SYSTEMY ZARZĄDZANIA TREŚCIĄ CMS

LABORATORIUM 2

Po zakończeniu pracy zachowaj prokejt na dysku sieciowym lub w innym bezpiecznym miejscu

Dostęp do zmiennych formularza

Podstawowym celem stosowania formularza jest przyjęcie zamówienia klienta. W PHP bardzo łatwo jest uzyskać szczegółowe informacje o tym zamówieniu, jednak konkretna metoda zależy od wykorzystywanej wersji PHP oraz ustawień w pliku *php.ini*.

Zmienne formularza

W skrypcie PHP można uzyskać dostęp do każdego pola formularza, traktując je jako zmienne o nazwach identycznych jak nazwy pól. Nazwy zmiennych PHP można rozpoznać po tym, że zaczynają się od znaku `$`. (Zapominanie o znaku dolara jest częstym błędem programistycznym).

Zależnie od wersji PHP i ustawień można stosować kilka różnych sposobów uzyskiwania dostępu do danych za pośrednictwem zmiennych. W ostatnich wersjach PHP wszystkie z tych metod oprócz jednej zostały uznane za przestarzałe, dlatego osoby, które używały PHP w przeszłości, muszą zwrócić na to uwagę.

Dostęp do zawartości pola `iloscopon` można uzyskać w następujący sposób:

```
$_POST['iloscopon']
```

`$_POST` jest tablicą zawierającą dane przesłane żądaniem HTTP `POST`, czyli wysłane z formularza używającego metody `POST`. Istnieją trzy takie tablice, które mogą zawierać dane przesyłane z formularzy: `$_POST`, `$_GET` oraz `$_REQUEST`. Jedna z tablic — `$_GET` lub `$_POST` — zawiera wszystkie szczegółowe dane o wszystkich zmiennych formularza. To, która z tych tablic zostanie użyta, zależy od tego, czy metodą wykorzystywaną w formularzu była metoda `GET`, czy `POST`. Ponadto wszystkie dane przesłane którąkolwiek z tych dwóch metod są dostępne w tablicy `$_REQUEST`.

Jeśli formularz zostanie przesłany metodą `POST`, to dane wpisane w polu `iloscopon` będą przechowywane w `$_POST['iloscopon']`. Jeżeli natomiast zostanie on przesłany metodą `GET`, dane będą się znajdować w `$_GET['iloscopon']`.

Tablice te są nowymi tak zwanymi tablicami *superglobalnymi*. Wrócimy do nich, gdy będziemy omawiać zasięg zmiennych.

Zajmijmy się przykładem, w którym utworzone zostaną łatwiejsze w użyciu kopie zmiennych.

W celu skopiowania wartości jednej zmiennej do drugiej zmiennej należy zastosować operator przypisania, którym w PHP jest znak równości (=). Poniższy wiersz kodu utworzy nową zmienną o nazwie `$iloscopon` i skopiuje zawartość `$_POST['iloscopon']` do nowej zmiennej:

```
$iloscopon = $_POST['iloscopon'];
```

Umieśćcie ten blok kodu na początku skryptu przetwarzającego formę.

```
<?php
  //utwórz krótkie nazwy zmiennych
  $iloscopon = $_POST['iloscopon'];
  $iloscoleju = $_POST['iloscoleju'];
  $iloscswiec = $_POST['iloscswiec'];
?>
```

Kod ten spowoduje utworzenie trzech nowych zmiennych: `$iloscopon`, `$iloscoleju` oraz `$iloscswiec`, a następnie przypisze im dane wysłane z formy metodą `POST`.

Aby wyświetlić wartości tych zmiennych na stronie w przeglądarce, należy użyć następującego zapisu:

```
echo $iloscopon.' opon<br />';.
```

Takie rozwiązanie nie jest jednak zalecane.

Na tym etapie nie sprawdzamy jeszcze zawartości zmiennych, aby upewniać się, że w każdym polu formularza wpisano sensowne dane. Można samodzielnie wpisać jakieś zupełnie nieprawidłowe dane i sprawdzić, co się stanie.

Pobranie danych bezpośrednio od użytkownika i wyświetlenie ich w przeglądarce JEST z punktu widzenia bezpieczeństwa działaniem wyjątkowo ryzykownym, którego w żadnym razie nie zalecamy. Dane wejściowe trzeba najpierw przefiltrować.

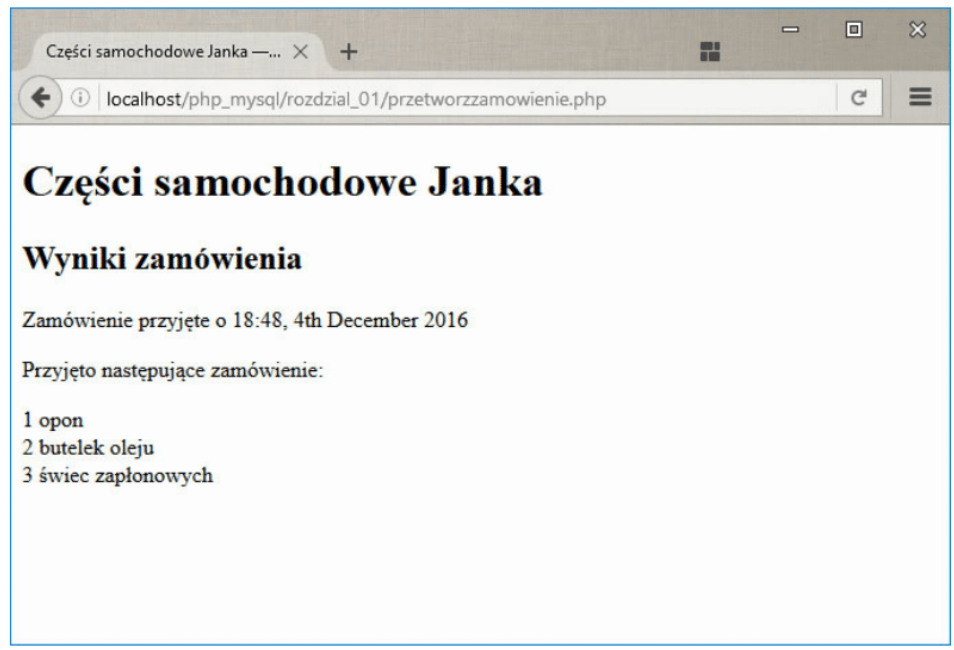
Na razie wystarczy wiedzieć, że dane wpisane przez użytkownika można wyświetlić w przeglądarce, przekazując je wcześniej do funkcji `htmlspecialcharc()`. Na przykład w poniższym przykładzie moglibyśmy to zrobić w następujący sposób:

```
echo htmlspecialchars($iloscopon).' opon<br />';
```

Aby skrypt zaczął dawać widoczne efekty, należy umieścić na jego końcu poniższy fragment kodu PHP:

```
echo '<p>Przyjęto następujące zamówienie:</p>';
echo htmlspecialchars($iloscopon).' opon<br />';
echo htmlspecialchars($iloscoleju).' butelek oleju<br />';
echo htmlspecialchars($iloscswiec).' świec zapłonowych<br />';
```

Po odświeżeniu zawartości okna przeglądarki skrypt powinien dać rezultat podobny do przedstawionego na rysunku 1.4. Konkretne wartości zależą oczywiście od danych wpisanych do formularza.



Rysunek 1.4. *W skrypcie przetworzzamowienie.php łatwo uzyskać dostęp do zmiennych formularza wpisanych przez użytkownika*

Łączenie łańcuchów znaków

W przykładowym skrypcie instrukcja `echo` została użyta do wyświetlenia informacji wpisanych w pola formularza, po czym następował tekst wyjaśniający. Po bliższym przyjrzeniu się instrukcjom `echo` można zauważyć, że pomiędzy nazwą zmiennej i następującym po niej tekstem została umieszczona kropka (`.`), na przykład:

```
echo htmlspecialchars($iloscopon).' opon<br />';
```

Kropka, nazywana operatorem łączenia łańcuchów znaków, jest używana do dodawania do siebie łańcuchów (fragmentów tekstu). Często stosuje się ją przy wyświetlaniu wyników w przeglądarce z użyciem instrukcji `echo`. W ten sposób można uniknąć konieczności wpisywania kilku takich instrukcji.

Każdą zmienną niebędącą tablicą można również umieścić w łańcuchu znaków otoczonym cudzysłowami w celu jej wyświetlenia

```
$iloscopon = htmlspecialchars($iloscopon);
echo "$iloscopon opon<br />";
```

Wyrażenie to jest równoznaczne z pierwszą instrukcją przedstawioną w tym punkcie. Każdy z tych formatów jest uznawany, a wybór jednego z nich — całkowicie dowolny. Proces taki, czyli zastępowanie wewnątrz łańcucha znaków zmiennej przez jej zawartość, nazywa się *interpolacją*.

Zwróćcie uwagę, że interpolacja jest cechą wyłącznie łańcuchów znaków otoczonych cudzysłowami. Nie można w taki sam sposób umieszczać nazw zmiennych w łańcuchach znaków otoczonych apostrofami. Uruchomienie poniższego wiersza kodu:

```
echo '$iloscopon opon<br />';
```

spowoduje wysłanie do przeglądarki echo `'$iloscopon opon<br />'`. W przypadku zastosowania cudzysłowów nazwa zmiennej zostanie zastąpiona jej wartością. W apostrofach nazwa zmiennej lub każdy inny tekst zostaną wysłane w postaci niezmienionej.

Zmienne i łańcuchy znaków

W każdej z powyższych instrukcji `echo` należy odróżnić zmienną od łańcucha znaków. Zmienne to symbole danych, łańcuchy są natomiast danymi samymi w sobie. Stosując fragment danych w programie takim jak powyższy, w celu odróżnienia od zmiennej nazywamy go *łańcuchem znaków* (w skrócie: łańcuchem). `$iloscopon` to zmienna, symbol reprezentujący dane wprowadzone przez klienta. Z kolei `' opon<br />'` to łańcuch znaków, bezpośrednio posiadający wartość. Istnieje wyjątek od tej reguły. W drugim z powyższych przykładów interpreter PHP zamienił nazwę znajdującej się w łańcuchu znaków zmiennej `$iloscopon` na wartość przechowywaną w zmiennej.

W PHP istnieją dwa typy łańcuchów znaków: zapisane w cudzysłowach i zapisane w apostrofach. PHP próbuje interpretować łańcuchy (czyli wyszukiwać nazwy zmiennych i zamieniać je na ich wartości) w cudzysłowach, dając rezultat, który można zobaczyć powyżej. Łańcuchy w apostrofach zostaną potraktowane jako prawdziwe surowe dane.

Istnieje również trzeci sposób definiowania łańcuchów. Jest to składnia *heredoc* (<<<), znana już użytkownikom języka Perl. Pozwala ona na zgrabne definiowanie długich łańcuchów znaków poprzez używanie znacznika zamykającego, który oznacza koniec łańcucha. Poniższy przykładowy kod tworzy łańcuch złożony z trzech wierszy, a następnie go wyświetla:

```
echo <<<koniec
  wiersz 1
  wiersz 2
  wiersz 3
koniec
```

Znacznik `koniec` ma charakter całkowicie arbitralny. Wystarczy jedynie zagwarantować, że nie pojawi się on w tekście. Aby zamknąć łańcuch *heredoc*, należy umieścić znacznik zamykający na początku wiersza.

Łańcuchy *heredoc* podlegają interpolacji, podobnie jak łańcuchy w cudzysłowach.

Identyfikatory

Identyfikatory to nazwy zmiennych. Istnieje kilka prostych zasad dotyczących identyfikatorów:

- Identyfikatory, które mogą mieć dowolną długość, składają się z liter, cyfr, znaków podkreślenia i znaków dolara.
- Identyfikatory nie mogą rozpoczynać się cyfrą.
- W PHP rozróżniana jest wielkość liter identyfikatorów; `$iloscopon` to nie to samo co `$IloscOpon`. Próba zamiennego ich stosowania to częsty błąd programistyczny. Wyjątkiem są funkcje wbudowane w PHP — ich nazwy mogą być używane w każdej formie.
- Identyfikatory zmiennych mogą mieć nazwę identyczną z wbudowaną funkcją. Należy jednak unikać tego kłopotliwego rozwiązania. Ponadto nie można utworzyć funkcji o identyfikatorze takim samym jak funkcja wbudowana.

Oprócz zmiennych przekazanych z formularza HTML można zadeklarować i używać także własnych.

Jedną z cech PHP jest to, że nie trzeba deklarować zmiennych przed ich użyciem. Zmienna zostanie utworzona po pierwszym przypisaniu jej wartości.

Wartości są przyporządkowane zmiennym za pomocą operatora przypisania: =. Na stronie Janka należy obliczyć całkowitą liczbę części zamówionych przez klienta oraz ich wartość. W celu przechowania tych wartości trzeba utworzyć dwie zmienne. Eleganckim sposobem jest zainicjowanie każdej z nich poprzez przypisanie im zera. W tym celu na końcu skryptu PHP należy dodać następujące wiersze:

```
$ilosc = 0;
$wartosc = 0.00;
```

Każdy z tych dwóch wierszy tworzy zmienną i przypisuje jej dosłowną wartość. Można również przyporządkować zmiennym wartości innych zmiennych, jak w poniższym przykładzie:

```
$ilosc = 0;
$wartosc = $ilosc;
```

Typy zmiennych

Typ zmiennej odnosi się do rodzaju danych w niej zapisanych. PHP udostępnia cały zestaw typów danych. Dane różnego rodzaju mogą być przechowywane przy użyciu różnych typów danych.

Typy danych w PHP

PHP rozpoznaje następujące typy danych:

- `Integer` — stosowany dla liczb całkowitych;
- `Float` (zwany również `Double`) — stosowany dla liczb rzeczywistych;
- `String` — stosowany dla łańcucha znaków;
- `Boolean` — stosowany w przypadku wartości `true` lub `false`;
- `Array` — stosowany do przechowywania wielu danych
- `Object` — stosowany do przechowywania obiektów

Dostępne są również trzy specjalne typy danych: `NULL`, `resource` oraz `callable`.

Zmienne, którym nie nadano wartości, które nie są ustawione lub którym nadano wartość `NULL`, są typu `NULL`. Pewne funkcje wbudowane (na przykład funkcje obsługi baz danych) zwracają zmienne typu `resource`.

Reprezentują one zasoby zewnętrzne (na przykład połączenie z bazą danych). Niemal na pewno nigdy nie będziecie operować bezpośrednio na zmiennych typu `resource`, lecz są one często zwracane przez funkcje i muszą być przekazywane do innych funkcji jako ich parametry.

Dane typu `callable` są w zasadzie funkcjami, które mogą być przekazywane do innych funkcji.

Siła typu

Typy w PHP są słabo zaznaczone. W większości języków programowania, na przykład w C, zmienne mogą przechowywać tylko jeden typ danych, który musi zostać zadeklarowany przed ich użyciem. W PHP typ zmiennej jest określany przez dane do niej przypisane.

Na przykład kiedy zmienne `$ilosc` i `$wartosc` zostały zadeklarowane powyżej, nastąpiło również określenie ich początkowego typu:

```
$ilosc = 0;
$wartosc = 0.00;
```

Ponieważ `$ilosc` przypisano 0, liczbę całkowitą, jest ona w tym momencie zmienną typu `integer`. Podobnie `$wartosc` jest zmienną typu `float`.

Co dziwniejsze, do powyższego skryptu można dodać następujące wiersze:

```
$wartosc = 'Cześć';
```

Zmienna `$wartosc` jest teraz typu `string`. PHP dostosowuje typ zmiennej do danych przechowywanych w niej w danym momencie.

Zdolność do niewidocznej zmiany typów „w locie” może okazać się niezwykle przydatna. Należy pamiętać, że PHP „wie”, jaki typ danych zostaje umieszczony w zmiennej. Zwróci on dane o tym samym typie przy odzyskiwaniu ich ze zmiennej.

Rzutowanie typu

Stosując rzutowanie typu, można udawać, że zmienna bądź wartość są innego typu niż w rzeczywistości. Sposób ten działa identycznie jak w C. Należy po prostu podać w nawiasach przed właściwą zmienną nazwę pożądanego typu.

Można na przykład zadeklarować dwie zmienne z poprzedniego punktu za pomocą rzutowania.

```
$ilosc = 0;
$wartosc = (float)$ilosc;
```

Drugi wiersz oznacza: „Weź wartość zapisaną w `$ilosc`, zinterpretuj ją jako `float` i zapisz w `$wartosc`". Zmienna `$wartosc` jest w tym wypadku typu `float`. Rzutowane zmienne nie zmieniają typu, a zatem `$ilosc` pozostaje typem `integer`.

Można także użyć wbudowanej funkcji służącej do sprawdzania i ustawiania konkretnego typu, która zostanie przedstawiona w dalszej części tego rozdziału.

Zmienne zmiennych

PHP dostarcza zmiennych jeszcze jednego typu, zwanych zmiennymi zmiennych. Pozwalają one na dynamiczną zmianę nazwy zmiennej.

Jak widać, PHP daje dużo swobody w tej dziedzinie. Wszystkie języki pozwalają na zmianę wartości zmiennej, lecz niewiele umożliwia zmianę jej typu, a jeszcze mniej — zmianę nazwy.

Zmienna zmiennej działa poprzez użycie wartości jednej zmiennej jako nazwy drugiej. A oto przykład:

```
$nazwazmiennej = "iloscopon";
```

Można teraz użyć zmiennej `$$nazwazmiennej` zamiast `$iloscopon`. Przykładowo można nadać wartość `$iloscopon` w następujący sposób:

```
$$nazwazmiennej = 5;
```

Oznacza to dokładnie to samo, co:

```
$iloscopon = 5;
```

Metoda ta wydaje się nieco niejasna, lecz w dalszej części książki zostanie omówiona dokładniej. Zamiast wypisywania i oddzielnego używania każdej ze zmiennych formularza można użyć pętli i zmiennej do automatycznego przetworzenia ich wszystkich.

Deklarowanie i używanie stałych

Jak już wspomnieliśmy, możliwa jest zmiana wartości przypisanej do zmiennej. Można zadeklarować również stałe. Podobnie jak zmienna, stała przechowuje pewną wartość, lecz jest ona przypisana jednorazowo i nie może być zmieniona w żadnym innym miejscu skryptu.

W przykładowej aplikacji można przechować ceny każdej sprzedawanej części jako stałe. Definiuje się je, stosując funkcję `define`:

```
define("CENAOPON", 100);
define("CENAOLEJU", 10);
define("CENASWIEC", 4);
```

Powyższe wiersze należy dodać do kodu skryptu. W ten sposób w przykładowej aplikacji będą istnieć teraz trzy stałe, które będą mogły zostać użyte do obliczenia wartości zamówienia klienta.

Łatwo zauważyć, że nazwy stałych są zapisane w całości wielkimi literami. Jest to konwencja zapożyczona z C, która pozwala na łatwe odróżnienie zmiennych od stałych. Nie ma ona charakteru obligatoryjnego, lecz jej stosowanie czyni kod łatwiejszym do odczytania i utrzymania.

Jedna z ważnych różnic pomiędzy stałymi i zmiennymi polega na tym, że przy odwołaniu do stałej nie umieszcza się przed jej nazwą znaku dolara, lecz jedynie samą nazwę. Na przykład aby użyć jednej z powyższych stałych, można napisać:

```
echo CENAOPON;
```

Oprócz stałych zdefiniowanych przez użytkownika PHP tworzy dużą liczbę własnych stałych. Prostym sposobem na ich obejrzenie jest użycie polecenia `phpinfo()`:

```
phpinfo();
```

Polecenie to da wynik w postaci listy predefiniowanych przez PHP zmiennych i stałych, jak i innych pożytecznych informacji.

Jeszcze jedna różnica między zmiennymi i stałymi polega na tym, że stałe mogą przechowywać jedynie dane typów `boolean`, `integer`, `float` lub `string`. Typy te ogólnie nazywa się mianem wartości skalarnych.

Zasięg zmiennych

Termin *zasięg* odnosi się do części skryptu, w której widoczna jest dana zmienna. Istnieje sześć podstawowych typów zasięgów w PHP:

- Wbudowane zmienne superglobalne są widoczne w całym skrypcie.
- Stałe, po ich zadeklarowaniu, są zawsze widoczne globalnie. Oznacza to, że można ich używać zarówno wewnątrz, jak i na zewnątrz funkcji.
- Zmienne globalne zadeklarowane w skrypcie są widoczne w całym skrypcie, *ale nie wewnątrz funkcji*.
- Zmienne używane w obrębie funkcji, które są deklarowane jako globalne, odnoszą się do zmiennej globalnej o tej samej nazwie.
- Zmienne utworzone wewnątrz funkcji i zadeklarowane jako statyczne są niewidoczne na zewnątrz funkcji, lecz zachowują swą wartość w czasie między wykonaniem tej i następnej funkcji
- Zmienne utworzone wewnątrz funkcji są jej zmiennymi lokalnymi i kończą swój żywot w momencie zakończenia wykonywania tej funkcji.

Tablice `$_GET` i `$_POST`, a także niektóre inne specjalne zmienne, posiadają własne zasady rządzące ich zasięgiem. Są to tak zwane zmienne *superglobalne* i mogą być widoczne wszędzie — zarówno wewnątrz funkcji, jak i poza nimi.

Kompletna lista zmiennych superglobalnych przedstawia się następująco:

- `$GLOBALS` — tablica wszystkich zmiennych globalnych (podobnie jak słowo kluczowe `global` pozwala ona wewnątrz funkcji na dostęp do zmiennych globalnych, na przykład jako `$GLOBALS['mojazmienna']`),
- `$_SERVER` — tablica zmiennych środowiskowych serwera,
- `$_GET` — tablica zmiennych przekazanych do skryptu metodą `GET`,
- `$_POST` — tablica zmiennych przekazanych do skryptu metodą `POST`,
- `$_COOKIE` — tablica zmiennych *cookie*,
- `$_FILES` — tablica zmiennych związanych z ładowaniem pliku,
- `$_ENV` — tablica zmiennych środowiskowych,
- `$_REQUEST` — tablica wszystkich zmiennych wprowadzonych przez użytkownika, właczając w to zawartość wprowadzonych zmiennych znajdujących się w `$_GET`, `$_POST` i `$_COOKIE` (już bez zmiennej `$_FILES`).
- `$_SESSION` — tablica zmiennych sesji.

Używanie operatorów

Operatory to symbole używane do manipulowania wartościami i zmiennymi poprzez wykonywanie na nich operacji. Niektóre będą potrzebne do obliczenia wartości i podatku od zamówienia klienta.

Dwa z nich zostały już opisane — operator przypisania (=) i operator łączenia łańcuchów znaków (.).

Ogólnie rzecz biorąc, operatory działają na jednym, dwóch bądź trzech argumentach, przy czym większość — na dwóch. Na przykład operator przypisania ingeruje w dwa — miejsce przechowywania po lewej stronie symbolu = i wyrażenie po prawej stronie. Argumenty te są nazywane *operandami*, to znaczy wartościami, na których się operuje.

Operatory arytmetyczne

Operatory arytmetyczne działają na bardzo prostej zasadzie — są zwykłymi operatorami matematycznymi. Zostały one przedstawione w tabeli 1.1.

Dla każdego z tych operatorów można zapisać wynik działania. Na przykład:

```
$wynik = $a + $b;
```

Dodawanie i odejmowanie działa tak, jak należy się tego spodziewać. Wynikiem działań jest, odpowiednio, dodawanie bądź odejmowanie wartości zapisanych w zmiennych `$a` i `$b`.

Tabela 1.1. *Operatory arytmetyczne w PHP*

Operator	Nazwa	Przykład
+	Suma	`$a + $b`
-	Różnica	`$a - $b`
*	Iloczyn	`$a * $b`
/	Iloraz	`$a / $b`
%	Modulo	`$a % $b`

Można również użyć symbolu różnicy (-) jako operatora jednoargumentowego — tzn. takiego, który posługuje się tylko jednym argumentem lub operandem — w celu oznaczenia liczb ujemnych. Na przykład:

```
$a = -1;
```

Mnożenie i dzielenie również działa w przeważającej liczbie wypadków tak, jak należy się tego spodziewać. Warto jedynie zauważyć symbol gwiazdki jako operator iloczynu, zamiast symbolu zwykle stosowanego, oraz symbol ukośnika jako operator ilorazu, również w miejsce symbolu standardowego.

Operator reszty zwraca pozostałość po podzieleniu zmiennej `$a` przez zmienną `$b`. Należy rozważyć następujący fragment kodu:

```
$a = 27;
$b = 10;
$wynik = $a%$b;
```

Wartość zachowana w zmiennej `$wynik` jest resztą z dzielenia 27 przez 10; wynosi ona 7.

Należy zauważyć, że operatory arytmetyczne są zazwyczaj używane ze zmiennymi typu `integer` bądź `double`. Przy zastosowaniu ich do zmiennej typu `string` PHP spróbuje przekonwertować łańcuch znaków na liczbę. Jeżeli zawiera on wyrażenie `e` lub `E`, zostanie on odczytany w notacji inżynierskiej i przekonwertowany na `float`, w przeciwnym wypadku — na `integer`. PHP będzie szukał cyfr na początku łańcucha i użyje ich jako wartości, jeżeli zaś nie znajdzie żadnych, wartość łańcucha wyniesie zero.

Operatory łańcuchowe

Powyżej omówiliśmy przykład zastosowania jedynego operatora łańcuchowego. Operatora łączenia (konkatenacji) używa się do dodawania łańcuchów oraz tworzenia i przechowywania wyniku w podobny sposób, jak w przypadku operatora sumy, aby dodać dwie cyfry.

```
$a = "Części samochodowe ";
$b = 'Janka';
$wynik = $a.$b;
```

Zmienna `$wynik` zawiera teraz łańcuch `"Części samochodowe Janka"`.

Operatory przypisania

Opisany już został podstawowy operator przypisania (=). Symbol = zawsze traktuje się jako operator przypisania i należy go odczytywać: „jest przypisana wartość”. Na przykład:

```
$ilosc = 0;
```

Powyższa instrukcja powinna być przeczytana następująco: „zmiennej `$ilosc` jest przypisana wartość zero”. Powody tego zostaną podane w dalszej części, podczas opisywania operatorów porównania.

Wartości zwracane podczas przypisywania

Zastosowanie operatora przypisania zwraca wartość ogólną, podobnie jak w wypadku innych operatorów. Dla kodu

```
$a + $b
```

wartością wyrażenia jest wynik dodawania zmiennych `$a` i `$b`. Podobnie, dla

```
$a = 0;
```

wartość wyrażenia wynosi zero.

Technika ta pozwala na tworzenie wyrażeń takich jak:

```
$b = 6 + ($a = 5);
```

Działanie to spowoduje przypisanie zmiennej `$b` wartości 11. Ogólna zasada działania operatorów przypisania polega na tym, że wartością całego wyrażenia jest wartość przypisana lewemu operandowi.

Jak pokazano powyżej, podczas obliczania wartości wyrażenia można, identycznie jak w matematyce, używać nawiasów w celu ustanowienia kolejności wykonywania poszczególnych części wyrażenia.

Łączone operatory przypisania

Oprócz powyższych prostych przypisań istnieje zbiór łączonych operatorów przypisania. Każdy z nich to skrócony sposób zapisu operacji przeprowadzonej na zmiennej i przypisanego do niej wyniku tej operacji. Na przykład zapis:

```
$a += 5;
```

jest równoznaczny z:

```
$a = $a + 5;
```

Istnieją łączone operatory przypisania dla każdego operatora arytmetycznego i dla operatora łączenia łańcuchów. Zestawienie wszystkich łączonych operatorów przypisania wraz z ich wynikami jest przedstawione w tabeli 1.2.

Tabela 1.2. *Łączone operatory przypisania w PHP*

Operator	Przykład użycia	Równoznaczne z
`+=`	`$a += $b`	`$a = $a + $b`
`-=`	`$a -= $b`	`$a = $a - $b`
`*=`	`$a *= $b`	`$a = $a * $b`
`/=`	`$a /= $b`	`$a = $a / $b`
`%=`	`$a %= $b`	`$a = $a % $b`
`.=`	`$a .= $b`	`$a = $a . $b`

Pre- i postinkrementacja oraz dekrementacja

Operatory pre- i postinkrementacji (++) oraz dekrementacji (--) są podobne do operatorów += i -=, występują jednak pewne różnice.

Wszystkie operatory inkrementacji wywołują dwa efekty — powiększają i przypisują wartość. Należy rozważyć następujący zapis:

```
$a=4;
echo ++$a;
```

Drugi wiersz zawiera operator preinkrementacji — nazywany tak, ponieważ symbol ++ pojawia się przed znakiem $a. Operator ten, po pierwsze, zwiększa $a o 1, a po drugie — zwraca powiększoną wartość. W tym przypadku wartość zmiennej $a rośnie do 5, po czym zostaje ona zwrócona i wydrukowana. Wartość całego wyrażenia wynosi 5 (należy zauważyć, że nie zostaje zwrócona wartość $a + 1, lecz na stałe zwiększona wartość $a).

Natomiast gdy symbol ++ występuje po $a, użyty zostanie operator postinkrementacji, co daje odmienny rezultat. Rozważmy następujący zapis:

```
$a=4;
echo $a++;
```

W tym przypadku efekty zostają odwrócone. Po pierwsze, zwrócona i wydrukowana zostaje wartość $a, która później ulega powiększeniu. Wartość całego wyrażenia wynosi 4, i właśnie ona zostaje zwrócona. Natomiast po wykonaniu tej instrukcji wartość $a zostanie zwiększona do 5.

Jak łatwo się domyślić, działanie operatora -- (dekrementacji) jest podobne, lecz wartość $a zostaje zmniejszona, a nie zwiększona.

Operatory referencji

Operator referencji (&, czyli ampersand), może być stosowany w połączeniu z przypisaniem. W większości przypadków, kiedy jedna zmienna jest przypisywana do drugiej, zostaje utworzona kopia pierwszej zmiennej, zapisana w drugiej. Na przykład:

```
$a = 5;
$b = $a;
```

Powyższe wiersze kodu tworzą kopię wartości zmiennej $a i zapisują ją w $b. Jeżeli później zostanie zmieniona wartość $a, $b nie ulegnie modyfikacji:

```
$a = 7; // $b łańcuch będzie miało wartość 5
```

Można uniknąć tworzenia kopii, stosując operator referencji, &. Na przykład:

```
$a = 5;
$b = &$a;
$a = 7; // Zarówno $a jak i $b mają wartość 7
```

Odwołania mogą sprawiać nieco kłopotów. Należy pamiętać, że odwołanie jest rodzajem nazwy zastępczej (określanej potocznie mianem *aliasu*), a nie wskaźnikiem. Z tego względu $a i $b wskazują na ten sam fragment pamięci. Można to zmienić, usuwając jedną z tych zmiennych w następujący sposób:

```
unset($a);
```

Usunięcie jej nie spowoduje zmiany wartości zmiennej $b (7), lecz zniszczy połączenie między zmienną $a i wartością 7 przechowywaną w pamięci.

Operatory porównań

Operatorów porównań używa się w celu porównania dwóch wartości. Wyrażenia, w których występują te operatory, zwracają jedną z dwóch wartości logicznych, `true` lub `false`, zależnie od wyniku porównania.

Operator równości

Operator równości, symbol == (dwa znaki równości) pozwala na sprawdzenie równości dwóch wartości. Na przykład można zastosować wyrażenie:

```
$a == $b
```

aby sprawdzić, czy wartości zmiennych `$a` i `$b` są sobie równe. Wynik zwrócony przez to wyrażenie będzie wynosił `true`, jeżeli są równe, lub `false`, jeżeli nie.

Łatwo jest pomylić ten znak z symbolem =, operatorem przypisań. Takie wyrażenie zadziała bez zwrócenia błędu, lecz raczej nie da pożądanego wyniku. Ogólnie rzecz biorąc, wartości niezerowe są przyjmowane jako `true`, a zerowe jako `false`. Przy założeniu, że zmienne zostały zainicjowane w następujący sposób:

```
$a = 5;
$b = 7;
```

Testowanie za pomocą wyrażenia `$a = $b` zwróci wynik `true`. Dzieje się tak dlatego, że wartość `$a = $b` jest wartością przypisaną do lewej strony tego wyrażenia, która w tym wypadku wynosi 7. Jest to wartość niezerowa, tak więc wyrażenie zostaje przyjęte jako `true`. Jeżeli założeniem było przetestowanie `$a == $b`, którego wynikiem jest `false`, został wprowadzony do kodu błąd logiczny, niekiedy bardzo trudny do odnalezienia. Należy zawsze sprawdzać, czy zastosowany został właściwy operator.

Użycie operatora przypisania zamiast operatora porównania jest błędem, który łatwo popełnić i który zdarza się niejednokrotnie w karierze każdego programisty.

Inne operatory porównań

PHP rozpoznaje również kilka innych operatorów porównań; wszystkie zostały przedstawione w tabeli 1.3. Operatorem godnym szczególnej uwagi jest operator identyczności (===), który zwraca wartość `true` tylko wtedy, gdy dwa operandy są równe i należą do tego samego typu. Na przykład wyrażenie `0=='0'` będzie miało wartość `true`, lecz `0==='0'` będzie już równe `false`, ponieważ pierwsze zero jest typu `integer`, a drugie jest łańcuchem znaków `string`.

Operatory logiczne

Operatory logiczne stosuje się w celu uzyskania wyników operacji logicznych. Aby na przykład ustalić, czy wartość zmiennej `$a` mieści się w zakresie od 0 do 100, należy sprawdzić warunki `$a >= 0` i `$a <= 100`, stosując operator `AND`:

```
$a >= 0 && $a <= 100
```

PHP rozpoznaje warunki logiczne `AND`, `OR`, `XOR` (wyłączne `OR`) i `NOT`.

Wszystkie operatory logiczne zostały przedstawione w tabeli 1.4.

Tabela 1.3. *Operatory porównań w PHP*

Operator	Nazwa	Przykład użycia
==	Równość	$a == $b
===	Identyczność	$a === $b
!=	Nierówność	$a != $b
!==	Nieidentyczność	$a !== $b
<>	Nierówność (operator porównania)	$a <> $b
<	Mniejszość	$a < $b
>	Większość (operator porównania)	$a > $b
<=	Mniejszość lub równość	$a <= $b
>=	Większość lub równość	$a >= $b

Tabela 1.4. *Operatory logiczne w PHP*

Operator	Nazwa	Przykład użycia	Wynik
!	NOT	!$b	Zwraca true, jeżeli $b wynosi false, i vice versa
&&	AND	$a && $b	Zwraca true, jeżeli zarówno $a i $b są true, w przeciwnym wypadku false
||	OR	$a || $b	Zwraca true, jeżeli $a lub $b, lub oba są true, w przeciwnym wypadku false
and	AND	$a and $b	Tak samo jak &&, lecz z mniejszym pierwszeństwem
or	OR	$a or $b	Tak samo jak ||, lecz z mniejszym pierwszeństwem
xor	XOR	$a xor $b	Zwraca true, jeżeli $a albo $b wynosi true, oraz false, jeżeli obie zmienne mają wartość true albo obie mają wartość false.

Operatory bitowe

Operatory bitowe pozwalają na traktowanie zmiennej typu integer jako ciągu bitów użytych do jej przedstawienia. W PHP operatory bitowe nie mają wprawdzie zbyt szerokiego zastosowania, lecz mimo to ich zestawienie przedstawione jest w tabeli 1.5.

Pozostałe operatory

Oprócz operatorów opisanych powyżej istnieje jeszcze kilka innych.

Przecinek (,), jest stosowany do oddzielania argumentów funkcji i innych list składników.

Dwa operatory specjalne, new i ->, są używane odpowiednio do tworzenia obiektów klas i dostępu do składowych klasy. Zostaną one opisane w rozdziale 6.

Istnieje poza tym kilka innych typów operatorów, które zostaną krótko scharakteryzowane poniżej.

Tabela 1.5. *Operatory bitowe w PHP*

Operator	Nazwa	Przykład użycia	Wynik
`&`	Bitowe `AND`	`$a & $b`	Ustawione bity, które były ustawione w `$a` i `$b`
`|`	Bitowe `OR`	`$a | $b`	Ustawione bity, które były ustawione w `$a` lub `$b`
`~`	Bitowe `NOT`	`~$a`	Ustawione bity, które nie były ustawione w `$a`, i vice versa
`^`	Bitowe `XOR`	`$a ^ $b`	Ustawione bity, które były ustawione w `$a` lub `$b`, lecz nie w obu
`<<`	Przesunięcie w lewo	`$a << $b`	Przesuwa `$a` w lewo o `$b` bitów
`>>`	Przesunięcie w prawo	`$a >> $b`	Przesuwa `$a` w prawo o `$b` bitów

Operator trójkowy

Operator trójkowy, symbol `?:`, działa według następującego wzoru:

```
warunek ? wartość, jeżeli prawdziwy : wartość, jeżeli fałszywy
```

Operator trójkowy jest podobny do wersji wyrażenia instrukcji `if - else`, które opiszemy w dalszej części tego rozdziału.

Oto prosty przykład:

```
($stopien > 50 ? 'Pozytywny' : 'Negatywny');
```

To wyrażenie dzieli stopnie studentów na „pozytywny” i „negatywny”.

Operator tłumienia błędów

Operator tłumienia błędów, symbol `@`, może być zastosowany na początku każdego *wyrażenia*, to znaczy wszystkiego, co ma bądź tworzy wartość. Na przykład:

```
$a = @(57/0);
```

Bez operatora `@` ten wiersz wygeneruje ostrzeżenie o dzieleniu przez zero (warto spróbować). Kiedy operator występuje, błąd zostaje stłumiony.

Przy stosowaniu takiego tłumienia ostrzeżeń należy stosować kod obsługujący błędy, aby sprawdzić, kiedy wystąpiło ostrzeżenie. Jeżeli w konkretnym egzemplarzu PHP w pliku *php.ini* jest włączona opcja `track_errors`, komunikat o błędzie będzie przechowywany w zmiennej globalnej `$php_errormsg`.

Operator wykonania

Operator wykonania to w istocie para operatorów (``). Symbol ten można znaleźć zazwyczaj na tym samym klawiszu, co tyldę (~).

PHP będzie próbował wykonać dowolny kod zawarty pomiędzy tymi symbolami, interpretując go jako polecenie wykonywane w wierszu poleceń serwera. Wartością wyrażenia jest wynik polecenia.

Na przykład dla systemów uniksowych polecenie może wyglądać następująco:

```
$out = `ls -la`;
echo '<pre>'.$out.'</pre>';
```

Dla serwera Windows to samo polecenie jest następujące:

```
$out = `dir c:`;
echo '<pre>'.$out.'</pre>';
```

Każde z tych poleceń utworzy wydruk katalogów i zapisze go w zmiennej `$out`. Może on być później wyświetlony przez przeglądarkę lub obrabiany w dowolny sposób.

Operatory tablicowe

Istnieje szereg operatorów tablicowych. Operatory elementu tablicy (`[]`) umożliwiają uzyskanie dostępu do elementów tablicy. W kontekście niektórych tablic można także używać operatora `=>`.

Nietrudno zauważyć, że wszystkie operatory tablicowe przedstawione w tabeli 1.6 mają odpowiadające sobie operatory działające na zmiennych skalarnych. Wystarczy tylko zapamiętać, że + wykonuje operację dodawania na danych skalarnych i unię na tablicach — nawet jeśli nie przejawia się szczególnego zainteresowania zasadami arytmetyki różniącymi te dwa działania — a działanie operatorów szybko nabierze sensu. Nie istnieje natomiast możliwość użytecznego porównywania tablic z danymi skalarnymi.

Tabela 1.6. *Operatory tablicowe w PHP*

Operator	Nazwa	Przykład użycia	Wynik
`+`	Unia	`$a + $b`	Zwraca tablicę zawierającą wszystkie elementy tablic `$a` i `$b`
`==`	Równość	`$a == $b`	Zwraca wartość `true`, jeśli `$a` i `$b` mają te same pary kluczy i wartości
`===`	Identyczność	`$a === $b`	Zwraca wartość `true`, jeśli `$a` i `$b` mają te same pary kluczy i wartości, ułożone w tej samej kolejności
`!=`	Nierówność	`$a != $b`	Zwraca wartość `true`, jeśli `$a` i `$b` nie są sobie równe
`<>`	Nierówność	`$a <> $b`	Zwraca wartość `true`, jeśli `$a` i `$b` nie są sobie równe
`!==`	Nieidentyczność	`$a !== $b`	Zwraca wartość `true`, jeśli `$a` i `$b` nie są identyczne

Operator typu

Dostępny jest jeden operator typu: `instanceof`. Jest on używany w programowaniu zorientowanym obiektowo, lecz wspominamy o nim tutaj dla zapewnienia kompletności wywodu

Operator `instanceof` pozwala na sprawdzanie, czy obiekt jest egzemplarzem konkretnej klasy, na przykład:

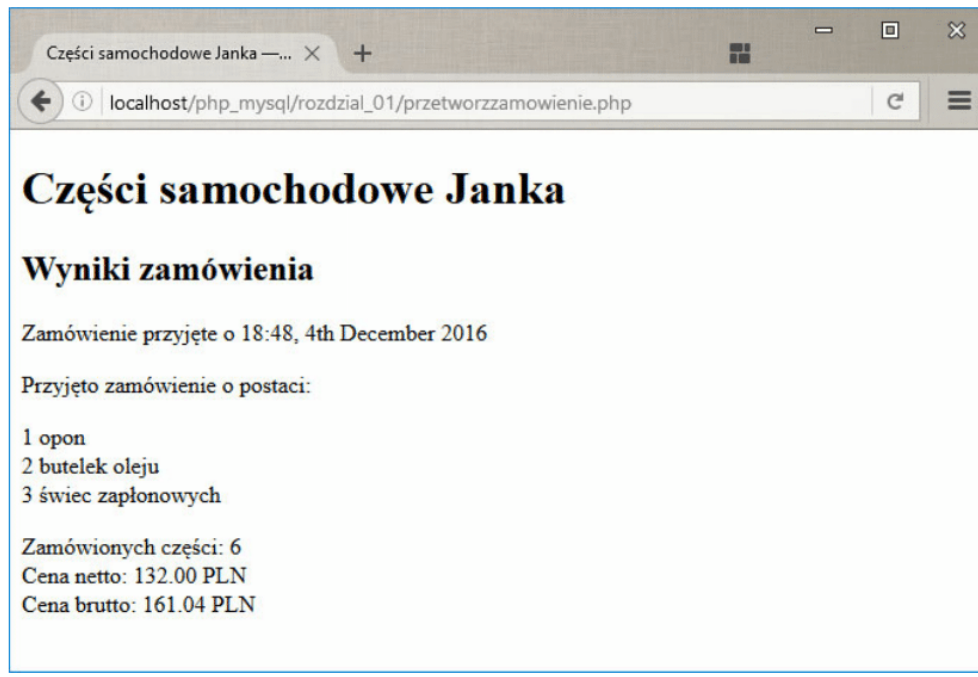
```
class przykladowaKlasa();
$mojObiekt = new przykladowaKlasa();
if ($mojObiekt instanceof przykladowaKlasa)
  echo "mojObiekt jest egzemplarzem klasy przykladowaKlasa";
```

Obliczanie sum w formularzu

Znając sposób stosowania operatorów, można obliczyć sumy i podatki w formularzu zamówień Janka. Aby to zrobić, należy dodać na końcu skryptu PHP następujący fragment kodu:

```
$ilosc = 0;
$ilosc = $iloscopon + $iloscoleju + $iloscswiec;
echo "<p>Zamówionych części:     ".$ilosc."<br />";

$wartosc = 0.00;

define('CENAOPON', 100);
define('CENAOLEJU', 10);
define('CENASWIEC', 4);

$wartosc = $iloscopon * CENAOPON
           + $iloscoleju * CENAOLEJU
           + $iloscswiec * CENASWIEC;

echo "Cena netto: ".number_format($wartosc, 2)." PLN<br />";

$stawkavat = 0.22;   // stawka VAT wynosi 22%
$wartosc = $wartosc * (1 + $stawkavat);
echo "Cena brutto: ".number_format($wartosc, 2)." PLN</p>";
```

Po odświeżeniu zawartości strony w oknie przeglądarki powinien pojawić się wynik podobny do przedstawionego na rysunku 1.5.



Rysunek 1.5. *Wszystkie sumy dla zamówienia klienta zostały obliczone, sformatowane i wyświetlone*

Jak łatwo zauważyć, w powyższym kodzie zastosowano kilka operatorów. Użyto operatorów sumy (+) i iloczynu (*), aby obliczyć wartość, oraz operatora łączenia łańcuchów (.) w celu sformatowania wyniku dla przeglądarki.

Zastosowano również funkcję `number_format()`, aby sformatować sumy jako łańcuch znaków z dwoma miejscami po przecinku. Funkcja ta pochodzi z biblioteki matematycznej (*Math*) PHP.

Po bliższym przyjrzeniu się obliczeniom można zastanawiać się, dlaczego zostały one wykonane właśnie w takim porządku. Rozważmy na przykład następującą instrukcję:

```
$wartosc = $iloscopon * CENAOPON
         + $iloscoleju * CENAOLEJU
         + $iloscswiec * CENASWIEC;
```

Całkowita wartość wydaje się poprawna, ale dlaczego operacje mnożenia zostały wykonane przed dodawaniem? Otóż należy zwrócić uwagę na *pierwszeństwo* operatorów, to znaczy porządek, w którym są one brane pod uwagę.

Pierwszeństwo i kolejność

Ogólnie rzecz biorąc, operatory posiadają zdefiniowane pierwszeństwo, czyli porządek, w jakim są obliczane. Operatory posiadają również kolejność, to znaczy porządek, w którym są obliczane operatory o takim samym pierwszeństwie. Generalnie istnieją trzy rodzaje kolejności: od lewej do prawej (nazywane w skrócie *lewą*), od prawej do lewej (nazywane w skrócie *prawą*), a do niektórych operatorów kolejność w ogóle *nie odnosi się*.

Tabela 1.7 przedstawia pierwszeństwo i kolejność operatorów w PHP. W tabeli tej operatory o najniższym pierwszeństwie umieszczone są na szczycie, a pierwszeństwo wzrasta w miarę przemieszczania się w dół tabeli.

Tabela 1.7. *Pierwszeństwo operatorów w PHP*

Kolejność	Operatory
Lewa	`,`
Lewa	`or`
Lewa	`xor`
Lewa	`and`
Prawa	`print`
Lewa	`= += -= *= /= .= %= &= |= ^= ~= <<= >>=`
Lewa	`? :`
Lewa	`||`
Lewa	`&&`
Lewa	`|`
Lewa	`^`
Lewa	`&`
Nie odnosi się	`== != === !==`
Nie odnosi się	`< <= > >=`
Lewa	`<< >>`
Lewa	`+ - .`

Tabela 1.7. *Pierwszeństwo operatorów w PHP (ciąg dalszy)*

Kolejność	Operatory
Lewa	.
Lewa	* / %
Prawa	!
Nie odnosi się	instanceof
Prawa	~ (int) (double) (string) (array) (object) @
Nie odnosi się	++ --
Prawa	[ ]
Nie odnosi się	clone new
Nie odnosi się	()

Należy zauważyć, że operatorem o najwyższym pierwszeństwie jest ten, o którym jeszcze nic nie zostało powiedziane — zwyczajne nawiasy. Efektem tego operatora jest podniesienie pierwszeństwa dowolnego kodu zawartego pomiędzy nawiasami. W razie konieczności to sposób na ominięcie reguł pierwszeństwa.

Przypomnijmy sobie fragment ostatniego przykładu:

```
$wartosc = $wartosc * (1 + $stawkavat);
```

Jeżeli powyższy wiersz wyglądałby tak:

```
$wartosc = $wartosc * 1 + $stawkavat;
```

to operator iloczynu, posiadający wyższe pierwszeństwo niż operator dodawania, zostałby wykonany wcześniej, i w związku z tym całe wyrażenie dałoby nieprawidłowy wynik. Stosując nawiasy, można wymusić wcześniejsze obliczenie wyrażenia `1 + $stawkavat`.

W wyrażeniu można zastosować dowolną liczbę nawiasów. Jako pierwsze obliczone zostaną te najbardziej wewnętrzne.

Należy zwrócić uwagę na jeszcze jeden operator z tej tabeli, o którym dotąd nie wspomniano: konstrukcję `print`, stanowiącą odpowiednik instrukcji `echo`. Obydwie konstrukcje generują dane wyjściowe.

Generalnie w tej książce używamy instrukcji `echo`, lecz można też używać instrukcji `print`, jeśli okaże się to bardziej czytelne. Ani `print`, ani `echo` nie są tak naprawdę funkcjami, obie jednak można wywoływać tak jak funkcje: podając parametry w nawiasach. Obydwie można również traktować podobnie jak operatory: wystarczy tylko umieścić przetwarzany łańcuch znaków za słowem kluczowym `echo` lub `print`.

Jeśli wywołamy `print` jak funkcję, zwróci ona wartość (`1`). Własność ta może okazać się użyteczna, gdy dane wyjściowe będą miały być generowane wewnątrz bardziej złożonego wyrażenia, lecz z drugiej strony `print` jest wolniejsza niż `echo`.