6

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 wywoływane z wiersza poleceń — skrypty są tworzone z wykorzystaniem parsera PHP i wywoływane 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

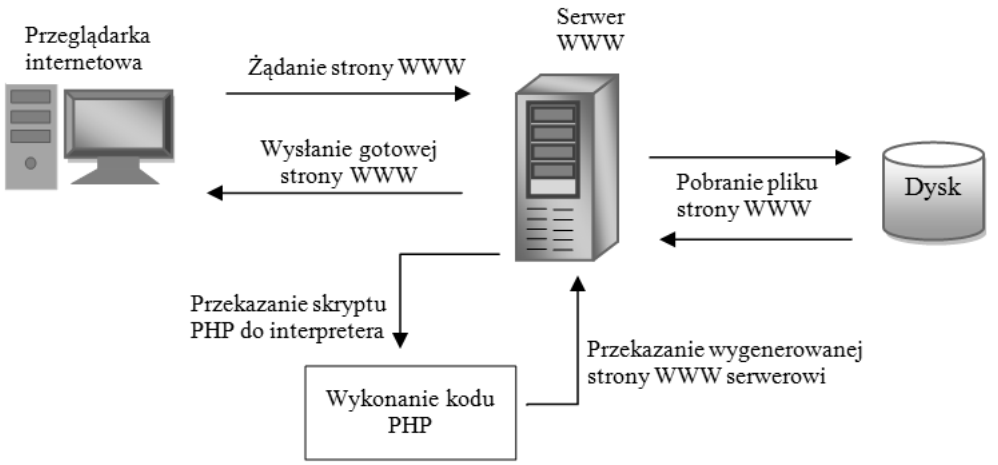
PHP jest skryptowym językiem programowania, za pomocą którego można tworzyć aplikacje WWW. Używając PHP, można:

- 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,
- szyfrować dane.

6.1.2. PHP w HTML

Skrypty pisane w języku PHP są umieszczane wewnątrz dokumentów HTML. Umownie dokumenty HTML zawierające osadzony kod PHP nazywa się skryptami.

Strony internetowe wygenerowane na podstawie skryptu PHP są zarządzane przez serwer WWW. Gdy w przeglądarce ma zostać wyświetlona taka strona, serwer WWW pobiera plik zawierający żądany skrypt i przekazuje go do interpretera PHP. Interpreter odnajduje w pliku skrypt PHP (jest on specjalnie oznaczony w kodzie HTML) i próbuje go wykonać. Wykonany skrypt jest zwracany do serwera WWW w postaci kodu HTML. Cała zawartość strony internetowej w kodzie HTML jest wysyłana do przeglądarki internetowej. Proces przetwarzania kodu PHP został pokazany na rysunku 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 wymiennie 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.

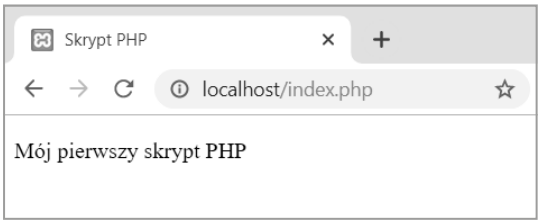
Przykład 6.5

```
<!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*.

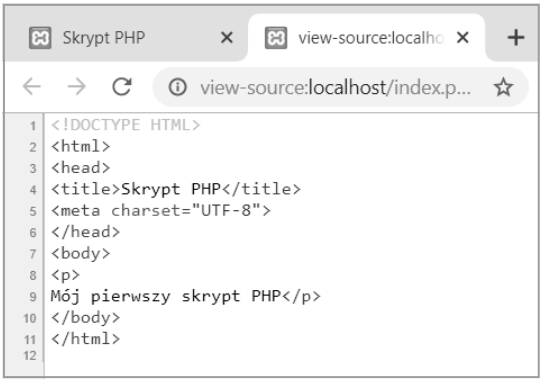
Jeżeli treść dokumentu została zapisana w innym katalogu, na przykład w katalogu *htdocs* został utworzony katalog *pliki_php*, to w przeglądarce należy wpisać adres *http://localhost/pliki_php/index.php*.

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).



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 `$`.

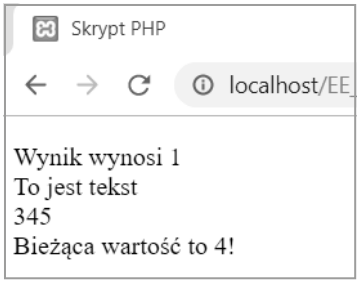
W języku PHP zmienna jest tworzona przy pierwszym jej użyciu. Nie trzeba wcześniej deklarować zmiennej ani określać jej typu.

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.



Rysunek 6.4. Rezultat wykonania kodu z przykładu

Występujący w przykładzie zapis:

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

jest równoważny zapisowi:

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

Zakres zmiennych

Zmienne mogą być deklarowane w dowolnym miejscu skryptu.

Zmienne deklarowane wewnątrz 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 rozpocząć 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";
$jezyk1 = "PHP";
$jezyk2 = "JavaScript";
$tekst = <<<TX
  Mam na imię $imie.
  Uczę się programować w języku $jezyk1.
  Umiem już programować w $jezyk2.
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,
  ...,
  kluczn => an
);
```

lub

```
$tab_a[klucz1] = a1;
$tab_a[klucz2] = a2;
  ...,
$tab_a[kluczn] = an;
```

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 oczekiwanego 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 trwały.

Rzutowanie typów odbywa się przez podanie nowego typu w nawiasie przed zmienną lub wartością, której typ należy zmienić. Dozwolone są określone typy rzutowań. Są to:

- `integer` — rzutowanie do typu całkowitego,
- `float` (`double`) — rzutowanie do typu rzeczywistego,
- `string` — rzutowanie do ciągu tekstowego,
- `array` — rzutowanie do tablicy,
- `object` — rzutowanie do obiektu.

Przykład 6.22

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

Zmianę typu w sposób trwały przeprowadza się za pomocą funkcji `settype()`. Funkcja ta 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 zmiennych nie używa się znaku dolara (`$`). Zgodnie z przyjętą konwencją nazwy stałych zawsze pisane są wielkimi literami. Zasięg stałych jest globalny i można odwoływać się do nich w każdym miejscu skryptu. Nie można tworzyć stałych poprzez zwykłe przypisanie. Do ich definiowania służy funkcja `define()`, która 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
==	Wynik `true`, gdy argumenty są równe.	`$a == $b`
!=	Wynik `true`, gdy argumenty są różne.	`$a != $b`
===	Wynik `true`, gdy argumenty są tego samego typu i są równe.	`$a === $b`
!==	Wynik `true`, gdy argumenty są różne lub są różnych typów.	`$a !== $b`
>	Wynik `true`, gdy argument pierwszy jest większy od drugiego.	`$a > $b`
<	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego.	`$a < $b`
>=	Wynik `true`, gdy argument pierwszy jest większy od drugiego lub mu równy.	`$a >= $b`
<=	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego lub mu równy.	`$a <= $b`
<>	Wynik `true`, gdy argumenty są różne.	`$a <> $b`

Operatory bitowe

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

Tabela 6.4. Operatory bitowe

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

Operatory logiczne

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

Tabela 6.5. Operatory logiczne

Operator	Działanie	Przykład
`and`	iloczyn logiczny	`$a and $b`
`&&`	iloczyn logiczny	`$a && $b`
`or`	suma logiczna	`$a or $b`
`||`	suma logiczna	`$a || $b`
`!`	negacja logiczna (NOT)	`!$a`
`xor`	różnica symetryczna	`$a xor $b`

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

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

Negacja logiczna zmienia wartość argumentu na przeciwną.

Wynik różnicy symetrycznej przyjmuje wartość `true` tylko wtedy, gdy obydwa argumenty mają różną wartość. Gdy obydwa argumenty mają równą wartość, wynikiem jest `false`.

Operatory przypisania

Operatory przypisania 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
`=`	`$x = 10`	`$x = 10`
`+=`	`$x += 5`	`$x = $x + 5`
`-=`	`$x -= 5`	`$x = $x - 5`
`*=`	`$x *= 5`	`$x = $x * 5`

Operator	Przykład	Znaczenie
/=	$x /= 5	$x = $x / 5
%=	$x %= 5	$x = $x % 5

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
.	łączenie łańcuchów znakowych	$x = "moje "."miasto";
.=	dołączanie do łańcucha znakowego	$x = "moje "; $x .= "miasto";

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 następnych 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 *instrukcje1* i wyjdź z bloku `switch` (polecenie `break`). Jeśli wynikiem jest *wartość2*, to wykonaj *instrukcje2* i wyjdź z bloku `switch`. Jeżeli wynikiem jest *wartość3*, to wykonaj *instrukcje3* i wyjdź z bloku `switch`. Jeśli wartość jest inna, wykonaj *instrukcje4* 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 wykonań 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 instrukcje, dopóki wyrażenie warunkowe jest prawdziwe. W pętli `do ... while` blok instrukcji jest wykonywany co najmniej raz, nawet jeżeli warunek zapisany w wyrażeniu warunkowym jest fałszywy, ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

Przykład 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 ($tab 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; ";
}
?>
```

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> Górniak</td><td> Mateusz</td><td> 638231484</td></tr>
</table>
<?php
}
?>
</body>
</html>
```

W podanym przykładzie przejście do HTML nastąpi tylko wtedy, gdy warunek `if` będzie spełniony (rysunek 6.6). Jeżeli zmienna `$tx` będzie miała wartość `false`, tabela nie zostanie wyświetlona na stronie.

Nazwisko	Imię	Telefon
Nowak	Adam	692399123
Kowalski	Jan	628345621
Górniak	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 × 10, która będzie wyświetlała wynik mnożenia, na przykład w podany na rysunku 6.7 sposób.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	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 pobierać je w postaci argumentów.

UWAGA

Taki sposób działania zmiennych globalnych jest charakterystyczny dla języka PHP. W większości języków programowania do zmiennych globalnych można odwoływać się z dowolnego miejsca.

Instrukcja global

Jeżeli istnieje konieczność odwołania się do zmiennej globalnej wewnątrz funkcji, można wykorzystać instrukcję `global`. Przed odwołaniem się do zmiennej należy umieścić konstrukcję:

```
global $nazwa zmiennej;
```

Za pomocą jednej instrukcji `global` można zadeklarować w funkcji wiele zmiennych globalnych.

```
global $zm1, $zm2, $zm3;
```

Przykład 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 zmienna `$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 zmienna lokalna jest tworzona w chwili wywołania funkcji i jest widoczna w obrębie tej funkcji. Gdy funkcja zakończy działanie, zmienna znika. Przy ponownym wywołaniu funkcji zmienna jest tworzona i przypisywana jest jej wartość początkowa.

Przykład 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).

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

Jeżeli przy tworzeniu zmiennej zostanie zadeklarowane, że jest to zmienna statyczna, jej wartość zostanie zachowana między kolejnymi wywołaniami funkcji. Deklaracja zmiennej statycznej ma postać:

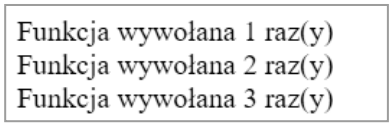
```
static $nazwa zmiennej = wartość;
```

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

ignorowana, a stosowana będzie wartość zapamiętana przy poprzednim wywołaniu funkcji (rysunek 6.9).



Rysunek 6.9. Efekt zadeklarowania w treści funkcji zmiennej statycznej

6.5.4. Argumenty funkcji

Argumenty mogą być przekazywane do funkcji na dwa sposoby:

- przez wartość,
- za pomocą referencji.

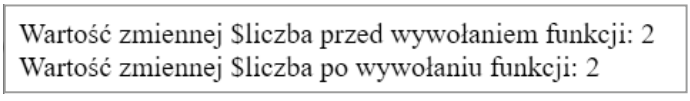
Argumenty przekazywane przez wartość

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 kopiach. 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 `$liczba` (wartość przekazanej zmiennej jest zwiększana o `3`). Przed jej wywołaniem zmienna `$liczba` została ustawiona na wartość `2` i wyświetlona w celu przetestowania jej stanu. Następnie została wywołana funkcja `wart()`. Po jej wykonaniu ponownie została wyświetlona zmienna `$liczba`, aby sprawdzić jej aktualny stan. Wszystkie operacje wewnątrz funkcji odbywały się na kopii zmiennej, natomiast wartość oryginalnej zmiennej nie uległa zmianie (rysunek 6.10).



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:

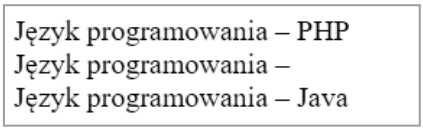
```
function nazwa ($argument1 = wartość, $argument2 = wartość,...) {
  instrukcje
}
```

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 wywołaniu 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).



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`).

Rysunek 6.13. Wynik wywołania funkcji z różnymi argumentami

Wartości domyślne można przypisać dowolnej liczbie argumentów. Ale jeżeli wartość domyślna zostanie przypisana określonemu argumentowi, to wszystkie następne również muszą mieć wartości domyślne.

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

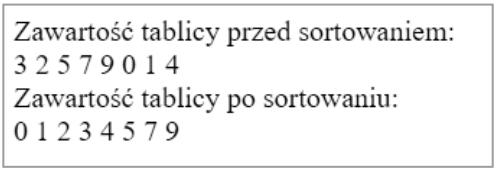
Sortowanie polega na ustawianiu elementów tablicy w określonym porządku, najczęściej rosnącym lub malejącym. Do sortowania tablic można wykorzystać kilka funkcji w zależności od tego, jakie sortowanie będzie realizowane oraz jakiego typu tablic będzie ono dotyczyło.

- `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 ";
}
?>
```

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 zwiększanie 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.

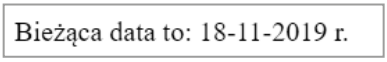
Tabela 6.8. Indeksy tablicy zwracanej przez funkcję getdate()

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 `$data` została przypisana tablica zwrócona przez funkcję `getdate()`. Wartości indeksów `mday`, `mon`, `year` zostały przypisane zmiennym `$dzien`, `$miesiac`, `$rok`. Gdy wartości zmiennych `$dzien` i `$miesiac` będą mniejsze od `10`, zostanie do nich dodany znak `0`. Po połączeniu wszystkich zmiennych w jeden ciąg jest on wyświetlony instrukcją `echo` (rysunek 6.15).



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()

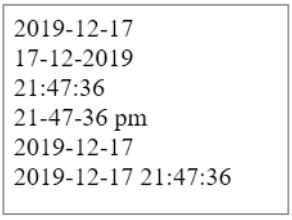
Znacznik	Znaczenie	Opis
`a`	Użycie określenia „przed południem” (am) lub „po południu” (pm)	am, pm
`A`	Użycie określenia „przed południem” (AM) lub „po południu” (PM)	AM, PM
`c`	Data i czas zgodne z formatem ISO 8601	2013-03-03T15:25:20

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.



Rysunek 6.16. Rezultat użycia funkcji date() z różnymi znacznikami formatującymi

Funkcja mktime()

Funkcja `mktime()` 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 `mktime()`, może zostać wykorzystany w funkcjach `getdate()` i `date()`.

Przykład 6.58

```
<?php
$czas = mktime(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 mktime() 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 uwzględni tych znaków. Można ominąć ten problem, dodając znaczniki `<br>` lub `<p>`. Ale nie zawsze takie rozwiązanie jest możliwe, szczególnie wtedy, gdy tekst jest wczytywany z pliku lub bazy danych. W takiej sytuacji można wykorzystać funkcję:

```
nl2br("ciąg_znaków")
```

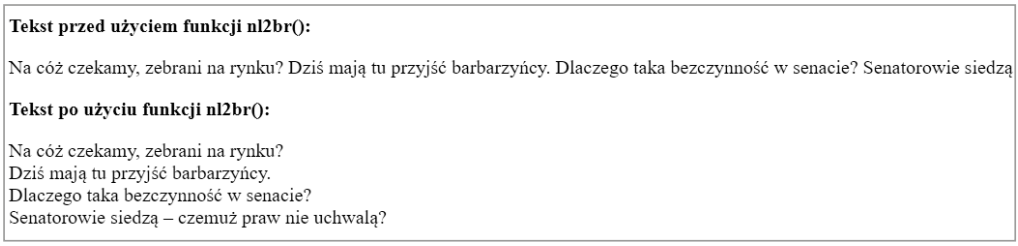
Funkcja ta dla wybranego bloku tekstu przed każdym znakiem końca linii automatycznie wstawi znacznik `<br>` i zwróci przetworzony tekst.

Przykład 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.



Rysunek 6.18. Wynik zastosowania do tekstu funkcji nl2br()

Funkcja wordwrap()

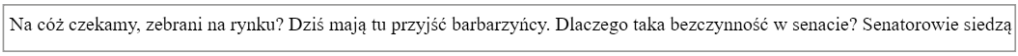
Do formatowania tekstu w postaci kolumny o określonej szerokości można wykorzystać funkcję `wordwrap()`. Dzieli ona ciąg podany jako argument na linie o maksymalnej długości 75 znaków. Do rozdzielenia linii domyślnie jest używany znak `\n`. Oprócz argumentu określającego ciąg źródłowy funkcja ma jeszcze 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).



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");
?>
```

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

Ostatni, czwarty argument funkcji będzie zastosowany, gdy w linii pojawi się słowo dłuższe niż zadeklarowana maksymalna długość linii. Domyślnie takie słowo nie zostanie podzielone między liniami. Użycie czwartego argumentu wymusi podział słowa między liniami. Aby się nim posłużyć, trzeba przypisać mu wartość logiczną `true`.

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-s
zkolne/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("ciąg_znakowy", "znaki do usunięcia")
```

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.'"';
$dl = mb_strlen($napis);
echo " ma długość $dl 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 znalezionego podciągu do jego końca. Funkcja `strstr()` rozróżnia wielkość liter. Jeżeli wielkość liter nie powinna mieć znaczenia, podczas przeszukiwania ciągu źródłowego należy użyć funkcji `stristr()`. Działa ona podobnie jak funkcja `strstr()`, ale nie rozróżnia wielkości liter.

Przykład 6.65

```
<?php
$dane = "Jan Kowalski, ul. Nowa 23, 80-874 Warszawa, tel. 693341678";
$tel = strstr($dane, "tel.");
echo $tel;
?>
```

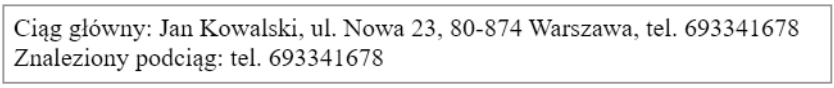
W podanym przykładzie zostanie wyświetlony tekst od znalezionego podciągu `tel.` do końca przeszukiwanego ciągu.

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.



Rysunek 6.22. Znaleziony podciąg w ciągu głównym

Funkcja strpos()

Podobne działanie do funkcji `strstr()` ma funkcja `strpos()`. Funkcja ta sprawdza, czy szukany podciąg znajduje się w ciągu źródłowym, a jeżeli tak, zwraca jego położenie. 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("ciąg1", "ciąg2")
```

- zwraca wartość mniejszą od `0`, jeżeli *`ciąg1`* jest mniejszy od *`ciąg2`*,
- zwraca wartość większą od `0`, jeżeli *`ciąg1`* jest większy od *`ciąg2`*,

- zwraca `0`, jeżeli ciągi są równe.

Funkcja rozróżnia wielkość znaków.

Funkcja strcasecmp()

Funkcja `strcasecmp()` działa jak funkcja `strcmp()`, ale nie uwzględnia wielkości liter. Ma postać:

```
strcasecmp("ciąg1", "ciąg2")
```

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` wewnątrz 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 zamykanie pliku

Za pomocą funkcji `fopen()` można otwierać istniejące pliki. Funkcja ma postać:

```
fopen('nazwa_pliku', 'tryb_otwarcia')
```

Pierwszy argument funkcji określa ścieżkę do pliku, który ma zostać otwarty. Drugi określa tryb, w jakim plik zostanie otwarty. Najczęściej stosowane tryby otwarcia to:

- `r` — plik zostanie otwarty w trybie tylko do odczytu,
- `w` — plik zostanie otwarty w trybie tylko do zapisu,
- `a` — plik zostanie otwarty w trybie dopisywania.

Przykład 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 ustawieniu 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 zamykania pliku służy funkcja `fclose()` zapisywana w postaci:

```
fclose(deskryptor)
```

Deskryptor to wartość zwrócona przez funkcję `fopen()`. Gdy zakończą się operacje wykonywane na otwartym pliku, powinien on zostać zamknięty. Jeżeli nie zostanie to zrobione w skrypcie, plik zostanie zamknięty, gdy skrypt zakończy działanie.

Przykład 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
$tekst = "Barbarzyńcy, gdy przyjdą, ustanowią prawa.\n";
if (!$p = fopen('dane.txt', 'a')) {
  echo "Nie można otworzyć pliku dane.txt";
} else {
  if (fwrite($p, $tekst) === false) {
    echo "Zapis do pliku nie powiódł się";
  } else {
    echo $tekst;
  }
  fclose($p);
}
?>
```

W podanym przykładzie tekst umieszczony w zmiennej `$tekst` będzie dopisywany na końcu pliku *dane.txt* przy każdym uruchomieniu skryptu. 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:

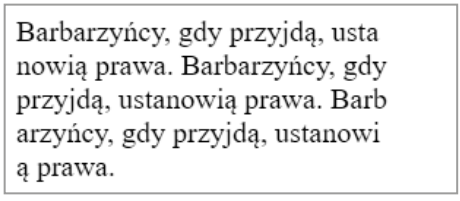
```
fread(deskryptor, ile_znaków)
```

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).



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 tekstowego lub wartość `false`, gdy wykonanie operacji się nie powiodło.

Funkcja `file()` służy do odczytywania zawartości plików tekstowych. Zapisywana jest w postaci:

```
file('nazwa_pliku')
```

Funkcja odczytuje całą zawartość pliku i zwraca tę zawartość w postaci tablicy. Każda komórka tablicy zawiera kolejny wiersz odczytanego tekstu. Argumentem jest nazwa odczytywanego pliku. Funkcja może 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.