



DO NOWEJ  
PODSTAWY PROGRAMOWEJ

## Część 3

Programowanie aplikacji  
internetowych

### Kwalifikacja INF.03

Tworzenie i administrowanie  
stronami i aplikacjami internetowymi  
oraz bazami danych



**Podręcznik do nauki zawodu  
technik informatyk i technik programista**

Jolanta Pokorska

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiekolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicielami.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Joanna Zaręba

Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.

Helion SA  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zatrzymaj pod adres  
[http://helion.pl/user/opinie?inf033\\_ebook](http://helion.pl/user/opinie?inf033_ebook)  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7054-8

Copyright © Helion 2020

- [Poleć książkę na Facebook.com](#)
- [Kup w wersji papierowej](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

<b>Wstęp</b> .....	5
<b>Rozdział 1.</b> Podstawy programowania .....	7
<b>1.1.</b> Zasady programowania .....	7
<b>1.2.</b> Proste algorytmy .....	15
<b>1.3.</b> Środowiska programistyczne .....	19
<b>1.4.</b> Pytania i zadania .....	22
<b>Rozdział 2.</b> Aplikacje internetowe .....	25
<b>2.1.</b> Wprowadzenie .....	25
<b>2.2.</b> Wzorce projektowe .....	26
<b>2.3.</b> Serwery aplikacji internetowych .....	28
<b>2.4.</b> Pakiet XAMPP .....	28
<b>2.5.</b> Pytania i zadania .....	33
<b>Rozdział 3.</b> Język JavaScript .....	35
<b>3.1.</b> Wprowadzenie .....	35
<b>3.2.</b> Struktura języka JavaScript .....	35
<b>3.3.</b> Składnia języka JavaScript .....	38
<b>3.4.</b> Instrukcje sterujące .....	49
<b>3.5.</b> Funkcje .....	59
<b>3.6.</b> Obiekty .....	67
<b>3.7.</b> Obiekty wbudowane języka JavaScript .....	74
<b>3.8.</b> Obiekty DOM .....	83
<b>3.9.</b> Obsługa zdarzeń .....	98
<b>3.10.</b> Wykorzystanie skryptów na stronie internetowej .....	103
<b>3.11.</b> Walidacja formularzy .....	109
<b>3.12.</b> Pytania i zadania .....	120
<b>Rozdział 4.</b> Biblioteka jQuery .....	123
<b>4.1.</b> Opis biblioteki .....	123
<b>4.2.</b> Zdarzenia biblioteki jQuery .....	133
<b>4.3.</b> Zastosowanie biblioteki jQuery na stronie internetowej .....	136

<b>4.4.</b> Walidacja formularzy .....	152
<b>4.5.</b> Pytania i zadania .....	155
<b>Rozdział 5.</b> Inne biblioteki języka JavaScript .....	157
<b>5.1.</b> Wprowadzenie .....	157
<b>5.2.</b> Angular .....	158
<b>5.3.</b> React.js .....	165
<b>5.4.</b> Pytania i zadania .....	169
<b>Rozdział 6.</b> Język PHP .....	171
<b>6.1.</b> Wprowadzenie .....	171
<b>6.2.</b> Struktura języka PHP .....	173
<b>6.3.</b> Składnia języka PHP .....	175
<b>6.4.</b> Instrukcje sterujące .....	192
<b>6.5.</b> Funkcje .....	205
<b>6.6.</b> Funkcje wbudowane .....	213
<b>6.7.</b> Funkcje obsługi plików .....	227
<b>6.8.</b> Obsługa formularzy .....	240
<b>6.9.</b> Pliki cookies i sesje .....	251
<b>6.10.</b> Bazy danych w PHP .....	267
<b>6.11.</b> Biblioteka PDO .....	290
<b>6.12.</b> Pytania i zadania .....	293
<b>Rozdział 7.</b> Walidacja kodu aplikacji .....	295
<b>7.1.</b> Wprowadzenie .....	295
<b>7.2.</b> Testy aplikacji .....	296
<b>7.3.</b> Debugowanie aplikacji .....	298
<b>7.4.</b> Pytania i zadania .....	302
<b>Rozdział 8.</b> Dokumentowanie aplikacji .....	303
<b>8.1.</b> Komentarze .....	303
<b>8.2.</b> Tworzenie dokumentacji programu .....	306
<b>8.3.</b> Automatyczne generowanie dokumentacji użytkownika .....	309
<b>8.4.</b> Pytania i zadania .....	311
<b>Bibliografia</b> .....	312
<b>Skorowidz</b> .....	313

# Wstęp

Szybki rozwój internetu i pojawienie się nowych technologii pozwoliły na tworzenie aplikacji internetowych, z których można korzystać za pomocą przeglądarki internetowej. Aplikacje webowe stały się narzędziem do nowoczesnego zarządzania firmą, umożliwiały także stałą kontrolę nad procesami biznesowymi z każdego miejsca na świecie.

Nauka zawodów technik informatyk i technik programista pozwala na pogłębienie wiedzy z zakresu projektowania i tworzenia aplikacji internetowych. Zdobyte kwalifikacje dają możliwość szybkiego znalezienia bardzo atrakcyjnej pracy.

Podręcznik *Kwalifikacja INF.03. Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych. Część 3. Programowanie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk i technik programista* zawiera treści przeznaczone dla osób kształcących się w zawodach technik informatyk i technik programista. Podręcznik został opracowany zgodnie z nową podstawą programową, obowiązującą od 1 września 2019 r., i obejmuje materiał kwalifikacji *INF.03. Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych* w zakresie programowania aplikacji internetowych.

W publikacji zostały omówione zagadnienia teoretyczne z zakresu programowania aplikacji webowych. Zamieszczono też liczne ćwiczenia, które ułatwia nabycie umiejętności praktycznych, a proponowane rozwiązania pomogą w ugruntowaniu zdobytej wiedzy.

Podręcznik składa się z ośmiu rozdziałów. Ich budowa pozwala na realizację treści programowych w sposób wybrany przez nauczyciela oraz umożliwia samodzielne rozwiązanie umiejętności przez uczniów.

Rozdział 1., „Podstawy programowania”, zawiera omówienie podstawowych zagadnień z zakresu programowania i algorytmiki. Dotyczy efektów związanych z przestrzeganiem zasad programowania. Efektami tymi są: stosowanie zasad programowania strukturalnego, analiza problemów programistycznych i korzystanie z algorytmów.

Rozdział 2., „Aplikacje internetowe”, zawiera omówienie podstawowych zagadnień z zakresu użycia aplikacji internetowych. Dotyczy efektów związanych z tworzeniem oraz wykorzystywaniem aplikacji webowych. Efektami tymi są: rozpoznanie technologii działających po stronie klienta i serwera, identyfikowanie wzorców projektowych, korzystanie z frameworków, instalowanie serwerów WWW.

Rozdział 3., „Język JavaScript”, zawiera omówienie zagadnień z zakresu stosowania skryptów wykonywanych po stronie klienta przy tworzeniu aplikacji internetowych. Dotyczy efektów związanych ze stosowaniem funkcji, obiektów i metod w języku JavaScript, z tworzeniem własnych funkcji, obiektów i metod oraz testowaniem powstającej aplikacji i modyfikowaniem jej kodu źródłowego. Efektami tymi są: identyfikowanie operatorów arytmetycznych, bitowych i logicznych, stosowanie obsługi zdarzeń myszy

i klawiatury, definiowanie skryptów obsługujących formularze i kontrolki HTML, wykorzystywanie mechanizmów validacji formularzy HTML, korzystanie z funkcji modelu DOM, analizowanie poprawności tworzonych obiektów i metod.

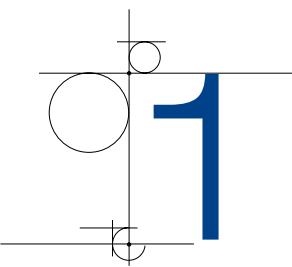
Rozdział 4., „Biblioteka jQuery”, zawiera omówienie zagadnień z zakresu wykorzystania biblioteki jQuery do tworzenia własnych aplikacji. Dotyczy efektów związanych ze stosowaniem funkcji, metod i obiektów biblioteki jQuery, tworzeniem własnych funkcji, obiektów i metod z użyciem biblioteki jQuery oraz wykorzystywaniem ich na stronie internetowej. Efektami tymi są: identyfikowanie i stosowanie instrukcji, metod, obiektów i zdarzeń biblioteki jQuery, używanie odpowiednich instrukcji, procedur i metod biblioteki jQuery do tworzenia efektów na stronie internetowej oraz do validacji formularzy.

Rozdział 5., „Inne biblioteki języka JavaScript”, zawiera omówienie zagadnień z zakresu wykorzystania bibliotek Angular i React do tworzenia własnych aplikacji. Dotyczy efektów związanych z przygotowaniem środowiska uruchomieniowego tych bibliotek oraz stosowaniem ich do tworzenia strony internetowej. Efektami tymi są: dobieranie środowisk programistycznych do określonych zadań, tworzenie programów w wybranych środowiskach programistycznych, instalowanie i konfigurowanie środowisk programistycznych, korzystanie z bibliotek i frameworków języka JavaScript.

Rozdział 6., „Język PHP”, zawiera omówienie podstawowych zagadnień z zakresu stosowania skryptów wykonywanych po stronie serwera przy tworzeniu aplikacji internetowych. Dotyczy efektów związanych z wykorzystaniem wbudowanych typów danych, stosowaniem funkcji i tworzeniem własnych funkcji w języku PHP oraz testowaniem powstającej aplikacji i modyfikowaniem jej kodu źródłowego. Efektami tymi są: identyfikowanie operatorów arytmetycznych, bitowych i logicznych, tworzenie własnych typów danych, stosowanie instrukcji i funkcji języka PHP do określonych zadań, wykorzystywanie operacji na plikach i katalogach, tworzenie i validacja formularzy, wysyłanie danych z formularza, używanie plików cookie i sesji, stosowanie bibliotek do obsługi bazy danych, analizowanie poprawności tworzonych aplikacji.

Rozdział 7., „Walidacja kodu aplikacji”, zawiera omówienie zagadnień z zakresu testowania i debugowania kodu aplikacji internetowej. Dotyczy efektów związanych z analizą testów tworzonej aplikacji oraz sposobami debugowania aplikacji. Efektami tymi są: analiza błędów w kodzie źródłowym programu, poprawianie błędów, stosowanie debugera w przeglądarce internetowej, rozpoznawanie rodzajów testów.

Rozdział 8., „Dokumentowanie aplikacji”, zawiera omówienie zagadnień z zakresu dokumentowania tworzonej aplikacji. Dotyczy efektów związanych z opracowywaniem dokumentacji programu. Efektami tymi są: identyfikowanie sposobów dokumentowania powstającej aplikacji, tworzenie dokumentacji programu, przygotowywanie instrukcji użytkownika, stosowanie komentarzy w celu automatycznego generowania dokumentacji użytkownika.



# Podstawy programowania

## 1.1. Zasady 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 i zarządzania przepływem sterowania. Niektóre języki programowania mają 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** — dostępne typy danych, ich właściwości oraz operacje, które mogą być wykonane na wartościach danego typu.

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

Niektóre z tych języków wywodzą się z języka C, na przykład C++ czy C#. Języki Java i Python są popularne, gdyż pozwalają na bardzo szybkie tworzenie aplikacji. Wydajniejsze języki, takie jak C czy C++, posłużyły do napisania programów biurowych typu edytory tekstu czy edytory grafiki. Języki PHP i Java są popularne w aplikacjach internetowych. Python bywa wykorzystywany w administrowaniu systemami i do tworzenia aplikacji.

Wybór języka programowania może zależeć od przeznaczenia końcowej aplikacji, indywidualnych upodobań lub strategii firmy tworzącej oprogramowanie. Najlepszym

wyjściem jest wybór języka programowania najbardziej dostosowanego do rozwiązywanego zadania i istniejącej infrastruktury.

Kod źródłowy programu jest tworzony i zapisywany w formacie tekstowym. Rozszerzenie pliku zależy od wybranego języka programowania. Identyfikuje ono język, w którym program został napisany, na przykład *c* dla języka C, *cpp* dla C++, *java* dla Javy, *cs* dla C#. Niezależnie od rozszerzenia plik zawierający kod źródłowy programu można otworzyć w dowolnym edytorze tekstu.

### 1.1.1. Paradygmaty programowania

**Paradygmat programowania** to pewien wzorzec określający sposób pisania i 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. W większości języków można programować strukturalnie.

W **programowaniu obiektowym** 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.

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

## 1.1.2. Programowanie strukturalne

W programowaniu strukturalnym kod programu jest dzielony na hierarchicznie ułożone bloki. Program tworzony jest z wcześniej zdefiniowanych struktur, takich jak sekwencja (ciąg kolejnych instrukcji), instrukcja warunkowa (`if`, `if...else`), pętla (`while`, `repeat`), instrukcja wyboru. Struktury te są stosowane do małych bloków programu zawierających podstawowe instrukcje typu podstawienia, wejścia-wyjścia lub wywołania procedur i funkcji. Dzięki temu kod jest przejrzysty i łatwy w utrzymaniu.

Występujące w językach strukturalnych instrukcje skoku (`goto`) lub przerwania (`continue`, `switch`) są niezgodne z zasadami strukturalności, ale pojawiają się w językach programowania, ponieważ zwiększą czytelność programu.

Prawie w każdym języku można programować strukturalnie. Jednak niektóre z nich są do tego celu specjalnie przeznaczone. Modelowym przykładem języka strukturalnego był język Pascal.

Reasumując: programowanie strukturalne zachęca do dzielenia programu na bloki (moduły, podprogramy), a powstałych bloków na mniejsze jednostki w kolejności wykonywania. Wykorzystuje instrukcje sterowania i pętle do kontrolowania przebiegu programu. Kod staje się dzięki temu bardziej przejrzysty i wzrasta jego wydajność.

Sposób programowania w tym paradygmacie może być różny w zależności od zastosowanego systemu operacyjnego i języka programowania. Jednak podstawowe zasady programowania powinny zostać niezmienne.

### Zmienna

Jednym z podstawowych elementów języków programowania jest **zmienna**. Jest ona abstrakcją komórki pamięci komputera. Każdą zmienną można opisać za pomocą nazwy, adresu, wartości, typu, okresu życia i zakresu widoczności.

**Nazwa** — określa nazwę, przez którą następuje odwołanie do zmiennej. Dla nazwy powinny zostać zdefiniowane: zakres dozwolonych znaków, liczba dozwolonych znaków, rozróżnialność małych i dużych liter.

**Adres** — określa bieżący adres zmiennej w pamięci.

**Wartość** — określa wartość przypisaną do zmiennej.

**Typ** — określa zbiór dopuszczalnych wartości, jakie zmienna może przyjmować.

**Zakres widoczności** — określa blok instrukcji, w których zmienna jest widoczna i można się do niej odwołać.

**Okres życia** — określa czas, przez który zmienna istnieje. Przykładowo okres życia zmiennej zadeklarowanej w funkcji zaczyna się od wejścia do funkcji i kończy się w momencie zakończenia funkcji. Istnieją również zmienne zadeklarowane w funkcji, które widoczne są tylko w tej funkcji, ale okres życia dotyczy całego czasu wykonania programu.

## Proste typy danych

Typ to zbiór wartości, które może przyjmować zmienna. Większość języków programowania ma zestaw typów prostych.

**Typ całkowity** (`int`, `integer`) odpowiada liczbom całkowitym, ale mogą wystąpić jego warianty związane z rozmiarem liczby (`byte`, `short`, `long`) i jej znakiem (`signed`, `unsigned`).

**Typy zmiennopozycyjne** odpowiadają liczbom rzeczywistym (`float`, `double`).

**Typy znakowe** przechowują informacje o znakach zapisane w kodzie ASCII lub coraz częściej w Unicode (`char`, `character`).

**Typ logiczny** przechowuje wartość logiczną *Prawda* lub *Fałsz* i jest zapisywany za pomocą jednego bitu lub całego bajta.

**Typy stałoprzecinkowe** mają ustaloną liczbę cyfr i miejsc po przecinku.

Z typów prostych można tworzyć typy złożone, na przykład *rekordy*, *tablice*, *typy wyliczeniowe*.

## Operatory

Z każdym typem związana jest grupa operacji, które mogą zostać wykonane na wartościach tego typu. Przykładowo:

**Typ całkowity** — operacje dodawania (+), odejmowania (-), mnożenia (\*), dzielenia (/).

**Typ logiczny** — operacje koniunkcji (`And`), alternatywy (`Or`), negacji (`Not`).

**Typ znakowy** — operacja łączenia tekstów (+).

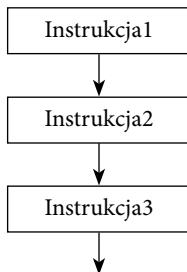
## Operacje wejścia-wyjścia

Operacje wejścia-wyjścia pozwalają na pobieranie i wypisywanie danych z konsoli.

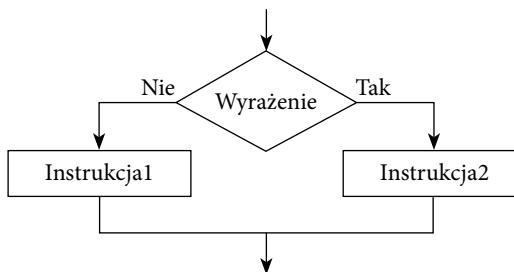
## Instrukcje

**Sekwencja** — ciąg kolejnych poleceń (rysunek 1.1).

**Instrukcja warunkowa** — pozwala na rozgałęzienie przebiegu działania programu (rysunek 1.2).



Rysunek 1.1. Sekwencja



Rysunek 1.2. Instrukcja warunkowa

Jeżeli wartość wyrażenia jest prawdą, to zostanie wykonana *Instrukcja2*, w przeciwnym razie — *Instrukcja1*.

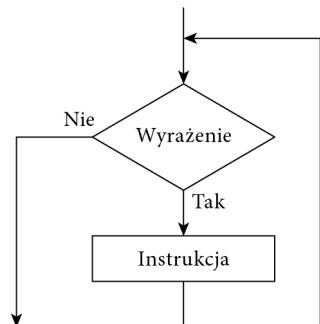
**Pętla** — pozwala na wykonywanie bloku programu kilkakrotnie w pętli, dopóki wartością sprawdzanego wyrażenia jest prawda (rysunek 1.3).

Dopóki wartość wyrażenia jest prawdą, wykonywana jest *Instrukcja*.

### Przykład 1.1

Użycie instrukcji warunkowej w języku C++:

```
#include <iostream>
using namespace std;
int main()
{
    int liczba1 = 30, liczba2 = 16;
    if (liczba1 > liczba2)
        cout << "Liczba: " << liczba1 << " jest wieksza od liczby " << liczba2;
    else
        cout << "Liczba: " << liczba2 << " jest wieksza od liczby " << liczba1;
    return 0;
}
```



Rysunek 1.3. Pętla

## 1.1.3. Programowanie obiektowe

Języki obiektowe w różnym stopniu stosują zasady obiektowości. Popularnymi językami obiektowymi są Java, C# 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 (abstrakcyjnych typów danych, dziedziczenia, polimorfizmu). Językami w pełni obiektowymi są C# 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 (na przykład liczbę znaków

łańcucha, wymiary okna) lub pozwalają określić jego stan. Są też nazywane polami obiektu, zmiennymi lub zmiennymi składowymi. Metody to funkcje, które wykonują operacje na obiekcie.

**Klasa** jest specjalną strukturą opisującą grupę powiązanych ze sobą obiektów. Definiuje ona metody (funkcjonalność) oraz atrybuty wybranych obiektów. Obiekt jest 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 skonstruowany, aby jak najlepiej odzwierciedlał opisywany fragment rzeczywistości. Klasa opisuje za pomocą właściwości (zmiennych) i metod (funkcji) fragment świata rzeczywistego. Właściwości zawierają informacje o stanie tego fragmentu, metody pozwalają na kontrolę i zmianę tego stanu. Obiekt to wystąpienie (konkretna instancja) danej klasy.

Dobrym zwyczajem jest dzielenie kodu źródłowego na klasy ze względu na funkcje. Przykładowo 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 elementami wybranej klasy (C#).

W **programowaniu opartym na prototypach** nie istnieje pojęcie klasy. Nowe obiekty tworzy się na bazie istniejącego już obiektu (prototypu), po którym dziedziczone są pola i metody (JavaScript).

## 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 zbioru elementów, jest **abstrakcją**. Analizując problem, skupiamy się na ogólnych cechach tego zbioru elementów, a nie na właściwościach poszczególnych elementów.

### Hermetyzacja

**Hermetyzacja**, inaczej **enkapsulacja**, to ukrywanie pewnych właściwości (danych składowych) lub metod obiektów określonej klasy. Dzięki temu są one dostępne tylko w obrębie klasy.

## Dziedziczenie

Mechanizm dziedziczenia pozwala na tworzenie klas na podstawie innych wcześniej utworzonych klas. Nowa klasa ma 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 metody zdefiniowane jako *private*.

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

## Polimorfizm

Słowo **polimorfizm** oznacza wielopostaciowość. W programowaniu obiektowym polimorfizm to możliwość stworzenia obiektu, który ma 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.

### 1.1.4. Skryptowe języki programowania

**Skrypt** to napisany w języku skryptowym program, 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. Bywają również osadzane w programach w celu zautomatyzowania powtarzających się czynności. Są używane do tworzenia dynamicznych stron internetowych. Stosowane w grach komputerowych służą do sterowania przebiegiem gry.

Języki skryptowe mogą być wykorzystywane do pisania zaawansowanych aplikacji, ale najczęściej są używane do szybkiego tworzenia niewielkich skryptów pozwalających na dynamiczne wyświetlanie stron internetowych lub zapamiętywanie i przetwarzanie wprowadzonych danych.

Języki skryptowe często stanowią dodatkowe narzędzie różnego rodzaju oprogramowania.

Do popularnych języków skryptowych należą:

- JavaScript,
- Python,
- PHP.

## JavaScript

**JavaScript** to skryptowy język programowania stosowany do obsługi stron internetowych. Skrypty działają po stronie użytkownika i najczęściej realizują interakcję

z użytkownikiem z wykorzystaniem obsługi zdarzeń, tworzą efekty animacji na stronie internetowej, obsługują walidację formularzy. Wszystkie popularne przeglądarki internetowe implementują JavaScript i obsługują opracowany przez W3C jednolity model obiektowy reprezentujący drzewo dokumentu (DOM). Umożliwiają również tworzenie plików cookie, wyświetlanie okien dialogowych, manipulowanie oknami przeglądarki.

## Python

**Python** jest językiem programowania wysokiego poziomu. Wspiera różne paradigmaty programowania. Często jest używany jako język skryptowy. Z wykorzystaniem odpowiedniego framework'a (na przykład Flask, Django) może być stosowany do tworzenia aplikacji internetowych.

## PHP

**PHP** to skryptowy język programowania stosowany w aplikacjach internetowych. Służy do tworzenia skryptów po stronie serwera, ale może być również używany do tworzenia aplikacji z interfejsem graficznym i pisania skryptów wykonywanych z wiersza poleceń.

### 1.1.5. Biblioteki

W tworzonych programach często powtarzane są te same czynności, na przykład wyświetlanie na ekranie, wprowadzanie danych z klawiatury, otwieranie pliku do edycji. Wiele języków programowania dysponuje modułami, które zawierają definicje takich czynności i mogą zostać wykorzystane przez programistę. Są to tak zwane **biblioteki**. Programista może również tworzyć własne biblioteki.

Biblioteki zawierają definicje funkcji najczęściej używanych w programie. Biblioteki rozszerzają zakres operacji dostępnych w danym języku i ułatwiają pisanie programów. Większość języków programowania udostępnia biblioteki standardowe, które zawierają definicje typowych operacji wykonywanych w programach. Należą do nich:

- operacje na ciągach tekstowych,
- operacje na typach danych i funkcje do zarządzania nimi,
- obsługa wejścia-wyjścia,
- obsługa plików,
- obsługa wielowątkowości,
- zarządzanie pamięcią.

Biblioteki mogą być statyczne lub dynamiczne. Biblioteki statyczne są dołączane do programu za pomocą dyrektywy preprocessora na etapie konsolidacji. Biblioteki dynamiczne są dołączane do programu dopiero w czasie jego uruchamiania.



## 1.2. Proste 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 zwracać zawsze 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 nie tylko przetwarzają dane, lecz również je weryfikują.

W rzeczywistości tak jak nie każdy problem można rozwiązać, tak nie każdą metodę rozwiązania problemu można zapisać przy użyciu algorytmu. Aby problem mógł być rozwiązyany za pomocą komputera, musi zostać zapisany w postaci algorytmu. Wynika to z tego, że komputer potrafi rozwiązywać tylko problemy, dla których rozwiązanie zostanie zdefiniowane w postaci jednoznacznych kroków, czyli algorytmu. Jeżeli nie można zdefiniować rozwiązania 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 (tabela 1.1).

#### Schemat blokowy

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

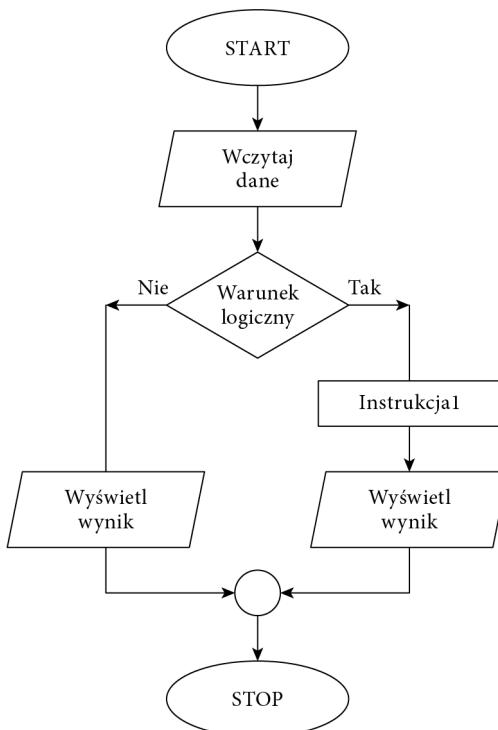


**Tabela 1.1.** Symbole wykorzystywane do tworzenia schematów blokowych

Symbol	Opis
	Początek algorytmu, start programu. Od tego miejsca rozpoczyna się wykonywanie operacji.
	Koniec algorytmu, zakończenie programu. W tym miejscu następuje zakończenie wykonywania operacji.
	Połączenie między blokami. Wskazuje kolejność wykonywania operacji.
	Wykonanie operacji, blok obliczeniowy. Wewnątrz tego symbolu znajdują się operacje do wykonania.
	Wprowadzanie danych. Wewnątrz tego symbolu określamy dane wejściowe, które muszą zostać wczytane.
	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, na przykład gdyby linie łączące na schemacie miały się krzyżować.
	Łącznik międzystronicowy. Służy do oznaczenia miejsc łączenia schematu, gdy nie mieści się on na jednej stronie.

## Przykład 1.2

Schemat blokowy (rysunek 1.4):



Rysunek 1.4. Przykład schematu blokowego

### 1.2.2. Przykłady algorytmów

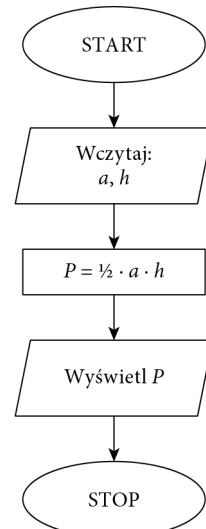
#### Obliczanie pola trójkąta

##### Przykład 1.3

Algorytm obliczania pola trójkąta przedstawiony w postaci schematu blokowego (rysunek 1.5).

#### 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



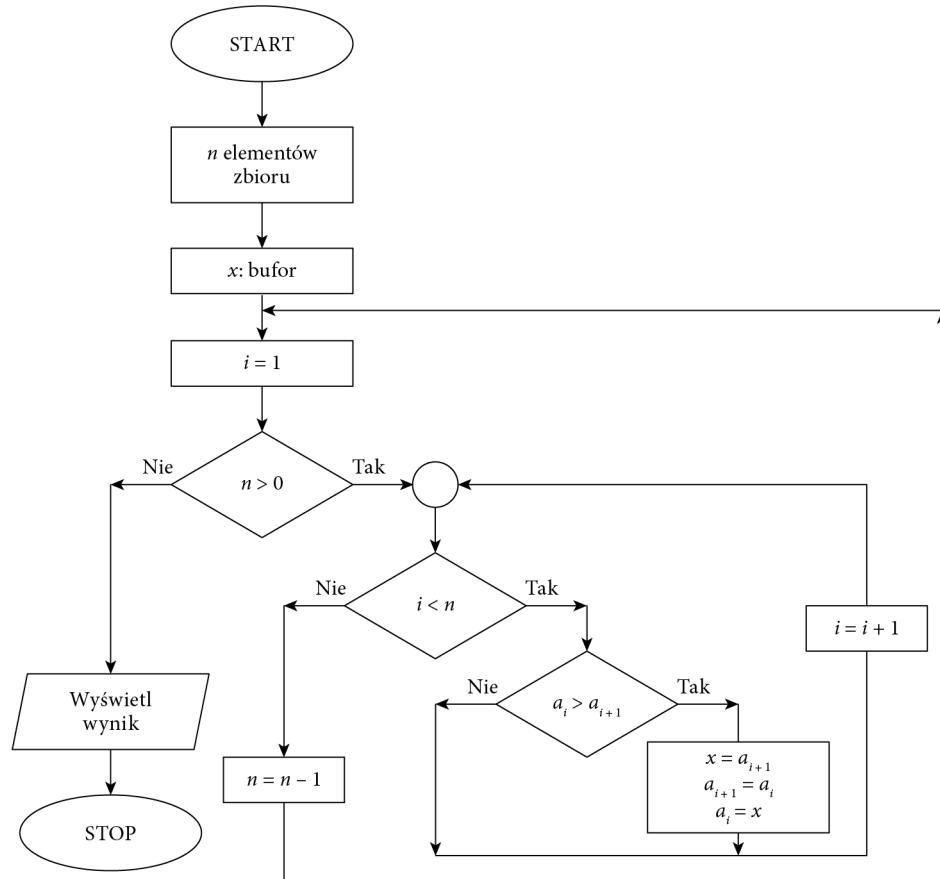
Rysunek 1.5.

Algorytm w postaci schematu blokowego

ono na porównywaniu dwóch sąsiednich elementów i zamianie ich miejscami, gdy są ustawione w nieprawidłowej kolejności. Sortowanie kończy się, gdy przy kolejnym przejściu nie ma żadnej zmiany kolejności elementów.

### Przykład 1.4

Sortowanie bąbelkowe — schemat blokowy (rysunek 1.6):



**Rysunek 1.6.** Schemat blokowy sortowania bąbelkowego

### Znajdowanie najmniejszego lub największego elementu w zbiorze

Sposób działania algorytmu szybkiego wyszukiwania elementu w zbiorze zależy od tego, czy dane zostały uporządkowane, czy zostały zapisane w przypadkowej kolejności. Jeśli dane są nieuporządkowane, należy przejrzeć wszystkie elementy, aby znaleźć ten właściwy.

### Przykład 1.5

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  $i = 2, 3, \dots, n$ , 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 1.6

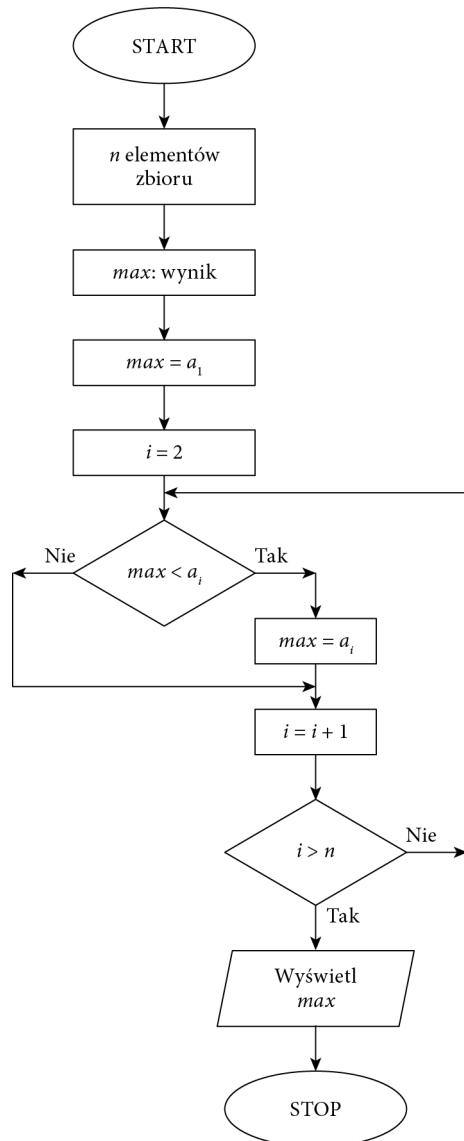
Znajdowanie największego elementu w zbiorze nieuporządkowanym — schemat blokowy (rysunek 1.7).

## ~~1.3.~~ Środowiska programistyczne

### 1.3.1. Zintegrowane środowisko programistyczne (IDE)

Zintegrowane środowisko programistyczne (ang. *Integrated Development Environment*) jest to program (lub zbiór programów) służący do tworzenia, modyfikowania i testowania oprogramowania. Umożliwia tworzenie aplikacji w określonych językach programowania. Najczęściej udostępnia narzędzia obejmujące edycję i komplikowanie kodu źródłowego, narzędzia do tworzenia interfejsu graficznego oraz narzędzia do testowania tworzonego oprogramowania (debugger). Główną zaletą korzystania ze środowiska IDE jest przyspieszenie i usprawnienie pracy nad programem.

Popularnymi środowiskami IDE stosowanymi przy programowaniu są: Microsoft Visual



Rysunek 1.7. Schemat blokowy znajdowania największego elementu w zbiorze nieuporządkowanym

Studio, Code::Blocks, Eclipse, NetBeans, Builder IntelliJ. 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.

## Visual Studio

**Visual Studio** to środowisko programistyczne firmy Microsoft. Zawiera między innymi kompilator języka C# i C++. Środowisko to umożliwia tworzenie oprogramowania dla mobilnych systemów Windows Phone, iOS oraz Android. Oferuje wiele dodatkowych narzędzi ułatwiających pracę z kodem, w tym bardzo rozbudowany debugger, oraz rozszerzone opcje pisania testów jednostkowych.

Wersja Community została stworzona z myślą o studentach i indywidualnych użytkownikach i jest dostępna za darmo. Pakiet można pobrać ze strony <https://visualstudio.microsoft.com/pl/downloads/>. Wersja ta nie zawiera edytora zasobów (brak możliwości projektowania obrazów, ikon itp.).

## Eclipse

**Eclipse** to platforma programistyczna otwartego oprogramowania stworzona przez firmę IBM przeznaczona do tworzenia i testowania aplikacji.

W skład platformy, oprócz środowiska programistycznego, wchodzą między innymi narzędzia do budowania usług i aplikacji sieciowych (Web Tools Platform Project), do rozwijania aplikacji w C/C++ (C/C++ Development Tooling), narzędzie do raportowania (Business Intelligence and Reporting Tools), generator kodu (Eclipse Modeling Framework) oraz narzędzie do tworzenia graficznych interfejsów użytkownika (Graphical Editing Framework). Program wymaga zainstalowanych w systemie bibliotek Java Runtime Environment. Dostępne są implementacje różnych języków programowania, między innymi w Javie, C/C++ i PHP. Oferuje narzędzia do współpracy z serwerami WWW oraz serwerami baz danych. Może pracować z systemami Windows, Linux i macOS.

## NetBeans

**NetBeans** to platforma otwartego oprogramowania, która dostarcza wiele narzędzi programistycznych służących do tworzenia aplikacji, w tym również usług sieciowych (Web Services) i aplikacji na urządzenia mobilne. Przeznaczona jest głównie dla języka Java, ale umożliwia komplikowanie programów napisanych w innych językach.

W skład platformy wchodzą między innymi NetBeans IDE oraz NetBeans Platform.

**NetBeans IDE** to zintegrowane środowisko programistyczne (IDE) stworzone dla języka Java. Pozwala na tworzenie aplikacji w języku Java oraz aplikacji mobilnych i usług sieciowych.

**NetBeans Platform** to platforma, która dostarcza gotowe narzędzia (bazę danych, okna, menu, zarządzanie i przechowywanie konfiguracji) umożliwiające tworzenie aplikacji.

## 1.3.2. Framework — platforma programistyczna

Do tworzenia aplikacji internetowych często wykorzystuje się platformy programistyczne, które definiują strukturę aplikacji, określają mechanizm ich działania oraz dostarczają biblioteki umożliwiające wykonywanie określonych zadań.

Framework ma zdefiniowaną podstawową strukturę aplikacji, czyli elementy, które pozostają niezmienne we wszystkich utworzonych za jego pomocą aplikacjach. Programista tworzy aplikację, dostosowując poszczególne komponenty do wymagań realizowanego projektu. Ma on również możliwość przygotowania nowych elementów niezbędnych w projektowanej strukturze aplikacji.

Typowe cechy frameworka to:

- *Odwrocenie sterowania* — przepływ sterowania jest narzucony przez framework, a nie przez programistę.
- *Domyślne zachowanie* — framework powinien mieć 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 napisania mniejszej ilości kodu. Aplikacja zbudowana w ten sposób ma dobrą wewnętrzną strukturę, jest właściwie 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.

Do popularnych frameworków należą:

- Angular,
- React,
- .NET core,
- Django,
- Zend Framework.

### Angular

Angular to wspierany i rozwijany przez firmę Google otwarty framework przeznaczony do tworzenia aplikacji jednostronnicowych w języku JavaScript. Jego zadaniem jest wdrażanie do aplikacji internetowych wzorca projektowego MVC (ang. *Model-View-Controller*). Zawiera własny kompilator HTML, wbudowany system szablonów i wiele funkcji ułatwiających tworzenie strony (animacje, filtrowanie, elementy interfejsu użytkownika).

## React

**React** to otwarty framework przeznaczony do tworzenia interfejsów graficznych aplikacji jednostronicowych i mobilnych w języku JavaScript. Interfejs użytkownika został oparty na komponentach. Każdy komponent jest funkcją JavaScript zdefiniowaną przez użytkownika. React został wyposażony w język JSX, który jest nakładką na JavaScript i rozszerza jego funkcjonalność.

## .NET core

**.NET core** to wieloplatformowy i otwarty framework przeznaczony do tworzenia aplikacji internetowych z użyciem wzorca MVC. Aplikacje mogą być tworzone za pomocą języka C#, F#, Visual Basic. Framework może działać w systemach Windows, Linux i macOS oraz na urządzeniach mobilnych korzystających z Xamarin. Może być używany do tworzenia aplikacji opartych na chmurze. Taka aplikacja działa w oparciu o mikroserwisy (podprogramy), a jej architektura jest zorientowana na usługi.

## Django

**Django** to otwarty framework przeznaczony do tworzenia aplikacji internetowych napisany w języku Python. Umożliwia przygotowanie aplikacji w oparciu o wzorzec MTV (ang. *Model-Template-View*) pokrewny wzorcowi MVC. Ma prosty system szablonów, automatycznie tworzony panel administracyjny oraz własny serwer do testowania aplikacji.

## 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 na przykład wysyłanie e-maili, komunikację z innymi aplikacjami internetowymi itp. Oferuje również implementację modelu MVC, który może zostać wykorzystany do utworzenia struktury aplikacji.

# 1.4. Pytania i zadania

## 1.4.1. Pytania

1. Co to jest paradymat programowania?
2. Wymień zasady programowania strukturalnego.
3. Co to jest typ prosty?
4. Wymień podstawowe cechy języków obiektowych.
5. Na czym polega *polimorfizm*?
6. Wymień podstawowe cechy języków skryptowych.

7. Co to jest *algorytm*?
8. Wymień podstawowe cechy algorytmów.
9. Wymień i omów narzędzia wspomagające tworzenie programów.

## 1.4.2. Zadania

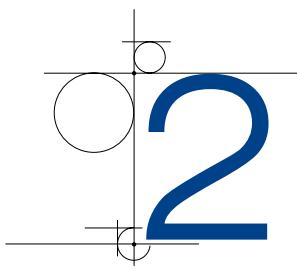
### Zadanie 1.

Utwórz algorytm w postaci listy kroków, schematu blokowego i drzewa algorytmu dla równania liniowego  $a \times x + b = 0$ .

### Zadanie 2.

Utwórz algorytm wyszukiwania najmniejszego elementu w nieuporządkowanym zbiorze danych.





# 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,
- 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,
- niższe koszty wytworzenia, uruchomienia i utrzymania aplikacji internetowej w porównaniu z innymi rozwiązaniami; większość technologii stosowanych do tworzenia aplikacji internetowych jest bezpłatna.



## 2.2. Wzorce projektowe

### 2.2.1. Wprowadzenie

Wzorce projektowe to sprawdzone w praktyce uniwersalne rozwiązania problemów często pojawiających się podczas projektowania aplikacji. Wzorce projektowe tworzone są najczęściej na bazie programowania obiektowego. Pokazują powiązania i zależności pomiędzy klasami oraz obiektami i ułatwiają tworzenie oraz modyfikację kodu źródłowego. Wprowadzają standaryzację kodu oraz zwiększą 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 obejmować:

- 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 i efektów ubocznych, jakie występują podczas używania 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 rozwiązaniami, które zostały zaimplementowane w wybranym języku obiektowym. Mogą przyspieszyć proces projektowania aplikacji przez wykorzystanie wypróbowanych rozwiązań dla sformułowanego problemu.

### 2.2.2. 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 trzy moduły:

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

## Model

**Model** zapewnia zestandardyzowany 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 (na przykład z bazy lub pliku) i nie jest to istotne pod kątem 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 między innymi 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. Wzorzec ten 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 wykorzystuje adaptację MVC zbudowaną na bazie technologii ASP.NET w postaci platformy ASP.NET MVC, która umożliwia tworzenie 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.

## 2.3. Serwery aplikacji internetowych

Serwery WWW (ang. *web server*) to programy, których zadaniem jest obsługa żądań protokołu komunikacyjnego HTTP. Przeważnie serwery WWW instalowane są na przeznaczonych do tego komputerach odgrywających rolę serwerów internetowych. Można również instalować serwer WWW na stacji roboczej (lokalnie) w celu na przykład testowania tworzonych aplikacji. Aby pobrać określona stronę internetową, klient łączy się z serwerem stron WWW za pośrednictwem przeglądarki internetowej. Do udostępniania 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 zadania serwera WWW to:

- przyjęcie połączenia TCP,
- dekodowanie żądań protokołu HTML,
- realizacja żądania (przesłanie zasobu, wykonanie programu),
- obudowanie dokumentu nagłówkami HTTP,
- wysłanie danych do klienta.

### 2.3.1. Serwer Apache

Apache jest najpopularniejszym i bezpłatnym serwerem WWW dostępnym dla wielu systemów operacyjnych (Linux, Windows, macOS). 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.

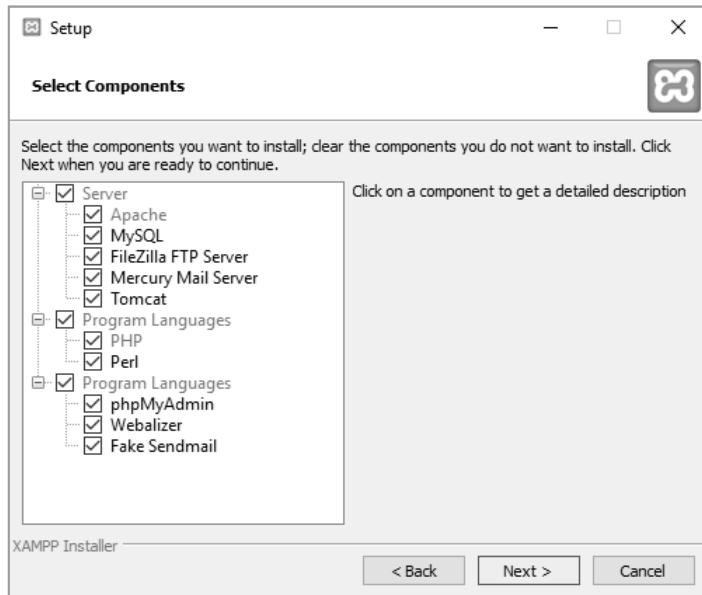
Serwer Apache jest podzielony na wiele modułów. Podczas instalowania serwera można wybrać tylko potrzebne moduły lub zainstalować wszystkie.

## 2.4. Pakiet XAMPP

Serwer stron internetowych Apache z serwerem baz danych MySQL lub MariaDB oraz językiem PHP 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**. Pakiet ten może być wykorzystywany do testowania działania aplikacji na lokalnym komputerze.

Poza serwerem aplikacji internetowych Apache, serwerem baz danych MySQL i językiem skryptowym PHP w pakiecie XAMPP jest dostępny język Perl oraz zestaw narzędzi, między innymi phpMyAdmin (rysunek 2.1).



**Rysunek 2.1.** Komponenty dostępne w pakiecie XAMPP

Plik instalacyjny w wersji dla systemów Linux oraz Windows można pobrać ze strony <http://www.apachefriends.org/en/xampp.html>.

## 2.4.1. Instalacja XAMPP 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 (na przykład [xampp-linux-x64-7.3.9-0-installer.run](#)) należy go uruchomić. W tym celu trzeba wpisać polecenia:

```
chmod +x xampp-linux-x64-7.3.9-0-installer.run
./xampp-linux-x64-7.3.9-0-installer.run
```

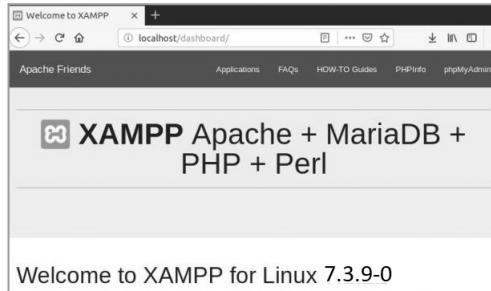
W wyniku wykonania podanych poleceń nastąpi uruchomienie pliku instalacyjnego XAMPP. W pierwszym oknie instalatora trzeba wybrać komponenty do zainstalowania. W kolejnych należy kliknąć przycisk **Next**, aż pakiet zostanie zainstalowany. Po tych czynnościach pojawi się okno z panelem sterującym (rysunek 2.2). Należy wybrać zakładkę **Manage Servers**, aby sprawdzić, czy serwer Apache został automatycznie uruchomiony.



**Rysunek 2.2.** Panel kontrolny pakietu XAMPP

By uruchomić bazę MySQL, trzeba wybrać opcję *MySQL Database* i kliknąć przycisk *Start*. Po tych czynnościach pakiet jest przygotowany do pracy.

Poprawność instalacji pakietu i uruchomienia serwerów można sprawdzić, wpisując w dowolną przeglądarkę internetową adres <http://localhost/dashboard/>. Nastąpi przekierowanie na serwer i pojawi się widok podobny do pokazanego na rysunku 2.3, co oznacza, że serwer WWW działa prawidłowo.



**Rysunek 2.3.** Test działania pakietu XAMPP

W pakiecie XAMPP głównym katalogiem serwera WWW jest </opt/lampp/htdocs>. W tym katalogu powinny być umieszczane tworzone skrypty 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 należy zdefiniować oddzielnie hasła dostępu do:

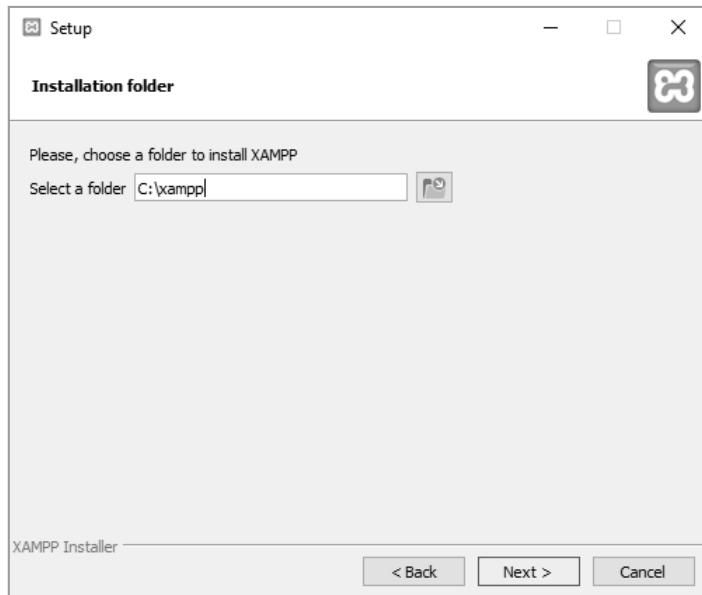
- panelu konfiguracyjnego pakietu XAMPP (dostępny pod adresem <http://opt/lampp/>, konto użytkownika to *lampp*),
- oprogramowania phpMyAdmin (zarządzanie bazą danych przez serwer WWW),
- konta administratora bazy danych MySQL,
- serwera FTP.

Po tych działaniach elementy składowe XAMPP będą w pełni zabezpieczone.

## 2.4.2. Instalacja XAMPP w systemie Windows

Przed zainstalowaniem pakietu XAMPP należy wyłączyć program antywirusowy i sprawdzić ustawienia firewalla. Po pobraniu pliku instalacyjnego (na przykład [xampp-windows-x64-7.3.9-OVC15-installer.run](#)) należy go uruchomić.

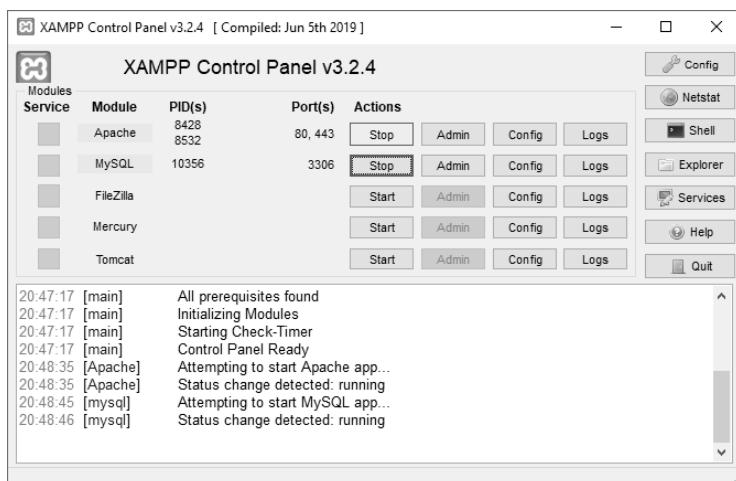
Po uruchomieniu pliku instalacyjnego można wybrać serwery, języki i narzędzia, które będą instalowane. Następnie należy określić folder, w którym pakiet zostanie zainstalowany. Domyślnie jest to folder *xampp* na dysku C: (rysunek 2.4).



**Rysunek 2.4.** Folder instalacyjny pakietu XAMPP

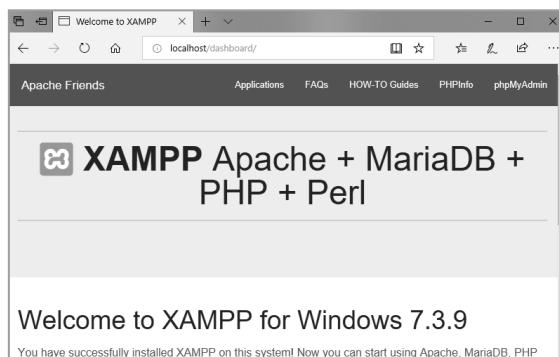
Po wskazaniu folderu docelowego nastąpi wypakowanie plików instalacyjnych i zainstalowanie pakietu. Po zakończeniu instalowania pojawi się pytanie, czy chcemy uruchomić panel kontrolny XAMPP Control Panel, którego zadaniem jest skonfigurowanie zainstalowanego oprogramowania. W panelu tym dostępne są (rysunek 2.5):

- serwer WWW Apache z interpreterem skryptów w PHP i Perl,
- serwer bazy danych MySQL,

**Rysunek 2.5.** Panel kontrolny XAMPP Control Panel

- serwer FTP FileZilla,
- serwer poczty elektronicznej Mercury,
- środowisko serwera dla Javy (Java Server Pages) — Java Tomcat.

W celu przetestowania poprawności działania serwera należy w dowolnej przeglądarce wpisać adres <http://localhost/>. Jeżeli pojawi się widok podobny do pokazanego na rysunku 2.6, oznacza to, że serwer WWW działa prawidłowo.

**Rysunek 2.6.** Test działania pakietu XAMPP

W celu przetestowania działania skryptów na serwerze trzeba odszukać folder `htdocs`. Znajduje się on w folderze instalacyjnym pakietu XAMPP. Standardowo jest to folder `C:\xampp\htdocs`. To katalog główny serwera WWW. W tym folderze będą umieszczone tworzone skrypty oraz pliki HTML. Jeżeli XAMPP został zainstalowany w innej lokalizacji, trzeba znaleźć odpowiednią ścieżkę do katalogu `htdocs`.



## 2.5. Pytania i zadania

### 2.5.1. Pytania

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 spełnia logika prezentacji w architekturze 4-warstwowej?
6. Z jakich modułów składa się wzorzec projektowy MVC?
7. Wymień podstawowe funkcje realizowane przez serwer WWW.
8. Wymień popularne serwery WWW.
9. Omów podstawowe cechy serwera Apache.
10. Omów podstawowe cechy serwera IIS.



# 3

# Język 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 sposób reprezentacji dokumentów XML i HTML 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. Za ich pomocą możliwy jest dostęp do struktury dokumentu oraz możliwa jest jego modyfikacja. Standard W3C definiuje interfejsy DOM tylko dla języków JavaScript i Java.

**SVG** (ang. *Scalable Vector Graphics*) to uniwersalny format grafiki wektorowej (statycznej i dynamicznej) stworzony 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 (prostokątów, elips, krzywych) można opisywać efekty specjalne, maski przezroczystości oraz sposób animacji elementów.

**Języki skryptowe** 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

### 3.2.1. Wprowadzenie

Utworzenie kodu kontrolującego zachowanie strony oraz określającego interakcje z użytkownikiem jest kolejnym, po napisaniu kodu HTML i zdefiniowaniu arkuszy CSS,

etapem budowy strony internetowej. Do tego celu stosuje się języki skryptowe. Najpopularniejszym z nich jest JavaScript. Odgrywa on bardzo dużą rolę w projektowaniu interaktywnych stron internetowych — umożliwia nie tylko prostą manipulację danymi, ale i dynamiczne modelowanie struktury strony, obsługę rozszerzeń multimedialnych oraz tworzenie odrębnych aplikacji.

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

Przy użyciu odpowiednich bibliotek języka JavaScript (na przykład jQuery) można tworzyć zaawansowane aplikacje sieciowe o atrakcyjnych interfejsach.

### 3.2.2. Opis języka

JavaScript jest obiektowym skryptowym językiem programowania nawiązującym do języka C. Jest językiem interpretowanym, co oznacza, że efekty jego użycia można zobaczyć bez konieczności kompilowania kodu. 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:

- modyfikowanie wyglądu bieżącego dokumentu,
- wyświetlanie prostych okien dialogowych,
- kontrola poprawności wypełnienia formularza,
- manipulowanie informacją dotyczącą daty i czasu,
- lokalne generowanie dokumentów HTML.

### 3.2.3. JavaScript w HTML

Kod źródłowy napisany w języku JavaScript może być umieszczony wewnętrz doku-  
mentu 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>.

#### Przykład 3.1

Skrypt umieszczony w dokumencie HTML:

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">  
<title>JavaScript - Przykładowy program</title>  
</head>  
<body>  
<script>  
document.write("Napisz program w języku JavaScript");  
</script>  
</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 mieć 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 instrukcja dołączana w postaci znacznika *<script>* i atrybutu *src*. Znacznik ten może zostać umieszczony w sekcji *<head>* lub w sekcji *<body>*. To rozwiązanie, podobnie jak w przypadku zewnętrznych arkuszy CSS, ułatwia wprowadzanie zmian w kodzie strony bez ingerencji w pliki HTML.

### Przykład 3.2

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

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>Przykładowy program</title>  
<script src="skrypt.js">  
</script>  
</head>  
<body>  
<p>Skrypt znajduje się w pliku "skrypt.js".</p>  
</body>  
</html>
```

Zawartość pliku *skrypt.js*:

```
document.write("Napisz program w języku JavaScript");
```

### 3.2.4. Instrukcja document.write

Instrukcja `document.write()` zapisuje ciąg tekstowy w dokumencie HTML i najczęściej jest używana do testowania.

`document` jest to obiekt JavaScript, który reprezentuje aktualnie wyświetlaną stronę, natomiast `write()` to jego metoda, czyli funkcja wykonująca określone działania na obiekcie — w tym wypadku wypisuje ona tekst.

Tekst umieszczamy w nawiasach, to argument wywołania metody.

Ogólny zapis: `obiekt.metoda(argumenty metody);`.

#### Znaczniki HTML w `document.write`

Wewnątrz instrukcji `document.write()` można używać znaczników HTML. Należy je tylko w odpowiednich miejscach otwierać i zamykać.

#### Przykład 3.3

```
document.write("<h1>JavaScript</h1>");
```

#### Przykład 3.4

```
document.write("<a href='spis.html'>Spis treści</a>");
```

Metoda `write()` w podanych przykładach przyjmuje jako argument łańcuch znakowy — dlatego jest on umieszczony w cudzysłowie. Można jako argument podawać również liczbę całkowitą lub zmiennoprzecinkową.

#### Przykład 3.5

```
document.write(201);
```

Argumentem metody `write()` może być ciąg powstały z połączenia tekstu i wartości zmiennych.

#### Przykład 3.6

```
var i = 30;  
document.write("Zmienna i ma wartość " + i + "<br>");
```

## 3.3. Składnia języka JavaScript

Składnia języka jest ściśle określonym zbiorem reguł, których należy przestrzegać. W języku JavaScript instrukcje zawierają wyrażenia, słowa kluczowe i komentarze. W wyrażeniach występują zmienne, literaly i operatory. Do elementów języka należy również zbiór predefiniowanych obiektów i funkcji (na przykład tablica, data, funkcje matematyczne).

### Przykład 3.7

```
var a, b, c;      // deklaracja zmiennych
a = 4; b = 9;    // przypisanie wartości
c = a + b;       // wyrażenie
```

### 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. Jeśli kilka instrukcji zostanie zapisanych w jednym wierszu, muszą być one oddzielone średnikiem.

Język JavaScript ignoruje powtarzające się spacje. Spacje można umieszczać w dowolnych miejscach, aby zwiększyć czytelność skryptu. Wskazane jest wstawianie spacji wokół operatorów `=`, `+`, `-`, `*`, `/`, na przykład:

```
var c = a + b;
```

Instrukcje mogą być grupowane w bloki za pomocą nawiasów klamrowych `{ }`. Instrukcje zgrupowane w bloki są wykonywane razem jedna po drugiej.

### 3.3.2. Wielkość liter

W języku JavaScript rozróżniana jest wielkość liter oraz wspierany jest standard znaków Unicode. W kodzie skryptu przy zapisywaniu nazw stosuje się następujące zasady:

- Słowa kluczowe pisane są małymi literami, na przykład `for`.
- Nazwy obiektów wbudowanych pisane są od dużej litery, a pozostałe litery są małe, na przykład `Date`.
- Nazwy obiektów DOM zapisywane są małymi literami, ale w nazwach metod tych obiektów dopuszczane są małe i duże litery, na przykład `toLowerCase`.
- Białe znaki, typu spacja, znak tabulacji itp., są nieistotne.

### 3.3.3. Słowa kluczowe

Słowa kluczowe określają czynności, które powinny zostać wykonane przez instrukcję zawierającą te słowa. Słowa kluczowe są słowami zastrzeżonymi i nie mogą być nazwami zmiennych.

### 3.3.4. Komentarze

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



### Przykład 3.8

```
// 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>.

Najczęściej stosowane są komentarze jednowierszowe. Natomiast komentarze blokowe są wykorzystywane w dokumentacjach.

### 3.3.5. Zmienne

Zmienne służą do przechowywania danych i wyników w celu dalszego ich wykorzystywania. W języku JavaScript zmienne deklaruje się za pomocą słowa kluczowego var poprzedzającego nazwę zmiennej. Można również stosować nowy sposób deklaracji zmiennej za pomocą słowa kluczowego let. Nazwa zmiennej może zawierać litery, cyfry oraz znak podkreślenia, natomiast nie może zaczynać się od cyfry. Wielkość liter używanych w nazwach ma znaczenie. 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. Dobrą praktyką jest deklarowanie wszystkich zmiennych na początku skryptu.

Po deklaracji zmienna nie ma wartości. Do przypisania jej wartości służy operator =.

### Przykład 3.9

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

### Przykład 3.10

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>Deklaracja zmiennych</title>  
<script>  
var x = "JavaScript - ";  
var x = "JavaScript - ";
```

```
var y = "Zmienna";
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 zmienia swój typ w zależności od typu przypisanej wartości (rysunek 3.1).

JavaScript - Zmienna  
JavaScript - 3.01

**Rysunek 3.1.** Rezultat zmiany typu zmiennej y

Jeżeli zadeklarowanej zmiennej nie zostanie przypisana wartość, to będzie ona miała wartość undefined.

### Ćwiczenie 3.1

Utwórz zmienne: nazwisko, imię, wiek, klasa. Przypisz tym zmiennym przykładowe dane. Wyświetl w przeglądarce internetowej informację o treści:

Uczeń naszej szkoły nazywa się (*tu powinno pojawić się: imię nazwisko*).

Uczęszcza do klasy (*tu powinna pojawić się klasa*).

Ma (*tu powinien pojawić się wiek*) lat.

### Rozwiązanie

```
<!DOCTYPE html>

<html>
<head>
<meta charset="UTF-8">
<title>Deklaracja zmiennych</title>
</head>
<body>
<script>
```

```
var nazwisko = "Witkowski";
var imie = "Maciej";
var wiek = "15";
var klasa = "2a";

document.write("Uczeń naszej szkoły nazywa się " + nazwisko +
" " + imie + ".<br>");
document.write("Uczęszcza do klasy " + klasa + ".<br>");
document.write("Ma " + wiek + " lat." + "<br>");
</script>
</body>
</html>
```

### 3.3.6. Literały

Stałe wartości w języku JavaScript są nazywane **literałami**. Literałem może być liczba lub ciąg tekstowy.

Liczby są zapisywane w postaci liczb całkowitych lub w postaci dziesiętnej. Do oddzielenia części dziesiętnej służy znak kropki, na przykład 157.23.

Ciągi znakowe zapisywane są w podwójnym lub pojedynczym cudzysłowie, na przykład "Programowanie w JavaScript".

### 3.3.7. Identyfikatory

**Identyfikatory** to nazwy zmiennych, funkcji, etykiet oraz słów kluczowych. Definiowanym zmiennym, obiektom, funkcjom można nadawać nazwy pisane w dowolny sposób.

Nazwy powinny być tworzone według następujących zasad:

- Oprócz liter można w nich stosować cyfry, znak podkreślenia i znak dolara.
- Nazwa może zaczynać się literą, znakiem podkreślenia lub znakiem dolara.
- Pierwszym znakiem nazwy nie może być liczba.
- W nazwach jest rozróżniana wielkość liter.
- Nazwą nie może być słowo zarezerowane.

Do łączenia słów w nazwie nie można używać jako łącznika znaku -. Łącznikiem może być znak podkreślenia, na przykład Data\_urodzenia, lub zbiór małych i dużych liter, na przykład KodPoczty, kolorOczu. W języku JavaScript przyjęte jest zapisywanie nazwy od małej litery, na przykład dobraKawa.

### 3.3.8. Skalarne typy danych

Język JavaScript jest językiem słabo typowanym, co oznacza, że w momencie deklarowania zmiennej nie trzeba określić jej typu. Dopuszczane są następujące typy danych:

- typ liczbowy `number`,
- typ łańcuchowy `string`,
- typ logiczny `boolean`,
- typ `null`,
- typ `undefined`.

#### Typ liczbowy `number`

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 (na przykład `042`). Jeżeli zostanie poprzedzona ciągiem znaków `0x` lub `0X`, to jest traktowana jako wartość szesnastkowa, inaczej heksadecymalna (na przykład `0x23A`). Wartości liczbowe mogą być zapisywane w notacji wykładniczej (na przykład `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 3.11

```
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, na przykład `"Anna"`, `'Nowak'`.

#### Przykład 3.12

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

#### 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 3.13

```
var k = true;
```



## Typ null

Jest to typ specjalny określający wartość pustą (`null`). Nie przechowuje żadnej wartości. W języku JavaScript ten typ danych jest obiektem.

## Typ undefined

Jest to typ zawierający zmienne, którym nie została nadana żadna wartość. Wartość oraz typ zmiennej są nieznane, na przykład `var x;`.

Zmiennej `x` nie przypisano wartości. Ma typ i wartość `undefined`.

## 3.3.9. Złożone typy danych

W języku JavaScript zostały zdefiniowane również złożone typy danych. Oznacza to, że wewnętrz nich można przechowywać więcej niż jedną wartość. Takie zmienne są obiektami i są typu referencyjnego. Zmienne obiektowe nie mają przypisanej bezpośrednio wartości, tylko wskazują adres w pamięci, gdzie dane są przechowywane.

## Obiekty

Służą do reprezentacji obiektów. Wykorzystywane są obiekty wbudowane oraz udostępniane przez przeglądarkę. Właściwości obiektów są zapisywane w nawiasach klamrowych jako para `nazwa: wartość` i są oddzielane przecinkami.

### Przykład 3.14

```
var osoba = {nazwisko: "Nowak", imie: "Paweł", wiek: 18};
```

## Tablice

Służą do przechowywania wielu wartości. Tablice są zapisywane w nawiasach kwadratowych, a elementy tablicy są oddzielane przecinkami.

### Przykład 3.15

```
var liczby = [10, 23, 57, 94, 32];
```

## 3.3.10. 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 (zmnieszania). 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

### Przykład 3.16

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript - Operatory</title>
<script>
var a = 12;
var b = 5;
var c = a - b;
document.write("Wynikiem odejmowania jest ");
document.write(c);
document.write("<br>Wynikiem dodawania jest ");
document.write(a + b);
document.write("<br>Wynikiem mnożenia jest ");
document.write(a * b);
</script>
</head>
<body>
...
</body>
</html>
```

**Operator inkrementacji** powoduje zwiększenie wartości o jeden. Może występować w postaci przedrostkowej (`++x`) lub przyrostkowej (`x++`).

Operacja `x++` zwiększa wartość zmiennej po jej wykorzystaniu.

Operacja `++x` zwiększa wartość zmiennej przed jej wykorzystaniem.

Operatory `x++` i `++x` zwiększą wartość zmiennej, ale nie są równoważne.

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

### Przykład 3.17

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

### Operatory porównania

Operatory porównania porównują argumenty. Wynikiem tej operacji jest wartość logiczna `true` (prawda) lub `false` (fałsz) (tabela 3.2).

**Tabela 3.2.** Operatory porównania

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

**Przykład 3.18**

Sprawdzenie, czy liczby są równe:

```
<script>
var a = 10;
var b = '10';
if (a == b) {
    document.write("Liczby są równe.");
}
else {
    document.write("Liczby nie są równe.");
}
</script>
```

**Przykład 3.19**

Sprawdzenie, czy liczby są identyczne:

```
<script>
var a = 10;
var b = '10';
if (a === b) {
    document.write("Liczby są identyczne.");
}
else {
    document.write("Liczby nie są identyczne.");
}
</script>
```

**Operatory bitowe**

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

**Tabela 3.3.** Operatory bitowe

Operator	Działanie	Wyrażenie	Przykład (binarna reprezentacja)	Wynik (binarna reprezen- tacja)
&	iloczyn bitowy (AND)	a & b	0101 & 0001	0001
	suma bitowa (OR)	a   b	0101   0001	0101
~	negacja bitowa (NOT)	~ a	~ 0101	1010

Operator	Działanie	Wyrażenie	Przykład (binarna reprezentacja)	Wynik (binarna reprezentacja)
$\wedge$	bitowa różnica symetryczna	<code>a <math>\wedge</math> b</code>	<code>0101 <math>\wedge</math> 0001</code>	<code>0100</code>
$>>$	przesunięcie bitowe w prawo	<code>a <math>&gt;&gt;</math> n</code>	<code>0101 <math>&gt;&gt;</math> 1</code>	<code>0010</code>
$<<$	przesunięcie bitowe w lewo	<code>a <math>&lt;&lt;</math> n</code>	<code>0101 <math>&lt;&lt;</math> 1</code>	<code>1010</code>
$>>>$	przesunięcie bitowe w prawo z wypełnieniem zerami	<code>a <math>&gt;&gt;&gt;</math> n</code>	<code>0101 <math>&gt;&gt;&gt;</math> 1</code>	<code>0010</code>

## Operatory logiczne

Przy użyciu operatorów logicznych wykonuje się operacje na argumentach, które mają przypisaną wartość logiczną — `true` lub `false` (tabela 3.4).

Wynikiem iloczynu logicznego jest wartość `true` tylko wtedy, gdy obydwa argumenty mają wartość `true`.

W pozostałych przypadkach wynik ma wartość `false`.

Wynikiem sumy logicznej jest wartość `false` tylko wtedy, gdy obydwa argumenty mają wartość `false`. W pozostałych przypadkach wynik ma wartość `true`.

Negacja logiczna zmienia wartość argumentu na przeciwną.

**Tabela 3.4.** Operatory logiczne

Operator	Działanie	Przykład
<code>&amp;&amp;</code>	iloczyn logiczny (AND)	<code>a <math>\&amp;\&amp;</math> b</code>
<code>  </code>	suma logiczna (OR)	<code>a <math>\ </math> b</code>
<code>!</code>	negacja logiczna (NOT)	<code>! a</code>

## Operatory przypisania

Z pomocą operatorów przypisania można przypisać wartości argumentom znajdującym się po lewej stronie operatora. Oprócz prostej operacji przypisania pozwalają na połączanie operacji przypisania z inną operacją, na przykład dodawania.

Zapis `i += 7` oznacza w praktyce to samo co zapis `i = i + 7`. Stosowanie 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
<code>=</code>	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>



## 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 pozwala na sprawdzenie w programie warunku i w zależności od tego, czy wynik jest prawdą, czy fałszem, dalsze wykonywanie instrukcji. Do takiego sprawdzania służy instrukcja `if ... else`.

```
if (warunek) {  
    instrukcje  
} else {  
    instrukcje  
}
```

#### Przykład 3.20

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Instrukcja warunkowa</title>  
    <script>  
      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>
```



## Ćwiczenie 3.2

Zmodyfikuj kod z przykładu 3.20 w taki sposób, aby po porównaniu wartości zmien-nych wyświetlna była odpowiedź w postaci:

$a > b$ : Zmienna  $a$  jest większa od zmiennej  $b$ .

$a = b$ : Zmienna  $a$  jest równa zmiennej  $b$ .

$a < b$ : Zmienna  $a$  jest mniejsza od zmiennej  $b$ .

## Rozwiążanie

```
<!DOCTYPE html>

<html>
<head>
<meta charset="UTF-8">
<title>Deklaracja zmiennych</title>
</head>
<body>
<script>
var a = 12;
var b = 17;
if (a > b) {
    document.write("Zmienna a jest większa od zmiennej b.");
} else {
    if (a < b) {
        document.write("Zmienna a jest mniejsza od zmiennej b.");
    } else {
        document.write("Zmienna a jest równa zmiennej b.");
    }
}
</script>
</body>
</html>
```

## Ćwiczenie 3.3

Przypisz zmiennym następujące wartości:  $x = 10$ ;  $y = 17$ ;  $z = 23$ .

Porównaj wartości podanych zmiennych i wyświetl wynik porównania wszystkich zmiennych w przeglądarce internetowej według podanego wzoru:

Zmienna *y* jest większa od zmiennej *x*, ale jest mniejsza od zmiennej *z*.

Zmienna *x* jest mniejsza od zmiennej *y* i jest mniejsza od zmiennej *z*.

Zmienna *z* jest większa od zmiennej *y* i jest większa od zmiennej *x*.

### Ćwiczenie 3.4

Zdefiniuj trzy zmienne i przypisz im dowolne wartości. Wykorzystując poznane operatory, sprawdź, czy wszystkie zmienne są:

- dodatnie — jeżeli tak, wyświetl komunikat: Wszystkie zmienne są dodatnie,
- ujemne — jeżeli tak, wyświetl komunikat: Wszystkie zmienne są ujemne,
- jeżeli zmienne mają różne znaki, wyświetl komunikat: Zmienne mają różne znaki.

## 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, jaką wartość ma *wyrażenie*, i jeżeli wynikiem jest *wartość1*, wykonaj *instrukcje1* i wyjdź z bloku `switch` (polecamie `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 3.21

```
<script>  
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>
```

### Ćwiczenie 3.5

Utwórz nowy skrypt. Przypisz zmiennej `dzien` wartość bieżącego dnia tygodnia. Za pomocą instrukcji `switch` sprawdź, jaka wartość została przypisana do zmiennej, i wyświetl informację według podanego wzorca.

Wartość zmiennej `dzien`:

poniedziałek → Cały tydzień przed nami.

wtorek → Kiedy będzie wolne?

środa → Dopiero środek tygodnia.

czwartek → Już czwartek.

piątek → Wreszcie piątek.

sobota → Czas na odpoczynek.

niedziela → Jutro znowu poniedziałek.

### 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`

Pętlę typu `for` wykorzystuje się, gdy znana jest liczba wykonania pętli oraz znany jest 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.

#### Przykład 3.22

```
<script>
    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 3.23

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

W przykładzie 3.23 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 3.24

```
<script>
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 jej prawidłowego działania.

### Ćwiczenie 3.6

Wykorzystując pętlę `for`, utwórz skrypt, który wyświetli w przeglądarce internetowej ciąg liczbowy powtórzony pięć razy w sposób pokazany na rysunku 3.2.

### Rozwiązańe

```
<!DOCTYPE html>
<html>
<head>
<title>Pętla for</title>
</head>
<body>
<script>
for (var j = 1; j < 6; j++) {
    for (var i = 1; i < 6; i++) {
        document.write( i + " ");
    }
    document.write("<br>");
}
</script>
</body>
</html>
```

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

**Rysunek 3.2.**

Ciąg liczbowy dla pętli `for`

## Ćwiczenie 3.7

Wykorzystując pętlę `for`, utwórz skrypt, który wyświetli w przeglądarce internetowej ciąg liczbowy powtórzony w sposób pokazany na rysunku 3.3.

### Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Pętla for</title>
</head>
<body>
<script>
var i = 1;
var z = 1;
for (var j = 1; j < 6; j++) {
    for (; i < 6; i++) {
        document.write(i + " ");
    }
    z = z + 1;
    i = z;
    document.write("<br>");
}
</script>
</body>
</html>
```

1	2	3	4	5
2	3	4	5	
3	4	5		
4	5			
5				

**Rysunek 3.3.**

Ciąg liczbowy dla pętli for

## 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 oznacza: „Dopóki *wyrażenie warunkowe* jest prawdziwe, wykonuj instrukcje”.

### Przykład 3.25

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

### Ćwiczenie 3.8

Wykorzystując pętlę while, utwórz skrypt, który wyświetli w przeglądarce internetowej powtórzone bloki tekstu w sposób pokazany na rysunku 3.4. Wyświetlanie bloków zakończ dla  $i = 10$ .

### Rozwiążanie

```
<!DOCTYPE html>
<html>
<head>
<title>Przykład</title>
</head>
<body>
<script>
var text = "";
var i = 0;
while (i < 10) {
    text += "<br>Liczba i jest równa " + i;
    document.write(text + "<br>");
    i++;
}
</script>
</body>
</html>
```

Liczba i jest równa 0

Liczba i jest równa 0  
Liczba i jest równa 1

Liczba i jest równa 0  
Liczba i jest równa 1  
Liczba i jest równa 2

Liczba i jest równa 0  
Liczba i jest równa 1  
Liczba i jest równa 2  
Liczba i jest równa 3

Liczba i jest równa 0  
Liczba i jest równa 1  
Liczba i jest równa 2  
Liczba i jest równa 3  
Liczba i jest równa 4

Liczba i jest równa 0  
Liczba i jest równa 1  
Liczba i jest równa 2  
Liczba i jest równa 3  
Liczba i jest równa 4  
Liczba i jest równa 5

**Rysunek 3.4.**

Bloki tekstu dla pętli while

## 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 jako *wyrażenie warunkowe* jest fałszywy — ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

### Przykład 3.26

```
<script>
var i = 1;
do {
    document.write("Pętla wykonana" + i + "razy<br>");
} while (i++ <= 5);
</script>
```

## Instrukcja break

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

### Przykład 3.27

```
<script>
var i = 0, k = 5;
do {
    var 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.

## Ćwiczenie 3.9

Przypisz zmiennej `x` wartość 10. Za pomocą instrukcji `while` utwórz pętlę, która będzie w nieskończoność zwiększała wartość zmiennej `x` o 1 i wyświetlała wartości tej zmiennej. Przerwij wykonywanie pętli, gdy zmienna `x` osiągnie wartość 27.

### Rozwiążanie

```
<!DOCTYPE html>
<html>
<head>
<title>Przykład</title>
<meta charset="UTF-8">
</head>
<body>
<script>
var x = 10;
while (true) {
    document.write("Wartość zmiennej i wynosi: " + x + "<br>");
    if (x++ >= 27) break;
}
</script>
</body>
</html>
```

## Instrukcja `continue`

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

### Przykład 3.28

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

W wyniku wykonania podanego kodu zostaną wyświetlane liczby z zakresu od 0 do 30 podzielne przez 3. 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).

### Ćwiczenie 3.10

Wykorzystując instrukcje sterujące wykonaniem skryptu, utwórz skrypt, który będzie wyświetlał tabliczkę mnożenia w postaci podanej na rysunku 3.5. Wartość zmiennej  $x$  zmienia się od 1 do 10, a zmiennej  $n$  — od 1 do 9.

#### Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Tabliczka mnożenia</title>
<meta charset="UTF-8">
</head>
<body>
<script>
for (var n = 1; n <= 10; n++) {
    for (var m = 1; m < 10; m++) {
        document.write(n, " * ", m, " = ", (n * m), "<br>");
    }
}
</script>
</body>
</html>
```

1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15

**Rysunek 3.5.**  
Tabliczka mnożenia

### Ćwiczenie 3.11

Utwórz skrypt, który będzie wyświetlał liczby od 1 z interwałem równym 4. Zakończenie wyświetlania nastąpi, gdy suma wyświetlanych liczb przekroczy 100.

## 3.5. Funkcje

Funkcja w języku JavaScript to jedno lub więcej poleceń zgrupowanych w całość za pomocą nawiasów klamrowych `{ }`. Tak zdefiniowana funkcja może zawierać listę argumentów umieszczonych w nawiasach okrągłych `()` i oddzielonych przecinkami oraz może zwracać wartość. Do funkcji można odwołać się poprzez nazwę.

### 3.5.1. Definiowanie funkcji

W języku JavaScript można definiować własne funkcje. Definicja funkcji musi zawierać słowo kluczowe `function`, po którym następuje nazwa funkcji, po czym w nawiasach okrągłych powinny zostać wymienione argumenty funkcji oddzielone przecinkami. W nawiasach klamrowych jest zapisywana treść funkcji, a na jej końcu powinna zostać wstawiona instrukcja `return`, określająca zwracaną przez funkcję wartość. Ponieważ funkcja zawsze zwraca jakąś wartość, to jeżeli wartość ta nie zostanie podana w jawnym sposób (z wykorzystaniem instrukcji `return`), automatycznie zostanie zwrócona 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 3.29

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

#### Wywołanie funkcji

Wykonanie funkcji nastąpi, gdy „coś” ją wywoła. Może to być:

- zdarzenie (na przykład kliknięcie myszą przycisku),
- podanie w kodzie skryptu nazwy funkcji (wraz z jej argumentami),
- automatyczne wywołanie funkcji.

#### Przykład 3.30

```
<script>
function trzy(x) {
    for (var i = 0; i <= x; i++) {
        if ((i % 3) != 0)
            continue;
        document.write(i + "; ");
    }
    document.write("<br>");
```

```

    }
    trzy(90);
    trzy(120);
</script>

```

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

### Przykład 3.31

```

<script>
function wynik(x, y) {
    var s = x + y;
    return s;
}
var suma = wynik(19, 7) + 27;
document.write("Wynik dodawania: " + suma);
</script>

```

Zdefiniowana w przykładzie funkcja `wynik()` oblicza sumę dwóch liczb i zwraca ją jako wartość zmiennej `s` (`return s`). W skrypcie funkcja `wynik()` została wywołana z argumentami 19 i 7 w wyrażeniu, w którym obliczona zostaje wartość zmiennej `suma`.

## 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 jest automatycznie tworzona dla każdej funkcji.

### Przykład 3.32

```

function lista_arg() {
    return arguments;
}

```

W wyniku wywołania funkcji bez podania parametrów nie zostanie zwrócona żadna wartość.

```
lista_arg();
```

Rezultat:

```
[ ]
```

W wyniku wywołania funkcji z parametrami zostanie zwrocona lista wartości.

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

Rezultat:

```
[1, 4, 5, 9, "A", "koło"]
```

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

### Przykład 3.33

```
<script>  
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, na przykład:

```
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. Należy starać się ograniczać liczbę zmiennych globalnych i deklarować zmienne ze słowem kluczowym `var`.

### Przykład 3.34

```
<script>
    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 operatora `var`. Może zatem być wykorzystana w dalszej części skryptu.

### Zasięg blokowy

Zadeklarowanie zmiennej za pomocą słowa kluczowego `let` powoduje, że taka zmienna ma zasięg blokowy. Oznacza to, że jest widoczna w bloku otoczonym nawiasami klamrowymi `{ }.`

### Przykład 3.35

```
<script>
    let a = 10;
    {
        let a = 15;
        document.write("Zmienna w bloku: " + a);
    }
    document.write("Zmienna globalna: " + a);
</script>
```

### 3.5.3. Funkcje wbudowane

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

- `parseInt()`,
- `parseFloat()`,
- `isNaN()`,
- `isFinite()`,
- `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 3.36

Funkcja	Wynik
<code>parseInt("123");</code>	123
<code>parseInt("123AB", 16);</code>	74667
<code>parseInt("123", 8);</code>	83
<code>parseInt("123AB", 8);</code>	83
<code>parseInt("0377");</code>	255
<code>parseInt("0x373");</code>	883

#### Przykład 3.37

```
<!DOCTYPE html>
<html>
<head>
<title>Funkcja JS</title>
<meta charset="UTF-8">
<script>
var c1 = "12";
var c2 = "34";
var wynik = c1 + c2;
```

```

document.write("Zwykłe dodawanie ciągów: c1 + c2 = ");
document.write(wynik);
document.write("<br />");

c1 = parseInt(c1);
c2 = parseInt(c2);
wynik = c1 + c2;

document.write("Dodawanie po konwersji na liczby: c1 + c2 = ");
document.write(wynik);
document.write("<br>");

</script>
</head>
<body>
</body>
</html>

```

## parseFloat()

Funkcja działa podobnie do `parseInt()`, ale można ją zastosować także do wartości ułamkowych. Pobiera argument dowolnego typu (najczęściej tekstuowego) i zamienia go na liczbę, która może zawierać część ułamkową. Funkcja poprawnie interpretuje zapis liczby w postaci wykładniczej.

## Przykład 3.38

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

## isNaN()

Funkcja sprawdza, czy wartość podana jako parametr nie jest liczbą. Za jej pomocą można na przykład zweryfikować, czy funkcji `parseInt()` udało się zamienić podaną wartość na liczbę. Funkcja zwraca wartość `true`, gdy argument wejściowy nie jest liczbą, lub `false`, gdy argument ten jest liczbą.

## Przykład 3.39

Funkcja	Wynik
<code>isNaN(NaN)</code>	<code>true</code>
<code>isNaN(567)</code>	<code>false</code>
<code>isNaN(37.2)</code>	<code>false</code>
<code>isNaN(parseInt("zx23"))</code>	<code>true</code>

## 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, a gdy argument ten jest liczbą, zwraca true.

### Przykład 3.40

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

## alert()

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

### Przykład 3.41

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script>
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 wewnętrz 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.



## 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 (na przykład liczbę znaków łańcucha, rozmiary 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 (na przykład nazwę zmiennej, która przechowuje dany obiekt), i kropkę.

### Przykład 3.42

```
var napisz = "Witaj w szkole";
document.write(napisz.toUpperCase() + "<br>");
document.write("Długość tekstu: " + napisz.length);
```

W podanym przykładzie zmienna `napisz` przechowuje obiekt. Funkcja `toUpperCase()` jest metodą obiektu, a `napisz.length` jego właściwością. Wynikiem będzie wypisanie tekstu pokazanego na rysunku 3.6.

**WITAJ W SZKOLE**  
**Długość tekstu: 14**

**Rysunek 3.6.** Zastosowanie metody obiektu i jego właściwości

Każdy element strony internetowej jest traktowany jako obiekt związany z obiektem `document`. Obiekty języka JavaScript zawierają informacje opisujące stronę i jej środowisko.

### 3.6.1. Tworzenie obiektów

Obiekty w języku JavaScript można tworzyć:

- używając literału obiektu,
- za pomocą konstruktora obiektu i słowa kluczowego `new`.

#### Tworzenie obiektu z użyciem literału

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

Ponieważ obiekty są zmiennymi, które mogą zawierać wiele wartości, można utworzyć zmienną, która jest obiektem. Każda jej wartość będzie właściwością obiektu, zostanie zapisana jako para nazwa i wartość (*nazwa: wartość*) i umieszczona w nawiasach



klamrowych {} . Działania, jakie można wykonywać na takim obiekcie, zostaną zdefiniowane jako metody i zapisane jako definicje funkcji.

### Przykład 3.43

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

Utworzony obiekt ma 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 osoba.

Dostęp do właściwości obiektu uzyskamy po podaniu nazwy obiektu i jego właściwości oddzielonych znakiem kropki (na przykład osoba.nazwisko). W ten sam sposób można wywołać metodę (osoba.pokaz).

### Przykład 3.44

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="UTF-8">  
        <title>Obiekt osoba</title>  
    </head>  
    <body>  
        <script>  
            var osoba = {  
                nazwisko: "Nowacki",  
                imie: "Marek",  
                zawod: "informatyk",  
                pokaz: function() {  
                    document.write(this.nazwisko + ' ' + this.imie);  
                }  
            };  
            osoba.pokaz();  
        </script>  
    </body>  
</html>
```

```

pokaz: function() {
    document.write(this.nazwisko + ' ' + this.imie);
}
};

osoba.pokaz();
</script>
</body>
</html>

```

W podanym przykładzie po utworzeniu obiektu `osoba` za pomocą konstrukcji `osoba.pokaz()` została wywołana metoda, która wyświetli nazwisko i imię osoby.

Dla istniejących obiektów można deklarować nowe właściwości i metody.

### Przykład 3.45

```

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

osoba.wiek = 19;

osoba.wypisz_wiek = function() {
    document.write('Wiek: ' + this.wiek + ' lat');
}

osoba.wypisz_wiek();

```

W podanym przykładzie po utworzeniu obiektu `osoba` i zdefiniowaniu jego właściwości i metod zostały zadeklarowane nowa właściwość `wiek` oraz nowa metoda `wypisz_wiek()`. W kolejnym kroku została wywołana nowa metoda `osoba.wypisz_wiek()`, za pomocą której został wyświetlony wiek osoby.

### Ćwiczenie 3.12

Utwórz w języku JavaScript obiekt opisujący pojazd. Zdefiniuj dla niego właściwości: `marka`, `model`, `rok_produkcji`, `przebieg` oraz metodę `wyswietlDane()` do wyświetlenia marki, modelu i roku produkcji. Wywołaj metodę `wyswietlDane()`, a następnie dodaj właściwość `numer_rej` i metodę `wyswietlDetal()` do wyświetlenia marki samochodu i numeru rejestracyjnego.

## Rozwiązańie

```
<!DOCTYPE html>

<html>
<head>
<meta charset="UTF-8">
<title>Obiekt - literal</title>
</head>
<body>
<h2>Ćwiczenie - Tworzenie obiektu za pomocą literała</h2>
<script>
var pojazd = {
    marka: "Ford",
    model: "Mustang",
    rok_produkcji: "2020",
    przebieg: 15000,
    wyswietlDane: function() {
        document.write(this.marka + ' ' + this.model + ', rok produkcji: '
+ this.rok_produkcji + "<br>");
    }
};

pojazd.wyswietlDane();

pojazd.numer_rej = "WA34789";
pojazd.wyswietlDetal = function() {
    document.write('Marka: ' + this.marka + ', numer rejestracyjny: '
+ this.numer_rej);
}
pojazd.wyswietlDetal();

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

## 3.6.2. Tworzenie obiektów z użyciem konstruktora

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

### Przykład 3.46

```
function Klient(nazwisko_k, imie_k, zawod_k) {
    this.nazwisko = nazwisko_k;
    this.imie = imie_k;
    this.zawod = zawod_k;
    this.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.

### Słowo kluczowe new

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

### Przykład 3.47

```
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`.

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

*nazwa\_obiektu.nazwa\_właściwości*

lub

*nazwa\_obiektu["nazwa\_właściwości"]*

### Przykład 3.48

```
osoba1.nazwisko;
osoba1["nazwisko"];
```

## Ćwiczenie 3.13

Utwórz konstruktor obiektu Uczen z właściwościami: nazwisko, imie, wiek, klasa, sport oraz metodę wypisz(), która wyświetli właściwości: nazwisko, imie, wiek oraz sport. Na podstawie konstruktora utwórz pięć obiektów (uczniów), które będą zawierały nazwisko, imię, wiek, klasa, sport. Przy tworzeniu obiektów posłuż się zaprojektowanym konstruktorem obiektu. Dla każdego ucznia wyświetl jego nazwisko, imię, wiek oraz sport, jaki uprawia.

### Rozwiążanie

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Konstruktor</title>
</head>
<body>
<h2>Ćwiczenie - Konstruktor</h2>
<script>
function Uczen(nazwisko, imie, wiek, klasa, sport) {
    this.nazwisko_u = nazwisko;
    this.imie_u = imie;
    this.wiek_u = wiek;
    this.klasa_u = klasa;
    this.sport_u = sport;
    this.wypisz = function() {
        document.write(this.nazwisko_u + ' ' + this.imie_u + ' '
+ this.wiek_u + ' lat, sport: ' + this.sport_u + '.' + "<br>");
    }
}

var uczen1 = new Uczen("Nowak", "Paweł", 12, "5b", "tenis");
var uczen2 = new Uczen("Polak", "Anna", 13, "6a", "pływanie");
var uczen3 = new Uczen("Czaja", "Maciej", 13, "6a", "siatkówka");
var uczen4 = new Uczen("Malak", "Julia", 12, "5c", "gimnastyka");
var uczen5 = new Uczen("Wojak", "Michał", 13, "6b", "pływanie");
```

```
uczen1.wypisz();  
uczen2.wypisz();  
uczen3.wypisz();  
uczen4.wypisz();  
uczen5.wypisz();  
  
</script>  
</body>  
</html>
```

## Właściwość prototype

Nie można dodawać nowych właściwości i metod do konstruktora obiektów tak, jak dodaje się właściwości i metody do istniejącego obiektu. Nowe właściwości i metody muszą być umieszczane w definicji konstruktora.

Sposobem na deklarowanie nowych metod i właściwości dla konstruktora jest wykorzystanie właściwości `prototype`.

### Przykład 3.49

```
function Klient(nazwisko, imie) {  
    this.nazwisko_k = nazwisko;  
    this.imie_k = imie;  
}  
  
Klient.prototype.zawod = 'kierowca';  
Klient.prototype.pisz_dane = function() {  
    document.write(this.nazwisko_k + ' ' + this.imie_k);  
}  
  
var osoba1 = new Klient("Malinowski", "Oskar", "kierowca");  
osoba1.pisz_dane();
```

W definicji konstruktora zostały zadeklarowane dwie właściwości. Po użyciu właściwości `prototype` została dodana metoda `pisz_dane()` oraz właściwość `zawod`. Od tej pory każdy obiekt, który będzie tworzony na podstawie konstruktora `Klient`, będzie miał tę dodatkową właściwość i metodę.

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

## Ćwiczenie 3.14

Uzupełnij kod utworzony w ćwiczeniu 3.13. Zmień imię ucznia Nowak z Paweł na Karol i klasę uczennicy Małak z 5c na 5a. Do konstruktora dodaj metodę `wypiszKlasa()`, która wyświetli nazwisko, imię ucznia i klasę. Dla każdego ucznia wywołaj utworzoną metodę.

## 3.7. Obiekty wbudowane języka JavaScript

Niektóre predefiniowane obiekty języka JavaScript to:

- `String` — łańcuch tekstowy,
- `Array` — tablica,
- `Date` — data,
- `Math` — obiekt do przeprowadzania operacji matematycznych.

### 3.7.1. Obiekt String

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

#### Przykład 3.50

```
var tekst = "Obiekty języka JavaScript";
var dlug = tekst.length;
```

Zmiennej `dlug` przypisana zostanie wartość 25 określająca długość tekstu.

Obiekt `String` ma dwa typy metod.

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

#### Przykład 3.51

```
var tekst = "Obiekty języka JavaScript";
var 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 3.52

```
var tekst = "Obiekty języka JavaScript";
var x = tekst.toLowerCase();
```

Zostanie zwrócony łańcuch `obiekty języka javascript`.

### Przykład 3.53

```
var tekst = "Obiekty języka JavaScript";
var 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` dołączymy dodatkową funkcję, której wywołanie spowoduje, że pierwsza litera łańcucha zostanie zmieniona na dużą.

### Przykład 3.54

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

Metoda `charAt()` zwraca znak z pierwszej pozycji łańcucha znaków. Natomiast 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 miał metodę, która zamieni jego pierwszą literę na dużą.

### Przykład 3.55

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

W wyniku wykonania skryptu wyświetli się tekst Komputer.

## 3.7.2. Obiekt Array

Tablice służą do przechowywania wielu zmiennych. W języku JavaScript do pracy z tablicami można używać wbudowanego obiektu `Array`. Ma 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 3.56

```
var tab1 = new Array(10);
var tab2 = [15];
```

W pierwszym przypadku uzyskamy tablicę zawierającą 10 pustych elementów, w drugim powstanie tablica zawierająca jeden element o wartości 15.

Tablicę można również tworzyć, wstawiając do niej konkretne wartości.

### Przykład 3.57

```
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 3.58

```
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 3.59

```
var tab5 = ['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 3.60

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

W wyniku zostaną wyświetlone po kolejni wszystkie elementy tablicy.

## Zadanie 3.1

Zmień zapis skryptu podanego w przykładzie 3.60, tak aby wyświetcone 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 3.61

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

## Łączenie elementów tablicy

Z pomocą metody `join()` można łączyć elementy tablicy w jeden ciąg znaków. 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 3.62

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

## Odwracanie kolejności elementów tablicy

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

**Przykład 3.63**

```
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 3.64**

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

**Przykład 3.65**

```
var Tablica = new Array(3000, 4567, 12459, 406745);
Tablica.sort();
document.write(Tablica.join());
```

Domyślnie tablica jest sortowana leksykograficznie. Powoduje to, że liczba 12459 będzie mniejsza od 4567, ponieważ w liczbie 12459 cyfra na pierwszej pozycji jest mniejsza. Aby to zmienić, 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:

- 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`.

**Przykład 3.66**

```
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("<br>" + 'Sortowanie domyślne: ');
Tablica.sort();
```

```

document.write(Tablica.join());
document.write("<br>" + '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 poprawnie uporządkowane od wartości najmniejszej do największej.

### Zadanie 3.2

Utwórz tablicę, której elementy będą zawierały cyfry i litery. Zdefiniuj funkcję, która pozwoli uporządkować elementy tablicy w ten sposób, że będą w niej umieszczone najpierw litery w kolejności alfabetycznej, a następnie cyfry w kolejności od wartości najmniejszej do największej.

## 3.7.3. Obiekt Date

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

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

### Przykład 3.67

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

Można utworzyć obiekt z określona 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 3.68

```
var data = new Date(2019, 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.

Jednym z przykładów wykorzystania języka JavaScript na stronach WWW jest wyświetlanie daty i czasu jako elementu strony internetowej.

### Przykład 3.69

```
<!DOCTYPE html>

<html>
  <head>
    <title>JavaScript - Data i czas</title>
    <meta charset="UTF-8">
  <body>
    <h2>Wyświetlam bieżącą datę i czas</h2>
    <p>
      <script>
        var data_n = new Date();
        var data_l = data_n.toString();
        var data_u = data_n.toGMTString();
        var 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 wyświetlania wartości otrzymanych zmiennych została użyta funkcja `document.write`. W rezultacie na stronie wyświetla się informacje o czasie lokalnym, uniwersalnym i regionalnym (rysunek 3.7).

## Wyświetlam bieżącą datę i czas

**Czas lokalny:** Mon Oct 07 2019 20:11:52 GMT+0200 (Środkowoeuropejski czas letni)

**Czas uniwersalny:** Mon, 07 Oct 2019 18:11:52 GMT

**Czas regionalny:** 7.10.2019, 20:11:52

**Rysunek 3.7.** Wyświetlenie na stronie informacji o dacie i czasie

### Ćwiczenie 3.15

Zdefiniuj funkcję, która będzie wyświetlała datę i czas po polsku, na przykład tak jak na rysunku 3.8.

## Wyświetlam bieżącą datę i czas po polsku

**Dzisiaj jest:**  
**Wtorek, 26 listopada 2019 roku**  
**Godzina: 19:38:24**

**Rysunek 3.8.** Data i czas wyświetlane na stronie internetowej

### Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Data</title>
</head>
<body>
<h2>Wyświetlam bieżącą datę i czas po polsku</h2>
<p>
<script>
function data_czas() {
    var miesiace = ["stycznia", "lutego", "marca", "kwietnia", "maja",
    "czerwca", "lipca", "sierpnia", "września", "października", "listopada",
    "grudnia"];
    var dni = ["Niedziela", "Poniedziałek", "Wtorek", "Środa", "Czwartek",
    "Piątek", "Sobota"];
    var data = new Date();
    var dzien = dni[data.getDay()];
    var miesiac = miesiace[data.getMonth()];
    var rok = data.getFullYear();
    var godzina = data.getHours();
    var minuta = data.getMinutes();
    var sekunda = data.getSeconds();
    godzina = godzina < 10 ? "0" + godzina : godzina;
    minuta = minuta < 10 ? "0" + minuta : minuta;
    sekunda = sekunda < 10 ? "0" + sekunda : sekunda;
    return `${dzien} ${miesiac} ${rok}, ${godzina}:${minuta}:${sekunda}`;
}
</script>
</p>

```

```
var rok = data.getFullYear();
var mies = data.getMonth();
var nr_dzien = data.getDate();
var dzien = data.getDay();
var godz = data.getHours();
var min = data.getMinutes();
var sek = data.getSeconds();

if (min < 10) {
    min = "0" + min;
}

if (sek < 10) {
    sek = "0" + sek;
}

var p_data_czas = dni[dzien] + ", " + nr_dzien + " " + miesiace[mies] + " " + rok +
" roku<br>" + "Godzina: " + godz + ":" + min + ":" + sek;
document.write(p_data_czas);

document.write("<b>Dzisiaj jest: <br>");
data_czas();
document.write("</b>");

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

### Zadanie 3.3

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

## 3.8. Obiekty DOM

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

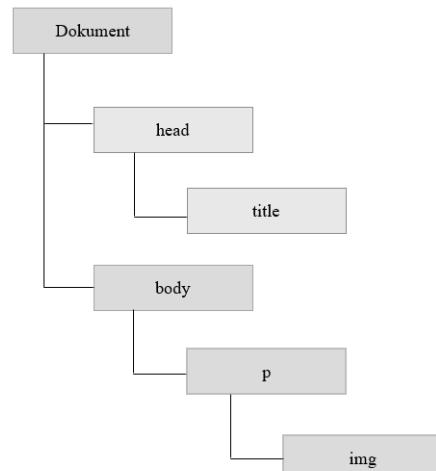
Gdy kod strony jest wczytywany do przeglądarki, przeglądarka zamienia ciąg znaków na stronę internetową. Informacje na temat interpretacji kodu HTML przeglądarka przechowuje w elementach będących obiektami (są to na przykład informacje o tym, które elementy przedstawić w postaci nagłówków, paragrafów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

DOM opisuje hierarchię obiektów na stronie oraz udostępnia metody i właściwości, które umożliwiają manipulowanie tymi obiektami. W tej hierarchii na samej górze znajduje się okno przeglądarki, czyli obiekt `window`. Zawiera ono 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 przez skrypty manipulujemy 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 3.70

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tytuł strony</title>
</head>
<body>
<p>Moje góry

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



Podany w przykładzie dokument można rysować w postaci drzewa (rysunek 3.9). Na samej górze jest dokument HTML, niżej znajdują się węzły (nodes).

**Rysunek 3.9.**

Hierarchia obiektów przykładowej strony internetowej

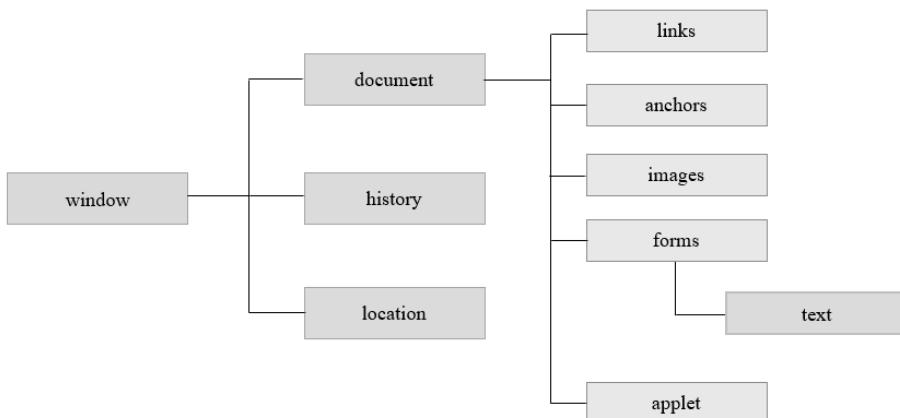
## 3.8.1. Hierarchia obiektów DOM

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

### Przykład 3.71

```
window.document.links
```

Wycinek hierarchii DOM z najważniejszymi obiektami strony internetowej został pokazany na rysunku 3.10.



**Rysunek 3.10.** Hierarchia obiektów DOM

## 3.8.2. Obiekty przeglądarki

Dla każdej strony internetowej zdefiniowane są następujące obiekty:

- `window`,
- `navigator`,
- `document`,
- `history`,
- `location`.

### Obiekt window

Obiektem nadzędnym dla wszystkich obiektów jest obiekt `window`, który reprezentuje okno przeglądarki. W danej chwili może istnieć wiele obiektów `window`. Każdy z nich reprezentuje otwarte okno przeglądarki. `window` jest najważniejszym obiektem w hierarchii, dlatego odwołanie do jego właściwości lub metod nie wymaga podania nazwy obiektu. Tworzony jest automatycznie podczas otwierania okna przeglądarki. Ma wiele właściwości przydatnych podczas tworzenia strony.

Do otwierania nowego okna używamy metody `open()`. Parametry metody to adres URL otwieranej strony oraz nazwa wewnętrzna okna (nie należy mylić jej 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 wykorzystywana jest metoda `close()`.

Do sterowania wykonywaniem kodu służą dwie metody:

- `window.setTimeout()`,
- `window.setInterval()`.

### **Metoda `window.setTimeout()`**

Metoda `setTimeout` przyjmuje dwa parametry. Pierwszy parametr jest ciągiem znaków lub referencją do funkcji, która ma się wykonać tylko raz. Drugi parametr to czas, po którym ma zostać wywołany kod podany jako pierwszy parametr.

```
window.setTimeout(kod, opóźnienie);
window.setTimeout(funkcja, opóźnienie);
```

### **Przykład 3.72**

```
window.setTimeout(function() {
    alert("Uwaga!!!!");
}, 1000);
```

### **Przykład 3.73**

```
setTimeout("showtime()",1000);
```

### **Metoda `window.setInterval()`**

Wywołanie metody `setInterval()` daje podobny efekt do wywołania metody `setTimeout()`. Różnica polega na tym, że w metodzie `setInterval()` podany kod będzie wykonywany wielokrotnie (w teorii do nieskończoności), co określony interwał czasowy.

```
window.setInterval(kod, opóźnienie);
window.setInterval(funkcja, opóźnienie);
```

### Przykład 3.74

```
window.setInterval("mojCzas()", 1000);
```

### Metoda window.clearInterval()

Do zakończenia wykonywanej co określony czas akcji zadanej w metodzie `setInterval()` służy metoda `clearInterval(int)`. Jako parametru tej metody należy użyć identyfikatora zwróconego przez metodę `setInterval()`.

### Przykład 3.75

```
<!DOCTYPE html>
<html>
<head>
<title>Metody obiektu Window</title>
<meta charset="UTF-8">
<script>
var licz = 0;
var Id = window.setInterval(
    function() {
        alert('działa');
        licz++;
        if (licz >= 5) {
            window.clearInterval(Id);
        }
    }, 3000);
</script>
</head>
<body>
</body>
</html>
```

W podanym przykładzie jako kod do wykonania w metodzie `setInterval()` została zadeklarowana funkcja `alert()`, która wyświetli w oknie dialogowym komunikat `działa`. Komunikat ten powinien być wyświetlany w nieskończoność, ale instrukcja `if` sprawdza stan zmiennej `licz` i po jego pięciokrotnym wyświetleniu zostanie wywołana metoda `clearInterval()`, która spowoduje zakończenie wykonywania akcji zadanej w metodzie `setInterval()`.

## navigator

Pozwala na dostęp do informacji dotyczących przeglądarki. Służy do ustalenia wersji przeglądarki, jaką posługuje się użytkownik. Właściwości tego obiektu mogą być tylko odczytywane. Najczęściej korzysta się z niego do sprawdzenia, czy przeglądarka jest odpowiednia do zastosowanej wersji języka JavaScript.

## Przykład 3.76

```
<!DOCTYPE html>

<html>
<head>
<meta charset="UTF-8">
<title>Jaka przeglądarka</title>
</head>
<body>
<script>
document.write("Masz przeglądarkę: ");
document.write(navigator.appName + "<br>");
document.write("Język przeglądarki: ");
document.write(navigator.language + "<br>");
document.write("Platforma: ");
document.write(navigator.platform);
</script>
</body>
</html>
```

Właściwość `appName` zawiera nazwę przeglądarki, `language` — język przeglądarki, a właściwość `platform` — nazwę platformy.

## Obiekt document

Obiekt `document` reprezentuje stronę internetową (zawiera informacje o bieżącym dokumencie HTML). Jest on potomkiem obiektu `window`. Poprzez właściwości obiektu `document` mamy wpływ na elementy strony (na przykład kolor tła, kolor czcionki). Jego metody umożliwiają wyświetlenie na przykład tekstu w oknie przeglądarki.

Za pomocą polecenia `window.document` można odwołać się do bieżącego dokumentu. Można to zrobić również przy użyciu polecenia `document`. Odwołanie nastąpi do bieżącego dokumentu w bieżącym oknie. Jeżeli zostało otwartych kilka okien, to aby określić, do którego dokumentu powinno nastąpić odwołanie, należy podać nazwę okna i nazwę dokumentu.

Informacje o bieżącym dokumencie otrzymamy, gdy odwołamy 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 `getElementsByName()`. Metoda `getElementById()` używana jest, gdy element, do którego się odwołujemy, ma atrybut `id`, natomiast metodę `getElementsByName()` wykorzystujemy do pobrania kolekcji zawierającej elementy danego typu.

### Przykład 3.77

```
<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.getElementsByTagName("p")` — odwołanie do kolekcji akapitów,
- `document.getElementsByTagName("p")[0]` — odwołanie do pierwszego akapitu w kolekcji.

Gdy używa się 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 3.78

```
<!DOCTYPE html>
<html>
<head>
<title>Przyciski</title>
<meta charset="UTF-8">
</head>
```

```

<body>
<input type="button" id="klik" value="Kliknij!">
<script>
var b = document.getElementById("klik");
document.write("<br>Tekst z przycisku: " + b.value);
</script>
</body>
</html>

```

W podanym przykładzie za pomocą metody `getElementById()` została pobrana i zapisana w zmiennej `b` wartość elementu o `id="klik"`. Następnie za pomocą metody `write()` wartość zmiennej została wyświetlona.

Możliwość odwołania się do elementu strony jest wykorzystywana do zmiany jej zawartości. Zawartość elementu strony można odczytać i zmienić, używając właściwości `innerHTML`. Właściwość tę ma każdy element strony internetowej. Określa ona wartość przypisaną elementowi.

### Przykład 3.79

```

<!DOCTYPE html>
<html>
<head>
<title>Tekst</title>
<meta charset="UTF-8">
</head>
<body>
<h2>Zmień tekst</h2>
<script>
function zmien_tekst()
{
    document.getElementById('blok').innerHTML = 'Jesień mimozami się zaczyna';
}
</script>
<p>Pory roku: <b id="blok">Lato, lato, lato czeka...</b></p>
<input type="button" onclick="zmien_tekst()" value="Zmień tekst"/>
</body>
</html>

```

### Zmień tekst

Pory roku: **Jesień mimozami się zaczyna...**

**Zmień tekst**

#### Rysunek 3.11.

Możliwość zmiany tekstu wyświetlonego na stronie

Po kliknięciu przycisku nastąpi zmiana wyświetlonego tekstu (rysunek 3.11).

## Ćwiczenie 3.16

W przykładzie 3.69 zostało zdefiniowane wyświetlanie daty i czasu. Zmiana czasu następowała po odświeżeniu strony. Zmodyfikuj powstały kod, tak aby czas wyświetlany na stronie był zawsze aktualny.

### Rozwiążanie

```
<!DOCTYPE html>
<html>
<head>
<title>Mój zegar</title>
<meta charset="UTF-8">
</head>
<body>
<h2>Bieżący czas</h2>
<div id="zegar"></div>

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

```

Metoda `getElementById()` odwołuje się do wcześniej zdefiniowanego identyfikatora `zegar`. W przykładzie za pomocą właściwości `innerHTML` elementowi o identyfikatorze `zegar` przypisana została wartość zmiennej `czas`, która zawiera bieżący czas. Operator warunkowy `((min < 10) ? ":0" : ":")) + min;`) ma trzy argumenty: `test`, po którym występuje znak `?` oraz wyrażenie1 i wyrażenie2 oddzielone znakiem `:`. Jeżeli wartość `test` to `true`, to wynikiem jest wyrażenie1, w przeciwnym razie wynikiem jest wyrażenie2. Wynik interpretacji kodu został pokazany na rysunku 3.12.

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

### Bieżący czas

12:23:42

#### Rysunek 3.12.

Wyświetlanie na stronie aktualnego czasu

### Zadanie 3.4

Zmodyfikuj kod pokazany w przykładzie 3.79 w ten sposób, aby kolejne kliknięcia przycisku `Zmień tekst` powodowały wyświetlanie kolejnych tekstów, na przykład „Hu, hu, ha, nasza zima zła” i „Wiosna, wiosna wonna i radosna”. Po wyświetleniu podanych tekstów powinien nastąpić powrót do wyświetlania tekstu pierwszego.

### Zadanie 3.5

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

### Zadanie 3.6

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 stronach odwiedzanych w bieżącej sesji. Ma 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, na przykład `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` ma właściwość `history.length`, która zawiera informację o długości listy historii.

### Ćwiczenie 3.17

Wykorzystaj metody `back()` i `forward()`, do utworzenia skryptu, który wyświetli na stronie przyciski *Wstecz* oraz *Dalej*, umożliwiające poruszanie się w przeglądarce po odwiedzanych stronach, jak pokazano na rysunku 3.13.



**Rysunek 3.13.** Przyciski Wstecz i Dalej z podpiętymi metodami `back()` i `forward()`

### Rozwiązań

```
<!DOCTYPE html>

<head>
<title>Przyciski</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Rewolucja przemysłowa 4.0</h1><br>
<input type="button" onclick="history.back()" value="Wstecz">
<input type="button" onclick="history.forward()" value="Dalej">
</body>
</html>
```

## Ćwiczenie 3.18

Modyfikując ćwiczenie 3.17, dodaj za pomocą znaczników HTML kod umożliwiający otwieranie kilku kolejnych stron internetowych. Wykorzystaj przyciski *Wstecz* i *Dalej* do poruszania się między otwartymi stronami.

## 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, a także 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.

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 (na przykład `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, path to łańcuch znakowy określający ścieżkę dostępu do dokumentu na serwerze, fragment to łańcuch znakowy odpowiadający nazwie zakotwiczenia (przypisanie tu jakiejś wartości spowoduje przewinięcie dokumentu do wskazanego punktu), query to człon lokalizatora URL.

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

## Przykład 3.80

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

### **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` ma 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 3.7

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 3.81

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

### Obiekt anchor

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

#### WSKAZÓWKA

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

Kotwice są zapisywane w tablicy o nazwie `anchors`. Każdy jej element jest obiektem `anchor`.

Właściwość tablicy `document.anchors.length` określa liczbę elementów kotwicy 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, wpisując w kodzie polecenie:

```
document.forms[0]
```

lub przez nazwę, wpisując w kodzie polecenie:

```
document.forms['Form1']
```

Inną metodą odwołania się do formularza jest wykorzystanie metody getElementById(), na przykład:

```
document.getElementById('form1')
```

### Przykład 3.82

```
<body>
<form id="form1" name="Form1" action="" method="post">
...
</form>
<script>
document.forms[0];
// lub
document.forms['Form1'];
// lub
document.getElementById('form1');
...
</script>
</body>
```

Jeżeli został zastosowany atrybut name (jak w przykładzie 3.82), to do formularza można odwołać się również w ten sposób: document.Form1.

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

### Przykład 3.83

```
<!DOCTYPE html>
<html>
<head>
<title>Co słyszać</title>
```

```

<meta charset="UTF-8">
</head>
<body>
<form id="form1" name="Form1" action=" " method="post">
Podaj imię: <input type="text" name="imie" id="imie"/>
<button onclick="witaj()">Kliknij!</button>
</form>
<script type="text/javascript">
function witaj() {
    var imie = document.forms['Form1'].imie.value;
    alert('Co słyszać, ' + imie + '?');
}
</script>
</body>
</html>

```

Podany kod definiuje formularz z polem imię 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 3.14).

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:

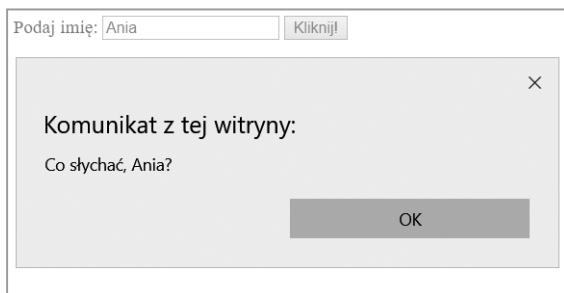
```

document.forms['Form1'].elements[0]
document.forms['Form1'].elements['imie']
document.forms['Form1'].imie

```

albo za pomocą metody `getElementById()`:

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



**Rysunek 3.14.**

Pobrane z formularza imię zostało wyświetlone w oknie z komunikatem

### Przykład 3.84

```
<!DOCTYPE html>
<html>
<head>
<title>Lista formularza</title>
<meta charset="UTF-8">
</head>
<body>
<form id="form1" action=" " 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>
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 3.15).

The screenshot shows a web page with a form and its output. At the top is a standard HTML form with two text input fields: 'Imię:' containing 'Jan' and 'Nazwisko:' containing 'Kowalski'. Below the form is a button labeled 'Wyślij'. Underneath the form, the text 'Lista elementów formularza:' is displayed, followed by a list of the values from the inputs: 'Jan', 'Kowalski', and 'Wyślij'.

Imię:	Jan
Nazwisko:	Kowalski
<b>Wyślij</b>	
<b>Lista elementów formularza:</b>	
Jan	
Kowalski	
Wyślij	

**Rysunek 3.15.** 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 obsługi zdarzeń. Skrypty takie definiują zachowanie się przeglądarki w przypadku wystąpienia określonego zdarzenia. Większość zdarzeń wywoływana jest przez działania użytkownika (na przykład 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 (na przykład zakończenie wczytywania strony).

Każdy skrypt opisujący zdarzenie skojarzony jest z określonym obiektem strony internetowej. Żeby takie zdarzenie mogło zostać obsłużone, element HTML musi być dostępny dla skryptu. Jedną z metod, by to zapewnić, jest wstawianie skryptu na końcu strony przed znacznikiem `</body>`. Inną metodą jest dodanie atrybutu `defer` do znacznika `<script>`, co spowoduje, że gdy przeglądarka napotka skrypt, będzie on wczytywany w tle razem ze stroną internetową, a uruchomiony zostanie po załadowaniu całego dokumentu. Dobrym rozwiązaniem jest użycie zdarzenia `DOMContentLoaded` — spowoduje to, że kod, który odwołuje się do elementów w kodzie HTML, zostanie wykonany po ich wczytaniu.

### Przykład 3.85

```
document.addEventListener("DOMContentLoaded", function() {  
    console.log("DOM został wczytany");  
});
```

Metoda `addEventListener()` przypisuje do elementu zdarzenie opisane daną funkcją.

Przykłady zdarzeń to:

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

W kodzie HTML w znaczniku opisującym obiekt musi znaleźć się specyfikacja skryptu opisującego zdarzenie.

Przykładowo zdarzenie `onmouseover` zachodzi wówczas, gdy wskaźnik myszy pojawi się nad obiektem (na przykład przyciskiem). Zdarzenie to może zostać opisane w następujący sposób w kodzie HTML:

```

```

Obsługa zdarzenia została opisana jako wartość atrybutu (`zmienna`), a sam atrybut otrzymał nazwę, która jest nazwą zdarzenia (`onmouseover`). W tym wypadku obsługa zdarzenia została zaprojektowana w postaci funkcji, czyli po wywołaniu funkcji zdarzenie zostanie obsłużone.

Nazwy zdarzeń w języku JavaScript zwykle zaczynają się od słowa `on`. Kliknięcie będzie mieć nazwę `onclick`, najechanie myszą `onmouseover` itd.

Jest kilka sposobów obsługi zdarzeń. Aby obsługiwać zdarzenie, nie zawsze trzeba tworzyć funkcję.

## Obsługa zdarzeń w kodzie HTML

Jeżeli kod, który ma zostać wywołany, jest pojedynczą instrukcją, można wpisać skrypt bezpośrednio w znaczniku.

### Przykład 3.86

```
<!DOCTYPE html>
<html>
<head>
<title>Okno ALERT</title>
<meta charset="UTF-8">
</head>
<body>
<input type="button" value="Pokaż" onclick="alert('Witaj !');"/>
</body>
</html>
```

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

Nie jest to najlepsza metoda obsługi zdarzenia, ponieważ miesza się w niej język HTML z językiem JavaScript.

## Obsługa zdarzenia jako właściwości obiektu

W tej metodzie musi zostać zdefiniowana funkcja obsługująca zdarzenie. Następnie funkcję należy przypisać do danego elementu jako jego właściwość. Funkcja powinna zostać przypisana po nazwie bez nawiasów, ponieważ nie zostanie ona wywołana.

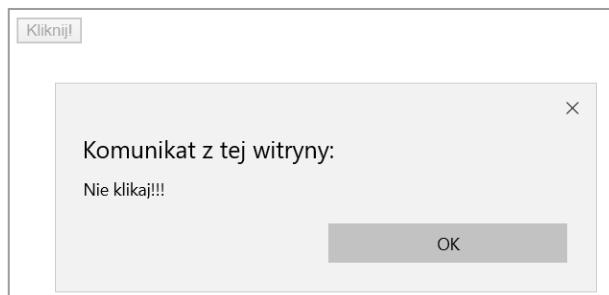
### Przykład 3.87

```
<!DOCTYPE html>
<html>
<head>
<title>Klik</title>
```

```
<meta charset="UTF-8">
</head>
<body>
<input type="button" id="klik" value="Kliknij!">
<script>
function pisz() {
    alert('Nie klikaj!!!');
}
document.getElementById("klik").onclick = pisz;
</script>
</body>
</html>
```

W podanym przykładzie został utworzony przycisk `<input type="button" id="klik" value="Kliknij!">` z identyfikatorem `klik` oraz została zdefiniowana funkcja `pisz()`. Następnie funkcja `pisz` została przypisana do elementu `klik` jako jego właściwość `document.getElementById("klik").onclick = pisz;`.

Wynik interpretacji kodu został pokazany na rysunku 3.16.



**Rysunek 3.16.** 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 funkcjonalności.

Do zdarzeń wywołanych działaniem myszy należą:

- `onclick` — występuje po kliknięciu myszą,
- `ondblclick` — występuje po dwukrotnym kliknięciu myszą,
- `onmousedown` — występuje, gdy przycisk myszy zostanie wciśnięty,

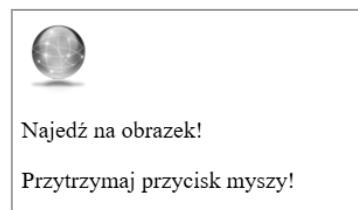
- onmouseup — występuje, gdy przycisk myszy zostanie zwolniony,
- onmouseover — występuje, gdy kurSOR myszy zostanie umieszczony na elemencie,
- onmousemove — występuje, gdy kurSOR myszy zostanie przesunięty wewnątrz elementu,
- onmouseout — występuje, gdy kurSOR myszy zostanie przesunięty poza element.

### Przykład 3.88

```
<!DOCTYPE html>
<html>
<head>
<title>Zdarzenie myszy</title>
<meta 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"'>
</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 ustawnieniu myszy na pierwszym rysunku ulegnie on zmianie. Wciśnięcie przycisku myszy spowoduje kolejną zmianę rysunku, a po przesunięciu myszy poza rysunek zostanie przywrócona pierwsza grafika. Kliknięcie rysunku spowoduje przejście do strony [helion.pl](http://www.helion.pl) (rysunek 3.17).



**Rysunek 3.17.**  
Obsługa zdarzeń myszy

### Przykład 3.89

```
<!DOCTYPE html>
<html>
<head>
<script>
function ik2() {
    document.getElementById('ikona').src = "Ikona2.png";
}
```

```
function ik1() {  
    document.getElementById('ikona').src = "Ikona1.png";  
}  
</script>  
</head>  
<body>  
  
<p>Kliknij i przytrzymaj!</p>  
</body>  
</html>
```

W przykładzie zostały wykorzystane zdarzenia myszy onmousedown i onmouseup. W celu odwołania się do elementu `id="ikona"` posłużyono się metodą `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 typu `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 typu `reset`).

## 3.9.4. Zdarzenia elementów formularza

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

- `onfocus` — element formularza został zaznaczony, 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 obsługi zdarzeń. Skrypty będą wykonywane w ścisłe 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 one 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 momencie 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.

### Ćwiczenie 3.19

Wykonaj animację dowolnego tekstu znajdującego się w górnej części strony internetowej. Tekst powinien przemieszczać się od prawej strony okna do lewej, a gdy dotrze do lewej krawędzi, animacja ma zostać wykonana ponownie.

### Rozwiążanie

```
<!DOCTYPE html>

<html>
<head>
<title>Przepływający tekst</title>
<meta charset="UTF-8">
<style type="text/css">
#napis {
    position: absolute;
    background-color: #c0cfc0;
    width: 350px;
    float: left;
    font-size: 18pt;
    font-family: Arial;
}
</style>
<script>
function animacja() {
    var blok = document.getElementById('napis');
    if (parseInt(blok.style.left) < 50) {
        blok.style.left = "500px";
    } else {
        blok.style.left = (parseInt(blok.style.left) - 3) + "px";
    }
}
window.setInterval(animacja, 100);
</script>
```

```

</head>

<body>

<div id="napis" style="left: 500px;">JavaScript - język skryptowy :-)

```

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: 500px;". 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 = "500px"). Jeżeli wynik jest większy niż 50px, to kontynuowane jest odejmowanie od aktualnej wartości 3px. Metoda window.setInterval() wywołuje funkcję animacja() co 0,1 sekundy.

## Baner (tekst płynący)

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

### Ćwiczenie 3.20

Umieść w bloku, w górnej części okna strony internetowej, długий tekst. Wykonaj jego animację, wzorując się na przepływie newsów na pasku w dolnej części ekranu telewizora. Tekst powinien przewijać się litera po literze w utworzonym bloku (rysunek 3.18).

ny internetowe Zostań z nami! To jest to! Najlepsze stro

**Rysunek 3.18.** Pasek z przewijanym tekstem

### Rozwiązanie

```

<!DOCTYPE html>

<html>

<head>

<title>Baner reklamowy</title>

<meta charset="UTF-8">

<style type="text/css">

#tekst {

    font-size: 18pt;
    font-family: Arial;
    text-align: center;
}

```

```
</style>
<script>
czas = 200;
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 id="tekst" name="text" size=60 value=" Najlepsze strony
internetowe. Zostań z nami! To jest to! ">
</form>
</body>
</html>
```

W podanym przykładzie została wykorzystana metoda `window.setTimeout()`, która wywołuje funkcję po określonej liczbie milisekund. Metoda ta ma dwa argumenty. Pierwszy z nich to nazwa uruchamianej funkcji, a 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.

#### Ćwiczenie 3.21

Przygotuj dwa dowolne pliki ze zdjęciami. Umieść na stronie internetowej pierwsze zdjęcie. Napisz skrypt, który spowoduje, że po wskazaniu tego zdjęcia myszą zostanie ono zamienione na zdjęcie drugie. Po odsunięciu myszy na stronie ponownie powinno zostać wyświetlane zdjęcie pierwsze.

## Rozwiązanie

```
<!DOCTYPE html>

<html>
<head>
<title>Zmiana obrazka</title>
</head>
<body>
<p>

</p>
<script>
var obraz = document.getElementById('widok');
obraz.onmouseover = function() {
    this.src = 'obraz2.png';
}
obraz.onmouseout = function() {
    this.src = 'obraz1.png';
}
</script>
</body>
</html>
```

W podanym rozwiążaniu zostały wykorzystane dwa obrazki zapisane w plikach *obraz1.png* oraz *obraz2.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('widok');` została utworzona zmienna *obraz* zawierająca odnośnik do elementu o *id="widok"*. Kolejnym działaniem jest obsługa zdarzenia *onmouseover* dla elementu *obraz*. Po wystąpieniu tego zdarzenia nastąpi zmiana atrybutu *src* obiektu (`this.src = 'obraz2.png'`). Po usunięciu kurSORA nad obrazkiem nastąpi ponowna zmiana atrybutu *src* (`this.src = 'obraz1.png'`). Słowo kluczowe *this* wskazuje na obiekt, który wywołał funkcję.

## Animowany baner (slider)

Innym rodzajem animowanej grafiki jest baner ze zmieniającymi się automatycznie obrazkami.

## Ćwiczenie 3.22

Utwórz stronę internetową z miejscem zarezerwowanym na slider. Przygotuj sześć zdjęć o takich samych rozmiarach, które będzie można umieścić w górnej części strony. Napisz skrypt, który automatycznie na przykład co 3000 milisekund będzie zmieniał zdjęcia wyświetlane w sliderze.

### Rozwiążanie

```
<!DOCTYPE html>
<html>
<head>
<title>Animowany baner</title>
<style type="text/css">
#baner {
    width: 900px;
    height: 150px;
}
</style>
<script>
window.onload = zmiana;
var nr = 0;
function zmiana() {
    var obrazy = ['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, 3000);
}
</script>
</head>
<body>
<div>

</div>
</body>
</html>
```

W podanym rozwiążaniu zastosowano tablicę do przechowywania nazw obrazów. Wielkość tablicy może być zmieniana w zależności od liczby wyświetlanych obrazów. Ich rozmiar został zdefiniowany za pomocą stylów.

Ponieważ próba odwołania się do elementów, które nie zostały jeszcze wczytane, mogłyby spowodować wystąpienie błędów i nieprawidłowe działanie skryptu, pierwsze polecenie skryptu `window.onload = zmiana;` obsługuje zdarzenie związane z kończeniem wczytywania strony i wywołuje funkcję `zmiana()`.

Pojawianie się kolejnych zdjęć jest obsługiwane przy użyciu licznika czasu. Zmienna `nr` to licznik wyświetlanych obrazów. Jej początkowa wartość została ustawiona na 0. Funkcja `zmiana()` zawiera definicję tablicy `obrazy`, która przechowuje nazwy plików z wyświetlonymi obrazami. Za pomocą kodu `if (nr == obrazy.length)` sprawdza się, czy wartość licznika obrazów (`nr`) jest równa 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.

## 3.11. Walidacja formularzy

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

Jednym ze sposobów kontrolowania danych z formularzy jest stosowanie skryptów działających po stronie serwera (na przykład PHP). Innym 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 wypełnieniu 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 skryptów JavaScript nie zwalnia z konieczności sprawdzania poprawności danych po stronie serwera (walidacja danych po stronie serwera zapewnia bezpieczne działanie całej aplikacji).

### 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 te pola są puste, formularz nie zostanie wysłany.

### Przykład 3.90

```
<!DOCTYPE html>
<html>
<head>
<title>Sprawdź_formularz</title>
<meta charset="UTF-8">
<script>
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: #fdfdfd;
    padding: 10px;
}
.pole {
    position: absolute;
    left: 90px;
}
```

```

}

#ak1 {
    font-size: 14pt;
}

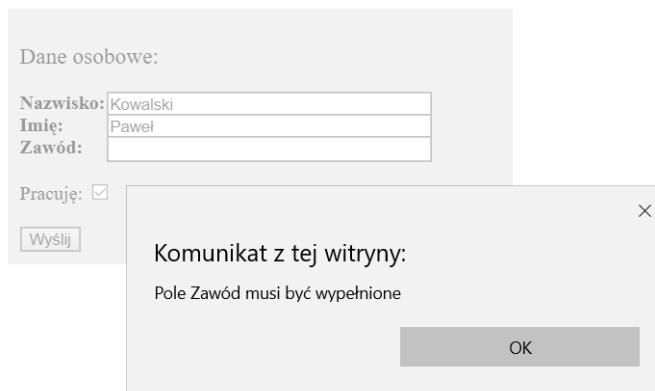
</style>
</head>
<body>
<div>

<form enctype="text/plain" action=" " method="post"
onsubmit="return sprawdz(this);">

<p id="ak1">Dane osobowe:</p>
<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 zostanie wysłany (rysunek 3.19).



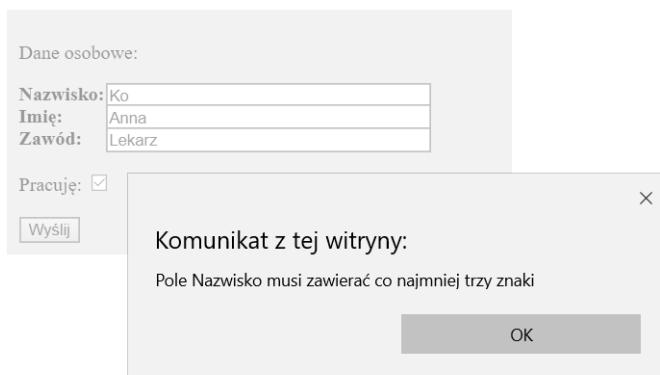
**Rysunek 3.19.** Walidacja formularza. Formularz nie zostanie wysłany

Inny sposób zdefiniowania funkcji sprawdz() z wykorzystaniem metody getElementById() został pokazany w kolejnym przykładzie (przykład 3.91). Za pomocą funkcji sprawdz() można nie tylko zweryfikować to, czy pole zostało wypełnione, ale również sprawdzić długość tekstu wprowadzonego do poszczególnych pól.

### Przykład 3.91

```
function sprawdz(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 wykonania podanej w przykładzie funkcji został pokazany na rysunku 3.20.



**Rysunek 3.20.** Formularz nie zostanie wysłany, ponieważ ciąg znaków wpisany w polu Nazwisko jest za krótki

## 3.11.2. Sprawdzenie pól formularza po wypełnieniu

Sprawdzenie pól formularza po wypełnieniu stosuje się w celu poinformowania użytkownika o niepoprawnym wypełnieniu formularza. Najprostszym przypadkiem jest sprawdzanie, czy w polu pojawił się wymagany tekst.

### Przykład 3.92

```
<!DOCTYPE html>
<html>
<head>
<title>Sprawdzenie po wpisaniu</title>
<meta charset="UTF-8">
<style type="text/css">
.pole {
    position: absolute;
    left: 90px;
}
</style>
<script>
function adres() {
    if (document.getElementById('mail').value != 'e-mail') {
        alert('Wpisz tekst: "e-mail"');
        return false;
    }
    return true;
}
</script>
</head>
<body bgcolor="#dfdfdf">
<form action="" method="post">
<b>Nazwisko:</b>
<input class="pole" name="nazw" id="nazw" value="" size="40"><br>
<b>e-mail:</b>
<input class="pole" name="mail" id="mail" value="" size="40"
onblur="return adres()"><br>
<b>Zawód:</b>
<input class="pole" name="zawod" id="zawod" value="" size="40"><br><br>
<input type="submit" value="Wyślij">&nbsp; &nbsp;
</form>
</body>
</html>
```

W podanym przykładzie zastosowano obsługę zdarzenia `onblur` (utrata fokusu) dla elementu e-mail. Po jego opuszczeniu zostaje uruchomiona funkcja `adres()`, która sprawdza wartość przypisaną do elementu. Jeżeli jest niepoprawna, pojawi się komunikat Wpisz tekst: "e-mail".

### 3.11.3. Wyrażenia regularne

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

Język JavaScript ma wbudowane mechanizmy obsługi wyrażeń regularnych. Za obsługę tych wyrażeń odpowiada obiekt `RegExp` (wzorzec, flaga). Można go utworzyć na dwa sposoby:

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

Podane 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]	finalny, tonalne
[ ^ ... ]	dowolny z niewymienionych znaków	kro[^st]	krowa, kroki, kropy
	jeden z ciągów rozdzielenych znakiem	pierwszy 1	pierwszy, 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

Symbol	Znaczenie	Przykład	Ciągi zgodne z podanym wzorcem
{ 2, 5 }	od dwóch do pięciu po-przedzających 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()` może wystąpić flaga, 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 wprowadzone specjalne klasy znaków. Ich symbole mogą być wykorzystane przy definiowaniu wyrażeń regularnych (tabela 3.8).

**Tabela 3.8.** Klasa 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 lub znakiem _
\W	każdy znak, który nie jest literą, cyfrą ani 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 do 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 sposób zapisu wzorca do weryfikacji imienia i nazwiska:

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

lub:

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

## Wzorzec do weryfikacji adresu e-mail

```
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 znaku -. 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.

Gdy zna się sposoby definiowania wzorców do weryfikacji danych zapisanych w formularzu, można przystąpić do napisania kodu, który będzie sprawdzał te dane na stronie internetowej.

### Przykład 3.93

```
<!DOCTYPE html>

<html>
<head>
<title>Sprawdź według wzorca</title>
<meta charset="UTF-8">
<script>
function Spr_wzorzec()
{
    var form = document.getElementById('in_form'),
    wzory = {
        'nazwisko' : /^[a-zA-Z]{2,}\s+[a-zA-Z]{2,}$/
    }
}
```

```
'E-mail' : /^[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: #dfdfdf;  
}  
p {  
font-size: 14pt;  
}  
.pole {  
position:absolute;  
left:130px;  
}
```

```
</style>

</head>
<body>
<div>
<form id="in_form" enctype="text/plain" action=""
method="post" onsubmit='return Spr_wzorzec();'>

<p>Dane osobowe:</p>
<b>Imię i nazwisko:</b>
<input class="pole" name="nazwisko" id="nazwisko" value=""><br>
<b>E-mail:</b>
<input class="pole" name="E-mail" 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 następuje sprawdzenie za pomocą funkcji `spr_wzorzec()` 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 wartością zapisanego w tablicy wyrażenia regularnego: `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 3.93 został pokazany na rysunku 3.21.

The screenshot shows a user interface for a form. On the left, there is a light gray panel labeled "Dane osobowe:" containing fields for "Imię i nazwisko" (Polak Marcin) and "E-mail" (Marcin). Below these fields is a button labeled "Wyślij". To the right, a modal dialog box is displayed with the title "Komunikat z tej witryny:". Inside the dialog, the message "Pole email ma nieprawidłową wartość" is shown, indicating an invalid email address. At the bottom right of the dialog is an "OK" button.

**Rysunek 3.21.** Formularz nie zostanie wysłany, ponieważ pole zawierające adres e-mail jest wypełnione nieprawidłowo

### 3.11.4. Zerowanie pól formularza

Wypełnione pola formularza można wyczyścić za pomocą metody `reset()`, która powoduje wyzerowanie danych wprowadzonych do wszystkich pól formularza.

#### Przykład 3.94

```
<!DOCTYPE html>
<html>
<head>
<title>Zerowanie formularza</title>
<meta charset="UTF-8">
<script>
function zeruj() {
    document.getElementById("form1").reset();
}
</script>
<style type="text/css">
div {
    width: 400px;
    background-color: #fdfdfdf;
    padding: 10px;
}
.pole {
    position: absolute;
    left: 130px;
}
```

```
</style>
</head>
<body>
<p>Wprowadź swoje dane:</p>
<div>
<form id="form1">
    Imię: <input class="pole" type="text" name="imie"><br>
    Nazwisko: <input class="pole" type="text" name="nazw"><br><br>
    <input type="button" onclick="zeruj()" value="Wyczyść dane">
</form>
</div>
</body>
</html>
```



## 3.12. Pytania i zadania

### 3.12.1. Pytania

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. Jak działa model obiektowy DOM?
8. Wymień podstawowe obiekty przeglądarki.
9. W jaki sposób obsługiwane są zdarzenia?
10. Na czym polega walidacja formularzy?
11. Do czego wykorzystywane są podczas walidacji danych wyrażenia regularne?

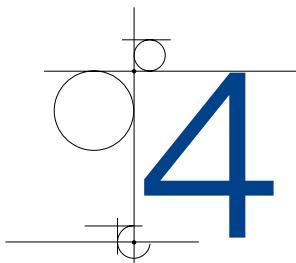
### 3.12.2. Zadania

#### Zadanie 1.

Utwórz stronę internetową, na której w prawym górnym rogu będą wyświetlane bieżąca data i czas. W nagłówku strony znajdzie się slider ze zdjęciami, które będą się zmieniały co 10 sekund. Poniżej slidera ma być umieszczony baner reklamowy z tekstem  *Witamy na naszej stronie*, który będzie „pływął” 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 ma spowodować zmianę jej wyglądu. Odsunięcie kurSORA ma skutkować przywróceniem poprzedniego wyglądu ikony. Kliknięcie pierwszej ikony spowoduje przeniesienie do strony [www.men.gov.pl](http://www.men.gov.pl), drugiej — do strony [www.cke.edu.pl](http://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 strony internetowej. Do najpopularniejszych należą: jQuery, Angular, Knockout, React, Dojo. 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, za pomocą których można realizować często spotykane zadania (na przykład animacje, obsługę zdarzeń, tworzenie efektów wizualnych na stronie internetowej). Jej głównymi zaletami są 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

Jest kilka sposobów na rozpoczęcie pracy z biblioteką jQuery. Najpopularniejsze to:

- pobranie biblioteki jQuery ze strony <http://www.jquery.com>,
- dołączenie biblioteki jQuery z sieci dostarczania treści, na przykład Google.

#### Pobieranie jQuery

Aby korzystać z biblioteki jQuery, należy pobrać jej plik ze strony <http://www.jquery.com>. Na stronie dostępne są dwie wersje biblioteki jQuery, wersja nieskompresowana i skompresowana.

Wersja nieskompresowana przeznaczona jest do programowania i testowania.

W celu skrócenia czasu ładowania w wersji skompresowanej usunięte są niepotrzebne komentarze, podziały wierszy, spacje i znaki tabulacji. Zmniejszenie objętości pliku uzyskano kosztem jego czytelności, dlatego ta wersja pliku nie nadaje się do testowania i analizy kodu.

## Dołączenie jQuery z sieci

Druga metoda to korzystanie z biblioteki za pośrednictwem sieci dostarczania treści. 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.

Biblioteka jQuery jest pojedynczym plikiem języka JavaScript, do którego można odwoływać się za pomocą znacznika `<script>`. Znacznik ten powinien znajdować się w sekcji `<head>`.

Po wybraniu metody korzystania z biblioteki należy dołączyć jQuery w kodzie projektowanej strony internetowej.

W sekcji `<head>` dokumentu HTML powinien zostać wstawiony wpis:

```
<script src="jquery-3.4.1.min.js"></script>
```

Podany kod dodaje do skryptu bibliotekę jQuery w wersji 3.4.1.

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 charset="UTF-8">
<link href="css/style.css">
<script src="js/jquery.js"></script>
<script>
// 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 bibliotek przez firmę Google został pokazany w przykładzie 4.2.

### Przykład 4.2

```
<!DOCTYPE html>

<html>
  <head>
    <title>Dołaczanie biblioteki jQuery</title>
    <meta charset="UTF-8">
    <link href="css/style.css">
    <script src="https://ajax.googleapis.com/ajax/
      libs/jquery/3.4.1/jquery.min.js">
    </script>
    <script>
      // treść skryptu
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Dołączanie biblioteki jQuery udostępnionej przez firmę Microsoft wygląda następująco:

```
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.4.1.min.js"
></script>
```

Z kolei dołączanie biblioteki jQuery udostępnionej na serwerze jQuery wygląda tak:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
```

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

Istnieje kilka sposobów na dołączanie funkcji, która zostanie uruchomiona, gdy struktura drzewa DOM otwieranej witryny zostanie załadowana do przeglądarki. Dla przykładu:

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

lub

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

Od wersji jQuery 3.0 zalecane jest stosowanie tylko tej drugiej wersji składni.

## 4.1.2. Selektory

Selektory to podstawowe elementy biblioteki jQuery. Są używane we wszystkich operacjach dotyczących modelu DOM. Za ich pomocą 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
<code>\$('div')</code>	pobiera wszystkie bloki <code>div</code> z dokumentu
<code>\$('.blok')</code>	pobiera z dokumentu wszystkie elementy, które mają klasę <code>blok</code>
<code>\$('#id')</code>	pobiera z dokumentu jeden element o identyfikatorze <code>id</code>
<code>\$(p)</code>	pobiera z dokumentu wszystkie akapity
<code>\$(*)</code>	pobiera wszystkie elementy modelu DOM lub innego elementu
<code>\$(this).hide()</code>	ukrywa bieżący element
<code>\$(p).hide()</code>	ukrywa wszystkie elementy <code>&lt;p&gt;</code>
<code>\$('#id').hide()</code>	ukrywa element o identyfikatorze <code>id</code>

Ogólny sposób użycia selektora jest taki:

```
$( '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 z metody biblioteki jQuery. Metodą biblioteki może być na przykład definicja stylów albo animowanie elementów na stronie.

### Przykład 4.3

```
$(function() {  
    $('p').css('font-family', 'Arial, Verdana');  
});
```

W podanym przykładzie zapis `$('p')` to selektor jQuery dla wszystkich elementów `p` na stronie. `$()` to funkcja, która zwraca nowy obiekt jQuery. Wywołana funkcja `css()` to metoda tego obiektu. Powoduje ona ustawienie właściwości `font-family` dla wszystkich elementów `p` na stronie.

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

### Przykład 4.4

Kod HTML:

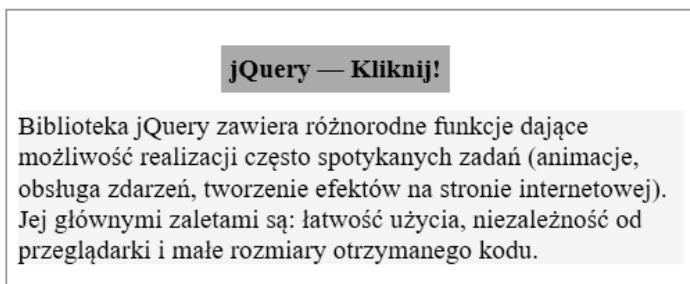
```
<!DOCTYPE HTML>  
  
<html>  
  <head>  
    <title>jQuery</title>  
    <meta charset="UTF-8">  
    <style type="text/css">  
      div {  
        width: 400px;  
        background-color: #dff;  
        display: none;  
      }  
      p {  
        background-color: #aaa;  
        position: absolute;  
        left: 130px;  
        font-weight: bold;  
        padding: 5px;  
      }  
    </style>  
    <script src="js/jquery-3.4.1.js"></script>  
    <script src="js/skrypt4.js"></script>  
  </head>
```

```
<body>
<p id="bibl">jQuery — Kliknij!</p><br><br><br>
<div id="bibl_info">
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 otrzymanego kodu.
</div>
</body>
</html>
```

Plik ze skryptem *skrypt4.js*:

```
$(function() {
    $('#bibl').click(function() {
        $('#bibl_info').toggle("slow");
    });
});
```

W podanym wyżej przykładzie biblioteka pobrana ze strony została zapisana w folderze *js*. W pliku *skrypt4.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 — Kliknij!* 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')` oznacza pobranie elementu o identyfikatorze `bibl` i po wystąpieniu zdarzenia `click` wywołanie funkcji obsługującej to zdarzenie. W wyniku wystąpienia zdarzenia 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 `skrypt4.js` spowoduje zmianę działania strony.

### Przykład 4.5

```
$(function() {
  $('#bibl').click(function() {
    $('*').css('font-family', 'Arial, Verdana');
  });
});
```

W podanym skrypcie za pomocą metody `css()` ustawione zostały właściwości `font-family` dla wszystkich elementów strony.

### Przykład 4.6

```
$(function() {
  $('#bibl').click(function() {
    $('p').css('background', 'red');
  });
});
```

W podanym skrypcie przy użyciu metody `css()` został zmieniony kolor tła dla wszystkich znaczników `<p>`.

### Ćwiczenie 4.1

Zaprojektuj na stronie internetowej blok zawierający kilka haseł związkanych z czwartą rewolucją przemysłową. Hasła mają być widoczne na stronie, natomiast opis danego hasła ma się pojawiać po kliknięciu tego hasła. Zdefiniuj odpowiednio style dla wyświetlanych bloków informacji.

## 4.1.3. Filtry selektorów jQuery

Z pomocą selektorów wybieramy określone elementy modelu DOM. Jeżeli użyjemy filtrów, otrzymamy selektory, które zwracają elementy spełniające wskazane 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 nadaniu innego koloru wierszom parzystym i nieparzystym. Do tego celu mogą zostać użyte filtry :even i :odd.

### Przykład 4.7

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Filtr even-odd</title>
<meta charset="UTF-8">
<script src="js/jquery-3.4.1.js"></script>
<script src="js/skrypt7.js"></script>
</head>
<body>
<table>
<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>
</body>
</html>
```

Kod jQuery:

```
$(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 stronie interneto-wą. Odróżnianie elementów listy poprzez ich naprzemienne kolorowanie może pomóc użytkownikom w poruszaniu się po menu.

### Przykład 4.8

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Menu</title>
<meta charset="UTF-8">
<style type="text/css">
.tlo1 {
    background: #EEE;
}
.tlo2 {
    background: #B0B0B0;
}
```

```

#menu {
    width: 100px;
}

ul {
    display: block;
    list-style-type: none;
    margin: 0;
    padding: 0;
}

</style>

<script src="js/jquery-3.4.1.js"></script>
<script src="js/skrypt8.js"></script>
</head>

<body>

<div id="menu">
<p>Menu</p>
<ul>
    <li> Start</li>
    <li> Aktualności</li>
    <li> Szkolenia</li>
    <li> Usługi</li>
</ul>
</div>
</body>
</html>

```

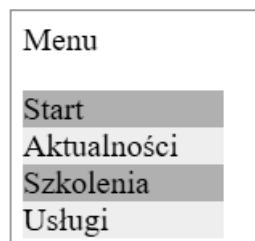
### Kod jQuery (*skrypt8*):

```

$(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, czy element ma 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 wszystkie elementy tych list zostałyby przypisane do klas `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 4.9

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

### Ćwiczenie 4.2

Utwórz menu dla strony internetowej, w którym pozycja menu wybrana przez kliknięcie myszą 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`.

### Przykład 4.10

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

Wybrane zdarzenia biblioteki jQuery zostały przedstawione w tabeli 4.4.

**Tabela 4.4.** Zdarzenia biblioteki jQuery

Nazwa zdarzenia	Opis
click ()	występuje po kliknięciu elementu myszą
mouseout ()	występuje po usunięciu kurSORA myszy z wybranego elementu
mouseover ()	występuje po umieszczeniu kurSORA myszy w obrębie wybranego elementu
hover ()	występuje po ustawnieniu kurSORA myszy nad wybranym elementem

## 4.2.2. Zdarzenia myszy

Przy użyciu zdarzeń obsługiwanych przez bibliotekę jQuery można animować menu utworzone na stronie internetowej, na przykład przez podświetlenie elementu, który pojawi się kurSOR myszy (rysunek 4.4). Po usunięciu kurSORA myszy zostanie poprzedni sposób wyświetlania elementu. Do tego celu służy zdarzenie hover (). Po najechaniu myszą na element listy naszego menu (addClass ()) , a po przesunięciu myszy poza element (removeClass ()) .

Przed przygotowaniem skryptu, który będzie powodował podświetlenie elementu, w dokumencie HTML w arkuszu stylów należy zdefiniować nową klasę (na przykład 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 charset="UTF-8">
<style type="text/css">
li.wybor {
    background: gray;
    color: #FFF;
}
#menu {
    width: 120px;
```

Menu  
Start  
Aktualności  
Szkolenia

Usługi  
Rysunek 4.4. Zdarzenie do podświetlenia elementu, nad którym pojawia się kurSOR myszy

```

}

ul {
    display: block;
    list-style-type: none;
    margin: 0;
    padding: 0;
}

</style>

<script src="js/jquery-3.4.1.js"></script>
<script src="js/skrypt11.js"></script>
</head>
<body>
<p>Menu</p>
<ul id="menu">
    <li> Start</li>
    <li> Aktualności</li>
    <li> Szkolenia</li>
    <li> Usługi</li>
</ul>
</body>
</html>

```

Kod jQuery:

```

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

```

Pierwsza funkcja w podanym kodzie dodaje klasę do elementu, druga ją usuwa. 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 na przykład do sprawdzania poprawności wypełnienia formularza, do powiadomienia użytkownika o jakimś działaniu lub do określonej akcji, 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 <i>Tab</i> 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 typu <i>submit</i>
reset ()	występuje w momencie zresetowania formularza po kliknięciu przycisku typu <i>reset</i>



## 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 pozwala wyświetlić ukrytą treść. Po kliknięciu innej opcji wyświetlana treść zostaje ukryta i wyświetla się inna (rysunek 4.6).

Język HTML
Podstawowym elementem języka HTML jest znacznik. Znaczniki są poleceńami umieszczonymi w nawiasach ostrych. Informują one przeglądarkę o wyglądzie otwieranej strony oraz o strukturze tekstu umieszczonego na niej.
Kaskadowe arkusze stylów CSS
Systemy zarządzania treścią – CMS
JavaScript
jQuery

**Rysunek 4.5.**

Menu nawigacyjne

## Przykład 4.12

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Rozwijany blok</title>
<script src="js/jquery-3.4.1.js"></script>
<script src="js/skrypt12.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;
}
```

Język HTML
Kaskadowe arkusze stylów CSS
Systemy zarządzania treścią – CMS
JavaScript
jQuery

**Rysunek 4.6.**

Wyświetlenie treści ukrytej w menu

```
.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>Systemy zarządzania treścią-CMS</h3>  
    <p>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.</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, za pomocą których można realizować często spotykane
zadania (na przykład animacje, obsługę zdarzeń, tworzenie efektów
na stronie internetowej). Jej głównymi zaletami są: łatwość użycia,
niezależność od przeglądarki i małe rozmiary otrzymanego kodu.</p>
</div>
</body>
</html>
```

### Kod jQuery:

```

$(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 wyświetlna 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 `js` `$('.blok h3').eq(0).addClass("aktywny")` odpowiada za pobranie z klasy `.blok` elementu `h3` o indeksie `0` oraz dodanie go do klasy `aktywny`. Metoda `eq(index)` wybiera element, który jest zgodny z podanym indeksem. W tym wypadku jest to pierwszy nagłówek `h3`, który po przypisaniu mu klasy `aktywny` zmieni swój wygląd. Instrukcja `$(".blok p").eq(0).show()` pobierze pierwszy element `p` i wykona na nim metodę `show()`. Jeżeli element jest ukryty, metoda ta zmienia wartość atrybutu `display` na `"block"`. Jeśli 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.

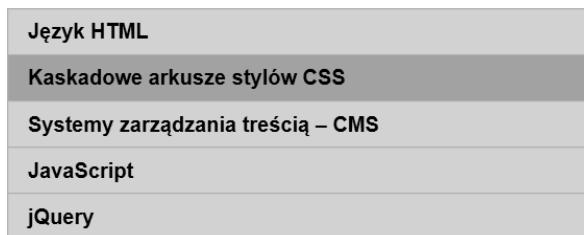
Polecenie `$(".blok h3").click(function() {})` mówi, że po kliknięciu wybranego nagłówka powinna zostać wywołana funkcja, która będzie wyświetlała sąsiadujący z nim akapit: `$(this).next("p").slideToggle("slow").siblings("p:visible").slideUp("slow")`. Metoda `next()` pobiera następny po wskazanym pasujący 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`.

Pominięcie w skrypcie `js` bloku

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

spowoduje wyświetlenie menu w postaci pokazanej na rysunku 4.7.



**Rysunek 4.7.** Menu nawigacyjne bez bloku z treścią

### 4.3.2. Proste animacje

Strony internetowe z pokazami slajdów, animowanymi menu lub innymi animacjami często są tworzone w języku JavaScript. Biblioteka jQuery korzysta z efektów dostępnych w tym języku, 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 tak samo 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 rysunku 4.8.

### Przykład 4.13

```
<!DOCTYPE HTML>
<html>
<head>
<title>Ukrywanie napisu</title>
<meta charset="UTF-8">
<style>
.blok {
    background-color: #c0c0c0;
    color: #000;
    margin: 15px;
    padding: 20px;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 40px;
    width: 300px;
    font: bold 24pt Arial, sans-serif;
}
</style>
<script src="js/jquery-3.4.1.js"></script>
<script>
$(function() {
    $("button").click(function() {
        $("#p1").css("color", "red")
            .slideUp(2000)
            .slideDown(2000);
    });
})</script>
</head>
```



**Rysunek 4.8.**  
Pokazanie i ukrywanie napisu

```
<body>  
<div id="blok" class="blok">  
  <p id="p1">Biblioteka jQuery!</p>  
</div>  
  <button>Kliknij</button>  
</body>  
</html>
```

### Ćwiczenie 4.3

Wstaw na stronę internetową zdjęcie. Utwórz animację, która będzie powodowała, że gdy użytkownik najedzie myszą na zdjęcie, zniknie ono w sposób podobny do tego, jak zniknął tekst w przykładzie 4.13. Po odsunięciu myszy zdjęcie będzie powoli pojawiało się na stronie.

### 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 (na przykład `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 w animacjach stosowane jest przejście liniowe. Ostatni parametr (opcjonalny) to funkcja zwrotna, czyli funkcja wywoływana po zakończeniu animacji.

#### Przykład 4.14

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

```
<script src="js/jquery-3.4.1.js"></script>
<script>
$(function() {
    $("button").click(function() {
        $("#blok").animate({
            width: "500px", height: "500",
        }, 1500);
    });
}</script>
</head>
<body>
<button>Start animacji</button>
<div id="blok">
</div>
</body>
</html>
```

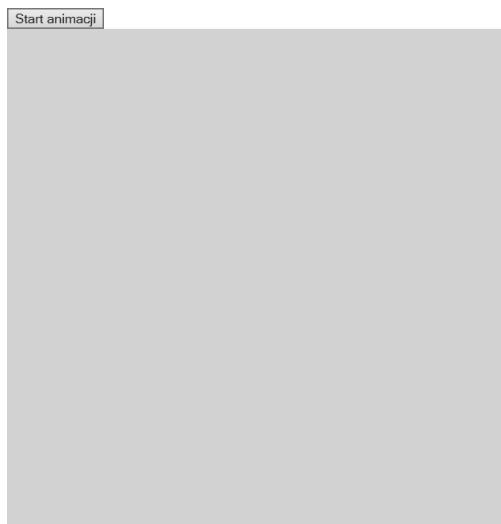
W podanym wyżej przykładzie funkcja `animate()` została wywołana dla elementu `<div>` o identyfikatorze `id="blok"`. Zmieniającymi się parametrami są wysokość i szerokość elementu. Czas trwania animacji to 1500 milisekund.

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

## Ćwiczenie 4.4

Umieść na stronie internetowej dowolny tekst. Wykorzystując metodę `animate()`, utwórz animację, która będzie powodowała, że gdy użytkownik najedzie myszą na tekst, tekst ten kilkakrotnie się powiększy (stopniowo), a po odsunięciu myszy wróci do początkowego rozmiaru.

## Zmiana rozmiaru i położenia elementu

Domyślnie każdy element dokumentu HTML ma określona 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

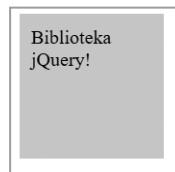
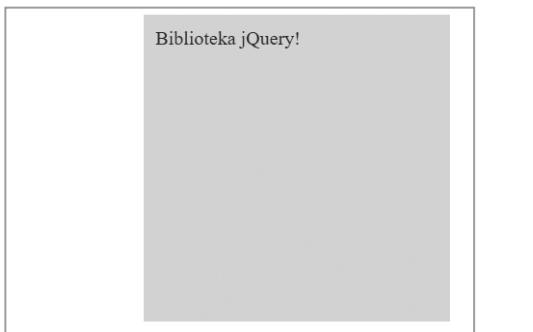
```
<!DOCTYPE HTML>
<html>
<head>
<style type="text/css">
#blok {
    width: 100px;
    height: 100px;
    background: #c6d2f2;
    position: absolute;
    padding: 10px;
}
</style>
<script src="js/jquery-3.4.1.js"></script>
<script>
$(function() {
    $("#blok").mouseover(function() {
        $(this).animate({
            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>
</head>
<body>
<div id="blok">
Biblioteka jQuery!
</div>
</body>
</html>

```

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 element ten powróci do pierwotnego położenia i rozmiaru (rysunek 4.12). Przy definiowaniu właściwości wykorzystana została możliwość ustalenia wartości względnych (`width: "+=300px"`, `height: "+=300px"`). Gdy podaje się wartości względne, przed podaną wartością należy umieścić `+=` lub `-=`.



**Rysunek 4.12.**  
Powrót do pierwotnego położenia i rozmiaru

#### Rysunek 4.11.

Zmiana rozmiaru i potożenia bloku

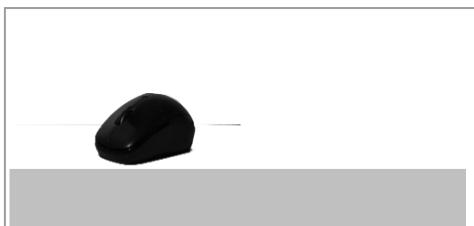
jQuery tworzy kolejki dla animacji. Dlatego jeżeli w kodzie występuje wiele wywołań tej samej metody, zostanie utworzona kolejka wywołań. Metody uruchamiane są jedna po drugiej. W podanym przykładzie kilkakrotne najechanie kurSorem myszy na obiekt i odsunięcie kurSora spowoduje kilkakrotne powtórzenie animacji.

#### UWAGA

Przy użyciu metody `animate()` można przetwarzać większość właściwości CSS. Należy pamiętać, że nazwy właściwości nie mogą zawierać znaku – (łącznik). Nazwa taka powinna mieć postać na przykład `padding-left` (a nie `padding-left`).

## Ćwiczenie 4.5

Umieść na stronie internetowej prostą grafikę, na przykład obrazek myszy. Wykorzystując metodę `animate()`, utwórz animację, która będzie powodowała, że po najechaniu myszą na grafikę obrazek zostanie przesunięty w inne miejsce, a następnie wróci do położenia początkowego (rysunki 4.13 i 4.14).



**Rysunek 4.13.**

Początkowe położenie elementu animowanego



**Rysunek 4.14.**

Końcowe położenie elementu animowanego

## Ćwiczenie 4.6

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, lub mogą wyjaśniać pewne terminy i zawierać istotne informacje.

## Przykład 4.16

Kod HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Dymki</title>
<script src="js/jquery-3.4.1.js"></script>
<script src="js/skrypt16.js"></script>
<style type="text/css">
body {
    background: #c0c0c0;
    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: 118px;
    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 młodzi 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:

```
$(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 4.15). Po wybraniu elementu pojawi się okienko z podpowiedź (rysunek 4.16).



**Rysunek 4.15.** Menu umieszczone na stronie internetowej



**Rysunek 4.16.** Wyświetlenie podpowiedzi do menu

Działanie kodu jQuery jest następujące: po wskazaniu myszą elementu menu `$(".menu a") .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 góre w stosunku do jego pierwotnego położenia i ukryje wcześniej wyświetlany element `p`.

### Ćwiczenie 4.7

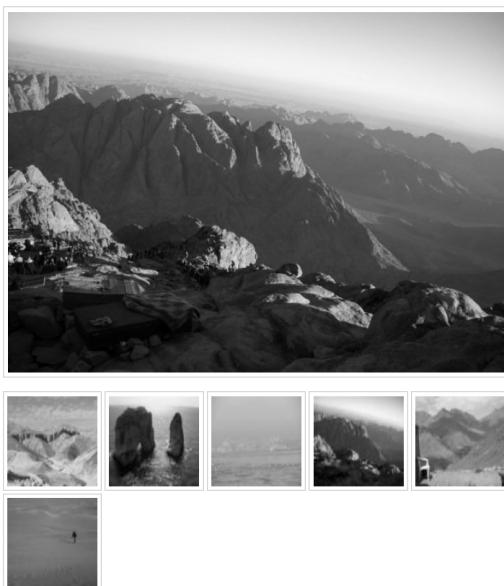
Wykorzystując przykład 4.16 i poznane narzędzia projektowania stron internetowych, zaprojektuj dla swojej strony internetowej menu z podpowiedziami w postaci dymków pojawiąjących się po ustawieniu myszy nad wybraną opcją tego menu.

## 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 4.17 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. Aby je przygotować, można wykorzystać na przykład program Web Album Generator, który generuje miniatury zdjęć oraz tworzy zdjęcia o określonych rozmiarach, najczęściej mniejszych od rozmiarów zdjęć oryginalnych.

Pokaz zdjęć (Zdjęcie 4)



**Rysunek 4.17.**

Pokaz zdjęć na stronie internetowej

Po przygotowaniu zdjęć i umieszczeniu ich w folderze *img* istotne jest takie zdefiniowanie stylów CSS, aby cały pokaz dobrze prezentował się na stronie internetowej.

## Przykład 4.17

```
<!DOCTYPE HTML>

<head>
<meta charset="UTF-8">
<title>Pokaz zdjęć</title>
<script src="js/jquery-3.4.1.js"></script>
<script>
$(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>  


## Pokaz zdjęć



</p>




</p>



</body>



</html>


```

W skrypcie jQuery została użyta metoda `append()`, która do nagłówka `h2` będzie dołączała 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 odpowiada za 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 3.11. Stanie się ono łatwiejsze dzięki zastosowaniu biblioteki jQuery oraz wtyczki Validation opracowanej dla tej biblioteki. Plik wtyczki można pobrać ze strony <https://jqueryvalidation.org/>. Po jego rozpakowaniu do walidacji formularza potrzebne będą tylko pliki: `jquery.validate.js`, zawierający validator, oraz `messages_pl.js`, zawierający standardowe komunikaty walidacji zapisane w języku polskim. Pliki te kopiujemy i umieszczamy w dowolnym folderze z kodem strony. Na przykład 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, wpisujemy 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 charset="UTF-8">
<title>Walidacja formularza</title>
<script src="js/jquery-3.4.1.js"></script>
<script src="js/validate/jquery.validate.js"></script>
<script src="js/validate/messages_pl.js"></script>
<script>
jQuery(function() {
    jQuery("#formularz").validate();
});
</script>
<style type="text/css">
```

```
body {  
    font-family: Arial, Helvetica, sans-serif;  
    font-size: 12px;  
    font-weight: normal;  
}  
  
#formularz {  
    width: 450px;  
    background-color: #c0c0c0;  
}  
  
</style>  
  
</head>  
  
<body>  
  
<form id="formularz" action="" 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" maxlength="11"></div><br>  
  
<label><b> Pracuję: </b><input type="checkbox" name="opcje" id="opcje" checked="checked" value="1" maxlength="1">  
</label><br><br>  
  
<b>Wiadomość:</b><br>  
  
<div><textarea id="inf" cols="30" name="inf" rows="5" class="required" minlength="5"></div><br><br>
```

```

<input class="submit" type="submit" value="Wyślij"/>&nbsp; &nbsp;
<input type="reset" value="Wyczyść"/>
</fieldset>
</form>
</body>
</html>

```

Efekt działania powyższego kodu pokazany jest na rysunku 4.18. Komunikaty pojawiają się automatycznie po błędny wypełnieniu pól formularza.

The screenshot shows a web form titled "Dane osobowe". It contains several input fields with validation messages:

- Nazwisko:** "No" - "Proszę o podanie przynajmniej 3 znaków."
- Imię:** "Anna"
- Zawód:** Empty field - "To pole jest wymagane."
- Adres e-mail:** "anna@gmail" - "Proszę o podanie prawidłowego adresu email."
- Telefon:** "344 567 23" - "Proszę o podanie prawidłowej liczby."
- Pracuję:** checked checkbox
- Wiadomość:** Empty text area - "To pole jest wymagane."

At the bottom are two buttons: "Wyślij" and "Wyczyść".

**Rysunek 4.18.** Weryfikacja formularza

W podanym przykładzie w pierwszej linii kodu po znaczniku `<title>` zostało zapisane dołączenie biblioteki jQuery, następnie dołączenie wtyczki Validate, w kolejnej dołączeniu 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 **Adres e-mail:** też jest obowiązkowe i powinno zostać sprawdzone pod kątem poprawnego adresu e-mail. W przypadku kolejnego pola, **Telefon:**, należy zweryfikować, 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 validatorze. 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, stosuje się atrybut `minlength` (na przykład `minlength="2"`). Do weryfikacji poprawności adresu e-mail jest wykorzystywana klasa `email(class="email")`, a do weryfikacji numeru telefonu klasa `number(class="number")`.

### Ćwiczenie 4.8

Wykorzystując kod z przykładu 4.18, utwórz na swojej stronie internetowej formularz rejestracyjny z podstawowymi danymi użytkownika. Przy użyciu wtyczki `Validation` przeprowadź validację formularza. Za pomocą kodu HTML i arkuszy CSS popraw wygląd utworzonego formularza.



## 4.5. Pytania i zadania

### 4.5.1. Pytania

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`?

### 4.5.2. Zadania

#### Zadanie 1.

Zbuduj stronę internetową z informacjami 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 określony 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 validację formularza, dzięki której sprawdzisz, czy użytkownik wypełnił wymagane pola oraz prawidłowo wpisał dane.



# 5

# Inne biblioteki języka JavaScript

## 5.1. Wprowadzenie

Biblioteki języka JavaScript ułatwiają tworzenie stron internetowych. Niektóre z nich zawierają narzędzia, pozwalające uatrakcyjnić strony internetowe, inne wspierają pracę z kodem, a jeszcze inne umożliwiają tworzenie aplikacji typu SPA. Do najpopularniejszych należą: jQuery, React, Angular. Są to biblioteki open source. Każda z nich zawiera specyficzny dla siebie zestaw funkcji, między innymi usprawniających pracę z modelem DOM.

Niektóre biblioteki, takie jak Angular i React.js, wymagają przygotowania odpowiedniego środowiska, aby można było tworzyć aplikację opartą na gotowym szablonie. W tym celu należy zainstalować wiele komponentów i uruchomić je, korzystając z komend wiersza poleceń.

Niektóre z komponentów potrzebnych do prawidłowej pracy z bibliotekami Angular i React.js zostały przedstawione poniżej.

### 5.1.1. Środowiska uruchomieniowe

#### Visual Studio Code

Visual Studio Code to bezpłatny edytor programistyczny stworzony przez firmę Microsoft. Jest dostępny dla użytkowników systemów Windows, Linux oraz macOS. Charakteryzuje się bardzo dobrze opracowanym interfejsem graficznym oraz lekkością i szybkością działania. Oferuje narzędzia do podpowiadania składni kodu, wbudowany debugger oraz dużą liczbę rozszerzeń. Wspiera bibliotekę React, Angular, język TypeScript oraz inne frameworki i języki.

## Node.js

Node.js jest środowiskiem uruchomieniowym do tworzenia aplikacji typu klient-serwer pisanych w języku JavaScript. Umożliwia budowanie aplikacji WWW za pomocą języka JavaScript i uruchamianie ich w środowisku innym niż przeglądarka. Tworzenie aplikacji opiera się na programowaniu sterowanym zdarzeniami, a używanie tego samego języka programowania po stronie klienta i serwera pozwala uniknąć duplikowania logiki fragmentów aplikacji.

Node.js oferuje wiele modułów stanowiących rozszerzenie dostępnych narzędzi programistycznych. Działa na wielu platformach jako środowisko uruchomieniowe o otwartym kodzie.

## NPM

NPM zarządza pakietami środowiska Node.js. Jest to aplikacja wiersza poleceń, za pomocą której można doinstalowywać aplikacje dostępne w repozytorium NPM. NPM jest instalowany razem z Node.js.

## TypeScript

TypeScript to skryptowy język programowania stworzony przez firmę Microsoft, który jest nadzbiorem (nakładką) języka JavaScript. Oznacza to, że każdy program napisany w języku JavaScript jest również programem napisanym w języku TypeScript. Aplikacje pisane w języku TypeScript przed wykonaniem są kompilowane do języka JavaScript. Jest to język silnie typowany, czyli można w nim deklarować zmienne, które przyjmują wartości tylko określonego typu. Ale można tworzyć aplikacje w języku JavaScript i napisany kod również zostanie skompilowany. Zadaniem języka TypeScript jest ułatwienie pisania kodu w języku JavaScript, sprawdzanie typów i wykrywanie błędów.

## 5.2. Angular

### 5.2.1. Wprowadzenie

Angular to wspierana przez Google otwarta biblioteka języka JavaScript napisana w języku TypeScript, służąca do tworzenia dynamicznych aplikacji internetowych. Z powodu rozbudowanych funkcjonalności często jest określana mianem platformy programistycznej. Oferuje takie funkcjonalności jak definiowanie struktury aplikacji i sposobu jej działania oraz dostarcza wiele różnych komponentów i bibliotek. Wspomaga tworzenie aplikacji internetowych na pojedynczej stronie (ang. *Single Page Application* — SPA). Zadaniem biblioteki jest wdrożenie wzorca *Model-View-Controller* (MVC) do aplikacji internetowych. Ma własny kompilator HTML, co pozwala ingerować w powstający kod HTML, a nawet tworzyć własne znaczniki. Angular zaleca korzystanie z języka TypeScript.

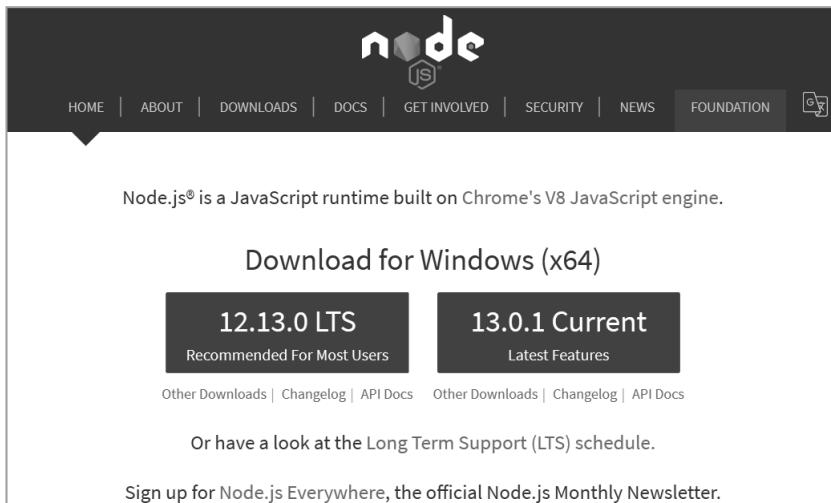
Angular początkowo był planowany jako druga wersja frameworka AngularJS, jednak ze względu na brak kompatybilności wstępnej jest rozwijany jako samodzielny projekt. W założeniu jego kolejne wersje powinny pojawiać się co sześć miesięcy. Obecnie dostępna wersja wydana w maju 2019 roku to Angular 8. Angular 8 obsługuje TypeScript 3.4.

## Single Page Application — SPA

Głównym założeniem modelu **Single Page Application** (SPA) jest zbudowanie aplikacji opartej na jednostronicowym interfejsie oraz przeniesienie logiki aplikacji z serwera do klienta. Logika aplikacji jest tworzona po stronie klienta w języku JavaScript i wykonywana w przeglądarce. Odwołania do serwera odbywają się asynchronicznie, a kod HTML, CSS i JavaScript jest pobierany tylko raz w trakcie uruchomienia aplikacji.

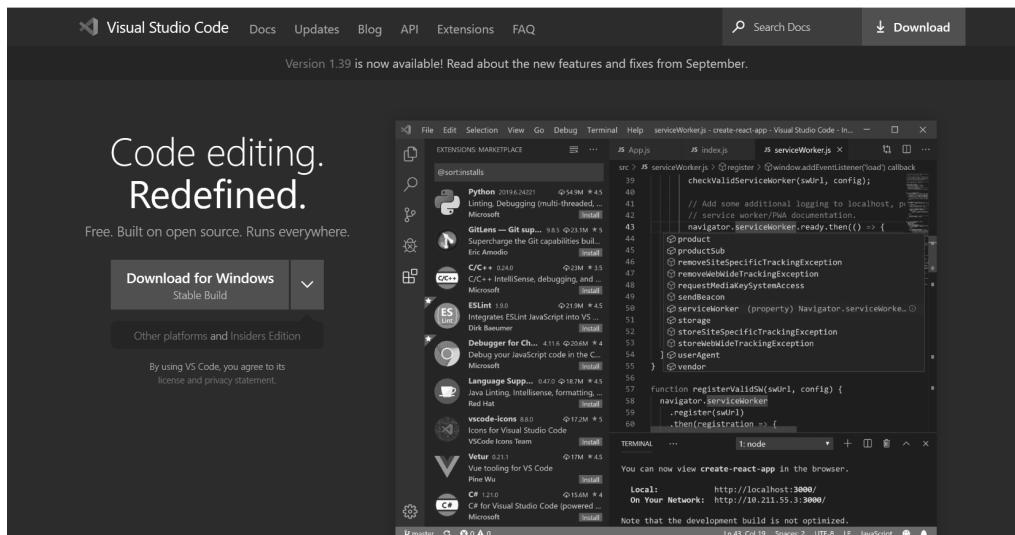
### 5.2.2. Środowisko pracy

Do pracy z aplikacją Angular potrzebny jest między innymi pakiet Node.js. Program najlepiej pobrać ze strony <https://nodejs.org/en>, wybierając wersję LTS (rysunek 5.1).



**Rysunek 5.1.** Strona pakietu Node.js

Kolejnym narzędziem pomagającym w pracy z aplikacją Angular jest edytor. Odpowiednim edytorem może być Visual Studio Code. Jest to darmowe narzędzie wspierające bibliotekę Angular i język TypeScript. Można je pobrać ze strony <https://code.visualstudio.com> (rysunek 5.2).



**Rysunek 5.2.** Strona pakietu Visual Studio Code

Zainstalowany edytor Visual Studio Code ma wiele dodatków związanych z frameworkm Angular. Jednym z nich jest Angular Extension Pack, który wspiera pracę z biblioteką Angular i językiem TypeScript. Po uruchomieniu Visual Studio Code należy z lewej strony odnaleźć ikonę *Extensions*, wpisać nazwę dodatku, znaleźć go i zainstalować (rysunek 5.3).



**Rysunek 5.3.** Instalacja dodatku Angular Extension Pack

Po zainstalowaniu Node.js i przed przystąpieniem do tworzenia aplikacji można sprawdzić, czy wszystkie pakiety działają prawidłowo. W tym celu w wierszu poleceń należy wpisać polecenie:

```
node -v
```

Po wprowadzeniu podanego kodu powinna zostać wyświetlona wersja Node.js.

Razem z Node.js instalowany jest pakiet NPM. Można to sprawdzić, wpisując polecenie:

```
npm -v
```

Powinna zostać wyświetlona wersja NPM (rysunek 5.4).

```
Microsoft Windows [Version 10.0.17134.1006]
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\dell>node -v
v12.13.0

C:\Users\dell> npm -v
6.12.0

C:\Users\dell>
```

**Rysunek 5.4.** Informacja o wersji Node.js i NPM

Dzięki aplikacji NPM można doinstalowywać pakiety potrzebne do pracy z Angulariem. Jednym z nich jest Angular CLI. Jest to narzędzie, które zarządza projektem, pomaga w jego budowaniu i tworzeniu nowych elementów. Można je pobrać ze strony <https://cli.angular.io/>. Po wejściu na tę stronę i kliknięciu przycisku **GET STARTED** następuje przekierowanie do witryny <https://angular.io/cli>, gdzie został opisany sposób instalacji pakietu oraz sposób posługiwania się nim.

Aby zainstalować pakiet, należy w wierszu poleceń wpisać polecenie:

```
npm install -g @angular/cli
```

By uzyskać pomoc i listę poleceń, trzeba wpisać polecenie:

```
ng help
```

Aby utworzyć nowy projekt, należy w wierszu poleceń wpisać:

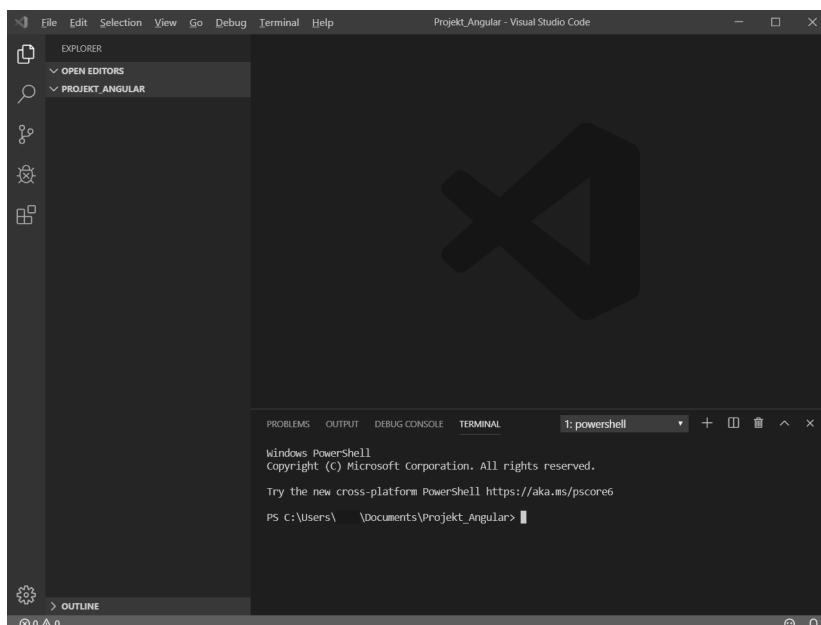
```
ng new nowy-projekt
cd nowy-projekt
ng serve
```

## Ćwiczenie 5.1

Utwórz nowy projekt do pracy z Angular CLI. Nazwij go *ProjektA*.

## Rozwiążanie

Przed przystąpieniem do pracy przygotuj folder, w którym zostanie zapisany projekt, na przykład w folderze *Dokumenty* utwórz folder *Projekt\_Angular*. Otwórz edytor Visual Studio Code, w menu kliknij *File/Open Folder...* i wybierz utworzony folder *Projekt\_Angular*. W tym samym edytorze otwórz konsolę wiersza poleceń, klikając *View/Terminal*. W terminalu zostanie pokazany folder, który został wybrany do pracy (rysunek 5.5).



**Rysunek 5.5.** Okno konsoli w edytorze Visual Studio Code

Można przystąpić do tworzenia projektu.

W wierszu poleceń wpisz polecenie:

```
ng new ProjektA
```

Po uruchomieniu polecenia pojawi się pytanie:

```
Would you like to add Angular routing? (y/N)
```

W Angularze występują komponenty, które mają przypisane różne zadania. Routing umożliwia użytkownikowi uzyskanie dostępu do tych komponentów lub stron poprzez hiperłącza w HTML, wykorzystanie metody nawigacji JavaScript lub poprzez wklejenie adresu w polu adresu przeglądarki.

Powstająca aplikacja będzie miała zaimplementowany routing, więc naciśnij klawisz *y*.

Rozpocznie się tworzenie projektu i generowanie potrzebnych zasobów. W wyniku otrzymamy folder *ProjektA* z aplikacją.

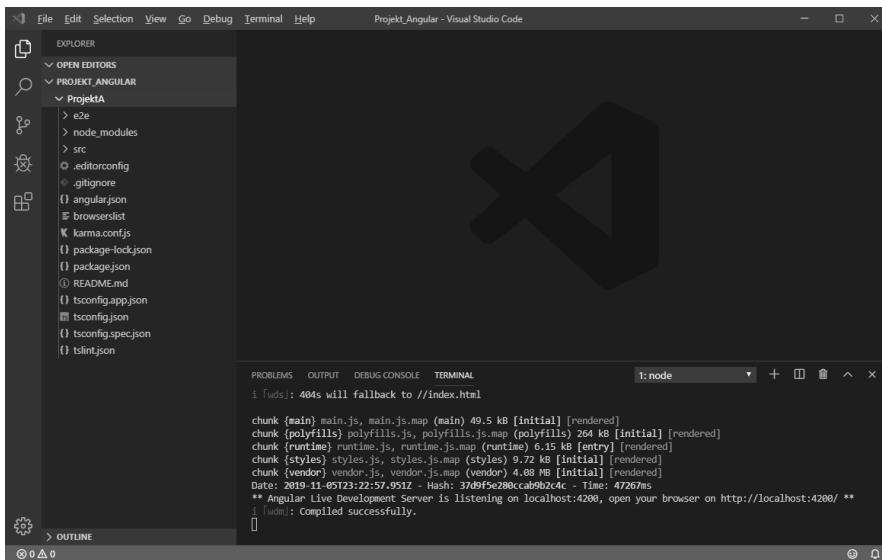
Przejdz do tego folderu, wpisując w wierszu polecień:

```
cd ProjektA
```

i uruchom projekt, wpisując polecenie:

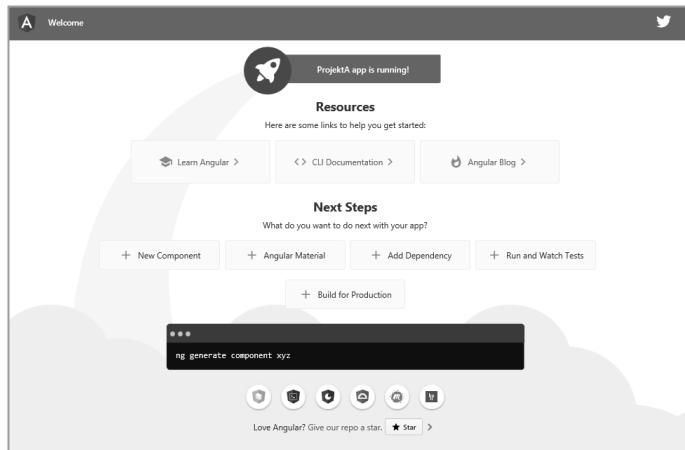
```
ng serve
```

które uruchomi wygenerowaną w Angular CLI aplikację. Aplikacja będzie dostępna pod adresem *localhost:4200* (rysunek 5.6).



**Rysunek 5.6.** Efekt uruchomienia projektu w Visual Studio Code

Ustaw kursor myszy na komunikacie z adresem <http://localhost:4200/>, wciśnij klawisz **Ctrl** i kliknij podany link. W przeglądarce internetowej zostanie otwarty wygenerowany projekt (rysunek 5.7).



**Rysunek 5.7.** Projekt wygenerowany przez Angular CLI

Ten sam rezultat uzyskasz, wykonując polecenie:

```
ng serve --open
```

Opcja `--open` spowoduje automatyczne otwarcie projektu w przeglądarce. Inne opcje polecenia `serve` można odczytać po wykonaniu komendy:

```
ng serve --help
```

### 5.2.3. Struktura projektu

Strukturę wygenerowanego projektu (foldery i pliki) można zobaczyć w oknie Visual Studio Code w eksploratorze plików tworzonego projektu. W dostępnych folderach znajdują się pliki źródłowe aplikacji. Każda aplikacja ma folder `src`, który zawiera logikę, dane i zasoby.

Wygenerowane pliki można edytować za pomocą polecień CLI.

Podstawowe pliki to `index.html` i `main.ts` znajdujące się w folderze `src`. Plik `index.html` zawiera widok startowy, a `main.ts` to plik odpowiedzialny za import głównego modułu aplikacji. Ważnym folderem jest `src/app`, w którym znajdują się wszystkie komponenty składające się na tworzoną aplikację. Komponenty wyświetlają dane na ekranie, nasłuchują informacji wprowadzanych przez użytkownika i podejmują działania na ich podstawie.

Implementacja powłoki aplikacji znajduje się w plikach:

- `app.component.ts` — klasa komponentu napisana w języku TypeScript,
- `app.component.html` — szablon komponentu napisany w HTML,
- `app.component.css` — style CSS dla komponentu.

Inne pliki konfiguracyjne utworzonego projektu to:

- `angular.json` — główny plik konfiguracyjny projektu; można w nim ustawić wartości domyślne dla projektu i konfigurację, która będzie używana podczas budowy projektu,
- `.editorconfig` — plik konfiguracyjny Visual Studio Code,
- `package.json` — zbiór wszystkich paczek i zależności wykorzystywanych w projekcie.

### Ćwiczenie 5.2

Zmień tytuł tworzonej aplikacji na *Moja aplikacja w Angularze*.

#### Rozwiążanie

W pliku `app.component.ts` znajduje się definicja elementów strony startowej, w tym także jej nagłówka w postaci:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'ProjektA';
}
```

Właściwość `title` klasy `AppComponent` zawiera nazwę tworzonej aplikacji.

Zmień jej wartość na `Moja aplikacja` w Angularze.

Odśwież w przeglądarce stronę z wygenerowanym projektem.

## 5.3. React.js

### 5.3.1. Wprowadzenie

React to biblioteka języka JavaScript opracowana przez Facebook do tworzenia interaktywnych interfejsów użytkownika (UI). Służy do budowania aplikacji jednostronnicowych lub mobilnych. Pozwala tworzyć komponenty interfejsu użytkownika, które można wielokrotnie wykorzystywać. React tworzy wirtualny DOM w pamięci i na nim wykonuje niezbędne działania przed dokonaniem zmian w DOM przeglądarki.

Do pracy z biblioteką React potrzebne są między innymi pakiety Node.js i NPM. Oprócz tego do działania React w przeglądarce potrzebne są biblioteki `react.js` i `react-dom.js` oraz `Babel.js`. Biblioteka `react.js` zawiera wszystkie funkcje React, natomiast `react-dom.js` odpowiada za renderowanie efektów w przeglądarce. Biblioteka `Babel.js` wspiera używanie składni JSX (HTML wewnętrz języka JavaScript) i ES6 w przeglądarkach.

Biblioteka Babel nie działa automatycznie, dlatego kod, który ma być przez nią obsługiwany, musi zostać oznaczony za pomocą atrybutu `type`. Dla atrybutu `type` można użyć wartości `text/babel` lub `text/jsx`.

#### Przykład 5.1

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Komponenty w React.js</title>
<script src="https://unpkg.com/react@16/umd/react.production.min.js">
</script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
<div id="blok"></div>
<script type="text/babel">
  ReactDOM.render(
```

```

<h1>Biblioteka React.js</h1>,
document.getElementById('blok')

);

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

```

W podanym przykładzie została użyta funkcja `ReactDOM.render()`, która pobiera kod napisany w JSX. Kod ten zawiera informację o tym, co ma być wyświetlane (`Biblioteka React.js`) i gdzie ma zostać wyświetlona renderowana aplikacja (wewnętrz elementu o `id="blok"`).

Powstały kod można zapisać w pliku `index.html` i uruchomić w celu przetestowania. Do celów użytkowych należy jeszcze skonfigurować środowisko React.

### 5.3.2. Środowisko React

Aby zbudować aplikację React, po zainstalowaniu pakietów Node.js i NPM należy zainstalować środowisko `create-react-app`. Pozwala ono na łatwe stworzenie projektu opartego na `react.js`. Tworzy strukturę katalogów i plików, zawiera wszystkie narzędzia potrzebne do budowania aplikacji i pozwala na jej uruchamianie i testowanie.

W celu dodania tego środowiska w wierszu poleceń należy wpisać:

```
npm install -g create-react-app
```

Po zainstalowaniu środowiska można przystąpić do budowania aplikacji. Aby utworzyć nowy projekt (na przykład o nazwie `projekt1`), należy w wierszu poleceń wpisać:

```
npx create-react-app projekt1
```

Powstanie nowy projekt, który można uruchomić, wpisując kolejno polecenie:

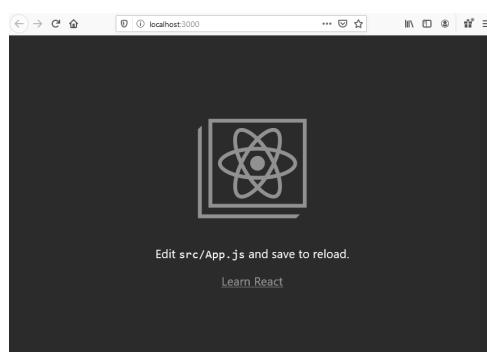
```
cd projekt1
```

które pozwoli na przejście do folderu z utworzonym projektem, oraz polecenie:

```
npm start
```

które uruchomi aplikację `projekt1`.

W oknie przeglądarki zostanie wyświetlona powstała aplikacja (rysunek 5.8). Jeżeli automatyczne wyświetlenie aplikacji nie nastąpi, należy samodzielnie otworzyć przeglądarkę i w pasku adresu wpisać `localhost:3000`.



**Rysunek 5.8.**

Okno projektu utworzonego w React

### 5.3.3. Modyfikacja aplikacji React

Istniejący projekt można modyfikować. W tym celu należy odnaleźć folder projektu (*projekt1*), a w nim folder *src*. W folderze *src* znajduje się plik o nazwie *App.js* z następującą zawartością:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Zawartość tego pliku może zostać zmieniona. Wprowadzone zmiany będą od razu widoczne w oknie przeglądarki.

## Ćwiczenie 5.3

Zmień tytuł tworzonej aplikacji na *Nowy projekt React*.

### Rozwiązanie

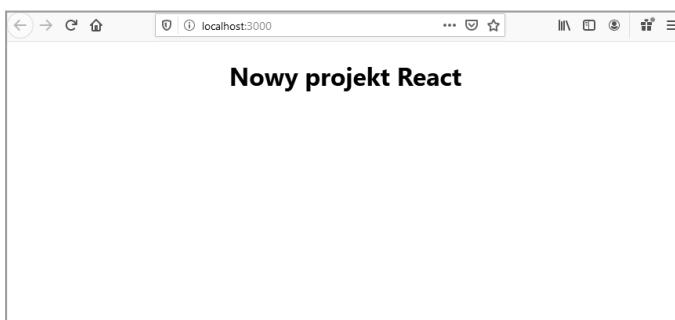
W pliku *App.js* zmień zawartość zapisaną w znaczniku `<div className="App">` na *Nowy projekt React*. Usuń również niepotrzebny import `logo.svg` i `App.css`.

```
import React from 'react';

function App() {
  return (
    <div className="App">
      <h1>Nowy projekt React</h1>
    </div>
  );
}

export default App;
```

Po modyfikacji kodu zmianie ulegnie zawartość wyświetlana w oknie przeglądarki (rysunek 5.9).



**Rysunek 5.9.** Zmieniona zawartość wyświetlona w oknie przeglądarki

### 5.3.4. React a ES6 i JSX

#### ES6

**ECMAScript** to standard obiektowego języka programowania, którego jedną z implementacji jest JavaScript. **ES6** jest szóstą wersją ECMAScript, która została opublikowana w 2015 roku i jest również znana pod nazwą **ECMAScript 2015**.

Korzystanie z modułów ECMAScript 2015 jest preferowanym sposobem pracy z biblioteką React.

## JSX

JSX lub **JavaScript XML** jest rozszerzeniem składni języka JavaScript. Komponenty biblioteki React zwykle są pisane przy użyciu JSX (choć mogą być również napisane w czystym języku JavaScript).

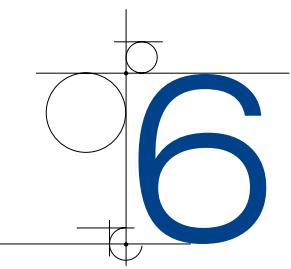


## 5.4. Pytania i zadania

### 5.4.1. Pytania

1. Wymień popularne biblioteki języka JavaScript.
2. Jakie środowiska uruchomieniowe są niezbędne do pracy z biblioteką Angular?
3. Jaki elementy strony internetowej można budować za pomocą biblioteki React?





# Język PHP

## 6.1. 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 ciasteczek (cookies). Obsługuje wiele protokołów sieciowych, w tym SMTP, POP3, IMAP, NNTP. Współpracuje z większością dostępnych systemów operacyjnych, między innymi z różnymi wersjami Linuksa, systemem Windows i macOS, oraz z 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, by zwiększyć 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, potrzebne są: parser PHP, serwer WWW oraz przeglądarka internetowa.
- Skrypty wywoływanego z wiersza poleceń — skrypty są tworzone z wykorzystaniem parsera PHP i wywoływanego w linii poleceń. Mogą być użyte również do prostego przetwarzania tekstu.
- Aplikacje po stronie klienta — zaawansowane funkcje PHP umożliwiają 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.

## 6.1.1. Opis języka

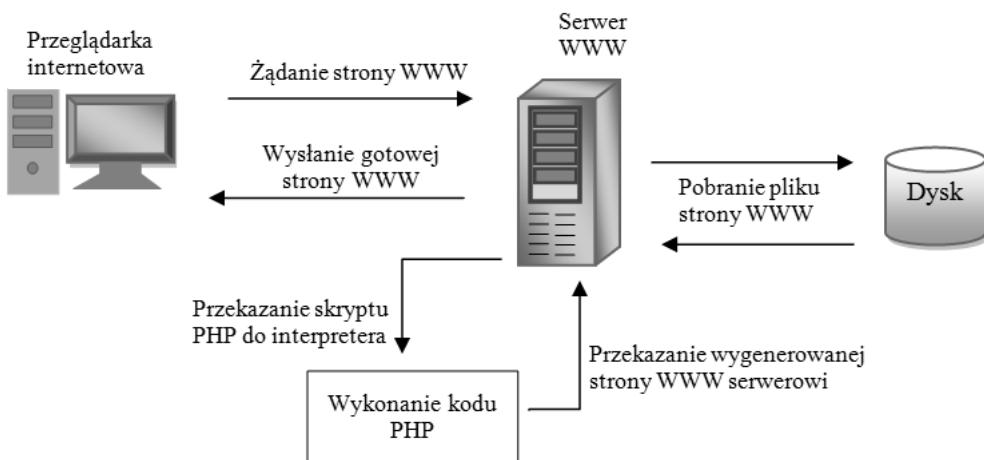
PHP jest skryptowym językiem programowania, za pomocą którego można tworzyć aplikacje WWW. Używając PHP, można:

- 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,
- szifrować dane.

## 6.1.2. PHP w HTML

Skrypty pisane w języku PHP są umieszczane wewnętrz 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 6.1.



**Rysunek 6.1.** Proces przetwarzania kodu PHP



## 6.2. Struktura języka PHP

Skrypty PHP są plikami tekstowymi, dlatego do ich pisania można wykorzystać dowolny edytor tekstu. Aby pliki tekstowe były prawidłowo rozpoznawane przez serwer WWW, muszą mieć rozszerzenie `.php`.

### 6.2.1. 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 6.1).

**Tabela 6.1.** Znaczniki bloku PHP

Rodzaj znacznika	Znacznik otwierający	Znacznik zamykający
Standardowy	<?php	?>
Skrócony	<?	?>

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.

#### Przykład 6.1

```
<?
echo "Mój pierwszy skrypt PHP";
?>
```

#### Przykład 6.2

```
<?php
echo "Mój pierwszy skrypt PHP";
?>
```

Użyta w przykładach instrukcja `echo` służy do wyświetlania danych. Instrukcja `echo` może być stosowana w sposób podany w przykładach 6.1 i 6.2 lub z użyciem nawiasów okrągłych. Obydwa sposoby jej zapisu są równoważne. Natomiast wyświetlany ciąg znaków zawsze musi być ujęty w cudzysłowy lub apostrofy.

#### Przykład 6.3

```
<?php
echo ("Mój pierwszy skrypt PHP");
?>
```

Tak samo jak instrukcja echo działa instrukcja print. Może być ona używana wyłącznie z instrukcją echo. Ciągi znaków przekazane do instrukcji print muszą być umieszczone w cudzysłowach lub apostrofach.

### Przykład 6.4

```
<?php  
print("Mój pierwszy skrypt PHP");  
?>
```

Polecenia języka PHP muszą kończyć się średnikiem. Wyjątkiem są instrukcje zawierające inne instrukcje oraz instrukcje kończące blok kodu. Brak średnika na końcu polecenia powoduje sygnalizację błędu składni w kodzie PHP.

## 6.2.2. Blok PHP w kodzie HTML

Kod zapisany w języku PHP może zostać wstawiony bezpośrednio do kodu HTML.

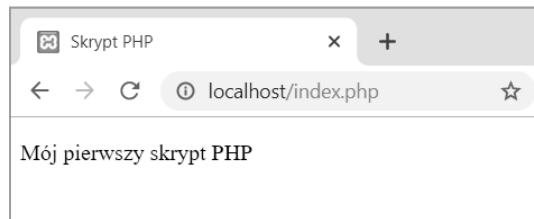
### Przykład 6.5

```
<!DOCTYPE HTML>  
  
<html>  
  <head>  
    <title>Skrypt PHP</title>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <p>  
      <?php  
        echo "Mój pierwszy skrypt PHP";  
      ?>  
    </p>  
  </body>  
</html>
```

W przykładzie 6.5 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 treść dokumentu została zapisana w innym katalogu, na przykład w katalogu `htdocs` został utworzony katalog `pliki_php`, to w przeglądarce należy wpisać adres `http://localhost/pliki_php/index.php`.

Jeśli wyświetlona strona wygląda tak jak na rysunku 6.2, to skrypt działa prawidłowo.



**Rysunek 6.2.** Efekt działania skryptu

Można teraz obejrzeć kod źródłowy wyświetlonej strony. W tym celu należy w jej obszarze kliknąć prawym przyciskiem myszy i wybrać opcję `Źródło`, `Wyświetl źródło strony` lub `Źródło strony`. Zostanie pokazany kod HTML wyświetlonej strony (rysunek 6.3).

```

1 <!DOCTYPE HTML>
2 <html>
3 <head>
4 <title>Skrypt PHP</title>
5 <meta charset="UTF-8">
6 </head>
7 <body>
8 <p>
9 Mój pierwszy skrypt PHP</p>
10 </body>
11 </html>
12

```

**Rysunek 6.3.** 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 przedstawionym 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 wynik został zwrócony do przeglądarki jako zwykły HTML.

## 6.3. 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.

### 6.3.1. Komentarze

W skrypcie PHP można umieszczać komentarze. Są one widoczne w kodzie źródłowym skryptu, natomiast w trakcie jego przetwarzania są usuwane i nie są wyświetlane 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 ciągnie się do końca linii.

### 6.3.2. Zmienne

Zmienne w programowaniu służą do przechowywania danych i wyników wykonywanych operacji. Wartością zmiennej może być liczba, ciąg znaków, tablica, obiekt. Zmienna musi mieć 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 określić jej typu.

#### Przykład 6.6

```
<?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 6.4.

Występujący w przykładzie zapis:

```
echo "Bieżąca wartość to $ilość!";
```

jest równoważny zapisowi:

```
echo "Bieżąca wartość to " . $ilość . "!";
```

```
Wynik wynosi 1
To jest tekst
345
Bieżąca wartość to 4!
```

**Rysunek 6.4.**

Rezultat wykonania kodu z przykładu

## Zakres zmiennych

Zmienne mogą być deklarowane w dowolnym miejscu skryptu.

Zmienne deklarowane wewnętrz skryptu lub funkcji są zmiennymi lokalnymi i istnieją tylko w obrębie danego skryptu albo funkcji. Do zmiennych lokalnych nie można odwoływać się w innych skryptach lub funkcjach.

Zmienne zadeklarowane poza funkcją mają zasięg globalny i można uzyskać do nich dostęp tylko poza funkcją.

## Zmienne predefiniowane

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.
- `$_REQUEST` — jest to tablica zawierająca zmienne przekazane do skryptu przez użytkownika. Obejmuje dane z `$_GET`, `$_POST` oraz `$_COOKIE`.

- `$_SESSION` — jest to tablica zawierająca zmienne zarejestrowane w bieżącej sesji.
- `$GLOBALS` — jest to tablica zawierająca odniesienie do każdej zmiennej utworzonej przez użytkownika, która ma zasięg globalny dla danego skryptu.

### 6.3.3. 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 na typ `float`.

### Przykład 6.7

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  $-1,8 \times 10^{308}$  do  $1,8 \times 10^{308}$ . Do zapisu liczby można używać notacji wykładniczej.

### Przykład 6.8

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 wykorzystany jako jeden ze znaków ciągu, to należy poprzedzić go znakiem \.

### Przykład 6.9

```
<?php
echo 'To jest symbol apostrofu \' użyty w kodzie PHP';
?>
```

## Przykład 6.10

```
<?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 zamienione na wartość przypisaną do zmiennej.

## Przykład 6.11

```
<?php
$x = "To jest tekst";
echo "$x";
?>
```

### Użycie składni heredoc

Jeżeli wykorzystywana jest składnia heredoc, ciąg znaków trzeba rozpoczęć od sekwencji <<<. Po tej sekwencji musi wystąpić identyfikator i nowy wiersz, w którym wprowadzany jest ciąg znaków. Ten sam identyfikator musi zostać użyty na końcu ciągu znakowego i zapisany w nowej linii. Nazwa identyfikatora powinna być tworzona według zasad, które obowiązują przy tworzeniu nazw zmiennych. Linia zawierająca identyfikator kończący ciąg nie może zawierać żadnych innych znaków oprócz tego identyfikatora i średnika.

## Przykład 6.12

```
<?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 zamienione na wartość przypisaną do zmiennej. Ten sposób tworzenia napisów jest wykorzystywany, kiedy są one długie i składają się z wielu linii.

## Przykład 6.13

```
<!DOCTYPE HTML>
<html>
<head>
<title>Skrypt PHP</title>
<meta charset="UTF-8">
</head>
<body>
<?php
$imie = "Anna";
$jazyk1 = "PHP";
$jazyk2 = "JavaScript";
$tekst = <<<TX
    Mam na imię $imie.
    Uczę się programować w języku $jazyk1.
    Umiem już programować w $jazyk2.
TX;
echo $tekst;
?>
</body>
</html>
```

## Użycie składni nowdoc

Składnia nowdoc jest podobna do składni heredoc. Różnica w definicji typu polega na umieszczeniu identyfikatora pomiędzy znakami apostrofu. Wszystkie wymagania dotyczące identyfikatora są takie same jak dla heredoc. Różnica w działaniu polega na tym, że gdy w ciągu znaków wystąpi odwołanie do zmiennej, nie jest ona zamieniana na odpowiadającą jej wartość.

## Przykład 6.14

```
<?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. Wartościami elementów tablic mogą być tablice, dlatego możliwe jest tworzenie tablic wielowymiarowych.

Struktury tablicowe w PHP są tworzone na bieżąco przez przypisanie wartości do poszczególnych indeksów albo za pomocą funkcji `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.

Funkcja `array` dla tablic indeksowanych tworzonych przez automatyczne przypisanie indeksu ma postać:

```
$tab = array(t1, t2, ..., tn);
```

lub

```
$tab = [t1, t2, ..., tn];
```

Powstała zmienna jest zmienną typu tablicowego, natomiast `t1, t2, ..., tn` to kolejne wartości znajdujące się w tablicy.

Składnia tworzenia tablicy indeksowanej poprzez 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 do następnej pozycji tablicy nawiasy kwadratowe pozostaną puste, zostanie jej nadany kolejny indeks. Stosując tę metodę, można uniknąć sytuacji, w której indeksy zostaną źle ponumerowane. Jeżeli w ten sposób będą dodawane wpisy do nowej tablicy, to pierwszy z nich będzie miał indeks równy 0.

### Przykład 6.15

```
<?php
$ks = array("Dżuma", "Potop", "Obcy");
echo $ks[1];
?>
```

## Przykład 6.16

```
<?php
$im[0] = "Jan";
$im[1] = "Paweł";
$im[2] = "Michał";
echo $im[2];
?>
```

## Przykład 6.17

```
<?php
$tab[] = 1;
$tab[] = 2;
$tab[] = 3;
echo $tab[2];
?>
```

**Tablice asocjacyjne**, czyli tablice skojarzeniowe (ang. *associative arrays*), to tablice, w których zamiast indeksów liczbowych używa się identyfikatorów (tzw. kluczy). W takich tablicach przechowywane są pary danych: unikatowy klucz i wartość. Dostęp do wartości uzyskuje się poprzez podanie wartości klucza. Różnicą między tablicą indeksowaną a tablicą asocjacyjną jest wartość 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,
    ...,
    klucz_n => a_n
);
```

lub

```
$tab_a[klucz1] = a1;
$tab_a[klucz2] = a2;
...,
$tab_a[klucz_n] = a_n;
```

### Przykład 6.18

```
<?php  
$osoba = array("nazwisko" => "Kowalski", "imię" => "Jan", "wiek" => 27);  
echo $osoba["nazwisko"];  
?>
```

### Przykład 6.19

```
<?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 6.20

```
<?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 6.21

```
echo $dane[2]["imię"];
```

Wynikiem będzie wyświetlenie tekstu **Tomasz.**

## Typ object

Jest to typ służący do przechowywania obiektów. W języku PHP obiekt musi być jawnie zadeklarowany.

## 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.

## Deklaracja typu

Język PHP jest językiem słabo typowanym. Typ danych jest automatycznie kojarzony ze zmienną w momencie przypisywania jej wartości. Dlatego możliwe jest na przykład dodawanie ciągu do liczby całkowitej.

W PHP 7 została dodana deklaracja typu. Umożliwia to określenie oczekiwanej typu argumentów podczas deklarowania funkcji. W przypadku niezgodności typów zostanie wygenerowany błąd krytyczny.

## Zmiana typu zmiennej

W języku PHP podczas przypisywania zmiennej nowej wartości ponownie ustalany jest jej typ. Czasami zachodzi jednak konieczność zmiany typu danych. Można to zrobić za pomocą rzutowania (**cast**) — wtedy efekt jest jednorazowy — lub za pomocą funkcji **settype()** — wówczas efekt jest trwałym.

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 6.22

```
<?php
$x = 23.75;
$y = (integer) $x;
echo "$x <br>";
echo "$y";
?>
```

Zmianę typu w sposób trwałym przeprowadza się za pomocą funkcji `settype()`. Funkcja ta ma 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`. Jeśli zmiana typu nie powiodła się, zwraca wartość `false`.

## Przykład 6.23

```
<?php
$x = 97.234;
echo "Zadeklarowana wartość zmiennej \$x: $x <br>";
settype($x, 'string');
echo "Wartość zmiennej \$x po zmianie typu na string: $x <br>";
settype($x, 'integer');
echo "Wartość zmiennej \$x po zmianie typu na integer: $x <br>";
?>
```

## Ćwiczenie 6.1

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.

## Ćwiczenie 6.2

Utwórz cztery zmienne. Przypisz im wartości typu `string`. Do tworzenia zmiennych wykorzystaj po kolei każdy z czterech sposobów: użyj 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.

## 6.3.4. 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 warunki podobne do tych, jakie dotyczą nazwy zmiennej. Jednak w nazwach stałych w odróżnieniu od nazw zmiennej 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 ma 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`).

Definicja stałej ma postać:

```
define ("NAZWA_STAŁEJ", wartość);
```

### Przykład 6.24

```
<?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.

### 6.3.5. Operatory i wyrażenia

Operatory służą do wykonywania działań na zmiennych. W języku PHP operatory można podzielić na:

- arytmetyczne,
- porównania,
- bitowe,
- logiczne,
- przypisania,
- łańcuchowe,
- inkrementacji i dekrementacji,
- pozostałe.

#### Operatory arytmetyczne

Służą do wykonywania operacji arytmetycznych (tabela 6.2).

**Tabela 6.2.** Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	\$a + \$b
-	odejmowanie	\$a - \$b
*	mnożenie	\$a * \$b
/	dzielenie	\$a / \$b
%	dzielenie modulo (reszta z dzielenia)	\$a % \$b
**	potęgowanie	\$a ** \$b

#### Operatory porównania

Operatory porównania porównują argumenty; wynikiem tego porównania jest wartość logiczna `true` (prawda) lub `false` (fałsz). Operatory porównania zostały przedstawione w tabeli 6.3.

**Tabela 6.3.** Operatory porównania

Operator	Działanie	Przykład
<code>==</code>	Wynik true, gdy argumenty są równe.	<code>\$a == \$b</code>
<code>!=</code>	Wynik true, gdy argumenty są różne.	<code>\$a != \$b</code>
<code>===</code>	Wynik true, gdy argumenty są tego samego typu i są równe.	<code>\$a === \$b</code>
<code>!==</code>	Wynik true, gdy argumenty są różne lub są różnych typów.	<code>\$a !== \$b</code>
<code>&gt;</code>	Wynik true, gdy argument pierwszy jest większy od drugiego.	<code>\$a &gt; \$b</code>
<code>&lt;</code>	Wynik true, gdy argument pierwszy jest mniejszy od drugiego.	<code>\$a &lt; \$b</code>
<code>&gt;=</code>	Wynik true, gdy argument pierwszy jest większy od drugiego lub mu równy.	<code>\$a &gt;= \$b</code>
<code>&lt;=</code>	Wynik true, gdy argument pierwszy jest mniejszy od drugiego lub mu równy.	<code>\$a &lt;= \$b</code>
<code>&lt;&gt;</code>	Wynik true, gdy argumenty są różne.	<code>\$a &lt;&gt; \$b</code>

## Operatory bitowe

Operatory bitowe umożliwiają wykonanie operacji na poszczególnych bitach liczb (tabela 6.4).

**Tabela 6.4.** Operatory bitowe

Operator	Działanie	Przykład
<code>&amp;</code>	iloczyn bitowy (AND)	<code>\$a &amp; \$b</code>
<code> </code>	suma bitowa (OR)	<code>\$a   \$b</code>
<code>~</code>	negacja bitowa (NOT)	<code>~\$a</code>
<code>^</code>	bitowa różnica symetryczna	<code>\$a ^ \$b</code>
<code>&gt;&gt;</code>	przesunięcie bitowe w prawo	<code>\$a &gt;&gt; n</code>
<code>&lt;&lt;</code>	przesunięcie bitowe w lewo	<code>\$a &lt;&lt; n</code>

## Operatory logiczne

Operatory logiczne wykonują operacje na argumentach, które mają wartość logiczną (`true` lub `false`). Operatory logiczne zostały przedstawione w tabeli 6.5.

**Tabela 6.5.** Operatory logiczne

Operator	Działanie	Przykład
<code>and</code>	iloczyn logiczny	<code>\$a and \$b</code>
<code>&amp;&amp;</code>	iloczyn logiczny	<code>\$a &amp;&amp; \$b</code>
<code>or</code>	suma logiczna	<code>\$a or \$b</code>
<code>  </code>	suma logiczna	<code>\$a    \$b</code>
<code>!</code>	negacja logiczna (NOT)	<code>! \$a</code>
<code>xor</code>	różnica symetryczna	<code>\$a xor \$b</code>

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 są wykorzystywane do przypisywania wartości argumentom znajdującym się po lewej stronie operatora (tabela 6.6). Oprócz prostego przypisania w języku PHP istnieje zestaw operacji łączących przypisanie z innymi operacjami, na przykład arytmetyczną, bitową lub łańcuchową. Przykładowo 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 6.6.** Niektóre operatory przypisania

Operator	Przykład	Znaczenie
<code>=</code>	<code>\$x = 10</code>	<code>\$x = 10</code>
<code>+=</code>	<code>\$x += 5</code>	<code>\$x = \$x + 5</code>
<code>-=</code>	<code>\$x -= 5</code>	<code>\$x = \$x - 5</code>
<code>*=</code>	<code>\$x *= 5</code>	<code>\$x = \$x * 5</code>

Operator	Przykład	Znaczenie
<code>/=</code>	<code>\$x /= 5</code>	<code>\$x = \$x / 5</code>
<code>%=</code>	<code>\$x %= 5</code>	<code>\$x = \$x % 5</code>

## Operatory łańcuchowe

Są dwa sposoby zapisu operatora łańcuchowego (konkatenacji). Obydwa pozwalają na łączenie ciągów znakowych i ich wykorzystanie daje ten sam rezultat. Niezależnie od tego, jaki jest typ danych, które będą łączone za pomocą tych operatorów, dane te są traktowane jako ciągi znaków i rezultat ich łączenia jest zawsze ciągiem znaków (tabela 6.7).

Operator łańcuchowy jest oznaczany pojedynczą kropką.

**Tabela 6.7.** Operatory łańcuchowe

Operator	Działanie	Przykład
<code>.</code>	łączenie łańcuchów znakowych	<code>\$x = "moje "."miasto";</code>
<code>.=</code>	dołączanie do łańcucha znakowego	<code>\$x = "moje ";</code> <code>\$x .= "miasto";</code>

### Przykład 6.25

```
<?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 zwiększania lub zmniejszania wartości zmiennej o jeden.

**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 6.26

```
<?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>";

?>
```



## 6.4. Instrukcje sterujące

Instrukcje sterujące służą do sterowania kolejnością wykonywania poleceń zapisanych w skrypcie. W języku PHP istnieje cała grupa instrukcji, które pozwalają na zmianę tej kolejności w zależności od spełnienia lub niespełnienia określonych warunków.

### 6.4.1. 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 wykonywanie kolejnych instrukcji. Instrukcja, która jest wykonywana, gdy warunek jest spełniony, może być instrukcją prostą (jedną instrukcją) lub złożoną (blokiem instrukcji).

## Przykład 6.27

```
<?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 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 6.28

```
<?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 kolejnych bloków *elseif*. Instrukcja ma postać:

```
if (warunek1)
    instrukcja1;
elseif (warunek2)
    instrukcja2;
```

```

...
elseif (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*.

### Przykład 6.29

```

<?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.";
?>

```

## 6.4.2. Instrukcja switch

Kolejną instrukcją sterującą przepływem jest instrukcja *switch*. Jest to instrukcja wyboru. Wyrażenie występujące w tej konstrukcji jest porównywane z listą szukanych wartości aż do znalezienia pasującej wartości.

```

switch (wyrażenie) {
    case wartość1:
        instrukcja1;
        break;
    case wartość2:

```

```

    instrukcje2;
    break;

case wartość3:
    instrukcje3;
    break;

default:
    instrukcje4;
}

```

Działanie instrukcji wygląda następująco: sprawdź wartość wyrażenia (*wyrażenie*) i jeżeli wynikiem jest *wartość1*, wykonaj *instrukcję1* i wyjdź z bloku switch (polecenie *break*). Jeżeli wynikiem jest *wartość2*, to wykonaj *instrukcję2* i wyjdź z bloku switch. Jeżeli wynikiem jest *wartość3*, to wykonaj *instrukcję3* i wyjdź z bloku switch. Jeżeli wartość jest inna, wykonaj *instrukcję4* i zakończ blok switch.

### Przykład 6.30

```

<?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!";
}
?>

```

### 6.4.3. 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 6.31

```
<?php
$x = 11;
$wynik = ($x < 0) ? "ujemna" : "dodatnia";
echo "Wartość zmiennej x jest $wynik.";
?>
```

## 6.4.4. 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 wykonania oraz warunek, który musi być spełniony, aby kolejny raz wykonać pętlę. Oto składnia instrukcji:

```
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 6.32

```
<?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 6.33

```
<?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 instrukcję, dopóki wyrażenie warunkowe jest prawdziwe. W pętli `do ... while` blok instrukcji jest wykonywany co najmniej raz, nawet jeśli warunek zapisany w wyrażeniu warunkowym jest fałszywy, ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

### Przykład 6.34

```
<?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 argumentu tablicy otrzymujemy wartość aktualnego klucza.

### Przykład 6.35

```
<?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 6.36

```
<?php
$tab = array(
    1 => 'biały',
    2 => 'czarny',
    3 => 'niebieski',
    4 => 'zielony'
```

```
) ;

foreach ($stab as $kl => $x) {
    echo "$kl = $x <br>";
}

?>
```

## Instrukcja break

Instrukcja `break` jest instrukcją modyfikującą zachowanie pętli. Służy do przerywania jej wykonywania. Można ją stosować niezależnie od rodzaju pętli.

### Przykład 6.37

```
<?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` osiągnie 20, nastąpi przerwanie wykonywania powtarzających się instrukcji i wyjście z pętli.

## Instrukcja continue

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

### Przykład 6.38

```
<?php
for ($i = 0; $i <= 30; $i++) {
    if (($i % 3) != 0) continue;
    echo "$i; ";
}
```

```
0; 3; 6; 9; 12; 15; 18; 21; 24; 27; 30;
```

### Rysunek 6.5.

Skrypt wyświetla liczby z zakresu od 0 do 30 podzielne przez 3

Podany kod wyświetli liczby z zakresu od 0 do 30 podzielne przez 3 (rysunek 6.5). 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).

## 6.4.5. 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 6.39

```
<!DOCTYPE HTML>
<html>
<head>
<title>Bloki PHP</title>
<meta charset="UTF-8">
</head>
<?php
$tx = true;
if ($tx) {
?>
<table align="left" border= "1" width="400"
hspace="40" vspace="20" cellspacing="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> Górnjak</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 6.6). Jeżeli zmienna `$tx` będzie miała wartość `false`, tabela nie zostanie wyświetlona na stronie.

Nazwisko	Imię	Telefon
Nowak	Adam	692399123
Kowalski	Jan	628345621
Górnjak	Mateusz	638231484

Rysunek 6.6. Wynik wykonania kodu HTML i PHP

## Ćwiczenie 6.3

Napisz skrypt, który będzie wyświetlał liczby podzielne przez 5 z przedziału od 150 do 100. Liczby powinny być wyświetlane od wartości największej do najmniejszej.

### Rozwiązanie

```
<!DOCTYPE HTML>

<html>
<head>
<title>Liczby podzielne przez 5</title>
<meta charset="utf-8">
</head>
<body>
<?php
for ($i = 150; $i >= 100; $i-=5) {
    echo $i . ', ';
}
?>
</body>
</html>
```

## Ćwiczenie 6.4

Napisz skrypt, który umieści w tablicy oceny semestralne z pięciu przedmiotów, a następnie wyświetli dane z tablicy w postaci: nazwa przedmiotu i ocena z tego przedmiotu.

### Rozwiązanie

```
<!DOCTYPE HTML>

<html>
<head>
<title>Przedmioty i oceny</title>
<meta charset="UTF-8">
</head>
<body>
<?php
$oceny = array(
    "J. polski" => "2",
    "Matematyka" => "4",
```

```
"Geografia" => "3",
"Historia" => "5",
"J. angielski" => "5"

);

foreach ($oceny as $kl => $x) {

echo "$kl = $x <br>";

};

?>

</body>
</html>
```

## Ćwiczenie 6.5

Połącz skrypt PHP z ćwiczenia 6.4 z kodem HTML, tak aby nazwy przedmiotów i oceny z tych przedmiotów były wyświetlane w dwukolumnowej tabeli z nagłówkiem.

### Rozwiązanie

```
<!DOCTYPE HTML>

<html>
<head>
<title>Oceny</title>
<meta charset="utf-8">
<style>
table {width: 200px}
table, tr, td {
    border: 1px solid;
    border-collapse: collapse;
}
</style>
</head>
<body>
<table>
<?php
$oceny = array (
    "Przedmiot" => "Ocena",
    "J. polski" => "2",
    "Matematyka" => "4",
```

```

"Geografia" => "3",
"Historia" => "5",
"J. angielski" => "5"

};

foreach ($oceny as $kl => $x) {
    echo '<tr><td>' . $kl . '</td><td>' . $x . '</td></tr>';
}

?>

</table>
</body>
</html>

```

## Ćwiczenie 6.6

Utwórz tabelę mnożenia  $10 \times 10$ , która będzie wyświetlała wynik mnożenia, na przykład w podany na rysunku 6.7 sposób.

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>1</b>	1	2	3	4	5	6	7	8	9	10
<b>2</b>	2	4	6	8	10	12	14	16	18	20
<b>3</b>	3	6	9	12	15	18	21	24	27	30
<b>4</b>	4	8	12	16	20	24	28	32	36	40
<b>5</b>	5	10	15	20	25	30	35	40	45	50
<b>6</b>	6	12	18	24	30	36	42	48	54	60
<b>7</b>	7	14	21	28	35	42	49	56	63	70
<b>8</b>	8	16	24	32	40	48	56	64	72	80
<b>9</b>	9	18	27	36	45	54	63	72	81	90
<b>10</b>	10	20	30	40	50	60	70	80	90	100

**Rysunek 6.7.** Tabliczka mnożenia

## Rozwiązanie

```

<!DOCTYPE HTML>

<html>
<head>
<title>Tabliczka mnożenia</title>
<meta charset="UTF-8">
<style>
table {
    width: 400px
}

```

```
table, tr, td, th {  
    border: 1px solid;  
    border-collapse: collapse;  
    text-align: center;  
}  
</style>  
</head>  
<body>  
<table>  
<tr>  
<th></th>  
<?php  
$wiersz = 10;  
$kolumna = 10;  
for ($i = 1; $i <= $kolumna; $i++) {  
    echo "<th>" . $i . "</th>";  
}  
?>  
</tr>  
<?php  
for ($i = 1; $i <= $wiersz; $i++) {  
    echo "<tr>";  
    echo "<th>" . $i . "</th>";  
    for ($j = 1; $j <= $kolumna; $j++) {  
        echo "<td>";  
        echo $i * $j;  
        echo "</td>";  
    }  
    echo "</tr>";  
}  
?>  
</table>  
</body>  
</html>
```



## 6.5. Funkcje

Funkcja jest ciągiem instrukcji stanowiącym blok kodu, który może być wielokrotnie wykorzystany w różnych programach lub w różnych miejscach programu. 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.

### 6.5.1. Definiowanie funkcji

W języku PHP można definiować własne funkcje. Funkcję definiujemy za pomocą słowa kluczowego `function`, po którym następuje nazwa funkcji, a dalej w nawiasach okrągłych powinny zostać wymienione argumenty funkcji oddzielone przecinkami. W nawiasach klamrowych jest zapisywane ciało funkcji. Definicja funkcji ma postać:

```
function nazwa($argument1, $argument2, ...) {
    instrukcje
}
```

Jeżeli funkcja ma argumenty, są one zapisane w nawiasach okrągłych jako lista zmiennych oddzielonych przecinkami. Jeżeli funkcja nie ma argumentów, nawiasy okrągłe powinny pozostać puste. Nazwa funkcji powinna być tworzona zgodnie z zasadami obowiązującymi dla nazw zmiennych i powinna odzwierciedlać to, co dana funkcja będzie robiła. Nazwa funkcji nie może zaczynać się od znaku \$.

#### Przykład 6.40

```
<?php
function napis() {
    echo "<h2>Programuj w PHP!</h2>";
}
napis();
?>
```

#### Przykład 6.41

```
<?php
function dodaj($a, $b) {
    $c = $a + $b;
    echo "Wynik dodawania $a i $b to " . $c;
}
dodaj(15, 37);
?>
```

## 6.5.2. Zwracanie wartości przez funkcje

Jeżeli chcemy, aby funkcja zwracała wartość, która będzie wykorzystywana w innych działaniach, należy użyć słowa kluczowego `return`. Zatrzymuje ono 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 6.42

```
<?php
function dodaj($a, $b) {
    $c = $a + $b;
    return $c;
}
$suma = dodaj(14, 7);
echo "Wynik dodawania to $suma";
?>
```

## 6.5.3. Zasięg zmiennych

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

- 1.** lokalna,
- 2.** 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 6.43

```
<?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 funkcja ta nie ma do niej dostępu. 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 pobrać 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 wewnętrz funkcji, można wykorzystać instrukcję global. Przed odwołaniem się do zmiennej należy umieścić konstrukcję:

```
global $nazwa_zmiennej;
```

Z pomocą jednej instrukcji global można zadeklarować w funkcji wiele zmiennych globalnych.

```
global $zm1, $zm2, $zm3;
```

### Przykład 6.44

```

<?php
$zm = 1;

function pokaz() {
    global $zm;
    echo "Wartość zmiennej \$zm wynosi $zm. <br>";
}

pokaz();
?>

```

Inną metodą odwołania się 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 6.45

```
<?php
$zm = 7;
function pokaz() {
    echo "Wartość zmiennej \$zm wynosi ";
    echo $GLOBALS['zm'];
    echo ". <br>";
}
pokaz();
?>
```

Przy posługiwaniu 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 6.46

```
<?php
function tekst() {
    $tx = "Tekst1";
}
echo "Tekst zapisany w zmiennej to: $tx. <br>";
?>
```

W podanym przykładzie zmieniona `$tx` ma zasięg lokalny, została zdefiniowana wewnątrz funkcji `tekst()` i poza tą funkcją nie istnieje. Próba uruchomienia tego kodu spowoduje wyświetlenie komunikatów o błędach.

### Zmienne statyczne

Zmienne statyczne to zmienne lokalne funkcji, które zachowują swoją wartość między wywołaniami funkcji. Domyślnie zmieniona lokalna jest tworzona w chwili wywołania funkcji i jest widoczna w obrębie tej funkcji. Gdy funkcja zakończy działanie, zmieniona znika. Przy ponownym wywołaniu funkcji zmieniona jest tworzona i przypisywana jest jej wartość początkowa.

### Przykład 6.47

```
<?php
function funk() {
    $i = 1;
    echo "Funkcja wywołana $i raz(y) <br>";
    $i++;
}
funk();
funk();
funk();
?>
```

W podanym przykładzie przy każdym wywołaniu funkcji funk() zmienna \$i będzie przyjmowała wartość 1, co spowoduje wyświetlenie tego samego komunikatu, mówiącego, że funkcja została wywołana jeden raz (rysunek 6.8).

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 6.48

```
<?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 wywołaniu funkcji zostanie utworzona zmienna statyczna, a przy opuszczaniu funkcji wartość tej zmiennej zostanie zapamiętana. Przy ponownym wywołaniu funkcji instrukcja przypisująca początkową wartość zmiennej \$i będzie

Funkcja wywołana 1 raz(y)
Funkcja wywołana 1 raz(y)
Funkcja wywołana 1 raz(y)

### Rysunek 6.8.

Efekt zadeklarowania w treści funkcji zmiennej lokalnej

ignorowana, a stosowana będzie wartość zapamiętana przy poprzednim wywołaniu funkcji (rysunek 6.9).

Funkcja wywołana 1 raz(y)
Funkcja wywołana 2 raz(y)
Funkcja wywołana 3 raz(y)

## 6.5.4. Argumenty funkcji

Argumenty mogą być przekazywane do funkcji 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 prezentowanych do tej pory przykładach. W praktyce oznaczało to, że do funkcji przekazywane są kopie argumentów źródłowych i wszystkie operacje wykonywane są na kopiiach. Zostanie to zilustrowane w przykładzie 6.49.

#### Przykład 6.49

```
<?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 `$liczba` (wartość przekazanej zmiennej jest zwiększana o 3). Przed jej wywołaniem zmieniona `$liczba` została ustawiona na wartość 2 i wyświetlona w celu przetestowania jej stanu. Następnie została wywołana funkcja `wart()`. Po jej wykonaniu ponownie została wyświetlona zmieniona `$liczba`, aby sprawdzić jej aktualny stan. Wszystkie operacje wewnętrz funkcji odbywały się na kopiach zmiennej, natomiast wartość oryginalnej zmiennej nie uległa zmianie (rysunek 6.10).

Wartość zmiennej \$liczba przed wywołaniem funkcji: 2
Wartość zmiennej \$liczba po wywołaniu funkcji: 2

**Rysunek 6.10.** 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 definiowaniu argumentów należy umieścić przed nimi znak & (ampersand).

### Przykład 6.50

```
<?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 6.11).

Wartość zmiennej `$liczba` przed wywołaniem funkcji: 2  
 Wartość zmiennej `$liczba` po wywołaniu funkcji: 5

**Rysunek 6.11.** 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 6.51

```
<?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 ma zdefiniowany argument domyślny. Przy pierwszym wywoaniu funkcja zostanie wykonana z domyślną wartością argumentu, przy drugim wartością argumentu będzie null, przy trzecim wartość argumentu zostanie ustawiona zgodnie z argumentem wywołania funkcji (rysunek 6.12).

Język programowania – PHP  
 Język programowania –  
 Język programowania – Java

### Rysunek 6.12.

Wynik wywołania funkcji z różnymi argumentami

### Przykład 6.52

```

<?php
function styl($tekst, $kolor = "red") {
    echo "<p style=\"color: $kolor\">" . $tekst . "</p>";
}
styl("Tytuł", "blue");
styl("Rozdział 1.", "green");
styl("Treść");
?>

```

W podanym przykładzie funkcja `styl()` ma dwa argumenty. Pierwszy argument `$tekst` przekazuje treść tekstu do wyświetlenia, drugi `$kolor` przekazuje kolor, w jakim tekst powinien zostać wyświetlony (rysunek 6.13). Wartość domyślna argumentu `$kolor` została ustawiona na `red`. Jeżeli przy wywołaniu funkcji `kolor` tekstu nie zostanie określony, tekst będzie wyświetlany w kolorze przekazanym jako argument domyślny (`red`).

Wartości domyślne można przypisać dowolnej liczbie argumentów. Ale jeżeli wartość domyślna zostanie przypisana określonemu argumentowi, to wszystkie następne również muszą mieć wartości domyślne.

Tytuł  
 Rozdział 1.  
 Treść

### Rysunek 6.13.

Wynik wywołania funkcji z różnymi argumentami

## 6.6. Funkcje wbudowane

W języku PHP istnieje duża grupa predefiniowanych funkcji, które są przydatne podczas tworzenia aplikacji internetowych. Funkcje te zostały podzielone na grupy tematyczne — ułatwia to znalezienie funkcji wykonującej określone zadania.

### 6.6.1. Funkcje tablic

Cała grupa funkcji wbudowanych dotyczy operacji wykonywanych na tablicach. Do najpopularniejszych należą `count()` i `sizeof()`, które zliczają liczbę elementów w tablicy.

#### Przykład 6.53

```
<?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 ustawianiu elementów tablicy w określonym porządku, najczęściej 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 6.54

```
<?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 ";
}
?>
```

Zawartość tablicy przed sortowaniem:  
3 2 5 7 9 0 1 4  
Zawartość tablicy po sortowaniu:  
0 1 2 3 4 5 7 9

Wynik interpretacji kodu został pokazany na rysunku 6.14.

**Rysunek 6.14.**

Wyświetlenie danych pobranych z tablicy

## Funkcje wyszukiwania

`in_array()` — funkcja ma dwa argumenty. Pierwszym jest poszukiwana wartość, drugim przeszukiwana tablica. Jeżeli szukana wartość zostanie znaleziona, funkcja zwróci wartość `true`, w przeciwnym razie zwróci wartość `false`.

### Ćwiczenie 6.7

Utwórz tablicę 20-elementową. Wartości tablicy to dowolne liczby całkowite. Napisz skrypt obliczający, ile jest w tablicy liczb parzystych, a ile nieparzystych.

### Ćwiczenie 6.8

Utwórz tablicę asocjacyjną składającą się z 15 elementów. Nazwy kluczy to: l1, l2 itd. Wartości to losowo podane liczby całkowite. Wyświetl pary klucz–liczba dla liczb, które są podzielne przez 3 i 4.

## 6.6.2. 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 ma żadnych argumentów. Informacje na temat bieżącej daty i czasu są zwracane w postaci liczby.

Odpowiada ona liczbie sekund, które upłynęły od godziny 00:00:00 1 stycznia 1970 roku do bieżącej daty. Jest to tak zwany *znacznik czasu* (timestamp).

### Przykład 6.55

```
<?php
echo time();
?>
```

Taki zapis daty i czasu pozwala na wykonywanie działań na dacie, na przykład zwiększenie daty o jeden dzień, czasu o jedną 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 uzyskany wynik przekształcić na bardziej czytelną 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 6.8.

**Tabela 6.8.** 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	na przykład 2005
yday	Numer kolejnego dnia roku	od 0 do 365
weekday	Nazwa dnia tygodnia (po angielsku)	na przykład Monday
month	Nazwa miesiąca (po angielsku)	na przykład January
0	Aktualny znacznik czasu	

## Przykład 6.56

```
<?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 zmiennej `$dzien`, `$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 6.15).

Bieżąca data to: 18-11-2019 r.

**Rysunek 6.15.** 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 6.9.

**Tabela 6.9.** 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

Znacznik	Znaczenie	Opis
d	Dzień miesiąca w formacie z zerem na początku	od 01 do 31
D	Dzień tygodnia 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
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 6.57

```
<?php
echo date("Y-m-d") . "<br>";
echo date("d-m-Y") . "<br>";
echo date("G:i:s") . "<br>";
echo date("H-i-s a") . "<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 6.57 pokazuje użycie tej funkcji z różnymi znacznikami formatującymi. Na rysunku 6.16 widoczny jest rezultat wykonania kodu.

2019-12-17  
 17-12-2019  
 21:47:36  
 21-47-36 pm  
 2019-12-17  
 2019-12-17 21:47:36

### Rysunek 6.16.

Rezultat użycia funkcji `date()` z różnymi znacznikami formatującymi

## Funkcja mkttime()

Funkcja `mkttime()` zwraca znacznik czasu daty podanej jako argument funkcji. Może mieć od zera do sześciu argumentów w postaci liczb całkowitych. Są to kolejno:

- godzina,
- minuta,
- sekunda,
- miesiąc,
- dzień miesiąca,
- rok.

Znacznik czasu, który zwróci funkcja `mkttime()`, może zostać wykorzystany w funkcjach `getdate()` i `date()`.

### Przykład 6.58

```
<?php
$czas = mkttime(16, 30, 0, 10, 24, 2020);
echo "Data: dzień, miesiąc, rok, godzina:minuta <br>";
echo date("d-m-Y G:i", $czas) . "<br>";
echo "Data: rok, miesiąc, dzień, godzina:minuta:sekunda <br>";
echo date("Y-m-d G:i:s", $czas);
?>
```

Wynik interpretacji kodu został pokazany na rysunku 6.17.

```
Data: dzień, miesiąc, rok, godzina:minuta
24-10-2020 16:30
Data: rok, miesiąc, dzień, godzina:minuta:sekunda
2020-10-24 16:30:00
```

**Rysunek 6.17.** Wykorzystanie funkcji `mkttime()` do ustawienia znacznika czasu

### Ćwiczenie 6.9

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.

### Ćwiczenie 6.10

Napisz skrypt, który będzie wyświetlał bieżący dzień tygodnia w podanej postaci: Dzisiaj jest [[dzień tygodnia](#)].

### Ćwiczenie 6.11

Napisz skrypt wyświetlający liczbę dni, które upłynęły od początku bieżącego roku, oraz liczbę dni, które pozostały do końca bieżącego roku.

### 6.6.3. Funkcje formatowania ciągów

Gdy pracuje się 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 uwzględnia 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("ciag_znakow")
```

Funkcja ta dla wybranego bloku tekstu przed każdym znakiem końca linii automatycznie wstawi znacznik <br> i zwróci przetworzony tekst.

#### Przykład 6.59

```
<!DOCTYPE HTML>
<html>
<head>
<title>Funkcja nl2br</title>
<meta charset="UTF-8">
<style>
p {
    font-weight: bold;
}
</style>
</head>
<body>
<?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 "<p>Tekst przed użyciem funkcji nl2br():</p>";
echo $tekst."<br>";
echo "<p>Tekst po użyciu funkcji nl2br():</p>";
echo nl2br($tekst);
?>
</body>
</html>

```

Wynik interpretacji kodu został pokazany na rysunku 6.18.

**Tekst przed użyciem funkcji nl2br():**

Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego taka bezczynność w senacie? Senatorowie siedzą

**Tekst po użyciu funkcji nl2br():**

Na cóż czekamy, zebrani na rynku?  
Dziś mają tu przyjść barbarzyńcy.  
Dlaczego taka bezczynność w senacie?  
Senatorowie siedzą – czemuż praw nie uchwalą?

**Rysunek 6.18.** 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 ma jeszcze trzy opcjonalne argumenty. Są to: liczba wskazująca maksymalną długość linii, ciąg znaków zastosowany do rozdzielenia linii oraz argument podziału słów dłuższych niż zadeklarowana maksymalna długość linii.

### Przykład 6.60

```

<?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 6.19). Po zastosowaniu dwóch kolejnych argumentów kod wysłany do przeglądarki pozwoli na wyświetlenie tekstu w liniach (rysunek 6.20).

Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego taka bezczynność w senacie? Senatorowie siedzą-

**Rysunek 6.19.** Brak interpretacji przez przeglądarkę znaku nowej linii

### Przykład 6.61

```
<?php
$tekst = <<<TX
Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy.
Dlaczego taka bezczynność w senacie? Senatorowie siedzą-
czemuż praw nie uchwalą?
TX;
echo wordwrap($tekst, 30, "<br>\n");
?>
```

Ostatni, czwarty argument funkcji będzie zastosowany, gdy w linii pojawi się słowo dłuższe niż zadeklarowana maksymalna długość linii. Domyslnie takie słowo nie zostanie podzielone między liniami. Użycie czwartego argumentu wymusi podział słowa między liniami. Aby się nim posłużyć, trzeba przypisać mu wartość logiczną true.

Na cóż czekamy, zebrani na rynku? Dziś mają tu przyjść barbarzyńcy. Dlaczego taka bezczynność w senacie? Senatorowie siedzą – czemuż praw nie uchwalą?

**Rysunek 6.20.**

Podział tekstu na linie o długości 30 znaków

### Przykład 6.62

```
<?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 6.21.

Potrzebne informacje znajdują się pod adresem:  
<http://helion.pl/kategorie/podreczniki-szkolne/technik-informatyk>

**Rysunek 6.21.** 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.

Jeżeli ciąg znakowy użyty jako argument funkcji `strtoupper()` lub `strtolower()` zawiera znaki narodowe (na przykład literę ę lub ą), to nie zostaną one zamienione.

Funkcje, które przy zamianie liter na małe lub duże uwzględniają sposób ich kodowania, to `mb_strtoupper()` i `mb_strtolower()` zapisywane w postaci:

```
mb_strtoupper(argument, [kodowanie])
```

```
mb_strtolower(argument, [kodowanie])
```

gdzie `[kodowanie]` to sposób kodowania znaków.

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 białych znaków z początku lub końca ciągu można użyć jednej z trzech funkcji: `trim()`, `ltrim()`, `rtrim()`.

Wszystkie 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("ciag_znakowy", "znaki do usuniecia")
```

## 6.6.4. Funkcje analizowania ciągów znaków

### Funkcja sprawdzająca długość ciągu

Do sprawdzenia długości ciągu znaków jest wykorzystywana funkcja `strlen()`. Funkcja ta zwraca prawidłowy wynik, jeśli ciąg znaków nie zawiera znaków narodowych. Funkcja, która uwzględnia sposób ich kodowania, to `mb_strlen()`.

#### Przykład 6.63

```
<?php
$napis = "Senatorowie siedzą - czemuż praw nie uchwalą?";
echo 'Tekst: "'.$napis.''";
$dł = mb_strlen($napis);
echo " ma długość $dł znaków.<br>";
?>
```

W podanym przykładzie zostanie wyświetlony tekst oraz jego długość.

### 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 6.64

```
<?php
$napis = "Litwo, Ojczyzno moja!";
echo $napis[0];
echo $napis[10];
?>
```

W podanym przykładzie dla zmiennej `$napis` w nawiasach kwadratowych został podany indeks znaku, który powinien być wyświetlony. Zostanie wyświetlony znak

znajdujący się na pozycji o indeksie 0 (litera L) oraz znak znajdujący się na pozycji o indeksie 10 (litera z).

## 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 zwróci wartość `false`, gdy podciąg nie zostanie znaleziony. Jeżeli szukany ciąg jest fragmentem ciągu źródłowego, to funkcja zwróci fragment ciągu źródłowego od znalezionej podciągu do jego końca. Funkcja `stristr()` 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 6.65

```
<?php
$dane = "Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, tel. 693341678";
$tel = strstr($dane, "tel.");
echo $tel;
?>
```

W podanym przykładzie zostanie wyświetlony tekst od znalezionej podciągu `tel.` do końca przeszukiwanego ciągu.

Funkcja zwróci wartość `false`, gdy ciąg nie zostanie znaleziony, zatem może być wykorzystana w instrukcjach warunkowych.

### Przykład 6.66

```
<?php
$dane = "Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, 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 6.22), w przeciwnym razie wyświetli się komunikat o braku numeru telefonu.

Ciąg główny: Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, tel. 693341678  
 Znaleziony podciąg: tel. 693341678

**Rysunek 6.22.** Znaleziony podciąg w ciągu głównym

### Funkcja strpos()

Podobne działanie do funkcji `stristr()` ma funkcja `strpos()`. Funkcja ta sprawdza, czy szukany podciąg znajduje się w ciągu źródłowym, a jeżeli tak, zwraca jego położenie. Ma trzy argumenty: pierwszy to przeszukiwany ciąg źródłowy, drugi to poszukiwany podciąg, trzeci to pozycja w ciągu źródłowym, od której rozpocznie się przeszukiwanie. Trzeci argument jest opcjonalny. Jeżeli podciąg nie zostanie znaleziony, zwrócona zostanie 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. Ma trzy argumenty: pierwszy to ciąg źródłowy, drugi to indeks wskazujący miejsce początku zwracanego podciągu, trzeci, opcjonalny, określa długość pobranego ciągu. Jeżeli zostanie podany trzeci argument, zwrócona zostanie określona liczba znaków. Gdy ten argument zostanie pominięty, zostanie zwrócony podciąg od wskazanego indeksu do końca ciągu.

### Przykład 6.67

```
<?php
$dane = "Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, tel. 693341678";
echo substr($dane, 4, 8);
?>
```

W podanym przykładzie zostanie wyświetlony tekst `Kowalski`.

Jeżeli jako drugi argument funkcji (indeks) zostanie podana wartość ujemna, pozycja indeksu będzie liczona od końca ciągu źródłowego.

### Przykład 6.68

```
<?php
$dane = "Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, tel. 693341678";
echo substr($dane, 4, 8)."<br>";
echo substr($dane, -14, 14);
?>
```

## Funkcja strtok()

Za pomocą funkcji `strtok()` można podzielić ciąg źródłowy na podciągi. Funkcja ma 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 6.69

```
<?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 zmiennej podanej jako argument jest ciągiem. Wywołana drugi raz funkcja `strtok()` nie zawiera jako argumentu ciągu źródłowego. Podanie tego argumentu spowodowałoby przeszukiwanie ciągu od początku. Wynik interpretacji kodu został pokazany na rysunku 6.23.

Jan Kowalski  
ul. Długa 23  
80-874 Gdańsk  
tel. 693341678

**Rysunek 6.23.**  
Wynik podzielenia  
ciągu głównego.  
Znakiem rozdzielającym  
był 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("ciag1", "ciag2")
```

- zwraca wartość mniejszą od 0, jeżeli `ciag1` jest mniejszy od `ciag2`,
- zwraca wartość większą od 0, jeżeli `ciag1` jest większy od `ciag2`,

- 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")
```

### Ćwiczenie 6.12

Napisz skrypt, który będzie wyszukiwał określony wyraz w podanym ciągu i obliczał, ile razy ten wyraz w nim wystąpił.

### Ćwiczenie 6.13

Napisz skrypt, który będzie zawierał tablicę wybranych wyrazów oraz ciąg źródłowy. Każde wystąpienie w tym ciągu wyrazu z tablicy powinno spowodować zamianę tego wyrazu na tekst `wyraz` z tablicy.

## 6.7. Funkcje obsługi plików

W języku PHP istnieją funkcje umożliwiające przeprowadzanie różnych operacji na plikach. Pozwalają one na odczytywanie informacji o strukturze plików i katalogów oraz na odczytywanie danych i zapisywanie ich do plików.

### 6.7.1. Dołączanie plików

Jeżeli skrypt PHP zawiera dużą ilość kodu, to można go podzielić i zapisać w kilku oddzielnych plikach. Do ponownego ich dołączenia można użyć polecenia `include` lub `require`. Obydwa polecenia wstawiają zawartość wskazanego pliku w miejscu, w którym wystąpią.

Dołączony skrypt zostanie wykonany tak, jakby był częścią kodu, w którym został wstawiony. Polecenie `include` ma postać:

```
include 'nazwa_pliku'
```

Polecenie `require` ma postać:

```
require 'nazwa_pliku'
```

### Przykład 6.70

#### *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*. Ponieważ polecenie `include` jest częścią języka PHP, to w kodzie HTML został użyty znacznik `<?php`, aby przełączyć się na ten tryb. Po wykonaniu polecenia tryb PHP zostaje wyłączony i dalsza część skryptu będzie przetwarzana jako kod HTML.

Różnica między poleceniami `include` i `require` występuje tylko wtedy, gdy dołączany plik nie może zostać odczytany. Polecenie `include` wygeneruje ostrzeżenie, ale skrypt zawierający jego wywołanie będzie nadal działał. Natomiast użycie polecenia `require` spowoduje zgłoszenie błędu i zakończenie działania skryptu.

Plik do dołączenia najłatwiej wskazać przez podanie ścieżki dostępu do niego (względnej lub bezwzględnej).

### Przykład 6.71

```
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` wewnętrz dołączonego kodu kończy jego działanie.

## 6.7.2. Operacje na plikach

### Sprawdzanie, 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 6.72

```
if (file_exists('skrypt.php')) {
    echo "Plik został znaleziony.";
}
```

Funkcja `is_file()` sprawdza, czy podany jako jej argument ciąg wskazuje na plik. Ma postać:

```
is_file(plik)
```

### Przykład 6.73

```
<?php
$p = 'plik.txt';
if (is_file($p)) {
    echo "To jest plik";
}
?>
```

Funkcja zwraca wartość `true`, jeżeli argument funkcji wskazuje na plik.

### 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. Wiedza na temat rozmiaru pliku jest potrzebna na przykład wtedy, gdy plik ma zostać dołączony jako załącznik do wiadomości 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 takiej 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 zamknięcie 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 6.74

```
$p = fopen('dane.txt', 'r');
```

Plik zostanie otwarty tylko do odczytu.

Funkcja `fopen()` zwraca wartość `false`, jeżeli plik nie może zostać otwarty.

Po odpowiednim ustaleniu atrybutu `tryb_otwarcia` funkcja ta może zostać wykorzystana do zapisywania w pliku nowych danych lub dopisywania ich do niego.

### Przykład 6.75

```
$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 6.76

```
$p = fopen('dane.txt', 'a');
```

Po wykonaniu przedstawionego kodu nowe dane zostaną dopisane na końcu pliku.

Do zamknięcia pliku służy funkcja `fclose()` zapisywana w postaci:

```
fclose(deskryptor)
```

Deskryptor to wartość zwrocona 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 6.77

```
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 w pliku się nie powiodło.

## Przykład 6.78

```
<?php
$tekst = "Barbarzyńcy, gdy przyjda, ustanowia prawa.\n";
if (!$p = fopen('dane.txt', 'a')) {
    echo "Nie można otworzyć pliku dane.txt";
} else {
    if (fwrite($p, $tekst) === false) {
        echo "Zapis do pliku nie powiodł 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. Jeżeli plik nie istnieje, zostanie utworzony przy pierwszym wywołaniu funkcji.

## Odczyt danych

Dane z pliku mogą być odczytywane na wiele sposobów. W języku PHP istnieje duża liczba 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 sprawdzania, czy osiągnięty został koniec pliku. Jeśli tak, zwraca ona wartość `true`. Funkcja `feof()` ma postać:

```
feof(deskryptor)
```

### Przykład 6.79

```
<?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`. Jeśli 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. 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>`.

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 jeden znak do przodu. Gdy zostanie osiągnięty koniec pliku, funkcja zwróci wartość `false`.

### Przykład 6.80

```
<?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 sprawdzania, czy został osiągnięty koniec pliku, wykorzystany został operator `!==`.

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 6.81

```

<?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 6.24).

Barbarzyńcy, gdy przyjdą, usta nowią prawa. Barbarzyńcy, gdy przyjdą, ustanowią prawa. Barbarzyńcy, gdy przyjdą, ustanowią prawa.

#### Rysunek 6.24.

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 tekstopowego 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 mieć 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.

### 6.7.3. Operacje na katalogach

Język PHP został wyposażony w 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 można usunąć 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 6.82

```
<?php
$katalog = "./";
if ($deskr = opendir($katalog)) {
    while (($plik = readdir($deskr)) !== false) {
        echo "$plik<br>";
    }
    closedir();
} else {
    echo "Nie można otworzyć katalogu";
}
?>
```

W podanym przykładzie ustwiona 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.

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 6.83

```
<?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.

### Ćwiczenie 6.14

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()`.

## 6.7.4. Praktyczne zastosowanie operacji na plikach

Omówione mechanizmy pracy z plikami zostaną wykorzystane do dodawania przez użytkowników na stronie internetowej komentarzy na wybrany temat.

### Ćwiczenie 6.15

Utwórz skrypt, który będzie zapisywał opinie użytkowników w pliku tekstowym *opinie.txt*. Na stronie wyświetli formularz, który pozwoli na wpisanie opinii. Nowe opinie powinny być dopisywane do pliku i umieszczane na jego końcu. Dotychczasowe opinie zapisane w pliku tekstowym powinny zostać wyświetlone na stronie i powinny być dostępne dla innych jej użytkowników.

### Rozwiążanie

*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 charset="UTF-8">  
</head>  
<body>  
  
<div>  
<form action="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>  
  
<b>Dodane opinie:</b>  
<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 zmian klimatu";
}
?>
</div>
</body>
</html>

```

W pierwszej części kodu sprawdzane jest, czy za pomocą metody POST została przesłany z formularza parametr komentarz zawierający treść opinii użytkownika. Jeżeli tak, to wartość tego parametru jest odczytywana i zapisywana w zmiennej \$tekst. Funkcja substr() skraca jego długość do 255 znaków. Następnie za pomocą funkcji strip\_tags() ze zmiennej \$tekst zostają usunięte znaczniki PHP i HTML (jeżeli się takie pojawią).

Kolejny blok instrukcji dodaje wartość zmiennej \$tekst do pliku *opinie.txt*. Jeżeli pliku nie można otworzyć (*if (!\$op = fopen())*) lub nie można dodać do niego nowej treści (*if (fwrite())*), zostaną wyświetlane odpowiednie komunikaty.

Następny blok to kod HTML tworzący formularz do wprowadzania tekstu za pomocą znacznika *<textarea>*. Dane z formularza będą przekazywane przy użyciu 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 nie było opinii, na stronie pojawia się komunikat o braku opinii (rysunek 6.25). Jeżeli były, to ich treść wyświetla się na stronie (rysunek 6.26).

The screenshot shows a web page with a light gray background. At the top, there is a header in Polish: "Dodaj swój komentarz na temat zmian klimatu" followed by "(Maksymalnie 255 znaków)". Below this is a large empty text area for input. At the bottom of the text area is a small button labeled "Wyślij". Below the text area, the text "Dodane opinie:" is displayed in bold. Underneath it, the message "Brak opinii na temat zmian klimatu." is shown in a standard font.

**Rysunek 6.25.** Brak opinii w pliku *opinie.txt*

**Dodaj swój komentarz na temat zmian klimatu**  
(Maksymalnie 255 znaków)

Zmiany klimatu mogą doprowadzić do głodu. Ewa

**Wyslij**

**Dodane opinie:**

Skutki zmian klimatu będąliśmy odczuwali przez wiele lat. Max  
Zmiany klimatu mogą doprowadzić do wyginięcia wielu gatunków zwierząt. Lala  
Na zmiany klimatu człowiek nie ma wpływu. Piotr

**Rysunek 6.26.** Opinie zapisane w pliku opinie.txt

## 6.7.5. 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 bez nawiasów, jeżeli nie mają 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 6.84

```
<?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.

## Zadanie 6.1

Dodaj do swojej strony internetowej podstronę z informacjami na wybrany przez siebie temat zgodny z tematyką strony. Utwórz formularz, który pozwoli na wpisanie opinii dotyczącej zamieszczonych informacji. Opinie użytkowników powinny zostać zapisane w pliku tekstowym. Wszystkie opinie wpisane przez użytkowników powinny być widoczne na stronie.

Utwórz przycisk [Wyczyść komentarze](#). Po kliknięciu tego przycisku wszystkie wpisane do tej pory komentarze powinny zostać usunięte.

## Zadanie 6.2

Przygotuj plik [artykuł.txt](#) i umieść w nim informacje na wybrany temat. Przygotuj szablon strony internetowej. W menu z lewej strony umieść odnośnik do podstrony [Moje artykuły](#). W górnej części tej podstrony wyświetl treść zapisaną w pliku [artykuł.txt](#) w liniach o długości 30 znaków.

Poniżej wyświetlonego tekstu utwórz pole tekstowe [Dodaj tekst](#), w którym użytkownik będzie mógł wpisywać komentarze do wyświetlonej treści. Po wpisaniu komentarza i wciśnięciu przycisku [Wyślij](#) tekst powinien zostać przesłany na serwer i po dodaniu znaku przecinka dopisany do pliku [Nowe\\_teksty.txt](#).

Utwórz pole tekstowe i wyświetl w nim zawartość pliku [Nowe\\_teksty.txt](#). Wyświetlaną zawartość pliku sformatuj tak, aby każdy tekst zakończony przecinkiem był wyświetlany w oddzielnej linii.

## 6.8. Obsługa formularzy

Dynamiczna strona WWW to strona, która jest generowana na podstawie danych przekazanych między innymi przez użytkownika. Podstawowym sposobem przekazywania danych z przeglądarki internetowej do serwera jest korzystanie z formularzy HTML. Użytkownik za pomocą formularza wprowadza różne informacje, które są wysyłane do serwera. Dane te są przetwarzane na serwerze przy użyciu skryptów PHP.

### 6.8.1. Formularz HTML

#### Przykład 6.85

Prosty formularz HTML może mieć postać:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Formularz rejestracyjny</title>
<meta charset="UTF-8">
<style>
```

```
p {
    font-weight: bold;
    font-size: 14pt;
}

#wyk {
    font-weight: bold;
    font-size: 11pt;
}

</style>
</head>
<body>

<form action="http://localhost/skrypt_form.php" method="post">
<p>Formularz kontaktowy:</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">

<p id="wyk">Wykształcenie:</p>
<input type="radio" value="podstawowe"
name="wykszt" checked> Podstawowe<br>
<input type="radio" value="średnie" name="wykszt"> Średnie<br>
<input type="radio" value="wyższe" 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">&ampnbsp &ampnbsp
<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 (*Wyczysć*) (rysunek 6.27).

**Formularz kontaktowy:**

Nazwisko:

Imię:

Zawód:

Adres e-mail:

**Wykształcenie:**

Podstawowe  
 Średnie  
 Wyższe

Zgadzam się na przetwarzanie moich danych osobowych

**Przyciski:**

**Rysunek 6.27.** Przykładowy formularz

Po kliknięciu przycisku *Wyślij* dane z formularza powinny zostać przesłane na serwer za pomocą metody POST. Parametr `action` wskazuje adres serwera i nazwę skryptu PHP, 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) przesyłania informacji do wskazanego skryptu. Może przyjąć jedną z dwóch wartości: `post` lub `get`. Wartością domyślną jest `get`, ale zaleca się stosowanie metody `post`.

## 6.8.2. Przekazywanie danych — metoda get i post

Każdy formularz zaczyna się od znacznika otwierającego `<form>` i kończy się znacznikiem zamkającym `</form>`. Pomiędzy tymi znacznikami znajdują się wszystkie polecenia dotyczące formularza. Jeżeli dane z formularza będą przesyłane na serwer, znacznik ten powinien mieć następujące atrybuty:

`action="adres"`. To atrybut, który ustala adres do wysyłki danych z formularza. Dane wprowadzone do formularza zostaną przekazane do podanego skryptu.

`method="metoda"` określa sposób (metodę) przekazania informacji do wskazanego skryptu lub na wskazany adres. Atrybut może przyjąć jedną z dwóch wartości: `post` lub `get`.

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](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 niewielu 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 liczby parametrów, a parametry nie są widoczne w pasku adresu 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 6.86

W celu sprawdzenia poprawności działania skryptu z przykładu 6.85 zostanie utworzony skrypt PHP (*skrypt\_form.php*) przetwarzający dane przesłane z formularza.

```
<?php
echo "Odpowiedź z PHP: <br>". $_POST['nazw'];
echo $_POST['im']. "<br>";
echo $_POST['zaw']. "<br>";
echo $_POST['adr']. "<br>";
echo "Zostało wybrane wykształcenie: {$_POST['wykszt']} .";
?>
```

Jeżeli podany kod zostanie zapisany w pliku *skrypt\_form.php*, to po otwarciu formularza, wpisaniu danych (rysunek 6.28) i jego wysłaniu przeglądarka powinna wyświetlić przesłane dane (rysunek 6.29).

**Formularz kontaktowy:**

Nazwisko:

Nowak

Imię:

Paweł

Zawód:

Malarz

Adres e-mail:

np@gmail.com

**Wykształcenie:**

- Podstawowe
- Średnie
- Wyższe

Zgadzam się na przetwarzanie moich danych osobowych

**Wyślij****Wyczyść****Rysunek 6.28.** Formularz z wypełnionymi polami

Odpowiedź z PHP:  
 Nowak Paweł  
 Malarz  
 np@gmail.com  
 Zostało wybrane wykształcenie: wyższe.

**Rysunek 6.29.** 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 6.87**

W przykładzie 6.85 formularz oprócz pól tekstowych zawiera również pola typu `radio`. Są to trzy pola wyboru, w których można zaznaczyć tylko jedną opcję z grupy opcji. Kliknięcie wartości powoduje zaznaczenie opcji, ale równocześnie usunięte zostaje 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.

Utworzmy skrypt PHP, który będzie rozpoznawał, jaka 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($_POST['wykszt'])` został użyty operator logicznej negacji `!`, czyli gdy pole nie zostanie wypełnione, warunek zwróci wartość `true` i zostanie wysłany komunikat *Proszę zaznaczyć pole Wykształcenie*. W przeciwnym razie za pomocą składni `{$_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 6.88

```
<!DOCTYPE HTML>
<html>
<head>
<title>Formularz językowy</title>
<meta charset="UTF-8">
<style>
#wyb {
    font-weight: bold;
    font-size: 12pt;
}
</style>
</head>
<body>
<form action="skrypt_form1.php" method="post">
```

```
Nazwisko i imię:<br>
<input type="text" name="nazw" value="" size="40"><br>

<p id="wyb">Wybór języka:</p>
<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 (`$_POST["języki"]`). Gdy kwadratowe nawiasy zostaną dodane w podany sposób: <select name="języki[]">, to elementy wybrane przez użytkownika w polu `języki[]` będą dostępne w tablicy `$_POST["języki"]`. Utworzony formularz został pokazany na rysunku 6.30.

Należy jeszcze dołączyć skrypt PHP (zapisany w pliku [skrypt\\_form1.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>";
}
```

**Rysunek 6.30.**  
Sposób wypełniania formularza

```

} 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 następuje wyświetlenie zawartości każdego elementu tablicy (rysunek 6.31). W tablicy znajdują się wartości znacznika `<option value=...>` wybrane przez użytkownika.

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.

**Kowalski Artur** zna:

- Język włoski
- Język hiszpański

#### Rysunek 6.31.

Po wystaniu formularza przeglądarka wyświetli przesłane dane

### Przykład 6.89

Po wprowadzonych zmianach kod zawierający formularz HTML będzie wyglądał tak:

```

<!DOCTYPE HTML>

<html>
    <head>
        <title>Formularz rejestracyjny</title>
        <meta charset="UTF-8">
        <style>
            #fo {
                font-weight: bold;
                font-size: 14pt;
            }
            p.wyk {
                font-weight: bold;
                font-size: 11pt;
            }
        </style>
    </head>
    <body>
        <form action="skrypt_form2.php" method="post">
            <p id="fo">Formularz kontaktowy:</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">

<p class="wyk">Wykształcenie:</p>
<input type="radio" value="podstawowe"
name="wykszt" checked> Podstawowe<br>
<input type="radio" value="średnie" name="wykszt"> Średnie<br>
<input type="radio" value="wyższe" name="wykszt"> Wyższe<br><br>

<p class="wyk">Wybór języka:</p>
<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="checkbox" name="opcje" maxlength="1">
Zgadzam się na przetwarzanie moich danych osobowych</p>
<input type="submit" value="Wyślij" name="wyslij">&nbsp; &nbsp;
<input type="reset" value="Wyczyść" name="zeruj">
</form>
</body>
</html>
```

A kod zawierający skrypt PHP (plik *skrypt\_form2.php*) będzie wyglądał tak:

```
<?php  
echo "Dziękujemy! Rejestracja przebiegła pomyślnie.<br>";  
echo "Wprowadzone dane:<br>";  
echo "Nazwisko: " . trim($_POST['nazw']) . "<br>";  
echo "Imię: " . trim($_POST['im']) . "<br>";  
echo "Zawód: " . trim($_POST['zaw']) . "<br>";  
echo "Adres e-mail: " . trim($_POST['adr']) . "<br>";  
  
if (!isset($_POST['wykszt'])) {  
    echo "Proszę zaznaczyć pole Wykształcenie";  
} else {  
    echo "Wykształcenie: " . trim($_POST['wykszt']) . "<br>";  
}  
  
echo "<p>Znajomość języków:</p>";  
  
if (!empty($_POST['języki'])) {  
    echo "<ul>";  
    foreach ($_POST['języki'] as $wartosc) {  
        echo "<li>$wartosc</li>";  
    }  
    echo "</ul>";  
} else {  
    echo "<p>".$_POST['nazw']."' nie zna żadnego języka.</p>";  
}  
?  
>
```

Po wprowadzeniu danych (rysunek 6.32) formularz zostanie wysłany, a w przeglądarce wyświetli się informacja o pomyślnym zarejestrowaniu użytkownika (rysunek 6.33).

<b>Formularz kontaktowy:</b>
Nazwisko: Marecka
Imię: Anna
Zawód: Lekarz
Adres e-mail: manna@gmail.com
<b>Wykształcenie:</b>
<input type="radio"/> Podstawowe <input type="radio"/> Średnie <input checked="" type="radio"/> Wyższe
<b>Wybór języka:</b>
Język angielski Język niemiecki Język francuski Język włoski
<input checked="" type="checkbox"/> Zgadzam się na przetwarzanie moich danych osobowych
<input type="button" value="Wyślij"/> <input type="button" value="Wyczyść"/>

**Rysunek 6.32.** Formularz z wprowadzonymi danymi

Dziękujemy! Rejestracja przebiegła pomyślnie.  
 Wprowadzone dane:  
 Nazwisko: Marecka  
 Imię: Anna  
 Zawód: Lekarz  
 Adres e-mail: manna@gmail.com  
 Wykształcenie: wyższe  
 Znajomość języków:  
 Marecka nie zna żadnego języka.

**Rysunek 6.33.** Wynik pobrania danych z formularza

### Ćwiczenie 6.16

Opracuj formularz przeznaczony do rejestracji 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.



## 6.9. Pliki cookies i sesje

### 6.9.1. 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 dostępu do pliku. 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 na serwer 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 wysłania informacji zawartych w cookie, to po otwarciu adresu pasującego do podanej domeny i ścieżki zawartość pliku cookie zostanie wysłana na serwer.

#### 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[,  
bezp_ścisłstwo[, 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 uplyną 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` sprawi, że plik będzie dostępny 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 6.90

```
<?php
setcookie("pismo", "Na skróty", time() + 3600, "/", "localhost", false);
if (isset($_COOKIE["pismo"])) {
    echo "<p>Jesteś naszym stałym klientem.</p>";
} else {
    echo "<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 wyrażony w sekundach czas ważności pliku (3600 s = 1 godz.). Ustawienie wartości `/` jako ścieżki dostępu oznacza, że plik będzie dostępny dla wszystkich stron na serwerze. Domena `localhost` wskazuje, 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. Jeśli czas ważności nie był ustalony, 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 `false`. Pozostałe argumenty funkcji powinny być takie same jak argumenty przy tworzeniu pliku.

### Przykład 6.91

```
setcookie("pismo", "Na skróty", time()-100, "/", "localhost", 0);
```

## 6.9.2. Zastosowania plików cookies

Jednym z zastosowań plików cookies może być przechowywanie na komputerze informacji o użytkowniku. Dzięki temu przy ponownej wizycie na stronie nie trzeba się powtórnie logować i podawać hasła. Pliki cookies mogą przechowywać również inne informacje, na przykład nazwisko i imię użytkownika, czas ostatniej wizyty, zawartość koszyka.

### Data ostatnich odwiedzin na stronie

#### Przykład 6.92

Wykorzystując plik cookie, określmy 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 ( $2\ 592\ 000 = 60\ s \times 60\ min \times 24\ godz. \times 30\ dni$ ).

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.

## Ćwiczenie 6.17

Utwórz skrypt, który będzie wyświetlał informacje o ostatnich odwiedzinach użytkownika na stronie przekazane do przeglądarki przez serwer.

### Rozwiązanie

```
<!DOCTYPE html>
<?php
$czas = date("Y-m-d G:i:s");
setcookie("wizyta", $czas, time() + 3600);
?>
<html>
<body>
<?php
if (isset($_COOKIE['wizyta'])) {
    $ostatnia = $_COOKIE['wizyta'];
    echo "Witamy ponownie! <br> Ostatni raz odwiedziłeś nas: " . $ostatnia;
} else {
    echo "Witamy na naszej stronie!";
}
?>
<p><strong>Uwaga:</strong> Potrzebne może być odświeżenie strony.</p>
</body>
</html>
```

## Przesyłanie danych użytkownika

### Ćwiczenie 6.18

Utwórz skrypt, który dane użytkownika przesłane za pomocą formularza będzie przekazywał do utworzonego pliku cookie. Formularz powinien zawierać imię i nazwisko użytkownika.

### Rozwiązanie

Skrypt *rejestracja.php* realizujący podane zadanie:

```
<?php
if (!isset($_COOKIE['dane']) && !isset($_POST['nazw'])) {
?>
<!DOCTYPE HTML>
```

```

<html>
<head>
<title>Dane_user</title>
<meta charset="UTF-8">
</head>
<body>
<form action="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 obsługiwane 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 — wówczas 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 365 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

### Ćwiczenie 6.19

Utwórz skrypt, który będzie zliczał wizyty użytkownika na stronie.

### Rozwiążanie

Skrypt `odwiedziny.php` realizujący podane zadanie:

```
<?php
if (!isset($_COOKIE ['odwiedz'])) {
    $odw = 1;
} else {
    $odw = intval($_COOKIE['odwiedz']) + 1;
}
setcookie("odwiedz", $odw, time() + 60 * 60 * 24 * 365);
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Odwiedziny</title>
<meta charset="UTF-8">
</head>
<body>
<?php
if ($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 `$odw` przechowująca liczbę odwiedzin ustawiana jest na 1. Jeżeli tak, to z tablicy `$_COOKIE` odczytywana jest wartość pliku `odwiedz`. Odczytana wartość za pomocą funkcji `intval()` jest zamieniana na liczbę całkowitą i zwiększa 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 `$odw`. Na stronie wyświetla się informacja o liczbie odwiedzin (rysunek 6.34). Dodatkowo sprawdzana jest wartość zmiennej `$odw`. Jeżeli wynosi ona 1, to wyświetlane zdanie będzie kończyło się słowem `raz`, dla pozostałych wartości będzie kończyło się słowem `razy`.

W ciągu ostatniego roku odwiedziłeś naszą stronę 3 razy.

**Rysunek 6.34.** Odczyt pliku cookie z przechowywaną liczbą odwiedzin

### Ćwiczenie 6.20

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.

## 6.9.3. 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 jego nazwa i wartość są zapisywane w stałej `$_SESSION`.

Wszystkie zmienne związane z użytkownikiem są przechowywane na serwerze w tablicy superglobalnej `$_SESSION`.

Do rozpoczęcia sesji najczęściej jest wykorzystywana funkcja `session_start()`. Funkcja zwykle nie przyjmuje żadnych argumentów, a jej wywołanie ma postać:

```
session_start();
```

W każdym skrypcie, który będzie korzystał z sesji, należy wywołać tę funkcję.

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 przyjmuje 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 6.93

```
<?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`, jeśli 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:

```
session_unset()
```

która usuwa wszystkie zmienne sesji, lub

```
unset($_SESSION['nazwa_zmiennej'])
```

która usuwa podaną zmienną sesji.

Zmienne utworzone w sesji są przechowywane przez PHP. Zapisane dane mogą zostać odczytane przez inne skrypty w obrębie tej samej sesji.

## Przykład 6.94

```
<?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.

## 6.9.4. Praktyczne zastosowania sesji

### Uwierzytelnienie użytkownika

#### Ćwiczenie 6.21

Przyjmij, że do strony internetowej mają dostęp tylko zalogowani użytkownicy. Wykorzystaj mechanizm sesji do przeprowadzenia autoryzacji użytkownika.

#### Rozwiązanie

Przygotuj 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` — procedury wylogowania. W tworzonym rozwiążaniu będzie sprawdzane uwierzytelnienie tylko jednego użytkownika o nazwie `janek` i haśle `jan23`.

Pierwszym krokiem będzie weryfikacja danych użytkownika. Po ich sprawdzeniu użytkownik będzie mógł zalogować się i korzystać ze strony. Przed opuszczeniem strony użytkownik będzie musiał się wylogować.

Skrypt `loguj.php`:

```
<?php
session_start();
if (isset($_SESSION['log'])) {
    header('location: strona.php');
    exit();
} elseif (isset($_POST['nazwa']) && isset($_POST['haslo'])) {
```

```
if ($_POST['nazwa'] == 'janek' && $_POST['haslo'] == 'jan23') {
    $_SESSION['log'] = $_POST['nazwa'];
    header('location: strona.php');
    exit();
} else {
    echo "Nieprawidłowe dane logowania<br>";
}
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Autoryzacja danych</title>
<meta charset="UTF-8">
<style>
p.fo {
    font-weight: bold;
    font-size: 11pt;
}
#log {
    font-weight: bold;
    font-size: 14pt;
}
</style>
</head>
<body>
<div>
<form action="http://localhost/loguj.php" method="post">
<p id="log">Logowanie</p>
<p class="fo">Nazwa użytkownika:</p>
<input type="text" name="nazwa" value="" size="25"><br>
<p class="fo">Hasło:</p>
<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 zweryfikować 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 6.35). Zmiennej sesji `$log` zostaje przypisana nazwa zalogowanego użytkownika i użytkownik zostaje przekierowany do strony głównej.

Jeżeli weryfikacja danych jest negatywna, wyświetla się komunikat informujący o nieprawidłowych danych logowania, a następnie wyświetla się formularz logowania (rysunek 6.36).

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;
}
?>
```

**Logowanie**

Nazwa użytkownika:

Hasło:

### Rysunek 6.35.

Próba logowania do witryny

Nieprawidłowe dane logowania

**Logowanie**

Nazwa użytkownika:

Hasło:

### Rysunek 6.36.

Niepoprawne dane logowania

```
<!DOCTYPE HTML>
<html>
<head>
<title>Strona główna</title>
<meta charset="UTF-8">
</head>
<body>
<?php
$imie = ucfirst($_SESSION['log']);
echo "Witaj " . $imie;
?>
<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 stronę próbuje otworzyć niezalogowany użytkownik — 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 6.37).

Próba wylogowania powoduje wywołanie kolejnego skryptu — *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;
```

Jesteś na stronie głównej

Przed opuszczeniem strony wyloguj się!

[Wyloguj](#)

### Rysunek 6.37.

Prawidłowo zalogowany użytkownik ma dostęp do strony głównej

```

}

$s = session_destroy();

?>

<!DOCTYPE HTML>

<html>
<head>
<title> Koniec_sesji</title>
<meta 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 przy użyciu funkcji `session_destroy` (rysunek 6.38).

Wylogowałeś się ze strony.  
[Logowanie](#)

### Rysunek 6.38.

Zakończenie sesji.  
 Wylogowanie

## Ćwiczenie 6.22

Zmodyfikuj skrypty z ćwiczenia 6.21 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`, jeśli użytkownik o podanym loginie i haśle istnieje. Wykorzystaj ją do zdefiniowania nowych funkcji skryptów.

## Ćwiczenie 6.23

Przy założeniu, że Twoja witryna internetowa składa się z kilku stron, napisz kod, który pozwoli na dostęp do podstron tylko ze strony głównej. Możliwość wejścia bezpośrednio na wybraną stronę (z pominięciem strony głównej) powinna zostać zablokowana.

## Koszyk zakupów

### Ćwiczenie 6.24

Wykorzystaj mechanizm sesji do utworzenia listy wybranych przez klienta produktów w sklepie internetowym.

Przygotuj dwa skrypty. W pierwszym skrypcie (*lista.php*) utwórz formularz, za pomocą którego możliwe będzie wybranie z listy różnych produktów. Wybrane produkty przechowuj w zmiennych sesji zapisanych w postaci tablicy. W drugim skrypcie (*koszyk.php*) wyświetl listę produktów wybranych przez użytkownika i umieszczonych w koszyku.

### Rozwiążanie

Skrypt *lista.php*:

```
<?php
session_start();
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Koszyk zakupów</title>
<meta 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="lista.php" method="post">
<p>Wybór produktu:</p>
<p>
<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>
<option value="Pendrive"> Pendrive</option>
<option value="Komputer"> Komputer</option>
</select></p>
<p>Wybierz produkty, trzymając wciśnięty klawisz Ctrl.</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 `$_POST['lista']`, a wynik będzie zapisany w zmiennej `$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 po pobraniu w sesji danych z tablicy `$_SESSION` należy zamienić je na wartości (funkcja `unserialize()`).

Przed połączeniem tablic dane pobrane z tablicy `$_SESSION['koszyk']` zostaną przekształcone na wartości (funkcja `unserialize()`). Po połączeniu tablic funkcja `array_unique()` usunie powtarzające się produkty umieszczone w koszyku. Następnie tablica `$_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 6.39).

**Lista artykułów**

Wybór produktu:

- Monitor
- Drukarka
- Klawiatura
- Myszka
- Głośniki
- Kamera internetowa
- Słuchawki
- Stacja DVD
- Dysk twardy

Wybierz produkty, trzymając wcisnięty klawisz Ctrl

[Przejdz do koszyka](#)

**Rysunek 6.39.** Formularz z listą dostępnych produktów

Skrypt `koszyk.php`:

```
<?php
session_start();
?>
<!DOCTYPE HTML>
<html>
<head>
<title>Koszyk</title>
<meta 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 `$_SESSION['koszyk']`. Jeżeli istnieje, w pętli wyświetlana jest lista wartości tablicy `$_SESSION['koszyk']`. Wynik działania kodu został pokazany na rysunku 6.40.

### Ćwiczenie 6.25

Zmodyfikuj kod skryptu z ćwiczenia 6.24 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.

#### Zawartość koszyka

- Klawiatura
- Głośniki
- Słuchawki

[Przejdź do listy produktów](#)

#### Rysunek 6.40.

Lista zamówionych produktów

## 6.10. Bazy danych w PHP

Gdy tworzymy aplikacje internetowe, niezbędne dane możemy przechowywać w plikach. Ale kiedy liczba danych zwiększa się, rozwiązaniem jest zastosowanie bazy danych. Pliki nadal mogą być wykorzystywane, na przykład do przechowywania podstawowych ustawień aplikacji, ale do gromadzenia i przetwarzania większych ilości danych używa się relacyjnych baz danych. Do ich obsługi wykorzystywane są systemy zarządzania bazami danych. Najpopularniejszym systemem zarządzania bazami danych, który współpracuje z PHP, jest MySQL. Bazy danych można obsługiwać w sposób proceduralny lub obiektowy.

Aby możliwa była współpraca języka PHP z systemem MySQL, musi istnieć baza, z którą można się połączyć.

Użytkownik, który chce połączyć się z taką bazą, musi mieć login i hasło, znać nazwę oraz adres tej bazy danych.

### UWAGA

We wszystkich skryptach tworzonych w tym rozdziale wykorzystywana jest baza danych *ksiegarnia* opisana w podręczniku *Kwalifikacja INF.03. Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych. Część 2. Projektowanie i administrowanie bazami danych. Podręcznik do nauki zawodu technik informatyk i technik programista*.

Współpraca PHP z MySQL może odbywać się przy użyciu:

- MySQLi (proceduralnie lub obiektowo),
- PDO (PHP Data Objects).

MySQLi obsługuje proceduralny i obiektowy paradymat programowania. Interfejs proceduralny jest podobny do rozszerzenia MySQL.

## 6.10.1. MySQLi proceduralny

### Połączenie z bazą danych

#### Nawiązanie połączenia

Do połączenia z bazą danych służy funkcja `mysqli_connect()`, która ma postać:

```
mysqli_connect('nazwa_hosta', 'nazwa_użytkownika', 'hasło', 'baza')
```

*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. Jeśli 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,
- *baza* — określa bazę, do której będą wysyłane zapytania.

### Przykład 6.95

```
<?php
$serwer = 'localhost';
$login = 'root';
$haslo = '';
$baza = 'ksiegarnia';
```

```
$do_bazy = mysqli_connect($serwer, $login, $haslo, $baza);

if (mysqli_connect_errno()) {
    exit("Błąd połączenia z serwerem MySQL:" . mysqli_connect_error());
} else {
    echo "Połączono z serwerem baz danych . <br>";
}

?>
```

W podanym kodzie po wywołaniu funkcji `mysqli_connect()` sprawdzane jest, czy połączenie zostało nawiązane. Jeżeli tak, to wyświetli się komunikat o uzyskany połączeniu, a w przeciwnym razie wyświetli się komunikat o braku tego połączenia i program zakończy swoje działanie. Funkcja `mysqli_connect_errno()` zwraca kod błędu połączenia, funkcja `mysqli_connect_error()` zwraca opis błędu połączenia.

### Kończenie połączenia

Do zakończenia połączenia z bazą danych używana jest funkcja `mysqli_close()` w postaci:

```
mysqli_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 `mysqli_close()` nie zostanie wywołana, połączenie zostanie automatycznie zamknięte, gdy skrypt zakończy swoje działanie. Funkcja zwróci wartość `true`, jeśli operacja zamknięcia połączenia zakończy się prawidłowo, a w przeciwnym razie zwróci wartość `false`.

### Przykład 6.96

```
<?php

$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');

if (mysqli_connect_errno()) {
    exit("Błąd połączenia z serwerem MySQL:" . mysqli_connect_error());
} else {
    echo "Połączono z serwerem baz danych";
    mysqli_close($do_bazy);
}

?>
```

## Zapytania do bazy danych

Po nawiązaniu połączenia z wybraną bazą danych można wysyłać do niej zapytania. Są one przesyłane za pomocą funkcji `mysqli_query()` zapisanej w postaci:

```
mysqli_query(połączenie, 'zapytanie', stan_wyniku)
```

- `połączenie` — określa połączenie z bazą,
- `zapytanie` — zawiera treść zapytania,
- `stan_wyniku` — opcjonalne, określa sposób przetwarzania wyniku (domyślnie `MYSQLI_STORE_RESULT`).

Zwracane przez funkcję wartości zależą od typu zapytania. Jeżeli zapytanie pobierało dane (na przykład `SELECT`), to zwracany jest obiekt `mysqli_result`. Jeżeli zapytanie nie pobierało danych (na przykład `INSERT`), zwracana jest wartość `true`. Jeżeli wykonanie zapytania się nie powiodło, zwracana jest wartość `false`.

### Przykład 6.97

Pobieranie danych:

```
$wynik = mysqli_query($do_bazy, 'SELECT * FROM Klient');
```

### Przykład 6.98

Wstawianie danych:

```
$wstaw = mysqli_query($do_bazy, "INSERT INTO Klient VALUES (NULL, 'Anna',
'Polak', '87-100', 'Toruń')";
```

## Zapytanie pobierające dane

Dla zapytań typu `SELECT` pobierających dane zwracany jest identyfikator zasobów. Może on zostać wykorzystany do odczytania tych danych. Można użyć w tym celu funkcji `mysqli_fetch_row()` lub `mysqli_fetch_array()`.

Funkcja `mysqli_fetch_row()` po wywołaniu odczytuje jeden wiersz tabeli i zapisuje go w kolejnym wierszu tablicy. Aby odczytać całą zawartość tabeli, należy wywołać tę funkcję w pętli. Gdy wszystkie dane zostaną odczytane, funkcja zwróci wartość `false`.

### Przykład 6.99

```
while ($wiersz = mysqli_fetch_row($wynik)) {
    // przetwarzanie danych z zapytania
}
```

W podanym przykładzie zmienna `$wynik` zawiera identyfikator zasobów zwróconych przez zapytanie. W pętli zostaną odczytane kolejne wiersze tabeli zwrócone przez zapytanie i umieszczone w tablicy `$wiersz`.

Funkcja `mysqli_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 6.100

```
$ile = mysqli_num_rows($wynik);
for ($i = 0; $i < $ile; $i++) {
    $wiersz = mysqli_fetch_row($wynik);
    // przetwarzanie danych z zapytania
}
```

W pętli zostaną odczytane kolejne wiersze tabeli zwrócone przez zapytanie i umieszczone w tablicy `$wiersz`.

Funkcja `mysqli_fetch_array()` po wywołaniu odczytuje całą tabelę i zwraca tablicę asocjacyjną. Kluczami tej tablicy są nazwy kolumn zwróconej przez zapytanie tabeli.

### Przykład 6.101

```
$tab = mysqli_fetch_array($wynik);
```

### Ćwiczenie 6.26

Dysponujesz utworzoną w MySQL bazą danych `ksiegarnia`. Odczytaj dane z tabeli `Klient`.

### Rozwiązanie

```
<!DOCTYPE HTML>
<html>
<head>
<title>Odczyt danych z tabeli Klient</title>
<meta charset="UTF-8">
</head>
<body>
<?php
$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');

if (!$do_bazy) {
    echo "Błąd połączenia z serwerem MySQL.<br>";
}
</body>
</html>
```

```
<?php
exit;
} else {
    $zapytanie = mysqli_query($do_bazy, 'SELECT * FROM Klient');
    if (!$zapytanie) {
        mysqli_close();
        echo "Błąd w zapytaniu<br>";
    }
    ?>
</body>
</html>
<?php
exit;
} else {
    ?>
<table>
<tr>
<td>Nazwisko</td>
<td>Imię</td>
<td>Kod pocztowy</td>
<td>Miejscowość</td>
<td>Ulica</td>
<td>Nr domu</td>
<td>PESEL</td>
<td>Telefon</td>
<td>Adres e-mail</td>
<td>Id klienta</td>
</tr>
<?php
while ($wiersze = mysqli_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

mysqli_close($do_bazy);

}

}

?>

</body>

</html>

```

W podanym przykładzie po połączeniu z serwerem i wybraniu bazy danych zostaje wywołana funkcja `mysqli_query()`, która wysyła do serwera zapytanie SQL (`SELECT * FROM Klient`). Wynik zwrócony przez funkcję zostaje przypisany do zmiennej `$zapytanie`. Jeżeli wynik ma wartość `false`, to znaczy, że w zapytaniu wystąpił błąd, i skrypt kończy działanie. Jeśli 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 `mysqli_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 6.41.

Nazwisko	Imię	Kod pocztowy	Miejscowość	Ulica	Nr domu	PESEL	Telefon	Adres e-mail	Id klienta
Kowalski	Jan	05-120	Warszawa	Mickiewicza	1	67010123456 456793456	jan@o2.pl		1
Nowak	Andrzej	87-100	Toruń	Szeroka	34	78021203121	44444	an@gmail.com	2
Kowalski	Maciej	05-120	Warszawa	Sowa	25	43562902821			3

**Rysunek 6.41.** 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.

## Ćwiczenie 6.27

Utwórz skrypt, który zwróci z bazy dane osób, które mają nazwisko Kowalski.

### Rozwiązanie

```
<?php
$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');
if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}

$zapytanie = mysqli_query($do_bazy, "SELECT * FROM Klient WHERE Nazwisko
= 'Kowalski'");

if (!$zapytanie === true) {
    mysqli_close();
    exit("Błąd w zapytaniu<br>");
}

while ($tab = mysqli_fetch_array($zapytanie)) {
    echo $tab['nazwisko'] . " " . $tab['imie'];
    echo "<br>";
}

mysqli_close($do_bazy);
?>
```

W wyniku wykonania podanego skryptu z bazy danych zwrócone zostaną dane osób, które mają nazwisko Kowalski. Dane zostały pobrane za pomocą funkcji `mysqli_fetch_array()`, która zapisała je w tablicy asocjacyjnej `$tab`. Kluczami tablicy `$tab` są nazwy kolumn tabeli SQL. Konstrukcje `$tab['nazwisko']` i `$tab['imie']` odwołują się do wartości zapisanych w tablicy i wyświetlały nazwiska i imiona klientów wybranych instrukcją `SELECT`.

### Zapytanie wstawiające dane

Dla zapytań modyfikujących zawartość tabel funkcja `mysqli_query()` zwraca wartość `true` lub `false`. Natomiast za pomocą funkcji `mysqli_affected_rows()` można odczytać, ile wierszy zostało zmodyfikowanych przez zapytanie. Funkcja ma postać:

```
mysqli_affected_rows([identyfikator])
```

Argument `identyfikator` jest opcjonalny i zawiera identyfikator połączenia z serwerem. Jeżeli zostanie pominięty, działania będą dotyczyć ostatnio otwartego połączenia.

Zapytanie dodające nowe dane do tabeli jest realizowane przez polecenie `INSERT INTO`.

### Ćwiczenie 6.28

Do tabeli `Klient` w bazie danych `ksiegarnia` dodaj nowego klienta o następujących danych: Anna Lisek, miejscowości: 34-100 Wadowice, ulica Lwowska, 11, PESEL: 95031203267, tel. 936789453, adres e-mail: ania@gmail.com.

### Rozwiążanie

```
<?php
$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');
if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}

$dodaj = mysqli_query($do_bazy, "INSERT INTO Klient VALUES('Lisek',
'Anna', '34-100', 'Wadowice', 'Lwowska', '11', '95031203267',
'936789453', 'ania@gmail.com', NULL)");
if (!$dodaj === true) {
    mysqli_close($do_bazy);
    exit("Błąd w zapytaniu<br>");
}
$ile = mysqli_affected_rows($do_bazy);
echo "Liczba rekordów dodanych do tabeli Klient wynosi: $ile<br>";
mysqli_close($do_bazy);
?>
```

W podanym kodzie po prawidłowym połączeniu z serwerem i wybraniu bazy danych zostaje utworzona zmienna `$dodaj`, w której jest przechowywany wynik zapytania SQL (`INSERT INTO`). Za pomocą funkcji `mysqli_query()` jest ono wysyłane do serwera. Instrukcja `if` sprawdza, czy funkcja `mysqli_query()` zwróciła wartość `false`, co oznacza, że zapytanie nie zostało obsłużone. Wtedy skrypt kończy działanie. Jeżeli zapytanie zostało przyjęte, to zmiennej `$ile` przypisywana jest wartość zwrócona przez funkcję `mysqli_affected_rows()`, określającą, ile wierszy w bazie danych zostało zmienionych. Ta informacja zostanie wyświetlona w przeglądarce.

Inną metodą dodawania nowych danych do tabeli jest wprowadzanie ich za pomocą formularza.

## Ćwiczenie 6.29

Utwórz skrypt, który dane pobrane z formularza (rysunek 6.42) będzie dodawał w bazie *ksiegarnia* do tabeli Klient.

**Rejestracja klienta:**

Nazwisko:

Imię:

Kod pocztowy:

Miejscowość:

Ulica:

Numer domu:

PESEL:

Numer telefonu:

Adres e-mail:

**Rysunek 6.42.** Formularz do wprowadzania danych do bazy

### Rozwiążanie

Utwórz dwa skrypty. Skrypt HTML (*form\_klient.html*) będzie zawierał formularz do wprowadzania danych klienta. W tym skrypcie dane wpisane do formularza zostaną przesyłane do pliku *dodaj\_klienta.php*.

Skrypt PHP (*dodaj\_klienta.php*) doda otrzymane z formularza dane do tabeli Klient.

Skrypt *form\_klient.html*:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Rejestracja klienta</title>
<meta charset="UTF-8">
<style>
#wyb {
    font-weight: bold;
```

```
    font-size: 12pt;  
}  
</style>  
</head>  
<body>  
<form action="dodaj_klienta.php" method="post">  
<p id="wyb">Rejestracja klienta:</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>
```

**Skrypt *dodaj\_klienta.php*:**

```

<?php
$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');
if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}

$nazwisko = $_POST['nazw'];
$imie = $_POST['im'];
$kod = $_POST['kod'];
$miasto = $_POST['mia'];
$ulica = $_POST['ul'];
$nr = $_POST['nr'];
$pesel = $_POST['pe'];
$tel = $_POST['tel'];
$mail = $_POST['adr'];

$dodaj = "INSERT INTO Klient VALUES ('$nazwisko', '$imie', '$kod',
'$miasto', '$ulica', '$nr', '$pesel', '$tel', '$mail', NULL)";
$zapytanie = mysqli_query($do_bazy, $dodaj);
if (!$zapytanie === true) {
    echo "Nowy klient nie został dodany do bazy!";
} else {
    echo "Klient " . $nazwisko . " " . $imie . " został dodany do bazy.";
}
mysqli_close($do_bazy);
?>

```

Podany skrypt jest podobny do skryptu z ćwiczenia 6.28. Jedyna zmiana to użycie w instrukcji `INSERT INTO` zmiennej `$_POST['nazwa_pola']` w miejscu, gdzie poprzednio wstawione były wartości danych.

**Aktualizowanie danych**

Dane można aktualizować za pomocą instrukcji `UPDATE`.

**Ćwiczenie 6.30**

Zaktualizuj w tabeli `Klient` dane klienta Nowak Andrzej. Dane do aktualizacji pobierz z formularza dostępnego na stronie internetowej.

## Rozwiążanie

Aby zaktualizować dane klienta, należy wykonać kilka czynności — wyszukać jego dane 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.

Skrypt [szukaj\\_klienta.php](#):

```
<?php

$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');

if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}

$zapytanie = mysqli_query($do_bazy, "SELECT * FROM Klient
WHERE Nazwisko = 'Nowak' and Imie = 'Andrzej'");
$ile = mysqli_num_rows($zapytanie);

if (!$zapytanie === true) {
    mysqli_close($do_bazy);
    exit("Błąd w zapytaniu.");
}

if ($ile = 0) {
    exit("Brak danych");
}

if ($ile > 1) {
    exit("Nie można wyodrębnić danych");
}

$wiersz = mysqli_fetch_array($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'];
?>

<!DOCTYPE HTML>
<html>
<head>
<title>Modyfikacja danych</title>
<meta charset="UTF-8">
<style>
p {
    font-weight: bold;
    font-size: 12pt;
}
</style>
</head>
<body>
<form action="aktualizuj.php" method="post">
<p>Modyfikacja danych klienta:</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 `mysqli_query()`, która wysyła do serwera zapytanie SQL (`SELECT * FROM Klient WHERE Nazwisko = 'Nowak' and Imie = 'Andrzej'`). W wyniku zostaną zwrócone dane wybranego klienta. Zwrócone dane zostają przypisane do zmiennej `$zapytanie`. Następnie za pomocą funkcji `mysqli_num_rows()` sprawdza się, 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. Jeśli zostały odczytane dane jednego klienta, to będą one przypisane do zmiennych i nastąpi przejście do kodu HTML. W kodzie HTML nastąpi wyświetlenie formularza z bieżącymi danymi pobranymi z utworzonych zmiennych (rysunek 6.43). Użytkownik może zaktualizować swoje dane i przesyłać je do skryptu [aktualizuj.php](#).

**Modyfikacja danych klienta:**

Nazwisko:  
Nowak

Imię:  
Andrzej

Kod pocztowy:  
87-100

Miejscowość:  
Toruń

Ulica:  
Szeroka

Numer domu:  
34

PESEL:  
78021203121

Numer telefonu:  
44444

Adres e-mail:  
an@gmail.com

Aktualizuj

**Rysunek 6.43.** Bieżące dane pobrane z bazy danych do aktualizacji

Kolejnym krokiem będzie aktualizacja danych w bazie.

Skrypt *aktualizuj.php*:

```
<?php  
$do_bazy = mysqli_connect('localhost', 'root', '', 'ksiegarnia');  
if (!$do_bazy) {  
    exit("Błąd połączenia z serwerem MySQL.");  
}  
  
$nazwisko = $_POST['nazw'];  
$imie = $_POST['im'];  
$kod = $_POST['kod'];  
$miasto = $_POST['mia'];  
$ulica = $_POST['ul'];  
$nr = $_POST['nr'];  
$pesel = $_POST['pe'];  
$tel = $_POST['tel'];  
$mail = $_POST['adr'];  
  
$aktualny = "UPDATE Klient SET nazwisko = '$nazwisko', imie = '$imie',  
kod_pocztowy = '$kod', miejscowosc = '$miasto', ulica = '$ulica',  
nr_domu = '$nr', PESEL = '$pesel', telefon = '$tel', adres_e_mail  
= '$mail' WHERE Nazwisko = 'Nowak' and Imie = 'Andrzej'";  
  
$zapytanie = mysqli_query($do_bazy, $aktualny);  
if (!$zapytanie === true) {  
    echo "Błąd zapytania! Dane nie zostały zaktualizowane!";  
} else {  
    echo "Dane zostały zaktualizowane.";  
}  
mysqli_close($do_bazy);  
?>
```

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 mysqli\_query().

### Przykład 6.102

```
<?php
$do_bazy = mysqli_connect('localhost', 'root', '');
if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}
$baza_sql = "CREATE DATABASE moje_kino";
if (mysqli_query($do_bazy, $baza_sql)) {
    echo "Baza została utworzona";
} else {
    echo "Błąd! Nie można utworzyć bazy";
}
mysql_close($do_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 mysqli\_query().

### Przykład 6.103

```
<?php
$do_bazy = mysqli_connect('localhost', 'root', '', 'moje_kino');
if (!$do_bazy) {
    exit("Błąd połączenia z serwerem MySQL.");
}
$tab = "CREATE TABLE Film (Tytul_filmu VARCHAR(30), Rezyser
VARCHAR(30), Rok_prod VARCHAR(4))";
if (mysqli_query($do_bazy, $tab)) {
    echo "Tabela Film została utworzona";
} else {
    echo "Błąd! Nie można utworzyć tabeli Film";
}
mysql_close($do_bazy);
?>
```

W przykładzie został pokazany fragment kodu PHP tworzący tabelę Film.

## 6.10.2. MySQLi zorientowany obiektowo

### Połączenie z bazą danych

#### Nawiązanie połączenia

Do połączenia z bazą danych służy funkcja `mysqli()`, która ma postać:

```
new mysqli('nazwa_hosta', 'nazwa_użytkownika', 'hasło', 'baza')
```

#### Przykład 6.104

```
<?php
$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');
if (!$do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:" . $do_bazy->connect_error);
} else {
    echo "Połączono z serwerem baz danych.";
}
?>
```

W podanym kodzie podobnie po wywołaniu funkcji `mysqli()` sprawdzane jest, czy połączenie zostało nawiązane. Jeżeli tak, to wyświetli się komunikat o uzyskanym połączeniu, w przeciwnym razie wyświetli się komunikat o braku tego połączenia i program zakończy swoje działanie.

#### Kończenie połączenia

Do zakończenia połączenia z bazą danych używana jest funkcja `close()` w postaci:

```
identyfikator->close()
```

#### Przykład 6.105

```
<?php
$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');
if (!$do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:" . $do_bazy->connect_error);
} else {
    echo "Połączono z serwerem baz danych.";
}
$do_bazy->close();
?>
```

Jeżeli funkcja `close()` nie zostanie wywołana, połączenie zostanie automatycznie zamknięte, gdy skrypt zakończy swoje działanie.

## Zapytania do bazy danych

Po nawiązaniu połączenia z wybraną bazą danych można wysyłać do niej zapytania. Są one przesyłane za pomocą funkcji `query()` zapisanej w postaci:

```
identyfikator->query('zapytanie')
```

Argumentem funkcji jest treść zapytania. Zwarcane przez funkcję wartości zależą od typu zapytania. Jeżeli zapytanie pobierało dane (na przykład `SELECT`), to zwarcany jest obiekt `mysqli_result`. Jeżeli zapytanie nie pobierało danych (na przykład `INSERT`), zwarcana jest wartość `true`. Jeśli wykonanie zapytania się nie powiodło, zwarcana jest wartość `false`.

### Zapytanie pobierające dane

Dla zapytań typu `SELECT` pobierających dane zwarcany jest identyfikator zasobów. Może on zostać wykorzystany do odczytania tych danych. Do odczytania danych pobranych zapytaniem `SELECT` można użyć funkcji `fetch_assoc()`.

#### Przykład 6.106

```
while ($wiersze = $wynik->fetch_assoc()) {  
    // 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 `num_rows()` zwraca jako wartość liczbę wierszy znajdujących się w wyniku zapytania. Może być użyta do zbudowania pętli odczytującej kolejne wiersze zwrócone przez zapytanie.

Funkcja `fetch_assoc()` po wywołaniu odczytuje całą tabelę i zwraca tablicę asocjalną. Kluczami tej tablicy są nazwy kolumn zwróconej przez zapytanie tabeli.

#### Przykład 6.107

Dysponujemy utworzoną w MySQL bazą danych `ksiegarnia`. Odczytamy dane z tabeli `Klient`.

```
<!DOCTYPE HTML>  
<html>  
<head>  
<title>Odczyt danych z tabeli Klient</title>  
<meta charset="UTF-8">  
</head>  
<body>
```

```

<?php

$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');

if ($do_bazy->connect_error) {

    exit("Błąd połączenia z serwerem MySQL:");

}

$zapytanie = 'SELECT * FROM Klient';

$wynik = $do_bazy->query($zapytanie);

if ($wynik->num_rows > 0) {

    while ($wiersze = $wynik->fetch_assoc()) {

        echo $wiersze['nazwisko'] . " " . $wiersze['imie'] . "<br>";

    }

} else {

    echo "0 wierszy";

}

$do_bazy->close();

?>

</body>

</html>

```

W podanym przykładzie po połączeniu z serwerem i wybraniu bazy danych zostaje wywołana funkcja `query()`, która wysyła do serwera zapytanie SQL (`SELECT * FROM Klient`). Wynik zwrocony przez funkcję zostaje przypisany zmiennej `$wynik`. Jeżeli wynik wynosi 0, to znaczy, że zapytanie zwróciło zero wierszy, i skrypt kończy działanie. Jeśli wszystko przebiegło poprawnie, to w kodzie PHP wyniki zapytania są pobierane w pętli za pomocą funkcji `fetch_assoc()`, która pobiera kolejne wiersze tabeli i umieszcza je w tablicy asocjacyjnej.

## Klauzula WHERE

Klauzula `WHERE` służy do wybierania rekordów, które spełniają określone kryterium.

### Przykład 6.108

```

<?php

$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');

if ($do_bazy->connect_error) {

    exit("Błąd połączenia z serwerem MySQL:");

}

$zapytanie = "SELECT * FROM Klient WHERE Nazwisko = 'Kowalski'";

$wynik = $do_bazy->query($zapytanie);

```

```

if (!$wynik === true) {
    $do_bazy->close();
    exit("Błąd w zapytaniu<br>");
}

while ($tab = $wynik->fetch_assoc()) {
    echo $tab['nazwisko'] . " " . $tab['imie'];
    echo "<br>";
}

$do_bazy->close();
?>

```

W wyniku wykonania podanego skryptu z bazy danych zwrócone zostaną nazwiska i imiona osób, które mają nazwisko Kowalski. Dane zostały pobrane za pomocą funkcji `fetch_assoc()`, która zapisała je w tablicy asocjacyjnej `$tab`. Kluczami tablicy `$tab` są nazwy kolumn tabeli SQL. Konstrukcje `$tab['nazwisko']` i `$tab['imie']` odwołują się do wartości zapisanych w tablicy i wyświetlały nazwiska i imiona klientów wybranych instrukcją `SELECT`.

## Zapytanie wstawiające dane

Dla zapytań modyfikujących zawartość tabel funkcja `query()` zwraca wartość `true` lub `false`.

Zapytanie dodające nowe dane do tabeli jest realizowane przez polecenie `INSERT INTO`.

### Przykład 6.109

Do tabeli `Klient` w bazie danych *ksiegarnia* zostanie dodany nowy wpis.

```

<?php
$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');

if ($do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:");
}

$dodaj = "INSERT INTO Klient (imie, nazwisko, miejscowosc, adres_e_mail)
VALUES ('Hanna', 'Nowicka', 'Kraków', 'hania@gmail.com')";

$wynik = $do_bazy->query($dodaj);

if ($wynik === true) {
    echo "Rekord został dodany";
} else {
    echo "Błąd w zapytaniu. Rekord nie został dodany";
}

$do_bazy->close();
?>

```

Inną metodą dodawania nowych danych do tabeli jest wprowadzanie ich za pomocą formularza.

### Ćwiczenie 6.31

Utwórz podobnie jak w ćwiczeniu 6.29 skrypt, który dane pobrane z formularza będzie dodawał w bazie [ksiegarnia](#) do tabeli Klient. W skrypcie zastosuj polecenia mysqli zorientowanego obiektowo.

### Aktualizowanie danych

Aktualizowanie danych jest realizowane za pomocą instrukcji UPDATE.

### Przykład 6.110

W tabeli Klient zostaną zaktualizowane dane klienta Nowicka Hanna.

```
<?php
$do_bazy = new mysqli('localhost', 'root', '', 'ksiegarnia');
if ($do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:");
}
$mody = "UPDATE Klient SET ulica = 'Franciszkańska',
nr_domu = 2 WHERE Nazwisko = 'Nowicka' and Imie = 'Hanna'";
$wynik = $do_bazy->query($mody);
if ($wynik === true) {
    echo "Dane zostały zaktualizowane.";
} else {
    echo "Błąd zapytania! Dane nie zostały zaktualizowane";
}
$do_bazy->close();
?>
```

### Ćwiczenie 6.32

Utwórz podobnie jak w ćwiczeniu 6.30 skrypt, który zaktualizuje w tabeli Klient dane wybranego klienta. Dane do aktualizacji pobierz z formularza dostępnego na stronie internetowej. W skrypcie zastosuj polecenia mysqli zorientowanego obiektowo.

### Tworzenie bazy danych

Do tworzenia bazy danych jest używana instrukcja CREATE DATABASE. Należy dodać ją do funkcji query().

## Przykład 6.111

```
<?php
$do_bazy = new mysqli('localhost', 'root', '');
if ($do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:");
}

$baza = "CREATE DATABASE szkola";
if ($do_bazy->query($baza) === true) {
    echo "Baza została utworzona";
} else {
    echo "Błąd! Nie można utworzyć bazy";
}
$do_bazy->close();
?>
```

W przykładzie został pokazany fragment kodu PHP tworzący bazę danych *szkola*.

## Tworzenie tabeli

Do tworzenia tabeli w istniejącej bazie danych jest używana instrukcja SQL CREATE TABLE. Należy dołączyć ją do funkcji query().

## Przykład 6.112

```
<?php
$do_bazy = new mysqli('localhost', 'root', '');
if ($do_bazy->connect_error) {
    exit("Błąd połączenia z serwerem MySQL:");
}

$tab = "CREATE TABLE Uczen (nazwisko VARCHAR(30),
imie VARCHAR(30), email VARCHAR(40))";
if ($do_bazy->query($tab) === true) {
    echo "Tabela Uczen została utworzona";
} else {
    echo "Błąd! Nie można utworzyć tabeli Uczen";
}
$do_bazy->close();
?>
```

W przykładzie został pokazany fragment kodu PHP tworzący tabelę Uczen.



## 6.11. 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 używać tych samych metod niezależnie od wykorzystywanej bazy. Kolejną zaletą jest walidacja danych wysyłanych w zapytaniach oraz filtrowanie tych danych pod kątem zabezpieczeń przed atakami.

### 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 z bazą. 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, na przykład 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 6.113

```
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 6.114

```
<?php  
$user = "root";  
$pass = "";
```

```
$dsn = "mysql:host=localhost;dbname=ksiegarnia";

try {
    $pdo = new PDO($dsn, $user, $pass);
    echo 'Połączenie nawiązane!';
} catch (PDOException $e) {
    echo 'Błąd połączenia: ' . $e->getMessage();
    exit;
}
?>
```

W podanym przykładzie ciąg określający 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 generuje PHP, wyświetli nazwę użytkownika i hasło do bazy danych.

### UWAGA

PDO ma 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 komunikat będzie zawierał login i hasło do bazy danych.

## Kończenie połączenia

Połączenie, które zostało nawiązane za pomocą obiektu PDO, istnieje aż do momentu usunięcia tego 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 6.115

```
$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 `PDOSStatement`, który zawiera wynik wykonania zapytania.

## Przykład 6.116

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=ksiegarnia', '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 zostają zapisane do tablicy asocjacyjnej `$wiersz` i będą wyświetlane instrukcją `echo`. Po zakończeniu wyświetlania wyników połączenie zostanie zamknięte.

## Aktualizacja danych

Zapytania typu `INSERT` czy `UPDATE` służące do aktualizacji danych i inne, które nie zwracają wyników, są wysyłane za pomocą metody `exec()`. Po wykonaniu zapytania metoda ta zwraca liczbę określającą, ile wierszy zostało zmodyfikowanych.

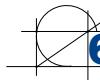
## Przykład 6.117

```
<?php
try {
    $pdo = new PDO('mysql:host=localhost;dbname=ksiegarnia', '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.

### Ćwiczenie 6.33

Zmodyfikuj skrypt z przykładu 6.117 tak, aby dane były wprowadzane do tabeli za pomocą formularza wyświetlanego na stronie internetowej.



## 6.12. Pytania i zadania

### 6.12.1. Pytania

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. Czym różni się przesyłanie danych z formularza na serwer za pomocą metody `POST` od przesyłania danych przy użyciu metody `GET`?
10. Jakie jest przeznaczenie plików `cookies`?
11. Jak działa dostępny w języku PHP mechanizm sesji?
12. Jakie parametry należy podać w kodzie PHP, aby nawiązać połączenie z bazą danych?
13. Jakie jest przeznaczenie biblioteki PDO dostępnej w języku PHP?

### 6.12.2. Zadania

#### Zadanie 1.

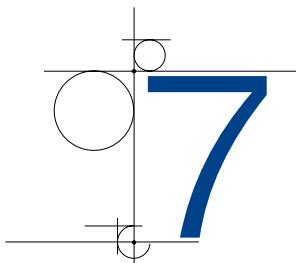
1. W MySQL utwórz bazę danych `Moja_szkola` lub wykorzystaj bazę danych `Moja_szkola` tworzoną w części II podręcznika. Baza danych powinna zawierać tabelę `Uczeń` z danymi uczniów oraz tabelę `Rejestracja` z dwoma polami: `login` i `haslo`. Zostanie ona wykorzystana do rejestracji loginów i haseł użytkowników 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.** Podczas rejestracji dane z formularza rejestracyjnego będą zapisywane w tabeli Uczeń, a login i hasło w tabeli Rejestracja.

**4.** Utwórz skrypt, który:

- dane z formularza rejestracyjnego zapisze w tabeli Uczeń,
- login i hasło zapisze w tabeli Rejestracja,
- po zalogowaniu przekieruje użytkownika do strony umożliwiającej mu zamieszczenie komentarza,
- wyświetli na stronie dotychczasowe komentarze innych użytkowników,
- wyświetli nowy komentarz poniżej wcześniejszych komentarzy,
- wyświetli na stronie przycisk wylogowania, po którego kliknięciu nastąpi wylogowanie ze strony.



# Walidacja kodu aplikacji

## 7.1. Wprowadzenie

Testowanie aplikacji ma na celu wykrycie błędów (walidacja oprogramowania) oraz sprawdzenie, czy oprogramowanie realizuje funkcje, do których wykonywania zostało przeznaczone (weryfikacja oprogramowania). Testowanie można przeprowadzić w dowolnym momencie podczas pracy nad oprogramowaniem, ale bardzo często jest to ostatni etap tworzenia aplikacji internetowej.

Testowanie nie spowoduje wykrycia wszystkich błędów oprogramowania, natomiast może dostarczyć informacji o zgodności oprogramowania z oczekiwaniami użytkowników oraz o występowaniu określonych błędów.

Celem testowania jest:

- ocena jakości aplikacji,
- wykrycie błędów oprogramowania i ich opisanie,
- określenie sposobu poprawienia oprogramowania.

Po przeprowadzeniu testów powinien zostać przygotowany raport podsumowujący, w którym będzie można znaleźć podsumowanie wykonanych prac, wskazanie potencjalnych zagrożeń, określenie poziomu bezpieczeństwa, opis proponowanych środków zaradczych.

Przeprowadzaniem testów całkowitych aplikacji zajmują się najczęściej wyspecjalizowane firmy posiadające odpowiednie narzędzia.



## 7.2. Testy aplikacji

### 7.2.1. Podział testów

Jest wiele sposobów podziału testów. Testy można podzielić na przykład ze względu na:

- sposób wykonywania,
- zakres,
- funkcjonalność.

#### Ze względu na sposób wykonywania

Ze względu na sposób wykonywania testy dzielimy na:

- automatyczne,
- manualne.

#### Testy automatyczne

Podczas testowania oprogramowania często mamy do czynienia z sytuacją, w której czynności związane z testowaniem wykonywane są automatycznie. Testy automatyczne często są 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 rano był dostępny raport.

Testy automatyczne najczęściej opierają się na specjalnie do tego celu przeznaczonym oprogramowaniu.

#### Testy manualne

Testy manualne wymagają indywidualnego podejścia do testowanej aplikacji. Najczęściej są to testy dotyczące zgodności aplikacji z wizją zamawiającego. Każdorazowo są przygotowywane pod konkretne wymagania. Są wykonywane przez testera, który na podstawie scenariuszy testowych weryfikuje oprogramowanie.

#### Ze względu na zakres

Ze względu na zakres testy dzielimy na:

- jednostkowe,
- systemowe,
- integracyjne.

## Testy jednostkowe

Testy jednostkowe (ang. *unit tests*) 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 jego działania. Do przeprowadzenia takiego testu może zostać wykorzystany program komputerowy, który wczytuje do aplikacji odpowiednie dane wejściowe, a następnie sprawdza, czy program generuje oczekiwane 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.

## Testy systemowe

Testy systemowe przeprowadza się w celu weryfikacji aplikacji pod kątem zgodności z wymaganiami funkcjonalnymi oraz wymaganiami dotyczącymi wydajności, użyteczności i niezawodności. Testy systemowe sprawdzają, czy zintegrowany system spełnia jako całość wymagania zawarte w specyfikacji.

Testy systemowe są pierwszym poziomem, na którym system jest testowany jako całość, i są przeprowadzane w środowisku zbliżonym do rzeczywistych warunków, w jakich będzie działał system.

## Testy integracyjne

Testy integracyjne służą do sprawdzania systemu na niższych poziomach. W takich testach weryfikowane są poszczególne komponenty systemu oraz interfejsy komunikacji między nimi. Testy integracyjne są przeprowadzane w celu określenia zgodności tych komponentów z wymaganiami funkcjonalnymi.

## Ze względu na funkcjonalność

Ze względu na funkcjonalność testy dzielimy na:

- funkcjonalne,
- niefunkcjonalne.

## Testy funkcjonalne

Testy funkcjonalne oceniają zgodność ze standardami, analizują przyczyny opóźnień w wyświetlaniu się różnych elementów strony, sprawdzają dostępność strony dla wyszukiwarek internetowych, weryfikują poprawność wyświetlanych komunikatów i komunikacji aplikacji z serwerem.

## Testy niefunkcjonalne

Testy niefunkcjonalne sprawdzają cechy jakościowe systemu. Do testów niefunkcjonalnych należą:

*Testy użyteczności* (ergonomia) — oceniają przejrzystość treści, strukturę aplikacji, a także nawigację i estetykę aplikacji.

*Testy kompatybilności* (architektura) — sprawdzają skalowalność aplikacji i bazy danych, oceniają architekturę serwera i jego konfigurację oraz architekturę aplikacji.

*Testy wydajnościowe* — szukają wąskich gardeł aplikacji webowej, analizują strukturę bazy danych i zapytań do bazy, oceniają kod źródłowy aplikacji. Wydajność jest jednym z podstawowych aspektów funkcjonowania aplikacji internetowej. Z aplikacji najczęściej korzysta wielu użytkowników — testy wydajnościowe mogą pokazać, czy aplikacja przy określonym obciążeniu będzie działała prawidłowo.

*Testy bezpieczeństwa* — sprawdzają, jakie zabezpieczenia przed atakami wykorzystuje aplikacja, w jakim stopniu działają elementy zabezpieczające systemu, w jakim stopniu próby ataków są wykrywane.

## Inne rodzaje testów

### Testy z udziałem użytkownika

Testy z udziałem użytkownika są przeprowadzane, gdy w aplikacjach występują obszary, które są istotne z punktu widzenia użytkownika. Określają one, jak użytkownik wchodzi w interakcję z aplikacją, aby uzyskać oczekiwany wynik.

### Testy akceptacyjne

Testy akceptacyjne są przeprowadzane w celu potwierdzenia, że wykonane oprogramowanie jest odpowiedniej jakości. Testy te sprawdzają, czy aplikacja jest zgodna z projektem i założeniami. Czy zawiera wszystkie przewidziane elementy i czy jej wygląd nie odbiega od oczekiwaneego.

## 7.3. Debugowanie aplikacji

Debugowanie aplikacji to proces wyszukiwania błędów w programie i ich usuwania.

Analizowanie kodu tworzonej aplikacji internetowej w celu znalezienia błędów powinno być podstawowym działaniem programisty. Coraz większa liczba frameworków używanych do tworzenia stron wymaga analizy nie tylko kodu strony, ale również zwracanego kodu wynikowego aplikacji internetowej.

Analizę kodu można przeprowadzić:

- na poziomie kodu źródłowego w sposób zautomatyzowany,
- w czasie wykonania programu, często również w sposób zautomatyzowany,
- korzystając z pomocy programisty, który zlokalizuje błędy oraz zaproponuje poprawki.

Analiza kodu powinna być wykonywana ręcznie przez programistę, a następnie należy przeprowadzić analizę automatyczną dotyczącą między innymi przepływu sterowania, zmian w drzewie DOM, wartości zmiennych. Trzeba także sprawdzić, jak zachowuje się przeglądarka w trakcie wykonywania określonych działań aplikacji.

## 7.3.1. Narzędzia do debugowania

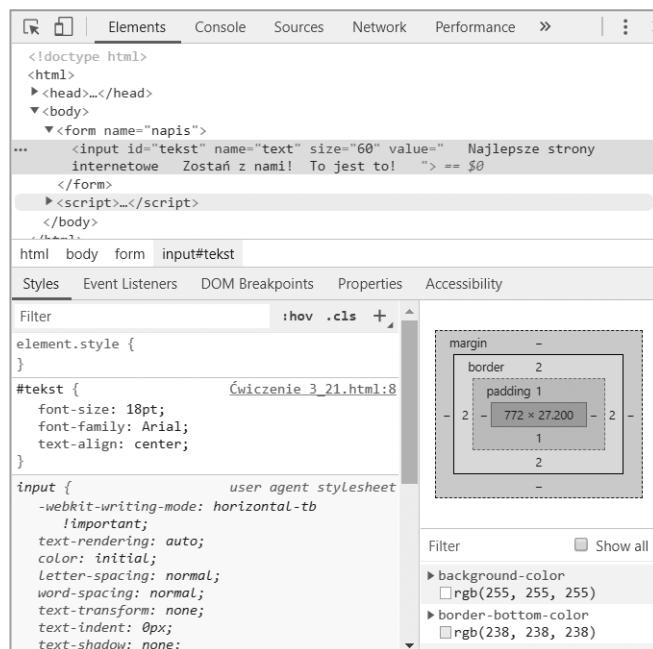
Podstawowe narzędzia potrzebne do edycji i debugowania kodu aplikacji znajdują się w przeglądarkach.

Na przykład dla przeglądarki Firefox dostępny jest dodatek Firebug oraz zestaw narzędzi Firefox Developer Tools, a przeglądarka Google Chrome ma zestaw narzędzi Chrome Developer Tools. Te i inne programy narzędziowe są do siebie bardzo podobne i udostępniają analogiczne funkcje.

### Chrome Developer Tools

Chrome Developer Tools to zestaw narzędzi dla programistów aplikacji internetowych. Jest wbudowany bezpośrednio w przeglądarkę Google Chrome. Można go uruchomić, klikając element strony prawym przyciskiem myszy i wybierając opcję *Zbadaj*. Inną metodą to wybranie klawiszy *Ctrl+Shift+I* lub klawisza *F12*.

Po uruchomieniu narzędzia zostanie otwarte okno Chrome Developer Tools. W górnej części okna znajduje się lista paneli, niżej dostępny jest obszar roboczy wybranego panelu (rysunek 7.1).



Rysunek 7.1. Okno Chrome Developer Tools. Panel Elements

## Panel Elements

Panel *Elements* służy do edycji wyświetlanej strony. Pokazuje drzewo DOM strony i pozwala je modyfikować. W panelu można zobaczyć, z jakich elementów składa się strona. Można przeprowadzić edycję tych elementów i stylów, które są do nich przypisane.

Element strony zostanie wybrany po jego kliknięciu w strukturze HTML. W dolnej części panelu zostaną pokazane style CSS przypisane do tego elementu. Z prawej strony zostanie wyświetlona informacja dotycząca wybranego węzła DOM. Są tu widoczne aktualne ustawienia elementów oraz lista właściwości przypisanych do elementu.

## Panel Sources

W panelu *Sources* wyświetlane są wszystkie zasoby, z których korzysta strona. Można w nim podglądać skrypty wykonywane na stronie, wykonać ich edycję i debugowanie. Można dodać skrypty zapisane jako plik *.js* lub dołączyć kody dostępne przez różne rozszerzenia Google Chrome.

Okno panelu zostało podzielone na trzy części. W pierwszej części u góry po lewej stronie można wybrać określony skrypt, w drugiej części z prawej strony widoczna jest zawartość tego skryptu. W trzeciej części u dołu można zobaczyć typowe funkcje debugera (rysunek 7.2). Dostępne opcje pozwalają na szukanie błędów zarówno w skrypcie, jak i w drzewie DOM.

```

1 /*!
2 * jQuery Validation Plugin - v1.10.0 -
3 * https://github.com/jzaefferer/jquery-validation
4 * Copyright (c) 2012 Jörn Zaefferer; Licei
5 */
6 (function($) {
7     $.extend($.fn, {
8         // http://docs.jquery.com/Plugins/Validation/validate
9         validate: function( options ) {
10             // if nothing is selected, return
11             if (!this.length) {
12                 if (options && options.debug)
13                     console.warn( "nothing selected" );
14             }
15             return;
16         }
17         // check if a validator for this
18         var validator = $.data(this[0], 'validator');
19         if ( validator ) {
20             return validator;
21         }
22     });
23     // Add new validate() fn if HTML5
24     // support is available
25     // ...
26 
```

Rysunek 7.2. Okno panelu Sources

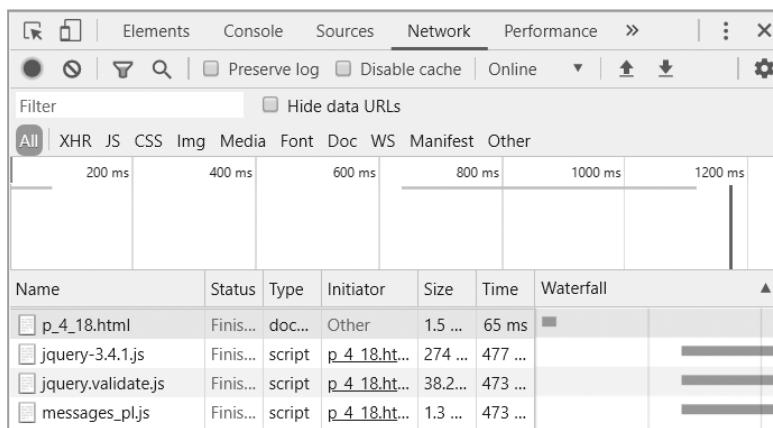
## Panel Console

Panel *Console* służy do pisania kodu i przeglądania zarejestrowanych komunikatów, które informują, czy program działa zgodnie z oczekiwaniemi.

## Panel Network

Panel *Network* pozwala analizować ruch sieciowy na stronie. Można podglądać komunikację, filtrować ją i analizować czas odpowiedzi na żądanie.

Panel pokazuje pliki pobierane przez stronę. Może służyć do sprawdzenia, czy wszystkie elementy strony zostały załadowane prawidłowo. Umożliwia monitorowanie statusu asynchronicznych odwołań (rysunek 7.3).



Rysunek 7.3. Okno panelu Network

## Panel Audits

Panel *Audits* pozwala na wykonanie statycznej i dynamicznej analizy kodu strony. Po przeprowadzeniu audytu narzędzie podpowiada, jakie elementy można zmienić, aby przyspieszyć działanie witryny.

## Panel Performance

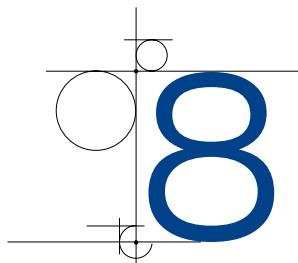
Panel *Performance* służy do symulowania pracy aplikacji w celu tworzenia optymalnego kodu i poprawienia jej wydajności.



## 7.4. Pytania i zadania

### 7.4.1. Pytania

1. Wymień rodzaje testów całościowych aplikacji internetowej.
2. Jak działają testy jednostkowe?
3. Jaki jest cel przeprowadzania testów wydajnościowych?
4. W jakim celu przeprowadza się testy funkcjonalne?
5. Na czym polega debugowanie aplikacji internetowej?
6. Jakie narzędzia służą do debugowania aplikacji internetowej?



# Dokumentowanie aplikacji

## 8.1. Komentarze

Komentarze to element każdego programu. Niezależnie od tego, w jakim języku programujemy, występują w nim komentarze. Ich głównym zadaniem jest opisywanie tworzonego kodu w celu ułatwienia późniejszego rozszyfrowania tego, co zostało napisane. Pisanie komentarzy jest szczególnie przydatne, gdy pracujemy w wieloosobowych zespołach i gdy planujemy automatyczne generowanie dokumentacji aplikacji.

Pisanie komentarzy powinno odbywać się w trakcie tworzenia kodu aplikacji. Spowoduje to, że będą one lepiej dopasowane do kodu i w każdej chwili będzie można wygenerować dokumentację.

Każdy język programowania ma swój własny sposób zapisywania komentarzy. W języku HTML komentarze są zapisywane pomiędzy znacznikami `<!-- -->`, w języku PHP, C++ i C# są zapisywane między znakami `/* */`.

Jeżeli planowane jest automatyczne generowanie dokumentacji tworzonej aplikacji, komentarze powinny być konstruowane w odpowiedni sposób. Powinny znaleźć się przy każdym dokumentowanym elemencie i zostać wyposażone w odpowiednie znaczniki. Takie komentarze należy zapisać w formacie DocBlocks. Jest to format najczęściej wykorzystywany przez generatory dokumentacji.

### 8.1.1. Komentarz w formacie DocBlocks

Komentarz tworzony w formacie DocBlocks jest zawsze ujęty w znaczniki `/** */`. Każdy wiersz między instrukcją otwierającą i zamkającą powinien zaczynać się gwiazdką `*`.

Aby wygenerować gotową dokumentację, dla każdego dokumentowanego elementu aplikacji należy stworzyć odpowiedni komentarz ujęty w znaczniki:

- Komentarze typu DocBlocks muszą się zaczynać od znaku `/**` i kończyć się znakiem `*/`.
- Każdy wiersz wewnętrz znaczników powinien zaczynać się gwiazdką `*`.
- Każdy komentarz poprzedza dokładnie jeden element konstrukcyjny, a cała zawartość komentarza ma zastosowanie do tego powiązanego elementu.

Komentarze powinny zawierać informację o tym, co robi dany fragment kodu, ale nie muszą opisywać dokładnie, co się dzieje w tym bloku programu. Powinny natomiast określać, co można za pomocą danego kodu osiągnąć.

Komentarz może na przykład opisywać argumenty metody oraz elementy zwracane. Dzięki temu użytkownik aplikacji łatwiej zrozumie przeznaczenie metody i wymagane dane.

Komentarze DocBlocks są podzielone na trzy części. Każda z nich jest opcjonalna, z tym że opis nie może istnieć bez podsumowania.

### *Podsumowanie*

Podsumowanie, czyli krótki opis, jest wprowadzeniem do funkcjonalności powiązanego elementu. Podsumowanie kończy się, gdy:

- na końcu linii wystąpi kropka
- lub
- wystąpiły dwa kolejne podziały linii.

### *Opis*

Opis dostarcza więcej informacji dotyczących funkcjonalności powiązanego elementu. Mogą to być informacje na temat algorytmu funkcji, przykładu użycia funkcji lub tego, w jaki sposób klasa pasuje do całej architektury aplikacji. Opis kończy się po napotkaniu pierwszego znacznika lub po zamknięciu DocBlocks.

### *Tagi i adnotacje*

Tagi i adnotacje dostarczają związkowych informacji (metadanych) o powiązanym elemencie. Mogą na przykład opisywać rodzaj danych zwracanych przez metodę lub funkcję. Każdy tag musi być poprzedzony znakiem `@` i musi zaczynać się od nowej linii. Oprócz nazwy tag może zawierać argumenty. Większość tagów jest powiązana z określonym typem elementu. Dlatego niektóre tagi dotyczą tylko klas, inne tylko metod. Najłatwiejszym sposobem weryfikacji, czy tag dotyczy określonego elementu, jest sprawdzenie dokumentacji jego opisu.

## Przykład 8.1

Ogólna postać komentarza w formacie DocBlocks:

```
/*
 * Podsumowanie - informuje użytkownika o tym, co robi powiązany element.
 *
 * *
 * * Opis * - może rozciągać się na wiele wierszy, może dotyczyć
 * szczegółów powiązanego elementu i dostarczać pewnych dodatkowych
 * informacji.
 *
 * @param string argument - tagi i adnotacje mogą również
 * obejmować wiele wierszy.
 *
 * @return void
 */
function mojaFunkcja(argument) {

}
```

Tagi są zapisane za pomocą polecenia:

```
* @znacznik parametr1, parametr2, ...
```

## Przykład 8.2

```
/*
 * @source
 * @param string argument To jest opis.
 */
```

## Przykład 8.3

Najprostszy komentarz może zawierać wyłącznie opis słowny całego elementu:

```
/** Ta metoda wysyła dane na ekran. */
public void Pisz() {
    Console.WriteLine("Nazwisko: " + nazwisko);
}
```

## Przykład 8.4

Komentarz może opisywać argumenty metody oraz elementy zwracane:

```
/*
 * Metoda sprawdza, czy ktoś wygrał grę
 *
 * Zwraca true, jeśli gra się zakończyła.
 * @param wynik zawiera stan gry.
 * @return boolean zwraca stan gry.
 */
public void KtoryGracz(int wynik) {
    if (wynik == 3 || wynik == 30)
    {
        koniec = true;
    }
}
```

W podanym przykładzie zostały użyte dwa znaczniki — `@param` i `@return`. W znaczniku `@param` została podana dodatkowo nazwa argumentu, którego znacznik dotyczy. Znacznik `@return` zawiera informację o typie zwróconej wartości.



## 8.2. Tworzenie dokumentacji programu

### 8.2.1. Wprowadzenie

Tworzenie dokumentacji aplikacji jest bardzo ważną częścią procesu programowania. Dobrze napisana dokumentacja oprogramowania powinna zawierać również informacje o sposobie korzystania z aplikacji. Może też służyć jako źródło informacji o przebiegu procesu tworzenia oprogramowania.

### 8.2.2. Tworzenie dokumentacji

Prawidłowo opracowana dokumentacja powinna obejmować następujące dokumenty:

- dokumentację wymagań dla systemu,
- dokumentację projektu oprogramowania,
- dokumentację techniczną,
- dokumentację użytkownika.

Uzupełnieniem całościowej dokumentacji może być dokumentacja przeznaczona dla administratorów systemów.

## Dokumentacja wymagań dla systemu

Dokumentacja wymagań dla systemu jest istotnym narzędziem dla projektantów oprogramowania, programistów i testerów. Powinna zawierać wszystkie funkcjonalne i niefunkcjonalne wymagania dla tworzonego oprogramowania. Dokumentacja ta jest podstawą do opracowania oprogramowania oraz do testowania i weryfikacji tworzonej aplikacji. Może zawierać dokumenty dotyczące projektu oprogramowania, raporty z przeprowadzonych rozmów z użytkownikami aplikacji, kwestionariusze i wyniki badań.

Dokumentacja wymagań często jest przygotowywana przez klienta zlecającego tworzenie aplikacji.

## Dokumentacja projektu oprogramowania

Dokumentacja projektu oprogramowania powinna zawierać wszystkie informacje potrzebne do zbudowania oprogramowania. W jej skład powinny wchodzić:

- opis architektury oprogramowania,
- szczegółowy projekt oprogramowania,
- diagramy przepływu danych,
- projekt bazy danych.

Zgromadzone dokumenty są informacją dla programistów, jak korzystając z wytworzego oprogramowania, można je rozwijać i modyfikować. Powstałe dokumenty nie zawierają szczegółów dotyczących kodowania programu, ale powinny zawierać informacje niezbędne do tworzenia oprogramowania.

Dokumentacja projektu najczęściej jest przygotowywana przez analityków biznesowych.

## Dokumentacja techniczna

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.

Dokumentacja techniczna powinna umożliwić ponowne użycia raz napisanego kodu i ułatwić śledzenie i debugowanie tego kodu.

Dokumentacja techniczna powinna zawierać opis:

- wykorzystanych technologii (na przykład XHTML, PHP, CSS),
- wykorzystanych bibliotek (na przykład jQuery),
- zastosowanych wzorców projektowych, architektury aplikacji (na przykład MVC),
- zdefiniowanych użytkowników systemu (na przykład *admin*, *user*),
- zdefiniowanych funkcjonalności dla każdego użytkownika.

Dokumentacja techniczna może zawierać:

- kod źródłowy aplikacji,
- graficzną reprezentację algorytmów.

## Dokumentacja użytkownika

Dokumentacja użytkownika to opis aplikacji przeznaczony dla jej użytkownika. Ma ona wyjaśnić, jak powinien działać program i jak z niego korzystać. Na tę dokumentację składają się na przykład pliki pomocy, ogólne informacje o aplikacji i sposobie jej obsługi.

Dokumentacja użytkownika jest tworzona po zakończeniu procesu budowania aplikacji.

Podstawowe elementy dokumentacji użytkownika to:

- *Opis funkcjonalny*. Ta część dokumentacji powinna określać przeznaczenie i główne możliwości aplikacji. Opis funkcjonalny powinien zawierać niezbędne informacje pozwalające ocenić, czy aplikacja zaspokaja konkretne 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 obsługuwać najczęściej wykorzystywane funkcje aplikacji,
  - » jak obsługuwać błędne operacje wykonane przez użytkownika,
  - » jak korzystać z systemu pomocy.

Podręcznik użytkownika powinien zawierać prosty przykład korzystania z aplikacji.

## Dokumentacja administratorów systemów

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.

## 8.3. Automatyczne generowanie dokumentacji użytkownika

Opracowywanie dokumentacji technicznej jest procesem czasochłonnym, dlatego wiele środowisk programistycznych wspomaga tworzenie dokumentacji i umożliwia automatyczne jej generowanie.

### 8.3.1. Narzędzia automatycznego generowania dokumentacji

#### JavaDoc

JavaDoc to narzędzie automatycznego generowania dokumentacji na podstawie zamieszczonych w kodzie źródłowym, napisanym w języku Java, specjalnie sformatowanych komentarzy. Program automatycznie dołącza informacje o nazwach komentowanych klas, metod i zmiennych. Komentarze umieszczone w kodzie programu muszą być zapisane w formacie DocBlocks. Otrzymane dokumenty są zapisane w formacie HTML.

#### Doxygen

Doxygen to narzędzie do dokumentowania oprogramowania pisane w językach C++, C, Java, Python, PHP, C#. Dokumentacja jest generowana na podstawie analizy kodu i komentarzy. Program wykorzystuje komentarze DocBlocks. Otrzymana dokumentacja może zostać zapisana w formacie RTF, PostScript, PDF i HTML.

Dokumentacja może być generowana zarówno bezpośrednio ze źródła, jak i z plików połączonych z nim.

#### phpDocumentor

phpDocumentor to narzędzie służące do tworzenia dokumentacji aplikacji napisanych w języku PHP. Dokumentacja jest generowana na podstawie specjalnie sformatowanych komentarzy (format DocBlocks) wewnętrz kodu źródłowego.

phpDocumentor wspiera zarówno kod napisany obiektowo, jak i strukturalnie. Tworzy dokumentację w formacie HTML i PDF.

### 8.3.2. ESDoc

ESDoc to narzędzie do generowania dokumentacji aplikacji napisanych w języku JavaScript. Jeżeli do programu nie zostały dodane komentarze, to dokumentacja zostanie wygenerowana tylko na podstawie kodu. Jeśli w kodzie zostaną umieszczone komentarze, które opiszą poszczególne jego fragmenty, to dokumentacja będzie wygenerowana na podstawie kodu i komentarzy. Program wykorzystuje komentarze DocBlocks.

Poniższy przykład pokazuje sposób dołączania komentarzy do opisu funkcji.

### Przykład 8.5

Do definicji funkcji zapisanej w języku JavaScript został dodany komentarz opisujący, czego dotyczy funkcja i jakie przyjmuje argumenty.

```
/**  
 * To jest klasa Licz.  
 */  
  
export default class Licz {  
    /**  
     * @param {number} a - liczba a.  
     * @param {number} b - liczba b.  
     * @return {number} wynik sumowania a i b.  
     */  
    sum(a, b) {  
        return a + b;  
    }  
}
```

## Instalacja ESDoc

Do zainstalowania aplikacji ESDoc wymagane jest zainstalowanie środowiska NPM.

Jeżeli środowisko NPM jest dostępne, w wierszu poleceń należy wpisać:

```
npm install -g esdoc
```

co spowoduje zainstalowanie ESDoc.

Następne polecenie:

```
cd projekt/
```

spowoduje przejście do katalogu z projektem.

Kolejny krok to utworzenie pliku konfiguracyjnego `esdoc.json`:

```
echo {"source": "./src", "destination": "./doc"} > .esdoc.json
```

Do wygenerowania dokumentacji za pomocą ESDoc należy wpisać polecenie:

```
esdoc
```



## 8.4. Pytania i zadania

### 8.4.1. Pytania

1. Jakie elementy powinien zawierać komentarz w formacie DocBlocks?
2. Jak wygląda zapis znacznika w komentarzu DocBlocks?
3. Jakie dokumenty powinny wchodzić w skład dokumentacji aplikacji?
4. Co powinna zawierać dokumentacja techniczna aplikacji?
5. Jakie elementy powinny znaleźć się w dokumentacji użytkownika aplikacji?
6. Wymień poznane narzędzia do automatycznego generowania dokumentacji.
7. Jakiego rodzaju dokumentację można tworzyć za pomocą narzędzi do automatycznego generowania dokumentacji?

# Bibliografia

- [1] Keith Wald, Jason Lengstorf, *PHP i jQuery. Techniki zaawansowane*, Helion, Gliwice 2017.
- [2] Jolanta Pokorska, *Kwalifikacja E.14. Tworzenie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk*, Helion, Gliwice 2014.
- [3] Jolanta Pokorska, *Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 4. Tworzenie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk*, Helion, Gliwice 2019.

## Źródła internetowe

- [1] <https://www.w3schools.com/js/>
- [2] <https://www.w3schools.com/jsref/default.asp>
- [3] <https://angular.io/start>
- [4] <https://www.javatpoint.com/angular-8>
- [5] <https://angular.io/cli>
- [6] <https://www.w3schools.com/nodejs/>
- [7] <https://www.w3schools.com/react/>
- [8] <https://www.w3schools.com/jquery/>
- [9] <https://www.php.net/manual/en/index.php>

# Skorowidz

.NET core, 22

## A

abstrakcja, 12

adnotacja, 304

adres URL, 93

aktualizacja danych bazy, 278, 288, 292

algorytm, 15

sortowania bąbelkowego, 18

wyszukiwania elementu, 18

Angular, 21, 158

struktura projektu, 164

środowisko pracy, 159

Angular Extension Pack, 160

animacje, 140

zaawansowane, 142

animowanie

grafiki, 106

tekstu, 103

Apache, 28

aplikacje

debugowanie, 298

dokumentowanie, 303, 306

internetowe, 25

testowanie, 295

argumenty funkcji, 61, 210

domyślne, 211

przekazywane przez wartość, 210

przekazywane przez referencję, 211

automatyczne generowanie dokumentacji,

309

## B

baner, 104, 105

bazy danych, 267

aktualizowanie danych, 278, 288, 292

kończenie połączenia, 269, 284, 291

nawiązanie połączenia, 268, 284, 290

pobieranie danych, 291

tworzenie, 283, 288

zapytania, 270, 285

biblioteka, 14

Angular, 158

jQuery, 123

PDO, 290

React.js, 165

## C

cechy

frameworka, 21

obiektowości, 12

Chrome Developer Tools, 299

panel

Audits, 301

Console, 301

Elements, 300

Network, 301

Performance, 301

Sources, 300

ciagi znakowe

analizowanie, 223

formatowanie, 219

indeksowanie, 223

porównywanie, 226

usuwanie, 222

zmiana wielkości liter, 222

znajdowanie podciągów, 224

cookies, 177, 251

tworzenie plików, 251

usuwanie plików, 252

zastosowania, 253

CSS

metody, 133

## D

data i czas, 79, 214, 216

debugowanie aplikacji, 298

narzędzia, 299

definiowanie funkcji, 205

deklaracja typu, 185

dekrementacja, 191

Django, 22

DocBlocks, 303  
 dodatek Angular Extension Pack, 160  
 dokument  
 zdarzenia, 103  
 dokumentacja  
 administratorów systemów, 308  
 projektu oprogramowania, 307  
 techniczna, 307  
 użytkownika, 308  
 wymagań dla systemu, 307  
 dokumentowanie aplikacji, 303  
 automatyczne generowanie, 309  
 DOM, Document Object Model, 35, 83  
 hierarchia obiektów, 84  
 Doxygen, 309  
 DSN, Data Source Name, 290  
 dziedziczenie, 13

**E**

Eclipse, 20  
 ECMAScript, 168  
 edytor programistyczny  
 Visual Studio Code, 157  
 ESDoc, 309

**F**

filtry selektorów jQuery, 129  
 format DocBlocks, 303  
 formatowanie tekstu, 219, 220  
 formularze  
 HTML, 240  
 przekazywanie danych, 242  
 walidacja, 109–113, 152  
 wprowadzanie danych, 276  
 zdarzenia, 102, 136  
 zerowanie pól, 119  
 framework, 21  
 .NET core, 22  
 Angular, 21  
 Django, 22  
 React, 22  
 funkcja, 59, 205  
 alert(), 66  
 close(), 284  
 date(), 216  
 empty(), 245  
 fclose(), 230  
 feof(), 232

fetch\_assoc(), 285  
 fgetc(), 233  
 fgets(), 231  
 file\_exists(), 229  
 file\_get\_contents(), 234  
 filesize(), 229  
 fopen(), 230  
 fread(), 233  
 fwrite(), 231  
 getdate(), 215  
 is\_file(), 229  
 isFinite(), 66  
 isNaN(), 65  
 isset(), 244, 258  
 mkdir(), 234  
 mktime(), 218  
 mysqli\_close(), 269  
 mysqli\_connect(), 268  
 mysqli\_connect\_errno(), 269  
 mysqli\_fetch\_array(), 271  
 mysqli\_fetch\_row(), 270  
 mysqli\_num\_rows(), 271  
 mysqli\_query(), 270, 273, 274  
 nl2br(), 219  
 num\_rows(), 285  
 parseFloat(), 65  
 parseInt(), 64  
 query(), 286  
 readdir(), 235  
 readfile(), 234  
 rmdir(), 234  
 session\_destroy(), 258  
 session\_start(), 261, 263  
 strcmp(), 226  
 strpos(), 225  
 strtok(), 226  
 substr(), 238  
 time(), 214  
 touch(), 229  
 unlink(), 230  
 wordwrap(), 220  
 funkcje, 59, 205  
 analizowania ciągów znaków, 223  
 argumenty, 61, 210  
 daty i czasu, 214  
 definiowanie, 60, 205  
 domyślne argumenty, 211  
 formatowania ciągów, 219  
 obsługą plików, 227

sortowania, 213  
 tablic, 213  
 usuwania ciągu znaków, 222  
 wbudowane, 64, 213  
 wyjścia, 239  
 wyszukiwania, 214  
 wywoływanie, 60  
 zmiany wielkości liter, 222  
 zwracanie wartości, 206

**G**

grafika, 106

**H**

hermetyzacja, 12  
 hierarchia obiektów DOM, 84  
 HTML, 36  
 obsługa zdarzeń, 99

**I**

IDE, Integrated Development Environment, 19  
 Eclipse, 20  
 NetBeans, 20  
 Visual Studio, 20  
 identyfikatory, 42  
 indeksowanie ciągu znaków, 223  
 inkrementacja, 191  
 instrukcja  
     break, 57, 199  
     continue, 58, 199  
     CREATE DATABASE, 288  
     die(), 239  
     document.write, 38  
     exit(), 239  
     global, 207  
     INSERT INTO, 275, 278, 287  
     SELECT, 273, 285  
     SQL CREATE DATABASE, 283  
     SQL CREATE TABLE, 283, 289  
     UPDATE, 278, 282, 288  
 instrukcje, 10, 39  
     sterujące, 49, 192  
     warunkowe, 10, 49, 192  
         if ... else, 49  
     wyboru, 51  
         switch, 51, 194

**J**

JavaDoc, 309  
 JavaScript, 13, 35  
     Angular, 158  
     animowanie  
         grafiki, 106  
         tekstu, 103  
     funkcje, 59  
     funkcje wbudowane, 64  
     identyfikatory, 42  
     instrukcje, 39  
     instrukcje sterujące, 49  
     jQuery, 123  
     komentarze, 39  
     literały, 42  
     obiekty, 67  
         DOM, 83  
         wbudowane, 74  
     obsługa zdarzeń, 98  
     operatory, 44  
     pętle, 53  
     React.js, 165  
     składnia języka, 38  
     słowa kluczowe, 39  
     typy danych  
         skalarne, 43  
         złożone, 44  
     w HTML, 36  
     walidacja danych, 109  
     wyrażenia regularne, 114  
     zmienne, 40  
 język  
     JavaScript, 13, 35  
     PHP, 14, 171  
     Python, 14  
     TypeScript, 158  
 języki programowania  
     semantyka, 7  
     składnia, 7  
     skryptowe, 13, 35  
     typy danych, 7  
 jQuery, 123  
     animacje, 140  
     animacje zaawansowane, 142  
     dołączenie, 124  
     filtry selektorów, 129  
     pobieranie, 123  
     pokaz zdjęć, 149  
     położenie elementu, 144

**j**Query

- selekatory, 126
- ukrywanie treści, 136
- Validation, 154
- walidacja formularzy, 152
- wyświetlanie podpowiedzi, 146
- zdarzenia biblioteki, 133
  - formularze, 136
  - mysz, 134
- zmiana rozmiaru elementu, 144

JSX, 169

**K**

## katalogi

- operacje, 234

klasa, 12

klauzula WHERE, 273, 286

klawiatura

- zdarzenia, 102

komentarze, 39, 176, 303

- format DocBlocks, 303

konstruktor, 71

kontroler, 27

**L**

literał, 42

**T**

łańcuch tekstowy, 74

**M**

## metoda

- animate(), 142, 145
- append(), 151
- attr(), 151
- css(), 129
- document.write(), 91
- fadeIn(), 140
- fadeOut(), 140
- GET, 177, 243
- getElementById(), 88–91
- getElementByTagName(), 88
- hide(), 140
- history.go(), 92
- location.reload(), 94
- location.replace(), 94
- POST, 177, 242

show(), 140

slideDown(), 141

slideToggle(), 140

slideUp(), 141

toggle(), 128

window.clearInterval(), 86

window.setInterval(), 85

window.setTimeout(), 85

metody CSS, 133

model, 27

Single Page Application, 159

MVC, Model-View-Controller, 26

MySQL, 267

MySQLi

proceduralny, 268

zorientowany obiektowo, 284

mysz

zdarzenia, 100, 134

**N**

narzędzia do debugowania, 299

narzędzie

Chrome Developer Tools, 299

Doxygen, 309

ESDoc, 309

JavaDoc, 309

phpDocumentor, 309

nawias klamrowy, 63

NetBeans, 20

Node.js, 158

NPM, 158

**O**

obiekt, 11, 44, 67

anchor, 94

Array, 75

Date, 79

document, 87

form, 94

history, 92

link, 94

location, 93

navigator, 86

String, 74

window, 84

obiektowość, 12

obiekty

DOM, 83

- 
- klasy PDO, 290
  - przeglądarki, 84
  - tworzenie, 67, 71
  - wbudowane, 74
  - obsługa
    - formularzy, 240
    - plików, 227
    - zdarzenia jako właściwości obiektu, 99
    - zdarzeń, event handler, 98
    - zdarzeń w kodzie HTML, 99
  - operacje
    - na katalogach, 234
    - na plikach, 229
    - wejścia-wyjścia, 10
  - operator warunkowy, 195
  - operatory, 10, 44, 188
    - arytmetyczne, 45, 188
    - bitowe, 47, 189
    - dekrementacji, 191
    - inkrementacji, 191
    - logiczne, 48, 190
    - łańcuchowe, 191
    - porównania, 46, 188
    - przypisania, 48, 190
  - opis, 304
- P**
- pakiet XAMPP, 28
  - PDO, PHP Data Objects, 268, 290
  - pętla, 11, 53, 196
    - do ... while, 57, 197
    - for, 53, 196
    - foreach, 198
    - while, 55, 197
  - PHP, Hypertext Preprocessor, 14, 171
    - bazy danych, 267
    - blok instrukcji, 173, 200
    - funkcje, 205
      - analizowania ciągów znaków, 223
      - daty i czasu, 214
      - formatowania ciągów, 219
      - obsługi plików, 227
      - tablic, 213
    - wbudowane, 213
    - wyjścia, 239
  - instrukcje
    - sterujące, 192, 194
    - warunkowa, 192
- komentarze, 176
  - MySQLi
    - proceduralny, 268
    - zorientowany obiektowo, 284
  - obsługa formularzy, 240
  - operacje na katalogach, 234
  - operator warunkowy, 195
  - operatory, 188
  - pętle, 196
  - pliki cookies, 251
  - sesje, 257
  - stałe, 187
  - struktura języka, 173
  - typy danych, 178
  - w HTML, 172, 174, 200
  - wyrażenia, 188
  - zmienne, 176
  - phpDocumentor, 309
  - pliki
    - cookies, 251
      - tworzenie, 251
      - usuwanie, 252
      - zastosowania, 253
    - dodawanie, 227
    - funkcje obsługi, 227
    - odczyt danych, 231
    - operacje, 229
    - otwieranie, 230
    - tworzenie, 229
    - usuwanie, 229
    - zamykanie, 230
    - zapisywanie, 230
  - pokaz zdjęć, 149
  - polimorfizm, 13
  - połączenie z bazą danych, 290
  - programowanie, 8
    - obiektowe, 11
    - strukturalne, 9
  - przekazywanie danych, 242
  - Python, 14
- R**
- React.js, 22, 165
  - modyfikacja aplikacji, 167
  - środowisko, 166

## S

schemat blokowy, 15, 17  
sekwencja, 10  
selektory CSS, 126  
serwer Apache, 28  
serwery aplikacji internetowych, 28  
sesje, 257  
    zastosowania, 259  
    zmienne, 258  
skalarne typy danych, 43  
składnia  
    heredoc, 180  
    nowdoc, 181  
skrypty, 13, 35  
     kolejność wykonywania, 103  
słowo kluczowe, 39  
    let, 63  
    new, 71  
sortowanie, 213  
    bąbelkowe, 18  
SPA, Single Page Application, 158  
sprawdzenie długości ciągu, 223  
stałe, 187  
    predefiniowane, 187  
struktura projektu, 164  
SVG, Scalable Vector Graphics, 35

## Ś

środowisko  
    Node.js, 158  
    React.js, 166

## T

tabele  
    tworzenie, 283  
tablice, 44, 75, 182, 213  
asocjacyjne, 183  
łączenie elementów, 77  
odwracanie kolejności elementów, 77  
skojarzeniowe, 183  
sortowanie elementów, 78  
wielowymiarowe, 77  
tagi, 304  
tekst  
    animowanie, 103  
    pływający, 105  
    przesuwany, 104

testowanie aplikacji, 295  
testy  
    akceptacyjne, 298  
    automatyczne, 296  
    funkcjonalne, 297  
    integracyjne, 297  
    jednostkowe, 297  
    manualne, 296  
    niefunkcjonalne, 298  
    systemowe, 297  
z udziałem użytkownika, 298

tworzenie  
    bazы danych, 283, 288  
    dokumentacji programu, 306  
    obiektów, 67, 71  
    pliku cookie, 251  
    tabeli, 283, 289

typ danych, 178  
    array, 182  
    boolean, 43, 178  
    float, 179  
    integer, 178  
    null, 44, 185  
    number, 43  
    object, 185  
    resource, 185  
    string, 43, 179  
    undefined, 44

TypeScript, 158

typy danych  
    proste, 10  
    skalarne, 178  
    specjalne, 178  
    złożone, 44, 178

## U

URL, 93  
usuwanie  
    cięgu znaków, 222  
    pliku cookie, 252  
uwierzytelnienie użytkownika, 259

## V

Visual Studio, 20  
Visual Studio Code, 157  
    okno konsoli, 162  
    uruchamianie projektu, 163

**W**

- validacja, 295
  - formularza, 109, 111, 113, 152
- widok, 27
- właściwość prototype, 73
- wtyczka Validation, 154
- wyrażenia, 188
- wyrażenia regularne, 114
- wzorce projektowe, 26
  - do weryfikacji
    - adresu e-mail, 116
    - imienia i nazwiska, 116
    - kodu pocztowego, 115
- MVC, 26

**X**

- XAMPP, 28
  - instalacja w systemie
    - Linux, 29
    - Windows, 31

**Z**

- zapytania do bazy danych, 285
  - pobierające dane, 270, 285
  - wstawiające dane, 274, 287
- zasięg
  - blokowy, 63
  - zmiennych, 62, 206
- zastosowania sesji, 259

**zdarzenia, 98**

- biblioteki jQuery, 133
- dokumentu, 103
- elementów formularza, 102
  - formularza, 102, 136
  - klawiatury, 102
  - myszy, 100, 134
- Zend Framework, 22

**zintegrowane środowisko programistyczne, IDE, 19**

zmiana wielkości liter, 222

**zmienne, 9, 40, 176**

- globalne, 62, 206
- lokalne, 62, 208
- predefiniowane, 177

sesji, 258

statyczne, 208

zakres, 177

zmiana typu, 185

**znacznik HTML, 38**

<br>, 219

<form>, 242

<head>, 37

<p>, 219

<script>, 36, 124

**znaczniki czasu, 218****znajdowanie podciągów, 224****znaki**

apostrofu, 179

cudzysłowu, 180

# PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**  
<http://program-partnerski.helion.pl>



## KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



IT



BIZNES



PROJEKTY



PROCESY

NASZE SZKOLENIA SĄ PROWADZONE  
ZGODNIE Z METODĄ

## BLENDDED LEARNING

modelem kształcenia, który łączy tradycyjne szkolenie z dostępem do nowoczesnych narzędzi - videokursów, e-booków i audiobooków

T: 609 850 372 E: SZKOLENIA@HELION.PL

[WWW.HELIONSZKOLENIA.PL](http://WWW.HELIONSZKOLENIA.PL)