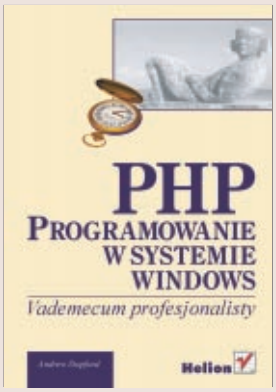
PHP. Programowanie w systemie Windows



Autor: Andrew Stopford
Tłumaczenie: Aleksandra Kula
ISBN: 83-7197-915-0
Tytuł oryginału: PHP Programming for Windows
Format: B5, stron: 290

Książka opisuje doskonałą platformę programistyczną dla MS Windows. Opisuje ona PHP na tyle szczegółowo, że osoby zainteresowane mogą potraktować ją jako podręcznik tego języka. Jej szczególną zaletą jest wyczerpujące omówienie integracji PHP z technologiami Windows. Dzięki książce dowiesz się, jak tworzyć i wykorzystywać obiekty COM w PHP, jak łączyć PHP z platformą .NET, jak korzystać z ActiveDirectory oraz jak używać MSXML.

„PHP. Programowanie w systemie Windows" zawiera również opis:

- Instalacji PHP w systemie Windows
- Konfigurowania i optymalizacji serwerów WWW
- Języka PHP i jego funkcji
- Łączenia PHP z bazami danych (Microsoft SQL Server, Oracle i MySQL)
- Użycia XML i XSLT w PHP
- Integracji PHP z ASP przy użyciu WDDX
- Tworzenia usług WWW w PHP

Jeśli planujesz używać PHP w systemach Windows, książka ta stanie się dla Ciebie nieocenioną pomocą. Gwarancję dobrej jakości stanowi również nazwisko autora. Andrew Stopford jest programistą z wieloletnim doświadczeniem, członkiem wielu zespołów pracujących nad projektami typu „open-source", między innymi XML-RPC for ASP i NMatrix.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 3.

Programowanie w PHP

Teraz, kiedy już zainstalowałeś PHP, możesz zacząć projektować aplikacje z jego wykorzystaniem. W tym rozdziale dowiesz się, od czego zacząć, projektując aplikacje dla WWW, oraz zapoznasz się z różnymi typami danych i strukturami składającymi się na język.

Narzędzia do tworzenia aplikacji w PHP

Zanim zaczniesz programować w PHP, potrzebny Ci będzie program do pisania, edytowania i zapisywania skryptów. Możesz korzystać z aplikacji Notatnik, jednak dostępne są również inne programy oferujące takie funkcje jak podświetlanie składni kodu oraz testowanie i uruchamianie. Omówimy to w kolejnych punktach.

PHP Edit

Edytor typu *open-source*, PHP Edit, oferuje podświetlanie kodu w różnych kolorach, testowanie go oraz dostęp do dokumentacji funkcji oraz obiektów i inne.

Więcej informacji, jak również samo narzędzie, znajdziesz na stronie *http://www.phpedit.com/*.

Macromedia HomeSite

Macromedia HomeSite to edytor HTML obsługujący technologię ASP, ColdFusion i PHP. Obsługa kolorowego podświetlania poszczególnych elementów kodu jest wbudowaną funkcją HomeSite, ale dostęp do innych funkcji, takich jak dokumentacja składni, jest możliwy jedynie przy wykorzystaniu narzędzi innych firm.

Więcej informacji i testową wersję możesz pobrać z witryny *http://www.macromedia.com/software/homesite/*.

Dodatki do programu Macromedia HomeSite przeznaczone do programowania w PHP można pobrać ze strony *http://www.wilk4.com/asp4hs/php4hs.htm*.

ActiveState Komodo

Narzędzie dostępne jest zarówno dla platformy Windows, jak i Linux. ActiveState Komodo to popularny edytor. Obsługa PHP jest wbudowaną funkcją narzędzia umożliwiającą podświetlanie w różnych kolorach składni oraz zdalne testowanie i uruchamianie kodu.

Więcej informacji i testową wersję ActiveState Komodo można pobrać z *http://www.activestate.com/Products/Komodo/*.

Zend IDE

Zend IDE to narzędzie zbudowane w oparciu o język Java, które zawiera obsługę podświetlania w różnych kolorach składni, jak również rozbudowane funkcje zdalnego testowania i uruchamiania. Więcej informacji i testową wersję narzędzia znajdziesz na stronie *http://www.zend.com/store/products/zend-ide.php*.

Podstawy składni PHP

Podobnie jak ASP, PHP wykorzystuje coś, co osobiście nazywam „otwierająco-zamykającym nawiasem" (z ang. *open-close bracket*). W klasycznym ASP program rozpoczyna się od znacznika początku programu (<%), a kończy się znacznikiem końca programu (%>):

```
<%
Response.write("Klasyczny program w ASP")
%>
```

Podobna sytuacja ma miejsce w przypadku PHP. Program rozpoczyna się od znacznika <?php, a kończy ?>:

```
<?php
print("Program w PHP");
?>
```

PHP dopuszcza również składnię dobrze znaną programistom tworzącym aplikacje uruchamiane po stronie klienta. Składnia ta rozpoczyna się poleceniem <SCRIPT LANGUAGE="php">, a kończy poleceniem </SCRIPT>:

```
<SCRIPT LANGUAGE="php">
print("test");
</SCRIPT>
```

Jeśli masz doświadczenie w projektowaniu w ASP, nie musisz się obawiać. PHP pozwala wykorzystywać składnię ASP:

```
<%
print("Program w PHP")
%>
```

Aby jednak taka składnia była poprawna, musisz zmienić wpis w pliku *php.ini*:

```
asp_tags = On
```

Ta zmienna ma domyślnie przypisaną wartość Off.

Programy wieloliniowe

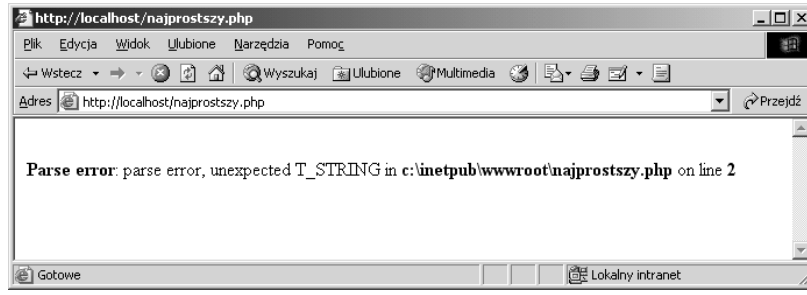
Wieloliniowe programy tworzone w PHP wymagają użycia dodatkowego znaku:

```
<?php
print("To jest twój pierwszy");
print("program w PHP");
?>
```

Zauważ, że PHP wymaga użycia operatora zamykającego linię — znaku średnika (;). Jeśli uruchomisz program, w którym nie ma znacznika końca linii, jak w poniżej przedstawionym przykładzie, pojawi się informacja o błędzie `Parse error (błąd przetwarzania)`:

```
<?php
print("To jest twój pierwszy")
print("program w PHP")
?>
```

Jeśli nigdy nie używałeś znaków końca linii, możesz czasami o nich zapominać. Jeśli pojawi się wiadomość `Parse error` podczas tworzenia aplikacji dla WWW, sprawdź, czy nie brakuje znaku końca linii w wierszu o numerze podanym przez komunikat o błędzie, jak pokazano na rysunku 3.1.



Rysunek 3.1. *Wiadomość o błędzie Parse error*

Zmienne

PHP jest określany jako język luźnych deklaracji. Oznacza to, że nie musisz deklarować typu zmiennych przed ich użyciem. PHP przypisuje w Twoim imieniu typy zmiennych. Aby to zrozumieć, przyjrzyj się następującemu przykładowi:

```
<?php
//nasze typy danych
$intdata = 1;
$doubledata = 5.00;
$stringdata = "Andrzej";
$booldata = TRUE;

//podaj typy danych
$IntType = gettype($intdata);
print("Zmienna $intdata jest typu $IntType<BR>");

$DoubleType = gettype($doubledata);
print("Zmienna $doubledata jest typu $DoubleType<BR>");
```

```
$StringType = gettype($stringdata);
print("Zmienna $stringdata jest typu danych $StringType<BR>");

$BoolType = gettype($booldata);
print("Zmienna $booldata jest typu danych $BoolType<BR>");
?>
```

Jeśli zapiszesz dane jako zmienną, PHP przypisuje typ zmiennej zgodnie z typem zapisanych w niej danych. Najpierw podaj, jakie dane będą przechowywane w zmiennej:

```
$intdatavar = 1;
$doubledatavar = 5.00;
$stringvar = "Andrzej";
$boolvar = TRUE;
```

PHP nadaje zmiennej typ odpowiadający danym zapisanym w zmiennej. Na przykład, ponieważ do zmiennej `$stringvar` przypisana jest wartość „Andrzej", PHP nada tej zmiennej typ `string`. Możesz to sprawdzić, wyświetlając typ zmiennej za pomocą funkcji `GetType`:

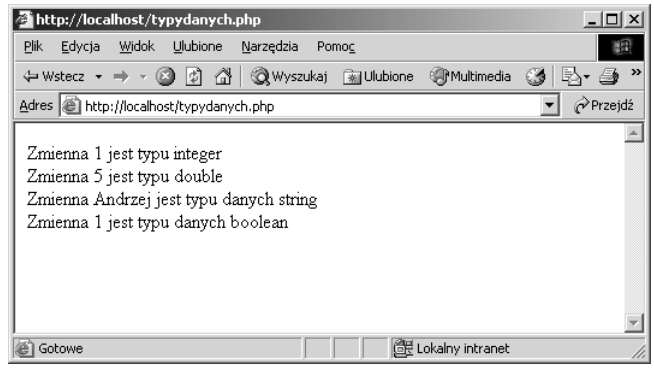
```
$IntType = GetType($intdatavar);
```

Teraz możesz wyświetlić wartość zmiennej i jej typ:

```
print("Zmienna $intdatavar jest typu $IntType<BR>");
```

Jeśli uruchomisz ten skrypt, zobaczysz takie wyniki, jakie pokazano na rysunku 3.2.

Rysunek 3.2.
Typy danych w PHP



Zwróć uwagę, że zmienna typu `boolean` jest wyświetlana jako `1`. PHP wyświetla zmienne typu `boolean` jako `1`, jeśli zmienna ma wartość `TRUE`, i jako `null` (wartość pusta), jeśli zmienna ma wartość `FALSE`. Przyjrzymy się temu dokładniej w dalszej części tego rozdziału.

Przypisywanie typów zmiennych

Chociaż luźne deklaracje mogą być przydatne w niektórych sytuacjach, czasem musisz podać typ danych wprost, na przykład przy przetwarzaniu danych pobranych z bazy danych. W tym celu musisz skorzystać z metody `SetType`:

```
<?php

//nasze typy danych
$intdata = 300;
```

```
$doubledata = 5.00;

$string = "False";

$bool = TRUE;

//wypisz typy danych
$IntType = gettype($intdata);
print("Zmienna $intdata jest typu $IntType<BR>");

$DoubleType = gettype($doubledata);
print("Zmienna $doubledata jest typu $DoubleType<BR>");

$StringType = gettype($string);
print("Zmienna $string jest typu $StringType<BR>");

$BoolType = gettype($bool);
print("Zmienna $bool jest typu $BoolType<BR>");

//zmień typy danych
SetType($doubledata, "integer");

SetType($intdata, "double");

SetType($bool, "string");

SetType($string, "boolean");

//wyświetl nowe typy danych
$IntType = gettype($intdata);
print("Zmienna $intdata jest teraz typu $IntType<BR>");

$DoubleType = gettype($doubledata);
print("Zmienna $doubledata jest teraz typu $DoubleType<BR>");

$StringType = gettype($string);
print("Zmienna $string jest teraz typu $StringType<BR>");

$BoolType = gettype($bool);
print("Zmienna $bool jest teraz typu $BoolType<BR>");

?>
```

Podobnie jak w pierwszym przykładzie, przypisaliśmy zmiennym wartości i wyświetliliśmy zawartość tych zmiennych oraz ich typ:

```
$intdata = 300;

$doubledata = 5.00;

$string = "False";

$bool = TRUE;

$IntType = gettype($intdata);
print("Zmienna $intdata jest typu $IntType<BR>");

$DoubleType = gettype($doubledata);
print("Zmienna $doubledata jest typu $DoubleType<BR>");
```

```
$StringType = gettype($string);
print("Zmienna $string jest typu $StringType<BR>");

$BoolType = gettype($bool);
print("Zmienna $bool jest typu $BoolType<BR>");
```

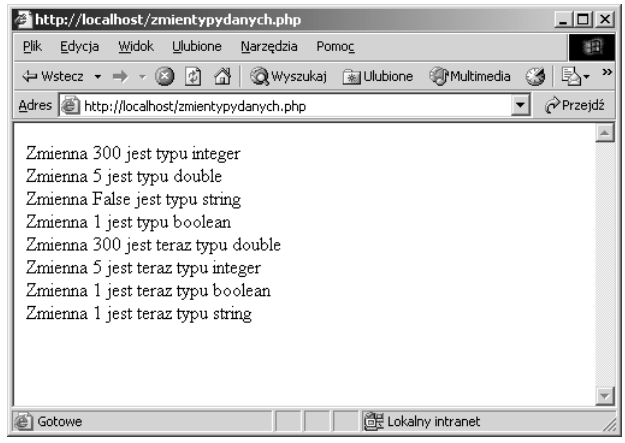
Następnie możliwa jest zmiana typu każdej ze zmiennych za pomocą funkcji `SetType`:

```
SetType($doubledata, "integer");
SetType($intdata, "double");
SetType($bool, "string");
SetType($string, "boolean");
```

Jako argumenty funkcji `SetType` podaj nazwę zmiennej, której typ chcesz zmienić (jak na przykład `$doubledata`), a następnie typ danych, na który typ zmiennej ma zostać zmieniony (na przykład `integer`). Na końcu możesz wyświetlić wartości i nowe typy zmiennych:

```
$IntType = gettype($intdata);
print("Zmienna $intdata jest typu $IntType<BR>");

$DoubleType = gettype($doubledata);
print("Zmienna $doubledata jest typu $DoubleType<BR>");

$StringType = gettype($string);
print("Zmienna $string jest typu $StringType<BR>");

$BoolType = gettype($bool);
print("Zmienna $bool jest typu $BoolType<BR>");
```

Jeśli uruchomisz ten skrypt, zobaczysz, że typy zmiennych zostały zmienione zgodnie z wydanymi poleceniami, jak pokazano na rysunku 3.3.



Rysunek 3.3. *Wykorzystanie funkcji SetType do zmiany typu zmiennej*

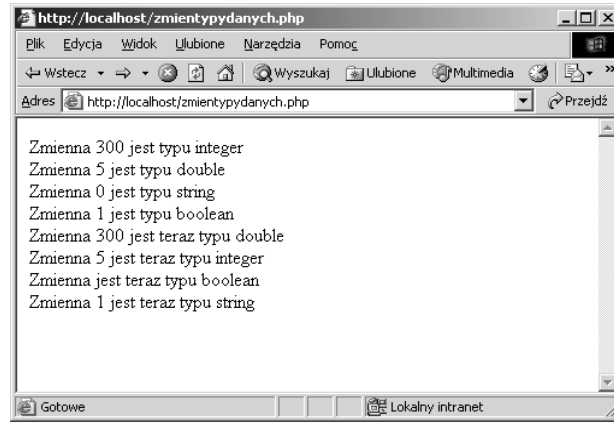
O ile zamiana zmiennych typu `integer` i `double` nie stanowiła większego problemu, to zamiana zmiennych typu `boolean` i `string` była większym wyzwaniem. Wyświetlone wyniki pokazują, że zmienna typu `string` ma wartość `1`. Jest to zgodne z zasadami wyświetlania przez PHP zmiennych typu `boolean`, ponieważ PHP wyświetla wartość `TRUE` jako `1`, dlatego po konwersji zmienna typu `string` ma wartość `1`.

Jak widać, również zmienna `boolean` ma wartość `1`. Zmienna typu `string` miała wartość `False`, lecz po konwersji na typ `boolean` zmienna nie była pusta i dlatego została uznana za zmienną o wartości `TRUE` czyli `1`. Aby po konwersji zmienna typu `Boolean` miała wartość `FALSE`, czyli `0`, zmienna typu `string` musiałaby przed konwersją mieć wartość `0` bądź być pusta:

```
$string = "0";
```

Jeśli dokonamy takiej zmiany w naszym przykładzie, to widać (patrz rysunek 3.4), że wartość zmiennej typu `boolean` jest widoczna jako pusta zmienna. Pamiętaj, że PHP traktuje wartość `0` jako wartość `FALSE` dla zmiennej typu `boolean`. Dlatego po konwersji zmiennej typu `string` o wartości `0` do typu `boolean` zawartość zmiennej jest traktowana jako `FALSE`. Należy również pamiętać, że PHP wyświetla zmienne typu `boolean` o wartości `False` jako zmienne puste. Stąd w naszym przykładzie została wyświetlona pusta zmienna.

Rysunek 3.4. *Dane zawierająca zmienną typu boolean o wartości False*



Rzutowanie typów

Rzutowanie typów to kolejny sposób przypisywania typów zmiennych.

```
<?php

$somedata = 6.23;

$newtype = GetType($somedata);

Print("Zmienna $somedata ma przypisany typ zmiennej $newtype oraz wartość
➥$somedata<BR><BR>");

//zmień na typ string
$zmientyp = (string) $somedata;

$newtype = GetType($zmientyp);

Print("Zmienna $somedata ma przypisany typ zmiennej $newtype oraz wartość
➥$zmientyp<BR>");

//zmień na typ integer
$zmientyp = (integer) $somedata;
```

```
$newtype = GetType($zmientyp);

Print("Zmienna $somedata ma przypisany typ zmiennej $newtype oraz wartość
➥$zmientyp<BR>");

//zmień na typ double
$zmientyp = (double) $somedata;

$newtype = GetType($zmientyp);

Print("Zmienna $somedata ma przypisany typ zmiennej $newtype oraz wartość
➥$zmientyp<BR>");

//zmień typ na boolean
$zmientyp = (boolean) $somedata;

$newtype = GetType($zmientyp);

Print("Zmienna $somedata ma przypisany typ zmiennej $newtype oraz wartość
➥$zmientyp<BR>");

?>
```

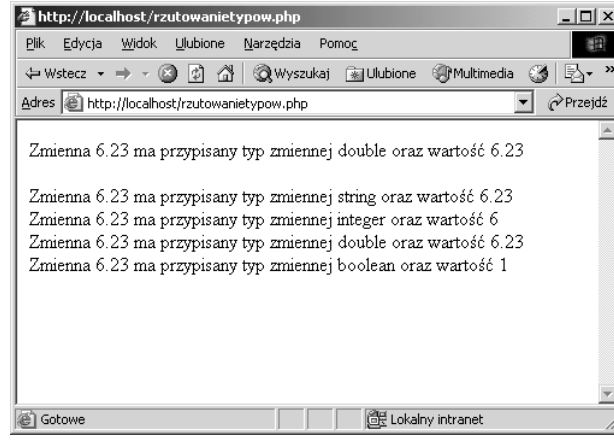
Najpierw przypisz wartość zmiennej:

```
$somedata = 6.23;
```

Wiadomo, że dla takiej wartości zmiennej, PHP ustawi typ zmiennej jako `double`. Możesz jednak zmienić ten typ. Jeśli chcesz zmienić typ zmiennej na `string`, możesz to zrobić za pomocą następującego polecenia:

```
$zmientyp = (string) $somedata;
```

Zmienna `$zmientyp` przyjmuje wartość zmiennej `$somedata`, ale przypisywany jest jej typ `string`. Takie rzutowanie typów jest możliwe dla wszystkich typów zmiennych, jak widać na przedstawionym przykładzie. Po uruchomieniu skryptu przekonasz się, jak działa rzutowanie skryptów (pokazano to na rysunku 3.5).



Rysunek 3.5.
Rzutowanie typów

W przedstawionym przykładzie widać, że wartość `6.23` została uznana przez PHP za typ `double`. Po wykonaniu rzutowania typów, widoczne są zmiany wartości zmiennej w zależności od bieżącego typu zmiennej.

Operatory arytmetyczne

PHP pozwala używać operatorów arytmetycznych na zgodnych typach zmiennych. Jakie to typy? Wszystkie przechowujące dane w formie numerycznej (oznacza to dowolny typ zmiennych z wyjątkiem typu `boolean`). W kolejnym przykładzie przedstawiony zostanie również operator konkatenacji (połączenia ciągów). Nie martw się, jeśli nie jesteś zaznajomiony z tą kwestią. Przyjrzymy się jej za moment:

```
<?php

//używane zmienne

$num1 = 10;
$num2 = 20;

//dodawanie
$suma = $num1 + $num2;
Print($num1 . " + " . $num2 . " = " . $suma);
Print("<BR>");

//odejmowanie
$roznica = $num2 - $num1;

Print($num2 . " - " . $num1 . " = " . $roznica);

Print("<BR>");

//dzielenie
$iloraz = $num2 / $num1;

Print($num2 . " / " . $num1 . " = " . $iloraz);

Print("<BR>");

//mnożenie
$iloczyn = $num2 * $num1;

Print($num2 . " * " . $num1 . " = " . $iloczyn);

Print("<BR>");

//modulo
$modulo = $num2 % $num1;

Print($num2 . " % " . $num1 . " = " . $modulo);

Print("<BR>");

?>
```

Zaczynamy od przypisania wartości dwóch zmiennych, na których będziemy wykonywać operacje arytmetyczne:

```
$num1 = 10;
$num2 = 20;
```

Następnie wykonujemy działanie arytmetyczne na tych zmiennych:

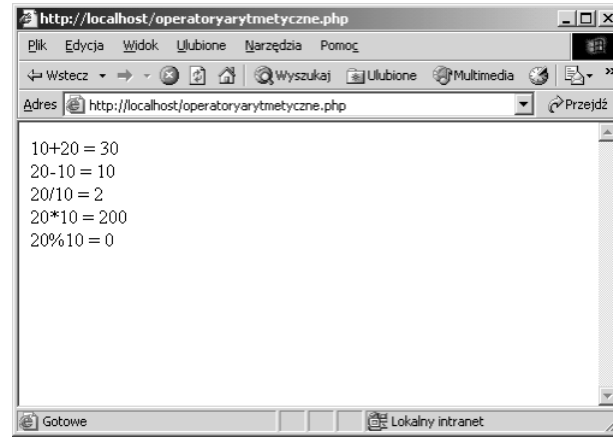
```
$suma = $num1 + $num2;
```

W końcu wyświetlany jest wynik operacji:

```
Print($num1 . " + " . $num2 . " = " . $suma);
```

Jeśli uruchomisz skrypt, wyświetlone zostaną wyniki operacji arytmetycznych pokazane na rysunku 3.6.

Rysunek 3.6.
Operacje arytmetyczne



Warto nadmienić, że następująca deklaracja jest również poprawna:

```
$num1 = 10;
$num2 = "20";
```

W ten sposób utworzone zmienne będą różnych typów. Zmienna `$num1` jest typu `integer`, a zmienna `$num2` jest typu `string`. Jeśli zmienisz wartości zmiennych zgodnie z poniższym przykładem, skrypt nadal będzie działał poprawnie:

```
$num1 = 10;
$num2 = "aa20";
```

Skrypt nadal będzie wykonany poprawnie, ale operatory arytmetyczne nie zadziałają. Jeśli PHP napotka dane innego typu niż numeryczne, zignoruje tę zmienną.

Operatory łańcuchowe i funkcje

PHP udostępnia kilka sposobów pracy na zmiennych łańcuchowych (`string`). Często korzystasz ze zmiennych łańcuchowych pobranych z takiego źródła jak formularze HTML, pliki czy bazy danych, dlatego PHP umożliwia manipulowanie tymi danymi na różne sposoby.

Rozpoczniemy naszą pracę ze zmiennymi łańcuchowymi od zdefiniowania kilku zmiennych:

```
$imie ="Fox";
$spacja = " ";
$nazwisko = "Mulder";
```

Łączenie ciągów

Aby połączyć zdefiniowane wcześniej ciągi, użyj operatora konkatenacji:

```
$imie_nazwisko = $imie . $spacja . $nazwisko;
```

Utworzyłeś nową zmienną łańcuchową przez połączenie trzech ciągów. Zauważ, że do łączenia łańcuchów służy znak kropki (`.`).

Długość ciągu

Aby poznać długość nowego ciągu, użyj funkcji `strlen`:

```
$dlugosc_ciagu = strlen($imie_nazwisko);
```

Dodatkowa spacja

PHP umożliwia usuwanie dodatkowego znaku spacji (często zwanego również „białą spacją") znajdującego się na końcu lub na początku łańcucha znaków. Ponieważ nasz nowy łańcuch nie kończy się ani nie zaczyna „białą spacją", musimy ją najpierw dodać, aby następnie użyć tych funkcji:

```
$imie_nazwisko_i_spacje = " $imie_nazwisko ";
```

Teraz możesz usunąć początkową spację za pomocą funkcji `ltrim`:

```
$ltrim_imie_nazwisko = ltrim($imie_nazwisko_i_spacje);
```

„Białą spację" zarówno rozpoczynającą, jak i kończącą łańcuch znaków możesz usunąć za pomocą funkcji `trim`:

```
$trim_imie_nazwisko = trim($imie_nazwisko_i_spacje);
```

Wielkość liter

PHP pozwala zmienić wielkość wszystkich liter w łańcuchu na wielkie lub na małe. Aby zmienić wszystkie litery w łańcuch na wielkie, użyj funkcji `strtoupper`:

```
$imie_nazwisko_wielkielitery = strtoupper($imie_nazwisko);
```

Funkcja `strtolower` służy do zamiany wszystkich liter w łańcuchu na małe:

```
$imie_nazwisko_malelitery = strtolower($imie_nazwisko);
```

Podłańcuchy znaków

PHP umożliwia wyszukanie lokalizacji, utworzenie oraz dodanie podłańcucha do łańcucha znaków. Do utworzenia podłańcucha znaków służy funkcja `substr`:

```
$pod_lancuch = substr($imie_nazwisko,3);
```

Jako argument podawany jest łańcuch znaków, z którego utworzony ma być podłańcuch (w naszym przypadku jest to łańcuch `$imie_nazwisko`), i pozycja w łańcuchu, od której podłańcuch ma się zaczynać (w naszym przykładzie podłańcuch ma się zaczynać od trzeciego znaku w łańcuchu).

Możesz odszukać lokalizację podłańcucha wewnątrz łańcucha za pomocą funkcji `strpos`:

```
$pozycja_podlancuch = strpos($imie_nazwisko, $pod_lancuch);
```

Argumentami tej funkcji jest łańcuch, wewnątrz którego podłańcuch ma być wyszukany (u nas jest to łańcuch `$imie_nazwisko`), oraz wyszukiwany podłańcuch (jest to wcześniej utworzony podłańcuch `$pod_lancuch`).

Możesz również zamienić wybrany podłańcuch na inny we wskazanym łańcuchu za pomocą funkcji `str_replace`:

```
$pod_imie_nazwisko = str_replace("Fox", "Scully", $imie_nazwisko);
```

Argumentem funkcji `str_replace` jest łańcuch, który ma być wyszukany (w naszym przypadku jest to `"Fox"`), łańcuch, którym zostanie on zastąpiony (w naszym przykładzie jest to łańcuch `"Scully"`), oraz łańcuch, wewnątrz którego zamiana zostanie dokonana (w naszym przykładzie jest to `$imie_nazwisko`).

Testowanie metod obsługi łańcuchów

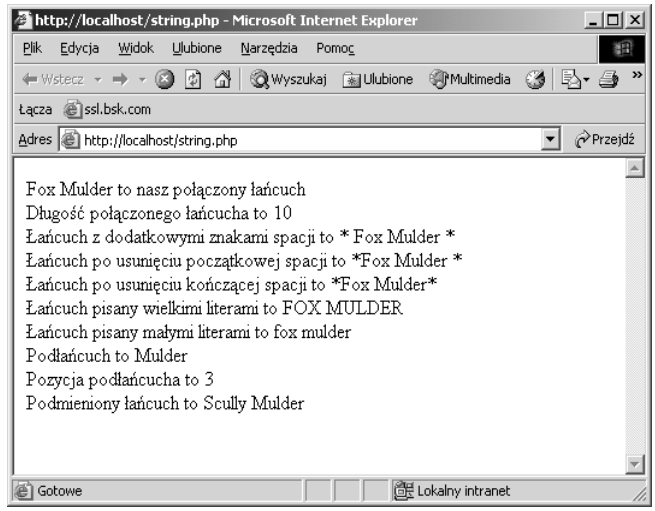
Możesz sprawdzić wszystkie opisane wcześniej metody za pomocą następującego skryptu:

```
<?php

//łańcuchy
$imie = "Fox";
$spacja = " ";
$nazwisko = "Mulder";

//łączenie
$imie_nazwisko = $imie . $spacja . $nazwisko;

print("$imie_nazwisko to nasz połączony łańcuch");

print("<BR>");

//długość
$dlugosc_lancucha = strlen($imie_nazwisko);

print("Długość połączonego łańcucha to $dlugosc_lancucha");

print("<BR>");

//dodatkowa spacja
$imie_nazwisko_i_spacja = " $imie_nazwisko ";

print("Łańcuch z dodatkowymi znakami spacji to *$imie_nazwisko_i_spacja*");

print("<BR>");
```

```
$ltrim_imie_nazwisko = ltrim($imie_nazwisko_i_spacja);

print("Łańcuch po usunięciu początkowej spacji to *$ltrim_imie_nazwisko*");

print("<BR>");

$trim_imie_nazwisko = trim($imie_nazwisko_i_spacja);

print("Łańcuch po usunięciu kończącej spacji to *$trim_imie_nazwisko*");

print("<BR>");

//wielkość liter
$imie_nazwisko_wielkielitery = strtoupper($imie_nazwisko);

print("Łańcuch pisany wielkimi literami to $imie_nazwisko_wielkielitery");

print("<BR>");

$imie_nazwisko_malelitery = strtolower($imie_nazwisko);

print("Łańcuch pisany małymi literami to $imie_nazwisko_malelitery");

print("<BR>");

//podłańcuchy

//podłańcuch
$pod_lancuch = substr($imie_nazwisko,3);

print("Podłańcuch to $pod_lancuch");

print("<BR>");

//pozycja podłańcucha
$podlancuch_pozycja = strpos($imie_nazwisko, $pod_lancuch);

print("Pozycja podłańcucha to $podlancuch_pozycja");

print("<BR>");

//podmień podłańcuch
$pod_imie_nazwisko = str_replace ("Fox", "Scully", $imie_nazwisko);

print("Podmieniony łańcuch to $pod_imie_nazwisko");

print("<BR>");

?>
```

Jeśli uruchomisz powyższy skrypt, zobaczysz, jak działają metody obsługi łańcuchów, co pokazano na rysunku 3.7. Zauważ, że wyświetlając wyniki działania metody konkatenacji, po obu stronach łańcucha umieściłem znaki *. Pozwala to pokazać znaki spacji umieszczone pomiędzy tymi znakami a łańcuchem.

Rysunek 3.7.
Metody obsługi łańcuchów



Operatory logiczne i pętle

Kontrolowanie działania programu jest niezwykle istotną kwestią. Być może musisz sprawdzić określone warunki logiczne lub wykonać pętlę po sprawdzeniu warunku. Takie elementy były zawsze obecne w językach programowania i w tym przypadku PHP nie jest wyjątkiem.

Operatory logiczne

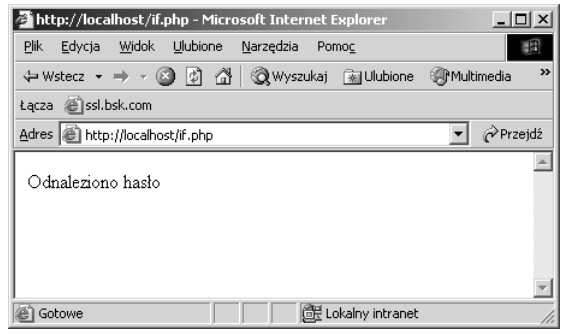
W PHP istnieje kilka sposobów sprawdzania warunków logicznych. Sprawdzając warunek logiczny, porównujesz jedną wartość z inną wartością i sprawdzasz, czy są sobie równe:

```
<?php

$haslo = "a12b";

if ($haslo == "a12b") {
     print("Odnaleziono hasło");
}

?>
```

Ten fragment kodu wykorzystuje instrukcję `if`. Składnia tej instrukcji jest prosta. Jeśli wartość jest równa innej wartości, PHP zwraca wartość wyrażenia jako „prawda". W powyższym fragmencie kodu sprawdzamy, czy wartość zmiennej `$haslo` jest równa `a12b`. Jeśli tak, to wyświetlany jest komunikat, jak pokazano na rysunku 3.8.

Co by się stało, gdyby wartość zmiennej `$haslo` była różna od podanej w warunku wartości? Poprzedni przykład nie wyświetliłby żadnej wartości. Jednak za pomocą PHP również możesz wykonywać określone działania, jeśli warunek nie zostanie spełniony.

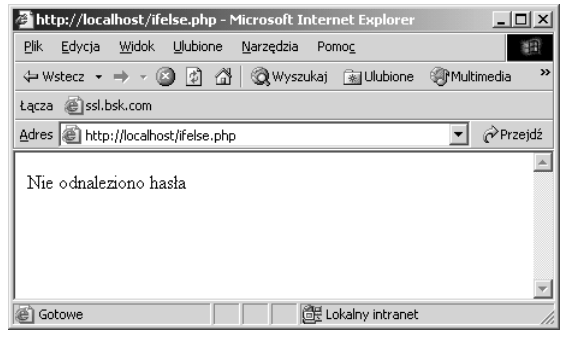
Rysunek 3.8. *Testowanie prawdziwości wyrażenia za pomocą instrukcji if*




```
<?php

$haslo = "3333";

if($haslo == "a12b") {
   print ("Odnaleziono hasło");

} else {
   print("Nie odnaleziono hasła");
}

?>
```

Jeśli wartości, której szukasz, nie odnaleziono, wykonane zostanie działanie zdefiniowane po instrukcji `else`. Jeśli wartość zmiennej `$haslo` jest równa podanej wartości, to wyświetlony zostanie komunikat `"Odnaleziono hasło"`. Jeśli jednak wartość zmiennej jest różna od podanej wartości, to wyświetlony zostanie komunikat `"Nie odnaleziono hasła"`. Nadanie zmiennej `$haslo` wartości innej niż podana w warunku powoduje wyświetlenie komunikatu `"Nie odnaleziono hasła"`, jak pokazano na rysunku 3.9.

Rysunek 3.9. *Sprawdzanie nieprawdziwości wyrażenia za pomocą instrukcji if else*



Jeśli masz wiele warunków do sprawdzenia za pomocą instrukcji `if`, możesz to zrobić w następujący sposób:

```
<?php

$haslo = "bb22";

if ($haslo == "aa11") {
     print("Odnaleziono hasło dla użytkownika Jan");
}
```

```
if ($haslo == "bb22") {
     print("Odnaleziono hasło dla użytkownika Alfred");
}

if ($haslo = "cc33") {
     print("Odnaleziono hasło dla użytkownika Jacek");
}

?>
```

Największym problemem sprawia tutaj fakt, że PHP wykona wszystkie instrukcje `if`, nawet jeśli pierwszy warunek zostanie spełniony. Aby tego uniknąć, możesz wykorzystać zagnieżdżone instrukcje `if`:

```
<?php

$haslo = "bb22";

If($haslo == "aa11") {
        Print("Odnaleziono hasło dla użytkownika Jan");

} elseIf ($haslo == "bb22") {
        Print("Odnaleziono hasło dla użytkownika Alfred");

} elseIf ($haslo == "cc33") {
        Print("Odnaleziono hasło dla użytkownika Jacek");
}

?>
```

Jednak, jeśli utworzyłeś wiele zagnieżdżonych instrukcji `if`, pisany przez Ciebie kod szybko stanie się nieczytelny. Lepszym sposobem będzie użycie instrukcji `switch` i `case`:

```
<?

$haslo = "bb22";

switch($haslo) {
            case "aa11":
            Print("Odnaleziono hasło dla użytkownika Jan");
            break;

            case "bb22":
            Print("Odnaleziono hasło dla użytkownika Alfred");
            break;

            case "cc33":
            Print("Odnaleziono hasło dla użytkownika Jacek");
            break;
}

?>
```

W tym przykładzie sprawdzana jest każda wartość zmiennej `$haslo` w instrukcji `case`. Jeśli odnaleziona zostanie pasująca wartość, wyświetlany jest komunikat o odnalezieniu hasła dla danego użytkownika.

Zauważ, że użyto instrukcji `break`, aby po odszukaniu odpowiedniej wartości przerwane zostało wykonywanie dalszych instrukcji wewnątrz instrukcji `switch`. W przeciwnym razie instrukcja `switch` byłaby wykonywana analogicznie jak wcześniej przedstawiona instrukcja `if`, czyli sprawdzane zostałyby kolejno wszystkie warunki.

Iteracje

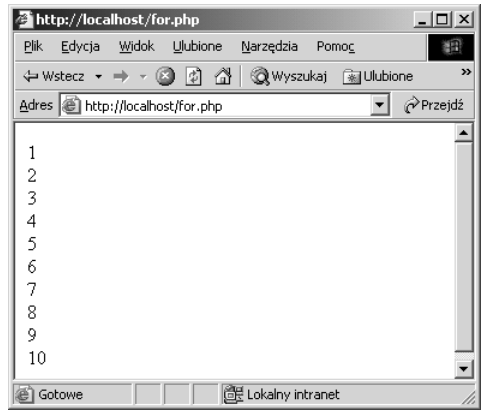
PHP umożliwia wykonywanie w programie pętli na kilka sposobów. Wszystkie pętle pozwalają wykonywać serie poleceń, aż do momentu, kiedy warunek wykonania pętli zostanie spełniony, na przykład licznik osiągnie określoną wartość. Pierwszą metodą jest pętla `for`:

```
<?php

for ($licznik = 1; $licznik <= 10; $licznik++) {
                              print("$licznik");
                              print("<BR>");
}

?>
```

Pętla `for` składa się z następujących elementów:

1. Wartości rozpoczynającej pętlę — w naszym przypadku jest to 1 (`$licznik = 1`).
2. Sposobu sprawdzania wartości. W tym przypadku, jeśli wartość zmiennej wyniesie 10, pętla przestanie być wykonywana (`$licznik <= 10`).
3. Sposobu zmiany wartości wewnątrz pętli. W naszym przykładzie wartość jest zwiększana o jeden przy każdym przebiegu pętli (`$licznik++`).

Dlatego pętla wyświetla liczby od 1 do 10 (rysunek 3.10).



Rysunek 3.10. *Inkrementacja licznika pętli for od 1 do 10*

Wewnątrz pętli `for` ma miejsce kilka ważnych zdarzeń. Pierwszym z nich jest testowanie warunku logicznego. W naszym przykładzie sprawdzaliśmy, czy wartość zmiennej jest równa lub mniejsza od 10. Możesz również sprawdzać inne warunki, jak pokazano w tabeli 3.1.

Tabela 3.1. *Inne warunki*

Warunki	Znaczenie
==	równy
!=	różny
<	mniejszy niż
>	większy niż
>=	większy lub równy
<=	mniejszy lub równy
===	identyczny (równa wartość i taki sam typ)

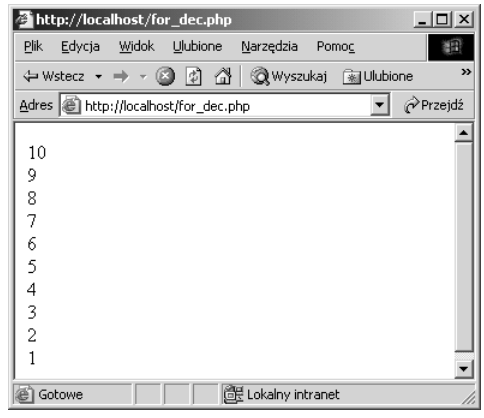
Możesz używać tych warunków nie tylko w połączeniu z instrukcją `for`, ale z dowolną instrukcją, która sprawdza wartość warunku logicznego, na przykład z instrukcją `if`. Przyjrzyj się, w jaki sposób w poprzednim przykładzie zmieniana jest wartość licznika podczas wykonywania pętli. Wartość licznika może być inkrementowana lub dekrementowana. Dekrementowanie licznika będzie miało postać:

```
$licznik--
```

Możemy zmienić nasz przykład w następujący sposób:

```
<?php

for ($licznik = 10; $licznik >=1; $licznik -- ){
          print($licznik);
          print("<BR>");
}

?>
```

Po uruchomieniu tego skryptu pętla wykona odliczanie od 10 do 1, jak pokazano na rysunku 3.11.



Rysunek 3.11. *Dekrementacja licznika w pętli for od 10 do 1*

Innym sposobem wykonania pętli jest instrukcja `while`:

```
<?php

$licznik = 1;

while ($licznik <= 10) {
          print("$licznik");
          print("<BR>");
          $licznik++;
}

?>
```

Na początku przypisywana jest wartość początkowa zmiennej `$licznik` — w naszym przykładzie jest to 1:

```
$licznik =1;
```

Następnie wykonywana jest pętla do chwili, gdy wartość licznika osiągnie 10:

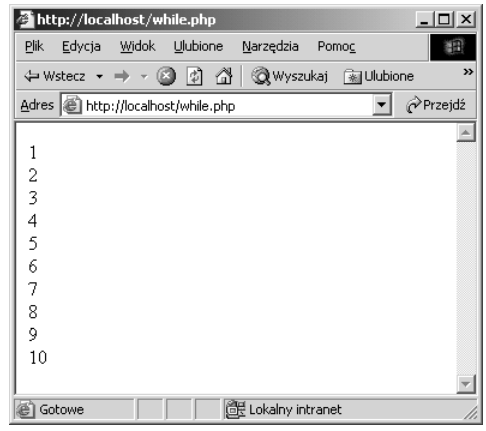
```
while ($licznik <= 10) {
```

Podobnie jak w przypadku pętli `for` możesz użyć różnych operatorów porównania. Wewnątrz pętli wartość zmiennej `$licznik` jest inkrementowana:

```
$licznik ++
```

Oczywiście możesz również zastosować podobną metodę jak w naszym drugim przykładzie z pętlą `for` — dekrementować wartość licznika. Po uruchomieniu skryptu pętla `while` odlicza od 1 do 10, jak pokazano na rysunku 3.12.

Rysunek 3.12. *Inkrementowana pętla while odliczająca od 1 do 10*



PHP umożliwia sterowanie wykonaniem pętli za pomocą instrukcji `do while`:

```
<?php

$licznik = 1;

do {
         print("$licznik");
         print("<BR>");
         $licznik++;
```

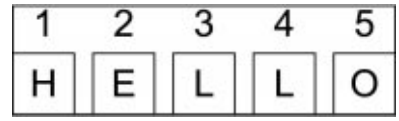
```
} while ($licznik <=10);

?>
```

Przedstawiona pętla wygląda podobnie jak pętla `while`. Po uruchomieniu skryptu wykonane zostanie liczenie od 1 do 10, a następnie pętla zostanie zakończona. Czym się w takim razie różni od pętli `while`? Różnica pomiędzy tymi pętlami polega na lokalizacji instrukcji `while`. W pętli `do while` wartość zmiennej licznika jest modyfikowana (w naszym przykładzie jest inkrementowana), a następnie wartość tej zmiennej jest sprawdzana. W pętli `while` wartość zmiennej jest testowana, a następnie zmieniana. Istotny jest moment zmiany licznika i moment jego sprawdzenia. Tutaj zmienna `$licznik` przy wartości 1 nigdy nie jest sprawdzana, ponieważ przed sprawdzeniem jest inkrementowana do wartości 2.

Tablice

Tablica zapewnia uporządkowany sposób przechowywania danych, do których można się w prosty sposób odwoływać. Na rysunku 3.13 widać, że tablica wygląda jak talia numerowanych kart, a na każdej karcie jest wydrukowana wartość. Na przykład karta 3 ma nadrukowaną wartość L.

Rysunek 3.13.
Struktura tablicy



Każda karta w naszym przykładzie jest określana jako *element tablicy*. Wartość poszczególnych elementów tablicy to *wartość tablicy*, a do każdego elementu tablicy można się odwołać poprzez *klucz*. W PHP dostępnych jest kilka sposobów obsługi tablicy.

```
<?php

$wiadomosc[0] = "H";
$wiadomosc[1] = "E";
$wiadomosc[2] = "L";
$wiadomosc[3] = "L";
$wiadomosc[4] = "O";

while (list($klucz, $wartosc) = each ($wiadomosc)) {
            print("$wartosc");
            print(" klucz tablicy $klucz równa się wartości $wartosc");
            print("<BR>");
}

?>
```

Rozpoczynamy od utworzenia tablicy:

```
$wiadomosc[0] = "H";
$wiadomosc[1] = "E";
$wiadomosc[2] = "L";
$wiadomosc[3] = "L";
$wiadomosc[4] = "O";
```

Wykorzystywana jest zmienna typu `array`. Mówi o tym nawias kwadratowy umieszczony za nazwą zmiennej. Zauważ, że tablica zaczyna się od pozycji 0; w PHP wszystkie tablice zaczynają się od tej pozycji. W naszym przykładzie wartość elementu `$wiadomosc[1]` jest równa E. Teraz, kiedy utworzona została tablica, musisz mieć możliwość dostępu do jej elementów. PHP umożliwia odwołanie do kolejnych elementów tablicy w następujący sposób:

```
while (list($klucz, $wartosc) = each ($wiadomosc)) {
            print("$wartosc");
            print("<BR>");
}
```

Pętla `while` przechodzi przez kolejne elementy tablicy. Tablica jest dzielona na klucze elementów i wartości elementów za pomocą instrukcji `list`, a następnie instrukcja `each` pobiera wartości kolejnych elementów tablicy. Wewnątrz pętli wyświetlane są kolejne elementy tablicy:

```
print("klucz tablicy $klucz równa się wartości $wartosc");
```

Po uruchomieniu skryptu wyświetlone zostaną kolejne elementy tablicy.

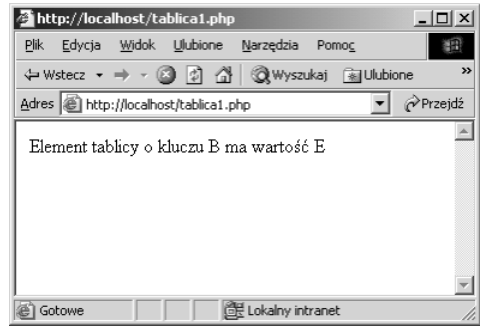
Jako kluczy poszczególnych elementów tablicy nie musisz używać liczb. Możesz używać dowolnych wartości. Możemy zmienić powyższy przykład, zastępując liczby literami:

```
<?php

$wiadomosc["A"] = "H";
$wiadomosc["B"] = "E";
$wiadomosc["C"] = "L";
$wiadomosc["D"] = "L";
$wiadomosc["E"] = "O";

while (list($klucz, $wartosc) = each ($wiadomosc)) {
            print("$wartosc");
            print("<BR>");
}

?>
```

Możesz również wyszukiwać wartości w tablicy, posługując się kluczem elementu:

```
<?php

$wiadomosc["A"] = "H";
$wiadomosc["B"] = "E";
$wiadomosc["C"] = "L";
$wiadomosc["D"] = "L";
$wiadomosc["E"] = "O";

while (list($klucz, $wartosc) = each ($wiadomosc)) {
       if($klucz == "B"){
            print("Element tablicy o kluczu $klucz ma wartość $wartosc");
       }
}

?>
```

Powyższy kod wyszukuje element tablicy o kluczu B i wyświetla jego wartość, jak pokazano na rysunku 3.14.

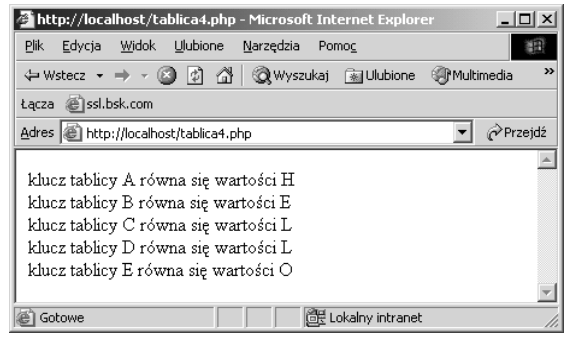


Rysunek 3.14. *Wartość elementu tablicy wyszukana poprzez klucz tego elementu*

PHP umożliwia również tworzenie tablicy za pomocą instrukcji `array`:

```
<?

$wiadomosc = array ("A" => "H", "B" => "E", "C" => "L", "D" => "L", "E" => "O");

while (list ($klucz, $wartosc) = each ($wiadomosc)) {
                print("klucz tablicy $klucz równa się wartości $wartosc");
                print("<BR>");
}

?>
```

Zmienna `$wiadomosc` zawiera tablicę utworzoną za pomocą instrukcji `array`. Instrukcja `array` pozwala tworzyć tablicę przez podanie kolejnych kluczy elementów i ich wartości. Po uruchomieniu skryptu zobaczysz zawartość struktury utworzonej przez instrukcję `array`, jak pokazano na rysunku 3.15.

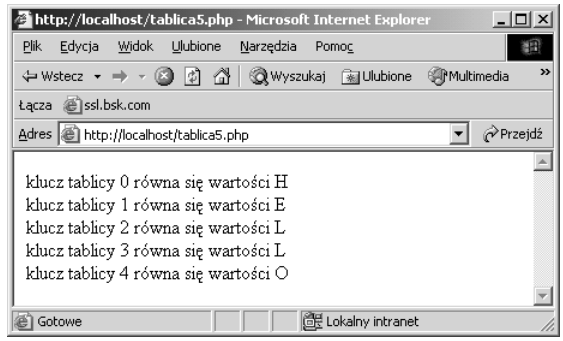


Rysunek 3.15. *Klucze i wartości elementów tablicy utworzonej przez instrukcję array*

Instrukcja `array` samodzielnie nadaje klucze kolejnym elementom tablicy, jeśli nie zostaną one osobno zdefiniowane:

```
<?

$wiadomosc = array ("H" , "E", "L", "L", "O");

while (list ($klucz, $wartosc) = each ($wiadomosc)) {
              print("klucz tablicy $klucz równa się wartości $wartosc");
              print("<BR>");
}

?>
```

W tym przykładzie podaliśmy jedynie wartości elementów tablicy tworzonej instrukcją `array`. Po uruchomieniu skryptu widać, że instrukcja `array` samodzielnie nadała klucze kolejnym elementom tablicy (patrz rysunek 3.16).



Rysunek 3.16. *Klucze kolejnych elementów tablicy są automatycznie tworzone przy użyciu instrukcji array*

Sortowanie tablicy

Zdarza się, że tablica zawiera nieuporządkowane elementy; wartości elementów i kluczy nie są umieszczone na kolejnych pozycjach tablicy i należy je uporządkować. Tworzenie fragmentów kodu, który wykonuje porządkowanie tablicy może być bardzo czasochłonne (nawet, jeśli korzystamy z napisanych przez innych programistów fragmentów kodu). Na szczęście PHP zawiera zestaw instrukcji, które służą do tego celu.

W kolejnych punktach omówimy cztery typy instrukcji `sort` przeznaczonej do sortowania tablicy według wartości elementów tablicy lub ich kluczy.

Funkcja sort

Instrukcja `sort` pozwala posortować wartości elementów tablicy w porządku rosnącym:

```
<?php

//sortowanie zawartości tablicy w porządku rosnącym
$numerki = array ( 5, 2, 3, 1, 4);

//nieuporządkowane
print("nieuporządkowana tablica <BR>");

while (list ($klucz, $wartosc) = each ($numerki)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

print ("<BR>");

//uporządkowane
print("uporządkowana tablica <BR>");

sort ($numerki);
```

```
while (list($klucz, $wartosc) = each ($numerki)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

?>
```

Powyższy fragment kodu powoduje utworzenie tablicy za pomocą instrukcji `array`, która samodzielnie nadaje klucze kolejnym elementom tablicy. Następnie skrypt wyświetla zawartość nieuporządkowanej tablicy, porządkuje elementy tablicy i ponownie wyświetla zawartość tablicy, tym razem już posortowaną.

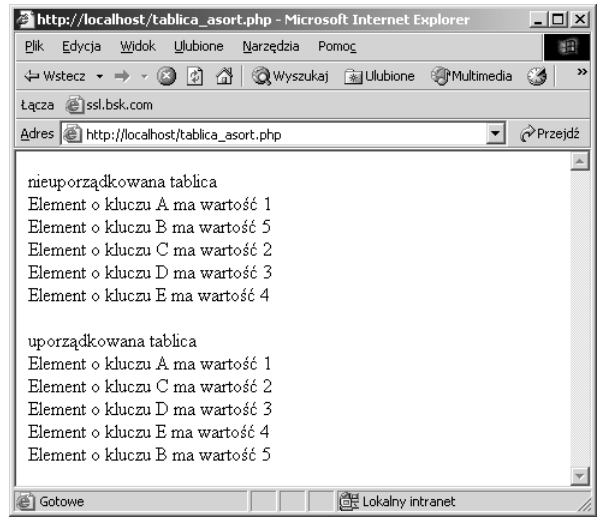
Funkcja asort

Zauważ, że podczas sortowania tablicy przy użyciu instrukcji `sort` zmieniane są klucze elementów tablicy. Przed sortowaniem klucz 0 wskazywał wartość 5, natomiast po sortowaniu ten sam klucz wskazywał wartość 1. Możesz zachować pary klucz-wskazywany element, równocześnie porządkując rosnąco elementy tablicy. Jak? Użyj funkcji `asort`:

```
<?php

//sortowanie zawartości tablicy w porządku rosnącym, zachowując przypisania kluczy
$wiadomosc = array ( "A"=> "1", "B"=> "5", "C"=> "2", "D"=> "3", "E"=> "4");

//nieuporządkowane
print("nieuporządkowana tablica <BR>");

while (list ($klucz, $wartosc) = each ($wiadomosc)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

print ("<BR>");

//uporządkowane
print("uporządkowana tablica <BR>");

asort ($wiadomosc);

while (list($klucz, $wartosc) = each ($wiadomosc)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

?>
```

W naszym kodzie podczas definiowania tablicy podawane są zarówno wartości elementów, jak i ich klucze, jednak możesz również skorzystać z automatycznego nadawania kluczy przez wykorzystanie instrukcji `array`, podobnie jak w poprzednim przykładzie opisującym instrukcję `sort`. Następnie skrypt wyświetla zawartość nieuporządkowanej tablicy, sortuje elementy tablicy za pomocą instrukcji `asort` i ponownie wyświetla zawartość tablicy, jak pokazano na rysunku 3.17.

Elementy tablicy zostały uporządkowane, ale powiązania pomiędzy kluczami i elementami tablicy pozostały niezmienione. Na przykład klucz E wskazuje element tablicy o wartość 4 przed i po sortowaniu.



Rysunek 3.17. *Tablica posortowana w porządku rosnącym z wykorzystaniem instrukcji asort, która nie modyfikuje powiązań kluczy z elementami tablicy*

Funkcja rsort

Możliwe jest również posortowanie elementów tablicy w porządku malejącym za pomocą funkcji `rsort`:

```
<?php

//sortowanie tablicy w porządku malejącym
$wiadomosc = array( 5, 2, 3, 1, 4);

//nieposortowana
print("nieposortowana tablica<BR>");

while(list($klucz, $wartosc) = each($wiadomosc)) {
          print("Element o kluczu $klucz ma wartość $wartosc");
          print("<BR>");
}

print("<BR>");

//posortowana
print("posortowana tablica<BR>");

rsort($wiadomosc);

while(list($klucz, $wartosc) = each ($wiadomosc)) {
          print("Element o kluczu $klucz ma wartość $wartosc");
          print("<BR>");
}

?>
```

Powyższy przykład działa analogiczne jak przykładowy skrypt opisujący funkcję `sort`. Funkcja `rsort` sortuje tablicę w porządku malejącym.

Funkcja ksort

PHP umożliwia sortowanie kluczy tablicy w porządku rosnącym za pomocą funkcji `ksort`:

```
<?php

//sortowanie kluczy tablicy w porządku rosnącym
$wiadomosc = array ( "B"=> "1", "C"=> "5", "D"=> "2", "A"=> "3", "E"=> "4");

//nieuporządkowane
print("nieuporządkowana tablica <BR>");

while (list ($klucz, $wartosc) = each ($wiadomosc)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

print ("<BR>");

//uporządkowane

ksort ($wiadomosc);

print("uporządkowana tablica <BR>");

while (list($klucz, $wartosc) = each ($wiadomosc)) {
            print("Element o kluczu $klucz ma wartość $wartosc");
            print("<BR>");
}

?>
```

Powyższy skrypt wygląda analogiczne jak poprzednie przykłady, z tą różnicą, że wykorzystuje instrukcję `ksort` do porządkowania tablicy.

W tym przykładzie kluczami elementów tablicy są litery. Po posortowaniu za pomocą instrukcji `ksort`, klucze są sortowane w porządku alfabetycznym.

Kod strukturalny i kod wielokrotnego użytku

Moje pierwsze spotkanie z programowaniem miało miejsce na studiach. Moim nauczycielem był Chris Pickford, a był on jednym z najbardziej cierpliwych ludzi, jakich spotkałem w życiu. Chris dawał z siebie wszystko, próbując nauczyć grupę niedoświadczonych, ale bardzo chętnych, programistów podstaw i zalet programowania strukturalnego i nadającego się do wielokrotnego wykorzystania kodu. Dopiero kiedy zacząłem żyć z programowania zrozumiałem, jak ważne było to, czego nauczył mnie Chris. Ten punkt dedykuję Tobie, Chris.

Kod strukturalny i kod wielokrotnego użytku pozwala rozbić program na uporządkowane fragmenty, w celu efektywnego wykorzystania fragmentów kodu wewnątrz aplikacji. Takie podejście jest korzystne, ponieważ aplikacja może wielokrotnie wykonywać te same zadania, takie jak łączenie się z bazą danych, wystawianie zapytania do bazy danych

i wyświetlanie danych otrzymanych z bazy danych. Podzielenie aplikacji na fragmenty pozwala zmniejszyć możliwość wystąpienia błędów oraz skrócić czas tworzenia aplikacji (oszczędzanie czasu to zawsze ważna rzecz). PHP pozwala dzielić kod na kilka sposobów przy wykorzystaniu funkcji, klas i dołączanych bibliotek.

Korzystanie z funkcji

Korzystanie z funkcji to jeden z najprostszych sposobów dzielenia kodu na fragmenty, które mogą być wielokrotnie wykorzystywane.

```
<?php

function WiadomoscPowitalna($imie) {
return "Hello" . $imie;
}

$zwrot = WiadomoscPowitalna("Andrzej");

print("$zwrot");

?>
```

W powyższym przykładzie utworzona została funkcja wyświetlająca wiadomość. Do funkcji jako argument przesyłane jest imię, które następnie jest umieszczane jako część wyświetlanej wiadomości. Najpierw tworzona jest funkcja:

```
function WiadomoscPowitalna($imie) {
return "Hello" . $imie;
}
```

Zauważ, że wyniki wykonania funkcji są zwracane w następujący sposób:

```
return "Hello" . $imie;
```

Innymi słowy, instrukcja `return` zwraca wyniki wykonania funkcji po jej wywołaniu. Dlatego, aby pobrać wyniki wykonania funkcji, należy zapisać je jako zmienną:

```
$zwrot = WiadomoscPowitalna("Andrzej");
```

W końcu wyświetlane są wyniki wykonania funkcji:

```
print("$zwrot");
```

Po uruchomieniu skryptu wyświetlone zostaną wyniki wywołania funkcji.

Przekazywanie danych przez zmienną lub przez wartość

Za pomocą funkcji można również modyfikować zawartość przekazywanych do nich zmiennych. Przekazując zmienną do funkcji, możesz zachować niezmienioną wartość zmiennej po zakończeniu wykonywania funkcji (nazywane jest to przekazywaniem przez wartość) lub zmienić dane (nazywamy to przekazywaniem przez zmienną).

```
<?php

function WhoIsCool($argument) {
$argument .= ' są cool';
```

```
}
$nazwa = "Programiści PHP";

//przez wartość
WhoIsCool ($nazwa);
print($nazwa);

//nowa linia
print("<BR>");

//przez zmienną
WhoIsCool(&$nazwa);
print($nazwa);

?>
```

Powyższy kod modyfikuje zmienną `$nazwa`. Najpierw zmiennej nadawana jest wartość:

```
$nazwa = "Programiści PHP";
```

Funkcja łączy łańcuch danych ze zmienną łańcuchową przekazywaną do funkcji:

```
function WhoIsCool($argument) {
$argument .= ' są cool';
}
```

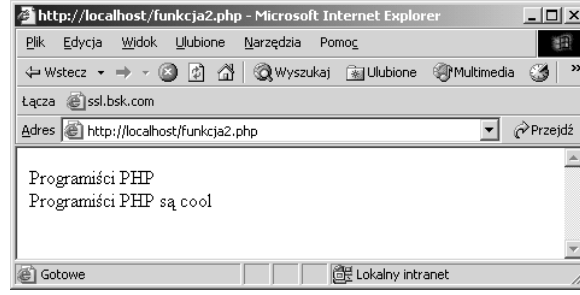
Następnie wywoływana jest funkcja, do której argument jest przekazywany przez wartość i wyniki działania funkcji są wyświetlane. Zauważ, że w PHP argumenty do funkcji są domyślnie przekazywane przez wartość:

```
WhoIsCool ($nazwa);
print($nazwa);
```

Następnie ta sama funkcja jest wywoływana z przekazaniem argumentu przez zmienną, a wynik działania funkcji jest wyświetlany.

```
WhoIsCool(&$nazwa);
print($nazwa);
```

Aby przekazać do funkcji argument przez zmienną, należy dodać znak `&` przed zmienną przekazywaną do funkcji (jak na przykład `&$name`). Po uruchomieniu skryptu widać, że przy wywołaniu funkcji z przekazaniem argumentu przez wartość, wartość zmiennej łańcuchowej pozostaje niezmieniona (patrz rysunek 3.18). Jednak przy przekazaniu argumentu przez zmienną, jej wartość jest zmieniana.



Rysunek 3.18. *Wyniki przekazania argumentu przez wartość i przez zmienną*

Korzystanie z obiektów

PHP pozwala na rozbudowywanie i wielokrotne wykorzystanie napisanego fragmentu kodu w postaci obiektów. Obiekty PHP umożliwiają dzielenie kodu na segmenty, które w PHP są określane jako *klasy*. Jest to kolejny krok w kierunku strukturyzacji i wielokrotnego wykorzystania kodu, ponieważ umożliwia całkowite oddzielenie różnych części aplikacji. Rozdzielając w ten sposób kod, programista ma pewność, że podczas testowania i uruchamiania segmentu kodu wszystkie potrzebne instrukcje i definicje znajdują się w jednej klasie. Jak, w takim razie, korzystać z obiektów w PHP?

```
<?php

class mojaklasa {
         function WiadomoscPowitalna($argument){
         return "Witaj " . $argument;
        }
}

//tworzenie nowej kopii klasy
$mojobiekt = new mojaklasa();

//wywołanie funkcji WiadomoscPowitalna i zapisanie wyniku wykonania funkcji
$zwrot = $mojobiekt->WiadomoscPowitalna("Andrzej");

//wyświetlenie wyników
print($zwrot);

?>
```

Najpierw należy zdefiniować klasę dla naszego obiektu:

```
class mojaklasa {
         function WiadomoscPowitalna($argument){
         return "Witaj " . $argument;
        }
}
```

Należy tutaj wspomnieć o kilku istotnych kwestiach. Po pierwsze, cały kod dotyczący klasy jest umieszczony wewnątrz nawiasów funkcji klasy. Po drugie, zauważ, że funkcja `WiadomoscPowitalna` użyta w poprzednich przykładach tego rozdziału, ponownie się pojawia. Nie ma obowiązku umieszczania definiowanych funkcji wewnątrz klasy, ale jest to naturalny krok, jeśli dążysz do strukturyzacji kodu. Podobnie jak w poprzednich przykładach, funkcja `WiadomoscPowitalna` pobiera dane poprzez argument, dokleja dane do zmiennej i zwraca zmienną jako wynik działania.

```
$mojobiekt = new mojaklasa();
```

W dalszej części tworzonego kodu możesz wykorzystywać zdefiniowaną wcześniej klasę. W tym celu musisz najpierw stworzyć kopię obiektu i przypisać ją do zmiennej. W jakim celu? Jest to kolejny krok w kierunku wielokrotnego wykorzystania fragmentu kodu. Możesz korzystać z tego samego obiektu, nadając mu różne wartości i uzyskując różne wyniki. Nie możesz jednak używać tego samego obiektu, równocześnie nadając mu różne wartości, dlatego musisz utworzyć kilka kopii tego obiektu i używać tych kopii. W naszym przykładzie obiekt jest wykorzystywany tylko jeden raz, ale zasady składni pozostają niezmienione. Kopia obiektu jest przypisana do zmiennej `$mojobiekt`:

```
$zwrot = $mojobiekt->WiadomoscPowitalna("Andrzej");
```

Teraz możesz korzystać z obiektu. W PHP obowiązuje następująca składnia:

```
KopiaObiektu->NazwaFunkcji
```

W tym przykładzie wywoływana jest funkcja `WiadomoscPowitalna` obiektu `mojaklasa`. Po uruchomieniu skryptu pojawią się wyniki wywołania funkcji wewnątrz obiektu. Ten przykład jest raczej prosty. Aby przedstawić zalety wynikające z wykorzystania obiektów w PHP, musimy go rozbudować:

```
<?php

class mojaklasa {
         var $wyswietlananazwa = "Wszyscy";
         function DodajNazwe($argument){
         $this->wyswietlananazwa = $argument;
        }

         function WiadomoscPowitalna(){
         return "Witaj " . $this->wyswietlananazwa;
        }
}

//tworzenie pierwszej kopii klasy
$mojobiekt1 = new mojaklasa();

//wywołanie funkcji WiadomoscPowitalna i zapisanie wyniku wykonania funkcji
$zwrot = $mojobiekt1->WiadomoscPowitalna();

//wyświetlenie wyników
print($zwrot);
print("<BR>");

//tworzenie drugiej kopii klasy
$mojobiekt2 = new mojaklasa();
$mojobiekt2->DodajNazwe("Andrzej");
$zwrot = $mojobiekt2->WiadomoscPowitalna();

//wyświetlenie wyników
print($zwrot);

?>
```

W tym przypadku klasa składa się z kilku elementów:

```
class mojaklasa {
         var $wyswietlananazwa = "Wszyscy";
         function DodajNazwe($argument){
         $this->wyswietlananazwa = $argument
        }

         function WiadomoscPowitalna(){
         return "Witaj" . $this->wyswietlananazwa;
        }
}
```

Najpierw zdefiniowano zmienną wewnątrz klasy. Na niej operuje kod znajdujący się wewnątrz definicji klasy i dlatego jest określana jako zmienna klasy.

```
var $wyswietlananazwa = "Wszyscy";
```

Musisz zadeklarować zmienną klasy przy użyciu instrukcji `var`. Dzięki temu PHP „wie", co jest zmienną, a co funkcją.

W naszym przykładzie obiekt zawiera dwie funkcje:

```
 function DodajNazwe($argument){
 $this->wyswietlananazwa = $argument;
}

 function WiadomoscPowitalna(){
 return "Witaj" . $this->wyswietlananazwa;
}
```

Pierwsza z nich pobiera argument i przypisuje zmiennej klasy wartość tego argumentu:

```
 function DodajNazwe($argument){
 $this->wyswietlananazwa = $argument;
}
```

Zwróć uwagę na składnię przypisania zmiennej klasy:

```
$this->wyswietlananazwa = $argument
```

Odwołanie dotyczy zmiennej `wyswietlananazwa` zdefiniowanej w bieżącej klasie. Użycie następującej składni gwarantuje, że odwołanie dotyczy zmiennej klasy i PHP nie pomyli jej z inną zmienną znajdującą się poza klasą.

```
 funkcja WiadomoscPowitalna(){
 return "Witaj" . $this->wyswietlananazwa;
}
```

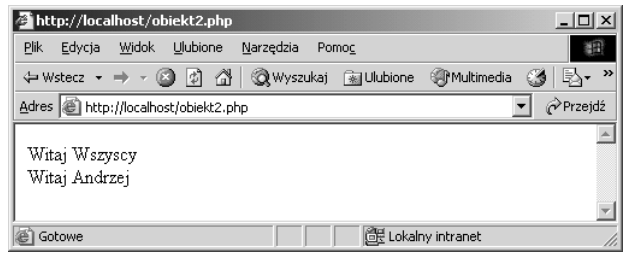
Druga funkcja przekazuje wartość zmiennej klasy poza klasę. Kod wywołujący klasę spełnia dwa zadania. Najpierw tworzy kopię klasy i wyświetla wartość zmiennej klasy:

```
$mojobiekt1 = new mojaklasa();
$zwrot = $mojobiekt1->WiadomoscPowitalna();
print($zwrot);
```

Następnie tworzy kolejną kopię klasy i przekazuje wartość funkcji `DodajNazwe` w celu przypisania wartości zmiennej klasy. Ostatnim krokiem jest wyświetlenie nowej wartości zmiennej klasy:

```
$mojobiekt2 = new mojaklasa();
$mojobiekt2->DodajNazwe("Andrzej");
$zwrot = $mojobiekt2->WiadomoscPowitalna();

//wyświetlenie wyników
print($zwrot);
```

Po uruchomieniu skryptu zobaczysz wyniki wykonania dwóch części skryptu; w pierwszej części wartość zmiennej klasy pozostawała niezmieniona, w drugiej części widoczna jest nowa wartość zmiennej klasy, jak pokazano na rysunku 3.19.



Rysunek 3.19. *Zmiana wartości zmiennej klasy*

Podsumowanie

W tym rozdziale pokrótce przyjrzeliśmy się różnym rodzajom edytorów w systemie Windows, za pomocą których możesz tworzyć skrypty PHP. Omówiliśmy również podstawy języka PHP, takie jak składnia, zmienne, tablice, operatory logiczne i pętle, funkcje i obiekty. Rozdział 4., „PHP i pliki", poświęcony jest współpracy PHP z plikami oraz systemem plików Windows.