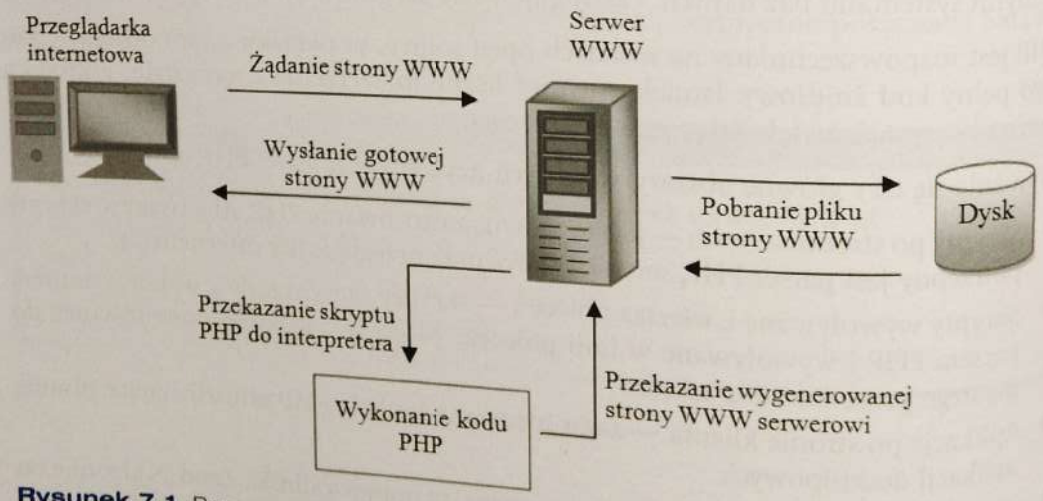
PHP jest skryptowym językiem programowania, za pomocą którego można tworzyć aplikacje WWW. Używając PHP, można:

- generować dynamicznie zawartość strony internetowej,
- tworzyć i edytować pliki na serwerze,
- pobierać dane z formularzy,
- odbierać i wysyłać cookies,
- wykonywać operacje na bazie danych,
- sterować dostępem użytkowników do stron witryny internetowej,
- szyfrować dane.

7.1.2. PHP w HTML

Skrypty pisane w języku PHP są umieszczane wewnątrz dokumentów HTML. Umownie dokumenty HTML zawierające osadzony kod PHP nazywa się skryptami.

Strony internetowe wygenerowane na podstawie skryptu PHP są zarządzane przez serwer WWW. Gdy w przeglądarce ma zostać wyświetlona taka strona, serwer WWW pobiera plik zawierający żądany skrypt i przekazuje go do interpretera PHP. Interpreter odnajduje w pliku skrypt PHP (jest on specjalnie oznaczony w kodzie HTML) i próbuje go wykonać. Wykonany skrypt jest zwracany do serwera WWW w postaci kodu HTML. Cała zawartość strony internetowej w kodzie HTML jest wysyłana do przeglądarki internetowej. Proces przetwarzania kodu PHP został pokazany na rysunku 7.1.



Rysunek 7.1. Proces przetwarzania kodu PHP

7.2. Instalowanie PHP

Przed rozpoczęciem pracy z językiem PHP należy zainstalować i skonfigurować parser PHP. Do pracy niezbędna jest również instalacja serwera WWW. Popularnym serwerem stron internetowych, który współpracuje z PHP, jest Apache. Jest on dostępny w wersji dla systemów Linux, dla systemu Windows oraz wielu innych.

Zamiast samodzielnie instalować wszystkie niezbędne narzędzia, można zainstalować w systemie Windows gotowe środowisko WAMP zawierające serwer stron internetowych Apache, bazę danych MySQL oraz język skryptowy PHP. Natomiast dla platformy Linux dostępne jest środowisko LAMP (Linux, Apache, MySQL, PHP). XAMPP jest środowiskiem dostępnym na wyżej wymienione systemy i nie tylko te. Pomimo że żaden z elementów wchodzących w skład podanych środowisk nie został zaprojektowany specjalnie do współpracy z pozostałymi, to środowiska WAMP, XAMPP oraz LAMP są bardzo popularne ze względu na dostępność, łatwość instalowania oraz niewielki koszt.

Oprogramowanie PHP może zostać zainstalowane jako moduł serwera WWW lub jako skrypt CGI. Jeżeli PHP jest instalowany jako moduł, należy wziąć pod uwagę rodzaj oraz wersję serwera, ponieważ dla każdej wersji występuje inny moduł PHP. Przy instalowaniu PHP jako skryptu CGI występuje większa niezależność od stosowanego serwera. Ponieważ PHP zainstalowany jako moduł pozwala na szybkie i bezpieczne wykonywanie kodu strony, obecnie głównie stosowana jest ta opcja jego instalowania.

7.2.1. Instalowanie PHP w systemach Linux

Serwer WWW Apache najczęściej jest instalowany razem z systemem operacyjnym. Jeżeli serwer stron internetowych został zainstalowany, wystarczy zainstalować pakiet PHP. Poniższy przykład instalacji dotyczy systemu Ubuntu 12.04.2. Trzeba pamiętać, że w przypadku innych dystrybucji w sposobie ich instalacji mogą występować mniejsze lub większe różnice.

Instalacja PHP

Oprogramowanie dla PHP nie jest standardowo instalowane w systemach Linux. Dlatego należy uruchomić *Centrum oprogramowania Ubuntu*, wybrać *System* i w polu wyszukiwania wpisać *apache php*. Następnie wybieramy w wyszukanych elementach *Server-side, HTML-embedded scripting language (Apache 2 module)*.

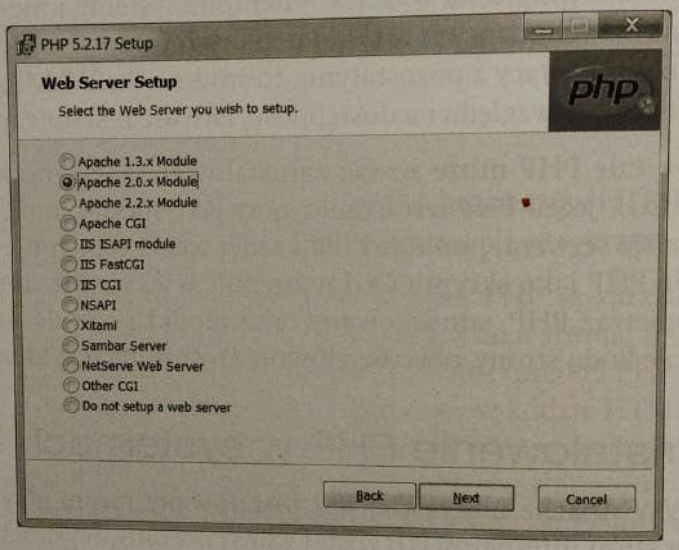
Po zaznaczeniu należy kliknąć przycisk *Zainstaluj*. Wyświetli się pełen opis instalowanego pakietu oraz pasek postępu instalacji. Po zainstalowaniu oprogramowania należy przystąpić do jego konfiguracji.

7.2.2. Instalowanie PHP w systemie Windows

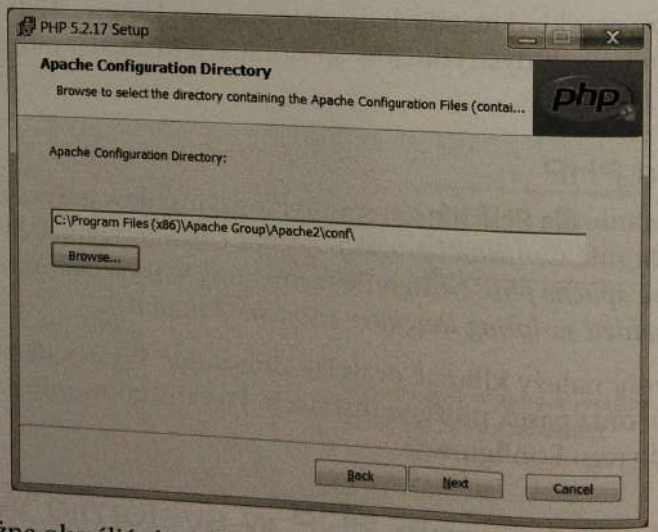
Po zainstalowaniu serwera stron internetowych można przystąpić do instalowania pakietu PHP.

Instalacja PHP

Po uruchomieniu pliku instalacyjnego należy zaakceptować licencję i wybrać katalog docelowy instalacji. W następnym oknie można wybrać serwer WWW, z którym będzie współpracował PHP (rysunek 7.2), może to być np. opcja *Apache 2.0*. Aby instalator mógł połączyć PHP z serwerem Apache, w kolejnym oknie należy wskazać katalog zawierający plik konfiguracyjny serwera. Jest to katalog *conf* znajdujący się w katalogu, w którym został zainstalowany serwer WWW (rysunek 7.3). W dalszym etapie instalacji należy wybrać składniki pakietu, które powinny zostać zainstalowane. Można zostawić ustawienia domyślne. Po zakończeniu instalacji należy zrestartować serwer Apache, aby zostały w nim zapisane wprowadzone przez instalator PHP zmiany konfiguracyjne.



Rysunek 7.2. Instalowanie PHP. Wybór serwera WWW



Rysunek 7.3. Instalowanie PHP. Wybór ścieżki do pliku konfiguracyjnego serwera

W kolejnym oknie można określić elementy pakietu PHP, które powinny zostać zainstalowane, i klikając przycisk *Next*, przejść do instalowania pakietu. Po jego zainstalowaniu należy zrestartować serwer Apache, aby zostały wprowadzone zmiany konfiguracyjne dokonane podczas instalacji PHP. Po ponownym uruchomieniu serwera można

sprawdzić, czy działa on prawidłowo, wpisując w dowolnej przeglądarce adres *http://localhost/*. Następnie można sprawdzić, jak współpracuje on z PHP. W tym celu w katalogu, w którym został zainstalowany serwer Apache, w podkatalogu *htdocs*, należy utworzyć plik tekstowy *index.php* z treścią:

```
<?php echo „Test PHP"; ?>
```

Gdy wpiszemy w przeglądarce adres *http://localhost/index.php*, na ekranie powinien się wyświetlić tekst *Test PHP*.

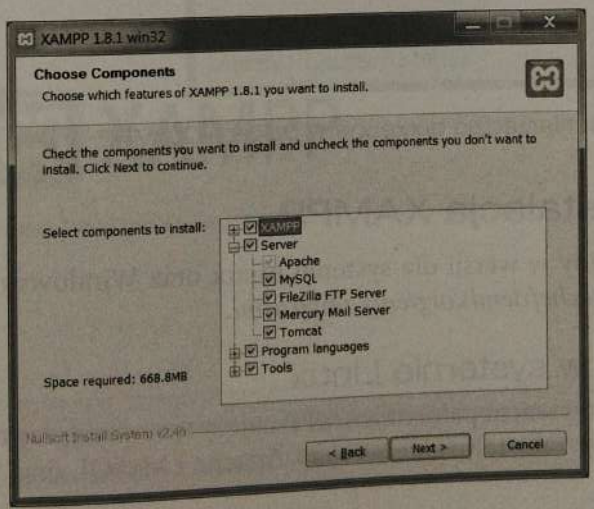
Zdarza się, że pomimo iż instalacje serwera Apache i PHP zostały wykonane prawidłowo, nie współpracują one ze sobą. Najczęściej spotykane problemy to: brak połączenia z serwerem lub niemożność uruchomienia serwera, serwer przestał działać po instalacji PHP, serwer działa, ale nie obsługuje PHP. Aby wyeliminować pojawiające się nieprawidłowości, trzeba ręcznie dokonywać zmian w plikach konfiguracyjnych zainstalowanego oprogramowania, ustawieniach systemu operacyjnego i serwera WWW oraz sprawdzać, czy oprogramowanie PHP współpracuje z daną wersją serwera. Jeżeli zatem będziemy używać standardowych ustawień serwera Apache i oprogramowania PHP oraz wykorzystywać bazę danych MySQL, wygodniejszym rozwiązaniem może okazać się instalacja pakietu XAMPP.

UWAGA

Na oficjalnej stronie PHP pod adresem *http://php.net/manual/en/install.php* znajduje się instrukcja instalacji i konfiguracji oprogramowania PHP.

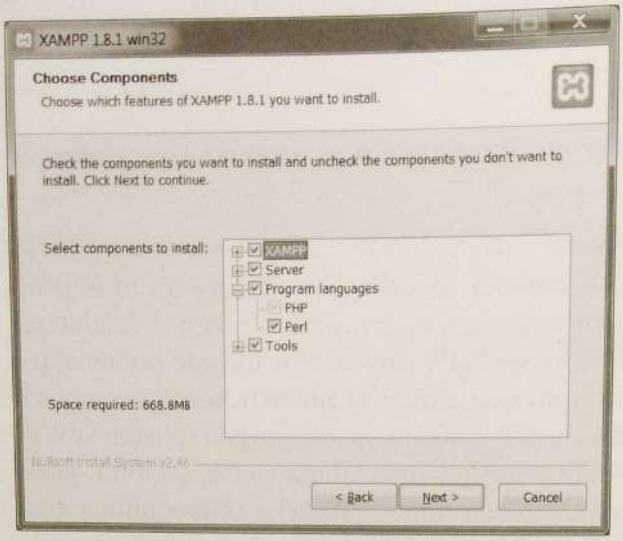
7.3. Pakiet XAMPP

XAMPP jest pakietem zawierającym między innymi serwer stron internetowych Apache oraz serwer baz danych MySQL (rysunek 7.4).

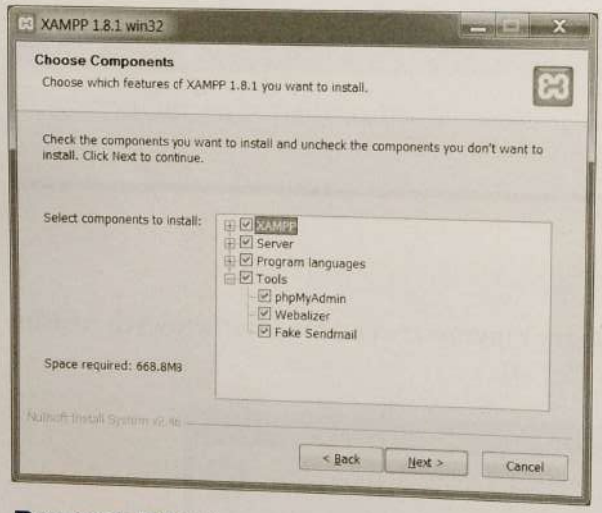


Rysunek 7.4. Serwery dostępne w pakiecie XAMPP

Poza językiem skryptowym PHP jest w nim dostępny język Perl (rysunek 7.5) oraz zestaw narzędzi, między innymi phpMyAdmin (rysunek 7.6).



Rysunek 7.5. Języki dostępne w pakiecie XAMPP



Rysunek 7.6. Narzędzia dostępne w pakiecie XAMPP

7.3.1. Instalacja XAMPP

Plik instalacyjny w wersji dla systemu Linux oraz Windows można pobrać ze strony *http://www.apachefriends.org/en/xampp.html*.

Instalacja w systemie Linux

Przed zainstalowaniem pakietu XAMPP należy sprawdzić, czy dla systemu Linux wcześniej nie zostały zainstalowane serwery Apache i MySQL oraz język PHP. Po pobraniu

pliku instalacyjnego (np. *xampp-linux-1.8.2.installer.run*) trzeba z konsoli wpisać polecenie:

```
chmod 755 xampp-linux-1.8.2-installer.run && ./xampp-linux-1.8.2-installer.run
```

W wyniku wykonania podanego polecenia nastąpi wypakowanie plików instalacyjnych pakietu do folderu */opt/lampp*. Po tych czynnościach pakiet jest przygotowany do pracy. Aby uruchomić serwery wchodzące w skład pakietu, należy z konsoli wpisać polecenie:

```
sudo /opt/lampp/lampp start
```

Po wykonaniu polecenia zostaną uruchomione serwery, co zostanie zakomunikowane wyświetleniem odpowiednich informacji na ekranie.

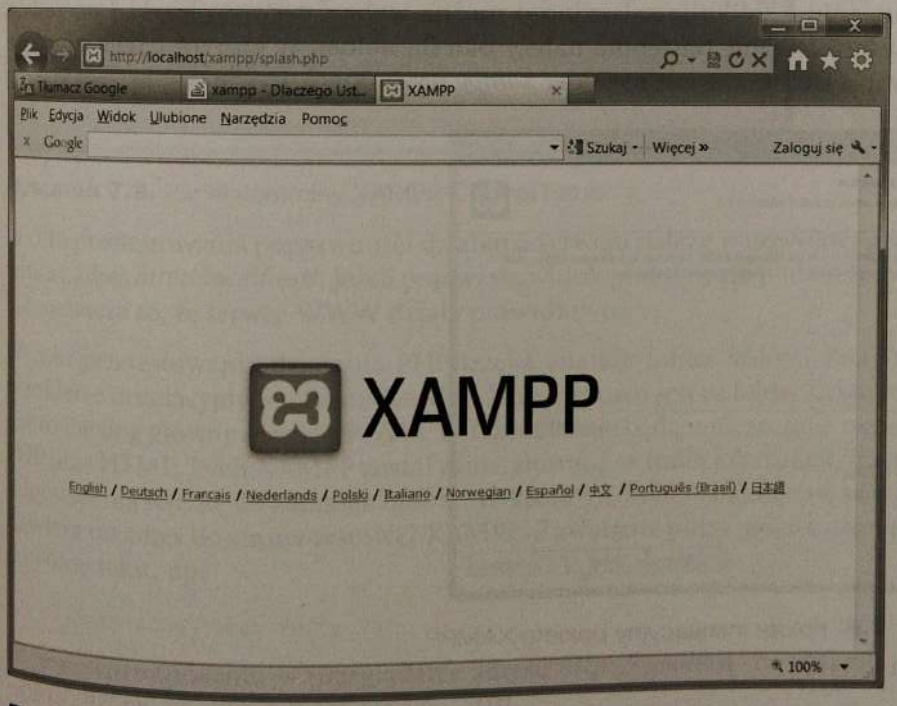
Serwery można zatrzymać poleceniem:

```
sudo /opt/lampp/lampp stop
```

a zrestartować poleceniem:

```
sudo /opt/lampp/lampp restart
```

Poprawność instalacji pakietu i uruchomienia serwerów można sprawdzić, wpisując w dowolną przeglądarkę internetową adres *http://localhost/*. Nastąpi przekierowanie na adres *http://localhost/xampp/splash.php* i pojawi się widok podobny do pokazanego na rysunku 7.7, co oznacza, że serwer WWW działa prawidłowo.



Rysunek 7.7. Test działania pakietu XAMPP

W pakiecie LAMPP głównym katalogiem serwera WWW jest */opt/lampp/htdocs/*. W tym katalogu powinny być umieszczane tworzone pliki PHP oraz pliki HTML.

Ponieważ domyślnie zasoby pakietu XAMPP są dostępne dla każdego, należy zabezpieczyć je, tworząc konta użytkowników i definiując hasła dostępu. W tym celu z konsoli trzeba wpisać polecenie:

```
sudo /opt/lampp/lampp security
```

W ramach konfigurowania zabezpieczeń trzeba będzie zdefiniować oddzielnie hasła dostępu do:

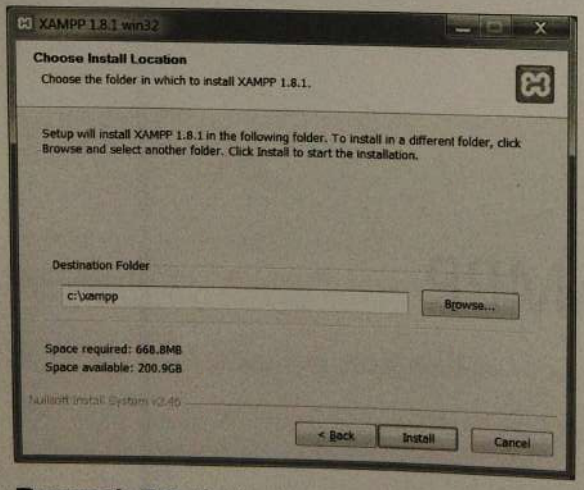
- panelu konfiguracyjnego pakietu XAMPP (dostępny pod adresem *http://localhost/xampp/*, a konto użytkownika to *lampp*),
- oprogramowania phpMyAdmin (zarządzanie bazą danych przez serwer WWW),
- konta administratora bazy danych MySQL,
- dostępu do serwera FTP.

Po tych działaniach elementy składowe XAMPP będą w pełni zabezpieczone.

Instalacja w systemie Windows

Plik instalacyjny dla systemu Windows jest dostępny w trzech wersjach. Najprostsza w instalacji jest wersja z kreatorem instalacji.

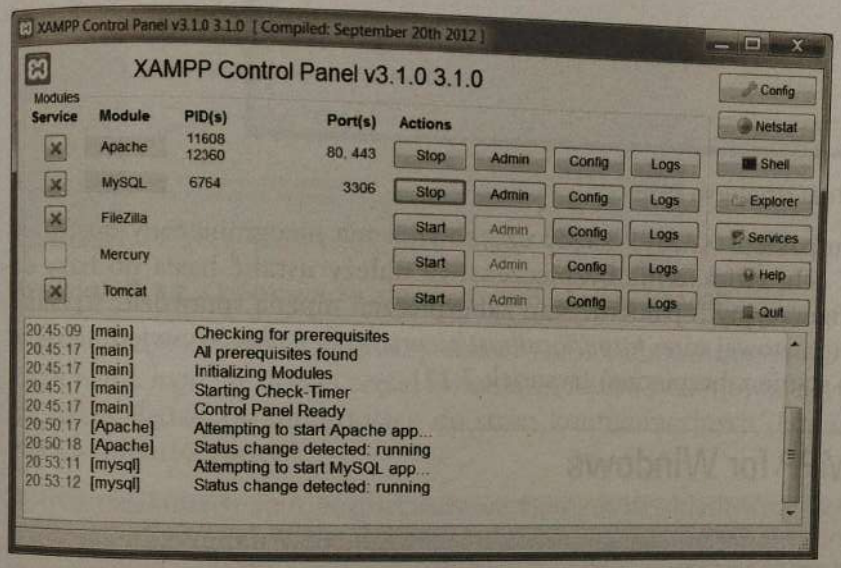
Po uruchomieniu 1. pliku instalacyjnego można wybrać serwery, języki i narzędzia, które będą instalowane. Następnie należy określić folder, w którym pakiet zostanie zainstalowany. Domyślnie jest to folder *xampp* na dysku *C:* (rysunek 7.8).



Rysunek 7.8. Folder instalacyjny pakietu XAMPP

Po wskazaniu folderu docelowego nastąpi wypakowanie plików instalacyjnych i zainstalowanie pakietu. Po zakończeniu instalowania pojawi się pytanie, czy chcemy uruchomić panel kontrolny XAMPP Control Panel, którego zadaniem jest skonfigurowanie zainstalowanego oprogramowania. W panelu tym dostępne są (rysunek 7.9):

- serwer WWW Apache z interpreterem skryptów w PHP i Perl,
- serwer bazy danych MySQL,
- serwer FTP FileZilla,
- serwer poczty elektronicznej Mercury,
- środowisko serwera dla Javy (Java Server Pages) — Java Tomcat.



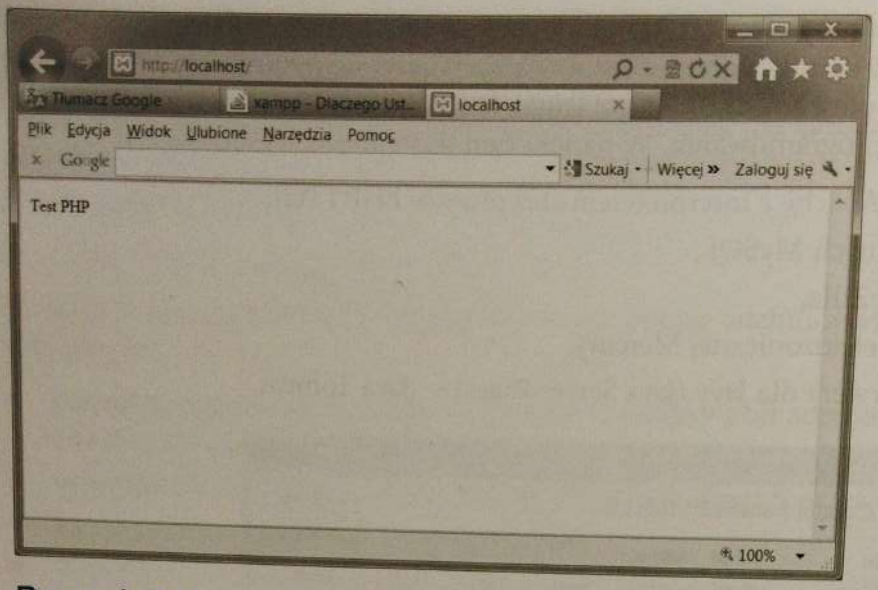
Rysunek 7.9. Panel kontrolny XAMPP Control Panel

W celu przetestowania poprawności działania serwera należy w dowolnej przeglądarce wpisać adres *http://localhost/*. Jeżeli pojawi się widok podobny do podanego na rysunku 7.7, oznacza to, że serwer WWW działa prawidłowo.

W celu przetestowania działania PHP trzeba znaleźć folder *htdocs*. Znajduje się on w folderze instalacyjnym pakietu XAMPP. Standardowo jest to folder *C:\xampp\htdocs*. Jest to katalog główny serwera WWW. W tym folderze będą umieszczane tworzone pliki PHP oraz HTML. Jeżeli XAMPP został zainstalowany w innej lokalizacji, trzeba znaleźć odpowiednią ścieżkę do katalogu *htdocs*. W katalogu *htdocs* znajduje się plik *index.php*. Zawiera on adres do strony testowej XAMPP. Zawartość pliku można usunąć i wpisać dowolny tekst, np.:

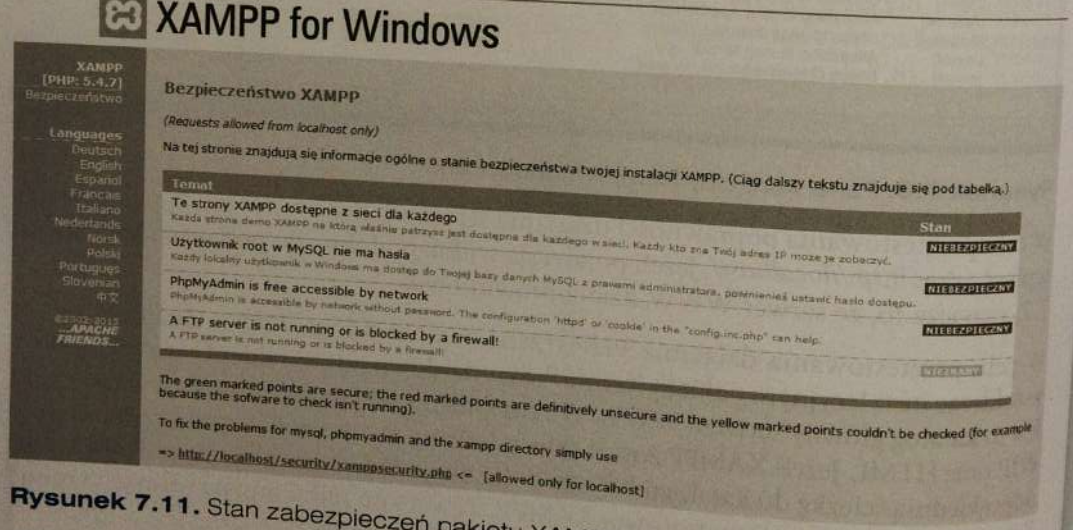
```
<?php echo "Test PHP"; ?>
```

Po ponownym podaniu w przeglądarce adresu *http://localhost/* na ekranie pojawi się wprowadzony do pliku tekst (rysunek 7.10).



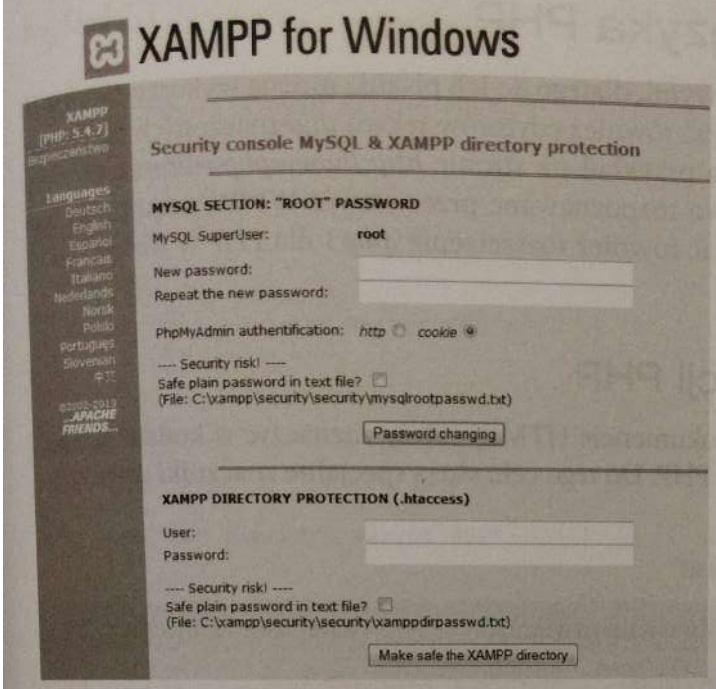
Rysunek 7.10. Test działania PHP

Po zainstalowaniu pakietu XAMPP każdy użytkownik ma nieograniczony dostęp do jego zasobów. W celu zwiększenia bezpieczeństwa należy ustalić hasła do bazy danych i stron konfiguracyjnych pakietu. Stan zabezpieczeń można sprawdzić, wpisując w przeglądarce internetowej adres *http://localhost/security/index.php*. Wyświetli się okno z informacjami o stanie zabezpieczeń (rysunek 7.11).



Rysunek 7.11. Stan zabezpieczeń pakietu XAMPP

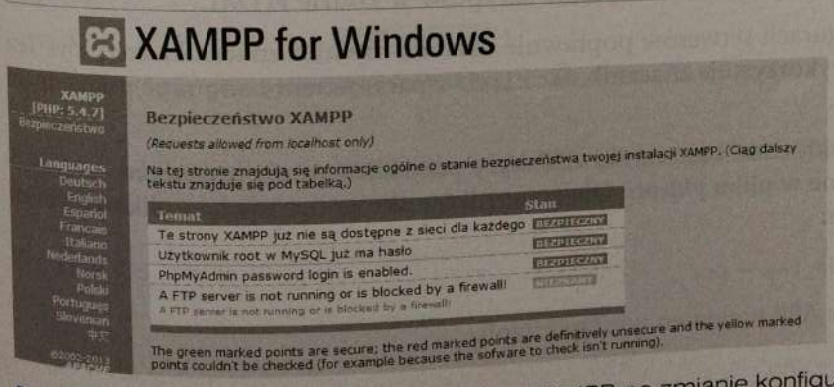
Zabezpieczenia można definiować, uruchamiając stronę o adresie *http://localhost/security/xamppsecurity.php*. Wyświetli się formularz pokazany na rysunku 7.12.



Rysunek 7.12. Definiowanie zabezpieczeń pakietu XAMPP

W górnej części formularza należy ustawić hasło administratora bazy danych (jest to użytkownik *root*). W dolnej części formularza trzeba zdefiniować użytkownika i nadać mu hasło. Będzie on miał dostęp do stron konfiguracyjnych (*http://localhost/xampp*, *http://localhost/security*).

Po tych działaniach stan bezpieczeństwa elementów składowych XAMPP będzie wyglądał jak na rysunku 7.13.



Rysunek 7.13. Stan zabezpieczeń pakietu XAMPP po zmianie konfiguracji

7.4. Struktura języka PHP

Skrypty PHP są plikami tekstowymi, dlatego do ich pisania można wykorzystać dowolny edytor tekstu. Można używać również edytorów tekstu przeznaczonych do pisania skryptów PHP, dostępnych na przykład na stronie *http://www.php-editors.com/*. Aby pliki tekstowe były prawidłowo rozpoznawane przez serwer WWW, muszą mieć rozszerzenie *.php*. Można stosować również rozszerzenie *.php3* dla PHP w wersji 3. oraz *.php4* dla PHP w wersji 4.

7.4.1. Blok instrukcji PHP

Umieszczając skrypt PHP w dokumencie HTML, należy zaznaczyć w kodzie, gdzie zaczynają się i kończą polecenia PHP. Do tego celu służą specjalne znaczniki otwierające i zamykające (tabela 7.1).

Tabela 7.1. Znaczniki bloku PHP

Rodzaj znacznika	Znacznik otwierający	Znacznik zamykający
Standardowy	`<?php`	`?>`
Skrócony	`<?`	`?>`
ASP	`<%`	`%>`
Skryptowy	`<script language="php">`	`</script>`

Konfiguracje serwerów są różne, zatem jeżeli tworzony jest kod, który będzie przenoszony między serwerami, najbezpieczniejsze jest używanie znaczników standardowych. Jest to też zalecany sposób umieszczania skryptów w kodzie HTML.

W każdej konfiguracji serwerów poprawnie działają również znaczniki skryptów. Ten sposób zapisu wykorzystuje znacznik `<script>` z parametrem `language` ustawionym na wartość `php`.

Aby znaczniki skrócone i znaczniki ASP były prawidłowo rozpoznawane, muszą zostać uaktywnione w pliku *php.ini*. Dla znaczników skróconych w tym pliku powinien znaleźć się zapis:

```
short_open_tag = On;
```

Dla znaczników ASP powinien wystąpić zapis:

```
asp_tags = On;
```

Gdy te wpisy są umieszczone w pliku konfiguracyjnym, w skryptach można wykorzystywać każdy z podanych rodzajów znaczników.

Przykład 7.1

```
<?
echo "Mój pierwszy skrypt PHP";
?>
```

Przykład 7.2

```
<?php
echo "Mój pierwszy skrypt PHP";
?>
```

Przykład 7.3

```
<%
echo "Mój pierwszy skrypt PHP";
%>
```

Przykład 7.4

```
<script language="php">
echo "Mój pierwszy skrypt PHP";
</script>
```

Użyta w przykładach instrukcja `echo` służy do wyświetlania danych. Instrukcja `echo` może być stosowana w podany wyżej sposób lub z użyciem nawiasów okrągłych. Obydwa sposoby jej zapisu mają takie samo znaczenie. Natomiast wyświetlany ciąg znaków zawsze musi być ujęty w cudzysłowy lub apostrofy.

Przykład 7.5

```
<?
echo ("Mój pierwszy skrypt PHP");
?>
```

Tak samo jak instrukcja `echo` działa instrukcja `print()`. Może być ona używana wymiennie z instrukcją `echo`.

Przykład 7.6

```
<?
print("Mój pierwszy skrypt PHP");
?>
```

Instrukcje języka PHP muszą kończyć się średnikiem. Wyjątkiem są instrukcje zawierające inne instrukcje oraz instrukcje kończące blok kodu. Brak średnika na końcu instrukcji powoduje sygnalizację błędu składni w kodzie PHP.

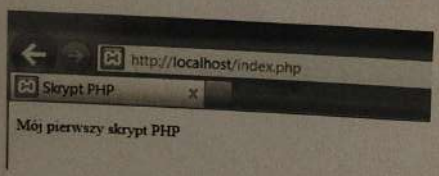
7.4.2. Blok PHP w kodzie HTML

Kod zapisany w języku PHP może zostać wstawiony bezpośrednio do kodu HTM

Przykład 7.7

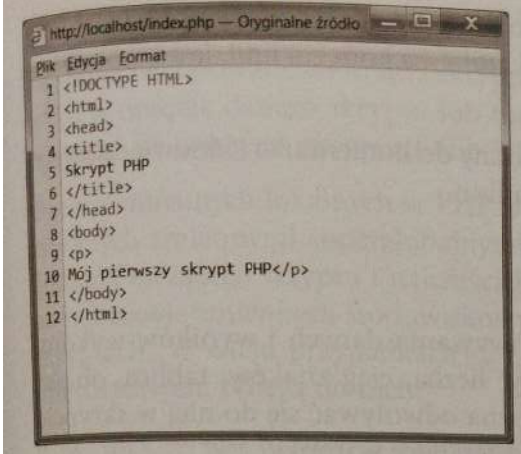
```
<!DOCTYPE HTML>
<html>
<head>
<title>Skrypt PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p>
<?php
echo "Mój pierwszy skrypt PHP";
?>
</p>
</body>
</html>
```

W podanym wyżej przykładzie w dokumencie HTML został umieszczony kod języka PHP. Aby zobaczyć efekt działania skryptu, należy go uruchomić w przeglądarce. W tym celu treść dokumentu zapisujemy w katalogu głównym serwera WWW (*/opt/lampp/htdocs/* dla pakietu LAMPP w systemie Linux, *C:\xampp\htdocs* dla pakietu XAMPP w systemie Windows), w pliku *index.php*. Następnie wpisujemy w przeglądarce adres *http://localhost/index.php* lub *http://127.0.0.1/index.php*. Jeżeli wyświetlona strona wygląda tak jak na rysunku 7.14, to skrypt działa prawidłowo.



Rysunek 7.14. Efekt działania skryptu

Można teraz obejrzeć kod źródłowy wyświetlonej strony. W tym celu należy w menu przeglądarki wybrać *Widok/Źródło* lub *Widok/Źródło strony*. Zostanie pokazany kod HTML wyświetlonej strony (rysunek 7.15).



Rysunek 7.15. Kod źródłowy strony

Widać, że w kodzie nie ma znaczników PHP. Pozostał jedynie kod HTML. Tak właśnie działają języki skryptowe wykonywane po stronie serwera. Ich zadaniem jest wygenerowanie kodu, który będzie mogła zinterpretować przeglądarka internetowa. W podanym wyżej przykładzie w miejsce skryptu PHP została wstawiona treść, która może być w poprawny sposób wyświetlona przez przeglądarkę internetową. Kod PHP został wykonany na serwerze, a jego wynik został zwrócony do przeglądarki jako zwykły HTML.

Zadanie 7.1

Wybierz jedną z metod instalowania PHP. Zainstaluj oprogramowanie PHP w swoim systemie. Zdefiniuj zabezpieczenia dla zainstalowanych zasobów. Uruchom serwer WWW i przetestuj jego działanie. Sprawdź działanie prostego skryptu PHP.

7.5. Składnia języka PHP

Podstawową zasadą podczas tworzenia skryptów PHP jest oddzielanie kolejnych instrukcji znakiem średnika. Instrukcje mogą być pisane pojedynczo w jednym wierszu lub w kilku.

7.5.1. Komentarze

W skrypcie PHP można umieszczać komentarze. Są one widoczne w kodzie źródłowym skryptu, natomiast w trakcie jego przetwarzania są usuwane i niewidoczne w przeglądarce. Występują trzy rodzaje komentarzy:

- komentarz blokowy,
- komentarz jednowierszowy,
- komentarz jednowierszowy uniksowy.

Komentarz blokowy zaczyna się od znaków `/*`, a kończy się znakami `*/`. Wszystko, co znajduje się między tymi znakami, jest ignorowane podczas przetwarzania skryptu.

Komentarz jednowierszowy zaczyna się od znaków // i kończy się w bieżącej linii skryptu. Wszystko, co znajduje się między znakami // a końcem linii, jest ignorowane podczas przetwarzania skryptu.

Komentarz jednowierszowy uniksowy jest podobny do komentarza jednowierszowego. Zaczyna się od znaku # i obowiązuje do końca linii.

7.5.2. Zmienne

Zmienne w programowaniu służą do przechowywania danych i wyników wykonywanych operacji. Wartością zmiennej może być liczba, ciąg znaków, tablica, obiekt. Zmienna musi posiadać nazwę, przez którą można odwoływać się do niej w skrypcie. Nazwa może być dowolna, ale musi spełniać następujące warunki:

- musi zaczynać się od litery lub znaku podkreślenia,
- może składać się jedynie z liter, cyfr i znaku podkreślenia,
- w nazwach rozróżniane są małe i duże litery,
- w nazwach można stosować polskie litery.

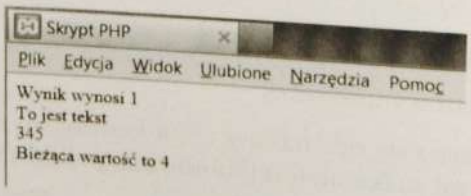
Przed nazwą zmiennej należy umieścić znak $.

W języku PHP zmienna jest tworzona przy pierwszym jej użyciu. Nie trzeba wcześniej deklarować zmiennej ani określać jej typu.

Przykład 7.8

```
<?php
$x=1;
$nazwa_1="tekst";
$liczba7=345;
$ilość=4;
echo "Wynik wynosi $x <br />";
echo "To jest $nazwa_1 <br/>";
echo "$liczba7 <br/>";
echo "Bieżąca wartość to $ilość";
?>
```

Wynik działania kodu został pokazany na rysunku 7.16.



Rysunek 7.16. Wynik kodu z przykładu

Zmienne predefiniowane

Zmienne deklarowane wewnątrz skryptu lub funkcji są zmiennymi lokalnymi i istnieją tylko w obrębie danego skryptu lub funkcji. Do zmiennych lokalnych nie można odwoływać się w innych skryptach lub funkcjach.

Oprócz zmiennych lokalnych w PHP istnieje grupa zmiennych predefiniowanych, nazywanych zmiennymi superglobalnymi. Przechowują one informacje dotyczące konfiguracji bieżącego skryptu i najczęściej mają postać tablicy. Ich wartość jest ustalana na podstawie zmiennych środowiskowych serwera WWW. Są dostępne we wszystkich skryptach. W wielu przypadkach są niezbędne do poprawnego tworzenia aplikacji internetowych. Należą do nich:

- `$_GET` — jest to tablica zawierająca zmienne przesyłane do skryptu za pomocą metody `GET`.
- `$_POST` — jest to tablica zawierająca zmienne przesyłane do skryptu za pomocą metody `POST`.
- `$_COOKIE` — jest to tablica zawierająca zmienne przekazane z serwera do skryptu za pomocą cookies.
- `$_FILES` — jest to tablica zawierająca zmienne przekazane do skryptu podczas przesyłania plików do serwera.
- `$_SERVER` — jest to tablica zawierająca zmienne przekazane do skryptu przez serwer WWW. Są to dane takie jak wersja serwera, ścieżka do pliku, adres skryptu, wysłane nagłówki.
- `$_ENV` — jest to tablica zawierająca wartości zmiennych środowiskowych serwera.
- `$_REQUEST` — jest to tablica zawierająca zmienne przekazane do skryptu przez użytkownika. Zawiera dane z `$_GET`, `$_POST`, `$_COOKIE` oraz `$_FILES`.
- `$_SESSION` — jest to tablica zawierająca zmienne zarejestrowane w bieżącej sesji.
- `$GLOBALS` — jest to tablica zawierająca odniesienie do każdej zmiennej utworzonej przez użytkownika, która ma zasięg globalny dla danego skryptu.

7.5.3. Typy danych

W większości języków programowania każda zmienna ma swój typ. Określa on, jakiego rodzaju wartości mogą zostać przypisane do zmiennej. W języku PHP typ danych określany jest w zależności od kontekstu, w jakim zmienna została użyta.

Występujące w PHP typy danych można podzielić na trzy rodzaje.

Typy **skalarne**, czyli typy proste. Należą do nich:

- typ `boolean`,
- typ `integer`,
- typ `float` lub `double`,
- typ `string`.

Typy złożone. Należą do nich:

- typ `array` (tablicowy),
- typ `object` (obiektowy).

Typy specjalne. Należą do nich:

- typ `resource`,
- typ `null`.

Typ boolean

Jest to typ logiczny. Przyjmuje jedną z dwóch wartości: prawda (`true`) lub fałsz (`false`). Wartości typu logicznego są wykorzystywane przy budowaniu wyrażeń logicznych, porównywaniu danych, sprawdzaniu warunków.

Typ integer

Jest to typ liczb całkowitych. Może służyć do przedstawiania dodatnich i ujemnych liczb całkowitych. Liczby mogą zostać zapisane w formacie dziesiętnym, ósemkowym (`oktalnym`) lub szesnastkowym (`heksadecymalnym`). Domyślnie stosowany jest format dziesiętny. Liczby w formacie ósemkowym należy poprzedzić znakiem `0` (`zero`). Liczby w formacie szesnastkowym należy poprzedzić znakami `0x`. Przy zapisie szesnastkowym można używać dużych i małych liter od `A` do `F`. Ich zakres jest zależny od platformy sprzętowo-systemowej, z reguły są to wartości od -2^{31} do $2^{31}-1$. W przypadku przekroczenia zakresu wartość jest konwertowana do typu `float`.

Przykład 7.9

- `537` — dodatnia liczba całkowita,
- `-451` — ujemna liczba całkowita,
- `032` — dodatnia liczba całkowita zapisana w formacie ósemkowym,
- `-021` — ujemna liczba całkowita zapisana w formacie ósemkowym,
- `0xFF` — dodatnia liczba całkowita zapisana w formacie szesnastkowym,
- `-0x0c` — ujemna liczba całkowita zapisana w formacie szesnastkowym.

Typ float

Jest to liczba zmiennoprzecinkowa (zmiennopozycyjna, rzeczywista). Może przedstawiać zarówno dodatnie, jak i ujemne liczby rzeczywiste. Ich zakres jest zależny od platformy sprzętowo-systemowej, z reguły są to wartości od `-1,8 * 10^308` do `1,8 * 10^308`. Do zapisu liczby można używać notacji wykładniczej.

Przykład 7.10

```
1,47; -2,346; 0.17E2; -1.0E-2
```

Typ string

Jest to typ łańcuchowy, który służy do zapamiętywania ciągu znaków. Pojedynczy znak ciągu jest zapamiętywany w jednym bajcie. Łańcuch znaków można utworzyć, korzystając z jednego z czterech sposobów:

- używając znaków apostrofu,
- używając znaków cudzysłowu,
- używając składni heredoc,
- używając składni nowdoc.

Użycie znaków apostrofu

Ciąg znakowy można utworzyć poprzez ujęcie go w znaki apostrofu. Taki ciąg pojawi się na ekranie w większości przypadków w niezmienionej postaci. Jeżeli apostrof zostanie użyty jako jeden ze znaków ciągu, to należy poprzedzić go znakiem \.

Przykład 7.11

```
<?php
echo 'To jest symbol apostrofu \' użyty w kodzie PHP';
?>
```

Przykład 7.12

```
<?php
$z = 'To jest tekst';
echo $z;
?>
```

Użycie znaków cudzysłowu

Drugim sposobem deklaracji ciągu znakowego jest ujęcie go w znaki cudzysłowu. Jeśli w takim ciągu wystąpi odwołanie do zmiennej, zostanie ono zmienione na wartość przypisaną do zmiennej.

Przykład 7.13

```
<?php
$x = "To jest tekst";
echo "$x";
?>
```

Użycie składni heredoc

Jeżeli używana jest składnia `heredoc`, ciąg znaków trzeba rozpocząć od sekwencji <<<. Po tej sekwencji musi wystąpić identyfikator. Ten sam identyfikator musi zostać użyty na końcu ciągu znakowego. Nazwa identyfikatora powinna być tworzona według tych

samych zasad, które obowiązują przy tworzeniu nazw zmiennych. Linia zawierająca identyfikator kończący ciąg nie może zawierać żadnych innych znaków oprócz tego identyfikatora i średnika.

Przykład 7.14

```
<?php
$napis = „napis";
$tekst = <<<TX
 Tutaj rozpoczyna się $napis
TX;
echo $tekst;
?>
```

Jeżeli w ciągu znaków wystąpi odwołanie do zmiennej, podobnie jak w przypadku składni z cudzysłowami, zostanie ono zmienione na wartość przypisaną do zmiennej. Ten sposób tworzenia napisów jest wykorzystywany, kiedy są one długie i składają się z wielu linii.

Użycie składni nowdoc

Składnia `nowdoc` jest podobna do składni `heredoc`. Różnica w definicji typu polega na umieszczeniu identyfikatora w znakach apostrofu. Różnica w działaniu polega na tym, że gdy w ciągu znaków wystąpi odwołanie do zmiennej, nie jest ona zamieniana na odpowiadającą jej wartość.

Przykład 7.15

```
<?php
$napis = „napis";
$tekst = <<<'PC'
 Tutaj rozpoczyna się $napis
PC;
echo $tekst;
?>
```

Typ array

Tablice przechowują zbiory danych najczęściej jednego typu, a dostęp do nich jest możliwy przez indeks. Tablice mogą być jednowymiarowe lub wielowymiarowe.

Struktury tablicowe w PHP są tworzone na bieżąco przez przypisanie wartości dla poszczególnych indeksów albo za pomocą instrukcji `array()`. W PHP można tworzyć tablice zwykłe, które są indeksowane kolejnymi liczbami całkowitymi (tablice indeksowane), oraz tablice asocjacyjne, które są indeksowane kluczem.

Są dwa sposoby tworzenia **tablic indeksowanych**: przez automatyczne tworzenie indeksu lub przez jego ręczne przypisanie.

Polecenie `array` dla tablic indeksowanych tworzonych przez automatyczne przypisanie indeksu ma postać:

```
$tab = array(t1, t2, ..., tn);
```

Powstała zmienna jest zmienną typu tablicowego, natomiast `t1`, `t2`, . . . , `tn` to kolejne wartości znajdujące się w tablicy.

Składnia tworzenia tablicy indeksowanej przez ręczne przypisanie indeksu ma postać:

```
$tab[0] = t1;
$tab[1] = t2;
$tab[2] = t3;
...
$tab[n-1] = tn;
```

Numer indeksu może zostać pominięty. Wtedy nawiasy kwadratowe powinny pozostać puste; PHP nada te indeksy automatycznie.

Niezależnie od tego, jak tablica została utworzona, można do niej dodać kolejny element. Jeżeli podczas przypisywania wartości dla kolejnej pozycji tabeli nawiasy kwadratowe pozostaną puste, zostanie jej nadany kolejny indeks tabeli. Stosując tę metodę, można uniknąć sytuacji, w której indeksy zostaną źle ponumerowane. Jeżeli w ten sposób będą dodawane wpisy do nowej tabeli, to pierwszy z nich będzie miał indeks równy `0`.

Przykład 7.16

```
<?php
$ks = array("Dżuma", "Potop", "Obcy");
echo $ks[1];
?>
```

Przykład 7.17

```
<?php
$im[0] = "Jan";
$im[1] = "Paweł";
$im[2] = "Michał";
echo $im[2];
?>
```

Przykład 7.18

```
<?php
$tab[] = 1;
$tab[] = 2;
$tab[] = 3;
echo $tab[2];
?>
```

Tablice asocjacyjne, czyli tablice skojarzeniowe (ang. *associative array*), to tablice, w których zamiast indeksów liczbowych używa się identyfikatorów (tzw. kluczy). W danych tego typu przechowywane są pary danych: unikatowy klucz i wartość. Dostęp do wartości uzyskuje się poprzez podanie wartości klucza. Różnica między tablicą indeksowaną a tablicą asocjacyjną polega na wartości klucza. Dla tablic indeksowanych klucz jest liczbą, dla tablic asocjacyjnych jest on ciągiem znaków mających określone znaczenie.

Polecenie `array` dla tablic asocjacyjnych ma postać:

```
$tab_a = array(
      klucz1 => a1,
     klucz2 => a2,
      ...,
     kluczn => an
);
```

lub

```
$tab_a[klucz1] = a1;
$tab_a[klucz2] = a2;
      ...,
$tab_a[kluczn] = an;
```

Przykład 7.19

```
<?php
$osoba = array("nazwisko" => "Kowalski", "imię" => "Jan", "wiek" => 27);
echo $osoba["nazwisko"];
?>
```

Przykład 7.20

```
<?php
$osoba["nazwisko"] = "Kowalski";
$osoba["imię"] = "Jan";
$osoba["wiek"] = 27;
```

```
echo $osoba["nazwisko"];
?>
```

W języku PHP można tworzyć **tablice wielowymiarowe**. Są to tablice, które zawierają w sobie inne tablice.

Przykład 7.21

```
<?php
$dane = array(
        array("nazwisko" => "Kowalski",
              "imię" => "Jan",
              "wiek" => 27),
        array("nazwisko" => "Nowak",
              "imię" => "Paweł",
              "wiek" => 24),
        array("nazwisko" => "Górka",
              "imię" => "Tomasz",
              "wiek" => 29)
             );
?>
```

Każdy z elementów tablicy `$dane` to tablica asocjacyjna składająca się z trzech elementów: `nazwisko`, `imię`, `wiek`. Aby wyświetlić zawartość tej tablicy, należy użyć indeksu elementu głównego razem z nazwą klucza elementu wewnętrznego.

Przykład 7.22

```
echo $dane[2]["imię"];
```

Wynikiem będzie wyświetlenie tekstu `Tomasz`.

Typ object

Jest to typ służący do przechowywania obiektów.

Typ resource

Jest to specjalny typ, wskazujący, że zmienna przechowuje odwołanie do zasobu zewnętrznego (aplikacji lub pliku) utworzonego za pomocą specjalnych funkcji.

Typ null

Typ null jest przeznaczony dla zmiennych, które zostały zadeklarowane, ale nie przypisano im żadnych wartości. Wielkość liter przy deklaracji tego typu nie ma znaczenia. Poprawne są zapisy: `NULL`, `null`, `Null` i inne.

Przykład 7.23

```
$x1 = null;
```

Zmiana typu zmiennej

W języku PHP podczas przypisywania zmiennej nowej wartości ponownie ustalany jest jej typ. Czasami zachodzi jednak konieczność zmiany typu danych. Można tego dokonać za pomocą rzutowania (`cast`) — wtedy efekt jest jednorazowy — lub za pomocą funkcji `settype()` — wtedy efekt jest trwały.

Rzutowanie typów odbywa się przez podanie nowego typu w nawiasie przed zmienną lub wartością, której typ należy zmienić. Dozwolone są określone typy rzutowań. Są to:

- `integer` — rzutowanie do typu całkowitego,
- `float` (`double`) — rzutowanie do typu rzeczywistego,
- `string` — rzutowanie do ciągu tekstowego,
- `array` — rzutowanie do tablicy,
- `object` — rzutowanie do obiektu.

Przykład 7.24

```
<?php
$x = 23.75;
$y = (integer) $x;
echo "$x <br/>";
echo "$y";
?>
```

Zmianę typu w sposób trwały przeprowadza się za pomocą funkcji `settype()`. Funkcja ta posiada dwa argumenty. Pierwszym jest nazwa zmiennej, której nadajemy nowy typ, drugim parametr określający nowy typ zmiennej. Parametr ten może przyjmować wartości: `integer`, `float`, `string`, `array`, `object`.

```
settype($zmienna, 'nowy typ');
```

Jeżeli zmiana typu przebiegła pomyślnie, funkcja zwraca wartość `true`. Gdy zmiana typu nie powiodła się, zwraca wartość `false`.

Przykład 7.25

```
<?php
$x = 97.234;
echo "Wartość zmiennej \$x: $x <br />";
settype($x, 'string');
echo "Wartość zmiennej \$x: $x <br />";
settype($x, 'integer');
```

```
echo "Wartość zmiennej \$x: $x <br />";
?>
```

Zadanie 7.2

Utwórz trzy zmienne. Pierwszej przypisz wartość liczby całkowitej, drugiej wartość liczby rzeczywistej, trzeciej ciąg znaków. Utwórz w języku PHP skrypt, który będzie wyświetlał wartości tych zmiennych na ekranie.

Zadanie 7.3

Utwórz cztery zmienne. Przypisz im wartości typu `string`. Do tworzenia zmiennych wykorzystaj po kolei każdy z czterech sposobów, używając znaków apostrofu, znaków cudzysłowu, składni heredoc oraz składni nowdoc. Utwórz w języku PHP skrypt, który będzie wyświetlał wartości tych zmiennych na ekranie. Przeanalizuj sposób wyświetlania zmiennych i ich wartości.

7.5.4. Operatory i wyrażenia

Operatory służą do wykonywania działań na zmiennych. W języku PHP można je podzielić na operatory:

- arytmetyczne,
- porównania,
- bitowe,
- logiczne,
- przypisania,
- inkrementacji i dekrementacji,
- pozostałe.

Operatory arytmetyczne

Służą do wykonywania operacji arytmetycznych.

Tabela 7.2. Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	`$a + $b`
-	odejmowanie	`$a - $b`
*	mnożenie	`$a * $b`
/	dzielenie	`$a / $b`
%	dzielenie modulo (reszta z dzielenia)	`$a % $b`

Operatory porównania

Operatory porównania porównują argumenty; ich wynikiem jest wartość logiczna `true` (prawda) lub `false` (fałsz).

Tabela 7.3. Operatory porównania

Operator	Działanie	Przykład
==	Wynik `true`, gdy argumenty są równe.	`$a == $b`
!=	Wynik `true`, gdy argumenty są różne.	`$a != $b`
===	Wynik `true`, gdy argumenty są tego samego typu i są równe.	`$a === $b`
!==	Wynik `true`, gdy argumenty są różne lub są różnych typów.	`%a != $b`
>	Wynik `true`, gdy argument pierwszy jest większy od drugiego.	`$a > $b`
<	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego.	`$a < $b`
>=	Wynik `true`, gdy argument pierwszy jest większy od drugiego lub mu równy.	`$a >= $b`
<=	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego lub mu równy.	`$a <= $b`
<>	Wynik `true`, gdy argumenty są różne.	`$a <> $b`

Operatory bitowe

Operatory bitowe umożliwiają wykonanie operacji na poszczególnych bitach liczb.

Tabela 7.4. Operatory bitowe

Operator	Działanie	Przykład
&	iloczyn bitowy (AND)	`$a & $b`
|	suma bitowa (OR)	`$a | $b`
~	negacja bitowa (NOT)	`~$a`
^	bitowa różnica symetryczna	`$a ^ $b`
>>	przesunięcie bitowe w prawo	`$a >> n`
<<	przesunięcie bitowe w lewo	`$a << n`

Operatory logiczne

Operatory logiczne wykonują operacje na argumentach, które posiadają wartość logiczną (`true` lub `false`).

Tabela 7.5. Operatory logiczne

Operator	Działanie	Przykład
`and`	iloczyn logiczny	`$a and $b`
`&&`	iloczyn logiczny	`$a && $b`
`or`	suma logiczna	`$a or $b`
`||`	suma logiczna	`$a || $b`
`!`	negacja logiczna (NOT)	`!$a`
`xor`	różnica symetryczna	`$a xor $b`

Wynik iloczynu logicznego przyjmuje wartość `true` tylko wtedy, gdy obydwa argumenty mają wartość `true`. W pozostałych przypadkach wynik przyjmuje wartość `false`.

Wynik sumy logicznej przyjmuje wartość `false` tylko wtedy, gdy obydwa argumenty mają wartość `false`. W pozostałych przypadkach wynik przyjmuje wartość `true`.

Negacja logiczna zmienia wartość argumentu na przeciwną.

Wynik różnicy symetrycznej przyjmuje wartość `true` tylko wtedy, gdy obydwa argumenty mają różną wartość. Gdy obydwa argumenty mają równą wartość wynikiem jest `false`.

Operatory przypisania

Operatory przypisania przypisują wartości argumentom znajdującym się po lewej stronie operatora. Oprócz prostej operacji przypisania w języku PHP istnieje zestaw operacji łączących operację przypisania z innymi operacjami, np. arytmetyczną, bitową lub łańcuchową. Np. zapis: `$a += 4` oznacza w praktyce to samo co zapis: `$a = $a + 4`. Stosowanie takich skróconych zapisów upraszcza tworzenie bardziej rozbudowanych skryptów. W języku PHP występuje cała grupa operatorów tego typu.

Tabela 7.6. Niektóre operatory przypisania

Operator	Przykład	Znaczenie
`=`	`$x = 10`	$x = 10
`+=`	`$x += 5`	$x = $x + 5
`-=`	`$x -= 5`	$x = $x – 5
`*=`	`$x *= 5`	$x = $x * 5
`/=`	`$x /= 5`	$x = $x / 5
`%=`	`$x %= 5`	$x = $x % 5
`.=`	`$x .= "ab"`	$x = $x . "ab"

Operator konkatenacji

Operator konkatenacji służy do łączenia ciągów znakowych. Niezależnie od typów danych, które będą łączone za pomocą tego operatora, są one traktowane jako ciągi znaków i rezultat ich łączenia jest zawsze ciągiem znaków. Operator konkatenacji jest oznaczany pojedynczą kropką.

Tabela 7.7. Operator konkatenacji

Operator	Działanie	Przykład
.	łączenie łańcuchów znakowych	`"łańcuch "." tekstowy"`

Przykład 7.26

```
<?php
$osoba["nazwisko"] = "Kowalski";
$osoba["imie"] = "Jan";
$osoba["wiek"] = 27;
echo $osoba["nazwisko"] . " " . $osoba["imie"] . " ma " . $osoba["wiek"]
. " lat.";
?>
```

Operatory inkrementacji i dekrementacji

Operatory *inkrementacji* (zwiększania) i *dekrementacji* (zmniejszania) służą do zwiększenia lub zmniejszania wartości zmiennej o `1`.

Operator inkrementacji powoduje zwiększenie wartości o jeden. Może występować w postaci przedrostkowej (`++$x`) lub przyrostkowej (`$x++`). Operacja `$x++` zwiększa wartość zmiennej po jej wykorzystaniu, natomiast `++$x` przed jej wykorzystaniem. Oba operatory zwiększają wartość zmiennej, ale nie są równoważne.

Operator dekrementacji działa analogicznie, tylko zamiast zwiększać wartości zmiennych, zmniejsza je.

Przykład 7.27

```
<?php
$x = 7;
echo $x++; echo "<br />";
echo ++$x; echo "<br />";
$v = $x;
$t = $x++;
$z = $x;
$y = ++$x;
```

```
echo "Wartość zmiennej \$v = $v <br />";
echo "Wartość zmiennej \$t = $t <br />";
echo "Wartość zmiennej \$z = $z <br />";
echo "Wartość zmiennej \$y = $y <br />";
?>
```

7.5.5. Stałe

W języku PHP występują stałe, czyli identyfikatory, których wartości nie ulegają zmianie. Stałe, podobnie jak zmienne, są identyfikowane w skrypcie przez nazwę. Nazwa musi spełniać podobne warunki jak nazwa zmiennej. Jednak w nazwach stałych w odróżnieniu od zmiennych nie używa się znaku dolara (`$`). Zgodnie z przyjętą konwencją nazwy stałych zawsze pisane są wielkimi literami. Zasięg stałych jest globalny i można odwoływać się do nich w każdym miejscu skryptu. Nie można tworzyć stałych poprzez zwykłe przypisanie. Do ich definiowania służy funkcja `define()`, która posiada dwa argumenty: nazwę stałej oraz przypisaną do niej wartość. Stałe raz zdefiniowane nie mogą być zmieniane ani usuwane. Mogą zawierać tylko wartości skalarne (typ `boolean`, `integer`, `float` lub `string`).

Definiowanie stałej ma postać:

```
define("NAZWA_STAŁEJ", wartość);
```

Przykład 7.28

```
<?php
define("WIEK", "21");
echo "Mamy wiek " . WIEK;
?>
```

Stałe predefiniowane

W języku PHP istnieje grupa stałych predefiniowanych, które towarzyszą każdemu uruchamianemu skryptowi. Część z nich jest dostępna dzięki różnym rozszerzeniom i można z nich korzystać, gdy te rozszerzenia zostały wczytane. Przykładowe stałe predefiniowane to:

- `TRUE` — stała zawierająca logiczną wartość prawdy,
- `FALSE` — stała zawierająca logiczną wartość fałszu,
- `PHP_VERSION` — stała reprezentująca aktualnie używaną wersję parsera PHP,
- `PHP_OS` — stała zawierająca nazwę systemu operacyjnego, na którym uruchamiany jest parser PHP.

Istnieje grupa tak zwanych **magicznych stałych**, które zmieniają się w zależności od miejsca ich użycia. Należą do nich:

- `__FILE__` — stała zawierająca nazwę pliku ze skryptem, który jest aktualnie przetwarzany,
- `__LINE__` — stała zawierająca numer linii w skrypcie, która aktualnie jest przetwarzana,
- `__DIR__` — stała zawierająca nazwę katalogu pliku,
- `__FUNCTION__` — stała zawierająca nazwę funkcji,
- `__CLASS__` — stała zawierająca nazwę klasy,
- `__METHOD__` — stała zawierająca nazwę metody.

7.6. Instrukcje sterujące

Instrukcje sterujące służą do sterowania kolejnością wykonywania poleceń zapisanych w skrypcie. W języku PHP istnieje cała grupa instrukcji, które pozwalają na zmianę kolejności wykonywania poleceń w zależności od spełnienia lub niespełnienia określonych warunków.

7.6.1. Instrukcja warunkowa

Instrukcja warunkowa może występować w kilku wersjach. Najprostsza ma postać:

```
if (warunek){
   instrukcja1;
   instrukcja2;
   instrukcjaN;
}
```

i działa w następujący sposób: jeśli warunek jest spełniony, wykonywany jest ciąg instrukcji zapisanych między nawiasami { . . }. Jeśli warunek nie jest spełniony, program pomija ten ciąg instrukcji i kontynuuje wykonanie kolejnych instrukcji. Instrukcja, która jest wykonywana, gdy warunek jest spełniony, może być instrukcją prostą (jedna instrukcja) lub złożoną (blokiem instrukcji).

Przykład 7.29

```
<?php
$x = 11;
$y = 7;
if ($y > $x)
echo "Wartość zmiennej y jest większa od wartości zmiennej x.";
if ($y < $x)
echo "Wartość zmiennej x jest większa od wartości zmiennej y.";
?>
```

Jeżeli trzeba sprawdzić dwa wykluczające się wzajemnie warunki, można użyć rozbudowanej instrukcji warunkowej w postaci:

```
if (warunek)
   instrukcja1;
else
   instrukcja2;
```

Instrukcja1 i *instrukcja2* mogą być instrukcjami prostymi lub złożonymi. Gdy w bloku `if` lub `else` występuje wiele instrukcji, należy je umieścić w nawiasach klamrowych. Jeżeli w takim przypadku spełniony jest warunek, wykonywana jest *instrukcja1*, w przeciwnym razie *instrukcja2*. Po wykonaniu jednej z tych instrukcji program kontynuuje działanie.

Przykład 7.30

```
<?php
$x = 11;
$y = 7;
if ($y > $x)
echo "Wartość zmiennej y jest większa od wartości zmiennej x.";
else
echo "Wartość zmiennej x jest większa od wartości zmiennej y.";
?>
```

Kolejna wersja instrukcji warunkowej pozwala sprawdzać wiele warunków. W tej wersji po `if` może wystąpić kilka następnych bloków `else if`. Instrukcja ma postać:

```
if (warunek1)
   instrukcja1;
else if (warunek2)
   instrukcja2;
...
else if (warunekn)
   instrukcjan;
else
   instrukcjan+1;
```

Działanie instrukcji polega na sprawdzeniu warunku *warunek1*. Jeżeli jest on spełniony, wykonana zostanie *instrukcja1*. W przeciwnym razie sprawdzany jest *warunek2* i jeżeli jest spełniony, wykonana zostanie *instrukcja2* itd. Jeżeli żaden warunek nie będzie spełniony, wykonana zostanie *instrukcjan+1*. Instrukcję `if ... else if` można zapisywać, łącząc słowa `else` w konstrukcję `elseif`. W standardach pisania kodu obowiązuje składnia ze słowem `elseif`.

Przykład 7.31

```
<?php
$x = 11;
$y = 7;
$z = $x + $y;
if ($z < 0)
echo "Wartość zmiennej z jest ujemna.";
elseif ($z < 10)
echo "Wartość zmiennej z jest zawarta w zakresie od 0 do 10.";
elseif ($z < 20)
echo "Wartość zmiennej z jest zawarta w zakresie od 10 do 20.";
elseif ($z < 30)
echo "Wartość zmiennej z jest zawarta w zakresie od 20 do 30.";
else
echo "Wartość zmiennej z jest większa od 30.";
?>
```

7.6.2. Instrukcja switch

Kolejną instrukcją sterującą przepływem jest instrukcja `switch`. Jest to instrukcj wyboru. Wyrażenie występujące w tej konstrukcji jest porównywane z listą szukanyc wartości aż do znalezienia pasującej wartości.

```
switch (wyrażenie) {
   case wartość1:
      instrukcje1;
      break;
   case wartość2:
      instrukcje2;
      break;
   case wartość3:
      instrukcje3;
      break;
   default:
      instrukcje4;
}
```

Działanie instrukcji wygląda następująco: sprawdź wartość *wyrażenia* i jeżeli wy- nikiem jest *wartość1*, wykonaj *instrukcje1* i wyjdź z bloku `switch` (polecenie

break). Jeżeli wynikiem jest *wartość2*, to wykonaj *instrukcje2* i wyjdź z bloku switch. Jeżeli wynikiem jest *wartość3*, to wykonaj *instrukcje3* i wyjdź z bloku switch. Jeżeli wartość jest inna, wykonaj *instrukcje4* i zakończ blok switch.

Przykład 7.32

```
<?php
$kolor="red";
switch ($kolor)
{
case "red":
   echo "Dominującym kolorem jest czerwony!";
   break;
case "blue":
   echo " Dominującym kolorem jest niebieski!";
   break;
case "green":
   echo " Dominującym kolorem jest zielony!";
   break;
default:
   echo "Brak dominującego koloru!";
}
?>
```

7.6.3. Operator warunkowy

Operator warunkowy działa podobnie jak instrukcja if. Zwraca wartość jednego z dwóch wyrażeń rozdzielonych dwukropkiem. Ma on postać:

```
warunek ? wartość1 : wartość2
```

i działa w następujący sposób: jeżeli warunek jest prawdziwy, to jako wartość całego wyrażenia podstaw *wartość1*, w przeciwnym razie jako wartość całego wyrażenia podstaw *wartość2*.

Przykład 7.33

```
<?php
$x = 11;
$wynik = ($x < 0) ? "ujemna" : "dodatnia";
echo "Wartość zmiennej x jest $wynik.";
?>
```

7.6.4. Pętle

Pętle są używane do wykonywania powtarzających się czynności. W języku PHP występują następujące rodzaje pętli: `for`, `while`, `do ... while`, `foreach`.

Pętla for

Pętla typu `for` służy do budowania pętli, gdy został podany licznik jej wykonań oraz warunek, który musi być spełniony, aby kolejny raz wykonać pętlę. Składnia instrukcji jest następująca:

```
for (wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące) {
instrukcje do wykonania;
}
```

- *wyrażenie początkowe* — inicjuje zmienną, która jest używana jako licznik pętli,
- *wyrażenie warunkowe* — określa warunek, który musi być spełniony, aby pętla została wykonana kolejny raz,
- *wyrażenie modyfikujące* — modyfikuje zmienną, która jest licznikiem.

Pętlę `for` wykorzystuje się zwykle wtedy, gdy znamy liczbę wykonywanych powtórzeń.

Przykład 7.34

```
<?php
for ($i = 1; $i <= 5; $i++) {
echo "Pętla wykonana $i raz/y <br />";
}
?>
```

Pętla while

Pętla `while` jest zwykle wykorzystywana wtedy, gdy liczba wykonywanych powtórzeń nie jest znana, a zakończenie pętli zależy od spełnienia określonego warunku. Składnia instrukcji jest następująca:

```
while (warunek) {
instrukcje;
}
```

Blok instrukcji jest wykonywany w pętli, dopóki wyrażenie warunkowe jest prawdziwe. Konstrukcja mówi: dopóki wyrażenie warunkowe jest prawdziwe, wykonuj instrukcje.

Przykład 7.35

```
<?php
$i = 0;
```

```
while ($i++ < 5) {
echo("Pętla wykonana $i raz/y </br>");
}
?>
```

Pętla do ... while

Pętla `do ... while` jest odmianą pętli `while`. Jej składnia jest następująca:

```
do {
instrukcje;
}
while (warunek);
```

Konstrukcja mówi: wykonuj instrukcje, dopóki wyrażenie warunkowe jest prawdziwe. W pętli `do ... while` blok instrukcji jest wykonywany co najmniej raz, nawet jeżeli warunek zapisany w wyrażeniu warunkowym jest fałszywy, ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

Przykład 7.36

```
<?php
$i = 1;
do {
echo("Pętla wykonana $i raz/y </br>");
}
while ($i++ < 5);
?>
```

Pętla foreach

Pętla `foreach` umożliwia dostęp do elementów tablicy lub właściwości obiektu. Może występować w postaci:

```
foreach($tablica as $wartość){
instrukcje;
}
```

lub

```
foreach($tablica as $klucz => $wartość){
instrukcje;
}
```

W drugim przypadku oprócz wartości elementu tablicy otrzymujemy wartość aktualnego klucza.

Przykład 7.37

```
<?php
$tab = array(
1 => 'biały',
2 => 'czarny',
3 => 'niebieski',
4 => 'zielony'
);
foreach($tab as $x){
echo "$x <br />";
};
?>
```

Jeżeli na listingu chcemy uzyskać nazwy indeksów, musimy zastosować drugą konstrukcję.

Przykład 7.38

```
<?php
$tab = array(
1 => 'biały',
2 => 'czarny',
3 => 'niebieski',
4 => 'zielony'
);
foreach($tab as $kl =>$x){
echo "tab[$kl] = $x <br />";
};
?>
```

Instrukcja break

Instrukcja `break` jest instrukcją modyfikującą zachowanie pętli. Służy do przerwania jej wykonywania. Można ją stosować niezależnie od rodzaju pętli.

Przykład 7.39

```
<?php
$i = 0;
while(true){
echo("Wypisz $i <br />");
```

```
if ($i >= 20) break;
$i++;
}
?>
```

Pętla `while` będzie wykonywana, dopóki `$i < 20`, ale jeżeli wartość zmiennej `$i` przekroczy `20`, nastąpi przerwanie wykonywania powtarzających się instrukcji i wyjście z pętli.

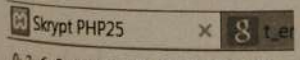
Instrukcja continue

Instrukcja `continue`, podobnie jak `break`, służy do modyfikowania zachowania się pętli. Po jej napotkaniu następuje przerwanie wykonywania bieżącej iteracji i przejście na początek tej pętli.

Przykład 7.40

```
<?php
for ($i = 0; $i <= 30; $i++) {
if (($i % 3) != 0)
continue;
echo"$i; ";
}
?>
```

Podany kod wyświetli liczby z zakresu od `0` do `30` podzielne przez 3 (rysunek 7.17). Jeżeli wynik dzielenia przez 3 nie jest liczbą całkowitą, następuje przerwanie wykonywania pętli i powrót na jej początek (liczby niepodzielne przez 3 nie będą wyświetlane).



0; 3; 6; 9; 12; 15; 18; 21; 24; 27; 30;

Rysunek 7.17. Skrypt wyświetla liczby z zakresu od 0 do 30 podzielne przez 3

7.6.5. Naprzemienne bloki kodu PHP i HTML

Definiując blok PHP w kodzie HTML, należy korzystać ze znaczników początku i końca PHP. Jeżeli wewnątrz bloku PHP wystąpi kod HTML, który będzie wykonywany opcjonalnie — tylko wtedy, gdy zostanie spełniony określony warunek — to można zdefiniować przełączanie się do trybu HTML wewnątrz bloku warunkowego PHP.

Przykład 7.41

```
<!DOCTYPE HTML>
<html>
```

```
<head>
<title>Bloki PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body bgcolor="#dfdfdf">
<?php
$tx = true;
if ($tx){
?>
<table align="left" border= "1" width="400" hspace="40" vspace="20" cell-
spacing="4" >
<tr><td> Nazwisko</td><td> Imię</td><td> Telefon</td></tr>
<tr><td> Nowak</td><td> Adam</td><td> 692399123</td></tr>
<tr><td> Kowalski</td><td> Jan</td><td> 628345621</td></tr>
<tr><td> Górniak</td><td> Mateusz</td><td> 638231484</td></tr>
</table>
<?php
}
?>
</body>
</html>
```

W podanym przykładzie przejście do HTML nastąpi tylko wtedy, gdy warunek `if` będzie spełniony (rysunek 7.18).



Rysunek 7.18. Wynik wykonania kodu HTML i PHP

Zadanie 7.4

Napisz skrypt, który będzie wyświetlał liczby nieparzyste z przedziału od 0 do 30. Liczby powinny być wyświetlane w kolumnie od wartości największej do najmniejszej.

Zadanie 7.5

Napisz skrypt, który umieści w tabeli oceny semestralne z pięciu przedmiotów, a następnie wyświetli dane z tabeli w postaci dwóch kolumn. Pierwsza kolumna będzie zawierać nazwy przedmiotów, a druga oceny z tych przedmiotów.

Zadanie 7.6

Połącz skrypt PHP z zadania poprzedniego z kodem HTML, tak aby nazwy przedmiotów i oceny z tych przedmiotów były wyświetlane w dwukolumnowej tabeli z nagłówkami.

7.7. Funkcje

Funkcja jest to ciąg instrukcji stanowiący blok kodu, który może być wielokrotnie wykorzystany w różnych programach lub jego miejscach. Funkcja jest wywoływana przez podanie jej nazwy i listy argumentów. Po zakończeniu działania funkcja często zwraca pewną wartość.

Istnieją dwa rodzaje funkcji. Funkcje, które zostały wbudowane w język, oraz funkcje zdefiniowane przez programistę.

7.7.1. Definiowanie funkcji

W języku PHP można definiować własne funkcje. Funkcję definiujemy za pomocą słowa kluczowego `function`, po której następuje nazwa funkcji, następnie w nawiasach okrągłych powinny zostać wymienione argumenty funkcji oddzielone przecinkami. W nawiasach klamrowych jest zapisywana treść funkcji. Definicja funkcji ma postać:

```
function nazwa ($argument1, $argument2, ...) {
instrukcje
}
```

Jeżeli funkcja posiada argumenty, są one zapisane w nawiasach okrągłych jako lista zmiennych oddzielonych przecinkami. Jeżeli funkcja nie posiada argumentów, nawiasy okrągłe powinny pozostać puste. Nazwy funkcji powinny być tworzone zgodnie z zasadami obowiązującymi dla nazw zmiennych i powinny odzwierciedlać to, co funkcja będzie robiła. Nazwa funkcji nie może zaczynać się od znaku `$`.

Przykład 7.42

```
<?php
function napis() {
echo "<h2>Programuj w PHP!</h2>";
}
napis();
?>
```

Przykład 7.43

```
<?php
function dodaj($a, $b) {
$c = $a + $b;
echo "Wynik dodawania $a i $b to " . $c;
}
dodaj(15,37);
?>
```

7.7.2. Zwracanie wartości przez funkcje

Jeżeli chcemy, aby funkcja zwracała wartość, która będzie wykorzystywana w innych działaniach, należy użyć słowa kluczowego `return`. Zatrzymuje ona działanie funkcji i zwraca wskazaną wartość do bloku kodu, z którego funkcja została wywołana. Definicja funkcji ze słowem kluczowym `return` ma postać:

```
function nazwa ($argument1, $argument2, ...) {
instrukcje
return wartość;
}
```

Słowo kluczowe return podane wewnątrz funkcji bez żadnego argumentu powoduje przerwanie działania funkcji.

Przykład 7.44

```
<?php
function dodaj($a, $b) {
$c = $a + $b;
return $c;
}
$suma = dodaj(14,7);
echo "Wynik dodawania to $suma";
?>
```

7.7.3. Zasięg zmiennych

Zasięg zmiennej to obszar, w którym można odwoływać się bezpośrednio do zmiennej. Zmienna może być:

- lokalna,
- globalna.

Zmienne globalne

Zmienne **globalne** są widoczne w całym skrypcie. Są to zmienne, które zostały zdefiniowane poza funkcją. Można z nich korzystać w każdym miejscu skryptu z wyjątkiem wnętrza funkcji.

Przykład 7.45

```
<?php
$zm =1;
function pokaz() {
echo "Wartość zmiennej \$zm wynosi $zm. <br />";
}
pokaz();
?>
```

W podanym przykładzie wartość zmiennej `$zm` nie zostanie wyświetlona. Zmienna ma zasięg globalny, ale została zdefiniowana poza funkcją `pokaz()`, w związku z tym nie ma ona dostępu do zmiennej. Jest to przydatna cecha języka pozwalająca uniknąć konfliktów wynikających ze stosowania takich samych nazw zmiennych używanych w różnych obszarach skryptu. Funkcje wykorzystujące dane zewnętrzne powinny pobierać je w postaci argumentów.



UWAGA

Taki sposób działania zmiennych globalnych jest charakterystyczny dla języka PHP. W większości języków programowania do zmiennych globalnych można odwoływać się z dowolnego miejsca.

Instrukcja global

Jeżeli istnieje konieczność odwołania się do zmiennej globalnej wewnątrz funkcji, można wykorzystać instrukcję `global`. Przed odwołaniem się do zmiennej należy umieścić konstrukcję:

```
global $nazwa_zmiennej;
```

Za pomocą jednej instrukcji `global` można zadeklarować w funkcji wiele zmiennych globalnych.

```
global $zm1, $zm2, zm3;
```

Przykład 7.46

```
<?php
$zm =1;
function pokaz() {
```

```
global $zm;
echo "Wartość zmiennej \$zm wynosi $zm. <br />";
}
pokaz();
?>
```

Inną metodą odwołania do zmiennej globalnej jest odwołanie się do superglobalnej tablicy `$GLOBALS`. Zawiera ona odwołania do wszystkich zdefiniowanych w skrypcie zmiennych globalnych. Nazwy tych zmiennych są indeksami tablicy `$GLOBALS`. Odwołanie do tablicy ma postać:

```
$GLOBALS['nazwa zmiennej']
```

Przykład 7.47

```
<?php
$zm =7;
function pokaz() {
echo "Wartość zmiennej \$zm wynosi ";
 echo $GLOBALS['zm'];
echo  ". <br />";
}
pokaz();
?>
```

Posługując się zmiennymi globalnymi, należy zachować szczególną ostrożność, ponieważ każda zmiana ich wartości będzie widoczna w całym skrypcie.

Zmienne lokalne

Zmienne **lokalne** mają zasięg lokalny i są definiowane wewnątrz funkcji. Ich zasięg dotyczy tylko funkcji, w której zostały zdefiniowane, i poza nią nie są widoczne.

Przykład 7.48

```
<?php
function tekst() {
$tx = "Tekst1";
}
echo "Tekst zapisany w zmiennej to: $tx. <br />";
?>
```

W podanym przykładzie wartość zmiennej `$tx` nie zostanie wyświetlona. Zmienna ma zasięg lokalny, została zdefiniowana wewnątrz funkcji `tekst()` i poza tą funkcją nie istnieje.

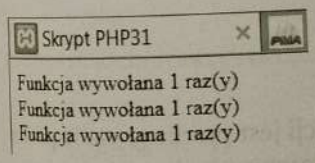
Zmienne statyczne

Zmienne statyczne to zmienne lokalne funkcji, które zachowują swoją wartość między wywołaniami funkcji. Domyślnie zmienna lokalna jest tworzona w chwili wywołania funkcji i jest widoczna w obrębie tej funkcji. Gdy funkcja zakończy działanie, zmienna znika. Przy ponownym wywołaniu funkcji zmienna jest tworzona i przypisywana jest jej wartość początkowa.

Przykład 7.49

```
<?php
function funk()
{
$i = 1;
echo "Funkcja wywołana $i raz(y) <br />";
$i++;
}
funk();
funk();
funk();
?>
```

W podanym przykładzie przy każdym wywołaniu funkcji `funk()` wartość zmiennej `$i` będzie przyjmowała wartość `1`, co spowoduje wyświetlenie tego samego komunikatu, mówiącego, że funkcja została wywołana jeden raz (rysunek 7.19).



Rysunek 7.19. Efekt zadeklarowania w treści funkcji zmiennej lokalnej

Jeżeli przy tworzeniu zmiennej zostanie zadeklarowane, że jest to zmienna statyczna, jej wartość zostanie zachowana między kolejnymi wywołaniami funkcji. Deklaracja zmiennej statycznej ma postać:

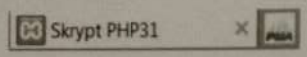
```
static $nazwa zmiennej = wartość;
```

Przykład 7.50

```
<?php
function funk()
{
static $i = 1;
```

```
echo "Funkcja wywołana $i raz(y) <br />";
$i++;
}
funk();
funk();
funk();
?>
```

W podanym przykładzie zmienna `$i` została zadeklarowana jako zmienna statyczna, dlatego przy pierwszym jej wywołaniu zostanie utworzona zmienna statyczna i przy opuszczaniu funkcji jej wartość zostanie zapamiętana. Przy ponownym wywołaniu funkcji instrukcja przypisująca początkową wartość zmiennej będzie ignorowana, a stosowana będzie wartość zapamiętana przy poprzednim wywołaniu funkcji (rysunek 7.20).



Funkcja wywołana 1 raz(y)
Funkcja wywołana 2 raz(y)
Funkcja wywołana 3 raz(y)

Rysunek 7.20. Efekt zadeklarowania w treści funkcji zmiennej statycznej

7.7.4. Argumenty funkcji

Argumenty do funkcji mogą być przekazywane na dwa sposoby:

- przez wartość,
- za pomocą referencji.

Argumenty przekazywane przez wartość

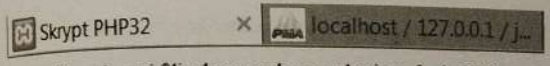
Domyślnym sposobem przekazywania argumentów do funkcji jest przekazywanie przez wartość. Ten sposób przekazywania był stosowany w podawanych do tej pory przykładach. W praktyce oznaczało to, że do funkcji przekazywane są kopie argumentów źródłowych i wszystkie operacje wykonywane są na kopiach. Zostanie to zilustrowane na przykładzie.

Przykład 7.51

```
<?php
function wart($liczba)
{
$liczba += 3;
}
$liczba =2;
```

```
echo "Wartość zmiennej \$liczba przed wywołaniem funkcji: $liczba <br />";
wart($liczba);
echo "Wartość zmiennej \$liczba po wywołaniu funkcji: $liczba <br />";
?>
```

W przykładzie została zdefiniowana funkcja `wart()` z argumentem `$liczba` (wartość przekazanej zmiennej jest zwiększana o 3). Przed jej wywołaniem zmienna `$liczba` została ustawiona na wartość 2 i wyświetlona w celu przetestowania jej stanu. Następnie została wywołana funkcja `wart()`. Po jej wykonaniu ponownie została wyświetlona zmienna `$liczba`, aby sprawdzić jej aktualny stan. Wszystkie operacje wewnątrz funkcji odbywały się na kopii zmiennej, natomiast wartość oryginalnej zmiennej nie uległa zmianie (rysunek 7.21).



Wartość zmiennej $liczba przed wywołaniem funkcji: 2
Wartość zmiennej $liczba po wywołaniu funkcji: 2

Rysunek 7.21. Wartość oryginalnej zmiennej nie ulega zmianie

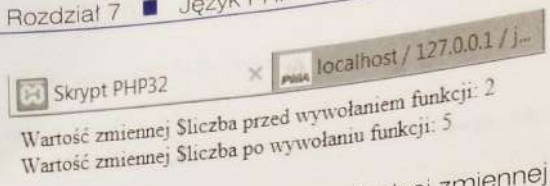
Argumenty przekazywane za pomocą referencji

Jeżeli modyfikowanie kopii przekazanego do funkcji argumentu nie wystarcza, można zastosować przekazywanie argumentów za pomocą referencji. Wtedy funkcja będzie modyfikowała argumenty oryginalne. Przy takim przekazywaniu argumentów przed przekazywanymi argumentami należy umieścić znak `&` (ampersand).

Przykład 7.52

```
<?php
function wart(&$liczba)
{
$liczba += 3;
}
$liczba =2;
echo "Wartość zmiennej \$liczba przed wywołaniem funkcji: $liczba <br />";
wart($liczba);
echo "Wartość zmiennej \$liczba po wywołaniu funkcji: $liczba <br />";
?>
```

W tym przykładzie wartość zmiennej `$liczba` będzie inna przed wywołaniem funkcji `wart()` i inna po jej wywołaniu (rysunek 7.22).



Rysunek 7.22. Wartość oryginalnej zmiennej ulega zmianie

Domyślne argumenty funkcji

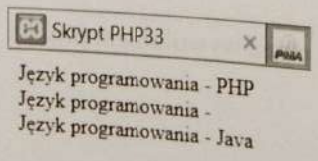
W języku PHP można tworzyć funkcje, których argumenty będą miały określone wartości domyślne. Mechanizm ten pozwala na wywołanie funkcji bez podawania jej argumentów. Jako argumenty zostaną wstawione wartości zdefiniowane podczas tworzenia funkcji. Jeżeli natomiast argumenty będą podane, funkcja zostanie wywołana z tymi argumentami. Definicja funkcji z argumentami domyślnymi ma postać:

```
function nazwa ($argument1 = wartość, $argument2 = wartość,...)
{
instrukcje
}
```

Przykład 7.53

```
<?php
function progr($język = "PHP")
{
return "Język programowania - $język";
}
echo progr();
echo "<br/>";
echo progr(null);
echo "<br/>";
echo progr("Java");
?>
```

Utworzona w przykładzie funkcja posiada zdefiniowany argument domyślny. Przy pierwszym wywołaniu funkcja zostanie wykonana z domyślną wartością argumentu, przy drugim wartość argumentu będzie `null`, przy trzecim wartość argumentu zostanie ustawiona zgodnie z argumentem wywołania funkcji (rysunek 7.23).

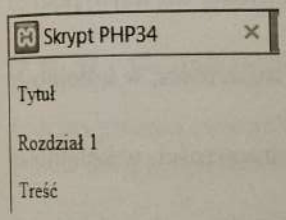


Rysunek 7.23. Wynik wywołania funkcji z różnymi argumentami

Przykład 7.54

```
<?php
function styl($tekst, $kolor ="red")
{
 echo "<p style = \"color:  $kolor\">" . $tekst . "</p>";
}
styl("Tytuł <br />", "blue");
styl("Rozdział 1. <br />", "green");
styl("Treść <br />");
?>
```

W podanym przykładzie funkcja `styl()` posiada dwa argumenty. Pierwszy argument `$tekst` przekazuje treść tekstu do wyświetlenia, drugi `$kolor` przekazuje kolor w jakim tekst powinien zostać wyświetlony (rysunek 7.24). Wartość domyślna argumentu `$kolor` została ustawiona na `red`. Jeżeli przy wywołaniu funkcji kolor tekstu nie zostanie określony, będzie on wyświetlany w kolorze przekazanym jako argument domyślny (`red`).



Rysunek 7.24. Wynik wywołania funkcji z różnymi argumentami

Wartości domyślne można przypisać dowolnej liczbie argumentów. Ale jeżeli wartość domyślna zostanie przypisana określonemu argumentowi, to wszystkie następne również muszą posiadać wartości domyślne.

7.8. Funkcje wbudowane

W języku PHP istnieje duża grupa predefiniowanych funkcji, które są przydatne podczas tworzenia aplikacji WWW. Funkcje te zostały podzielone na grupy tematyczne — ułatwia to znalezienie funkcji wykonującej określone zadania.

7.8.1. Funkcje tablic

Cała grupa funkcji wbudowanych dotyczy operacji wykonywanych na tablicach. Do najpopularniejszych należą `count()` i `sizeof()`, które zliczają liczbę elementów w tablicy.

Przykład 7.55

```
<?php
$tab = array("e1", "e2", "e3", "e4", "e5");
$dl = count($tab);
for($i =0; $i < $dl; $i++){
echo $tab[$i], " ";
}
?>
```

Funkcje sortowania

Sortowanie polega na uporządkowaniu elementów tablicy w określonym porządku, najczęściej w porządku rosnącym lub malejącym. Do sortowania tablic można wykorzystać kilka funkcji w zależności od tego, jakie sortowanie będzie realizowane oraz jakiego typu tablic będzie ono dotyczyło.

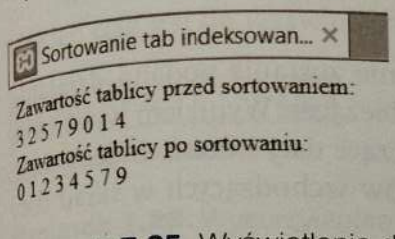
- `sort()` — sortuje elementy tablicy indeksowanej w kolejności od najmniejszego do największego.
- `rsort()` — sortuje elementy tablicy indeksowanej w kolejności od największego do najmniejszego.
- `asort()` — sortuje elementy tablicy asocjacyjnej według zawartości, w kolejności od najmniejszego do największego.
- `arsort()` — sortuje elementy tablicy asocjacyjnej według zawartości, w kolejności od największego do najmniejszego.
- `ksort()` — sortuje elementy tablicy asocjacyjnej według klucza, w kolejności od najmniejszego do największego.
- `krsort()` — sortuje elementy tablicy asocjacyjnej według klucza, w kolejności od największego do najmniejszego.

Przykład 7.56

```
<?php
$tab = array(3, 2, 5, 7, 9, 0, 1, 4);
echo "Zawartość tablicy przed sortowaniem: <br/>";
foreach($tab as $x){
echo "$x ";
}
sort($tab);
echo "<br/>Zawartość tablicy po sortowaniu: <br/>";
foreach($tab as $x){
echo "$x ";
```

```
}
?>
```

Wynik interpretacji kodu został pokazany na rysunku 7.25.



Rysunek 7.25. Wyświetlenie danych pobranych z tablicy

Funkcje wyszukiwania

`array_search()` — funkcja posiada dwa argumenty. Pierwszym argumentem funkcji jest poszukiwana wartość, drugim przeszukiwana tablica. Jeżeli szukana wartość zostanie znaleziona, funkcja zwróci wartość `true`, w przeciwnym razie zwróci wartość `false`.

Zadanie 7.7

Napisz skrypt obliczający, ile jest w tablicy liczb parzystych, a ile nieparzystych.

7.8.2. Funkcje daty i czasu

Podczas pisania skryptów często wymagane jest podanie daty lub czasu. Język PHP dysponuje dużą grupą funkcji, których zadaniem jest wykonywanie operacji na dacie i czasie.

Funkcja time()

Funkcja `time()` zwraca informacje na temat bieżącej daty i czasu. Nie posiada żadnych argumentów. Informacje na temat bieżącej daty i czasu są zwracane w postaci liczby. Liczba ta odpowiada liczbie sekund, które upłynęły od godziny 00:00:00 1 stycznia 1970 roku do bieżącej daty. Jest to tak zwany *`znacznik_czasu`* (`timestamp`).

Przykład 7.57

```
<?php
echo time();
?>
```

Taki zapis daty i czasu pozwala na wykonywanie działań na dacie, np. zwiększanie daty o 1 dzień, czasu o 1 godzinę, minutę lub sekundę. Wystarczy do znacznika czasu dodać wartość określającą liczbę sekund, jakie upłynęły, aby otrzymać interesujący nas wynik. Następnie za pomocą funkcji PHP można otrzymany wynik przekształcić na czytelniejszą postać.

Funkcja getdate()

Drugą funkcją związaną z przekazywaniem informacji o dacie i czasie jest funkcja `getdate()`. Ma ona postać:

```
getdate([znacznik_czasu])
```

Argument *znacznik_czasu* jest opcjonalny. Jeżeli nie zostanie podany, działania wykonywane przez funkcję będą się odnosić do daty bieżącej. Wynikiem wykonania funkcji jest tablica asocjacyjna zawierająca dane dotyczące daty i czasu. Indeksy oraz wartości tej tablicy dotyczą poszczególnych elementów wchodzących w skład daty i czasu. Zostały one przedstawione w tabeli 7.8.

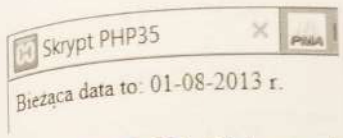
Tabela 7.8. Indeksy tablicy zwracanej przez funkcję getdate()

Nazwa indeksu	Znaczenie indeksu	Opis
seconds	Liczba sekund	od 0 do 59
minutes	Liczba minut	od 0 do 59
hours	Godzina	od 0 do 23
mday	Dzień miesiąca	od 1 do 31
wday	Dzień tygodnia w postaci liczby	od 0 (niedziela) do 6 (sobota)
mon	Miesiąc w postaci liczby	od 1 do 12
year	Rok w postaci czterocyfrowej	np. 2005
yday	Numer kolejnego dnia roku	od 0 do 365
weekday	Nazwa dnia tygodnia (po angielsku)	np. Monday
month	Nazwa miesiąca (po angielsku)	np. January
0	Aktualny znacznik czasu	

Przykład 7.58

```
<?php
$data = getdate();
$dzien = $data["mday"];
$miesiac = $data["mon"];
$rok = $data["year"];
if($dzien<10) $dzien = "0" . $dzien;
if($miesiac<10) $miesiac = "0" . $miesiac;
echo "Bieżąca data to: $dzien-$miesiac-$rok r.";
?>
```

W podanym przykładzie zmiennej `$data` została przypisana tablica zwrócona przez funkcję `getdate()`. Wartości indeksów `mday`, `mon`, `year` zostały przypisane zmiennym `$dzien`, `$miesiac`, `$rok`. Gdy wartości zmiennych `$dzien` i `$miesiac` będą mniejsze od `10`, zostanie do nich dodany znak `0`. Po połączeniu wszystkich zmiennych w jeden ciąg jest on wyświetlony instrukcją `echo` (rysunek 7.26).



Rysunek 7.26. Wykorzystanie funkcji daty do wyświetlenia bieżącej daty

Funkcja date()

Funkcją, która pozwala na formatowanie w odpowiedni sposób daty i czasu, jest funkcja `date()`. Ma ona postać:

```
date(format[, znacznik_czasu])
```

Parametr *`format`* określa, jakie informacje na temat daty i czasu powinny zostać zwrócone. Parametr *`znacznik_czasu`* zawiera opisany już znacznik czasu. Jest on opcjonalny i określa interesującą nas datę. Jeżeli zostanie pominięty, funkcja zwróci datę bieżącą.

Parametr *`format`* jest ciągiem znaków składającym się ze znaczników. Opis niektórych znaczników został podany w tabeli 7.9.

Tabela 7.9. Niektóre znaczniki formatujące funkcji date()

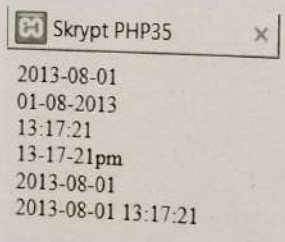
Znacznik	Znaczenie	Opis
a	Użycie określenia „przed południem” (am) lub „po południu” (pm)	am, pm
A	Użycie określenia „przed południem” (AM) lub „po południu” (PM)	AM, PM
c	Data i czas zgodne z formatem ISO 8601	2013-03-03T15:25:20
d	Dzień miesiąca w formacie z zerem na początku	od 01 do 31
D	Dzień miesiąca w formacie trzyliterowego skrótu	Mon, Tue
F	Pełna nazwa miesiąca	January
g	Godzina w formacie dwunastogodzinnym bez zera na początku	od 1 do 12
G	Godzina w formacie dwudziestoczterogodzinnym bez zera na początku	od 1 do 24
H	Godzina w formacie dwudziestoczterogodzinnym z zerem na początku	od 01 do 24

Znacznik	Znaczenie	Opis
i	Liczba minut z zerem na początku	od 01 do 59
l	Nazwa dnia tygodnia	Monday
m	Miesiąc w postaci liczby dwucyfrowej z zerem na początku	od 01 do 12
s	Liczba sekund z zerem na początku	od 01 do 59
Y	Rok w postaci czterech znaków	2013

Przykład 7.59

```
<?php
echo date("Y-m-d") . "<br/>";
echo date("d-m-Y") . "<br/>";
echo date("G:i:s") . "<br/>";
echo date("H-i-sa") . "<br/>";
echo date("Y-m-d") . "<br/>";
echo date("Y-m-d G:i:s") . "<br/>";
?>
```

Funkcja `date()` zwraca angielskie nazwy miesięcy i dni tygodnia. Przykład 7.59 pokazuje użycie tej funkcji z różnymi znacznikami formatującymi (rysunek 7.27).



Rysunek 7.27. Użycie funkcji date() z różnymi znacznikami formatującymi

Funkcja mktime()

Funkcja `mktime()` zwraca znacznik czasu daty podanej jako argument funkcji. Może posiadać od 0 do 6 argumentów podanych w postaci liczb całkowitych. Są to kolejno:

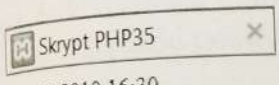
- godzina,
- minuta,
- sekunda,
- miesiąc,
- dzień miesiąca,
- rok.

Znacznik czasu, który zwróci funkcja `mktime()`, może zostać wykorzystany w funkcjach `getdate()` i `date()`.

Przykład 7.60

```
<?php
$czas = mktime(16, 30, 0, 12, 24, 2010);
echo date("d-m-Y G:i", $czas) . "<br/>";
echo date("Y-m-d G:i:s", $czas);
?>
```

Wynik interpretacji kodu został pokazany na rysunku 7.28.



Rysunek 7.28. Wykorzystanie funkcji mktime() do ustawienia znacznika czasu

Zadanie 7.8

Napisz skrypt, który na podstawie danych pobranych z tablicy zwracanej przez funkcję `getdate()` wyświetli bieżącą datę. W dacie zostanie podana nazwa miesiąca w postaci tekstu w języku polskim.

Zadanie 7.9

Napisz skrypt, który będzie wyświetlał bieżący dzień tygodnia w podanej postaci: *Dzisiaj jest środa*.

Zadanie 7.10

Napisz skrypt wyświetlający liczbę dni, które upłynęły od początku roku, oraz liczbę dni, które pozostały do końca roku.

7.8.3. Funkcje formatowania ciągów

Pracując z ciągami znakowymi, często trzeba formatować je tak, aby były wyświetlane w określony sposób. Do tego celu można wykorzystywać funkcje formatujące. W języku PHP istnieje wiele funkcji, za pomocą których możemy ustalić określony wygląd ciągu.

Funkcja nl2br()

Jeżeli wyświetlamy w przeglądarce blok tekstu, który zawiera znaki końca linii, to przeglądarka nie uwzględni tych znaków. Można ominąć ten problem, dodając znaczniki `<br>` lub `<p>`. Ale nie zawsze takie rozwiązanie jest możliwe, szczególnie wtedy, gdy tekst jest wczytywany z pliku lub bazy danych. W takiej sytuacji można wykorzystać funkcję:

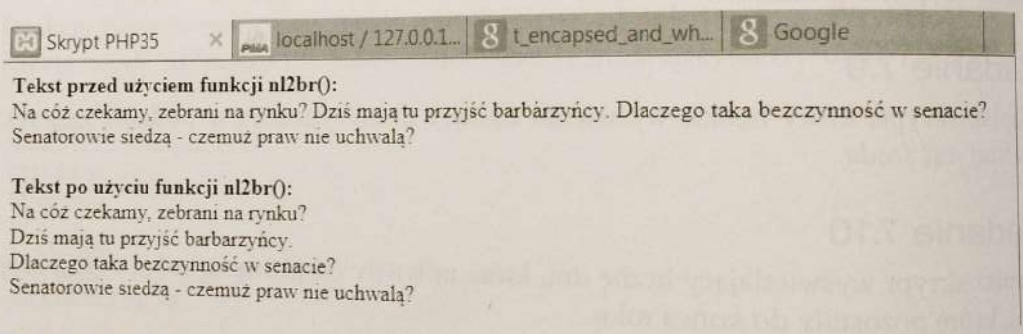
```
nl2br("ciąg_znaków")
```

Funkcja ta dla wybranego bloku tekstu przed każdym znakiem końca linii automatycznie wstawi znacznik <br/> i zwróci przetworzony tekst.

Przykład 7.61

```
<?php
$tekst = <<<TX
Na cóż czekamy, zebrani na rynku?
Dziś mają tu przyjść barbarzyńcy.
Dlaczego taka bezczynność w senacie?
Senatorowie siedzą - czemuż praw nie uchwalą?
TX;
echo "<b>Tekst przed użyciem funkcji nl2br():</b><br/>";
echo $tekst."<br/><br/>";
echo "<b>Tekst po użyciu funkcji nl2br():</b><br/>";
echo nl2br($tekst);
?>
```

Wynik interpretacji kodu został pokazany na rysunku 7.29.



Rysunek 7.29. Wynik zastosowania do tekstu funkcji nl2br()

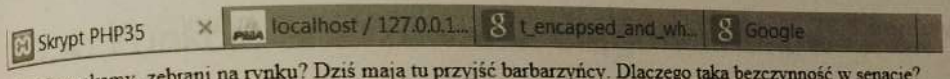
Funkcja wordwrap()

Do formatowania tekstu w postaci kolumny o określonej szerokości można wykorzystać funkcję `wordwrap()`. Dzieli ona ciąg podany jako argument na linie o maksymalnej długości 75 znaków. Do rozdzielenia linii domyślnie jest używany znak `\n`. Oprócz argumentu określającego ciąg źródłowy funkcja posiada jeszcze trzy opcjonalne argumenty. Są to: liczba określająca maksymalną długość linii, ciąg znaków zastosowany do rozdzielenia linii oraz argument podziału słów dłuższych niż zadeklarowana maksymalna długość linii.

Przykład 7.62

```
<?php
$tekst = <<<TX
Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego
taka bezczynność w senacie? Senatorowie siedzą - czemuż praw nie uchwalą?
TX;
echo wordwrap($tekst);
?>
```

W podanym przykładzie funkcja `wordwrap()` została wywołana z jednym argumentem, a więc linie zostaną rozdzielone znakiem `\n`, który nie jest interpretowany przez przeglądarki (rysunek 7.30). Po zastosowaniu dwóch kolejnych argumentów kod wysłany do przeglądarki pozwoli na wyświetlenie tekstu w liniach (rysunek 7.31).

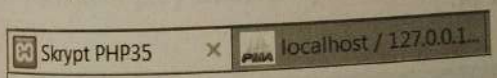


Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego taka bezczynność w senacie? Senatorowie siedzą - czemuż praw nie uchwalą?

Rysunek 7.30. Brak interpretacji przez przeglądarkę znaku nowej linii

Przykład 7.63

```
<?php
$tekst = <<<TX
Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego
taka bezczynność w senacie? Senatorowie siedzą - czemuż praw nie uchwalą?
TX;
echo wordwrap($tekst, 30, "<br/>\n");
?>
```



Na cóż czekamy, zebrani na
rynku? Dziś mają tu
przyjść barbarzyńcy.
Dlaczego taka bezczynność w
senacie? Senatorowie siedzą -
czemuż praw nie uchwalą?

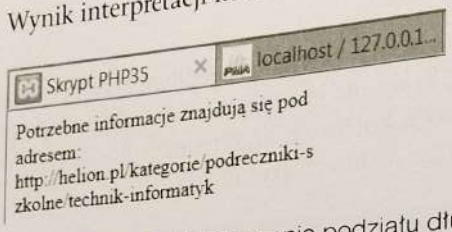
Rysunek 7.31. Podział tekstu na linie o długości 30 znaków

Ostatni, czwarty argument funkcji będzie zastosowany, gdy w linii pojawi się słowo dłuższe niż zadeklarowana maksymalna długość linii. Domyślnie takie słowo nie zostanie podzielone między liniami. Użycie czwartego argumentu wymusi podział słowa między liniami. Użycie tego argumentu polega na przypisaniu mu wartości logicznej `true`.

Przykład 7.64

```
<?php
$tekst = "Potrzebne informacje znajdują się pod adresem: ";
$tekst .= "http://helion.pl/kategorie/podreczniki-szkolne/technik-informatyk";
echo wordwrap($tekst, 40, "<br/>\n", true);
?>
```

Wynik interpretacji kodu został pokazany na rysunku 7.32.



Rysunek 7.32. Wymuszenie podziału długiego słowa między liniami

Funkcje zmiany wielkości liter

Częstym działaniem na ciągach znakowych jest zamiana wszystkich liter na duże lub małe. Do zamiany wszystkich liter na duże służy funkcja:

```
strtoupper(argument)
```

Ciąg podany w postaci argumentu zostanie zmodyfikowany w ten sposób, że wszystkie litery zostaną zamienione na duże.

Do zamiany wszystkich liter na małe służy funkcja:

```
strtolower(argument)
```

Ciąg podany w postaci argumentu zostanie zmodyfikowany w ten sposób, że wszystkie litery zostaną zamienione na małe.

Kolejnymi funkcjami o podobnym działaniu są:

```
ucfirst(argument)
```

i

```
ucwords(argument)
```

Funkcja `ucfirst()` zmodyfikuje podany argument w ten sposób, że pierwsza litera ciągu zostanie zamieniona na dużą literę.

Funkcja `ucwords()` zmodyfikuje ciąg podany jako argument w ten sposób, że wszystkie pierwsze litery wyrazów zostaną zamienione na duże litery.

Funkcje usuwania ciągu znaków

Do usunięcia z początku lub końca ciągu białych znaków można użyć jednej z trzech funkcji: `trim()`, `ltrim()`, `rtrim()`.

Wszystkie one mają taką samą konstrukcję:

```
nazwa funkcji("ciąg_znakowy")
```

Usuwają one następujące znaki: znak spacji (kod 32), znak tabulacji (kod 9), znak nowej linii (kod 10), znak powrotu karetki (kod 13), znak tabulacji pionowej (kod 11), znak o kodzie 0.

- `trim()` — usuwa podane znaki z początku i końca ciągu,
- `ltrim()` — usuwa podane znaki z początku ciągu,
- `rtrim()` — usuwa podane znaki z końca ciągu.

Do usuwania innych znaków należy użyć podanych wyżej funkcji z dodatkowym parametrem w postaci:

```
nazwa funkcji("ciąg_znakowy", "znaki do usunięcia")
```

7.8.4. Funkcje analizowania ciągów znaków

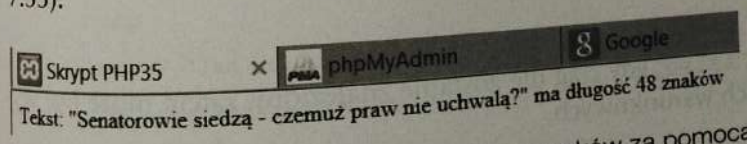
Funkcja sprawdzająca długość ciągu

Do sprawdzenia długości ciągu znaków jest wykorzystywana funkcja `strlen()`.

Przykład 7.65

```
<?php
$napis = "Senatorowie siedzą - czemuż praw nie uchwalą?";
echo 'Tekst: "'.$napis.'"';
$dl = strlen($napis);
echo " ma długość $dl znaków<br/>";
?>
```

W podanym przykładzie wyświetli się tekst oraz będzie podana jego długość (rysunek 7.33).



Rysunek 7.33. Określenie długości ciągu znaków za pomocą funkcji strlen()

Indeksowanie ciągu znaków

Z utworzonego ciągu znaków podobnie jak z tablicy można pobierać pojedyncze znaki. Każdy znak ciągu ma swoje położenie, które można określić poprzez indeks. Znak na pierwszej pozycji ma indeks 0, następny 1. Wartość indeksu dla kolejnych znaków rośnie o 1.

Przykład 7.66

```
<?php
$napis = "Senatorowie siedzą - czemuż praw nie uchwalą?";
echo $napis[0];
echo $napis[20];
?>
```

W podanym przykładzie dla zmiennej `$napis` w nawiasach kwadratowych został podany indeks znaku, który powinien być wyświetlony.

Znajdowanie podciągów

Funkcja strstr()

Do sprawdzenia, czy podany ciąg jest fragmentem innego ciągu, służy funkcja `strstr()` w postaci:

```
strstr(argument1, argument2)
```

Pierwszy argument jest przeszukiwanym ciągiem źródłowym, drugi poszukiwanym ciągiem. Funkcja zwraca wartość `false`, gdy podciąg nie został znaleziony. Jeżeli szukany ciąg jest fragmentem ciągu źródłowego, to funkcja zwraca fragment ciągu źródłowego od znalezionego podciągu do jego końca. Funkcja `strstr()` rozróżnia wielkość liter. Jeżeli wielkość liter nie powinna mieć znaczenia, podczas przeszukiwania ciągu źródłowego należy użyć funkcji `stristr()`. Działa ona podobnie jak funkcja `strstr()`, ale nie rozróżnia wielkości liter.

Przykład 7.67

```
<?php
$dane = "Jan Kowalski, ul. Długa 23, 80-874 Gdańsk, tel. 693341678";
$tel = strstr($dane, "tel.");
echo $tel;
?>
```

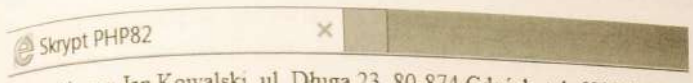
Funkcja zwraca wartość `false`, gdy ciąg nie zostanie znaleziony, zatem może być wykorzystana w instrukcjach warunkowych.

Przykład 7.68

```
<?php
$dane = "Jan Kowalski, ul. Długa 23, 80-874 Gdańsk, tel. 693341678";
echo "Ciąg główny:".$dane."</br>";
$tel = strstr($dane, "tel.");
if($tel == false)
```

```
echo "Brak numeru telefonu";
else
echo "Znaleziony podciąg: ".$tel;?>
```

Jeżeli w ciągu występuje podciąg `tel.`, wynikiem będzie wyświetlenie podciągu z numerem telefonu (rysunek 7.34), w przeciwnym razie wyświetli się komunikat o braku numeru telefonu.



Rysunek 7.34. Znaleziony podciąg w ciągu głównym

Funkcja strpos()

Podobne działanie do funkcji `strstr()` ma funkcja `strpos()`. Funkcja ta sprawdza, czy szukany podciąg znajduje się w ciągu źródłowym, a jeżeli tak, zwraca jego położenie. Posiada trzy argumenty: pierwszy to przeszukiwany ciąg źródłowy, drugi to poszukiwany podciąg, trzeci to pozycja w ciągu źródłowym, od którego rozpocznie się przeszukiwanie. Trzeci argument jest opcjonalny. Jeżeli podciąg nie zostanie znaleziony, zwrócona będzie wartość `false`. Gdy podciąg zostanie znaleziony, funkcja zwróci indeks określający miejsce znalezienia podciągu. Indeksowanie pozycji w ciągu rozpoczyna się od `0`.

Funkcja substr()

Funkcja `substr()` zwraca część ciągu źródłowego. Posiada trzy argumenty: pierwszy to ciąg źródłowy, drugi to indeks określający miejsce początku zwracanego podciągu, trzeci, opcjonalny, określa długość pobranego ciągu. Jeżeli zostanie podany trzeci argument, zwrócona będzie określona liczba znaków. Gdy ten argument zostanie pominięty zostanie zwrócony podciąg od wskazanego indeksu do końca ciągu.

Przykład 7.69

```
<?php
$dane = "Jan Kowalski, ul. Długa 23, 80-874 Gdańsk, tel. 693341678";
echo substr($dane, 4, 8);
?>
```

Jeżeli jako drugi argument funkcji (indeks) zostanie podana wartość ujemna, pozycja indeksu będzie liczona od końca ciągu źródłowego.

Przykład 7.70

```
<?php
$dane = "Jan Kowalski, ul. Długa 23, 80-874 Gdańsk, tel. 693341678";
echo substr($dane, 4, 8)."<br/>";
```

```
echo substr($dane, -0, 14);
?>
```

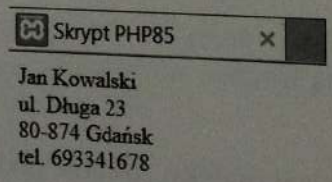
Funkcja strtok()

Za pomocą funkcji `strtok()` można podzielić ciąg źródłowy na poszczególne podciągi. Funkcja posiada dwa argumenty: pierwszy to ciąg źródłowy, który zostanie podzielony, drugi to ciąg znaków rozdzielających podciągi. Funkcja ta przy pierwszym wywołaniu zwraca pierwszy wydzielony ciąg i zapamiętuje przeszukiwany ciąg w pamięci podręcznej. Przy kolejnym wywołaniu funkcja zwraca kolejne ciągi. Jeżeli zostanie osiągnięty koniec ciągu źródłowego, funkcja zwróci wartość `false`. Z powodu takiego działania funkcja ta najczęściej jest wywoływana w pętli.

Przykład 7.71

```
<?php
$dane = "Jan Kowalski, ul. Długa 23, 80-874 Gdańsk, tel. 693341678";
$znak = ",";
$ciag = strtok($dane, $znak);
while (is_string($ciag)){
if ($ciag){
echo "$ciag<br/>";
}
$ciag = strtok($znak);
}
?>
```

Użyta w przykładzie funkcja `is_string()` sprawdza, czy typ podanej jako argument zmiennej jest ciągiem. Wywołana drugi raz funkcja `strtok()` nie zawiera ciągu źródłowego. Podanie tego argumentu spowodowałoby przeszukiwanie ciągu od początku. Wynik interpretacji kodu został pokazany na rysunku 7.35.



Rysunek 7.35. Wynik podzielenia ciągu głównego. Znakiem rozdzielającym jest przecinek

Porównania ciągów

Działaniem niezbędnym podczas pracy z ciągami jest ich porównywanie. Porównywanie ciągów znakowych można wykonywać za pomocą operatorów porównania lub funkcji porównujących.

Funkcja strcmp

Funkcja `strcmp()` ma postać:

```
strcmp("ciąg1", "ciąg2")
```

- zwraca wartość mniejszą od 0, jeżeli *ciąg1* jest mniejszy od *ciąg2*.
- zwraca wartość większą od 0, jeżeli *ciąg1* jest większy od *ciąg2*.
- zwraca 0, jeżeli ciągi są równe.

Funkcja rozróżnia wielkość znaków.

Funkcja strcasecmp

Funkcja `strcasecmp()` działa jak funkcja `strcmp()`, ale nie uwzględnia wielkości liter. Ma postać:

```
strcasecmp("ciąg1", "ciąg2")
```

Zadanie 7.11

Napisz skrypt, który w podanym ciągu będzie wyszukiwał określony wyraz i obliczał, ile razy ten wyraz wystąpił w podanym ciągu.

Zadanie 7.12

Napisz skrypt, który będzie zawierał tablicę wybranych wyrazów oraz ciąg źródłowy. Każde wystąpienie w tym ciągu wyrazu z tablicy powinno spowodować jego zamianę na tekst *wyraz z tablicy*.

7.9. Pliki i katalogi

W języku PHP istnieją funkcje umożliwiające przeprowadzanie różnych operacji na plikach. Umożliwiają one odczytywanie informacji o strukturze plików i katalogów oraz odczytywanie i zapisywanie danych do plików.

7.9.1. Dołączanie plików

Jeżeli skrypt PHP zawiera dużą ilość kodu, to można go podzielić i zapisać w kilku oddzielnych plikach. Do ponownego ich połączenia można użyć instrukcji `include` lub `require`. Obydwie instrukcje w miejscu wystąpienia wstawiają zawartość wskazanego pliku.

Dołączony skrypt zostanie wykonany tak, jakby był częścią kodu, w którym został wstawiony. Instrukcja `include` ma postać:

```
include('nazwa_pliku')
```

lub

```
include 'nazwa_pliku'
```

Instrukcja `require` ma postać:

```
require ('nazwa_ pliku')
```

lub

```
require 'nazwa_ pliku'
```

Przykład 7.72

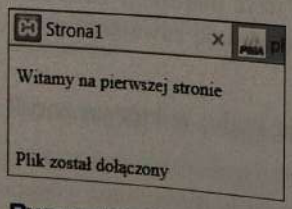
skrypt1.php

```
<?php
echo "<p>Plik został dołączony</p>";
?>
```

strona1.php

```
<!DOCTYPE HTML>
<html>
<head>
<title>Strona1</title>
</head>
<body>
<p>Witamy na pierwszej stronie</p>
<?php
include('skrypt1.php');
?>
</body>
</html>
```

Jeżeli obydwa pliki zostaną umieszczone w tym samym folderze, to do uruchomionego pliku *strona1.php* zostanie dołączona zawartość pliku *skrypt1.php* (rysunek 7.36). Ponieważ instrukcja `include()` jest częścią języka PHP, to w kodzie HTML został użyty znacznik `<?php`, aby przełączyć się na ten tryb. Po wykonaniu instrukcji tryb PHP zostaje wyłączony i dalsza część skryptu będzie przetwarzana jako kod HTML.



Rysunek 7.36. Wynik połączenia kodu z plików strona1.php oraz skrypt1.php

Różnica między instrukcjami `include()` i `require()` występuje tylko wtedy, gdy dołączany plik nie może zostać odczytany. Instrukcja `include()` wygeneruje ostrzeżenie,

ale skrypt zawierający jej wywołanie będzie nadal działał. Natomiast użycie instrukcji `require()` spowoduje zgłoszenie błędu i zakończenie działania skryptu.

Najprostsza lokalizacja dołączanego pliku jest realizowana przez podanie ścieżki dostępu do pliku (względnej lub bezwzględnej).

Przykład 7.73

```
include('./skrypty/skrypt1.php');
include('/moja_strona/skrypty/skrypt1.php');
```

W pierwszym przypadku dołączony zostanie plik znajdujący się w podkatalogu *skrypty* katalogu bieżącego. W drugim przypadku dołączony zostanie plik znajdujący się w podkatalogu *moja_strona/skrypty* katalogu głównego.

Dołączone pliki mogą zwracać wartość tak jak funkcje. Wywołanie instrukcji `return` wewnątrz dołączonego kodu kończy jego działanie.

7.9.2. Operacje na plikach

Czy plik istnieje

Do ustalenia, czy plik lub katalog istnieje, służy funkcja `file_exists()` zapisywana w postaci:

```
file_exists('nazwa_pliku')
```

Jako argument funkcji występuje nazwa pliku lub katalogu wraz ze ścieżką dostępu. Jeśli plik istnieje, funkcja zwraca wartość `true`, w przeciwnym razie zwraca wartość `false`.

Przykład 7.74

```
if (file_exists('skrypt.php')) {
echo "Plik został znaleziony.";
}
```

Funkcja `is_file()` sprawdza, czy podany jako jej argument ciąg jest plikiem. Ma postać:

```
is_file(plik)
```

Przykład 7.75

```
<?php
$p ='plik.txt';
if(is_file($p)) {
echo "To jest plik";
}
?>
```

Funkcja zwraca wartość `true`, jeżeli argument funkcji jest plikiem.

Rozmiar pliku

Do określenia rozmiaru pliku służy funkcja `filesize()` w postaci:

```
filesize('nazwa_pliku')
```

Funkcja zwraca wartość typu `integer`, która określa wielkość pliku w bajtach. Znajomość rozmiaru pliku jest potrzebna na przykład wtedy, gdy plik ma zostać dołączony jako załącznik do poczty elektronicznej lub gdy rozmiar pliku musi zostać wyświetlony, zanim zacznie się jego pobieranie.

Tworzenie i usuwanie pliku

Do tworzenia pliku używana jest funkcja `touch()` zapisywana w postaci:

```
touch("nazwa_pliku")
```

Funkcja tworzy pusty plik o podanej nazwie. Jeżeli istnieje już plik o podanej nazwie, nie ulegnie on zmianie. Zmieniona zostanie tylko data jego modyfikacji.

Do usuwania istniejącego pliku używana jest funkcja `unlink()` w postaci:

```
unlink('nazwa_pliku')
```

Funkcja zwraca wartość `true`, jeżeli plik został usunięty. W przeciwnym razie zwraca wartość `false`.

Otwieranie, zapisywanie i zamykanie pliku

Za pomocą funkcji `fopen()` można otwierać istniejące pliki. Funkcja ma postać:

```
fopen('nazwa_pliku', 'tryb_otwarcia')
```

Pierwszy argument funkcji określa ścieżkę do pliku, który ma zostać otwarty. Drugi określa tryb, w jakim plik zostanie otwarty. Najczęściej stosowane tryby otwarcia to:

- `r` — plik zostanie otwarty w trybie tylko do odczytu,
- `w` — plik zostanie otwarty w trybie tylko do zapisu,
- `a` — plik zostanie otwarty w trybie dopisywania.

Przykład 7.76

```
$p = fopen('dane.txt', 'r');
```

Plik zostanie otwarty tylko do odczytu.

Funkcja `fopen()` zwraca wartość `false`, jeżeli plik nie może zostać otwarty.

Po odpowiednim ustawieniu atrybutu *`tryb_otwarcia`* funkcja ta może zostać wykorzystana do zapisywania lub dopisywania nowych danych do pliku.

Przykład 7.77

```
$p = fopen('dane.txt', 'w');
```

Po wykonaniu podanego kodu znajdujące się w pliku dane zostaną usunięte, a nowe dane zostaną zapisane na początku pliku. Jeżeli podany plik nie istnieje, zostanie utworzony.

Przykład 7.78

```
$p = fopen('dane.txt', 'a');
```

Po wykonaniu podanego kodu nowe dane zostaną dopisane na końcu pliku.

Do zamykania pliku służy funkcja `fclose()` zapisywana w postaci:

```
fclose(deskryptor)
```

Deskryptor to wartość zwrócona przez funkcję `fopen()`. Gdy zakończą się operacje wykonywane na otwartym pliku, powinien on zostać zamknięty. Jeżeli nie zostanie to zrobione w skrypcie, plik zostanie zamknięty, gdy skrypt zakończy działanie.

Przykład 7.79

```
if($p = fopen('dane.txt', 'w')) {
//wykonanie działań na danych
fclose($p);
}
```

Kolejną funkcją stosowaną do zapisywania pliku jest funkcja `fwrite()` w postaci:

```
fwrite(deskryptor, ciąg_znaków)
```

Pierwszy argument oznacza plik zwrócony za pomocą funkcji `fopen()`, drugi to ciąg znaków, które mają zostać zapisane. Funkcja zwraca wartość `false`, jeżeli zapisanie ciągu znaków do pliku się nie powiodło.

Przykład 7.80

```
<?php
$tekst = "Barbarzyńcy, gdy przyjdą, ustanowią prawa.\n";
if(!$p = fopen('dane.txt', 'a')) {
echo "Nie można otworzyć pliku dane.txt";
}
else {
if(fwrite($p, $tekst) === false) {
echo "Zapis do pliku nie powiódł się";
}
else {
echo $tekst;
}
```

```
fclose($p);
}
?>
```

W podanym przykładzie tekst umieszczony w zmiennej `$tekst` będzie dopisywany na końcu pliku *dane.txt* przy każdym uruchomieniu skryptu (rysunek 7.37). Jeżeli plik nie istnieje, zostanie utworzony przy pierwszym wywołaniu.

dane.txt

```
1 Na cóż czekamy, zebrani na rynku?
2 Dziś mają tu przyjść barbarzyńcy.
3 Dlaczego taka bezczynność w senacie?
4 Senatorowie siedzą - czemuż praw nie uchwalą?
5
6 Barbarzyńcy, gdy przyjdą, ustanowią prawa.
7 Barbarzyńcy, gdy przyjdą, ustanowią prawa.
8
```

Rysunek 7.37. Zawartość pliku dane.txt po dwukrotnym wykonaniu skryptu

Odczyt danych

Dane z pliku mogą być odczytywane na wiele sposobów. W języku PHP istnieje wiele funkcji, które umożliwiają odczyt pojedynczych znaków, całych wierszy lub wybranych fragmentów pliku.

Po otwarciu pliku można odczytywać pojedyncze wiersze za pomocą funkcji `fgets()`. Ma ona postać:

```
fgets(deskryptor, ile_znaków)
```

Pierwszy argument oznacza plik otwarty za pomocą funkcji `fopen()`, drugi określa maksymalną liczbę znaków, które można odczytać. Funkcja zwraca odczytany ciąg znaków.

Funkcja `fgets()` często jest stosowana razem z funkcją `feof()`, która służy do testowania osiągnięcia końca pliku. Zwraca ona wartość `true`, jeśli został osiągnięty koniec pliku. Funkcja `feof()` ma postać:

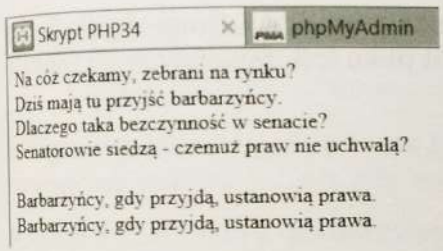
```
feof(deskryptor)
```

Przykład 7.81

```
<?php
if(!$p = fopen('dane.txt', 'r')) {
echo "Nie można otworzyć pliku dane.txt";
}
else {
while(!feof($p)) {
$w = fgets($p,100);
```

```
echo "$w<br/>";
}
fclose($p);
}
?>
```

W podanym przykładzie za pomocą funkcji `fopen()` jest otwierany tylko do odczytu plik *dane.txt*. Gdy otwarcie pliku się nie powiedzie (funkcja zwróci `false`), skrypt wyświetli komunikat i zakończy działanie. Gdy plik zostanie otwarty, w pętli będą odczytywane kolejne wiersze tekstu, aż do osiągnięcia końca pliku (rysunek 7.38). Przy każdym wykonaniu pętli sprawdzana jest wartość zwracana przez funkcję `feof()`. Gdy zostanie osiągnięty koniec pliku (funkcja `feof()` zwróci wartość `true`), pętla zakończy działanie. Wewnątrz pętli za pomocą funkcji `fgets()` pobierane są kolejne wiersze i są one przypisywane do zmiennej `$w`. Następnie zmienna ta jest wyświetlana z dodanym znacznikiem `<br/>`.



Rysunek 7.38. Odczyt pliku dane.txt za pomocą funkcji fgets()

Do odczytywania pojedynczych znaków służy funkcja `fgetc()`, zapisana w postaci:

```
fgetc(deskryptor)
```

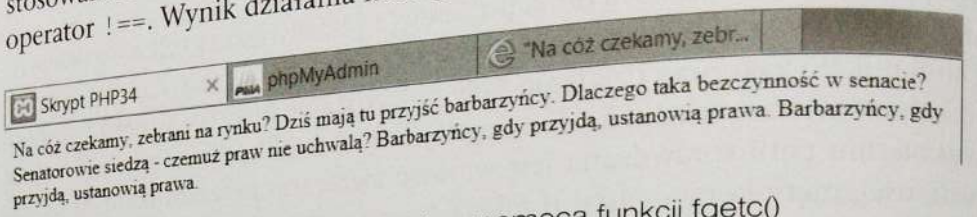
Funkcja zwraca ciąg zawierający jeden znak. Po wykonaniu funkcji wskaźnik pliku jest przesuwany o 1 znak do przodu. Gdy jest osiągnięty koniec pliku, funkcja zwraca wartość `false`.

Przykład 7.82

```
<?php
if(!$p = fopen('dane.txt', 'r')) {
echo "Nie można otworzyć pliku dane.txt";
}
else {
while(($z=fgetc($p))!==false) {
echo $z;
}
fclose($p);
```

```
}
?>
```

Skrypt przedstawiony w tym przykładzie jest bardzo podobny do skryptu z poprzedniego przykładu. Ponieważ funkcja `fgetc()` sama rozpoznaje koniec pliku, nie ma potrzeby stosowania funkcji `feof()`. Do testowania osiągnięcia końca pliku wykorzystany został operator `!==`. Wynik działania kodu pokazano na rysunku 7.39.



Rysunek 7.39. Odczyt pliku dane.txt za pomocą funkcji fgetc()

Do odczytu bloków danych służy funkcja `fread()`, przyjmująca postać:

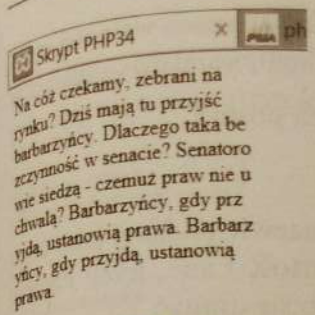
```
fread(deskryptor, ile_znaków)
```

Pierwszy argument funkcji to plik otwarty przez funkcję `fopen()`, drugi określa liczbę znaków, które należy odczytać. Odczytany fragment pliku jest zwracany przez funkcję w postaci ciągu znaków.

Przykład 7.83

```
<?php
if(!$p = fopen('dane.txt', 'r')) {
echo "Nie można otworzyć pliku dane.txt";
}
else {
while(!feof($p)) {
$b = fread($p, 32);
echo "$b<br/>";
}
fclose($p);
}
?>
```

W podanym przykładzie dane są odczytywane w blokach 32-bajtowych i wysyłane do przeglądarki (rysunek 7.40).



Rysunek 7.40. Odczyt pliku dane.txt za pomocą funkcji fread()

Kolejna funkcja służąca do odczytu danych z pliku to `readfile()`, zapisywana w postaci:

```
readfile('nazwa_pliku')
```

Wysyła ona zawartość pliku podanego jako argument do przeglądarki. Zwraca jako wartość liczbę odczytanych bajtów lub wartość `false`, gdy wykonanie operacji się nie powiodło.

Podobnie działa funkcja `file_get_contents()`, zapisana w postaci:

```
file_get_contents('nazwa_pliku')
```

Zwraca ona odczytaną zawartość pliku podanego jako argument w postaci ciągu tekstowego lub wartość `false`, gdy wykonanie operacji się nie powiodło.

Funkcja `file()` służy do odczytywania zawartości plików tekstowych. Zapisywana jest w postaci:

```
file('nazwa_pliku')
```

Funkcja odczytuje całą zawartość pliku i zwraca tę zawartość w postaci tablicy. Każda komórka tablicy zawiera kolejny wiersz odczytanego tekstu. Argumentem jest nazwa odczytywanego pliku. Funkcja może posiadać drugi opcjonalny argument, który może przyjmować jedną z wartości:

- `FILE_USE_INCLUDE_PATH` — po nazwie opcji powinny zostać wymienione katalogi, które zostaną przeszukane, jeżeli wskazanego pliku nie będzie w bieżącym katalogu.
- `FILE_IGNORE_NEW_LINES` — opcja ignoruje znaki końca linii.
- `FILE_SKIP_EMPTY_LINES` — opcja ignoruje puste linie.

7.9.3. Operacje na katalogach

Język PHP został wyposażony w cały zestaw funkcji, które umożliwiają wykonywanie różnych operacji na strukturze katalogów.

Za pomocą funkcji `mkdir()` można tworzyć nowe katalogi. Ma ona postać:

```
mkdir('nazwa_katalogu')
```

Argumentem funkcji jest nazwa tworzonego katalogu. Funkcja zwraca wartość `true`, jeżeli katalog został utworzony, lub wartość `false`, jeżeli katalogu nie udało się utworzyć.

Katalog usunąć można za pomocą funkcji `rmdir()`. Ma ona postać:

```
rmdir('nazwa_katalogu')
```

W wyniku wykonania funkcji zostanie usunięty katalog o nazwie podanej jako argument, pod warunkiem że jest on pusty. Funkcja zwraca wartość `true`, jeżeli katalog został usunięty, lub wartość `false`, jeżeli katalogu nie udało się usunąć.

Do otwierania katalogu służy funkcja `opendir()`, zapisana w postaci:

```
opendir('nazwa_katalogu')
```

Funkcja zwraca deskryptor katalogu lub wartość `false`, jeżeli katalogu o podanej nazwie nie uda się otworzyć.

Do odczytu zawartości katalogu stosowana jest funkcja `readdir()`, zapisana w postaci:

```
readdir(deskryptor)
```

Argumentem funkcji jest deskryptor uzyskany w wyniku wykonania funkcji `opendir()`. Każde wywołanie funkcji powoduje zwrócenie nazwy kolejnego elementu znajdującego się w katalogu. Po osiągnięciu końca katalogu funkcja zwróci wartość `false`.

Do zamykania katalogu służy funkcja `closedir()`, zapisana w postaci:

```
closedir(deskryptor)
```

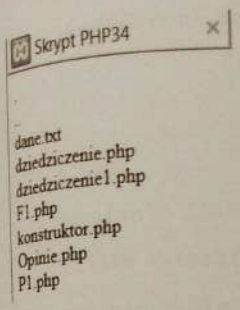
Funkcja nie zwraca żadnej wartości. Jeżeli nie zostanie użyta, zamknięcie katalogu następuje po zakończeniu działania skryptu. Jeżeli funkcja zostanie wywołana bez podania argumentu, zostanie zamknięty ostatnio otwarty katalog.

Przykład 7.84

```
<?php
$katalog = "./";
if($deskr = opendir($katalog)) {
while(($plik = readdir($deskr)) !==false)
echo "$plik<br/>";
closedir(deskr);
}
else {
echo "Nie można otworzyć katalogu";
}
?>
```

W podanym przykładzie ustawiona została zmienna `$katalog` zawierająca nazwę katalogu, którego zawartość należy odczytać (w przykładzie jest to katalog bieżący).

Następnie sprawdzana jest możliwość otwarcia katalogu i jeżeli operacja przebiegła pomyślnie, za pomocą pętli `while` wyświetlane są nazwy wszystkich elementów umieszczonych we wskazanym katalogu (rysunek 7.41).



Rysunek 7.41. Odczyt zawartości wskazanego katalogu

Do odczytu zawartości katalogu może również zostać wykorzystana funkcja `scandir()`, która pobiera zawartość całego katalogu i zapisuje ją w tablicy. Dodatkowo sortuje odczytane nazwy plików i folderów. Funkcja ma postać:

```
scandir('nazwa_katalogu')
```

Przykład 7.85

```
<?php
$katalog = "./";
$tablica = scandir($katalog);
foreach($tablica as $plik) {
echo "$plik<br/>";
}
?>
```

Zarówno funkcja `readdir()`, jak i funkcja `scandir()`, odczytując zawartość wskazanego katalogu, nie rozróżnia w nim plików i podkatalogów.

Zadanie 7.13

Napisz skrypt, który będzie odczytywał zawartość wskazanego katalogu, a odczytane elementy podzieli na katalogi i pliki, i wyświetli je w kolejności: wszystkie katalogi, następnie wszystkie pliki. W skrypcie wykorzystaj funkcje `is_file()` oraz `is_dir()`.

7.9.4. Praktyczne zastosowanie operacji na plikach

Poznane mechanizmy pracy z plikami zostaną wykorzystane do dodawania przez użytkowników na stronie internetowej komentarzy na wybrany temat. Utworzony skrypt będzie zapisywał opinie użytkowników w pliku tekstowym *opinie.txt*. Na stronie wyświetli się formularz, który pozwoli na wpisanie opinii. Nowa opinia zostanie

umieszczona na końcu pliku. Dotychczasowe opinie zapisane w pliku tekstowym będą wyświetlane na stronie i będą dostępne dla innych użytkowników witryny.

Przykład 7.86

skrypt1.php

```
<?php
if(isSet($_POST['komentarz'])){
$tekst = substr($_POST['komentarz'],0,255);
$tekst = strip_tags($tekst)."\n";

if(!$op = fopen('opinie.txt', 'a')) {
echo "Błąd!. Nie można otworzyć pliku opinie.txt";
}
else {
if(fwrite($op, $tekst) === false) {
echo "Dodanie komentarza nie powiodło się";
}
}
}
?>

<html>
<head>
<title>Opinie użytkowników</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body >

<div>
<form action="http://localhost/skrypt1.php" method="post">
<p><b>Dodaj swój komentarz na temat globalnego ocieplenia</b><br/>
(Maksymalnie 255 znaków)</p>

<textarea name="komentarz" rows="6" cols="50" wrap="virtual"></textarea><br/>
<input type="submit" value="Wyślij">
```

```
</div>
</form>

<p><b>Dodane opinie:</b></p><br/>
<div>

<?php
$opinie = '';
if(file_exists('./opinie.txt')) {
$opinie = file_get_contents('./opinie.txt');
$opinie = nl2br($opinie);
}
if($opinie != '') {
echo $opinie;
}
else {
echo "Brak opinii na temat globalnego ocieplenia.";
}
?>
</div>
</body>
</html>
```

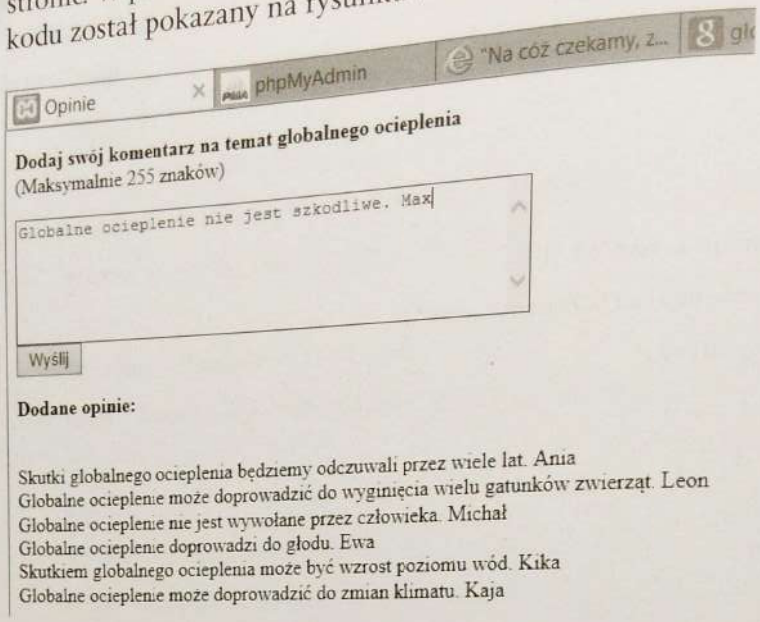
W pierwszej części kodu sprawdzane jest, czy za pomocą metody `POST` został przesłany z formularza parametr `komentarz` zawierający treść opinii użytkownika. Jeżeli tak, to odczytywana jest wartość tego parametru i zapisywana jest w zmiennej `$tekst`. Funkcja `substr()` skróci jego długość do 255 znaków. Następnie za pomocą funkcji `strip_tags()` zostaną usunięte ze zmiennej `$tekst` znaczniki PHP i HTML (jeżeli się takie pojawią).

Kolejny blok instrukcji dołącza wartość zmiennej `$tekst` do pliku *opinie.txt*. Jeżeli pliku nie można otworzyć (`if(!$op = fopen)`) lub nie można dołączyć do niego nowej treści (`if(fwrite)`), zostaną wyświetlone odpowiednie komunikaty.

Następny blok to kod HTML tworzący formularz do wprowadzania tekstu za pomocą znacznika `<textarea>`. Dane z formularza będą przekazywane za pomocą metody POST.

Ostatni blok to ponownie kod PHP, który spowoduje odczytanie wszystkich opinii umieszczonych w pliku *opinie.txt* i wyświetli je na stronie. Sprawdzane jest, czy plik istnieje, a jeżeli tak, to odczytywana treść jest zapisywana w zmiennej `$opinie`.

Następnie za pomocą funkcji `nl2br()` występujące znaki końca linii są zamieniane na znaczniki `<br/>`. Jeżeli w pliku były zapisane opinie, to ich treść wyświetli się na stronie. W przeciwnym razie pojawi się komunikat o braku opinii. Wynik interpretacji kodu został pokazany na rysunku 7.42.



Rysunek 7.42. Przykład wykorzystania mechanizmów pracy z plikami

7.9.5. Funkcje wyjścia

Do natychmiastowego zakończenia wykonywania skryptu stosowane są instrukcje: `exit()` i `die()`. Mogą występować w postaci:

```
exit([argument])

die([argument])
```

Instrukcje `exit()` i `die()` są elementami składni języka i mogą występować w postaci bez nawiasów, jeżeli nie posiadają argumentów. Argumentem może być komunikat, który wyświetli się przed wyjściem ze skryptu, lub liczba całkowita z przedziału od `0` do `254`. Jeżeli jest to liczba, określa ona stan wyjścia i nie jest wyświetlana. Stan równy `0` oznacza, że program w sposób prawidłowy zakończył działanie. Stan równy `255` jest zarezerwowany dla PHP i nie powinien być używany.

Przykład 7.87

```
<?php

if(!$p = fopen('dane.txt', 'r')) {

exit("Nie można otworzyć pliku dane.txt");

}

else {
```

```
//wykonanie działań na danych
}
?>
```

Podany kod sprawdza, czy plik *dane.txt* został otwarty. Jeżeli nie, wyświetla komunikat i kończy działanie.

7.10. Obiekty

Podstawowym elementem programowania obiektowego jest klasa. Klasa zawiera zmienne (właściwości) i funkcje (metody). Obiekt jest wystąpieniem (instancją) klasy. Jest tworzony na podstawie klasy.

W PHP występują takie struktury, jak klasa i obiekt, które umożliwiają tworzenie kodu w sposób obiektowy, ale język nie wymusza programowania obiektowego.

Zalety programowania obiektowego to:

- przejrzystość kodu programu,
- elastyczność kodu,
- łatwość rozbudowy aplikacji,
- możliwość powtórnego wykorzystania kodu.

7.10.1. Definiowanie klasy i obiektu

Do definiowania klasy służy polecenie `class` w postaci:

```
class nazwa_klasy
{
//definicja klasy
}
```

Definicja klasy zawiera definicje zmiennych (właściwości) oraz funkcji (metod). Te elementy klasy nazywane są **składowymi klasy**.

Nazwa klasy może zawierać litery (w tym znaki narodowe), cyfry oraz znak podkreślenia, ale nie może zaczynać się od cyfry. Jako nazwy klasy nie można używać słów kluczowych języka PHP.

Najprostsza definicja klasy ma postać:

```
class nowa
{
}
```

Tak zdefiniowana klasa nie zawiera żadnych zmiennych ani funkcji, ale jest poprawna. Można tworzyć dla niej obiekty.

Polecenie tworzące obiekt ma postać:

```
$nazwa_zmiennej = new nazwa_klasy();
```

Polecenie tworzenia obiektu klasy nowa będzie miało postać:

```
$obiekt_1 = new nowa();
```

7.10.2. Składowe klasy — właściwości i metody

Definicja właściwości klasy ma postać:

```
class nazwa_klasy
{
$nazwa_właściwości1;
$nazwa_właściwości2;
...
}
```

lub

```
class nazwa_klasy
{
public $nazwa_właściwości1, $nazwa_właściwości2,...
}
```

Metody klasy definiuje się podobnie jak zwykłe funkcje, używając słowa kluczowego `function`. Definicja metody klasy ma postać:

```
class nazwa_klasy
{
   function nazwa_metody (argumenty)
   {
      //treść metody;
   }
}
```

Przykład 7.88

Zostanie utworzona prosta klasa o nazwie `osoba`, która będzie obsługiwała dane osobowe i będzie zawierała następujące właściwości: nazwisko, imię, identyfikator osoby:

```
<?php
class osoba
{
public $id, $nazwisko, $imię;
```

```
}
?>
```

Do tak utworzonej klasy można dodać metody:

- `wpiszNazwisko` — ustawia wartość właściwości `$nazwisko`,
- `wpiszImię` — ustawia wartość właściwości `$imię`,
- `wpiszId` — ustawia wartość właściwości `$id`,
- `pobierzNazwisko` — odczytuje wartość właściwości `$nazwisko`,
- `pobierzImię` — odczytuje wartość właściwości `$imię`,
- `pobierzId` — odczytuje wartość właściwości `$id`.

Po zdefiniowaniu podanych metod definicja klasy `osoba` będzie miała postać:

```
<?php
class osoba
{
public $id, $nazwisko, $imię;
function wpiszNazwisko($arg1)
   {
      $this -> nazwisko = $arg1;
   }
function pobierzNazwisko()
   {
      return $this -> nazwisko;
   }
function wpiszImię($arg2)
   {
      $this -> imię = $arg2;
   }
function pobierzImię()
   {
      return $this -> imię;
   }
function wpiszId($arg3)
   {
      $this -> id = $arg3;
   }
```

```
function pobierzId()
  {
    return $this -> id;
  }
}
?>
```

Metoda `wpiszNazwisko()` posiada jeden argument o nazwie `$arg1`, który zostanie przypisany właściwości `$nazwisko`. Podobnie działają metody `wpiszImię()` oraz `wpiszId()`. Metoda `pobierzNazwisko()` za pomocą polecenia `return` zwraca wartość właściwości `$nazwisko`, metoda `pobierzImię()` zwraca wartość właściwości `$imię`, a metoda `pobierzId()` zwraca wartość właściwości `$id`. W definicji metod został użyte słowo kluczowe `this`, które oznacza odwołanie do obiektu bieżącego. Użycie tego słowa w zapisie `$return $this -> nazwisko` oznacza odwołanie do właściwości `$nazwisko` z klasy `osoba`. Pominięcie słowa kluczowego `this` spowodowałoby błędne działanie kodu, ponieważ zapis `$nazwisko` oznaczałby odwołanie do zmiennej lokalnej funkcji `pobierzNazwisko()`, a nie do właściwości obiektu.

7.10.3. Tworzenie obiektów

Na podstawie istniejącej klasy można tworzyć obiekty należące do niej. Obiekt jest instancją, czyli wystąpieniem danej klasy. Obiekty tworzymy poleceniem `new`, po którym występuje nazwa klasy. Aby nowo utworzony obiekt mógł być używany, należy przypisać go do zmiennej:

```
$nazwa_zmiennej = new nazwa_klasy();
```

Przez nazwę zmiennej możliwe będzie odwoływanie się do utworzonego obiektu typu `nazwa_klasy`.

Do składowych klasy można odwoływać się, używając operatora `->`.

Odwołanie do właściwości klasy ma postać:

```
$nazwa_obiektu -> nazwa_właściwości;
```

W ten sposób można dla wskazanych właściwości zarówno odczytywać, jak i zapisywać ich wartości.

Jeżeli został utworzony obiekt `$postać` należący do klasy `osoba`, to żeby odczytać wartość właściwości `nazwisko` i przypisać ją do zmiennej `$nazw_postać`, należy użyć polecenia:

```
$nazw_postać = $postać -> nazwisko;
```

Aby natomiast przypisać właściwości `imię` tego obiektu wartość `Jan`, należy użyć polecenia:

```
$postać -> imię = 'Jan';
```

Odwołanie do metod klasy ma postać:

```
$nazwa_obiektu -> nazwa_metody(argumenty_metody);
```

Przykład 7.89

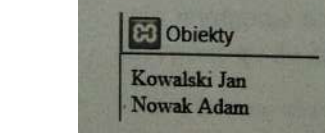
```
<?php
class osoba
{
public $id, $nazwisko, $imię;
function wpiszNazwisko($arg1)
   {
      $this -> nazwisko = $arg1;
   }
 function pobierzNazwisko()
   {
      return $this -> nazwisko;
   }
 function wpiszImię($arg2)
   {
      $this -> imię = $arg2;
   }
 function pobierzImię()
   {
      return $this -> imię;
   }
 function wpiszId($arg3)
   {
      $this -> id = $arg3;
   }
 function pobierzId()
   {
      return $this -> id;
   }
}
 $postać1 = new osoba();
 $postać2 = new osoba();
```

```
$postać1 -> nazwisko = 'Kowalski';
$postać1 -> imię = 'Jan';
$postać1 -> id = 1;

$postać2 -> wpiszNazwisko('Nowak');
$postać2 -> wpiszImię('Adam');
$postać2 -> wpiszId(2);

echo $postać1 -> nazwisko." ".$postać1 -> imię;
echo "<br/>";
echo $postać2 -> pobierzNazwisko()." ".$postać2 -> pobierzImię();
?>
```

W podanym przykładzie została utworzona klasa `osoba`. Zostały również utworzone dwa obiekty klasy `osoba`, które zostały przypisane zmiennym `$postać1` i `$postać2`. Dla obiektu `postać1` właściwościom `nazwisko`, `imię` oraz `id` w sposób bezpośredni zostały przypisane wartości. Natomiast dla obiektu `postać2` podanym właściwościom wartości zostały przypisane przez wywołanie odpowiednich metod. Następnie ustalone wartości zostały odczytane i wyświetlone na ekranie (rysunek 7.43). Dla obiektu `postać1` był to odczyt bezpośredni, dla `postać2` był to odczyt przez wywołanie metod



Rysunek 7.43. Wyświetlenie właściwości przypisanych obiektom postać1 i postać2

7.10.4. Hermetyzacja

Przedstawiony sposób definiowania pól i metod nie jest do końca zgodny z zasadami programowania obiektowego. Do definiowania klasy powinien być zastosowany mechanizm hermetyzacji. Dopiero definicja klasy zawierająca określenie, do których składowych klasy (właściwości i metody) mamy dostęp z zewnątrz, a które należą wyłącznie do klasy i nie są dostępne poza nią, odzwierciedla istotę obiektowości.

Sposób dostępu do składowych klasy jest definiowany przez specyfikatory dostępu (modyfikatory dostępu). Są to:

- `public` — dostęp publiczny,
- `private` — dostęp prywatny,
- `protected` — dostęp chroniony.

Właściwości i metody publiczne są dostępne zarówno wewnątrz klasy, jak i na zewnątrz.

Właściwości i metody prywatne są własnością danej klasy i nie są widoczne na zewnątrz. Mogą być wykorzystywane tylko wewnątrz tej klasy.

Właściwości i metody chronione działają jak prywatne, jednak mogą być dziedziczone przez klasy pochodne. W klasach pochodnych również nie są widoczne na zewnątrz.

Sposób dostępu do właściwości i metod określa się przez umieszczenie przed ich nazwami określenia: `private`, `protected` lub `public`. Jeśli nie zostanie użyte żadne z określeń, właściwość lub metoda są traktowane jako publiczne.

Podany niżej przykład zawiera kod z zastosowaniem hermetyzacji.

Przykład 7.90

```
<?php
class osoba
{
private $nazwisko, $imię;
public function wpiszNazwisko($arg1)
   {
      $this -> nazwisko = $arg1;
   }
 public function pobierzNazwisko()
   {
      return $this -> nazwisko;
   }
 public function wpiszImię($arg2)
   {
      $this -> imię = $arg2;
   }
 public function pobierzImię()
   {
      return $this -> imię;
   }
}
$postać1 = new osoba();
$postać2 = new osoba();

$postać1 -> nazwisko = 'Kowalski';
$postać1 -> imię = 'Jan';
```

```
$postać2 -> wpiszNazwisko('Nowak');
$postać2 -> wpiszImię('Adam');

echo $postać1 -> nazwisko, $postać1 -> imię;
echo "<br/>";
echo $postać2 -> pobierzNazwisko(), $postać2 -> pobierzImię();
?>
```

W podanym kodzie nie ma bezpośredniego dostępu do właściwości obiektu (`nazwisko`, `imię`), ponieważ dostały one zadeklarowane jako prywatne. Dostęp do nich jest możliwy tylko przez metody. Dlatego przy próbie wykonania kodu pojawi się komunikat informujący o braku dostępu do tych właściwości. Należy odwoływać się do właściwości obiektu (`nazwisko`, `imię`) przez metody lub zmienić ich deklarację na `public`.

Zadanie 7.14

Przeanalizuj kod z przykładu 7.90. Odpowiedz, dlaczego nie będzie on działał prawidłowo. Doprowadź kod do prawidłowej postaci i sprawdź jego działanie.

7.10.5. Konstruktor i destruktor

Po utworzeniu obiektu jego właściwości nie zawierają żadnych wartości. Jeżeli taka sytuacja nie jest pożądana i po utworzeniu obiektu właściwości powinny mieć ustalone wartości początkowe, to można zastosować specjalny mechanizm nazywany konstruktorem.

Konstruktor

Konstruktor to specjalna metoda każdej klasy wywoływana podczas tworzenia obiektu tej klasy. Głównym jego przeznaczeniem jest inicjalizacja właściwości obiektu.

Definiowanie konstruktora polega na utworzeniu w kodzie klasy metody o nazwie `__construct`, w postaci:

```
class nazwa_klasy
{
    function __construct()
    {
        //treść konstruktora
    }
    // składowe klasy
}
```

Konstruktor wywoływany jest automatycznie po utworzeniu obiektu danej klasy.

Przykład 7.91

Jeżeli chcemy, aby obiekty klasy `osoba` przyjmowały automatycznie określone wartości początkowe, to do definicji klasy należy dodać definicję konstruktora, np. w postaci:

```
<?php
class osoba
{
public $nazwisko, $imię;
function __construct()
   {
      $this -> nazwisko = "Anonim";
      $this -> imię = "Gall";
   }
public function wpiszNazwisko($arg1)
   {
      $this -> nazwisko = $arg1;
   }
public function pobierzNazwisko()
   {
      return $this -> nazwisko;
   }
public function wpiszImię($arg2)
   {
      $this -> imię = $arg2;
   }
public function pobierzImię()
   {
      return $this -> imię;
   }
}
$postać1 = new osoba();
$postać2 = new osoba();

$postać2 -> wpiszNazwisko('Nowak');
$postać2 -> wpiszImię('Adam');
```

```
echo $postać1 -> pobierzNazwisko(), $postać1 -> pobierzImię();
echo "<br/>";
echo $postać2 -> pobierzNazwisko(), $postać2 -> pobierzImię();
?>
```

Destruktor

Destruktor jest przeciwieństwem konstruktora. Jest to specjalna metoda każdej klasy, wywoływana, gdy obiekt jest usuwany z pamięci (niszczony). Głównie jest używany w dużych projektach, gdy klasa używa zasobów, które trzeba zwolnić przed usunięciem obiektu.

Definiowanie destruktora polega na utworzeniu w kodzie klasy metody o nazwie `__destruct`, w postaci:

```
class nazwa_klasy
{
    function __destruct()
    {
        //treść destruktora
    }
    // składowe klasy
}
```

Destruktor jest wywoływany automatycznie podczas niszczenia obiektu (usuwania obiektu z pamięci). Jego przeznaczeniem jest wykonanie czynności końcowych dla obiektu, np. zwalnianie zasobów, zamykanie plików, zamykanie połączeń z bazami danych.

7.10.6. Dziedziczenie

Dziedziczenie to jeden z paradygmatów programowania obiektowego. Umożliwia budowanie hierarchii klas przejmujących swoje właściwości. Dziedziczenie stosowane jest do tworzenia nowych klas opartych na istniejącej klasie bazowej przez dodanie nowych elementów (właściwości lub metod).

Dziedziczenie w PHP jest realizowane przez dodanie w definicji nowej klasy opartej na innej klasie słowa `extends`. Definicja takiej klasy ma postać:

```
class klasa_potomna extends klasa_bazowa
{
  //kod klasy potomnej
}
```

Taki zapis oznacza, że klasa potomna dziedziczy po klasie bazowej właściwości i metody

Przykład 7.92

```
<?php
class osoba
{
public $nazwisko, $imię;
public function wpiszNazwisko($arg1)
   {
      $this -> nazwisko = $arg1;
   }
public function pobierzNazwisko()
   {
      return $this -> nazwisko;
   }
public function wpiszImię($arg2)
   {
      $this -> imię = $arg2;
   }
public function pobierzImię()
   {
      return $this -> imię;
   }
}
class klient extends osoba
{
public $adres;
public function wpiszAdres($arg3)
   {
      $this -> adres = $arg3;
   }
public function pobierzAdres()
   {
      return $this -> adres;
   }
}
```

```
$osoba1 = new osoba();
$osoba1 -> wpiszNazwisko('Żak');
$osoba1 -> wpiszImię('Ewa');
echo "Dane obiektu klasy osoba <br/>";
echo $osoba1 -> pobierzNazwisko()." ".$osoba1 -> pobierzImię();
echo "<br/>";

$klient1 = new klient();
$klient1 -> wpiszNazwisko('Król');
$klient1 -> wpiszImię('Marcin');
$klient1 -> wpiszAdres('Warszawa ul. Kręta 7');

echo "Dane obiektu klasy klient <br/>";
echo $klient1 -> pobierzNazwisko()." ".$klient1 -> pobierzImię()." ".$klient1
-> pobierzAdres();
echo "<br/>";
?>
```

W podanym kodzie zostały utworzone dwie klasy: `osoba` i `klient`. Klasa `klient` dziedziczy właściwości i metody z klasy `osoba`. W klasie `klient` zostały zdefiniowane właściwość `adres` oraz dwie metody: `wpiszAdres()` i `pobierzAdres()`. Dzięki dziedziczeniu każdy obiekt klasy `klient` będzie również zawierał wszystkie właściwości i metody z klasy `osoba`. Następnie został utworzony obiekt `osoba1` należący do klasy `osoba` i poprzez użycie metod tej klasy właściwościom obiektu zostały nadane odpowiednie wartości. W celu sprawdzenia stanu tego obiektu została wyświetlona jego zawartość. W dalszej części kodu został utworzony obiekt `klient1` należący do klasy `klient`. Właściwościom tego obiektu poprzez użycie metod klasy `osoba` i `klient` również zostały nadane odpowiednie wartości. Po wyświetleniu jego zawartości widać, że obiekt klasy `klient` posiada wszystkie właściwości i metody klasy `osoba`.

Przesłanianie składowych

Klasa potomna dziedziczy składowe po klasie bazowej. Ale jeżeli w klasie potomnej zostaną zdefiniowane właściwości lub metody o takich samych nazwach jak w klasie bazowej, to składowe z klasy bazowej zostaną przysłonięte przez składowe klasy potomnej. Oznacza to, że w obiektach klasy bazowej będą obowiązywały składowe klasy bazowej, a w obiektach klasy pochodnej składowe zdefiniowane w klasie bazowej, które nie zostały przysłonięte, oraz składowe klasy pochodnej.

Przykład 7.93

Przyjmijmy, że kod z przykładu 7.92 został uzupełniony o definicję metody `pobierzImię()` dla klasy `klient`. Definicja klasy `klient` będzie miała postać:

```
class klient extends osoba
{
public $adres;
public function wpiszAdres($arg3)
   {
      $this -> adres = $arg3;
   }
public function pobierzAdres()
   {
      return $this -> adres;
   }
public function pobierzImię()
   {
      return "Imię nieznane";
   }
}
```

Tym razem po wyświetleniu zawartości obiektu `klient1` widać, że metoda `pobierzImię()` klasy `klient` przysłoniła metodę `pobierzImię()` klasy `osoba`.

Ale istnieje możliwość wyświetlenia w klasie pochodnej metody klasy bazowej, gdy w obu klasach mają one takie same nazwy. Do tego celu służy polecenie `parent::` zapisywane w postaci:

```
parent::nazwa_metody(argumenty);
```

Przykład 7.94

```
<?php
class osoba
{
public $nazwisko, $imię;
public function wpiszNazwisko($arg)
   {
      $this -> nazwisko = $arg;
   }
public function pobierzNazwisko()
```

```
    {
        return $this -> nazwisko;
    }
public function wpiszImię($arg)
    {
        $this -> imię = $arg;
    }
public function pobierzImię()
    {
        return $this -> imię;
    }
}
class klient extends osoba
{
public $imię2;
public function wpiszImię($arg)
    {
        parent:: wpiszImię($arg);
        $this -> imię2 = $arg;
    }
public function pobierzImię()
    {
        parent:: pobierzImię();
        return $this -> imię2;
    }
}

$klient1 = new klient();
$klient1 -> wpiszNazwisko('Król');
$klient1 -> wpiszImię('Marcin');

echo "Dane obiektu klient1 <br/>";
echo $klient1 -> pobierzNazwisko(), $klient1 -> pobierzImię();
echo "<br/>";
?>
```

W podanym przykładzie zostały utworzone dwie klasy: `osoba` i `klient`. Klasa `klient` dziedziczy właściwości i metody z klasy `osoba`. W klasie `klient` zostały zdefiniowane właściwość `imię2` oraz dwie metody: `wpiszImię()` i `pobierzImię()`. Metody o takich samych nazwach istnieją w klasie bazowej (`osoba`). Ponowne ich zdefiniowanie w klasie potomnej (`klient`) domyślnie spowodowałoby przysłonięcie metod klasy bazowej. Ale ponieważ w definicji metody `wpiszImię()` zostało użyte odwołanie `parent::`, spowoduje to najpierw wywołanie metody `wpiszImię()` klasy `osoba`, a następnie metody `wpiszImię()` klasy `klient`.

Dziedziczenie konstruktorów i destruktorów

Dziedziczenie wiąże się również z dziedziczeniem konstruktorów i destruktorów. Jeżeli zostaną one zdefiniowane w klasach potomnych, to będą wykonane zamiast konstruktorów i destruktorów klas bazowych. Jeżeli klasa potomna nie zawiera konstruktora lub destruktora, to wykonane będą konstruktor i destruktor z klasy bazowej.

7.10.7. Wyjątki

Obsługa błędów jest jednym z istotniejszych elementów tworzonych aplikacji. Język PHP został wyposażony w mechanizmy, które mogą pomóc w ochronie danych. Jednym z nich jest obsługa wyjątków. Obsługa wyjątków w PHP dotyczy tylko niektórych jego elementów. Większość z nich nie generuje wyjątku, tylko zgłasza wystąpienie błędu. Wyjątki głównie dotyczą elementów nowych, napisanych w sposób obiektowy. Należą do nich między innymi obsługa wymiany danych z bazą MySQL i obsługa obiektowego modelu dokumentu XML.

Zgłaszanie wyjątków

Wyjątek w PHP jest obiektem klasy `Exception` (lub jej pochodnej) zgłoszonym za pomocą specjalnej instrukcji `throw` (tzw. rzucanie). Konstruktor klasy `Exception` przyjmuje dwa argumenty: komunikat o błędzie oraz kod błędu. Jeżeli zostanie pominięty argument *kod błędu*, to przyjmie on wartość domyślną równą `0`. Jeżeli zostanie pominięty argument *komunikat błędu*, to przyjmie on wartość `null`. Te właściwości będą przechowywane przez utworzony obiekt. Jego wywołanie może mieć postać:

```
Exception("komunikat", kod_błędu)
```

Inne metody klasy `Exception` to:

- `getMessage` — zwraca komunikat powiązany z danym wyjątkiem,
- `getCode` — zwraca kod powiązany z danym wyjątkiem,
- `getFile` — zwraca nazwę pliku, w którym wystąpił dany wyjątek,
- `getLine` — zwraca numer linii kodu, w której wystąpił dany wyjątek.

Metody te nie mogą być przesłaniane w klasach pochodnych.

Klasy pochodne dostępne standardowo (od wersji 5.1) to między innymi:

- `LogicException` — dla logicznych błędów skryptów,
- `RuntimeException` — dla błędów powstałych w trakcie wykonywania skryptu.

Do zgłaszania wyjątków służy instrukcja `throw`, której argumentem jest wyrażenie dające obiekt klasy `Exception` lub pochodnych. Najczęściej spotykanym zapisem jest:

```
throw new Exception()
```

Instrukcja `throw` działa podobnie jak instrukcja `return`. Nie zwraca jednak wartości, ale rzuca wyjątek, który powinien zostać obsłużony.

Skrypt może zgłosić wyjątek podczas wystąpienia sytuacji, która zostanie uznana za błędną (np. brak połączenia z serwerem, niedostępna baza danych). Zgłoszenie wyjątku powoduje przerwanie wykonywania bieżącego fragmentu kodu i przejście do obsługi zgłoszonego wyjątku. W kodzie obsługi wyjątku można sprawdzić, jakiego rodzaju błąd wystąpił, i wyświetlić odpowiednie informacje o problemie.

Do obsługi wyjątków służy instrukcja `try...catch` w postaci:

```
try {
    // instrukcje, w których może wystąpić wyjątek
}
catch (Exception $obiekt1) {
    // obsługa wyjątku
}
catch (Exception $obiekt2) {
    // obsługa kolejnego wyjątku
}
```

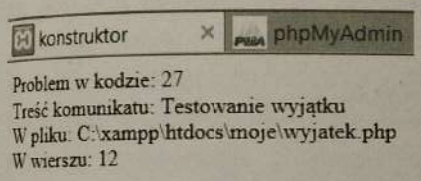
Po konstrukcji `try {...}` powinien znaleźć się kod, który może wygenerować sytuację wyjątkową (np. nawiązanie połączenia z serwerem). Im mniej wierszy kodu w tym miejscu, tym łatwiej będzie można zidentyfikować źródło błędu. Po słowie `catch` występuje deklaracja obiektu klasy `Exception` (lub klas pochodnych). Dalej umieszczany jest w nawiasach klamrowych `{..}` kod obsługi wyjątku (np. komunikat o błędzie).

Działanie instrukcji polega na wykonaniu kodu umieszczonego w konstrukcji `try {...}`. Jeżeli kod został wykonany pomyślnie, blok `catch` zostanie pominięty. Gdy wystąpi błąd (wyjątek), będzie on przechwycony (ang. *catch*) jako obiekt klasy `Exception` i zostaną wykonane działania zapisane w bloku `catch {..}`. W bloku tym można użyć metod udostępnianych przez obiekt klasy `Exception`. W instrukcji może pojawić się kilka bloków `catch`, jeżeli spodziewane jest wystąpienie kilku wyjątków.

Przykład 7.95

```
<?php
try {
throw new Exception("Testowanie wyjątku",27);
}
catch (Exception $e) {
echo "Problem w kodzie: ".$e->getCode()."<br/>";
echo "Treść komunikatu: ".$e->getMessage()."<br/>";
echo "W pliku: ".$e->getFile()."<br/>";
echo "W wierszu: ".$e->getLine()."<br/>";
}
?>
```

Wynik interpretacji kodu został pokazany na rysunku 7.44.



Rysunek 7.44. Testowanie wyjątku

Zadanie 7.15

Utwórz skrypt PHP zawierający strukturę hierarchiczną klas dla osób związanych ze szkołą. W strukturze tej uwzględnij obiekty: uczniowie, nauczyciele, rodzice, pracownicy administracji. Zdefiniuj odpowiednio klasy bazowe oraz klasy potomne. Określ właściwości i metody dla każdej ze zdefiniowanych klas. Określ sposób dziedziczenia właściwości i metod. Określ sposób dostępu do składowych klas. Dla każdej ze zdefiniowanych klas utwórz przykładowe obiekty. Sprawdź funkcjonowanie utworzonej struktury.

7.11. Formularze

Dynamiczna strona WWW to strona, która jest generowana na podstawie danych przekazanych m.in. przez użytkownika. Podstawowym sposobem przekazywania danych z przeglądarki internetowej do serwera są formularze HTML. Użytkownik za pomocą formularza wprowadza różne informacje, które są wysyłane do serwera. Dane te są przetwarzane na serwerze za pomocą skryptów PHP.

7.11.1. Formularz HTML

Przykład 7.96

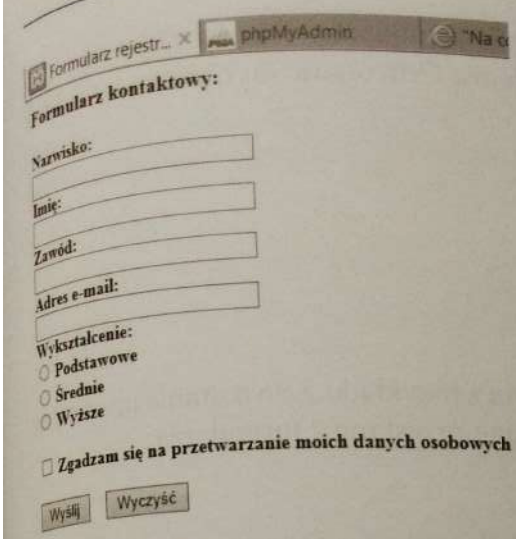
Prosty formularz HTML może mieć postać:

```
<html>
<head>
<title>Formularz rejestracyjny</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form action="http://localhost/skrypt_form.php" method="post">
<b><font size="4"><b>Formularz kontaktowy:</font></b><br><br>
Nazwisko:<br/>
<input type="text" name="nazw" value=" " size="30"><br>
Imię:<br/>
<input type="text" name="im" value=" " size="30"><br>
Zawód:<br/>
<input type="text" name="zaw" value=" " size="30"><br>
Adres e-mail:<br/>
<input type="text" name="adr" value=" " size="30"><br>

<b>Wykształcenie:</b><br>
<input type="radio" value="podst" name="wykszt" checked> Podstawowe<br>
<input type="radio" value="srednie" name="wykszt"> Średnie<br>
<input type="radio" value="wyzsze" name="wykszt"> Wyższe<br><br>

<input type="checkbox" name="opcje"  maxlength="1">
Zgadzam się na przetwarzanie moich danych osobowych<br><br>
<input type="submit" value="Wyślij" name="wyslij">   
<input type="reset" value="Wyczyść" name="zeruj">
</form>
</body>
</html>
```

Podany w przykładzie formularz składa się z czterech pól do wprowadzania tekstu, pola wyboru typu `radio`, pola wyboru typu `checkbox` oraz przycisków typu `submit` (*Wyślij*) oraz `reset` (rysunek 7.45).



Rysunek 7.45. Przykładowy formularz

Po kliknięciu przycisku *Wyślij* dane z formularza powinny zostać przesłane na serwer za pomocą metody `POST`. Parametr `action` wskazuje nazwę skryptu PHP oraz adres serwera, który zostanie wywołany po zatwierdzeniu formularza. W tym przykładzie zostanie uruchomiony skrypt o nazwie *skrypt_form.php*, który znajduje się na serwerze lokalnym o adresie *http://localhost*. Dane wprowadzone do formularza zostaną przekazane do wybranego skryptu. Parametr `method` określa sposób (metodę) przekazania informacji do wskazanego skryptu. Może przyjąć jedną z dwóch wartości: `post` i `get`. Wartością domyślną jest `get`, ale zaleca się stosowanie metody `post`.

7.11.2. Przekazywanie danych z formularza

Metoda `GET` jest używana, kiedy parametrów jest niewiele. W tej metodzie parametry są przekazywane za pomocą adresu URL. Schematyczna postać URL wygląda następująco:

http://www.nazwa_serwera.pl/strona.php?parametr1=wartość1&parametr2=wartość2

Długość adresu URL jest ograniczona, a przekazywane parametry są widoczne w pasku adresu przeglądarki, dlatego metoda ta jest stosowana do przesyłania niedużych ilości danych. Adres skryptu PHP od przesyłanych parametrów oddzielany jest znakiem zapytania. Przesyłane pary *parametr=wartość* są oddzielane znakiem *ampersand* (`&`).

Metoda `POST` do przekazywania parametrów wykorzystuje nagłówek strony. Metoda ta umożliwia przekazywanie większej ilości parametrów, a parametry nie są widoczne w pasku przeglądarki.

Przesłane wartości parametrów są zapisywane w odpowiednich tablicach asocjacyjnych. Dane przesłane metodą `GET` są zapisywane w tablicy `$_GET`, natomiast dane przesłane metodą `POST` w tablicy `$_POST`. Tablice `$_GET` i `$_POST` są superglobalne. Oznacza to, że są widoczne w każdym miejscu kodu PHP bez konieczności użycia składni `globals`.

Indeksami tablicy `$_GET` i `$_POST` są nazwy pól formularza (wartość atrybutu `name` w formularzu HTML), do których się odwołujemy. Odwołanie do elementów tablicy ma postać:

```
$zmienna = $_GET['nazwa_pola'];
```

lub

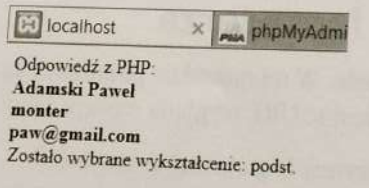
```
$zmienna = $_POST['nazwa_pola'];
```

Przykład 7.97

W celu sprawdzenia poprawności działania skryptu z przykładu 7.96 zostanie utworzony skrypt PHP (*skrypt_form.php*) przetwarzający dane przesłane z formularza.

```
<?php
echo "Odpowiedź z PHP: <b><br/>".$_POST['nazw']."</b>";
echo "<b>".$_POST['im']."</b><br>";
echo "<b>".$_POST['zaw']."</b><br>";
echo "<b>".$_POST['adr']."</b><br>";
echo "Zostało wybrane wykształcenie: {$_POST['wykszt']}.";
?>
```

Jeżeli podany kod zostanie zapisany w pliku *skrypt_form.php*, to po otwarciu formularza, wpisaniu danych i jego wysłaniu przeglądarka powinna wyświetlić przesłane dane (rysunek 7.46).



Rysunek 7.46. Dane przesłane za pomocą formularza

Funkcja isSet

Podczas przetwarzania formularza można wykorzystać funkcję `isSet()`. Pozwala ona stwierdzić, czy określone pole formularza zostało wypełnione, a dokładniej, czy do skryptu została przekazana wartość danego pola. Funkcja zwraca `true`, gdy wartość została przekazana, w przeciwnym razie zwraca `false`.

Przykład 7.98

W podanym wyżej przykładzie formularz oprócz pól tekstowych zawierał również pola typu `radio`. Są to trzy pola wyboru, gdzie z grupy opcji można zaznaczyć tylko jedną. Kliknięcie wartości powoduje zaznaczenie opcji, ale równocześnie usuwa zaznaczenie innej opcji. Każde z pól tego typu ma taką samą wartość atrybutu `name`. Atrybut `value`

w tym przypadku jest konieczny i musi być inny dla każdego pola. Przez wartość atrybutu `value` pola będą identyfikowane w skrypcie.

Utworzymy skrypt PHP, który będzie rozpoznawał, która opcja pola typu `radio` została zaznaczona w formularzu (kod może zostać dołączony do pliku *skrypt_form.php*).

```
<?php
if(!isSet($_POST['wykszt'])){
echo "<br/>Proszę zaznaczyć pole <b>Wykształcenie</b>";
}
else {
echo "<br/>Zostało wybrane wykształcenie: {$_POST['wykszt']}.";
}
?>
```

W podanym skrypcie do sprawdzenia, czy zaznaczone zostało pole opcji, zastosowano instrukcję warunkową `if` oraz funkcję `isSet()`. W warunku `!isSet($_POST['wykszt'])` został użyty operator logicznej negacji `!`, czyli warunek zwróci wartość `true`, gdy pole nie zostanie wypełnione, i będzie wysłany komunikat *Proszę zaznaczyć pole Wykształcenie*. W przeciwnym razie za pomocą składni `{$_POST['wykszt']}` zostanie odczytana z tablicy `$_POST` wartość opcji `wykszt` (będzie to wartość atrybutu `value` dla znacznika `<input type="radio">`).

Funkcja empty

Funkcja `empty()` sprawdza, czy istniejąca zmienna użyta jako argument jest pusta. Jeżeli zmienna jest pusta, to funkcja zwraca wartość `true`, w przeciwnym razie zwraca wartość `false`.

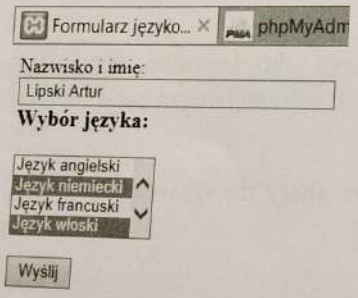
W formularzu może wystąpić pole typu `SELECT`, które służy do wyświetlania listy wartości i pozwala na wybranie jednej z nich lub kilku.

Przykład 7.99

```
<html>
<head>
<title>Formularz językowy</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body >
<form action="http://localhost/skrypt_form.php" method="post">
Nazwisko i imię:<br/>
<input type="text" name="nazw" value=" " size="40"><br>
```

```
<b><font size="4"><b>Wybór języka:</font></b><br><br>
<select name="języki[]" multiple>
<option value="Język angielski"> Język angielski</option>
<option value="Język niemiecki"> Język niemiecki</option>
<option value="Język francuski"> Język francuski</option>
<option value="Język włoski"> Język włoski</option>
<option value="Język rosyjski"> Język rosyjski</option>
<option value="Język hiszpański"> Język hiszpański</option>
</select>
<p><input type="submit" value="Wyślij" name="wyslij"></p>
</form>
</body>
</html>
```

Gdyby utworzona w formularzu lista wyboru wartości została zdefiniowana w postaci `<select name="języki" multiple>`, to skrypt obsługujący formularz miałby dostęp tylko do jednej wartości (`$_POST["języki"]`). Gdy zostaną dodane kwadratowe nawiasy w podany sposób: `<select name="języki[]">`, to elementy wybrane przez użytkownika w polu `języki[]` będą dostępne w tablicy `$_POST["języki"]`. Utworzony formularz został pokazany na rysunku 7.47.



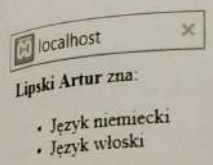
Rysunek 7.47. Sposób wypełniania formularza

Należy jeszcze dołączyć skrypt PHP (zapisany w pliku *skrypt_form.php*) w postaci:

```
<?php
if(!empty($_POST['języki'])) {
echo "<p><b>".$_POST['nazw']."</b> zna:</p>";
echo "<ul>";
foreach ($_POST['języki'] as $wartosc) {
echo "<li>$wartosc</li>";
}
```

```
echo "</ul>";
}
else {
echo "<p><b>".$_POST['nazw']."</b> nie zna żadnego języka.</p>";
}
?>
```

Instrukcja `if(!empty($_POST['języki']))` sprawdza, czy tablica `$_POST['języki']` jest pusta. Jeżeli nie, to nastąpi wyświetlenie zawartości każdego elementu tablicy (rysunek 7.48). W tablicy znajdują się wartości znacznika `<option value=...>` wybrane przez użytkownika.



Rysunek 7.48. Po wysłaniu formularza przeglądarka wyświetla przesłane dane

Ta technika może być stosowana do różnych elementów, nie tylko do pól `SELECT`. Jeżeli kilku polom nadamy taką samą nazwę i nazwa pola będzie kończyła się nawiasem kwadratowym, to PHP połączy wszystkie wartości o tej samej nazwie pola w jedną tablicę.

Po wprowadzonych zmianach kod zawierający formularz HTML będzie wyglądał tak:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Formularz rejestracyjny</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form action="http://localhost/skrypt_form.php" method="post">
<p><b><font size="4"><b>Formularz kontaktowy:</font></b></p>
Nazwisko:<br/>
<input type="text" name="nazw" value=" " size="30"><br>
Imię:<br/>
<input type="text" name="im" value=" " size="30"><br>
Zawód:<br/>
<input type="text" name="zaw" value=" " size="30"><br>
```

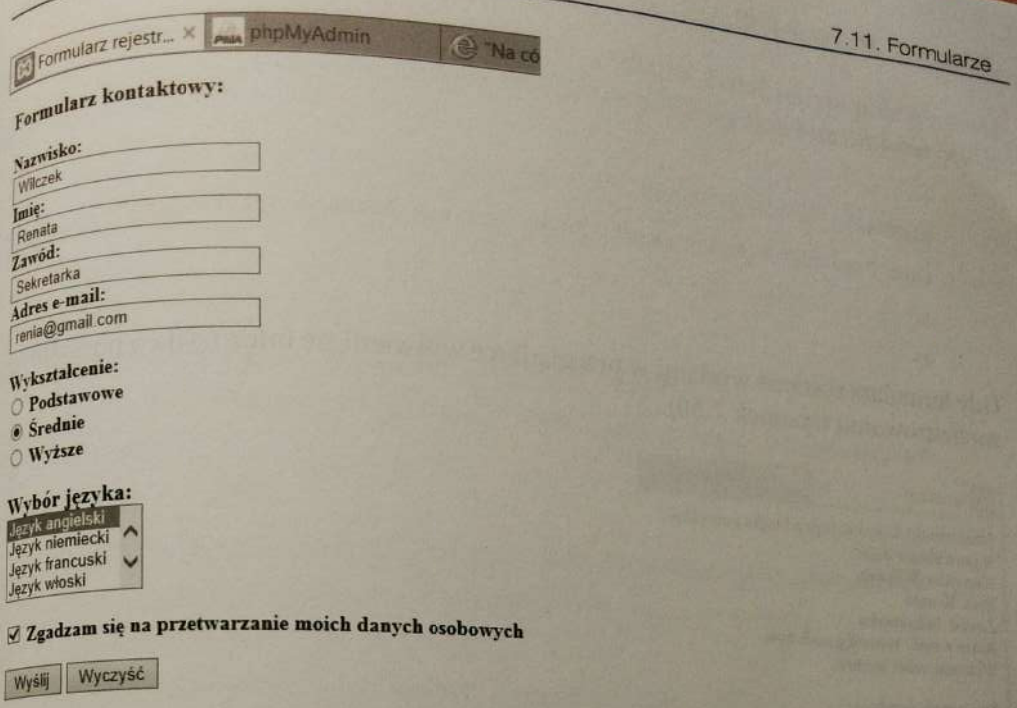
```
Adres e-mail:<br/>
<input type="text" name="adr" value=" " size="30"><br>

<p><b>Wykształcenie:</b><br>
<input type="radio" value="podstawowe" name="wykszt"> Podstawowe<br>
<input type="radio" value="średnie" name="wykszt"> Średnie<br>
<input type="radio" value="wyższe" name="wykszt"> Wyższe</p>

<p><b><font size="4"><b>Wybór języka:</font></b><br>
<select name="języki[]" multiple="multiple">
<option value="Język angielski"> Język angielski</option>
<option value="Język niemiecki"> Język niemiecki</option>
<option value="Język francuski"> Język francuski</option>
<option value="Język włoski"> Język włoski</option>
<option value="Język rosyjski"> Język rosyjski</option>
<option value="Język hiszpański"> Język hiszpański</option>
</select></p>

<p><input type="checkbox" name="opcje"  maxlength="1">
Zgadzam się na przetwarzanie moich danych osobowych</p>
<p><input type="submit" value="Wyślij" name="wyslij">
<input type="reset" value="Wyczyść" name="zeruj"></p>
</form>
</body>
</html>
```

Wynik interpretacji tego kodu został pokazany na rysunku 7.49.



Rysunek 7.49. Wprowadzanie danych przez formularz

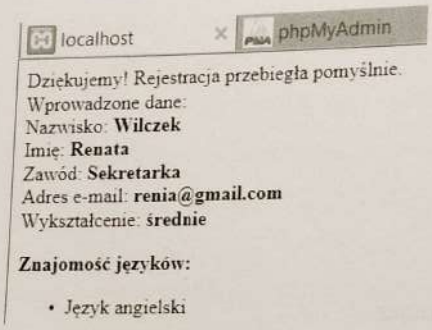
Kod zawierający skrypt PHP (plik *skrypt_form.php*):

```
<?php
echo "Dziękujemy! Rejestracja przebiegła pomyślnie.<br/>";
echo "Wprowadzone dane:<br/>";
echo "Nazwisko: <b>".trim($_POST['nazw'])."</b><br/>";
echo "Imię: <b>".trim($_POST['im'])."</b><br/>";
echo "Zawód: <b>".trim($_POST['zaw'])."</b></br>";
echo "Adres e-mail: <b>".trim($_POST['adr'])."</b><br/>";

if(!isSet($_POST['wykszt']))
echo "Proszę zaznaczyć pole <b>Wykształcenie</b>";
else
echo "Wykształcenie: <b>".trim($_POST['wykszt'])."</b><br>";
echo "<p><b>Znajomość języków:</b></p>";
if(!empty($_POST['języki'])) {
echo "<ul>";
foreach ($_POST['języki'] as $wartosc) {
echo "<li>$wartosc</li>";
```

```
}
echo "</ul>";
}
else {
echo "<p><b>".$_POST['nazw']."</b> nie zna żadnego języka.</p>";
}
?>
```

Gdy formularz zostanie wysłany, w przeglądarce wyświetli się informacja o pomyślnym zarejestrowaniu (rysunek 7.50).



Rysunek 7.50. Wynik pobrania danych z formularza

Zadanie 7.16

Opracuj formularz przeznaczony do rejestrowania uczniów zapisujących się do szkoły. Określ informacje, które powinny znaleźć się w formularzu, oraz sposób ich wprowadzania. Utwórz skrypt przetwarzający dane przesłane z formularza. W skrypcie powinno być sprawdzane wypełnienie wszystkich pól.

7.11.3. Przesyłanie plików na serwer

Często zachodzi potrzeba przekazania pliku na serwer. W tym celu należy przygotować formularz, w którym będzie można wskazać plik do wysłania, oraz utworzyć skrypt PHP, który odbierze wysłany plik. Dodatkowo należy odpowiednio przygotować środowisko PHP.

Formularze służące do wysyłania plików w definicji znacznika `<form>` muszą zawierać atrybut:

- `enctype` — określa sposób kodowania danych; musi zostać podana wartość: `enctype="multipart/form-data"`;
- `action` — określa adres skryptu PHP;
- `method` — określa metodę wysyłania danych; musi to być metoda `POST`.

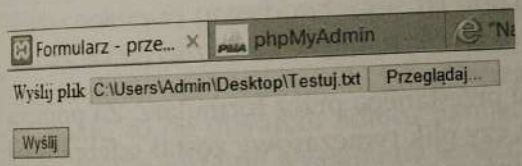
Do wysyłania pliku służy pole `input` typu `file`. Można dodać ukryte pole o nazwie `max_file_size` w celu określenia maksymalnego rozmiaru pliku, który można będzie przesłać na serwer w postaci:

```
<input type="hidden" name ="max_file_size" value="wielkość_pliku">
```

Przykład 7.100

```
<html>
<head>
<title>Formularz - przesyłanie pliku</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form enctype="multipart/form-data" action="odbierz_plik.php" method="post">
<input type="hidden" name="max_file_size" value ="40960">
<p>Wyślij plik
<input type="file" name="plik1" size="30"></p>
<p><input type="submit" value="Wyślij" name="wyslij"></p>
</form>
</body>
</html>
```

Podany skrypt utworzy formularz, za pomocą którego będzie można wysyłać pliki na serwer (rysunek 7.51).



Rysunek 7.51. Formularz do przesyłania plików na serwer

Informacje o plikach, które zostały wysłane, są przechowywane w superglobalnej tablicy asocjacyjnej `$_FILES`. Indeksami tej tablicy są nazwy pól `input` typu `file` pobrane z formularza (wartość atrybutu `name` w formularzu HTML). Pod indeksem zapisana jest cała informacja o przesłanym pliku w postaci kolejnych indeksów. Są to:

- `$_FILES['nazwa_pliku']['name']` — nazwa przesłanego pliku,
- `$_FILES['nazwa_pliku']['type']` — typ MIME pliku,
- `$_FILES['nazwa_pliku']['size']` — wielkość pliku w bajtach,
- `$_FILES['nazwa_pliku']['tmp_name']` — ścieżka tymczasowa, pod którą plik został zapisany na serwerze,
- `$_FILES['nazwa_pliku']['error']` — kod błędu.

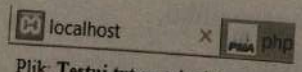
Wysłany do serwera plik jest umieszczany w katalogu tymczasowym. Do przeniesienia pliku do właściwej lokalizacji służy funkcja `move_uploaded_file()`. Przed przeniesieniem pliku do katalogu docelowego można sprawdzić za pomocą funkcji `is_uploaded_file()`, czy plik tymczasowy istnieje.

Przykład 7.101

Kod zapisany w pliku *odbierz_plik.php*:

```
<?php
$katalog_plik = "./";  //katalog docelowy
$max_rozm = $_POST['max_file_size'];
if (is_uploaded_file($_FILES['plik1']['tmp_name'])) {
  if ($_FILES['plik1']['size'] > $max_rozm) {
     echo "Błąd! Plik ma za duży rozmiar!";
  }
  else {
   echo "Plik:<b> " . $_FILES['plik1']['name'] . " </b>został odebrany.<br/>";
     if (isSet($_FILES['plik1']['type']))
     echo "Typ pliku: ".$_FILES['plik1']['type']."<br/>";
       move_uploaded_file($_FILES['plik1']['tmp_name'], $katalog_plik.$_
FILES['plik1']['name']);
  } }
else
echo "Błąd podczas przesyłania pliku";
?>
```

Przedstawiony kod służy do odebrania pliku przesłanego przez formularz. Za pomocą funkcji `is_uploaded_file()` sprawdzamy, czy plik tymczasowy został odebrany od użytkownika. Potem kolejno sprawdzamy informacje na temat wielkości oraz typu pliku. Jeżeli efekty tych działań są pozytywne, plik zostanie przeniesiony za pomocą funkcji `move_uploaded_file()` w miejsce docelowe (rysunek 7.52). Jeżeli plik o takiej samej nazwie istnieje, to zostanie zastąpiony plikiem odebranym. Aby plik mógł zostać przeniesiony w docelowe miejsce, użytkownik musi mieć odpowiednie prawa dostępu do katalogu docelowego.



Plik: **Testuj.txt** został odebrany.
Typ pliku: text/plain

Rysunek 7.52. Informacja o tym, że plik został umieszczony na serwerze

Zadanie 7.17

Do opracowanego w poprzednim zadaniu formularza dodaj pole umożliwiające przesyłanie pliku ze zdjęciem. Przyjmij, że przesyłane pliki mogą być w formacie JPG, JPEG lub PNG. Ich rozmiar nie może przekroczyć 900 kB. Zmodyfikuj skrypt PHP tak, aby możliwe było odbieranie pliku przesłanego przez formularz.

7.12. Pliki cookies i sesje

7.12.1. Pliki cookies w PHP

Pliki cookies

Pliki cookies to niewielkie pliki tekstowe wysyłane przez serwer lub skrypt do przeglądarki i umieszczane przez nią na dysku użytkownika. Pliki te są częścią specyfikacji protokołu HTTP i są wysyłane do przeglądarki w postaci nagłówka o nazwie `Set-Cookie`. Służą głównie do identyfikacji użytkownika. Plik cookie zawiera między innymi nazwę serwera, datę wygaśnięcia pliku oraz informacje na temat domeny i ścieżki. Rozmiar pliku nie może przekroczyć 4 kB.

Zasada działania plików cookies jest następująca:

- Po nawiązaniu połączenia serwer wysyła do przeglądarki nagłówek `Set-Cookie`, który zawiera plik cookie.
- Przeglądarka zapisuje plik cookie na dysku użytkownika.
- Przy kolejnym połączeniu z serwerem przeglądarka wysyła do niego przechowywany na dysku plik cookie.

Pliki cookies są przechowywane na dysku do czasu wygaśnięcia. Jeżeli nie została podana data wygaśnięcia, to plik straci ważność po zamknięciu przeglądarki. Jeżeli zostały określone serwer i ścieżka, do której przeglądarka powinna wysłać informacje zawarte w cookie, to po otwarciu adresu pasującego do podanej domeny i ścieżki zawartość pliku cookie zostanie wysłana na serwer.

Przeglądarka może zostać skonfigurowana tak, że przed pobraniem pliku cookie będzie pojawiał się odpowiedni komunikat lub będą odrzucane wszystkie pliki cookies.

Tworzenie pliku cookie

W skryptach PHP pliki cookies tworzone są za pomocą funkcji `setcookie()`, która ma postać:

```
setcookie(nazwa[, wartość[, czas_trwania[, ścieżka_dostępu[, domena[,
bezpieczeństwo[, tylko_http]]]]]])
```

Argumenty funkcji to:

- `nazwa` — nazwa pliku cookie.
- `wartość` — wartość, która będzie przechowana w pliku cookie.

- *czas_trwania* — czas, po jakim plik cookie zostanie usunięty przez przeglądarkę. Jest on podawany jako liczba sekund, które upłyną od 1 stycznia 1970 roku.
- *ścieżka_dostępu* — ścieżka dostępu na serwerze dla pliku cookie. Ustawienie tej wartości na / spowoduje, że plik cookie będzie dostępny w całej domenie. Ustawienie `/www/skrypty` spowoduje dostępność pliku w podanym katalogu oraz w katalogach podrzędnych.
- *domena* — domena, w której plik cookie będzie dostępny.
- *bezpieczeństwo* — może przyjmować wartość `true` lub `false`. Ustawienie `true` spowoduje, że plik cookie będzie mógł być przesyłany tylko przez bezpieczne połączenia HTTPS. Domyślnym ustawieniem jest `false`.
- *tylko_http* — może przyjmować wartość `true` lub `false`. Ustawienie `true` spowoduje, że plik cookie będzie dostępny tylko przez protokół HTTP i nie będzie dostępny z poziomu skryptów osadzonych na stronie.

Jedynym argumentem, który musi wystąpić w funkcji, jest *nazwa*, pozostałe są opcjonalne.

Ponieważ pliki cookies są częścią nagłówka HTTP, funkcja `setcookie()` musi być wywołana, zanim jakiekolwiek dane zostaną wysłane do przeglądarki; powinna zatem zostać umieszczona w kodzie przed znacznikiem `<html>`.

Każdy wysłany plik cookie jest automatycznie umieszczany w superglobalnej tablicy `$_COOKIE`. Jej indeksami są nazwy plików cookies. Dostęp do pliku cookie jest możliwy po wpisaniu:

```
$_COOKIE['nazwa_pliku']
```

Przykład 7.102

```
<?php
setcookie("pismo", "Na skróty", time()+3600, "/", "localhost",false);
if (isSet($_COOKIE["pismo"]))
echo "<p>Jesteś naszym stałym klientem.</p>";
else "<p>Witamy po raz pierwszy na naszej stronie.</p>";
?>
```

Utworzony w przykładzie plik cookie zostanie zapisany przez przeglądarkę pod nazwą `pismo`. Zawartością pliku będzie tekst `Na skróty`. Do określenia czasu trwania pliku została wykorzystana funkcja `time()`, która zwraca bieżący czas. Do czasu zwróconego przez tę funkcję został dodany w sekundach czas trwania pliku (3600 s = 1 godz.). Ustawienie wartości / jako ścieżki dostępu oznacza, że plik będzie dostępny dla wszystkich stron na serwerze. Domena `localhost` oznacza, że plik będzie dostępny na lokalnym serwerze. Ostatni argument ustawiony na `false` oznacza, że plik cookie może być przesyłany przez nieszyfrowane połączenia.

Usuwanie pliku cookie

Jeżeli dla pliku cookie został ustalony czas ważności, to zostanie on automatycznie usunięty przez przeglądarkę po określonym czasie. Gdy czas ważności nie był określony, to plik zostanie usunięty przy zamykaniu przeglądarki. Aby usunąć samodzielnie plik cookie, można ustawić dla niego czas ważności, który już upłynął, albo ustawić jako *wartość* ciąg pusty lub wartość `false`. Pozostałe argumenty funkcji powinny być takie same jak argumenty tworzenia pliku.

Przykład 7.103

```
setcookie("pismo", "Na skróty", time()-100, "/", "localhost",0);
```

7.12.2. Zastosowania plików cookies

Jednym z typowych zastosowań plików cookies jest przechowywanie nazwy użytkownika i hasła na komputerze. Przy ponownej wizycie na stronie nie trzeba logować się ponownie i podawać hasła. Pliki cookies mogą przechowywać również inne informacje, np. nazwisko i imię użytkownika, czas ostatniej wizyty, zawartość koszyka itp.

Data ostatnich odwiedzin na stronie

Przykład 7.104

Wykorzystując plik cookie, określimy datę ostatnich odwiedzin strony przez użytkownika:

Skrypt tworzący plik cookie:

```
<?php
$mies = 2592000 + time();
setcookie("wizyta", date("F jS - g: ia"), $mies);
 ?>
```

Zostanie utworzony plik cookie `wizyta`. Termin wygaśnięcia to 30 dni od chwili jego utworzenia (2 592 000 = 60 s * 60 min * 24 godz. * 30 dni).

Plik tworzący skrypt odczytujący zawartość pliku cookie:

```
<?php
if (isSet($_COOKIE['wizyta']))
{
$ostatnia = $_COOKIE['wizyta'];
echo "Witamy ponownie! <br> Ostatni raz odwiedziłeś nas: ".$ostatnia;
}
else
{
```

```
echo "Witamy na naszej stronie!";
}
?>
```

W podanym przykładzie sprawdzamy, czy istnieje plik cookie. Jeżeli tak, to wyświetli się informacja powitalna oraz data ostatniej wizyty.

Przesyłanie danych użytkownika

Przykład 7.105

Dane użytkownika zostaną przekazane przez formularz, który będzie zawierał pole do wprowadzania imienia i nazwiska użytkownika. Pobrany z formularza parametr zawierający nazwisko i imię użytkownika zostanie przekazany do utworzonego pliku cookie.

Skrypt PHP realizujący podane zadanie ma postać:

```
<?php
 if (!isSet($_COOKIE['dane']) && !isSet($_POST['nazw'])){
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Dane_user</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form action="http://localhost/rejestracja.php" method="post">
Podaj nazwisko i imię:<br/>
<input type="text" name="nazw" value=" " size="35">
<p><input type="submit" value="Wyślij" name="wyslij"></p>
</form>
<?php
}
else {
if (isSet($_POST['nazw'])){
setcookie ('dane', $_POST['nazw'], time()+60*60*24*365);
echo "<p>Dziękujemy za wprowadzenie danych.</p>";
}
else
```

```
echo "<p>Witamy po raz kolejny! ".$_COOKIE['dane']."</p>";
}
?>
</body>
</html>
```

W podanym skrypcie rozpatrywane są trzy sytuacje. W pierwszej plik cookie nie został utworzony i z formularza nie zostały przesłane dane — wtedy należy wywołać formularz rejestracyjny. W drugiej dane z formularza zostały przesłane, ale nie istnieje plik cookie — wtedy należy utworzyć taki plik. W trzeciej dane z formularza zostały przesłane oraz istnieje plik cookie — wtedy wyświetli się komunikat witający użytkownika po raz kolejny na stronie.

Zapisana w kodzie instrukcja `if (!isSet ($_COOKIE ['dane']) && !isSet($_POST['nazw']))` sprawdza, czy istnieje plik cookie i czy z formularza został przekazany parametr zawierający nazwisko i imię użytkownika. Jeżeli nie, to zostanie utworzona strona zawierająca formularz. Z formularza za pomocą metody `POST` zostanie przekazany parametr `nazw`. Użyty operator `!` oznacza negację, a operator `&&` iloczyn logiczny.

Druga instrukcja `if (isSet($_POST['nazw']))` sprawdza, czy do skryptu zostały przesłane dane z formularza. Jeżeli tak, to za pomocą funkcji `setcookie()` do przeglądarki jest wysyłany plik cookie o nazwie `dane`. Jego wartością jest wartość parametru `nazw` pobrana z tablicy `$_POST`. Czas ważności pliku został ustawiony na `236 dni`. Na stronie wyświetli się tekst z podziękowaniem za wprowadzenie danych.

Trzecia sytuacja wystąpi, jeśli istnieje plik cookie. Wtedy na stronie wyświetli się informacja o rozpoznaniu użytkownika.

Zliczanie liczby odwiedzin

Przykład 7.106

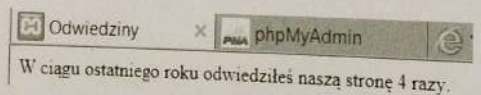
Zliczanie liczby odwiedzin strony przez użytkownika:

```
<?php
if (!isSet($_COOKIE ['odwiedz'])) {
$odw = 1;
}
else {
$odw = intval($_COOKIE['odwiedz']) +1;
}
setcookie("odwiedz", $odw, time()+60*60*24*365);
?>
```

```
<!DOCTYPE HTML>
<html>
<head>
<title>Odwiedziny</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<?php
if ($odw == 1)
$wyraz ="raz";
else
$wyraz ="razy";
echo "W ciągu ostatniego roku odwiedziłeś naszą stronę $odw $wyraz.<br/>";
?>
</body>
</html>
```

W podanym kodzie za pomocą funkcji `isSet()` sprawdzane jest, czy istnieje plik cookie o nazwie `odwiedz`. Jeżeli nie, to zmienna `$odw` przechowująca liczbę odwiedzin ustawiana jest na `1`. Jeżeli tak, to odczytywana jest z tablicy `$_COOKIE` wartość pliku `odwiedz`. Odczytana wartość za pomocą funkcji `intval()` jest zamieniana na liczbę całkowitą i zwiększana o `1`. Funkcja `intval()` jest niezbędna, ponieważ wartości odczytywane z tablicy `$_COOKIE` są ciągami znaków.

Po wykonaniu tych czynności do przeglądarki wysyłany jest plik cookie o nazwie `odwiedz`. Jego wartością jest liczba odwiedzin zapisana w zmiennej `$odw`. Na stronie wyświetli się informacja o liczbie odwiedzin (rysunek 7.53). Dodatkowo sprawdzana jest wartość zmiennej `$odw`. Jeżeli wynosi ona `1`, to wyświetlane zdanie będzie kończyło się słowem `raz`, dla pozostałych wartości będzie kończyło się słowem `razy`.



Rysunek 7.53. Odczyt pliku cookie z przechowywaną liczbą odwiedzin

Zadanie 7.18

Zmodyfikuj skrypty z przykładu 7.105, dodając instrukcje weryfikujące poprawność wprowadzonych danych.

Zadanie 7.19

Napisz skrypt, w którym będzie tworzony plik cookie zapamiętujący wprowadzoną przez formularz datę urodzin użytkownika. Skrypt powinien wyświetlać informację, za ile dni użytkownik będzie obchodził urodziny.

7.12.3. Sesje

Mechanizm sesji dostępny w języku PHP pozwala na identyfikację użytkownika i śledzenie jego poczynań na stronie internetowej. Sesja to czas, w którym użytkownik przegląda stronę internetową. W momencie pierwszego wejścia na stronę tworzony jest dla użytkownika specjalny identyfikator (losowa, unikatowa liczba wygenerowana przez PHP). Identyfikator jest przechowywany na komputerze użytkownika i na serwerze. Na komputerze użytkownika jest najczęściej zapisywany w pliku cookie, a jeżeli nie jest możliwe utworzenie takiego pliku, identyfikator jest dodawany automatycznie do adresu URL. Jeżeli identyfikator jest przekazywany jako element adresu, to nazwa i wartość identyfikatora są zapisywane w stałej `SID`.

Wszystkie zmienne związane z użytkownikiem są przechowywane na serwerze w tablicy superglobalnej `$_SESSION`.

Do rozpoczęcia sesji najczęściej jest wykorzystywana funkcja `session_start()`. Funkcja nie posiada żadnych argumentów, a jej wywołanie ma postać:

```
session_start();
```

W każdym skrypcie, który będzie korzystał z sesji, należy wywołać tę funkcję. Jeżeli wszystkie skrypty będą korzystały z sesji, można zmienić w pliku `php.ini` wartość opcji `session.auto_start` na `1`. Od tego momentu sesje będą otwierane automatycznie.

Gdy do zapisania identyfikatora wykorzystywany jest plik cookie, funkcja musi zostać wywołana, zanim do przeglądarki zostaną wysłane dane. Jej wywołanie powinno znaleźć się na początku pliku.

Do zakończenia sesji należy wykorzystać funkcję `session_destroy()`. Usuwa ona zasoby związane z sesją, nie usuwa natomiast zmiennych zapisanych na serwerze oraz pliku cookie na komputerze użytkownika. Nie posiada argumentów, a jej wywołanie ma postać:

```
session_destroy();
```

Identyfikator sesji można odczytać za pomocą funkcji `session_id()`. Wywołując tę funkcję, można zarówno odczytać wartość identyfikatora, jak i zmienić ją.

Przykład 7.107

```
<?php
session_start();
echo "<p>Identyfikator sesji to: ".session_id()."</p>";
?>
```

Podany kod wyświetli informację na temat wartości identyfikatora. Przy pierwszych odwiedzinach strony PHP wygeneruje dla użytkownika nowy identyfikator. Przy kolejnych odwiedzinach przydzieli mu ten sam identyfikator, a sesja zostanie wznowiona.

Zmienne sesji

Po rozpoczęciu sesji w tablicy `$_SESSION` mogą zostać zapisane dowolne zmienne związane z sesją. Aby zapisać zmienną, należy użyć konstrukcji:

```
$_SESSION['nazwa_zmiennej'] = wartość;
```

Do odczytania wartości zmiennej trzeba użyć indeksu odnoszącego się do nazwy zmiennej w tablicy `$_SESSION`:

```
$zmienna = $_SESSION['nazwa_zmiennej'];
```

Do sprawdzenia, czy określona zmienna istnieje, można zastosować funkcję `isSet()` w postaci:

```
isSet($_SESSION['nazwa_zmiennej'])
```

Funkcja `isSet()` zwraca wartość `true`, gdy zmienna została zarejestrowana, w przeciwnym razie zwraca wartość `false`. Zmienne zarejestrowane w czasie sesji powinny zostać wyrejestrowane przed jej zakończeniem. Służy do tego funkcja `unset()` w postaci:

```
unset($_SESSION['nazwa_zmiennej'])
```

Zmienne utworzone w sesji są przechowywane w pliku tymczasowym. Dodanie nowej zmiennej do tablicy `$_SESSION` powoduje dopisanie jej nazwy i wartości do tego pliku. Zapisane dane mogą zostać odczytane przez inne skrypty w obrębie tej samej sesji.

Przykład 7.108

```
<?php
session_start();
if (!isSet($_SESSION['ile'])) {
$_SESSION['ile'] = 0;
}
$_SESSION['ile']++;
echo "<p>Odwiedziłeś ".$_SESSION['ile']." razy nasze strony.</p>";
?>
```

Skrypt pokazuje, jak za pomocą zmiennej sesji `ile` można obliczać liczbę odwiedzin na stronie. Po otwarciu sesji sprawdzana jest zawartość tablicy `$_SESSION`. Jeżeli zmienna sesji `ile` nie jest zarejestrowana, zostanie jej przypisana początkowa wartość `0`. Jeżeli jest zarejestrowana, jej wartość zwiększy się o `1`. Następnie wyświetli się komunikat o liczbie odwiedzin.

7.12.4. Praktyczne zastosowania sesji

Uwierzytelnienie użytkownika

Mechanizm sesji zostanie wykorzystany do przeprowadzenia autoryzacji użytkownika. Przyjmiemy, że dostęp do witryny internetowej mają tylko zalogowani użytkownicy.

Przykład 7.109

Przygotujemy trzy pliki. Plik *strona.php* będzie wyświetlał zawartość strony głównej witryny. Plik *loguj.php* będzie zawierał procedury autoryzacji użytkownika i formularz logowania. Plik *wyloguj.php* będzie zawierał procedury wylogowania. W tej wersji będziemy sprawdzali uwierzytelnienie tylko jednego użytkownika o nazwie `janek` i haśle `jan23`.

Pierwszym krokiem będzie weryfikacja danych użytkownika. Po ich weryfikacji użytkownik będzie mógł zalogować się i korzystać ze strony. Przed opuszczeniem strony użytkownik powinien się wylogować.

Skrypt *loguj.php*:

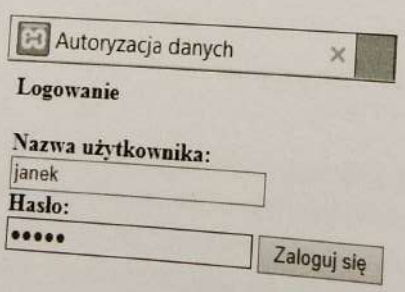
```
<?php
session_start();
if (isSet($_SESSION['log'])) {
header('location: strona.php');
exit();
}
else if (isSet($_POST['nazwa']) && isSet($_POST['haslo'])) {
if ($_POST['nazwa']=='janek' && $_POST['haslo'] == 'jan23') {
$_SESSION['log'] = 'nazwa';
header('location: strona.php');
exit();
}
else {
echo "Nieprawidłowe dane logowania<br/>";
}
}
?>
<!DOCTYPE HTML>
<html>
<head>
```

```
<title>Autoryzacja danych</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<div>
<form action="http://localhost/loguj.php" method="post">

<b>Logowanie</b><br/><br/>
<b>Nazwa użytkownika:</b><br/>
<input type=text name="nazwa" value="" size="25"><br/>
<b>Hasło:</b><br/>
<input type=password name="haslo" value="" size="25">
<input type="submit" value="Zaloguj się">
</form>
</div>
</body>
</html>
```

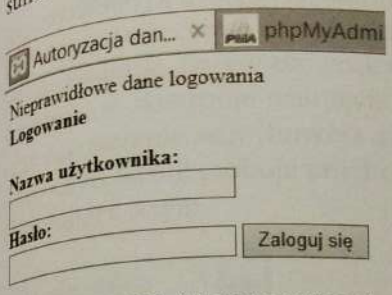
Skrypt autoryzacji użytkownika zaczyna się od wywołania funkcji `session_start()`, która rozpoczyna sesję. Następnie sprawdzane jest, czy istnieje zmienna sesji `$log`. Jest to zmienna, która powinna zawierać nazwę zalogowanego użytkownika. Jeżeli zmienna istnieje, to znaczy, że stronę próbuje otworzyć zalogowany użytkownik. Zostanie on przekierowany do strony głównej.

Jeżeli zmienna `$log` nie została ustawiona, ale w tablicy `$_POST` znajdują się dane (nazwa użytkownika i hasło), należy sprawdzić poprawność wprowadzonych danych. Następuje sprawdzenie, czy dane wprowadzone w czasie logowania są prawidłowe. Jeżeli weryfikacja przebiegła pomyślnie, oznacza to, że niezalogowany użytkownik wypełnił poprawnie formularz logowania i należy go zalogować (rysunek 7.54). Zmiennej sesji `$log` zostaje przypisana nazwa zalogowanego użytkownika i użytkownik zostanie przekierowany do strony głównej.



Rysunek 7.54. Próba logowania do witryny

Jeżeli weryfikacja danych jest negatywna, wyświetla się komunikat informujący o nieprawidłowych danych logowania, a następnie wyświetla się formularz logowania (rysunek 7.55).



Rysunek 7.55. Niepoprawne dane logowania

Jeżeli w tablicy `$_POST` nie zostały ustawione nazwa użytkownika i hasło, to znaczy, że na stronę próbuje dostać się niezalogowany użytkownik. Wyświetli się formularz logowania.

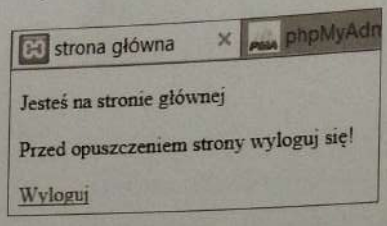
Kod strony głównej witryny został zapisany w skrypcie *strona.php*.

Skrypt *strona.php*:

```
<?php
session_start();
if (!isSet($_SESSION['log'])) {
header('location: loguj.php');
exit;
}
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Strona główna</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>

<p>Jesteś na stronie głównej</p>
<p>Przed opuszczeniem strony wyloguj się!</p>
<a href = "wyloguj.php">Wyloguj</a>
</body>
</html>
```

Po otwarciu sesji sprawdzane jest, czy istnieje zmienna logowania `$_SESSION['log']`. Jeżeli zmienna nie istnieje, to znaczy, że niezalogowany użytkownik próbuje otworzyć stronę — zostanie on przekierowany do strony logowania. Na stronie dostępnej tylko dla zalogowanych użytkowników wyświetlana jest informacja, że przed jej opuszczeniem użytkownik powinien się wylogować (rysunek 7.56).



Rysunek 7.56. Prawidłowo zalogowany użytkownik ma dostęp do strony głównej

Próba wylogowania wywołuje kolejny skrypt — *wyloguj.php*. Skrypt wylogowania zawiera polecenia zakończenia sesji oraz usunięcia jej zmiennych.

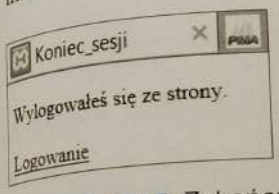
Skrypt *wyloguj.php*:

```
<?php
session_start();
if (isSet($_SESSION['log'])) {
unset($_SESSION['log']);
}
else {
header('location: loguj.php');
exit;
}

$s = session_destroy();
?>
<!DOCTYPE HTML>
<html>
<head>
<title> Koniec_sesji</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p>Wylogowałeś się ze strony.</p>
```

```
<a href = "loguj.php">Logowanie</a>
</body>
</html>
```

Pierwszym poleceniem skryptu jest wywołanie funkcji `session_start()`. Następnie jest sprawdzane, czy użytkownik wywołujący procedurę wylogowania był zalogowany. Jeżeli tak, to następuje usunięcie pliku cookie i zmiennej sesji `$_SESSION['log']` oraz zakończenie sesji (funkcja `session_destroy`) (rysunek 7.57). Jeżeli niezalogowany użytkownik próbuje uruchomić procedurę wylogowania, pojawia się komunikat informujący o tym.



Rysunek 7.57. Zakończenie sesji. Wylogowanie

Zadanie 7.20

Zmodyfikuj skrypty z przykładu 7.109 tak, aby weryfikacja danych dotyczyła wielu użytkowników. Utwórz tabelę *użytkownicy*, która będzie zawierała loginy oraz hasła użytkowników korzystających z danej strony internetowej. Zdefiniuj funkcję, która będzie przeszukiwała tablicę i zwracała wartość `true`, gdy użytkownik o podanym loginie i haśle istnieje. Wykorzystaj ją do zdefiniowania nowych funkcji skryptów.

Zadanie 7.21

Przy założeniu, że Twoja witryna internetowa składa się z kilku stron, napisz kod, który pozwoli na dostęp do stron witryny tylko ze strony głównej. Wejście bezpośrednio na wybraną stronę (z pominięciem strony głównej) powinno zostać zablokowane.

Koszyk zakupów

Mechanizm sesji może zostać wykorzystany do tworzenia w sklepie internetowym listy wybranych przez klienta artykułów.

Przykład 7.110

Zostaną przygotowane dwa skrypty. W pierwszym (*lista.php*) zostanie utworzony formularz, za pomocą którego możliwe będzie wybranie z listy różnych produktów. Wybrane produkty będą przechowywane w zmiennych sesji zapisanych w postaci tablicy. Drugi (*koszyk.php*) będzie wyświetlał listę produktów wybranych przez użytkownika i umieszczonych w koszyku.

Skrypt *lista.php*:

```
<?php
session_start();
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Koszyk zakupów</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p><b>Lista artykułów</b></p>
<?php
if (isSet($_POST['lista'])) {
  if (!empty($_SESSION['koszyk'])) {
  $koszyk=array_unique(array_merge(unserialize($_SESSION['koszyk']),
  $_POST['lista']));
$_SESSION['koszyk'] = serialize($koszyk);
}
else {
$_SESSION['koszyk'] = serialize($_POST['lista']);
}
echo "<p>Wybrane produkty zostały umieszczone w koszyku</p>";
}
?>
<form action="http://localhost/lista.php" method="post">
<p><b><font size="4"><b>Wybór produktu:</font></b><br>
<select name="lista[]" multiple="multiple" size = "9">
<option value="Monitor"> Monitor</option>
<option value="Drukarka"> Drukarka</option>
<option value="Klawiatura"> Klawiatura</option>
<option value="Myszka"> Myszka</option>
<option value="Głośniki"> Głośniki</option>
```

```
<option value="Kamera internetowa"> Kamera internetowa</option>
<option value="Słuchawki"> Słuchawki</option>
<option value="Stacja DVD"> Stacja DVD</option>
<option value="Dysk twardy"> Dysk twardy</option>
</select></p>
<p><input type="submit" value="Wyślij"></p>
</form>
<p><a href="koszyk.php">Przejdź do koszyka</a></p>
</body>
</html>
```

W skrypcie zostały użyte funkcje działające na tablicach:

- `array_unique(tablica)` — pobiera tablicę i zwraca nową tablicę bez powtarzających się wartości;
- `array_merge(tablica1, tablica2, ...)` — łączy elementy wybranych tablic i zwraca tablicę wynikową;
- `unserialize(łańcuch)` — pobiera łańcuch znaków i przekształca go na wartość.

Po otwarciu lub wznowieniu sesji (funkcja `session_start()`) sprawdzamy, czy formularz z listą produktów został wysłany (tablica `$_POST['lista']`). Jeżeli tak, to sprawdzamy, czy istnieje tablica `$_SESSION['koszyk']`. Jeżeli taka tablica istnieje, to znaczy, że pewne produkty zostały już wybrane i przesłane do koszyka. Zostaną one połączone (funkcja `array_merge()`) z listą produktów znajdujących się w tablicy `$_POST['lista']`, a wynik będzie zapisany do zmiennej `$koszyk`.

Dane umieszczane w sesji w tablicy `$_SESSION` muszą być ciągiem znaków (w formacie tablicy), dlatego przed ich zapisaniem należy za pomocą funkcji `serialize()` zamienić je na taki ciąg znaków. Podobnie, pobierając w sesji dane z tablicy `$_SESSION`, należy zamienić je na wartości (funkcja `unserialize()`).

Przed połączeniem tablic dane pobrane z tablicy `$_SESSION['koszyk']` zostaną przekształcone na wartości (funkcja `unserialize()`). Po połączeniu tablic funkcja `array_unique()` usunie powtarzające się produkty umieszczone w koszyku. Następnie tablicy `$_SESSION['koszyk']` zostanie przypisana wartość zmiennej `$koszyk`.

Jeżeli nie istnieje tablica `$_SESSION['koszyk']`, to znaczy, że koszyk jest zapisywany po raz pierwszy. Po utworzeniu tablicy zostaną jej przypisane wartości tablicy `$_POST['lista']`.

Jeżeli nie istnieje tablica `$_POST['lista']`, to znaczy, że nie został wypełniony i wysłany formularz z listą produktów. Wyświetli się formularz, za pomocą którego użytkownik będzie mógł wybrać produkty do koszyka (rysunek 7.58).



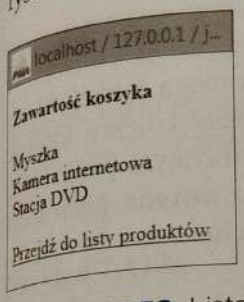
Rysunek 7.58. Formularz z listą dostępnych produktów

Skrypt *koszyk.php*:

```
<?php
session_start();
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Koszyk</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p><b>Zawartość koszyka</b></p>
<?php
if (isSet($_SESSION['koszyk'])) {
foreach(unserialize($_SESSION['koszyk']) as $produkt) {
echo "<li>" . $produkt . "</li>";
}
}
else {
echo "brak sesji";
}
?>
<p><a href="lista.php">Przejdź do listy produktów</a></p>
```

```
</body>
</html>
```

Podany kod otwiera sesję (funkcja `session_start()`), następnie sprawdza, czy istnieje tablica `$_SESSION['koszyk']`. Jeżeli istnieje, to w pętli wyświetlana jest lista wartości tablicy `$_SESSION['koszyk']`. Wynik działania kodu został pokazany na rysunku 7.59.



Rysunek 7.59. Lista zamówionych produktów

Zadanie 7.22

Zmodyfikuj kod skryptu z przykładu 7.110 tak, aby była wyświetlana informacja o liczbie produktów w koszyku, oraz utwórz nowy skrypt, który będzie wyświetlał zawartość koszyka i umożliwiał usunięcie z niego wybranych produktów.

7.13. Sieć

W języku PHP informacje dotyczące środowiska sieciowego (np. połączenia sieciowe, obsługiwane przeglądarki, położenie skryptów) są przechowywane w tablicy superglobalnej `$_SERVER`. Wpisy w tej tablicy są tworzone przez serwer WWW, dlatego mogą nieznacznie różnić się w zależności od stosowanego środowiska systemowego. Każda informacja zapisana w tablicy składa się z nazwy klucza oraz odpowiadającej mu wartości.

7.13.1. Adresy IP

Adres IP, z którego nastąpiło połączenie z witryną, jest zapisany w kluczu `REMOTE_ADDR`. Jego znajomość pozwala między innymi na kontrolę dostępu do strony. Można zablokować połączenia z wybranych adresów lub pozwolić na połączenia tylko z niektórymi adresami.

Przykład 7.111

Skrypt blokowania wybranych adresów IP. Lista blokowanych adresów znajduje się w pliku `ip.txt`.

```
<?php
$ip_tab=file("ip.txt");
```

```
foreach($ip_tab as $b){
if(trim($b)==$_SERVER['REMOTE_ADDR']){
echo "Twój adres IP został zablokowany.";
exit;
}
}
?>
```

W podanym przykładzie odczytane z pliku *ip.txt* adresy IP za pomocą funkcji `file()` zapisywane są w tablicy `$ip_tab`. Każda komórka tablicy zawiera kolejny adres IP. Adresy są odczytywane w pętli `foreach` i porównywane ze zwróconym przez klucz `REMOTE_ADDR` adresem użytkownika, który próbuje połączyć się z witryną. Jeżeli adres znajduje się na liście, wyświetli się komunikat o jego zablokowaniu.

7.13.2. Połączenie FTP

Za pomocą funkcji dostępnych w języku PHP można realizować obsługę protokołu FTP służącą między innymi do wysyłania oraz pobierania plików z serwera FTP.

Przed samym wysłaniem lub pobraniem pliku należy nawiązać połączenie z serwerem FTP za pomocą funkcji `ftp_connect()`. Ma ona postać:

```
ftp_connect(nazwa_hosta[, port[, timeout]])
```

- *nazwa_hosta* — to nazwa lub adres IP serwera, z którym należy się połączyć,
- *port* — numer portu, na którym działa usługa FTP,
- *timeout* — określa czas, po którym próba połączenia z serwerem zostanie przerwana, jeżeli połączenie nie zostało nawiązane.

Dwa ostatnie argumenty są opcjonalne i jeżeli nie zostaną podane, przyjmują wartości domyślne — dla argumentu `port`: `21`, dla argumentu `timeout`: `90`. Funkcja zwraca identyfikator połączenia wykorzystywany w kolejnych funkcjach lub wartość `false`, jeżeli połączenia nie udało się nawiązać.

Po nawiązaniu połączenia kolejnym działaniem jest zalogowanie się. Umożliwia to funkcja `ftp_login()` w postaci:

```
ftp_login(ftp_id, nazwa_użytkownika, hasło)
```

- *ftp_id* — to identyfikator połączenia zwrócony przez funkcję `ftp_connect()`.

Logowanie powinno zostać wykonane również wtedy, gdy połączenie dotyczy serwerów FTP oferujących dostęp anonimowy. Wtedy jako nazwę użytkownika podajemy `anonymous`, a hasło to adres e-mail.

Pobieranie pliku można zrealizować za pomocą funkcji `ftp_fget()` w postaci:

```
ftp_fget(ftp_id, plik_lokalny, plik_zdalny, tryb[, pozycja])
```

- *ftp_id* — to identyfikator połączenia zwrócony przez funkcję `ftp_connect()`,
- *plik_lokalny* — to nazwa pliku na dysku lokalnym, do którego będą zapisywane pobrane dane,
- *plik_zdalny* — to nazwa pliku do pobrania (ze ścieżką dostępu),
- *tryb* — to tryb transmisji; przyjmuje wartość `FTP_ASCII` (transmisja tekstowa) lub `FTP_BINARY` (transmisja binarna),
- *pozycja* — określa po wznowieniu przerwanej transmisji pozycję pliku, od której powinny być wysyłane dane; jest to argument opcjonalny.

Po zakończeniu transmisji połączenie powinno zostać zamknięte. Służy do tego funkcja `ftp_close()` w postaci:

```
ftp_close(ftp_id)
```

Przykład 7.112

```
<?php
$serwer = "ADRES_FTP";
$user = "LOGIN";
$pass = "HASŁO";
$plik_lok = "plik.txt";
$plik_zdal = "plik.txt";
$dir = "/FOLDER";
if (!$id = ftp_connect($serwer)) {
exit ("Błąd połączenia z serwerem");
if (!ftp_login($id, $user , $pass)) {
exit ("Błąd logowania do serwera FTP");
echo "Połączenie z serwerem nawiązane.<br/>";
if (!ftp_get($id, $plik_lok, $dir."/".$plik_zdal, FTP_BINARY)) {
exit ("Błąd podczas pobierania pliku!");
}
else {
echo "Plik $plik_zdal został pobrany.<br/>";
}
ftp_close($id);
?>
```

Przykład przedstawia sposób połączenia z serwerem FTP i pobrania pliku *plik.txt*. Za zmienne zdefiniowane na początku kodu należy podstawić własne dane.

Zadanie 7.23

Zdefiniuj funkcję `pobierz_plik`, która będzie realizowała pobieranie pliku (jak w kodzie podanym w przykładzie 7.112). Argumentami przekazywanymi do funkcji powinny być: adres serwera FTP, użytkownik, hasło, folder, nazwa pliku do zapisania oraz nazwa pliku pobieranego.

Zadanie 7.24

Napisz skrypt, który umożliwi połączenia ze stroną internetową tylko z wybranych adresów IP. Adresy powinny być zapisane w pliku tekstowym.

7.14. Bazy danych w PHP

Tworząc aplikacje internetowe, niezbędne dane możemy przechowywać w plikach. Ale gdy rośnie ich liczba, rozwiązaniem jest zastosowanie bazy danych. Pliki nadal mogą być wykorzystywane np. do przechowywania podstawowych ustawień aplikacji, ale do gromadzenia i przetwarzania większych ilości danych stosuje się relacyjne bazy danych. Do ich obsługi wykorzystywane są systemy zarządzania bazami danych. Najpopularniejszym systemem zarządzania bazami danych, który współpracuje z PHP, jest MySQL. Obsługę baz danych można realizować w sposób proceduralny lub obiektowy.

Aby możliwa była współpraca języka PHP z systemem MySQL, musi istnieć baza, z którą można się połączyć. Użytkownik chcący połączyć się z taką bazą musi posiadać login i hasło, znać nazwę oraz adres tej bazy danych.

7.14.1. Połączenie z bazą danych

Nawiązanie połączenia

Do połączenia z bazą danych służy funkcja `mysql_connect()`, która ma postać:

```
mysql_connect('nazwa_hosta', 'nazwa_użytkownika', 'hasło')
```

Nazwa_hosta oznacza nazwę lub adres IP serwera, na którym została umieszczona baza danych. Funkcja zwraca identyfikator połączenia lub wartość `false`, jeżeli połączenie nie zostało zrealizowane. Jeżeli argumenty *nazwa_hosta*, *nazwa_użytkownika* i *hasło* zostaną pominięte, interpreter zastosuje wartości domyślne:

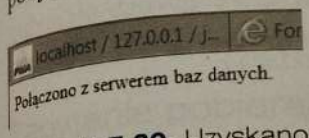
- *nazwa_hosta* — localhost:3306,
- *nazwa_użytkownika* — użytkownik domyślny,
- *hasło* — pusty ciąg znaków.

Przykład 7.113

```
<?php
$do_bazy = mysql_connect('localhost', 'adam', '2%azxs');
if (!$do_bazy) {
```

```
exit ("Błąd połączenia z serwerem MySQL.");
}
else {
echo "Połączono z serwerem baz danych.<br/>";
}?>
```

W podanym kodzie po wywołaniu funkcji `mysql_connect()` sprawdzane jest, czy połączenie zostało nawiązane. Jeżeli tak, to wyświetli się komunikat o uzyskanym połączeniu (rysunek 7.60), w przeciwnym razie wyświetli się komunikat o braku tego połączenia i program zakończy swoje działanie.



Rysunek 7.60. Uzyskano połączenie z serwerem baz danych

Za pomocą funkcji `mysql_get_server_info()` można wyświetlić informacje o uzyskanym połączeniu. Funkcja zwraca nazwę hosta, nazwę użytkownika oraz hasło.

Do współpracy z innymi systemami baz danych stosuje się inne wersje funkcji `connect()`.

Dla PostgreSQL:

```
pg_connect('host=nazwa_hosta user=nazwa użytkownika password=hasło
dbname=baza_danych')
```

Dla MSSQL:

```
mssql_connect('nazwa_hosta', 'nazwa_użytkownika', 'hasło')
```

Kończenie połączenia

Do zakończenia połączenia z bazą danych używana jest funkcja `mysql_close()` w postaci:

```
mysql_close([identyfikator])
```

Podany jako argument funkcji identyfikator jest identyfikatorem połączenia, które ma zostać zamknięte. Jeżeli identyfikator będzie pominięty, zamknięte zostanie połączenie ostatnio otwarte.

Jeżeli funkcja `mysql_close()` nie zostanie wywołana, połączenie będzie automatycznie zamknięte, gdy skrypt zakończy swoje działanie. Funkcja zwróci wartość `true`, gdy operacja zamykania połączenia zakończy się prawidłowo, w przeciwnym razie zwróci wartość `false`.

Wybór bazy

Po nawiązaniu połączenia z serwerem baz danych można określić bazę danych, z której będą pobierane dane. Służy do tego funkcja `mysql_select_db()` w postaci:

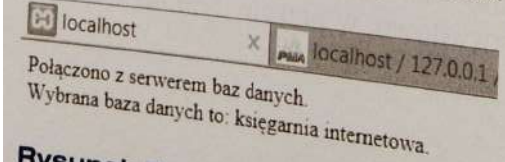
```
mysql_select_db('nazwa_bazy'[,identyfikator])
```

Argument `nazwa_bazy` określa bazę, do której będą wysyłane zapytania. Opcjonalnie może wystąpić drugi argument, który jest identyfikatorem połączenia. Jeżeli identyfikator zostanie pominięty, wykorzystywane będzie połączenie ostatnio otwarte.

Przykład 7.114

```
<?php
$do_bazy = mysql_connect('localhost', 'adam', 'hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
else {
echo "Połączono z serwerem baz danych.<br/>";
 if(!mysql_select_db('ksiegarnia_internetowa' ,$do_bazy)) {
   echo "Błąd podczas wyboru bazy danych<br/>";
 }
 else {
    echo "Wybrana baza danych to: księgarnia internetowa.<br/>";
    //operacje na wybranej bazie danych
 }
mysql_close($do_bazy);
}
?>
```

W podanym kodzie zostało zdefiniowane połączenie dla użytkownika `adam` z hasłem `hasło`. Jeżeli połączenie nie zostało nawiązane, wyświetli się komunikat i skrypt zakończy działanie. Jeśli połączenie zostało nawiązane, następuje wybranie bazy danych do dalszej współpracy (rysunek 7.61). Gdy nie będzie można połączyć się z bazą danych, wyświetli się komunikat i nastąpi zamknięcie połączenia. Jeżeli wszystko przebiegło pomyślnie, wykonywane są polecenia związane z dostępną bazą danych.



Rysunek 7.61. Połączenie z wybraną bazą danych

7.14.2. Zapytania do bazy danych

Po nawiązaniu połączenia z wybraną bazą danych można wysyłać do niej zapytania. Są one realizowane za pomocą funkcji `mysql_query()` zapisanej w postaci:

```
mysql_query('zapytanie'[,identyfikator])
```

Argumentem funkcji jest treść zapytania. Opcjonalnie można podać drugi argument określający identyfikator połączenia.

Zwracane przez funkcję wartości zależą od typu zapytania. Jeżeli zapytanie pobierało dane, to wartością zwracaną jest identyfikator do tych danych. Jeżeli zapytanie nie pobierało danych, zwracana jest wartość `true`. Jeżeli wykonanie zapytania się nie powiodło, zwracana jest wartość `false`.

Zapytanie pobierające dane

Dla zapytań pobierających dane typu `SELECT` zwracany jest identyfikator zasobów. Może on zostać wykorzystany do odczytania tych danych. Do ich odczytania można użyć funkcji `mysql_fetch_row()` lub `mysql_fetch_array()`.

Funkcja `mysql_fetch_row()` po wywołaniu odczytuje jeden wiersz tabeli i zapisuje go w kolejnym wierszu tablicy. Aby odczytać całą zawartość tabeli, należy wywołać funkcję w pętli. Gdy wszystkie dane zostaną odczytane, funkcja zwraca wartość `false`.

Przykład 7.115

```
while ($wiersze = mysql_fetch_row($wynik)) {
//przetwarzanie danych z zapytania
}
```

W podanym przykładzie zmienna `$wynik` zawiera identyfikator zasobów zwróconych przez zapytanie. W pętli zostaną odczytane kolejne wiersze tabeli zwrócone przez zapytanie i umieszczone w tablicy `$wiersze`.

Funkcja `mysql_num_rows()` zwraca jako wartość liczbę wierszy znajdujących się w wyniku zapytania. Może być użyta do zbudowania pętli odczytującej kolejne wiersze zwrócone przez zapytanie.

Przykład 7.116

```
$ile = mysql_num_rows($wynik);
for ($i = 0; $i < $ile; $i++) {
$wiersze = mysql_fetch_row($wynik) {
//przetwarzanie danych z zapytania
}
```

Podobnie jak w przykładzie 7.115 w pętli zostaną odczytane kolejne wiersze tabeli zwrócone przez zapytanie i umieszczone w tablicy `$wiersze`.

Funkcja `mysql_fetch_array()` po wywołaniu odczytuje całą tabelę i zwraca tablicę asocjacyjną. Kluczami tej tablicy są nazwy kolumn zwróconej przez zapytanie tabeli.

Przykład 7.117

```
$tab = mysql_fetch_array($wynik);
```



UWAGA

Na potrzeby kolejnych przykładów zakładamy, że istnieje baza danych *Ksiegarnia_internetowa* utworzona w części II podręcznika, w rozdziale 5.

Przykład 7.118

Dysponujemy utworzoną w MySQL bazą danych *Ksiegarnia_internetowa*. Odczytamy dane z tabeli `Klient`.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Odczyt danych z tabeli Klient</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>

<?php
$do_bazy = mysql_connect('localhost', 'user', 'hasło');
if (!$do_bazy) {
echo "Błąd połączenia z serwerem MySQL.<br/>";
?>
</body>
</html>
<?php
exit;
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
echo "Błąd podczas wyboru bazy Księgarnia internetowa<br/>";
?>
```

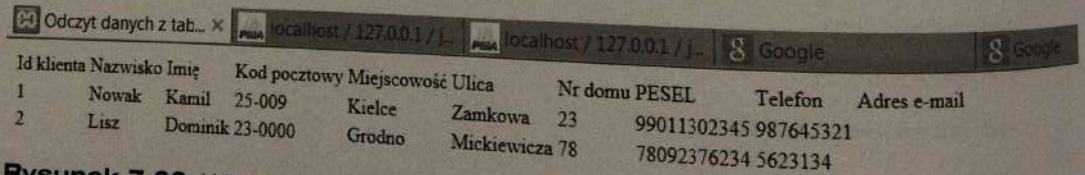
```
</body>
</html>
<?php
exit;
}
$zapytanie = mysql_query('SELECT * FROM Klient');
if(!$zapytanie) {
mysql_close();
echo "Błąd w zapytaniu<br/>";
?>
</body>
</html>
<?php
exit;
}
?>

<table>
<tr>
<td>Id klienta</td>
<td>Nazwisko</td>
<td>Imię</td>
<td>Kod pocztowy</td>
<td>Miejscowość</td>
<td>Ulica</td>
<td>Nr domu</td>
<td>PESEL</td>
<td>Telefon</td>
<td>Adres e-mail</td>
</tr>

<?php
while ($wiersze = mysql_fetch_row($zapytanie)) {
echo "<tr>";
echo "<td>$wiersze[0]</td>";
```

```
echo "<td>$wiersze[1]</td>";
echo "<td>$wiersze[2]</td>";
echo "<td>$wiersze[3]</td>";
echo "<td>$wiersze[4]</td>";
echo "<td>$wiersze[5]</td>";
echo "<td>$wiersze[6]</td>";
echo "<td>$wiersze[7]</td>";
echo "<td>$wiersze[8]</td>";
echo "<td>$wiersze[9]</td>";
echo "</tr>";
}
?>
</table>
<?php
mysql_close();
?>
</body>
</html>
```

W podanym przykładzie po połączeniu z serwerem i wybraniu bazy danych zostaje wywołana funkcja `mysql_query()`, która wysyła do serwera zapytanie SQL (`SELECT * FROM Klient`). Wynik zwrócony przez funkcję zostaje przypisany zmiennej `$zapytanie`. Jeżeli wynik wynosi `false`, to znaczy, że w zapytaniu wystąpił błąd i skrypt kończy działanie. Jeżeli wszystko przebiegło poprawnie, w HTML tworzona jest tabela w celu sformatowania wyświetlanych danych otrzymanych z zapytania. Następnie w kodzie PHP wyniki zapytania są pobierane w pętli za pomocą funkcji `mysql_fetch_row()`, która pobiera kolejne wiersze tabeli i umieszcza je w tablicy. Indeks `0` tablicy zawiera wartość pobraną z pierwszej kolumny wiersza tabeli, indeks `1` wartość z drugiej kolumny itd. Wynik działania kodu został pokazany na rysunku 7.62.



Rysunek 7.62. Wyświetlanie danych otrzymanych z zapytania do bazy

Klauzula WHERE

Klauzula `WHERE` jest dodawana do instrukcji `SELECT` wtedy, gdy należy wybrać tylko te wiersze, które spełniają określone kryterium.

Przykład 7.119

```
<?php
$do_bazy = mysql_connect('localhost', 'user', 'hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
exit ("Błąd podczas wyboru bazy Księgarnia internetowa.");
}
$zapytanie = mysql_query("SELECT * FROM Klient WHERE Nazwisko = 'Nowak'");

if(!$zapytanie=== true) {
mysql_close();
exit("Błąd w zapytaniu<br/>");
}
while ($tab = mysql_fetch_array($zapytanie)) {
echo $tab['nazwisko'] . " " . $tab['imie'];
echo "<br/>";
}
?>
```

W wyniku wykonania podanego skryptu z bazy danych zwrócone zostaną dane osób, które mają nazwisko `Nowak`. Dane zostały pobrane za pomocą funkcji `mysql_fetch_array()`, która zapisała je w tablicy asocjacyjnej `$tab`. Kluczami tablicy `$tab` są nazwy kolumn tabeli SQL. Konstrukcja `$tab['nazwisko']` i `$tab['imie']` odwołuje się do wartości zapisanych w tablicy i wyświetla nazwiska i imiona klientów wybranych instrukcją `SELECT`.

Zapytanie wstawiające dane

Dla zapytań modyfikujących zawartość tabel funkcja `mysql_query()` zwraca wartość `true` lub `false`. Natomiast za pomocą funkcji `mysql_affected_rows()` można odczytać, ile wierszy zostało zmodyfikowanych przez zapytanie. Funkcja ma postać:

```
mysql_affected_rows([identyfikator])
```

Argument *`identyfikator`* jest opcjonalny i zawiera identyfikator połączenia z serwerem. Jeżeli zostanie pominięty, działania dotyczą ostatnio otwartego połączenia.

Zapytanie dodające nowe dane do tabeli jest realizowane przez polecenie `INSERT INTO`.

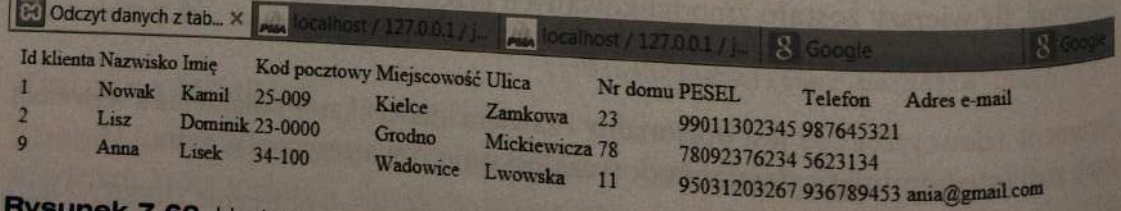
Przykład 7.120

Do tabeli `Klient` w bazie danych *Ksiegarnia_internetowa* zostanie dodany nowy wpis.

```
<?php
$do_bazy = mysql_connect('localhost', 'user', 'hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
exit ("Błąd podczas wyboru bazy Księgarnia internetowa.");
}

$dodaj = "INSERT INTO Klient VALUES(NULL, 'Anna', 'Lisek', '34-100', 'Wado-
wice', 'Lwowska', '11', '95031203267', '936789453', 'ania@gmail.com');
if(!$zapytanie= mysql_query($dodaj)) {
mysql_close();
exit("Błąd w zapytaniu");
}
$ile = mysql_affected_rows();
echo "Liczba dodanych rekordów do tabeli Klient wynosi: $ile<br/>";
mysql_close();
?>
```

W podanym kodzie po prawidłowym połączeniu z serwerem i wybraniu bazy danych zostaje utworzona zmienna `$dodaj`, której zostaje przypisane zapytanie SQL (`INSERT INTO`) dodające do tabeli `Klient` nowy wpis. Za pomocą funkcji `mysql_query()` jest ono wysyłane do serwera. Instrukcja `if` sprawdza, czy funkcja `mysql_query()` zwróciła wartość `false`, co oznacza, że zapytanie nie zostało obsłużone. Wtedy skrypt kończy działanie. Jeżeli zapytanie zostało przyjęte, to zmiennej `$ile` przypisywana jest wartość zwrócona przez funkcję `mysql_affected_rows()`, określająca, ile wierszy w bazie danych zostało zmienionych. Ta informacja jest wyświetlana w przeglądarce (rysunek 7.63).



Id klienta	Nazwisko	Imię	Kod pocztowy	Miejscowość	Ulica	Nr domu	PESEL	Telefon	Adres e-mail
1	Nowak	Kamil	25-009	Kielce	Zamkowa	23	99011302345	987645321	
2	Lisz	Dominik	23-0000	Grodno	Mickiewicza	78	78092376234	5623134	
9	Anna	Lisek	34-100	Wadowice	Lwowska	11	95031203267	936789453	ania@gmail.com

Rysunek 7.63. Umieszczenie nowych danych w tabeli

Inną metodą dodawania nowych danych do tabeli jest wprowadzanie ich za pomocą formularza.

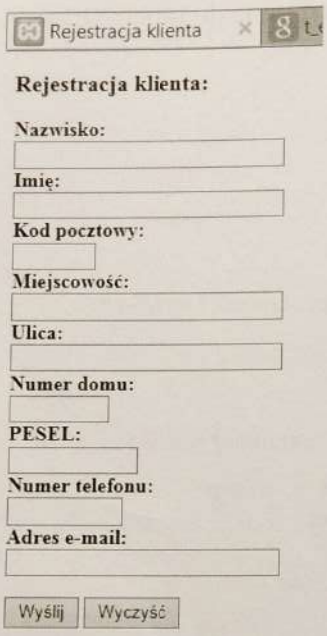
Przykład 7.121

W tym przykładzie zostanie utworzony skrypt HTML (*form_klient.html*) zawierający formularz do wprowadzania danych klienta (rysunek 7.64). Z kolei skrypt PHP (*dodaj_klienta.php*) będzie dodawał wprowadzone przez formularz dane do tabeli `Klient`.

Skrypt *form_klient.html*:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Rejestracja klienta</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form action="http://localhost/dodaj_klienta.php" method="post">
<p><b><font size="4"><b>Rejestracja klienta:</font></b></p>
Nazwisko:<br/>
<input type="text" name="nazw" value="" size="30"><br>
Imię:<br/>
<input type="text" name="im" value="" size="30"><br>
Kod pocztowy:<br/>
<input type="text" name="kod" value="" size="5"><br>
Miejscowość:<br/>
<input type="text" name="mia" value="" size="30"><br>
Ulica:<br/>
<input type="text" name="ul" value="" size="30"><br>
Numer domu:<br/>
<input type="text" name="nr" value="" size="7"><br>
PESEL:<br/>
<input type="text" name="pe" value="" size="11"><br>
Numer telefonu:<br/>
<input type="text" name="tel" value="" size="9"><br>
Adres e-mail:<br/>
<input type="text" name="adr" value="" size="30"><br>
```

```
<p><input type="submit" value="Wyślij" name="wyslij">
<input type="reset" value="Wyczyść" name="zeruj"></p>
</form>
</body>
</html>
```



Rysunek 7.64. Formularz do wprowadzania danych do bazy

Dane wprowadzone do formularza zostaną przesłane do pliku *dodaj_klienta.php*.

Skrypt *dodaj_klienta.php*:

```
<?php
$do_bazy = mysql_connect('localhost', 'user', 'hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
exit ("Błąd podczas wyboru bazy Księgarnia internetowa.");
}
```

```
$dodaj = "INSERT INTO Klient VALUES (NULL, '$_POST['nazw']',
'$_POST['im']', '$_POST['kod']', '$_POST['mia']', '$_POST['ul']', '$_
POST['nr']', '$_POST['pe']', '$_POST['tel']', '$_POST['adr']')";

$zapytanie= mysql_query($dodaj);
if(!$zapytanie === true) {
echo "Nowy klient został dodany do bazy.";
}
else {
echo "Nowy klient nie został dodany do bazy!";
}
mysql_close();
?>
```

Podany skrypt jest podobny do skryptu z przykładu 7.120. Jedyna zmiana to użycie w instrukcji `INSERT INTO` zmiennej `$_POST['nazwa_pola']` w miejscu, gdzie poprzednio wstawione były wartości danych.

Aktualizowanie danych

Aktualizowanie danych jest realizowane za pomocą instrukcji `UPDATE`.

Przykład 7.122

W tabeli `Klient` zostaną zaktualizowane dane klienta `Nowak Adam`. Aby zaktualizować te dane, należy wykonać kilka czynności — wyszukać dane klienta w bazie danych, wyświetlić je za pomocą formularza na stronie internetowej, pobrać przesłane z formularza zaktualizowane dane i zapisać je w bazie.

Pierwszy skrypt będzie pobierał dane z bazy i wyświetlał je na stronie w postaci formularza.

```
<?php
$do_bazy = mysql_connect('localhost', 'user','hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
exit ("Błąd podczas wyboru bazy Księgarnia internetowa.");
}
```

```
$zapytanie = mysql_query("SELECT * FROM Klient WHERE Nazwisko = 'Nowak'
and Imie = 'Adam'");
$ile = mysql_num_rows($zapytanie);
mysql_close();

if(!$zapytanie === true) {
mysql_close();
exit("Błąd w zapytaniu.");
}

if ($ile = 0)
exit ( "Brak danych");
if ($ile > 1)
exit ("Nie można wyodrębnić danych");

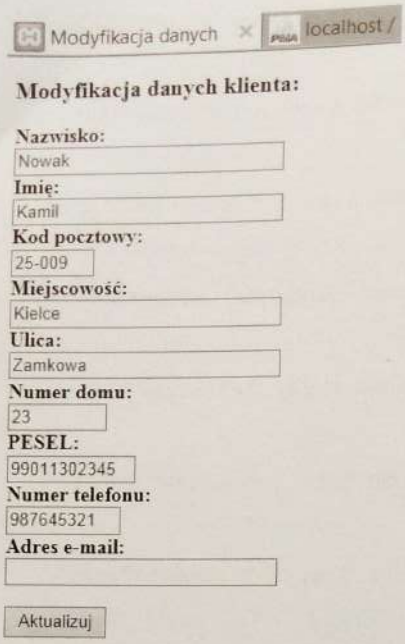
$wiersz = mysql_fetch_array($zapytanie);
$nazw = $wiersz['nazwisko'];
$im = $wiersz['imie'];
$kod = $wiersz['kod_pocztowy'];
$mia = $wiersz['miejscowosc'];
$ul = $wiersz['ulica'];
$nr = $wiersz['nr_domu'];
$pe = $wiersz['PESEL'];
$tel = $wiersz['telefon'];
$adr = $wiersz['adres_e_mail'];
?>

//formularz
<!DOCTYPE HTML>
<html>
<head>
<title>Modyfikacja danych</title>
</head>
<body>
<form action="http://localhost/aktualizuj.php" method="post">
```

```
<p><b><font size="4"><b> Modyfikacja danych klienta:</font></b></p>
Nazwisko:<br/>
<input type="text" name="nazw" value="<?php echo $nazw; ?>" size="30"><br>
Imię:<br/>
<input type="text" name="im" value="<?php echo $im; ?>" size="30"><br>
Kod pocztowy:<br/>
<input type="text" name="kod" value="<?php echo $kod; ?>" size="5"><br>
Miejscowość:<br/>
<input type="text" name="mia" value="<?php echo $mia; ?>" size="30"><br>
Ulica:<br/>
<input type="text" name="ul" value="<?php echo $ul; ?>" size="30"><br>
Numer domu:<br/>
<input type="text" name="nr" value="<?php echo $nr; ?>" size="7"><br>
PESEL:<br/>
<input type="text" name="pe" value="<?php echo $pe; ?>" size="11"><br>
Numer telefonu:<br/>
<input type="text" name="tel" value="<?php echo $tel; ?>" size="9"><br>
Adres e-mail:<br/>
<input type="text" name="adr" value="<?php echo $adr; ?>" size="30"><br>

<p><input type="submit" value="Aktualizuj" name="akt"></p>
</form>
</body>
</html>
```

Po połączeniu z serwerem i wybraniu bazy danych zostaje wywołana funkcja `mysql_query()`, która wysyła do serwera zapytanie SQL (`SELECT * FROM Klient WHERE Nazwisko = 'Nowak' and Imie = 'Adam'`). W wyniku zostaną zwrócone dane wybranego klienta. Zwrócone dane zostają przypisane do zmiennej `$zapytanie`. Następnie funkcją `mysql_num_rows()` sprawdzone będzie, ile wierszy zostało odczytanych z bazy. Jeżeli `0`, to brak danych do dalszego przetwarzania i następuje zakończenie działania skryptu. Jeżeli więcej niż `1`, to nie można określić, czyje dane powinny być aktualizowane, i również następuje zakończenie działania skryptu. Jeżeli zostały odczytane dane jednego klienta, to będą one przypisane do zmiennych i nastąpi przejście do kodu HTML. W kodzie HTML następuje wyświetlenie formularza z bieżącymi danymi pobranymi z utworzonych zmiennych (rysunek 7.65). Użytkownik może zaktualizować swoje dane i przesłać je do skryptu *aktualizuj.php*.



Rysunek 7.65. Bieżące dane pobrane z bazy danych do aktualizacji

Kolejnym krokiem będzie aktualizacja danych w bazie.

Skrypt *aktualizuj.php*:

```
<?php
$do_bazy = mysql_connect('localhost', 'user', 'hasło');
if (!$do_bazy) {
exit ("Błąd połączenia z serwerem MySQL.");
}
if(!mysql_select_db('Ksiegarnia_internetowa')) {
mysql_close();
exit ("Błąd podczas wyboru bazy Księgarnia internetowa.");
}

$aktualny = "UPDATE Klient SET nazwisko = '$_POST['nazw']', imie = '$_
POST['im']', kod_pocztowy = '$_POST['kod']', miejscowosc = '$_POST['mia']',
ulica = '$_POST['ul']', nr_domu = '$_POST['nr']', PESEL = '$_POST['pe']',
telefon = '$_POST['tel']', adres_e_mail = '$_POST['adr']' WHERE Nazwisko =
'Nowak' and Imie = 'Adam'";

$zapytanie= mysql_query($aktualny);
if(!$zapytanie === true) {
```

```
echo "Dane zostały zaktualizowane.";
}
else {
echo "Błąd zapytania! Dane nie zostały zaktualizowane";
}
mysql_close();
?>
```

Skrypt instrukcją UPDATE aktualizuje dane przesłane z formularza i przekazane za pomocą zmiennej $_POST.

Tworzenie bazy danych

Do tworzenia bazy danych jest używana instrukcja SQL CREATE DATABASE. Należy dołączyć ją do funkcji mysql_query().

Przykład 7.123

```
$baza = "CREATE DATABASE Moje_kino";
if (mysql_query($baza)) {
echo "Baza została utworzona";
}
else {
echo "Błąd! Nie można utworzyć bazy";
}
```

W przykładzie został pokazany fragment kodu PHP tworzący bazę danych *Moje_kino*.

Tworzenie tabeli

Do tworzenia tabeli w istniejącej bazie danych jest używana instrukcja SQL CREATE TABLE. Należy dołączyć ją do funkcji mysql_query().

Przykład 7.124

```
$tab = "CREATE TABLE Film (Tytul_filmu VARCHAR(30), Rezyser VARCHAR(30),
Rok_prod  VARCHAR(4)";
if (mysql_query($tab)) {
echo "Tabela Film została utworzona";
}
else {
echo "Błąd! Nie można utworzyć tabeli Film";
}
```

W przykładzie został pokazany fragment kodu PHP tworzący tabelę Film.

7.14.3. Biblioteka PDO

PDO (ang. *PHP Data Objects*) jest rozszerzeniem języka PHP, które udostępnia jednolity, uniwersalny interfejs do komunikacji z bazami danych w technice obiektowej. Od wersji 5.1 jest elementem środowiska PHP i poza dodaniem sterowników dla wykorzystywanych baz danych nie wymaga innych działań. Sterowniki różnych baz danych udostępniają takie same metody, chociaż mogą również dodawać własne, specyficzne metody.

Zaletą PDO jest to, że tworzone skrypty mogą korzystać z tych samych metod, niezależnie od wykorzystywanej bazy. Kolejną zaletą jest walidacja danych wysyłanych w zapytaniach oraz filtrowanie tych danych pod kątem zabezpieczeń przed atakami.

Dostępne są sterowniki między innymi dla baz: MySQL, PostgreSQL, SQLite, Oracle, MS SQL/Sybase.

Nawiązanie połączenia

Aby nawiązać połączenie z bazą danych, należy za pomocą konstruktora utworzyć nowy obiekt klasy PDO w postaci:

```
PDO(źródło_danych, nazwa_użytkownika, hasło, opcje)
```

- *źródło_danych* — inaczej DSN (ang. *Data Source Name*), jest to specjalny ciąg znaków opisujący rodzaj bazy danych i sposób połączenia. Zawiera takie informacje, jak nazwa sterownika, adres serwera, nazwa bazy, numer portu.
- *opcje* — to tablica zawierająca dodatkowe opcje związane z połączeniem.

Parametrem wymaganym jest *źródło_danych*. Pozostałe mogą być pominięte.

Wersja uproszczona źródła danych dla MySQL wygląda następująco:

```
mysql:host=nazwa_serwera;port=numer_portu;dbname=nazwa_bazy
```

- *nazwa_serwera* — określa nazwę lub adres serwera baz danych, np. localhost 127.0.0.1.
- *numer_portu* — port, przez który nastąpi połączenie z bazą danych. Jeżeli parametr zostanie pominięty, przyjęta będzie wartość standardowa.
- *nazwa_bazy* — nazwa bazy, z którą nastąpi połączenie.

Przykład 7.125

```
mysql:host=localhost;port=3306;dbname=moja_baza
```

W podanym przykładzie źródło danych pozwoli na nawiązanie połączenia z bazą *moja_baza* znajdującą się na serwerze MySQL pracującym na lokalnym komputerze. Połączenie będzie realizowane na porcie 3306.

Przykład 7.126

```
<?php

$user = "Janek";
```

```
$pass = "hasło";
$dsn = "mysql:host=localhost;dbname=ksiegarnia_internetowa";
try {
$pdo = new PDO($dsn, $user , $pass);
echo 'Połączenie nawiązane!';
}
catch(PDOException $e) {
echo 'Błąd połączenia: ' . $e->getMessage();
exit;
}
?>
```

W podanym przykładzie ciąg źródło danych został przypisany do zmiennej `$dsn`. Utworzony obiekt klasy PDO został przypisany do zmiennej `$pdo`. Jeżeli podczas nawiązywania połączenia wystąpi błąd, zostanie zgłoszony wyjątek `PDOException`. Musi on zostać przechwycony, ponieważ domyślny komunikat o błędzie, który wygeneruje PHP, wyświetli nazwę użytkownika i hasło do bazy danych.



UWAGA

PDO posiada obsługę błędów opartą na wyjątkach. Jeżeli zgłoszone wyjątki nie zostaną obsłużone, to skrypt zakończy działanie z komunikatem PHP, a w przypadku błędu podczas łączenia z bazą będzie on zawierał login i hasło do bazy danych.

Kończenie połączenia

Połączenie, które zostało nawiązane za pomocą obiektu PDO, istnieje aż do momentu usunięcia obiektu z pamięci. Obiekt zostanie usunięty automatycznie po zakończeniu działania skryptu. Można go również usunąć, przypisując zmiennej obiektowej przechowującej odwołanie do obiektu wartość `null`.

Przykład 7.127

```
$pdo = null;
```

Pobieranie danych

Zapytanie pobierające dane jest wysyłane za pomocą metody `query()`. Treść zapytania jest przekazywana do tej metody w postaci argumentu.

```
$obiekt_PDO->query("Treść_zapytania")
```

Metoda `query()` zwraca obiekt klasy `PDOStatement`, który zawiera wynik wykonania zapytania.

Przykład 7.128

```
<?php
try {
$pdo = new PDO('mysql:host=localhost;dbname=ksiegarnia_internetowa',
'root');
$zapytanie = $pdo->query('SELECT nazwisko, imie, pesel FROM Klient');
foreach($zapytanie as $wiersz) {
echo $wiersz['nazwisko'].', '.$wiersz['imie'].', '.$wiersz['pesel'].'<br/>';}
$pdo = null;
}
catch(PDOException $e) {
echo "Błąd połączenia:" . $e->getMessage();
}
?>
```

W podanym przykładzie metoda `query()` zwraca wynik wykonania zapytania. Następnie za pomocą pętli `foreach` kolejne wiersze wyniku zostaną zapisane do tablicy asocjacyjnej `$wiersz` i będą wyświetlone instrukcją `echo`. Po zakończeniu wyświetlania wyników połączenie zostanie zamknięte.

Aktualizacja danych

Zapytania służące do aktualizacji danych typu `INSERT` czy `UPDATE` i inne, które nie zwracają wyników, są wysyłane za pomocą metody `exec()`. Po wykonaniu zapytania metoda ta zwraca liczbę określającą ilość zmodyfikowanych wierszy.

Przykład 7.129

```
<?php
try {
$pdo = new PDO('mysql:host=localhost;dbname=ksiegarnia_internetowa',
'root');
$dodaj = $pdo->exec("INSERT INTO Klient (nazwisko, imie, pesel)
VALUES('Górski', 'Jakub', '92120203229')");
if ($dodaj > 0) {
echo "Dodano: ".$dodaj." wierszy<br/>";
}
else {
echo "Błąd podczas dodawania danych!<br/>";
}
```

```
$pdo = null;
}
catch(PDOException $e) {
echo "Błąd połączenia:" . $e->getMessage();
}
?>
```

W podanym przykładzie metoda `exec()` wysyła zapytanie dodające nowe wiersze do tabeli oraz zwraca liczbę dodanych wierszy. Następnie informacja ta jest wyświetlana. Jeżeli operacja nie została wykonana poprawnie, pojawią się odpowiednie komunikaty.

Zadanie 7.25

Zmodyfikuj skrypt z podanego wyżej przykładu tak, aby dane do tabeli były wprowadzane za pomocą formularza wyświetlanego na stronie internetowej.

PYTANIA KONTROLNE

1. Wymień obszary zastosowań języka PHP.
2. W jaki sposób w kodzie HTML oznaczane są skrypty PHP?
3. Jakiego rodzaju informacje przechowują zmienne predefiniowane?
4. Czym różni się operator inkrementacji `++$x` od operatora `$x++`?
5. Jakie jest przeznaczenie *stałych magicznych*?
6. Gdzie stosowana jest pętla `foreach`?
7. Czym różnią się zmienne statyczne od zmiennych lokalnych?
8. Jaką wartość w języku PHP przyjmuje *znacznik czasu* (*timestamp*)?
9. Podaj definicję klasy w programowaniu obiektowym.
10. Co zawierają składowe klasy?
11. Podaj przykład tworzenia obiektu w języku PHP.
12. Na czym polega hermetyzacja w programowaniu obiektowym?
13. Czym jest konstruktor w programowaniu obiektowym?
14. Jakie zadanie w programowaniu obiektowym wykonuje destruktor?
15. Na czym polega dziedziczenie w programowaniu obiektowym?
16. Na czym polega obsługa wyjątków w PHP?
17. Czym różni się przesyłanie danych z formularza na serwer za pomocą metody `POST` od przesyłania danych za pomocą metody `GET`?
18. Jakie jest przeznaczenie plików cookies?

PYTANIA KONTROLNE ciąg dalszy

19. Jak działa dostępny w języku PHP mechanizm sesji?
20. Jakie parametry należy podać w kodzie PHP, aby nawiązać połączenie z bazą danych?
21. Jakie jest przeznaczenie biblioteki PDO dostępnej w języku PHP?

ZADANIA

1. W MySQL utwórz bazę danych *Moja_szkola* lub wykorzystaj bazę danych *Moja_szkola* tworzoną w części II podręcznika. Baza danych powinna zawierać tabelę `Uczen` z danymi uczniów oraz tabelę `Rejestracja` zawierającą dwa pola: `login` i `haslo`. Zostanie ona wykorzystana do rejestracji loginów i haseł użytkowników strony.
2. Utwórz stronę internetową, która za pomocą skryptów PHP będzie komunikowała się z tą bazą danych. Strona główna ma zawierać formularz logowania oraz formularz rejestracji nowych uczniów.
3. Utwórz skrypt, który:
   - dane z formularza rejestracyjnego zapisze w tabeli Uczen,
   - login i hasło zapisze w tabeli Rejestracja,
   - po zalogowaniu przekieruje użytkownika do strony umożliwiającej mu zamieszczanie komentarza,
   - wyświetli na tej stronie dotychczasowe komentarze użytkowników,
   - wyświetli nowy komentarz poniżej wcześniejszych komentarzy,
   - wyświetli na stronie przycisk wylogowania, a po jego kliknięciu wyloguje użytkownika ze strony.

8

Technologia .NET

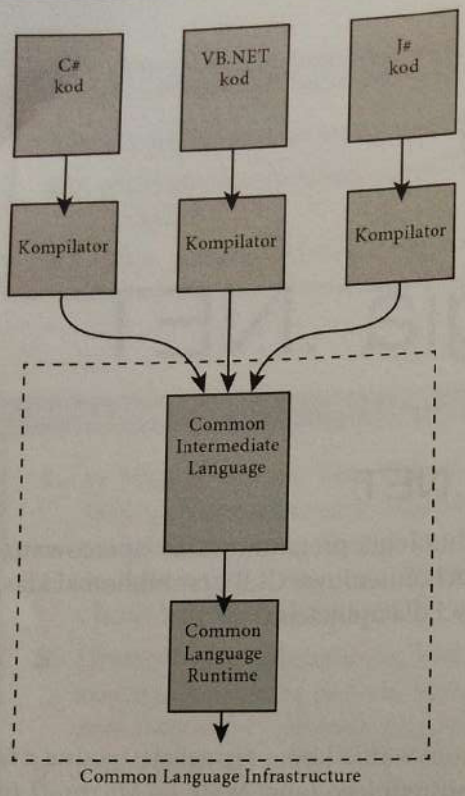
8.1. Założenia platformy .NET

Platforma programistyczna .NET jest nową technologią programowania opracowaną przez firmę Microsoft. Obejmuje środowisko uruchomieniowe CLR oraz biblioteki klas, które dostarczają podstawowych funkcjonalności dla aplikacji.

8.1.1. Wykonanie kodu

Najważniejszą cechą platformy jest sposób wykonywania kodu. Narzędzia zgodne z tą platformą kompilują kod źródłowy do języka pośredniego (ang. *MSIL — Microsoft Intermediate Language* lub *CIL — Common Intermediate Language*). Przygotowany w ten sposób kod może być rozpowszechniany jako gotowa aplikacja. Uruchomienie tej aplikacji na komputerze użytkownika powoduje przekompilowanie kodu pośredniego przez wspólne środowisko uruchomieniowe *CLR* (ang. *Common Language Runtime*) na kod maszynowy procesora dostępnego na komputerze. Taki sposób kodowania umożliwia optymalizację kodu dla posiadanego modelu procesora i pełne jego dopasowanie do możliwości komputera.

Narzędziem łączącym CIL i CLR jest *Common Language Infrastructure* (*CLI*). Jest to część platformy .NET wykorzystywana jako środowisko uruchomieniowe programów napisanych w różnych językach programowania (rysunek 8.1). Każdy język programowania, który spełni standardy platformy .NET, może mieć dostęp do jej bibliotek. W rezultacie tę samą skompilowaną wersję aplikacji można uruchamiać na każdej platformie sprzętowej, dla której został utworzony CLR. Oprócz wersji dla Windows zawartej w Microsoft .NET Framework istnieją implementacje CLR do systemów Linux i Mac OS X.



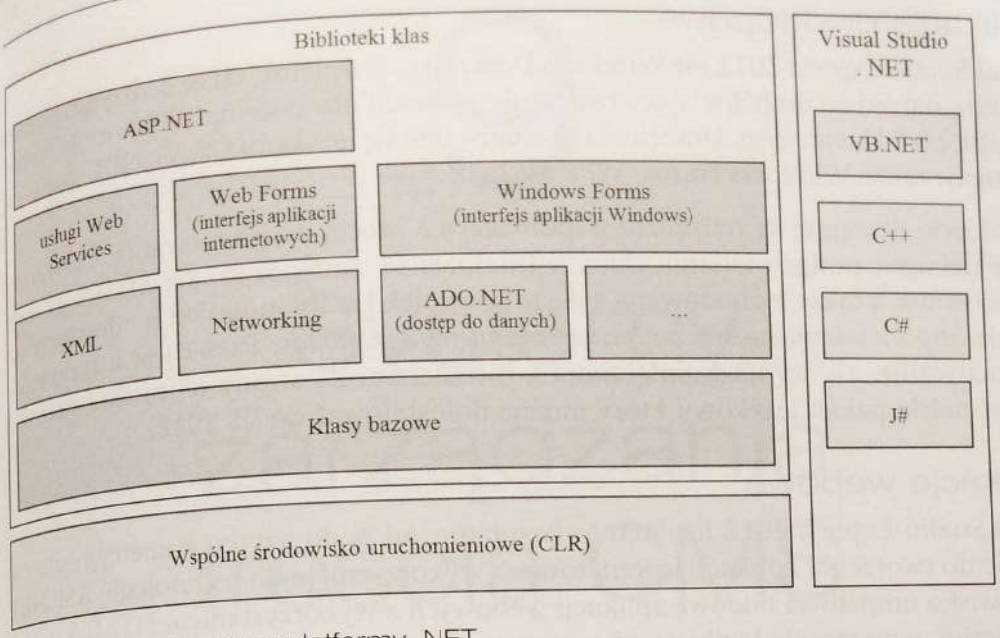
Rysunek 8.1. Sposób wykonania kodu platformy .NET

Technologia .NET nie jest związana z żadnym językiem programowania, a przeznaczone dla niej programy mogą być pisane w różnych językach. Każdy język programowania, który spełni odpowiednie standardy, ma dostęp do biblioteki .NET. Obecnie ponad 40 języków programowania jest zgodnych z tą platformą. Firma Microsoft dostarcza na platformę .NET języki: C++/CLI, C#, J#, Visual Basic .NET, JScript .NET.

8.1.2. Struktura platformy .NET

Zadaniem platformy .NET jest zarządzanie różnymi elementami systemu: kodem aplikacji, pamięcią i zabezpieczeniami. W środowisku tym można tworzyć oprogramowanie działające po stronie serwera internetowego (IIS) oraz pracujące w systemach, na które istnieje działająca implementacja tej platformy.

Głównymi składnikami platformy są wspólne środowisko uruchomieniowe oraz hierarchicznie zorganizowane biblioteki klas. Jej uzupełnieniem są narzędzia programistyczne (rysunek 8.2).



Rysunek 8.2. Struktura platformy .NET

Biblioteki ułatwiają obsługę graficznych interfejsów użytkownika, dostęp do baz danych i plików oraz komunikację sieciową. Biblioteka ADO.NET ułatwia dostęp do baz danych, a ASP.NET wykorzystywana jest do tworzenia dynamicznych witryn internetowych i stanowi podstawę budowy aplikacji internetowych. Windows Forms jest biblioteką wizualną umożliwiającą projektowanie graficznego interfejsu użytkownika.

Zintegrowane środowisko programistyczne to Visual Studio. Umożliwia ono pisanie programów w językach C#, Visual Basic .NET, C++, J#. Istnieje również darmowa jego wersja Visual Studio Express.

Uzupełnieniem platformy .NET jest platforma .NET Compact Framework przeznaczona dla urządzeń mobilnych (telefony, tablety) wyposażonych w system operacyjny oparty na Windows CE.

Platforma .NET najpełniej obsługuje systemy z rodziny Microsoft Windows, ale powstają również niezależne projekty, takie jak Mono i dotGNU, których celem jest stworzenie narzędzi kompatybilnych z platformą .NET, przeznaczonych dla użytkowników innych systemów operacyjnych.

8.1.3. Narzędzia programistyczne

Dla zastosowań związanych z internetem powstała technologia tworzenia dynamicznych stron WWW — ASP.NET. Aplikacje internetowe mogą być tworzone przy użyciu języka C#, Visual Basic.NET lub innego języka obsługującego platformę.

Aplikacje desktopowe

Visual Studio Express 2012 for Windows Desktop to bezpłatne, także do użytku komercyjnego, narzędzie umożliwiające tworzenie aplikacji dla systemu Windows w trybie pulpitu, czyli klasycznym. Umożliwia posługiwanie się językami C#, VB.NET oraz C++, obsługuje także Windows Forms, WPF i Win32.

W pakiecie dostępne są narzędzia wspomagające proces programowania: zaawansowany debuger, podpowiadanie składni IntelliSense, zintegrowana z MSDN pomoc kontekstowa, a także rozbudowana galeria kontrolek i szablonów. Jest on dostępny na stronie *http://www.microsoft.com/visualstudio/eng/downloads*. Pakiet jest instalowany automatycznie, ale wymaga połączenia z internetem. Ze strony tej można również pobrać polski pakiet językowy, który można doinstalować do VS 2012.

Aplikacje webowe

Visual Studio Express 2012 for Web to bezpłatne, także do użytku komercyjnego, narzędzie do tworzenia aplikacji internetowych wykorzystujących technologię ASP.NET. Środowisko umożliwia budowę aplikacji webowych z wykorzystaniem HTML5, CSS3 i JavaScript, a po stronie logiki z wykorzystaniem języków C# i Visual Basic. Aplikacja integruje się z SQL Server, co pozwala na projektowanie baz danych równolegle, bez konieczności opuszczania środowiska Visual Studio. Testowanie aplikacji jest możliwe dzięki wbudowanemu serwerowi IIS Express, który jest zminimalizowaną wersją pełnego serwera webowego IIS. Pakiet można pobrać ze strony *http://www.microsoft.com/visualstudio/eng/downloads*. Jest on instalowany automatycznie, ale wymaga połączenia z internetem.

PYTANIA KONTROLNE

1. Omów sposób wykonywania kodu platformy .NET.
2. Jaką rolę podczas wykonywania kodu odgrywa *Common Language Infrastructure* (*CLI*)?
3. Wymień podstawowe języki programowania dla platformy .NET.
4. Jaką rolę dla platformy .NET odgrywa biblioteka ASP.NET?

9

Systemy zarządzania treścią CMS

Systemy zarządzania treścią (CMS, ang. *Content Management System*) to aplikacje internetowe, które pozwalają na łatwe tworzenie i modyfikowanie oraz rozbudowę strony internetowej. Systemy CMS generują przy wykorzystaniu odpowiednich szablonów gotowe strony internetowe. Tworzenie strony oparte na szablonach pozwala na łatwą i szybką zmianę jej wyglądu, jej aktualizowanie i późniejszą rozbudowę. Tworzenie treści i sposobu ich prezentacji odbywa się za pomocą prostego w obsłudze interfejsu użytkownika, zazwyczaj przez rozbudowane formularze i moduły.

Systemy zarządzania treścią budowane są zgodnie z zasadą oddzielenia treści (zawartości informacyjnej witryny) od wyglądu (sposobu jej prezentacji). Wprowadzane treści są zapisywane w bazie danych. Następnie system CMS generuje dynamicznie strony internetowe na podstawie treści pochodzących z bazy danych oraz odpowiednio opracowanych szablonów. Pozwala to na łatwe i elastyczne zarządzanie treścią.

Tworzenie stron oparte na wykorzystaniu szablonów sprawia, że zmiana wyglądu witryny internetowej polega na przygotowaniu i wymianie szablonu. Dodatkowo te same dane mogą być prezentowane jednocześnie w różny sposób.

Obecnie najpopularniejszymi systemami zarządzania treścią są: WordPress, Quick.Cms, Drupal oraz Joomla!. Aby nie opisywać pobieżnie każdego z tych systemów, szczegółowo jest opisany tylko jeden, ostatni z wymienionych.

9.1. Joomla!

Joomla! jest bezpłatnym programem typu CMS. Może być stosowany w celach prywatnych lub komercyjnych bez dodatkowych opłat.

Joomla! jest systemem instalowanym na serwerze stron internetowych. Dlatego wcześniej należy na komputerze zainstalować serwer WWW (np. Apache), bazę danych (np. MySQL) i język PHP. Może to być również platforma webowa WAMP dla systemu Windows, LAMP dla systemu Linux lub wieloplatformowy XAMPP, względnie Microsoft Web Platform. Innym rozwiązaniem jest korzystanie z bezpłatnego lub komercyjnego konta założonego u jednego z dostawców usług internetowych, który oferuje dostęp do podanych wyżej zasobów.

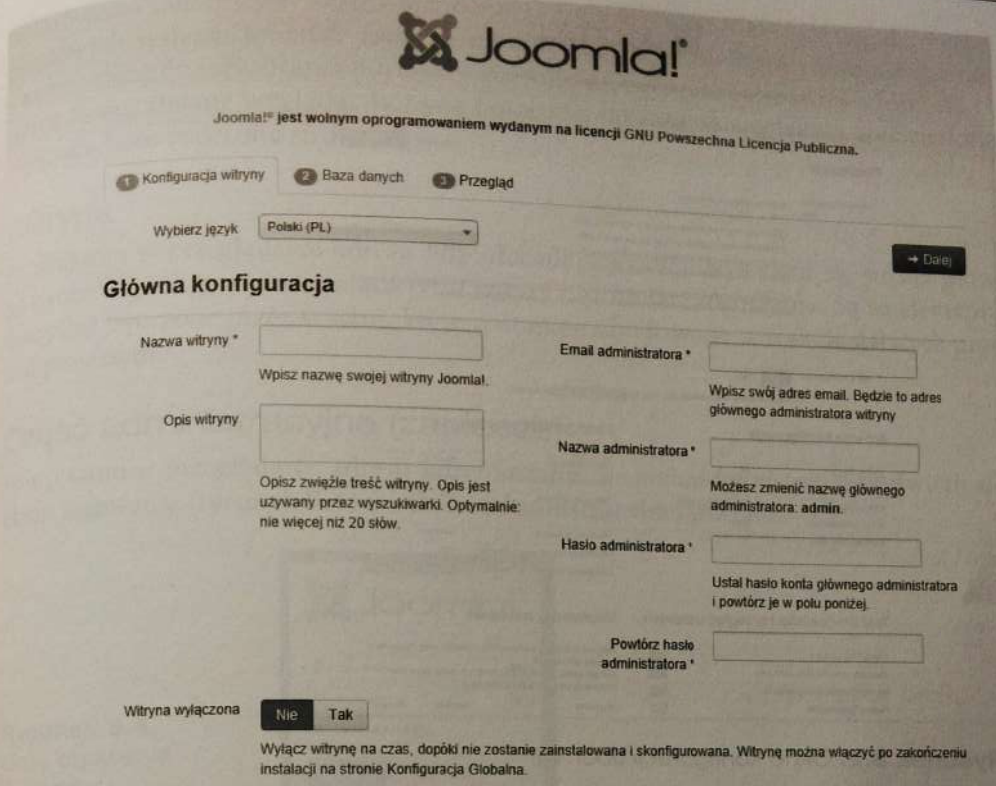
Do tworzenia stron internetowych z wykorzystaniem systemu Joomla! nie jest wymagana znajomość języków PHP, HTML i CSS oraz baz danych SQL, gdyż praca odbywa się w trybie wizualnym, natomiast kod HTML i CSS jest tworzony automatycznie. Dla systemu Joomla! dostępne są w internecie liczne rozszerzenia, które rozbudowują możliwości witryny.

Plik instalacyjny pakietu można pobrać ze strony *http://www.joomla.org/download.html*. Jeżeli pakiet jest instalowany na lokalnym komputerze, na którym został zainstalowany pakiet XAMPP, plik należy rozpakować w podkatalogu *htdocs* katalogu, w którym został zainstalowany XAMPP, np. *C:\xampp\htdocs \joomla*.

UWAGA

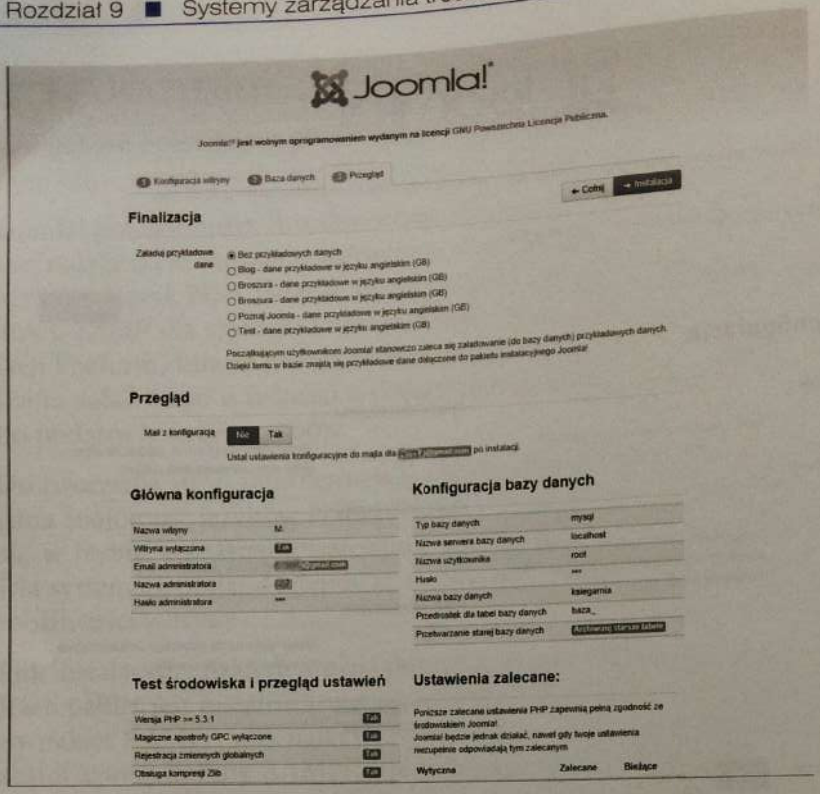
Na potrzeby podręcznika została zainstalowana wersja Joomla! 3.1.1.

Instalacja pakietu jest prosta, ponieważ Joomla! posiada działający w przeglądarce instalator, który poprowadzi nas przez cały proces instalacji. Podczas instalowania należy podać nazwę logowania i hasło do bazy danych, nazwę tworzonej witryny oraz opcjonalnie (jeżeli będzie wykorzystywana ta funkcjonalność) nazwę i hasło do konta FTP. Podczas instalowania systemu Joomla! niezbędne są uruchomiony serwer WWW oraz przeglądarka internetowa. Po wpisaniu w przeglądarce adresu, pod którym zostały umieszczone rozpakowane pliki Joomla!, np. *http://localhost/joomla*, wyświetli się okno instalatora (rysunek 9.1).



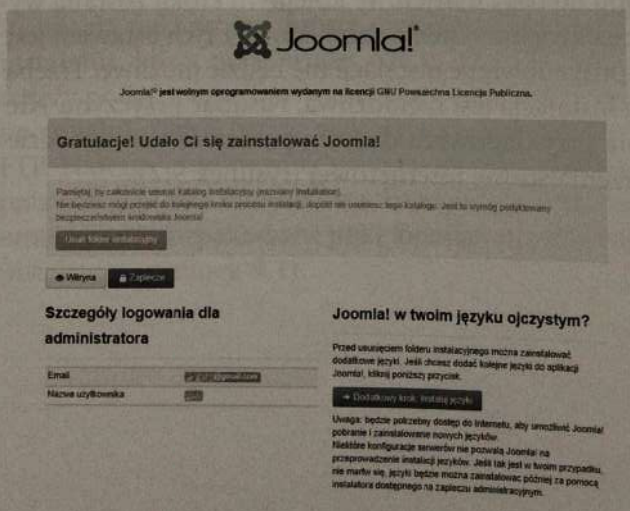
Rysunek 9.1. Okno instalatora Joomla!

Należy sprawdzić, czy został wybrany język polski, i wypełnić pola konfiguracji witryny, a następnie pola konfiguracji dostępu do bazy danych. W kolejnym kroku zostaną wyświetlone opcje konfiguracyjne oraz ustawienia systemu. Jeżeli któreś z tych ustawień jest wyróżnione kolorem czerwonym, kontynuowanie instalacji nie będzie możliwe. Trzeba sprawdzić, z jakiego powodu dalsza instalacja jest niemożliwa, i usunąć przyczynę. Nie należy zaznaczać opcji wczytywania przykładowych danych, ponieważ trzeba będzie je usunąć przed rozpoczęciem tworzenia strony internetowej (rysunek 9.2).



Rysunek 9.2. Okno konfiguracji Joomla!

Po wybraniu przycisku *Instalacja* pakiet zostanie zainstalowany. Jeżeli instalacja przebiegła prawidłowo, wyświetli się kolejne okno (rysunek 9.3). Aby móc zakończyć instalację, w oknie tym należy wybrać opcję usunięcia folderu instalacyjnego.



Rysunek 9.3. Okno podsumowujące instalację Joomla!

Po zainstalowaniu pakietu można rozpocząć pracę z witryną. Mamy dostęp do dwóch składowych systemu Joomla!: *zaplecza administracyjnego* i *witryny*. *Zaplecze administracyjne* służy do zarządzania portalem. Pozwala na tworzenie struktury strony, dodawanie menu, zmianę wyglądu. *Witryna* pokazuje, jak będzie wyglądała wyświetlona w przeglądarce tworzona strona.

Witryna

Po wpisaniu w przeglądarce adresu *http://localhost/joomla/* wyświetli się strona główna tworzonej witryny zawierająca tytuł strony i formularz logowania. Są to elementy domyślne tworzone przez system, które zostaną rozbudowane w trakcie dalszych prac nad projektem.

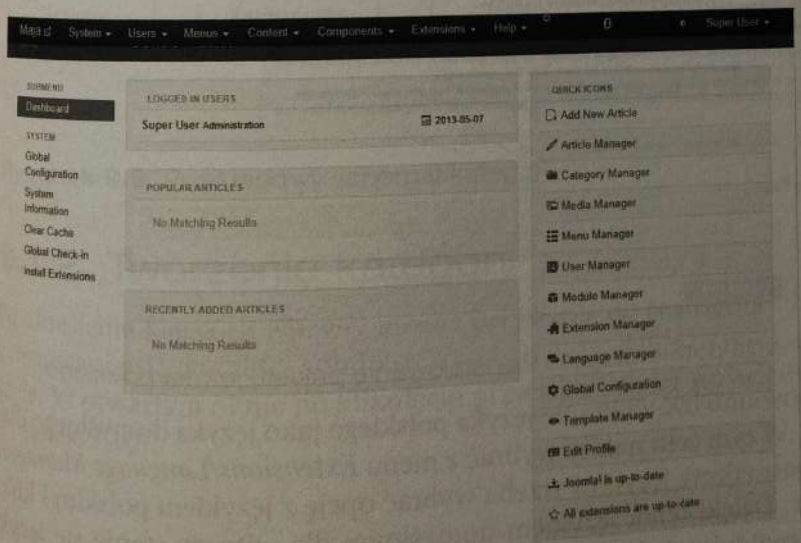
Część administracyjna (zaplecze)

Po wpisaniu w przeglądarce adresu *http://localhost/joomla/administrator/* wyświetli się ekran logowania (rysunek 9.4) do części administracyjnej portalu.



Rysunek 9.4. Okno logowania do panelu administracyjnego

W polu *User Name* należy wpisać zdefiniowaną w trakcie instalacji nazwę konta administratora, a w polu *Password* jego hasło. Po kliknięciu przycisku *Log in* wyświetli się część administracyjna witryny (rysunek 9.5).

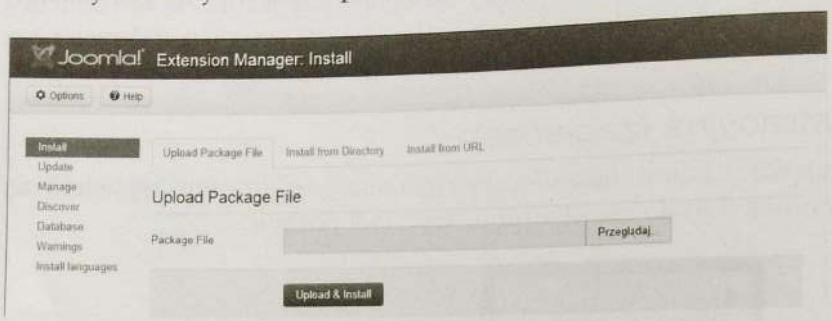


Rysunek 9.5. Część administracyjna portalu (Pulpit)

W górnej części pokazanego okna znajduje się pasek menu, w obszarze poniżej dostępne są przyciski (ikony) umożliwiające szybki dostęp do różnych funkcji.

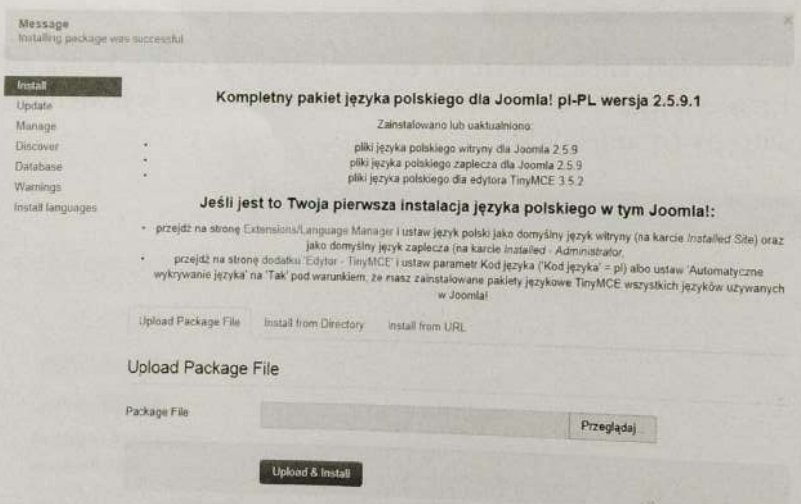
Przed rozpoczęciem pracy można zainstalować pakiet językowy zawierający spolszczoną wersję systemu Joomla!. Potrzebny plik można pobrać ze strony *http://joomla.pl/index.php/pobierz-joomla.html*.

Po pobraniu pliku i zalogowaniu się do panelu administracyjnego należy wybrać z menu opcję *Extensions/Extension Manager*. Pojawi się okno wyboru pliku (rysunek 9.6), w którym należy odnaleźć plik zawierający spolszczenie.



Rysunek 9.6. Pobranie pliku spolszczenia

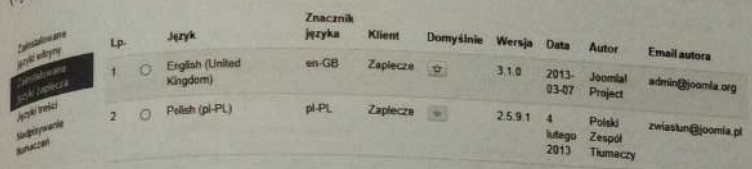
Po kliknięciu przycisku *Upload & Install* plik zostanie zainstalowany. W oknie przeglądarki wyświetli się informacja potwierdzająca zainstalowanie pakietu języka polskiego (rysunek 9.7).



Rysunek 9.7. Informacja potwierdzająca zainstalowanie pakietu języka polskiego

Kolejnym krokiem powinno być wybranie języka polskiego jako języka domyślnego dla strony internetowej. W tym celu należy wybrać z menu *Extensions/Language Manager*. W otwartym oknie menedżera języków trzeba wybrać opcję z językiem polskim i kliknąć przycisk *Default*. Dzięki temu językiem domyślnym dla witryny stanie się język polski. Teraz z listy znajdującej się w lewej części okna należy wybrać opcję *Installed*

Administrator i w taki sam sposób ustawić język domyślny dla części administracyjnej (rysunek 9.8).



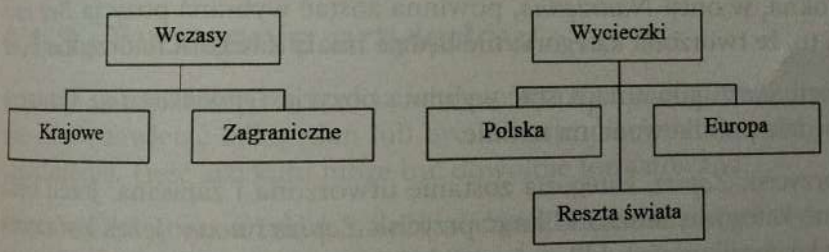
Lp.		Język	Znacznik języka	Klient	Domyślnie	Wersja	Data	Autor	Email autora
1	○	English (United Kingdom)	en-GB	Zaplecze	☆	3.1.0	2013-03-07	Joomla! Project	admin@joomla.org
2	○	Polish (pl-PL)	pl-PL	Zaplecze	★	2.5.9.1	4 lutego 2013	Polski Zespół Tłumaczy	zwiastun@joomla.pl

Rysunek 9.8. Język zaplecza administracyjnego

9.1.1. Tworzenie treści portalu

Treść jest najważniejszą częścią portalu. Powinna zostać zorganizowana tak, aby użytkownik witryny miał do niej łatwy dostęp. W Joomla! publikacje dostępne w witrynie internetowej są zorganizowane w postaci struktury złożonej z artykułów i kategorii. Artykułem jest każda publikacja, która powinna znaleźć się na stronie. Każdy artykuł musi zostać przypisany do jednej z kategorii. W kategorii można umieścić dowolną liczbę artykułów, a każda kategoria może zawierać inne kategorie. Struktura kategorii może być dowolnie rozbudowywana, w zależności od potrzeb tworzonej witryny. Po instalacji systemu Joomla! w witrynie pojawia się specjalna kategoria domyślna *Uncategorised*. W niej będą zamieszczane artykuły, które nie pasują do żadnej innej kategorii. Mogą to być publikacje przeznaczone na stronę tytułową lub inne pojedyncze teksty.

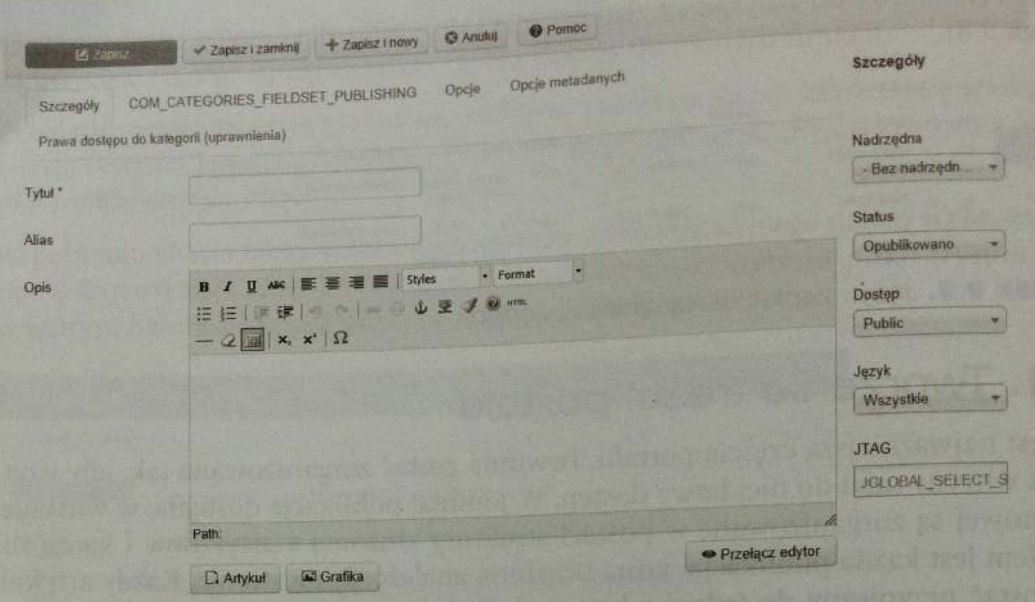
Jeżeli założymy, że tworzona witryna będzie dotyczyła biura podróży, to podział na kategorie mógłby mieć postać jak na rysunku 9.9.



Rysunek 9.9. Przykładowy schemat kategorii

9.1.2. Tworzenie kategorii

Po określeniu kategorii witryny można przystąpić do tworzenia struktury jej treści. Po zalogowaniu się do części administracyjnej należy wybrać z menu opcję *Artykuły/Kategorie*. W otwartym oknie po kliknięciu przycisku *Utwórz* wyświetli się *Edytor kategorii* (rysunek 9.10).



Rysunek 9.10. Blok edytowania kategorii

W polu *Tytuł* należy wpisać nazwę kategorii (np. *Wycieczki*). Można używać polski liter. Nazwa, która tutaj zostanie wpisana, będzie wyświetlana w serwisie.

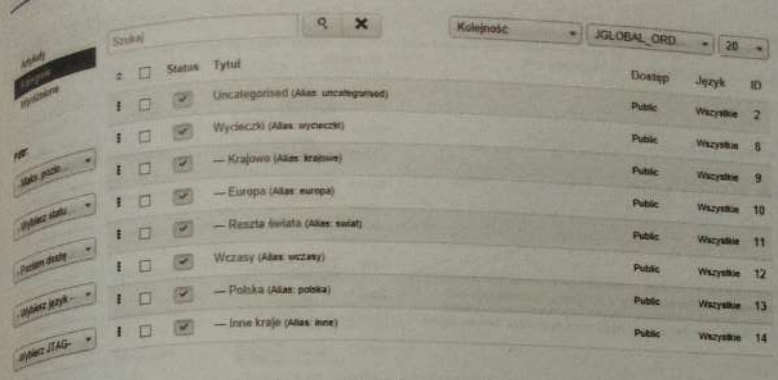
W polu *Alias* należy wpisać nazwę kategorii, która będzie wykorzystywana w adresa służących do nawigacji po witrynie. Nie należy używać polskich liter ani spacji. Ta n zwa będzie widoczna w polu adresu przeglądarki. Jeżeli pole pozostanie niewypełnio system sam utworzy wewnętrzny alias.

Z prawej strony okna, w opcji *Nadrzędna*, powinna zostać wybrana pozycja *Bez n rzędnej*. Oznacza to, że tworzona kategoria nie będzie miała kategorii nadrzędnej.

Natomiast dla opcji *Status* powinna zostać wybrana pozycja *Opublikowano*. Oznac to, że kategoria będzie publikowana na stronie.

Gdy klikniemy przycisk *Zapisz*, kategoria zostanie utworzona i zapisana. Jeżeli bę tworzone następne kategorie, można kliknąć przycisk *Zapisz i nowy*. Jeżeli kończy pracę z edytorem kategorii, można kliknąć przycisk *Zapisz i zamknij*.

W podobny sposób można utworzyć kolejne kategorie. Jeżeli tworzona będzie katego podrzędna, to po rozwinięciu listy w opcji *Nadrzędna* należy wybrać kategorię, do któ powinna zostać przypisana tworzona kategoria. Po utworzeniu wszystkich kateg należy kliknąć przycisk *Zapisz i zamknij*, co spowoduje zapisanie kategorii, opuszcze *Edytora kategorii* i przejście do *Menedżera kategorii* (rysunek 9.11).



Rysunek 9.11. Lista kategorii w Menedżerze kategorii

Jeżeli jakaś kategoria powinna zostać zmodyfikowana lub usunięta, to te czynności można wykonać w *Menedżerze kategorii*. Widoczna jest tu lista wszystkich utworzonych kategorii wraz z informacjami o nich. Jeżeli kategorii jest dużo, można w obszarze *Filtr* znajdującym się z lewej strony okna ograniczyć widoczność tylko do wybranych. Widoczne u góry przyciski pozwalają na wykonanie operacji na kategoriach: dodawanie nowych, usuwanie istniejących, ustawianie parametrów. Aby całkowicie usunąć kategorię z systemu, należy najpierw przenieść ją do kosza, a następnie opróżnić kosz.

Zadanie 9.1

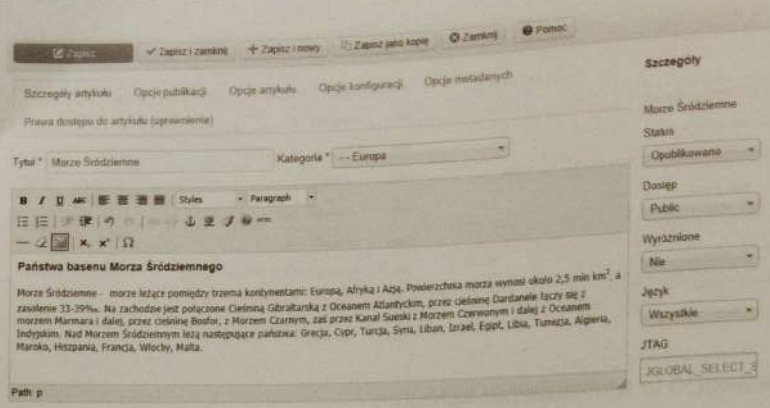
Zaprojektuj strukturę witryny, którą utworzysz za pomocą systemu Joomla!. Zebrane podczas projektowania witryny materiały podziel na kategorie. Utwórz kategorie zgodne z przygotowanym projektem.

9.1.3. Tworzenie artykułów

Artykuły są najważniejszą częścią witryny. Jest to dowolna treść umieszczana na stronie. Może zawierać kilka zdań lub być kilkustronicowym tekstem opisującym jakieś zagadnienie. Treść artykułu może być dowolnie formatowana.

Aby utworzyć nowy artykuł, należy w części administracyjnej wybrać z menu opcję *Artykuły/Artykuły*. Można też na *Pulpicie* wybrać z listy opcję *Artykuły*. W otwartym oknie wyświetlą się wszystkie utworzone artykuły. Początkowo lista jest pusta i aby dodać nowy artykuł, trzeba kliknąć przycisk *Utwórz*. Wyświetli się *Edytor artykułu*.

W polu *Tytuł* należy wpisać tytuł artykułu. W polu *Kategoria* trzeba wybrać z listy kategorię, do której będzie należał artykuł. Aby artykuł był widoczny dla użytkowników witryny, opcję *Stan* należy ustawić na wartość *Opublikowano*. Aby artykuł nie został wyświetlony na stronie głównej witryny, opcja *Wyróżnienie* powinna być ustawiona na wartość *Nie*. Po tych ustawieniach trzeba wpisać treść artykułu. W polu wpisywania treści dostępne są opcje formatowania tekstu, których można używać tak jak w innych edytorach tekstu (rysunek 9.12). Aby zapisać artykuł i przejść do *Menedżera artykułów*, należy kliknąć przycisk *Zapisz i zamknij*. Jeżeli będą tworzone kolejne artykuły, można wybrać przycisk *Zapisz i nowy*.



Rysunek 9.12. Edytowanie artykułu

Domyślnie udostępniany w systemie Joomla! *Edytor artykułów* zawiera podstawowe funkcje edycji tekstu. Jeżeli potrzebne są bardziej zaawansowane funkcje formatowania, można przełączyć się do trybu zaawansowanego. W tym celu z menu należy wybrać opcję *Dodatki* i w *Menedżerze dodatków* wybrać z listy edytor *TinyMCE*. Jest to domyślny edytor systemu Joomla!. W oknie ustawień edytora po wybraniu zakładki *Opcje podstawowe* należy wybrać z listy *Tryb pracy* pozycję *Rozszerzony* i kliknąć przycisk *Zapisz i zamknij*.

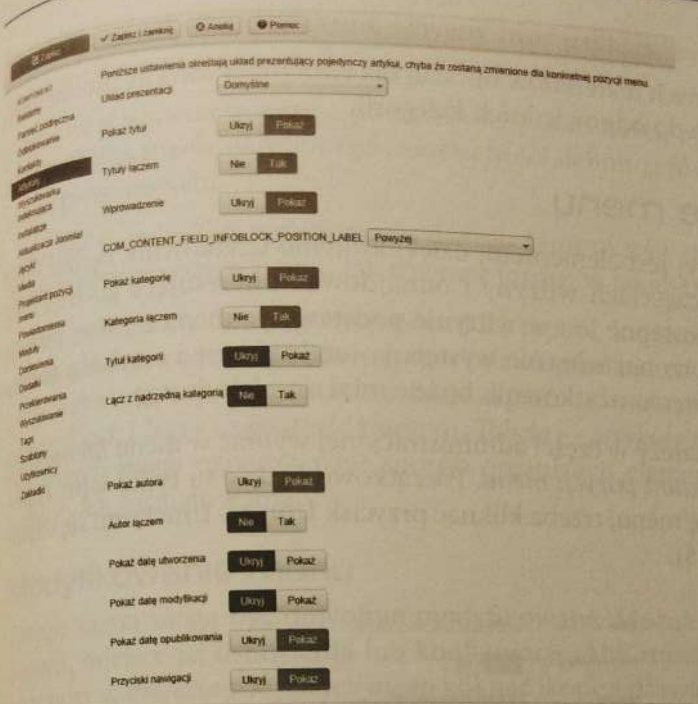
Artykuł na stronie głównej

Jeżeli tworzony artykuł będzie wyświetlany na stronie głównej witryny, wystarczy podczas jego edycji w opcji *Wyróżnione* wybrać wartość *Tak* oraz dla opcji *Kategoria* pozycję *Uncategorised*. Spowoduje to umieszczenie artykułu w grupie specjalnej, która domyślnie jest wyświetlana na stronie głównej.

Atrybuty artykułów

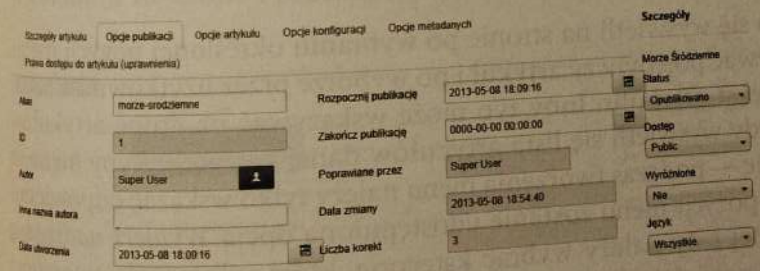
Podczas tworzenia witryny internetowej można określić, jakie atrybuty artykułów będą pokazywane przy tekście, a także zmodyfikować atrybuty grupy artykułów lub pojedynczego artykułu. Konfiguracja globalna pozwala ustawić cechy, które będą takie same dla wszystkich artykułów. Później cechy pojedynczych artykułów mogą być modyfikowane według potrzeb.

Globalna konfiguracja jest dostępna w *Menedżerze artykułów* po wybraniu przycisku *Opcje*. Wyświetlony ekran konfiguracji składa się z kilku części (rysunek 9.13), w których pokazywany jest aktualny stan atrybutów dla wszystkich artykułów. Modyfikacja dostępnych parametrów będzie dotyczyła wszystkich artykułów.



Rysunek 9.13. Globalna konfiguracja artykułów

Atrybuty pojedynczego artykułu można modyfikować po przejściu do trybu jego edycji. Wtedy w górnej części dostępne są zakładki *Opcje publikacji* oraz *Opcje artykułu*. Zakładka *Opcje publikacji* zawiera kilka przydatnych opcji. W polu *Rozpocznij publikację* można określić, kiedy ma się rozpocząć publikacja artykułu (rysunek 9.14). Domyślnie jest to data umieszczenia artykułu w systemie. W polu *Zakończ publikację* można określić, kiedy ma się zakończyć publikacja artykułu. Domyślnie artykuł nie ma ustawionych ograniczeń na czas publikacji. *Liczba korekt* określa liczbę zapisów artykułu w systemie. Pole *Poprawiane przez* pokazuje nazwę użytkownika, który zmodyfikował artykuł.



Rysunek 9.14. Opcje publikacji artykułu

Zakładka *Opcje artykułu* zawiera opcje takie same, jak opcje konfiguracji globalnej. Można zmienić ich wartość dla wybranego artykułu.

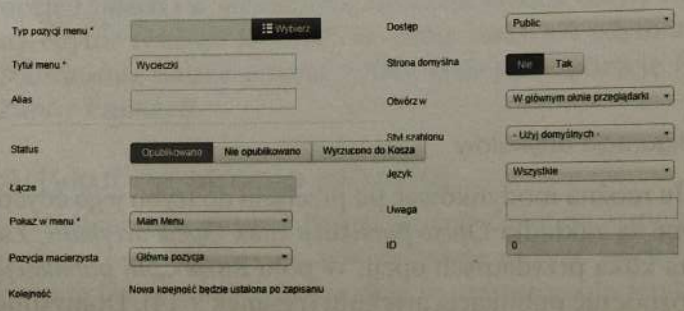
Zadanie 9.2

Na podstawie zgromadzonych materiałów opracuj dla tworzonej witryny internetowej artykuły i poprzydzielaj je do odpowiednich kategorii.

9.1.4. Tworzenie menu

Menu witryny internetowej jest elementem, dzięki któremu użytkownik będzie mógł poruszać się po różnych częściach witryny i odnajdować interesujące go treści. Po zainstalowaniu systemu dostępne jest w witrynie podstawowe menu z jedną pozycją — *Home*. Ponieważ w tworzonej witrynie występują już kategorie i artykuły, można utworzyć menu, dzięki któremu użytkownik będzie miał do nich dostęp.

Aby modyfikować menu, należy w części administracyjnej wybrać w menu *Menu/Main Menu*. Uruchomi się *Projektant pozycji menu*. Początkowo będzie tu tylko wpis *Home*. Aby utworzyć nową pozycję menu, trzeba kliknąć przycisk *Utwórz*. Uruchomi się edytor pozycji menu (rysunek 9.15).



Rysunek 9.15. Edytor menu witryny internetowej

W polu *Tytuł menu* należy wpisać nazwę wybranej pozycji menu (np. *Wycieczki*). Dla pola *Typ pozycji menu* należy kliknąć przycisk *Wybierz*. Wyświetli się lista pogrupowanych tematycznie typów dostępnych w systemie.

Typ pozycji określa, co się wyświetli na stronie po wybraniu określonej pozycji menu. Jeden typ może wskazywać pojedynczy artykuł i po wyborze przez użytkownika takiej pozycji wyświetli się tekst artykułu. Inny typ może wskazywać na grupę artykułów w danej kategorii i wtedy wyświetli się lista artykułów danej kategorii. Typy menu są zdefiniowane w systemie — podczas tworzenia menu należy tylko wskazać odpowiedni typ. Po wybraniu typu pozycji menu zostanie udostępniona opcja *Wybierz nadrzędną kategorię*. Z wyświetlonej listy należy wybrać kategorię, która odpowiada tworzonej pozycji menu (np. *Wycieczki*). W celu utworzenia pozycji menu trzeba kliknąć przycisk *Zapisz i zamknij*. W podobny sposób można tworzyć kolejne pozycje menu.

Typy pozycji menu

W systemie Joomla! każda pozycja menu jest połączona ze specjalnym komponentem, który odpowiada za generowanie treści. Dostępne komponenty pozwalają tworzyć

dowolnie rozbudowaną strukturę witryny (np. artykuły, kategorie artykułów, formularze kontaktowe).

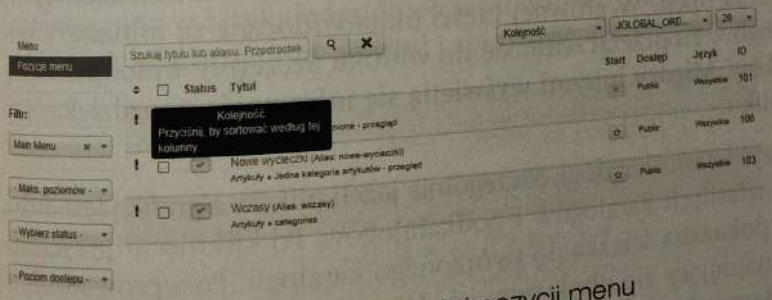
Każda pozycja menu wskazuje na określony zakres treści, np. artykuł, kategorię, kilka kategorii. Wskazanie określonego zakresu treści definiuje również formę, w jakiej treść będzie prezentowana.

Wybranie w polu *Typ pozycji menu* danego typu określi zakres treści, które pojawią się po wyborze pozycji menu, jak również formę, w jakiej treść będzie prezentowana.

W polu tym mamy do dyspozycji ponad 20 typów, które pomogą w sprawnym zarządzaniu strukturą witryny. Nazwy typów odzwierciedlają zakres i formę treści. Pierwszy człon wskazuje, w jakiej postaci zostanie zaprezentowana treść, drugi określa jej rodzaj, na przykład: *Lista — zawartość kategorii*, *Tabela — artykuły z kategorii*, *Łącze — kontakt*. *Typ pozycji menu* oznacza więc sposób organizacji elementów struktury treści oraz szablon ich formatowania.

Modyfikowanie menu

Pozycje menu mogą być dowolnie modyfikowane. Można zmieniać nazwę wybranej pozycji, miejsce jej ustawienia lub konfigurację. Aby zmienić nazwę lub konfigurację, wystarczy w *Projektancie pozycji menu* kliknąć ikonę z nazwą wybranej pozycji. Otworzy się okno edytowania, w którym można dowolnie zmieniać wprowadzone ustawienia. Aby usunąć pozycję menu, należy w *Projektancie pozycji menu* zaznaczyć pozycję i kliknąć ikonę *Kosz*. Pozycje umieszczone w *Koszu* mogą być definitywnie usunięte lub przywrócone do menu. Przed wykonaniem tych czynności trzeba wyświetlić zawartość *Kosza*. W oknie *Projektanta pozycji menu*, na liście z lewej strony okna, należy wybrać *Wybierz status*, a następnie *Wyrzucono do kosza*. Wyświetli się lista usuniętych do *Kosza* pozycji menu. Po zaznaczeniu wybranej pozycji trzeba wybrać w górnej części okna ikonę *Opróżnij*, aby definitywnie ją usunąć, lub ikonę *Opublikuj*, aby przywrócić pozycję do menu. Powrót do *Projektanta pozycji menu* nastąpi po wybraniu opcji *Wybierz status* z listy z lewej strony okna. Kolejność menu można zmienić, chwytając ikonę pozycji w kolumnie *Kolejność* i przeciągając ją w odpowiednie miejsce (rysunek 9.16).



Rysunek 9.16. Ustalanie kolejności pozycji menu

Rozbudowa menu

Tworzone menu może zawierać pozycje podrzędne, które po podziale artykułów na podkategorie umożliwią dostęp z poziomu menu do wybranej grupy artykułów.

Aby utworzyć w menu pozycję podrzędną, należy przejść do tworzenia menu (*Menu/Main Menu/Utwórz pozycję menu*), wpisać tytuł, jako *Typ pozycji menu* wybrać z listy *Artykuły*, a następnie *Pojedynczy artykuł*. Później w opcji *Pozycja macierzysta* należy z listy pozycji menu wybrać pozycję, która będzie nadrzędna w stosunku do tworzonej. W kolejnym kroku trzeba w opcji *Wybierz kategorię* wybrać z listy kategorię odpowiadającą tworzonej pozycji menu. Aby zapisać wprowadzone ustawienia, należy kliknąć przycisk *Zapisz i zamknij*. Na liście pozycji menu nowa pozycja powinna zostać pokazana jako podrzędna w stosunku do wybranej pozycji głównej. W podobny sposób można tworzyć kolejne pozycje menu podrzędnego.

Po wprowadzeniu zmian można przetestować menu, przechodząc do strony głównej witryny. Ponieważ pozycje podrzędne menu domyślnie są ukryte, wyświetlona witryna wygląda tak, jakby nic się nie zmieniło. Dopiero po kliknięciu pozycji nadrzędnej wyświetlą się wszystkie pozycje podrzędne. Modyfikacja pozycji menu podrzędnych odbywa się podobnie jak modyfikacja pozycji menu głównego.

Zadanie 9.3

Dla swojej witryny internetowej utwórz menu. Przydziel poszczególne pozycje menu do odpowiednich kategorii. Dla każdej pozycji menu określ odpowiedni typ tej pozycji.

9.1.5. Grafika

Korzystając z systemu Joomla!, można do witryny dodawać dowolne grafiki, wcześniej umieszczone na serwerze. Jeżeli konto WWW jest dostępne lokalnie, wystarczy potrzebne grafiki umieścić w przeznaczonym do tego folderze.

Plikami graficznymi można zarządzać za pomocą *Menedżera mediów*. Pozwala on na tworzenie struktury katalogów, wczytywanie plików na serwer oraz wybór odpowiedniego pliku. Uruchomienie *Menedżera mediów* nastąpi po wybraniu z menu *Artykuły/Media*. W otwartym oknie z lewej strony widoczna jest struktura katalogów z grafiką. U góry dostępne są dwa przyciski: *Miniatury* i *Szczegóły*, które pozwalają na przełączanie się między tymi widokami. Powyżej znajdują się ikony do tworzenia katalogów, usuwania i konfigurowania opcji. W głównej części okna widoczne są miniatury plików graficznych i katalogów. Po przełączeniu się do widoku *Szczegóły* widoczne będą szczegóły dotyczące plików. Między innymi wyświetlą się informacje o rozdzielczości obrazu oraz wielkości pliku.

W systemie można tworzyć własne katalogi. Szczególnie jeżeli plików z grafiką jest dużo, zalecane jest tworzenie własnych katalogów i podkatalogów. Po kliknięciu przycisku *Utwórz katalog* zostanie pokazana ścieżka do tworzonego katalogu. Po wprowadzeniu nazwy należy kliknąć znajdujący się obok przycisk *Utwórz katalog*. Plik lub katalog można usunąć, klikając ikonę z krzyżykiem znajdującą się w kolumnie *Usuwanie* (widok *Szczegóły*) lub przy jego miniaturze (widok *Miniatury*). Gdy grafiki zostaną umieszczone na serwerze lub gdy są dostępne lokalnie, można umieścić je na wybranych stronach witryny.

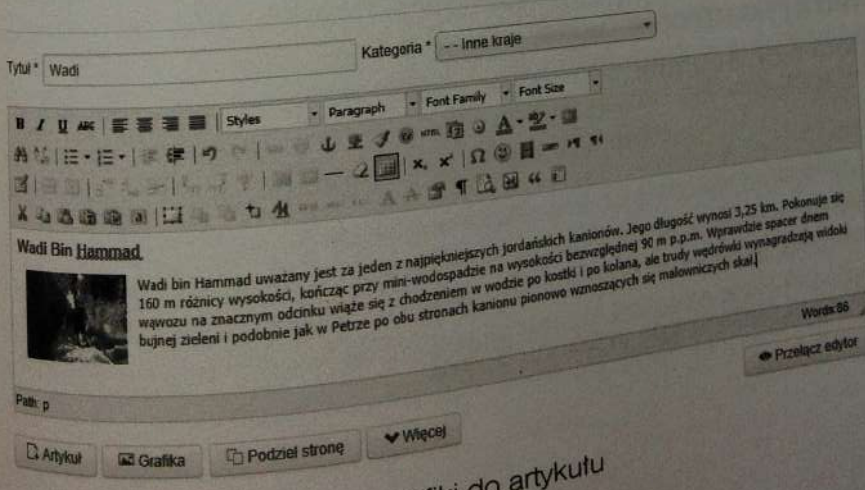
Dodawanie grafiki do kategorii

Utworzona wcześniej kategoria może zostać zilustrowana grafiką. W tym celu należy przejść do kategorii (w menu wybrać *Artykuły/Kategorie*) i po wybraniu kategorii przejść do jej edytowania. W oknie edytowania kategorii trzeba wybrać zakładkę *Opcje*. Następnie na liście opcji podstawowych trzeba kliknąć przycisk *Wybierz* przy pozycji *Ilustracja opisu*. W otwartym oknie należy wybrać z odpowiedniego katalogu plik z grafiką i kliknąć przycisk *Wstaw*. Aby zatwierdzić wprowadzone zmiany, w oknie edycji kategorii trzeba kliknąć przycisk *Zapisz i zamknij*. Żeby grafika pojawiła się na stronie, przy wybranej pozycji menu należy ustawić odpowiednio opcje tej pozycji.

W oknie *Projektanta pozycji menu* (*Menu/Main Menu*) wybieramy odpowiednią pozycję i edytujemy ją. W otwartym oknie edytowania pozycji wybieramy zakładkę *Opcje rozszerzone* i w sekcji *Opcje listy jednej kategorii* zmieniamy pozycję *Ilustracja kategorii* na *Pokaż*. Następnie klikamy przycisk *Zapisz* i w kolejnym oknie *Zapisz i zamknij*. Po wyświetleniu strony głównej i wybraniu pozycji z dołączoną grafiką pod tytułem kategorii powinna wyświetlić się grafika.

Dodawanie grafiki do artykułu

Grafika wstawiana na stronę internetową najczęściej jest ilustracją artykułu. Przed jej wstawieniem należy utworzyć artykuł i przejść do jego edycji. Jeżeli grafika ma zostać wstawiona w określonym miejscu artykułu, trzeba ustawić w tym miejscu kursor myszy i kliknąć przycisk *Grafika*. W otwartym oknie należy wybrać katalog i plik do wstawienia. W dolnej części tego okna znajdują się pola, które pozwalają na edycję parametrów wstawianego obrazu. Można ustawić sposób wyrównania obrazu, wstawić jego tytuł i opis. Po kliknięciu przycisku *Wstaw* obraz znajdzie się w tekście. Wstawiony obraz można edytować. Aby zmienić jego parametry, należy go zaznaczyć i wybrać ikonę (*Insert/Edit Image*). Otworzy się okno, w którym można edytować obraz. W zakładce *Appearance* można ustalić położenie obrazka. Pole *Alignment* służy do określenia pozycji obrazka względem tekstu. Pole *Dimension* określa aktualne jego rozmiary, a pola *Vertical space* i *Horizontal space* określają odstęp między obrazem i tekstem w pionie i poziomie. Po wprowadzeniu zmian należy kliknąć przycisk *Update*. Obrazek zostanie umieszczony w artykule zgodnie z wprowadzonymi zmianami (rysunek 9.17). Po tych zmianach należy zapisać artykuł i sprawdzić, jak wygląda na stronie internetowej.



Rysunek 9.17. Wstawianie grafiki do artykułu

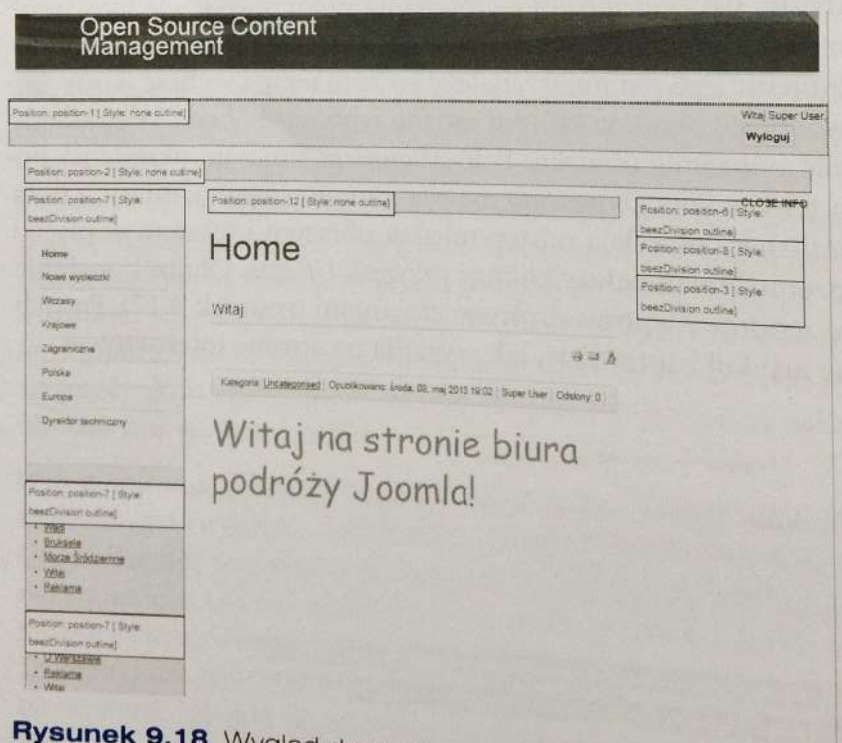
Zadanie 9.4

W projektowanej przez siebie witrynie do wybranych kategorii lub artykułów doda przygotowane wcześniej grafiki oraz przygotuj stronę zawierającą podstawowe info macje mówiące o właścicielu witryny. Utwórz artykuł *O mnie*, dodaj do niego grafik Utwórz nową pozycję w menu i przypisz do niej utworzony artykuł.

9.1.6. Moduły

W tworzonej witrynie internetowej można wyodrębnić pewne części odpowiedzialn za wykonywanie określonych operacji. Każdą wyodrębnioną i niezależnie wyświetlan część witryny można traktować jako moduł. Modułem jest menu, lista najnowszyc wiadomości czy liczba osób odwiedzających stronę. Wszystko, co jest wyświetlan w witrynie, jest modułem. Tworząc witrynę internetową, należy określić, jakie moduł powinny znaleźć się na niej, np. w przypadku biura podróży mogą to być: lista atra cyjnych wycieczek, lista nowych wycieczek, lista gości obecnych na stronie, oferta la minute. Moduł wyświetli te dane oraz umożliwi ich konfigurację.

Strona internetowa jest podzielona na kilka części. Tworząc nowy moduł, należ określić, w jakiej części strony powinien on zostać umieszczony. Jeżeli w jednej część zostanie umieszczonych kilka modułów, należy zdecydować o kolejności ich wyświe lania, podając wartość porządkową modułu. Po rozpoczęciu pracy w systemie Joomla domyślnie stosowany jest szablon Beez3, w którym obowiązuje podział strony pokazan na rysunku 9.18.



Rysunek 9.18. Wygląd domyślnego szablonu Beez3

Modyfikowanie modułu menu

Znajdujące się w witrynie menu jest modułem. Może ono być modyfikowane, ale można też utworzyć nowe menu. Aby można było tworzyć lub modyfikować moduł, należy przejść do trybu jego edycji. W tym celu trzeba wybrać w menu części administracyjnej *Rozszerzenia/Moduły*. Wyświetli się okno z dostępnymi modułami. Początkowo są to standardowe moduły dostępne w systemie.

Przykład 9.1

Dostępny na liście moduł *Main Menu* jest modułem związanym z wyświetlanym w witrynie menu. Zostanie zmieniona jego nazwa. W tym celu wybieramy i klikamy moduł widoczny na liście. W oknie edycji modułu wprowadzamy w polu *Tytuł* nową nazwę menu, np. *Menu główne*. Pole *Pokaż tytuł* decyduje o tym, czy tytuł ten będzie wyświetlany w witrynie. Zostawiamy ustawienia domyślne. Pole *Pozycja* decyduje, gdzie będzie umieszczone menu. Również zostawiamy ustawienia domyślne. Po ustawieniu opcji modułu klikamy przycisk *Zapisz i zamknij*. Teraz przechodzimy na stronę główną witryny (*http://localhost/joomla/*) i sprawdzamy, jak wygląda menu po wprowadzonych zmianach.

Dodawanie modułu menu

Przed rozpoczęciem tworzenia nowego modułu należy pamiętać, że moduł jest tylko elementem wyświetlającym dane. Dlatego przed utworzeniem modułu należy przygotować treść, która zostanie w nim pokazana.

Przykład 9.2

Na tworzonej witrynie zostanie dodane nowe menu. Najpierw należy utworzyć menu, a następnie moduł, w którym będzie ono umieszczone.

Uruchamiamy *Projektanta menu* (z menu wybieramy *Menu/Projektant menu/Utwórz menu*) i w polu *Tytuł menu* wpisujemy odpowiedni tytuł. W polu *Typ menu* wpisujemy dowolną wymyśloną przez nas nazwę typu menu. Będzie ona używana tylko wewnętrznie przez system Joomla! jako identyfikator. Nie można używać spacji i polskich liter. W polu *Opis* można wprowadzić dowolną treść opisującą tworzone menu. W celu zapisania menu klikamy przycisk *Zapisz i zamknij*. Tytuł menu pojawi się na liście dostępnych menu. Po kliknięciu tego tytułu wyświetli się znane już okno *Projektowania pozycji menu*. Aby utworzyć kolejne pozycje menu, klikamy przycisk *Utwórz* i w polu *Tytuł menu* wpisujemy kolejną pozycję menu. Jako typ pozycji menu wybieramy np. *Artykuły/Jedna kategoria artykułów — lista*. Następnie w polu *Wybierz kategorię* wybieramy z listy kategorię przypisaną do pozycji menu i klikamy przycisk *Zapisz i zamknij*. W ten sam sposób tworzymy kolejne pozycje menu.

Ponieważ zawartość menu wyświetlana jest za pomocą modułów, utworzymy nowy moduł. Aby dodać nowy moduł, wybieramy z menu *Rozszerzenia/Moduły* i klikamy przycisk *Utwórz*. Wyświetli się okno wyboru typu modułu, w którym można wybrać typ tworzonego modułu. Ponieważ moduł będzie służył do wyświetlania menu, wybieramy typ *Menu*. W kolejnym oknie wyświetlą się opcje konfiguracji modułu. Wpisujemy

tytuł modułu i decydujemy, czy ma być wyświetlany (pole *Pokaż tytuł*). W polu *Pozycja* określamy, w którym miejscu na stronie będzie wyświetlany moduł. Wybieramy np. *position-7*, co oznacza, że moduł wyświetli się w górnej części lewej kolumny. Przechodzimy do zakładki *Opcje* i w polu *Wybierz menu* wybieramy menu, które będzie wyświetlane w tworzonym module. Przechodzimy do zakładki *Przypisz do pozycji menu* i określamy, na jakich stronach menu będzie wyświetlane. Domyślnie wybrana jest pozycja *Na wszystkich stronach*, co oznacza, że menu będzie widoczne niezależnie od tego, która strona będzie wyświetlana. Po wybraniu pozycji *Tylko na zaznaczonych stronach* wyświetli się lista wszystkich pozycji menu. Zostawiamy zaznaczone tylko te, na których dodatkowe menu powinno być wyświetlane. Po ustawieniu wszystkich opcji klikamy przycisk *Zapisz i zamknij*. Aby ocenić, czy nowe menu wyświetla się prawidłowo, można przejść na stronę główną witryny i sprawdzić, jak wygląda i funkcjonuje nowo utworzone menu.

Włączanie i wyłączanie modułu

Istniejący moduł może w zależności od potrzeb zostać włączony lub wyłączony. Można w ten sposób zablokować wyświetlanie modułu na wszystkich lub niektórych stronach. Moduł logowania standardowo jest widoczny na wszystkich stronach. Jeżeli w tworzonej witrynie nie będzie potrzebny, można go wyłączyć.

Przykład 9.3

Wyłączymy moduł logowania. W tym celu uruchamiamy *Menedżera modułów* (w menu wybieramy *Rozszerzenia/Moduły*) i klikamy ikonę *Status* ✔ z lewej strony modułu logowania *Login Form*. Ikona zmieni się na ⊘. Po przejściu na stronę główną zobaczymy, że formularz logowania zniknął. Aby przywrócić wyświetlanie modułu, należy ponownie wejść do *Menedżera modułów*, wybrać określony moduł i kliknąć ikonę *Status*. Po zmianie wyglądu ikony moduł będzie dostępny na stronie.

Kolejność modułów

Kolejność wyświetlania modułów jest dowolna. W zależności od potrzeb można ją zmieniać. Po uruchomieniu *Menedżera modułów* wyświetli się lista utworzonych modułów. Wystarczy wybrany moduł w kolumnie *Kolejność* chwycić myszą i przesunąć na odpowiednią pozycję.

9.1.7. Moduły systemu Joomla!

System Joomla! zawiera kilka standardowych modułów realizujących podstawowe funkcje związane z obsługą strony internetowej. Są to: lista najpopularniejszych artykułów, liczba osób korzystających z witryny, lista nowości, ścieżka powrotu.

Najczęściej czytane

Za pomocą modułu *Najczęściej czytane* można wyświetlić na stronie listę najczęściej czytanych artykułów.

Przykład 9.4

Aby dołączyć moduł *Najczęściej czytane*, po uruchomieniu *Menedżera modułów* klikamy przycisk *Utwórz*. Z wyświetlonej listy typów modułu wybieramy pozycję *Popularne* i w otwartym oknie edycji modułu wpisujemy w polu *Tytuł* tytuł, pod jakim będzie wyświetlany moduł, np. *Najpopularniejsze artykuły*. Opcję *Pokaż tytuł* zostawiamy ustawioną na *Pokaż*, a opcję *Status* na *Opublikowano*. W polu *Pozycja* wybieramy pozycję modułu na stronie, np. *position-6*, aby moduł został umieszczony w prawej części witryny. Przechodzimy do zakładki *Przypisz do pozycji menu* i na liście *Zasada przypisania* określamy, kiedy moduł będzie wyświetlany, np. *Tylko na zaznaczonych stronach*. Przechodzimy do zakładki *Opcje* i w polu *Liczba artykułów* wpisujemy liczbę artykułów wyświetlanych w module. W polu *Kategorie* możemy wybrać kategorie, z których będą wybierane artykuły. Aby zapisać wprowadzone ustawienia, klikamy przycisk *Zapisz i zamknij*. Sprawdzamy, czy nowy moduł jest wyświetlany. W tym celu przechodzimy na stronę główną witryny i testujemy jej działanie.

Nowości

Podczas tworzenia witryny istnieje również możliwość umieszczenia w niej informacji na temat nowo dodanych i ostatnio zmodyfikowanych artykułów.

Przykład 9.5

W celu dodania modułu z nowościami tworzymy nowy moduł i na liście typów modułu wybieramy pozycję *Nowości*. W polu *Tytuł* wpisujemy tytuł, pod jakim będzie wyświetlany moduł. Ustalamy w polu *Pozycja* pozycję modułu na stronie. Przechodzimy do zakładki *Opcje* i w polach *Kategorie* i *Liczba artykułów* ustawiamy odpowiednie wartości. W polu *Kolejność* możemy ustalić, czy najpierw widoczne będą artykuły ostatnio dodane, ostatnio zmodyfikowane, czy ostatnio redagowane. Zapisujemy dane, klikając przycisk *Zapisz i zamknij*, i sprawdzamy, czy nowy moduł jest wyświetlany na stronach witryny.

Zadanie 9.5

Stosując metodę pokazaną w poprzednich przykładach, dodaj kolejny moduł do witryny internetowej. Będzie to moduł pokazujący liczbę odwiedzających, wykorzystujący typ modułu *Gościmy*.

9.1.8. Komponenty

Komponenty to narzędzia, które umożliwiają rozbudowę funkcjonalności witryny. Ich zadaniem jest zbieranie, przetwarzanie, przechowywanie i generowanie danych. Po odpowiednim przygotowaniu danych są one za pomocą modułu wyświetlane na stronie. W systemie Joomla! są dostępne komponenty realizujące tylko podstawowe zadania, np. zarządzanie listą kontaktów, reklam lub odnośników. Inne można pobrać z internetu jako rozszerzenia systemu.

Lista kontaktów

Często na stronie umieszczana jest lista kontaktów, szczególnie w witrynach różnych firm. System Joomla! pozwala nie tylko tworzyć takie listy, ale także dzielić je na kategorie, kontrolować, które kontakty powinny być wyświetlane oraz jakie dane dla każdego kontaktu powinny zostać wyświetlone.

Przykład 9.6

Aby utworzyć kategorię kontaktów, wybieramy z menu *Komponenty/Kontakty/Kategorie kontaktów*. Otworzy się okno zarządzania kategoriami kontaktów. Standardowo istnieje jedna kategoria, *Uncategorised*. W niej powinny być umieszczane kontakty, które nie pasują do żadnej innej kategorii. Dodajemy nową kategorię, klikając przycisk *Utwórz*. W oknie, które zostanie otwarte, w polu *Tytuł* wpisujemy tytuł tworzonej kategorii, np. *Administrator*. Na liście z prawej strony okna, w polu *Nadrzędna*, zostawiamy pozycję *Bez nadrzędnej*, co oznacza, że nad tworzoną kategorią nie będzie innych. W polu *Stan* zostawiamy pozycję *Opublikowano*. Wprowadzone dane zapisujemy, klikając przycisk *Zapisz i zamknij*. Utworzona kategoria powinna pojawić się na liście kategorii. W podobny sposób postępujemy, tworząc kolejne kategorie.

Kolejnym etapem będzie utworzenie kontaktu należącego do zdefiniowanej kategorii. W tym celu wybieramy z menu *Komponenty/Kontakty/Kontakty*. Wyświetli się okno zarządzania kontaktami (wizytówkami). Tworzymy nowy kontakt, klikając przycisk *Utwórz*. W otwartym oknie wpisujemy nazwę kontaktu (imię i nazwisko). W polu *Kategoria* wybieramy kategorię, do której zostanie przypisany kontakt. W polu *Połącz z użytkownikiem* wybierany jest użytkownik, dla którego tworzony jest ten kontakt. Można zostawić to pole niewypełnione. Przechodzimy do zakładki *Opcje wizytówki* i wypełniamy dostępne pola. W polu *Wybierz obraz* możemy dodać grafikę, np. zdjęcie, które zostanie przypisane do kontaktu. Przechodzimy do zakładki *Opcje wyglądu* i wybierając *Pokaż* lub *Ukryj*, decydujemy, które dane zostaną wyświetlone w witrynie. Aby zapisać wprowadzone dane, klikamy przycisk *Zapisz i zamknij*. W podobny sposób dodajemy kolejne kontakty.

Następnym krokiem będzie powiązanie utworzonego komponentu z menu. Spowoduje to po wybraniu odpowiedniej pozycji menu wyświetlenie w witrynie listy kontaktów. Przechodzimy do tworzenia nowej pozycji menu (z menu wybieramy *Menu/Main Menu/Utwórz pozycję menu*). W otwartym oknie w polu *Typ pozycji menu* wybieramy *Kontakty/Pojedynczy kontakt* (jeżeli na liście będzie tylko jeden kontakt). W tytule menu wpisujemy nazwę menu, np. *Administrator*. W polu *Wybierz kontakt* wybieramy kontakt przypisany do pozycji menu i zapisujemy dane, klikając przycisk *Zapisz i zamknij*. Wchodzimy na stronę główną witryny i sprawdzamy, czy poprzez menu dostępny jest utworzony kontakt.

9.1.9. Szablony

Oprócz tworzenia różnych treści i rozmieszczania ich na stronie równie istotny jest sposób ich prezentacji. W systemie Joomla! treść oraz sposób jej prezentacji są od siebie

oddzielone. Pozwala to na prezentowanie tej samej treści na wiele różnych sposobów. Za wygląd witryny odpowiadają szablony, co powoduje, że można zmodyfikować jej wygląd, zmieniając wykorzystywany szablon. W systemie Joomla! dostępnych jest kilka szablonów, które pozwalają zmienić wygląd witryny, ale w internecie jest bardzo dużo innych gotowych szablonów, które można zastosować. Część z nich jest płatna, a inne są udostępniane za darmo. Ich instalacja przebiega podobnie jak instalacja pakietu językowego.

Operacje na szablonach są wykonywane za pomocą *Menedżera zarządzania szablonami*. Jest on dostępny po wybraniu z menu *Rozszerzenia/Szablony*. W oknie *Szablony* jest wyświetlana lista dostępnych szablonów. Szablony dla strony internetowej w kolumnie *Klient* mają wpisane *Witryna*, a dla zaplecza administracyjnego mają wpis *Zaplecze*. Gwiazdka umieszczona w kolumnie *Domyślny* określa, który z szablonów jest domyślny. Szablonem domyślnym jest szablon, za pomocą którego wyświetlana będzie witryna internetowa. W wersji Joomla! 3.1.1 jest to szablon Beez3. Po kliknięciu tego szablonu otworzy się okno jego edycji. Po wybraniu zakładki *Opcje* można zmienić podstawowe parametry szablonu. Pola *Wrapper Small* i *Wrapper Large* pozwalają na zmianę szerokości witryny. W polu *Logo* można umieścić ścieżkę dostępu do pliku graficznego zawierającego logo witryny. W polu *Site Title* należy wpisać tekst, który będzie wyświetlany, gdy zostanie wyłączone graficzne logo. W polu *Position of Navigation* można określić, w jaki sposób będzie prezentowane menu nawigacyjne (przed treścią czy za nią). W polu *Template colour* można ustawić kolorystykę witryny. Po wprowadzeniu zmian można zapisać szablon, klikając przycisk *Zapisz i zamknij*, lub utworzyć kopię szablonu, klikając *Zapisz jako kopię*.

Domyślny szablon można zmienić w oknie *Szablony*. Wystarczy wybrać odpowiedni szablon i kliknąć gwiazdkę znajdującą się w kolumnie *Domyślny*.

Szablony zewnętrzne

Jeżeli zamierzamy skorzystać z szablonu zewnętrznego, po znalezieniu w internecie odpowiedniego pliku należy pobrać go na swój komputer, a następnie zainstalować w systemie Joomla!. Instalacja przebiega tak samo jak instalacja pliku zawierającego spolszczenie.

Zadanie 9.6

Sprawdź, jak wyglądają standardowe szablony systemu Joomla!. Poszukaj szablonów zewnętrznych i spróbuj dołączyć je do swojej witryny. Wybierz szablon, który najlepiej pasuje do treści zamieszczonych w Twojej witrynie, i zastosuj go.

PYTANIA KONTROLNE

1. Do czego służą systemy zarządzania treścią (CMS)?
2. Wymień podstawowe cechy systemów zarządzania treścią.
3. Jakie oprogramowanie jest niezbędne do prawidłowego działania systemu Joomla!?
4. Do czego służą dwie podstawowe składowe systemu Joomla! *zaplecze administracyjne* i *witryna*?

ZADANIE

Wykorzystując system zarządzania treścią Joomla!, zaprojektuj strukturę witryny dla Twojej szkoły. W tworzonej witrynie umieść następujące moduły: *nagłówek*, *menu*, *logowanie*, *treść*. Rozplanuj położenie modułów. Dla każdego z modułów przygotuj odpowiednią treść. W module menu umieść trzy pozycje: *Historia szkoły*, *Patron*, *Kierunki kształcenia*. Kliknięcie każdej pozycji menu otworzy podstronę zawierającą odpowiednią treść. W module *treść* umieść podstawowe informacje o szkole. W module *nagłówek* wstaw zdjęcie szkoły.

10

Testowanie i dokumentowanie aplikacji

10.1. Testowanie aplikacji

Testowanie to ostatni etap tworzenia aplikacji internetowej. Testowanie aplikacji ma na celu nie tylko wykrycie błędów, ale również sprawdzenie, czy oprogramowanie realizuje funkcje, do których wykonywania zostało przeznaczone. Przeprowadzaniem testów całościowych aplikacji zajmują się najczęściej wyspecjalizowane firmy posiadające odpowiednie narzędzia.

Rodzaje przeprowadzanych testów:

- *Testy funkcjonalne* (optymalizacja) — oceniają zgodność ze standardami, analizują przyczyny opóźnień w wyświetlaniu się różnych elementów strony, sprawdzają dostępność witryny dla wyszukiwarek internetowych.
- *Testy użyteczności* (ergonomia) — oceniają przejrzystość treści, strukturę serwisu, oceniają nawigację i estetykę serwisu.
- *Testy kompatybilności* (architektura) — sprawdzają skalowalność aplikacji i bazy danych, oceniają architekturę serwera i jego konfigurację oraz architekturę aplikacji.
- *Testy wydajności* — szukają wąskich gardeł aplikacji webowej, analizują strukturę bazy danych i zapytań do bazy, oceniają kod źródłowy aplikacji.
- *Testy bezpieczeństwa* — sprawdzają, jakie zabezpieczenia przed atakami wykorzystuje aplikacja, w jakim stopniu działają elementy zabezpieczające systemu, w jakim stopniu próby ataków są wykrywane.

Po przeprowadzeniu testów powinien zostać przygotowany raport podsumowujący, w którym będzie można znaleźć: podsumowanie wykonanych prac, określenie potencjalnych zagrożeń, określenie poziomu bezpieczeństwa, proponowane środki zaradcze.

10.1.1. Testy automatyczne

Podczas testowania oprogramowania często mamy do czynienia z sytuacją, w której czynności związane z testowaniem wykonywane są automatycznie. Testy automatyczne często są też stosowane w testach wydajnościowych aplikacji. Zaletą takich testów jest to, że raz napisane mogą być wykorzystywane wielokrotnie. Ponadto testy zautomatyzowane można uruchamiać regularnie o określonych porach lub na pewnych etapach tworzenia aplikacji. Na przykład można ustawić testy tak, żeby były wykonywane każdej nocy i by na rano był dostępny raport.

10.1.2. Testy jednostkowe

Testy jednostkowe (ang. *unit test*) są testami automatycznymi, które porównują oczekiwany wynik funkcji z rzeczywistym rezultatem. Może to być test, w którym krok po kroku opisane są czynności, jakie ma wykonać program, i w którym jest określone, jaki powinien być rezultat działania programu. Może do tego zostać wykorzystany program komputerowy, który wczytuje do aplikacji odpowiednie dane wejściowe, a następnie sprawdza, czy na ekranie pojawiają się spodziewane wyniki. Na przykład przy sprawdzaniu funkcji sumującej dwie liczby całkowite można w łatwy sposób ustalić dane wejściowe oraz oczekiwany wynik.

Testy jednostkowe często są nazywane testami czarnej skrzynki, ponieważ tester nie musi znać kodu zawartego w badanej funkcji. Podaje dane wejściowe i sprawdza, czy otrzymuje oczekiwane wyniki.

10.1.3. Testy wydajnościowe

Wydajność aplikacji jest jednym z podstawowych aspektów funkcjonowania aplikacji internetowej. Ponieważ z aplikacji najczęściej korzysta wielu użytkowników, testy wydajnościowe mogą pokazać, czy aplikacja przy określonym obciążeniu będzie działała prawidłowo.

10.1.4. Testy z udziałem użytkownika

Testy z udziałem użytkownika są przeprowadzane, gdy w aplikacjach występują obszary, które są istotne z punktu widzenia użytkownika. Określają one, w jaki sposób użytkownik wchodzi w interakcję z programem, aby uzyskać oczekiwany wynik.

10.2. Zagrożenia

Coraz częściej aplikacje internetowe stają się celem ataków, dlatego szczególną rolę odgrywają zabezpieczenia programowe, tworzone przez programistów na poziomie kodu. Brak odpowiedniej ochrony może umożliwić włamanie do aplikacji lub atak na użytkowników z niej korzystających.

Ataki polegające na wstrzyknięciu i wykonaniu złośliwego kodu są jednymi z najpopularniejszych, a zarazem najdotkliwszych zagrożeń dotyczących aplikacji internetowych. Poniżej opisano najbardziej znane.

10.2.1. SQL injection

SQL injection jest jednym z najpowszechniej stosowanych i jednocześnie najniebezpieczniejszych ataków w aplikacjach webowych, które korzystają z bazy SQL. Atakujący wstrzykuje do aplikacji nieautoryzowany fragment kodu SQL, co pozwala mu na manipulowanie rekordami znajdującymi się w bazie danych. Może to prowadzić do nieautoryzowanego dostępu do danych (np. do danych osobowych, loginów, haseł, adresów e-mail), pominięcia procesu uwierzytelnienia lub modyfikacji bazy danych. Ochrona polega na weryfikowaniu zmiennych przekazywanych od użytkownika do aplikacji.

10.2.2. Atak XSS

XSS (ang. *Cross Site Scripting*) to rodzaj ataków skierowanych głównie w użytkowników. Atak polega na umieszczeniu kodu (najczęściej JavaScript) w treści witryny i umożliwieniu atakującemu zmodyfikowania wyglądu witryny internetowej. Możliwe staje się również przejęcie sesji użytkownika poprzez wykorzystanie przechwyconego pliku cookie. W takim ataku wykorzystywane są trzy rodzaje luk w aplikacji:

- aplikacja pozwala na wczytanie niebezpiecznego kodu — przyczyna: aplikacja nie weryfikuje danych pochodzących od osoby atakującej;
- aplikacja wysyła niebezpieczny kod — przyczyna: aplikacja, pobierając dane z bazy, nie poddaje ich ponownej weryfikacji;
- przeglądarka użytkownika wykonuje niebezpieczny kod — przyczyna trudna do wyeliminowania, ponieważ należałoby wyłączyć wykonywanie kodu JavaScript przez przeglądarkę użytkownika.

10.2.3. Atak CSRF

Ataki typu CSRF (ang. *Cross Site Request Forgeries*) wykorzystują mechanizm działania protokołu HTTP. Przeglądarka stron internetowych po odebraniu kodu HTML analizuje zawartość strony i pobiera te zasoby, które są potrzebne do poprawnego wyświetlenia strony (np. arkusze stylów, skrypty JavaScript). Atak polega na umieszczeniu złośliwego kodu HTML i wysłaniu go do przeglądarki użytkownika. Zasada działania ataku jest podobna do ataku XSS. Różnica polega na wykorzystaniu mechanizmu działania protokołu HTTP, a nie wykorzystaniu kodu JavaScript. Zabezpieczenie aplikacji jest podobne jak przed atakiem XSS — walidacja zmiennych pochodzących z zewnątrz oraz zabezpieczenie kodu HTML wysyłanego do użytkownika.

10.2.4. Źródła ataków i zapobieganie atakom

Wyświetlanie błędów aplikacji

Do znalezienia informacji o aplikacji mogą zostać wykorzystane wyświetlane w przeglądarce komunikaty o błędach. W takich komunikatach mogą się pojawiać szczegółowe informacje o wersjach użytych bibliotek czy komponentów, fragmenty kodu źródłowego, ścieżki instalacji aplikacji, fragmenty zapytań SQL lub hasła bazodanowe.

Google hacking

Jedną z technik stosowanych w celu znalezienia informacji o aplikacji, które pojawiają się w komunikatach o błędach, jest tzw. *Google hacking* — czyli poszukiwanie w Google zaindeksowanych stron zawierających komunikaty z opisami błędów.

Odwołania do nieistniejących zasobów

Kolejną przyczyną może być odwoływanie się do nieistniejącej strony, co często skutkuje ujawnieniem wersji systemu operacyjnego oraz użytych w nim komponentów. Dzieje się tak np. na serwerach WWW Apache czy serwerze aplikacyjnym Tomcat.

Audyty i testy bezpieczeństwa

Jedną z metod weryfikacji stanu bezpieczeństwa aplikacji webowej są audyty i testy bezpieczeństwa. Audyt bezpieczeństwa polega na sprawdzeniu systemu zgodnie z wytycznymi czy też z normami. Test bezpieczeństwa weryfikuje stopień zabezpieczenia systemu poprzez symulację działań potencjalnego atakującego.

10.3. Dokumentowanie aplikacji

Na dokumentację aplikacji składają się dokumentacja techniczna oraz dokumentacja użytkownika.

Dokumentacja użytkownika to opis aplikacji przeznaczony dla jej użytkownika. Na tę dokumentację składają się np. pliki pomocy, ogólne informacje o aplikacji i sposobie jej obsługi.

Dokumentacja techniczna jest przeznaczona dla osób, które będą administrowały aplikacją. Zawiera dokładny opis metod działania programu, zastosowanych algorytmów, rozmieszczenia i sposobu działania poszczególnych komponentów. Mogą się w niej znajdować fragmenty kodów źródłowych, graficzne reprezentacje algorytmów, diagramy przepływu danych, opisy UML czy XML.

Podstawowe elementy dokumentacji użytkowej to:

- *Opis funkcjonalny*. Ta część dokumentacji powinna opisywać przeznaczenie i główne możliwości aplikacji. Opis funkcjonalny powinien zawierać niezbędne informacje pozwalające ocenić, czy aplikacja spełnia określone potrzeby użytkownika.

- *Podręcznik użytkownika*. Jest to opis aplikacji przeznaczony głównie dla jej użytkowników. Część ta powinna wyjaśniać:
  - jak rozpoczynać i kończyć pracę z aplikacją,
  - jak obsługiwać najczęściej wykorzystywane funkcje aplikacji,
  - jak obsługiwać błędne operacje wykonane przez użytkownika,
  - jak korzystać z systemu pomocy.

Podręcznik użytkownika powinien zawierać prosty przykład korzystania z aplikacji.

Dokumentacja przeznaczona dla administratorów systemów powinna zawierać:

- *Kompletny opis aplikacji*. W tej części powinny się znajdować:
  - szczegółowy opis wszystkich funkcji aplikacji,
  - opis formatów danych,
  - opis błędów, które mogą się pojawić podczas pracy z systemem,
  - informacje o wszelkich ograniczeniach związanych z wykorzystaniem aplikacji.
- *Opis instalacji*. Część ta powinna zawierać opis procedury instalacji oraz dostrojenia aplikacji do środowiska, w którym będzie pracować.
- *Podręcznik administratora systemu*. Część ta powinna opisywać możliwości zmian konfiguracji systemu i sposoby udostępniania aplikacji użytkownikom.

Dokumentacja techniczna powinna zawierać opis:

- wykorzystanych technologii (np. XHTML, PHP, CSS),
- wykorzystanych bibliotek (np. jQuery itp.),
- zastosowanych wzorców projektowych, architektury aplikacji (np. MVC),
- zdefiniowanych użytkowników systemu (np. *admin*, *user*),
- zdefiniowanych funkcjonalności dla każdego użytkownika.

Dokumentacja techniczna może zawierać:

- kod źródłowy aplikacji,
- graficzne reprezentacje algorytmów.

PYTANIA I ZADANIA KONTROLNE

1. Wymień rodzaje testów całościowych aplikacji internetowej.
2. Jak działają testy jednostkowe?
3. Jaki jest cel przeprowadzania testów wydajnościowych?
4. Wymień najbardziej znane ataki na aplikacje internetowe.
5. Wymień najczęściej występujące źródła ataków na aplikacje internetowe.
6. Co powinna zawierać dokumentacja techniczna aplikacji?
7. Jakie elementy powinny znaleźć się w dokumentacji użytkowej aplikacji?

Skorowidz

A

abstrakcja, 34
ADO.NET, 313
adres IP, 287
Ajax, 38, 153
algorytm, 14
  cechy, 18
  implementacja, 21
  iteracyjny, 20
  klasyfikacja, 19–21
  liniowy, 19
  matematyczny, 21
  porządkujący, 21
  probabilistyczny, 21
  przeszukujący, 21
  rekurencyjny, 20–21
  sekwencyjny, 20
  szyfrujący, 21
  warunkowy, 20
analiza
  leksykalna, 12
  semantyczna, 12
  składniowa, 12
  syntaktyczna, 12
  znaczeniowa, 12
animowanie
  grafiki, 102–106
  tekstu, 100–102
Apache, 163–164
  instalowanie serwera w systemach Linux, 164–165
  instalowanie serwera w systemie Windows, 165–166
aplet, 10
aplety Java, 38
aplikacja, 10
aplikacja internetowa, 10, 37
aplikacja webowa, *Patrz* aplikacja internetowa
architektura
  4-warstwowa, 40–41
  Model-View-Controller (MVC), 41
argumenty funkcji
  w JavaScript, 64–65
  w PHP, 212–215
argumenty
  przekazywane przez wartość, 212–213
  przekazywane za pomocą referencji, 213
ASP.NET, 39, 44, 167
ataki w aplikacjach internetowych, 340–342
audyt bezpieczeństwa, 340

B

baner animowany, 104–106
baner (przesuwany tekst), 100–101
baner (tekst pływający), 101–102
bazy danych w PHP, 290–306
biblioteka jQuery, 119–121
biblioteki
  języka JavaScript, 119
  standardowe, 13–14
błędy uruchomienia, 27

C

C#, 313
CGI, 39
CIL, 311
CLI, 311
CLR, 311
CMS, 315
cookies, 271, 273–276
CSRF, 339

D

debuger, 25–26
definiowanie funkcji
  w JavaScript, 63–65
  w PHP, 207–208
definiowanie klasy w PHP, 243–244
destruktor, 252, 257
długość drogi, 22
dokumentacja
  programu, 28
  techniczna, 28, 340–341
  użytkownika, 28, 340–341
dokumentowanie aplikacji, 340–341
dołączanie plików w PHP, 229–231
DOM, 47, 83–84
domyślne argumenty funkcji w PHP, 214–215
drzewa, 35
drzewo algorytmu, 15–16, 22
DSN, 306
dziedziczenie, 35
  konstruktorów i destruktorów, 257
  w PHP, 252–254
dziel i zwyciężaj, 19

E

edytory, 25

F

filtry biblioteki jQuery, 124–128
Flash, 38
formaty przesyłanych danych w technologii Ajax, 161
formularz HTML, 260–261
formularze
  służące do wysyłania plików, 268–270
  w PHP, 259–271
framework, 44
funkcja, 63, 207
  $(), 121
  alert(), 68
  array_merge(tablica1, tablica2, ...), 285
  array_search(), 217
  array_unique(tablica), 285
  arsort(), 216
  asort(), 216
  closedir(), 238
  count(), 215
  date(), 219–220
  define(), 197
  die(), 242
  each(), 216
  empty(), 263–268
  error, 159
  eval(), 68
  exit(), 242
  fclose(), 233
  feof(), 234
  fgetc(), 235–236
  fgets(), 234
  file(), 237
  file_exists(), 231
  file_get_contents(), 237
  filesize(), 232
  fopen(), 232
  foreach(), 216–217
  fread(), 236–237
  ftp_close(), 289
  ftp_connect(), 288
  ftp_fget(), 288–289
  ftp_login(), 288
  fwrite(), 233–234
  getdate(), 218–219
  include(), 229–231
  is_file(), 231
  is_uploaded_file(), 270
  isFinite(), 67–68
  isNaN(), 67
  isSet(), 262–263, 278
  krsort(), 216
  ksort(), 216
  list(), 216
  ltrim(), 225
  mkdir(), 237–238
  mktime(), 220–221
  move_uploaded_file(), 270
  mysql_affected_rows(), 297
  mysql_close(), 291
  mysql_connect(), 290–291
  mysql_fetch_array(), 293–294
  mysql_fetch_row(), 293
  mysql_get_server_info() 291
  mysql_num_rows(), 293
  mysql_query(), 293
  mysql_select_db(), 292
  nl2br(), 222
  opendir(), 238
  parseFloat(), 67
  parseInt(), 66–67
  print(), 181
  readdir(), 238
  readfile(), 237
  require(), 229–231
  rmdir(), 238
  rsort(), 216
  rtrim(), 225
  scandir(), 239
  serialize(), 278
  session_destroy(), 277
  session_id(), 277
  session_start(), 277
  setcookie(), 271–272
  settype(), 192
  sizeof(), 215
  sort(), 216
  strcasecmp(), 229
  strcmp(), 229
  strlen(), 225
  strpos(), 227
  strstr(), 226
  strtok(), 228
  strtolower(), 224
  strtoupper(), 224
  substr(), 227
  success(), 159
  time(), 217–218
  toDateString(), 77
  toGMTString(), 77
  toLocaleDateString(), 77
  toLocaleString(), 77
  toLocaleTimeString(), 77
  toString(), 77
  toTimeString(), 77
  touch(), 232
  toUTCString(), 77
  trim(), 225
  ucfirst(), 224
  ucwords(), 224
  unlink(), 232
  unserialize(łańcuch), 285
  wordwrap(), 222–224
funkcje
  analizowania ciągów znaków, 225–229
  daty i czasu, 217–221
  formatowania ciągów, 221–225
  tablic, 215–217
  usuwania ciągu znaków, 225
  wyjścia, 242–243
  wykonujące operacje na plikach, 231–237
  zmiany wielkości liter, 224
  zwrotne żądania Ajax, 158–160

funkcje wbudowane
w JavaScript, 66–69
w PHP, 215–229

G

Google hacking, 340

H

hermetyzacja, 34–35
w PHP, 248–250
heurystyka, 19
hierarchia klas, 35
hierarchia obiektów DOM, 84
hover, 129

I

IDE, *Patrz* zintegrowane środowisko programistyczne
IIS, 167–168
implementacja algorytmu, 14
indeksowanie ciągu znaków, 225–226
indeksy tablicy zwracanej przez funkcję getdate(), 218
inkapsulacja, *Patrz* hermetyzacja
instalacja XAMPP, 174–179
instalowanie PHP, 171–173
instrukcja
break
w JavaScript, 62
w PHP, 204–205
continue
w JavaScript, 62
w PHP, 205
echo, 181
function, 207
global, 209–210
if ... else, 57
new, 74
prototype, 74–75
return
w JavaScript, 63
w PHP, 208
SQL CREATE DATABASE, 305
SQL CREATE TABLE, 305
sterująca
w JavaScript, 56–62
w PHP, 198–207
switch
w JavaScript, 58
w PHP, 200–201
throw, 258
try...catch, 258
UPDATE, 301
var, 65–66
warunkowa
w JavaScript, 57
w PHP, 198–200
interpreter, 10
ISAPI, 39

J

JavaScript, informacje ogólne 38, 48–50
język
hybrydowy, 30
pośredni, 311
programowania, 9–11
skryptowy, 13, 47
Joomla!, 315–321
grafika 328–329
komponenty 333–334
moduły 330–333
szablony 334–335
tworzenie artykułów 323–326
tworzenie kategorii, 321–323
tworzenie menu 326–328
tworzenie treści portalu, 321
JSON, 161
JSP, 39

K

klasa, 31, 33, 73, 243
klasa
bazowa, 35
Exception, 257
pochodna, 35
klasy znaków, 111
klauzula WHERE, 296–297
klucz REMOTE_ADDR, 287
kolejność wykonywania skryptów, 99
komentarze
w JavaScript, 50–51
w PHP, 183–184
kompilacja, 11–13, 27
kompilator, 9
komunikacja między serwerem a użytkownikiem w technologii Ajax, 154
konsolidacja, 10, 27
konsolidator, 10
konstruktor
w JavaScript, 73
w PHP, 250–252, 257
kontroler, 43–44
kotwica, 92

L

linker, 10
lista kroków, 15

M

magiczne stałe, 197–198
menu nawigacyjne, 132
metoda, 70
addClass(), 127
animate(), 137–144
append(), 147
bind(), 128
CSS, 127
document.write(), 89
exec(), 308–309

fadeIn(), 136
fadeOut(), 136
getElementById(), 86
getElementByTagName(), 86
hide(), 136
history.back(), 89
history.forward(), 89
history.go(), 89
join(), 81
klasy w PHP, 244
location.reload(), 91
location.replace(), 91
open(), 157–158
query(), 307–308
ready(), 121
reset (), 114
reverse(), 81
send(), 158
show(), 136
slideDown(), 136
slideToggle(), 136
slideUp(), 136
sort(), 81
substring(), 75, 102
toggle(), 123
toLowerCase(), 75
toUpperCase(), 75
window.setTimeout(), 102
zachłanna, 19
metoda GET
w technologii Ajax, 158
w PHP, 261–262
metoda POST
w technologii Ajax, 158
w PHP, 263–264
model, 41–43
modelowanie obiektowe, 33–34
moduł, 10
modyfikatory dostępu
w PHP, 248
MSIL, 311

N

naprzemienne bloki kodu PHP i HTML, 205–207
narzędzia
aplikacji internetowych, 38–39
programistyczne, 25, 44–45

O

obiekt, 31, 243
anchor, 91–92
Array, 79–82
Date, 76–78
document, 71, 85–89
form, 92–94
history, 71–72, 89–90
klasy PDO, 306
link, 91
location, 71, 90–91
navigator, 71
RegExp (), 110
String, 75–76
window, 70, 85
XMLHttpRequest, 153–157
obiektowość, 21, 34
obiekty
w JavaScript, 70, 75–82
w PHP, 246–248
obiekty
predefiniowane
w JavaScript, 72
przeglądarki, 70–72
obsługa protokołu FTP
w PHP, 288
obsługa zdarzeń
w JavaScript, 96–99
w jQuery, 128–131
opcje żądania Ajax, 157
operacje na katalogach, 237–239
operatory arytmetyczne
w JavaScript, 54
w PHP, 193
operatory bitowe
w JavaScript, 55
w PHP, 194
operatory dekrementacji
w JavaScript, 54
w PHP, 196–197
operatory inkrementacji
w JavaScript, 54
w PHP, 196
operatory i wyrażenia w PHP, 193–197
operatory konkatenacji, 196
operatory logiczne
w JavaScript, 56
w PHP, 194–195
operatory porównania
w JavaScript, 55
w PHP, 194
operatory przypisania
w JavaScript, 56
w PHP, 195
operatory warunkowe, 201
opis słowny algorytmu, 14–15
optymalizacja
kodu wynikowego, 12
programu, 28

P

paradygmat programowania, 28
Pascal, 29
PDO, 306–309
pętla, 59, 202
pętla do ... while
w JavaScript, 61
w PHP, 203
pętla for
w JavaScript, 59–60
w PHP, 202
pętla foreach, 203–204
pętla while
w JavaScript, 60–61
w PHP, 202–203

PHP, informacje ogólne, 39, 169–170
planowanie programu komputerowego, 26
platforma
.NET Compact Framework, 313
Eclipse, 44
NetBeans IDE, 45
Zend, 45
platforma programistyczna, 44
.NET, 311–313
plik php.ini, 180
pliki cookies, 271, 273–276
podręcznik
administratora systemu, 341
użytkownika, 341
pole
input typu file, 269
typu SELECT, 263
polecenie INSERT INTO, 297
polimorfizm, 35
połączenie z bazą danych w PHP, 290–292
porównania ciągów, 228–229
poszukiwanie i wyliczanie, 19
praca
rownoległa, 21
sekwencyjna, 21
wielowątkowa, 21
private, 248–249
proceduralność, 21
proces przetwarzania kodu PHP, 170
program, 9
program komputerowy, 10
programowanie, 9
dynamiczne, 19
obiektowe, 30–32, 243
proceduralne, 30
strukturalne, 29
uogólnione, 30
projektowanie aplikacji internetowych, 42–43
protected, 248–249
przekazywanie danych z formularza, 261–268
przesłanianie składowych, 254–257
pseudokod, 15
public, 248–249

R

rekurencja, 21
reprezentacja algorytmów, 14
rzutowanie typów, 192

S

schemat blokowy, 16–18
selektory, 121–124
semantyka, 11
serwer aplikacji, 40
serwery WWW, 163
sesja, 277
sesje w PHP, 277–287
składnia, 10
heredoc, 187–188
języka JavaScript, 50–56
języka PHP, 183–197
nowdoc, 188
składowe klasy, 243
skrypt, 13
skrypty PHP, 170, 180
slider, 104–106
słowa kluczowe, 9
słowo kluczowe this, 73
sortowanie, 216
bąbelkowe, 23
liczb, 23
sprawdzanie wypełnienia pól formularza, 106–109
SQL injection, 339
stałe, 197
stałe predefiniowane, 197
struktura
aplikacji internetowej, 39–42
języka PHP, 180–183
SVG, 47
symbole
flagi, 111
wzorca, 110
systemy zarządzania treścią, 315
szablon Beez3, 330

Ś

środowisko sieciowe w PHP, 287

T

tabela, 79
tablica, 188
$_FILES, 269
$_SERVER, 287
$_SESSION, 277–278
$GLOBALS, 210
asocjacyjna, 188, 190–191
indeksowana, 188–190
test bezpieczeństwa, 340
testowanie
aplikacji, 337
programu komputerowego, 27–28
skryptów, 115–117
testy
automatyczne, 338
bezpieczeństwa, 337
czarnej skrzynki, 338
funkcjonalne, 337
jednostkowe, 338
kompatybilności, 337
użyteczności, 337
wydajności, 337
wydajnościowe, 338
z udziałem użytkownika 338
translator, 9

tworzenie
bazy danych, 305
obiektów
w JavaScript, 72–73
w PHP, 246–248
z użyciem
konstruktora, 73–75
pliku cookie w PHP, 271–272
programu komputerowego, 26–28
tabeli w bazie danych, 305
tablicy, 79–80
typy danych, 11
array
w JavaScript, 53
w PHP, 188
boolean
w JavaScript, 53
w PHP, 186
float, 186
integer, 186
liczbowy, 52
string
w JavaScript, 52–53
w PHP, 187
null
w JavaScript, 53
w PHP, 191
object
w JavaScript, 53
w PHP, 191
resource, 191
undefined, 53
w JavaScript, 52–53
w PHP, 185–193

U

usuwanie pliku cookie, 273

V

Visual Studio Express 2012 for Web, 314
Visual Studio Express 2012 for Windows Desktop, 314
Visual Studio, 313

W

walidacja formularzy
przy zastosowaniu biblioteki jQuery, 147–150
w JavaScript, 106–115
warstwa
aplikacji, 40
danych, 40
klienta, 40
WEB.PI, 167
widok, 43
Windows Forms, 313
właściwość
onreadystatechange, 155
readyState, 155
responseText, 156
responseXML, 157
status, 156
właściwości
klasy w PHP, 244
obiektu, 70
wstępne przetwarzanie kodu, 11–12
wyjątki w PHP, 257–259
wyrażenia regularne, 110
wysokość drzewa, 22
wysyłanie żądania Ajax do serwera, 157
wzorce, 110–114
wzorzec projektowy, 42
MVC, 43–44

X

XAMPP, 173–179
XML, 153
XSS, 339

Z

zapytania do bazy danych, 293–305
zasięg zmiennej, 65, 208
zastosowanie biblioteki jQuery na stronie internetowej, 131–147
animacje, 136–144
pokaz zdjęć, 144–147
pokazywanie i ukrywanie treści, 132–135
zdarzenia, 95
biblioteki jQuery, 129–131
dokumentu, 99
elementów formularza, 99
formularza
w JavaScript, 98
w jQuery, 131
klawiatury, 98
myszy
w JavaScript, 96–98
w jQuery, 129
zerowanie pól formularza, 114–115
zgłaszanie wyjątków w PHP, 257–258
zintegrowane środowisko programistyczne, 26
złożoność
czasowa algorytmu, 22
obliczeniowa
algorytmu, 21–22
oczekiwana, 22
optymistyczna, 22
pamięciowa
algorytmu, 22
pesymistyczna, 22
zmiana grafiki w JavaScript, 102–104
zmiana typu zmiennej, 192–193

zmienne, 51, 184
  predefiniowane 185
  sesji, 278
  statyczne, 211–212
  superglobalne, 185
  w JavaScript, 51–52
  w PHP, 184–185
zmienne globalne
  w JavaScript, 65
  w PHP, 209–210
zmienne lokalne
  w JavaScript, 65
  w PHP, 185, 210
znaczniki
  bloku PHP, 180
  formatujące funkcji date(), 219–220
znajdowanie
  największego elementu w zbiorze nieuporządkowanym, 24
  podciągów, 226–228

Ź

źródło danych dla MySQL, 306



Kwalifikacja E.14 Część 3

Tworzenie aplikacji internetowych

Podręcznik do nauki zawodu **technik informatyk**

Technik informatyk nie jest zwykłym użytkownikiem komputerów. Jeśli uczeń wybiera szkołę o takim profilu, z czasem staje się prawdziwym komputerowym ekspertem.

Kwalifikacje wyodrębnione w zawodzie technik informatyk dotyczą między innymi tworzenia aplikacji internetowych i baz danych oraz administrowania tymi bazami. Trzecia część podręcznika, omawiająca kwalifikację E.14, składa się z dziesięciu rozdziałów, których budowa pozwala na realizację treści programowych w sposób wybrany przez nauczyciela. Przygotowane materiały obejmują nie tylko zagadnienia teoretyczne, ale także projekty różnych zadań oraz – co bardzo istotne – sposoby ich praktycznej realizacji, dzięki czemu prowadzą do uzyskania efektów kształcenia wymienionych w podstawie programowej. Podczas zajęć uczniowie zdobędą niezbędną wiedzę na temat aplikacji internetowych. Nie tylko będą samodzielnie programować w wybranych technologiach, ale też staną się świadomi ich indywidualnych cech, wad i zalet. Poznają działanie serwerów stron internetowych oraz zasady umieszczania opracowanych aplikacji w internecie. Zaczną wykorzystywać zasoby biblioteki jQuery do tworzenia własnych aplikacji. Będą korzystać z systemów zarządzania treścią CMS przy tworzeniu menu, zamieszczaniu grafiki i artykułów oraz dostosowywaniu aplikacji do potrzeb użytkownika. Na koniec przeprowadzą testy aplikacji oraz przeanalizują zagrożenia. Tak skonstruowany podręcznik pomaga także w samodzielnym poszerzaniu umiejętności.

Technik informatyk to doskonały, charakteryzujący się wysoką jakością, kompletny zestaw edukacyjny, przygotowany przez dysponującego ogromnym doświadczeniem lidera na rynku książek informatycznych – wydawnictwo Helion.

W skład zestawu *Technik informatyk* wchodzą także:

Kwalifikacja E.12. Montaż i eksploatacja komputerów osobistych oraz urządzeń peryferyjnych. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.13. Projektowanie lokalnych sieci komputerowych i administrowanie sieciami. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.14. Część 1. Tworzenie stron internetowych. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.14. Część 2. Tworzenie baz danych i administrowanie bazami. Podręcznik do nauki zawodu technik informatyk

Podręczniki oraz inne pomoce naukowe należące do tej serii zostały opracowane z myślą o wykształceniu kompetentnych techników, którzy bez trudu poradzą sobie z wyzwaniami w świecie współczesnej informatyki. Według nowych przepisów, aby otrzymać tytuł technika informatyka, należy potwierdzić trzy kwalifikacje wyodrębnione w tym zawodzie – to niewątpliwe wyzwanie i dla adeptów nauki o komputerach, i dla ich pedagogów. Ta książka pozwoli zarówno przygotować się do egzaminów, jak i uzyskać wiedzę i umiejętności przydatne w przyszłej pracy.

helion.pl
księgarnia internetowa

Nr katalogowy: 15976

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-246-8132-7

9 788324 681327