

SYSTEMY ZARZĄDZANIA TREŚCIĄ CMS

LABORATORIUM 3

Funkcje zarządzania zmiennymi

Aby zakończyć eksplorację świata zmiennych i operatorów, należy jeszcze przedstawić funkcje zarządzające zmiennymi w PHP. Jest to biblioteka funkcji służących do uzyskiwania informacji na temat zmiennych i wykonywania różnych operacji na nich.

Sprawdzanie i ustawianie typów zmiennych

Większość funkcji zarządzających zmiennymi jest związana ze sprawdzaniem ich typu. Dwie najbardziej ogólne funkcje to `gettype()` i `settype()`. Poniżej przedstawione zostały prototypy tych funkcji, czyli argumenty przez nie wymagane i wyniki przez nie zwracane:

```
string gettype(mixed zmienna);  
bool settype(mixed zmienna, string typ_zmiennej);
```

Aby zastosować funkcję `gettype()`, trzeba przekazać jej zmienną. Funkcja ta określi jej typ i zwróci łańcuch zawierający nazwę typu: `bool`, `integer`, `double` (dla wartości typu `float`, co ze względów historycznych może być nieco mylące), `string`, `array`, `object`, `resource` lub `NULL`. Jeżeli nie będzie to żaden z wymienionych typów standardowych, funkcja zwróci wyrażenie `"unknown type"`.

Aby zastosować `settype()`, trzeba przekazać jej zmienną, której typ ma zostać zmieniony, oraz łańcuch zawierający nazwę pożądanego typu zmiennej, wybraną z powyższej listy.



Konwencja stosowana w tej książce oraz w dokumentacji *php.net* odnosi się do typu danych `mixed`. Taki typ danych nie istnieje, ale ponieważ PHP jest tak elastyczny w zakresie obsługi typów, wiele funkcji może pobierać jako argumenty różne (lub dowolne) typy danych. Argumenty, które mogą różne typy danych, są oznaczane jako `mixed`.

Oto przykład użycia powyższych funkcji:

```
$a = 56;  
echo gettype($a). '<br />';  
settype($a, 'float');  
echo gettype($a). '<br />';
```

Po pierwszym wywołaniu funkcji `gettype()` typ zmiennej `$a` to `integer`. Po wywołaniu funkcji `settype()` typ zostanie zmieniony na `float`, choć funkcja `gettype()` będzie go prezentować jako `double`. (Koniecznie trzeba pamiętać o tej różnicy).

PHP posiada również pewne funkcje testujące typ zmiennej, zależne od tego typu. Każda z nich pobiera zmienną jako argument i zwraca wartość `true` lub `false`. Funkcje te są następujące:

- `is_array()` — sprawdza, czy zmienna jest tablicą;
- `is_double()`, `is_float()`, `is_real()` (funkcje równoznaczne) — sprawdza, czy zmienna jest liczbą zmiennoprzecinkową;
- `is_long()`, `is_int()`, `is_integer()` (funkcje równoznaczne) — sprawdza, czy zmienna jest liczbą całkowitą;
- `is_string()` — sprawdza, czy zmienna jest łańcuchem znaków;
- `is_bool()` — sprawdza, czy zmienna ma wartość logiczną;
- `is_object()` — sprawdza, czy zmienna jest obiektem;

- `is_resource()` — sprawdza, czy zmienna jest wskaźnikiem zasobów;
- `is_null()` — sprawdza, czy zmienna jest zmienną `null`;
- `is_scalar()` — sprawdza, czy zmienna jest skalar, to znaczy czy jest typu `integer`, `boolean`, `string` lub `float`;
- `is_numeric()` — sprawdza, czy zmienna ma wartość liczbową lub jest numerycznym łańcuchem znaków;
- `is_callable()` — sprawdza, czy zmienna stanowi nazwę prawidłowej funkcji.

Sprawdzanie stanu zmiennej

PHP posiada kilka funkcji sprawdzających stan zmiennej. Pierwszą z nich jest `isset()`, która posiada następujący prototyp:

```
bool isset(mixed zmienna [, mixed zmienna[...]]);
```

Funkcja ta pobiera zmienną jako argument i zwraca wartość `true`, jeżeli dana zmienna istnieje, lub `false` w przeciwnym wypadku. Można do niej przekazać także listę zmiennych oddzielonych znakiem przecinka. Wówczas `isset()` zwróci wartość `true`, jeżeli wszystkie zmienne będą istnieć.

Można usunąć z pamięci zmienną, stosując funkcję towarzyszącą powyższej, `unset()`. Posiada ona następujący prototyp:

```
void unset(mixed zmienna [, mixed zmienna[...]]);
```

Funkcja ta pozbywa się z pamięci przekazanej jej zmiennej i zwraca wartość `true`.

Istnieje również funkcja `empty()`. Sprawdza ona, czy dana zmienna istnieje i czy posiada wartość pustą i zerową, i zwraca odpowiednio `true` lub `false`. Posiada następujący prototyp:

```
boolean empty(mixed zmienna);
```

Oto przykład z zastosowaniem powyższych trzech funkcji.

Dodajmy tymczasowo do skryptu następujące wiersze kodu:

```
echo 'isset($iloscopon): ' . isset($iloscopon) . '<br />';
echo 'isset($niema): ' . isset($niema) . '<br />';
echo 'empty($iloscopon): ' . empty($iloscopon) . '<br />';
echo 'empty($niema): ' . empty($niema) . '<br />';
```

Odświeżmy zawartość strony i obejrzymy wyniki.

Zmienna `$iloscopon` powinna zwrócić wartość `1` (`true`) dla funkcji `isset()` niezależnie od tego, czy w formularzu została do niej wprowadzona wartość i ile ona wynosi. Wynik funkcji `empty()` zależy od wprowadzonej wartości.

Zmienna `$niema` nie istnieje, tak więc funkcja `isset()` zwróci wartość pustą (`false`), a funkcja `empty()` wynik `1` (`true`).

Funkcje powyższe mogą być przydatne przy sprawdzaniu, czy użytkownik wypełnił odpowiednie pola formularza.

Reinterpretacja zmiennych

Można osiągnąć wyniki podobne do rzutowania typu zmiennej poprzez wywołanie odpowiedniej funkcji. Oto trzy funkcje przydatne do tego zadania:

```
int intval(mixed $zmienna);  
float floatval(mixed $zmienna);  
string strval(mixed $zmienna);
```

Każda z nich pobiera na wejściu zmienną i zwraca jej wartość przekształconą na odpowiedni typ.

Podejmowanie decyzji za pomocą instrukcji warunkowych

Struktury kontrolujące to struktury w obrębie języka, pozwalające na kontrolę przebiegu wykonania programu bądź skryptu. Można podzielić je na struktury warunkowe (inaczej zwane rozgałęzionymi) i struktury powtarzalne, czyli pętle. Specyficzne zastosowania tych struktur w PHP zostaną omówione poniżej.

Aby właściwie odpowiedzieć na życzenia klienta, kod musi być zdolny do podejmowania decyzji. Konstrukcje dające programowi możliwość ich podejmowania są nazywane *instrukcjami warunkowymi*.

Instrukcja if

Do podejmowania decyzji może być zastosowana instrukcja `if`, która powinna otrzymać warunek użycia. Jeżeli wartość warunku wynosi `true`, zostanie wykonany następny fragment kodu. Warunki w instrukcji `if` muszą być oznaczone nawiasami `()`.

Na przykład jeżeli klient nie zamówi ani opon, ani oleju, ani świec, zazwyczaj oznacza to przypadkowe naciśnięcie przycisku *Złóż zamówienie* jeszcze przed wypełnieniem formularza. W tym wypadku powinna zostać wyświetlona wiadomość znacząca więcej niż „Zamówienie przyjęte”.

Kiedy odwiedzający stronę nic nie zamawia, powinna pojawić się wiadomość w rodzaju: „Na poprzedniej stronie nie zostało złożone żadne zamówienie!”. Można to łatwo wykonać za pomocą następującej instrukcji `if`:

```
if( $ilosc == 0 )  
    echo 'Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />';
```

W powyższym przykładzie został zastosowany warunek `$ilosc == 0`. Należy pamiętać, że operator równości (`==`) zachowuje się inaczej niż operator przypisania (`=`).

Wartość warunku `$ilosc == 0` wynosi `true`, jeżeli wartość zmiennej `$ilosc` jest równa zero. Jeśli wartość `$ilosc` nie jest równa zero, warunek będzie miał wartość `false`. Instrukcja `echo` zostanie wykonana, gdy wartość warunku wynosi `true`.

Bloki kodu

W poleceniu warunkowym, takim jak `if`, często występuje konieczność wykonania więcej niż jednej instrukcji. Nie ma wtedy potrzeby stosowania większej liczby warunków `if`. Zamiast tego można zebrać kilka instrukcji, tworząc *blok*. Aby zadeklarować blok, należy zamknąć go nawiasami klamrowymi:

```
if($ilosc == 0) {  
    echo '<p style="color:red">';  
    echo 'Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />';  
    echo '</p>';  
}
```

Powyższe trzy wiersze kodu otoczone nawiasami klamrowymi są teraz traktowane jako blok kodu. Kiedy wartość warunku wynosi `true`, zostaną wykonane wszystkie trzy wiersze. Kiedy warunek wynosi `false`, wszystkie zostaną zignorowane.



Uwaga

Jak już wspomniano, PHP nie zwraca uwagi na wygląd kodu. Zalecane jest wcinanie kodu w celu zwiększenia jego czytelności. Ogólnie rzecz biorąc, wcinanie stosuje się w celu łatwego sprawdzenia, które wiersze kodu zostaną wykonane w przypadku spełnienia warunków, które instrukcje pogrupowane są w bloki, a które są częścią pętli bądź funkcji. W powyższych przykładach wcięta została instrukcja zależna od instrukcji `if` oraz instrukcje tworzące blok.

Instrukcja else

Zazwyczaj podejmuje się decyzje nie tylko co do warunków, w których zostanie wykonana instrukcja, lecz są opisane wszystkie możliwe działania.

Polecenie `else` pozwala na zdefiniowanie działania alternatywnego, podejmowanego wtedy, gdy wartość instrukcji `if` wynosi `false`. Należy ostrzec klientów Janka, kiedy nic nie zamawiają, w przeciwnym wypadku trzeba im pokazać złożone przez nich zamówienie.

Jeżeli kod przykładu ulegnie zmianie i zostanie dodana instrukcja `else`, to jest możliwe wyświetlenie albo ostrzeżenia, albo podsumowania zamówienia.

```
if($ilosc == 0) {  
    echo "Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />";  
} else {  
    echo htmlspecialchars($iloscoPON).' opon<br />';  
    echo htmlspecialchars($iloscoLEJU).' butelek oleju<br />';  
    echo htmlspecialchars($iloscoSWIEC).' świec zapłonowych<br />';  
}
```

Można wypracować bardziej skomplikowane procesy logiczne poprzez stosowanie zagnieżdżonych w sobie instrukcji `if`. W poniższym kodzie nie tylko podsumowanie zostanie wyświetlone, gdy wartość warunku `$ilosc == 0` wynosi `true`. Każdy wiersz podsumowania zostanie wyświetlony jedynie w razie spełnienia jego odpowiedniego warunku.

```
if ($ilosc == 0) {  
    echo "Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />";  
} else {  
    if ($iloscoPON > 0)  
        echo htmlspecialchars($iloscoPON).' opon<br />';  
    if ($iloscoLEJU > 0)  
        echo htmlspecialchars($iloscoLEJU).' butelek oleju<br />';  
    if ($iloscoSWIEC > 0)  
        echo htmlspecialchars($iloscoSWIEC).' świec zapłonowych<br />';  
}
```


Instrukcja elseif

Dla wielu podejmowanych decyzji istnieją więcej niż dwie opcje wyboru. Można utworzyć sekwencję wielu opcji, stosując instrukcję `elseif`, która jest połączeniem instrukcji `else` i `if`. Przy użyciu sekwencji warunków program sprawdza każdy z nich, dopóki nie znajdzie tego, którego warunek wynosi `true`.

Janek wprowadził system zniżek dla zamówień na duże ilości opon. Poniżej przedstawiono schemat tych zniżek:

- zamówiono mniej niż 10 opon — brak zniżki,
- zamówiono 10 – 49 opon — zniżka 5%,
- zamówiono 50 – 99 opon — zniżka 10%,
- zamówiono powyżej 100 opon — zniżka 15%.

Za pomocą instrukcji `if` i `elseif` można utworzyć fragment kodu obliczający wielkość zniżki. W takim przypadku należy użyć operatora `AND (&&)`, aby połączyć dwa warunki w jeden.

```
if( $iloscopon < 10 )
    $znizka = 0;
elseif( $iloscopon >= 10 && $iloscopon <= 49 )
    $znizka = 5;
elseif( $iloscopon >= 50 && $iloscopon <= 99 )
    $znizka = 10;
elseif( $iloscopon > 100 )
    $znizka = 15;
```

Warto zauważyć, że można zamiennie stosować zapis `elseif` oraz `else if` — oba są poprawne.

Przy tworzeniu listy instrukcji `elseif` należy pamiętać, że tylko jedna z instrukcji lub ich bloków zostanie wykonana. W powyższym przykładzie nie miało to znaczenia, ponieważ warunki wykluczały się wzajemnie — tylko jeden z nich mógł być spełniony w tym samym czasie. Jeżeli nastąpi sytuacja, w której więcej niż jeden warunek okaże się prawdziwy, zostanie wykonany tylko pierwszy z nich.

Instrukcja switch

Instrukcja `switch` działa na zasadach podobnych do instrukcji `if`, lecz w jej przypadku warunek może mieć więcej niż dwie wartości. W przypadku instrukcji `if` wartość warunku wynosi jedynie `true` lub `false`. W instrukcji `switch` warunek może posiadać dowolną ilość wartości. Muszą one jednak należeć do jednego z typów prostych (`integer`, `string` lub `float`). Należy wprowadzić instrukcję `case` dla każdej wartości oraz opcjonalnie wartość domyślną, dla której nie trzeba tworzyć instrukcji `case`.

Janek chciałby wiedzieć, jaki rodzaj reklamy jest najbardziej skuteczny. Można w tym celu dodać do formularza zamówień odpowiednie pytanie. Wprowadźmy do formularza następujący kod HTML. Powinno to dać wynik podobny do przedstawionego na rysunku 1.6:

```
<tr>
  <td>Jak dowiedzieli się Państwo o sklepie Janka?</td>
  <td><select name="jak">
    <option value = "a">Jestem stałym klientem
    <option value = "b">Reklama telewizyjna
    <option value = "c">Książka telefoniczna
    <option value = "d">Znajomy
  </select>
</td>
</tr>
```

Rysunek 1.6.
Formularz zamówienia
„zapytuje” klientów
o źródło informacji
na temat sklepu

Produkt	Ilość
Opony	<input type="text"/>
Olej	<input type="text"/>
Świece zapłonowe	<input type="text"/>
Jak dowiedzieli się Państwo o sklepie Janka?	
	<div>Jestem stałym klientem Jestem stałym klientem Reklama telewizyjna Książka telefoniczna Znajomy</div>

Powyższy kod HTML dodał nową zmienną formularza (o nazwie `jak`), której wartość wynosi `'a'`, `'b'`, `'c'` lub `'d'`. W następujący sposób można obsługiwać tę zmienną za pomocą serii instrukcji `if i elseif`:

```
if($jak == "a") {  
    echo "<P>Stały klient.</p>";  
} elseif($jak == "b") {  
    echo "<P>Reklama telewizyjna. </p>";  
} elseif($jak == "c") {  
    echo "<P>Książka telefoniczna. </p>";  
} elseif($jak == "d") {  
    echo "<P>Znajomy. </p>";  
} else {  
    echo "<P>Źródło nieznane.</P>";  
}
```

Alternatywnie można napisać instrukcję `switch`:

```
switch($jak) {  
    case "a" :  
        echo "<p>Stały klient.</p>";  
        break;  
    case "b" :  
        echo "<p>Reklama telewizyjna.</p>";  
        break;  
    case "c" :  
        echo "<p>Książka telefoniczna.</p>";  
        break;  
    case "d" :  
        echo "<p>Znajomy.</p>";  
        break;  
    default :  
        echo "<p>Źródło nieznane.</p>";  
        break;  
}
```

(Zwróćmy uwagę, że w obu tych przykładach zakłada się, iż wartość zmiennej `$jak` odczytano z tablicy `$_POST`).

Instrukcja `switch` funkcjonuje w nieco inny sposób niż instrukcje `if` i `elseif`. Polecenie `if` działa tylko na jedną instrukcję, chyba że zostaną użyte nawiasy klamrowe w celu utworzenia bloku instrukcji. Instrukcja `switch` działa w odwrotny sposób. Kiedy w `switch` zostanie włączona instrukcja `case`, PHP będzie wykonywał kolejne polecenia, aż napotka instrukcję `break`. Bez instrukcji przerywającej `switch` wykona cały kod następujący po prawdziwej wartości `case`. Kiedy osiągnie instrukcję `break`, przeskoczy do wykonywania pierwszej wiersza kodu po poleceniu `switch`.

Porównanie różnych instrukcji warunkowych

Osoby nieznające zbyt dobrze instrukcji przedstawionych w poprzednich punktach mogą zapytać: „Która z nich jest najlepsza?”.

Na to pytanie nie sposób odpowiedzieć wprost. Nie ma zadań wykonywanych za pomocą jednej lub kilku instrukcji `else`, `elseif` lub `switch`, których nie dałoby się wykonać przy użyciu serii instrukcji `if`. Należy stosować takie instrukcje, które czynią kod jak najbardziej czytelnym. Wraz ze wzrostem doświadczenia decyzje takie są coraz łatwiejsze.

Powtarzanie działań przy użyciu iteracji

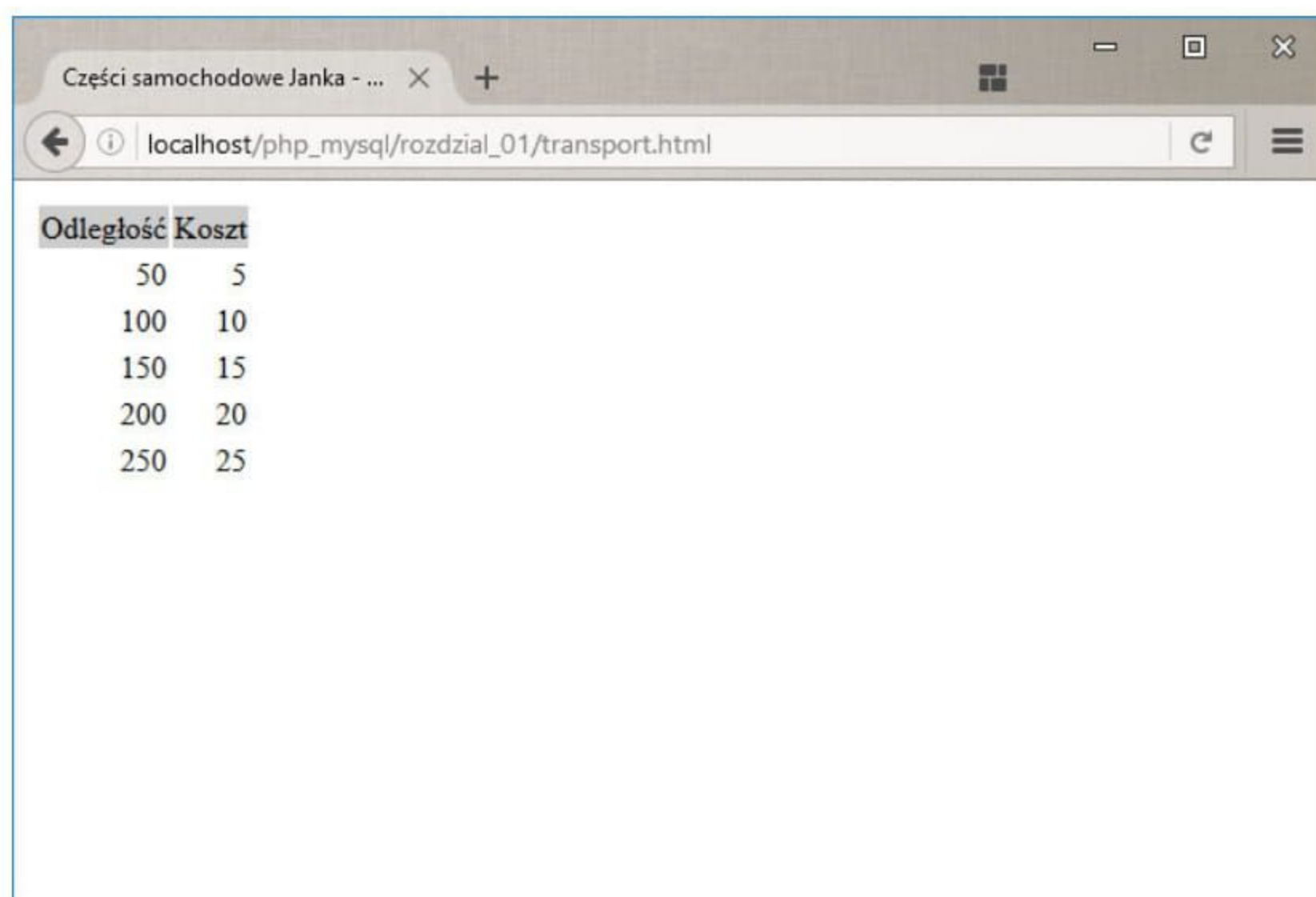
Do zadań, w których komputery zawsze przodowały, należy automatyzowanie powtarzanych zadań. Jeżeli istnieje coś, co trzeba wykonać wiele razy w ten sam sposób, można użyć pętli, aby powtórzyć pewne fragmenty programu.

Janek chciałby mieć tabelę wyświetlającą koszty transportu, które zostaną dodane do wartości zamówienia klienta. Kurier, z którego usług korzysta, uzależnia koszty od odległości, na jaką przesyłany zostaje dany towar. Koszt może być obliczony za pomocą prostego wzoru.

Tabela kosztów transportu powinna przypominać tę na rysunku 1.7.

Rysunek 1.7.

Tabela pokazuje koszt transportu, zmieniający się wraz ze wzrostem odległości

A screenshot of a web browser window. The address bar shows 'localhost/php_mysql/rozdzial_01/transport.html'. The browser title is 'Części samochodowe Janka - ...'. The main content area displays a table with two columns: 'Odległość' and 'Koszt'. The table contains five rows of data: (50, 5), (100, 10), (150, 15), (200, 20), and (250, 25).

Odległość	Koszt
50	5
100	10
150	15
200	20
250	25

Na listingu 1.2 przedstawiony jest kod HTML tworzący powyższą tabelę. Warto zwrócić uwagę na to, że jest on długi i posiada powtarzające się elementy.

Listing 1.2. *transport.html* — kod HTML tabeli kosztów transportu Janka

```
<!DOCTYPE>
<html>
<head>
  <title>Części samochodowe Janka - zestawienie kosztów przesyłek</title>
</head>
<body>
<table style="border: 0px; padding: 3px">
<tr>
  <td style="background: #cccccc; text-align: center;">Odległość</td>
  <td style="background: #cccccc; text-align: center;">Koszt</td>
</tr>
<tr>
  <td style="text-align: right;">50</td>
  <td style="text-align: right;">5</td>
</tr>
<tr>
  <td style="text-align: right;">100</td>
  <td style="text-align: right;">10</td>
</tr>
<tr>
  <td style="text-align: right;">150</td>
  <td style="text-align: right;">15</td>
</tr>
<tr>
  <td style="text-align: right;">200</td>
  <td style="text-align: right;">20</td>
</tr>
<tr>
  <td style="text-align: right;">250</td>
  <td style="text-align: right;">25</td>
</tr>
</table>
</body>
</html>
```

Do wpisania powyższego kodu znacznie wygodniej byłoby użyć taniego i niemęczącego się komputera niż płatnego, nudzącego się pracownika.

Instrukcje pętli nakazują PHP powtarzające się wykonywanie instrukcji lub bloku.

Pętla while

Pętla `while` jest najprostszym typem pętli w PHP. Podobnie jak instrukcja `if`, zależy ona od warunku. Różnica pomiędzy pętlą `while` a instrukcją `if` polega na tym, że instrukcja `if` wykonuje następujący po niej blok kodu jednokrotnie, jeżeli jej warunek ma wartość `true`. Pętla `while` będzie powtarzać wykonywanie bloku tak długo, jak jej warunek będzie miał wartość `true`.

Pętla `while` jest zazwyczaj używana w przypadkach, gdy nie wiadomo, ile iteracji trzeba wykonać, by warunek pętli został spełniony. Jeśli liczba tych iteracji jest ściśle określona, to warto zastanowić się nad zastosowaniem pętli `for`.

Podstawowa struktura pętli `while` jest następująca:

```
while( warunek ) wyrażenie
```

Poniższa pętla `while` wyświetla cyfry od 1 do 5.

```

$cyfra = 1;
while ($cyfra <= 5 ) {
    echo $cyfra."<br />";
    $cyfra++;
}

```

Na początku każdej iteracji sprawdzany jest warunek. Jeżeli jego wartość wynosi false, blok nie zostanie wykonany i pętla zakończy się. Zostanie wtedy wykonana pierwsza instrukcja następująca po pętli.

Pętla `while` może być zastosowana do bardziej użytecznego zadania, na przykład do wyświetlenia tabeli kosztów transportu pokazanej na rysunku 1.7. Kod na listingu 1.3 stosuje pętlę `while` do utworzenia tabeli kosztów transportu.

Listing 1.3. *transport.php — tworzenie tabel kosztów transportu Janka za pomocą PHP*

```

<!DOCTYPE>
<html>
<head>
    <title>Części samochodowe Janka - zestawienie kosztów przesyłek</title>
</head>
<body>
<table style="border: 0px; padding: 3px">
<tr>
    <td style="background: #cccccc; text-align: center;">Odległość</td>
    <td style="background: #cccccc; text-align: center;">Koszt</td>
</tr>
<?php
$odleglosc = 50;
while ($odleglosc <= 250) {
    echo "<tr>
        <td style=\"text-align: right;\">".$odleglosc."</td>
        <td style=\" text-align: right;\">". ($odleglosc / 10) . "</td>
    </tr>\n";

    $odleglosc += 50;
}

?>
</table>
</body>
</html>

```

Aby zwiększyć czytelność kodu HTML generowanego przez ten skrypt, należy dołączyć nowe wiersze i znaki spacji. Jak już wspomniano, przeglądarki zignorują je, lecz okażą się one ważne dla użytkowników. Kod HTML trzeba przeglądać często — zwłaszcza gdy dane wyjściowe nie są takie, jak oczekiwano.

Na listingu 1.3 umieszczono wewnątrz niektórych łańcuchów znaków znaki `\n`. W łańcuchach znaków ograniczonych cudzysłowem sekwencja ta reprezentuje znak nowego wiersza.

Pętle `for` i `foreach`

W poprzednim punkcie przedstawiono bardzo popularny sposób użycia pętli `while`. Na jej początku została ustawiona początkowa wartość licznika. Przed każdą iteracją licznik był sprawdzany co do zgodności z warunkiem, zaś na końcu każdej iteracji — modyfikowany.

Ten typ pętli może być zapisany w krótszy sposób, przy użyciu pętli `for`. Podstawowa struktura pętli `for` jest następująca:

```
for( wyrażenie1; warunek; wyrażenie2 )  
    wyrażenie3;
```

- *Wyrażenie1* jest wykonywane raz, na początku. Zazwyczaj jest to początkowe ustawienie licznika.
- *Warunek* jest sprawdzany przed każdą iteracją. Jeżeli jego wartość wynosi `false`, iteracja zatrzymuje się. W tym miejscu zazwyczaj testuje się przekroczenie limitu licznika.
- *Wyrażenie2* jest wykonywane na końcu każdej iteracji. Tutaj zazwyczaj zmienia się wartość licznika.
- *Wyrażenie3* wykonywane jest jednokrotnie podczas każdej iteracji. Wyrażenie to zazwyczaj blok kodu, zawierający główną część kodu pętli.

Pętla `while` z listingu 1.3 może zostać zmieniona na pętlę `for`. W takim przypadku kod PHP jest następujący:

```
<?php  
for($odleglosc = 50; $odleglosc <= 250; $odleglosc += 50) {  
    echo "<tr>  
        <td style=\"text-align: right;\">".$odleglosc."</td>  
        <td style=\"text-align: right;\">". ($odleglosc / 10) . "</td>  
    </tr>\n";  
}  
?>
```

Funkcjonalnie wersje z pętlą `while` i pętlą `for` są identyczne. Pętla `for` jest jednak nieco krótsza, dokładnie o dwa wiersze.

Oba typy pętli są równoważne — żadna z nich nie jest ani lepsza, ani gorsza od drugiej. W konkretnej sytuacji używa się intuicyjnie wygodniejszej.

Nawiasem mówiąc, z pętlą `for` można łączyć zmienne zmiennych w celu iteracji serii podobnych pól formularza. Na przykład jeżeli dane są pola formularza o nazwach `nazwa1`, `nazwa2`, `nazwa3` itd., można wykonać na nich działanie, takie jak:

```
for($i=1; $i <= $ilosc_pol; $i++) {  
    $temp= "nazwa$i";  
    echo htmlspecialchars($temp). '<br />'; // czy też jakiegokolwiek inne działanie  
}
```

Stosując dynamiczne tworzenie nazw zmiennych, można uzyskać po kolei dostęp do każdego z tych pól.

Oprócz pętli `for` istnieje również pętla `foreach`, przeznaczona do stosowania względem tablic. Jej zastosowanie omówimy w rozdziale 3.

Pętle `do..while`

Ostatni omówiony tutaj typ pętli działa w trochę inny sposób. Ogólna struktura pętli `do..while` wygląda następująco:

```
do {  
    wyrażenie  
}  
while( warunek );
```

Pętla `do...while` różni się od pętli `while` tym, że warunek jest w niej testowany na końcu. Oznacza to, że wyrażenie zawarte w pętli `do...while` zostanie wykonane co najmniej raz.

W poniższym przykładzie wartość warunku wynosi na początku `false` i nigdy nie zmieni się na `true`, lecz pętla zostanie wykonana raz, zanim sprawdzony zostanie warunek i pętla zakończy się.

```
$cyfra = 100;
do {
    echo $cyfra."<br />";
} while($cyfra < 1);
```

Wyłamywanie się ze struktury skryptu

Istnieją trzy sposoby na przerwanie wykonywania danego fragmentu kodu, zależnie od pożądanego efektu.

Aby zatrzymać wykonywanie pętli, należy zastosować instrukcję `break`, tak jak to zostało opisane w podrozdziale na temat instrukcji `switch`. Po napotkaniu instrukcji `break` zostanie wykonany pierwszy wiersz kodu za końcem pętli.

Aby przeskoczyć do następnej iteracji pętli, należy zastosować instrukcję `continue`.

Aby zakończyć wykonywanie całego skryptu PHP, stosuje się instrukcję `exit`, szczególnie przy wykrywaniu błędów. Na przykład można zmodyfikować powyższy przykład w następujący sposób:

```
if($ilosc == 0) {
    echo "Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />";
    exit;
}
```

Wywołanie funkcji `exit` powoduje zaprzestanie wykonywania pozostałej części skryptu PHP.

Używanie alternatywnych składni struktur sterujących

Dla wszystkich struktur sterujących, które dotychczas analizowaliśmy, istnieją składnie alternatywne. Sprowadzają się one do zastąpienia otwierającego nawiasu klamrowego (`{`) znakiem dwukropka (`:`) oraz zamykającego nawiasu klamrowego (`}`) jednym z nowych słów kluczowych: `endif`, `endswitch`, `endwhile`, `endfor` lub `endforeach`, zależnie od tego, która struktura sterująca jest akurat używana. Jedynie dla pętli `do...while` składnia alternatywna nie jest dostępna.

Na przykład kod postaci:

```
if($ilosc == 0) {
    echo "Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />";
    exit;
}
```

można przekształcić, używając składni alternatywnej ze słowami kluczowymi `if` i `endif`:

```
if($ilosc == 0) :
    echo "Na poprzedniej stronie nie zostało złożone żadne zamówienie!<br />";
    exit;
endif;
```