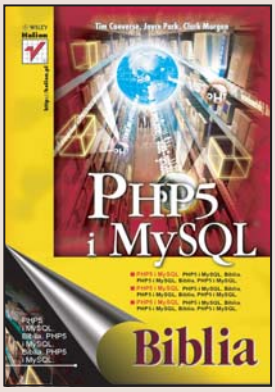
PHP i MySQL. Biblia



Autorzy: Tim Converse, Joyce Park, Clark Morgan
Tłumaczenie: Robert Górczyński, Daniel Kaczmarek, Maja Królikowska, Marek Pałczyński
ISBN: 83-7361-940-2
Tytuł oryginału: PHP5 and MySQL Bible
Format: B5, stron: 1080

Wyczerpujące omówienie najpopularniejszego środowiska do tworzenia dynamicznych witryn WWW

- Poznaj zasady programowania w PHP5 i w pełni wykorzystaj jego nowe możliwości
- Zastosuj bazę danych jako zaplecze dla stron WWW
- Zaprojektuj i stwórz dynamiczną witrynę WWW za pomocą PHP i MySQL

O PHP i MySQL słyszał już chyba każdy, kto zajmuje się projektowaniem witryn WWW. Skrypty napisane w języku PHP i korzystające z bazy danych MySQL „napędzają" tysiące dynamicznych stron WWW – sklepów, portali oraz aplikacji e-commerce. Duet PHP i MySQL to stabilne, wydajne i proste do opanowania środowisko o ogromnych możliwościach. Dostępne są nieodpłatnie, co jest jednym z głównych powodów ich popularności. Za ich pomocą można stworzyć zarówno proste wiadomości na stronie WWW, jak i rozbudowane systemy autoryzacji użytkowników, płatności elektronicznych, galerii i wiele innych aplikacji internetowych.

„PHP5 i MySQL. Biblia" to kompleksowe omówienie zagadnień związanych z tworzeniem witryn WWW z wykorzystaniem tych technologii. Przedstawia zasady programowania w języku PHP5 z uwzględnieniem reguł projektowania obiektowego. Opisuje bazę danych MySQL, język SQL oraz sposoby połączenia skryptów PHP z tabelami w bazie danych. Czytając ją, nauczysz się osadzać kod PHP w dokumentach HTML, korzystać z mechanizmów obsługi sesji oraz języka XML, korzystać z innych baz danych oraz zabezpieczać witryny WWW stworzone za pomocą PHP. Dowiesz się, jak zaimplementować obsługę wyjątków oraz jak usuwać błędy z kodu źródłowego.

- Typy danych, zmienne, polecenia i funkcje w PHP
- Operacje na tekstach, wartościach liczbowych i tablicach
- Administrowanie bazą MySQL
- Łączenie skryptów PHP z bazą danych
- Tworzenie elementów generowanych dynamicznie
- Programowanie obiektowe w PHP
- Mechanizmy obsługi sesji i plików cookie
- Korzystanie z biblioteki PEAR
- Obsługa wyjątków i usuwanie błędów z kodów źródłowych
- Korzystanie z baz danych PostgreSQL oraz Oracle
- Łączenie PHP z JavaScript i Javą
- Przetwarzanie plików XML i tworzenie usług sieciowych
- Generowanie grafiki za pomocą PHP
- Uwierzytelnianie użytkowników

W tej książce znajdziesz wszystko, czego potrzebujesz aby w pełni wykorzystać możliwości PHP i MySQL

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

Spis treści

Rozdział 8.
Ciągi

W tym rozdziale:

- Ciągi w PHP
- Funkcje obsługi ciągów
- Rozszerzony przykład: kalkulator do ćwiczeń

Mimo że rysunki, dźwięki, animacje i aplety stają się coraz ważniejszą częścią sieci WWW, nadal bazuje ona na tekście. Podstawowym typem PHP przechowującym tekst jest typ `string`.

W niniejszym rozdziale opisane zostaną niemal wszystkie dostępne w PHP narzędzia do manipulowania ciągami (jedynie bardziej zaawansowane funkcje operujące na ciągach oraz dopasowywanie ciągów za pomocą wyrażeń regularnych zostanie opisane później, w rozdziale 22.). Na początku przedstawimy podstawowe informacje o ciągach, a następnie zaprezentujemy ich działanie w praktyce, kontynuując rozbudowę kalkulatora do ćwiczeń z rozdziału 7.

Ciągi w PHP

Ciągi to sekwencje znaków traktowane jako odrębna jednostka. Mogą być przypisywane do zmiennych, używane jako parametry funkcji, zwracane z funkcji lub wysyłane na wyjście i oglądane w przeglądarce klienta. Najprostszą metodą stworzenia ciągu w PHP jest otoczenie znaków cudzysłowami (") bądź apostrofami (').

```
$ciag = 'Ciąg';
$inny_ciag = "Inny ciąg";
```

Różnica pomiędzy ciągami w cudzysłowach i apostrofach leży w interpretacji ich przez PHP. Jeżeli otoczysz ciąg apostrofami, prawie żadna interpretacja nie zostanie zastosowana, jeżeli zaś otoczysz cudzysłowami, PHP wklei wartości zmiennych w miejsce ich nazw i zamieni sekwencje sterujące na odpowiadające im znaki. Jeżeli np. wstawisz następujący kod do strony:

```
$wyrazenie = 'wszystko, co powiem';
$pytanie_1 =
  "Czy musisz brać $wyrazenie dosłownie?\n<BR>";
$pytanie_2 =
  'Czy musisz brać $wyrazenie dosłownie?\n<BR>';
echo $pytanie_1;
echo $pytanie_2;
```

powinieneś spodziewać się następującego wyniku:

```
Czy musisz brać wszystko, co powiem, dosłownie?
Czy musisz brać $wyrazenie dosłownie?\n
```

Szczegółowy opis traktowania przez PHP ciągów z apostrofami i cudzysłowami znajduje się w części „String" w rozdziale 5.

Dołączanie ciągów przy użyciu nawiasów klamrowych

W większości przypadków wystarczy umieścić zmienną w ciągu otoczonym cudzysłowami, aby wartość tej zmiennej została umieszczona jako część ciągu znaków w trakcie interpretowania kodu. Istnieją jednak dwie sytuacje, w których interpreter może wymagać bardziej szczegółowych wskazówek od programisty. Pierwsza z nich zachodzi wówczas, gdy zamierzenie programisty co do umiejscowienia zmiennej kłóci się z zasadami stosowanymi przez interpreter, druga natomiast ma miejsce wtedy, gdy dołączane wyrażenie nie jest zwykłą zmienną. W obydwu przypadkach wszelkie wątpliwości można rozwiać umieszczając zmienną, która ma zostać dołączona, w nawiasach klamrowych `{}`.

PHP nie będzie na przykład miał trudności z interpretacją następującego kodu:

```
$sport = 'volleyball';
$plan = "W lecie będę grał w $sport";
```

W tej sytuacji analizator napotka symbol `$`, po czym rozpocznie odczytywanie znaków składających się na nazwę zmiennej do momentu, gdy dotrze do znaku spacji występującego po słowie `$sport`. Znaki spacji nie mogą znajdować się w nazwie zmiennej, zatem oczywiste jest, że zmienna nosi nazwę `$sport`. PHP bez trudu ustali wartość tak zidentyfikowanej zmiennej (`'volleyball'`) i umieści ją w ciągu znaków.

Czasami jednak umieszczanie znaku spacji zaraz za nazwą zmiennej nie leży w interesie programisty. Spójrzmy na poniższy przykład:

```
$sport1 = 'volley';
$sport2 = 'foot';
$sport3 = 'basket';
$plan1 = "W lecie będę grał w $sport1ball";      // Źle
$plan2 = "Na jesieni będę grał w $sport2ball";   // Źle
$plan3 = "W zimie będę grał w $sport3ball";      // Źle
```

W tym przypadku programista nie osiągnie pożądanego efektu, ponieważ PHP zinterpretuje słowo `$sport1` jako część nazwy zmiennej `$sport1ball`, która najprawdopodobniej będzie nieokreślona. Zamiast takiego zapisu należałoby zatem napisać:

```
$plan1 = "W lecie będę grał w {$sport1}ball";    // Dobrze
```

Dzięki takiemu zapisowi PHP najpierw obliczy wartość zmiennej znajdującej się w nawiasach klamrowych, a dopiero potem dołączy ją do ciągu.

Z analogicznych powodów PHP nie radzi sobie z dołączaniem wyrażeń zawierających zmienne złożone, takie jak tablice wielowymiarowe czy obiekty. Również wówczas konieczne jest użycie nawiasów klamrowych. Ogólna zasada stanowi, że jeżeli zaraz za znakiem { występuje znak $, PHP najpierw obliczy wartość wyrażenia ciągnącego się aż do znaku }, po czym obliczony wynik dołączy do ciągu. (Jeżeli chcesz, by w ciągu pojawił się sam symbol {$, musisz w tym celu obydwa znaki poprzedzić znakiem lewego ukośnika).

Inne sposoby radzenia sobie z takimi sytuacjami zostaną opisane w punkcie „Złączenie i przypisanie" w dalszej części niniejszego rozdziału.

Znaki i indeksy ciągu

W przeciwieństwie do większości języków programowania, PHP nie posiada osobnego typu znakowego. Zwykle funkcje, które w innych językach pobierają znak, w PHP spodziewają się ciągu o długości 1.

Można uzyskać znak z ciągu, traktując ciąg jak tablicę z pierwszym indeksem równym 0, wpisanym w nawiasach klamrowych zaraz po zmiennej ciągu. Uzyskany w ten sposób znak jest jednoliterowym ciągiem. Na przykład:

```
$ciag = "Podwojony";
for ($index = 0 ; $index < 9; $index++ ) {
  $ciag_do_wyswietlenia = $ciag{$index};
  print("$ciag_do_wyswietlenia$ciag_do_wyswietlenia");
}
```

daje w efekcie:

```
PPooddwwoojjoonnyy
```

Każda litera została wypisana dwa razy (liczba 9 wpisana jest w tym przykładzie dlatego, że nie wiemy jeszcze, jak określić długość ciągu — popatrz na opis funkcji `strlen()` w części „Sprawdzanie ciągów").

We wcześniejszych wersjach PHP można było, zależnie od własnego uznania, umieszczać odpowiedni indeks w nawiasach kwadratowych i w taki sposób odczytywać pojedyncze znaki ciągu (np. `$ciag[3]` zamiast `$ciag{3}`). Wprawdzie cały czas można używać zapisu z nawiasami kwadratowymi (takiego samego jak w przypadku tablic), lecz mechanizm ten został już unieważniony i rekomendowane jest używanie do tego celu nawiasów klamrowych.

Operatory dla ciągów

PHP posiada tylko jeden prawdziwy operator dla ciągów: kropkę (.) — operator złączenia. Operator ten umieszczony pomiędzy ciągami tworzy nowy ciąg, będący ich sklejeniem. Na przykład:

```
$zdanie_1 = "Chciałbym podzielić się z wami ";
$zdanie_2 = "moimi uwagami";
print ($zdanie_1. $zdanie_2. "...");
```

da w efekcie:

```
Chciałbym podzielić się z wami moimi uwagami...
```

Zauważmy, że nie przekazujemy do instrukcji `print` wielu argumentów — przekazujemy jeden argument stworzony przez połączenie trzech ciągów. Pierwszy i drugi ciąg to zmienne, trzeci jest literałem.

Operator złączenia nie jest tym samym co operator + w języku Java, nie przeciąża innego operatora. Jeżeli pomylisz się i dodasz dwa ciągi za pomocą +, zostaną zinterpretowane jako liczby. Na przykład `jeden + dwa` będzie równe `0` (ponieważ nie udało się przeprowadzić prawidłowej konwersji ciągów).

Złączenie i przypisanie

Podobnie jak w przypadku operatorów arytmetycznych, PHP posiada operator skrócony (`.=`), który jest złączeniem z przypisaniem. Wyrażenie:

```
$ciag .= $dodatek;
```

jest równoznaczne z:

```
$ciag = $ciag . $dodatek;
```

Zwróć uwagę, że w odróżnieniu od naprzemiennego w swej naturze dodawania i mnożenia, przy użyciu tego operatora nowy ciąg jest dodawany po prawej stronie starego. Jeżeli chcesz zmienić kolejność, musisz użyć dłuższej formy:

```
$ciag = $dodatek . $ciag;
```

Trzeba również pamiętać, że w trakcie złączania zmienne bez przypisanej wartości są traktowane jak ciągi puste. Zmienna `$ciag` pozostanie zatem niezmieniona, jeżeli zmiennej `$dodatek` nie zostanie przypisana żadna zmienna.

Składnia heredoc

Oprócz składni z apostrofami i cudzysłowami PHP udostępnia również dodatkową metodę definiowania wartości ciągów — jest to tak zwana *składnia heredoc*. Składnia ta okazuje się szczególnie przydatna w sytuacji, gdy trzeba zdefiniować obszerne fragmenty tekstu zawierające w sobie wartości zmiennych, ponieważ programista nie musi w jej przypadku poprzedzać apostrofów ani cudzysłowów znakiem ukośnika. Składnia ta przydaje się przede wszystkim do tworzenia stron zawierających formularze HTML.

W składni heredoc operatorem jest `<<<`. Zaraz po nim powinna nastąpić etykieta (bez cudzysłowów ani apostrofów) wskazująca rozpoczęcie ciągu złożonego z wielu wierszy. PHP będzie dołączać do ciągu kolejne wiersze aż do czasu, gdy w nowym wierszu napotka tę samą etykietę. Opcjonalnie etykieta zamykająca może posiadać na końcu znak średnika.

Spójrzmy na poniższy przykład:

```
$my_string_var = <<<EOT
Wszystko, co znajduje się w tym mało przydatnym
fragmencie tekstu, zostanie dołączone
do zmiennej mającej postać ciągu znaków,
tworzonego znak po znaku, wiersz po wierszu, aż
do osiągnięcia wiersza ostatniego, czyli właśnie tego.
EOT;
```

Zwróć uwagę, że zamykająca etykieta `EOT` nie może być w jakikolwiek sposób wcięta, ponieważ zostanie wówczas potraktowana jak jeszcze jeden fragment, który powinien zostać dołączony do ciągu. Etykieta nie musi mieć postaci `EOT` — może to być dowolna nazwa zgodna z regułami nazewnictwa zmiennych w PHP.

Dołączanie zmiennych realizowane jest w sposób identyczny, jak w ciągach otoczonych cudzysłowami. Zaletą składni heredoc jest natomiast fakt, że znaki apostrofów i cudzysłowów mogą być dołączane bez poprzedzania ich znakiem ukośnika i bez konieczności wcześniejszego zakańczania ciągu. Spójrzmy na kolejny przykład:

```
echo <<<ENDOFFORM
<FORM METHOD=POST ACTION="{$_ENV['PHP_SELF']}">
<INPUT TYPE=TEXT NAME=FIRSTNAME VALUE=$firstname>
<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=SUBMIT>
</FORM>
ENDOFFORM
```

Powyższy kod w bardzo prosty sposób zwróci do przeglądarki kod definiujący najprostszy formularz.

Funkcje operujące na ciągach

PHP zawiera wiele funkcji działających na ciągach. Jeżeli zamierzasz napisać własną funkcję, która analizuje ciąg znak po znaku, aby stworzyć nowy ciąg, pomyśl, które z tych zadań będą często wykonywane. Jeżeli takie wystąpią, prawdopodobnie istnieje wbudowana funkcja realizująca tę czynność.

W tej części przedstawimy podstawowe funkcje sprawdzające, porównujące, modyfikujące i drukujące ciągi. Jeżeli chcesz naprawdę dobrze poznać operacje na ciągach wykonywane w PHP, powinieneś przyswoić sobie co najmniej treści przedstawione w tym punkcie. Funkcje wyrażeń regularnych oraz bardziej skomplikowane funkcje obsługi ciągów zostaną opisane w rozdziale 22.

Dla programistów C: powinniście znać wiele funkcji operujących na ciągach. Należy jedynie zapamiętać, że PHP zajmuje się przydziałem pamięci. Funkcje zwracające ciągi rezerwują pamięć na wynikowy ciąg i nie ma potrzeby wcześniej rezerwować pamięci.

Sprawdzanie ciągów

Jakie pytania można zadać na temat ciągu? Na początek sprawdźmy przy użyciu funkcji `strlen()`, ile znaków zawiera.

```
$ciag = "Ten ciąg zawiera 26 znaków";
print("To ma". strlen($ciag)."znaków");
```

Uruchomienie tego kodu daje następujący wynik:

```
To ma 26 znaków
```

Odczytanie długości ciągu jest użyteczne w sytuacjach, gdy chcemy za pomocą pętli dostać się do wszystkich znaków ciągu. Bezużytecznym, ale pouczającym przykładem może być (używamy ciągu z poprzedniego przykładu):

```
for ($index=0; $index <strlen ($ciag); $index++)
  print("$ciag[$index]");
```

W przeglądarce pojawi się ciąg taki sam, jak na początku:

```
Ten ciąg zawiera 26 znaków
```

Szukanie znaków i podciągów

Następną kategorią pytań na temat ciągów może być pytanie o ich zawartość. Funkcja `strpos()` szuka pozycji określonego znaku w ciągu, o ile w nim występuje.

```
$twister = "Król Karol kupił królowej Karolinie";
print("'k' występuje na pozycji ". strpos($twister, 'k'). "<BR>");
print("'q' występuje na pozycji". strpos($twister, 'q'). "<BR>");
```

Wykonanie tego przykładu daje w efekcie:

```
'k' występuje na pozycji 11
'q' występuje na pozycji
```

Pozycja litery `'q'` pozostała niewypełniona, ponieważ funkcja `strpos()` zwraca `FALSE` w przypadku nieznalezienia znaku, a `FALSE` zostało skonwertowane do pustego ciągu. Należy zwrócić uwagę, że funkcja `strpos()` odróżnia duże i małe litery.

Funkcja `strpos()` jest jednym z przykładów, kiedy brak ścisłego określenia typów może stanowić problem. Jeżeli nie zostanie odnaleziona wartość, funkcja zwraca wartość `FALSE`. Jeżeli odnaleziony zostanie pierwszy znak ciągu, funkcja zwróci zero (ponieważ indeksowanie rozpoczyna się od zera). Jeżeli użyjemy tych wartości w wyrażeniu logicznym, obie zostaną zinterpretowane jako `FALSE`. Jedynym sposobem rozróżnienia tych wartości jest użycie operatora sprawdzającego identyczność obiektów (`===`, wprowadzony w PHP4), który jest prawdziwy, jeżeli jego argumenty są identyczne i mają te same typy. Możesz go bezpiecznie użyć do porównania `0` z `FALSE`. Jeżeli używasz PHP3, musisz sprawdzić typ zwracanej wartości, używając funkcji np. `is_ integer()`.

Za pomocą funkcji `strpos()` można również szukać ciągów, nie tylko pojedynczych znaków. W tym celu wystarczy jedynie przekazać do niej ciąg wieloznakowy. Możesz także podać dodatkowy parametr numeryczny określający pozycję, od której funkcja ma rozpocząć szukanie.

Czy ciągi są niezmienne?

W niektórych językach programowania (np. C) normalne jest manipulowanie ciągami przez ich bezpośrednią modyfikację, na przykład zapisywanie nowych znaków w środku ciągu. Inne języki (np. Java) trzymają programistów z daleka od takich operacji, tworząc klasę reprezentującą ciągi, które są *niezmienne*. Można utworzyć nowy ciąg tylko poprzez stworzenie zmodyfikowanej kopii ciągu. Po jego utworzeniu nie można bezpośrednio zmieniać jego znaków.

W PHP ciągi mogą być zmieniane, ale w powszechnej praktyce jest traktowanie ciągów, jak gdyby były niezmienne.

Można zmieniać ciąg, traktując go jako tablicę znaków i przypisywać bezpośrednio do tej tablicy:

```
$ciag = "abcdefg";
$ciag[5] = "X";
print ($ciag . "<BR>");
```

Daje to następujący wynik:

```
abcdeXg
```

Taka operacja jest jednak nieudokumentowana i nie występuje nigdzie w podręczniku, chociaż analogiczne odczytywanie znaków jest zamieszczone (i uaktualnione o mechanizm nawiasów klamrowych). Prawie wszystkie funkcje PHP operujące na ciągach zwracają zmienioną kopię ciągu, a nie przeprowadzają zmian na oryginale. Projektanci PHP preferują ten styl programowania. Radzimy nie modyfikować bezpośrednio ciągów, chyba że dzięki temu oszczędza się pamięć.

Możliwe jest też szukanie wstecz przy użyciu funkcji `strrpos()` (dodatkowe `r` pochodzi od słowa „reverse”, czyli w tył). Funkcja ta pobiera ciąg do szukania oraz pojedynczy znak, którego szukamy, i zwraca ostatnią pozycję, na której w pierwszym argumencie występuje argument drugi. W przeciwieństwie do `strpos()`, nie można podać podciągu do odszukania. Jeżeli użyjemy tej funkcji w naszym przykładowym zdaniu, odszukamy inne wystąpienie litery `k`.:

```
$twister = "Król Karol kupił królowej Karolinie";
print("'k' występuje na pozycji ". strrpos($twister, 'k'). "<BR>");
```

W tym przypadku odnajdziemy pierwsze „k” w słowie „królowej”:

```
'k' występuje na pozycji 17
```

Porównywanie i przeszukiwanie

Często trzeba sprawdzać, czy dwa ciągi są identyczne. Szczególnie często w programach pracujących na danych wprowadzonych przez użytkownika.

Ciągi są takie same dla operatora `'=='`, jeżeli zawierają dokładnie taką samą sekwencję znaków. Operator ten nie wykonuje żadnych dokładniejszych porównań (np. sprawdzenia obszarów pamięci), zwraca natomiast baczną uwagę na wielkość liter.

Najprostszą metodą porównania ciągów jest zastosowanie operatora równości (==), który porównuje zarówno liczby, jak i ciągi.

Porównanie ciągów za pomocą operatora '==' (a także < i >) jest prawidłowe tylko w przypadku, gdy oba argumenty są takimi samymi ciągami i nie wykonano żadnej konwersji typów (więcej na ten temat w rozdziale 5.). Jedynie używając funkcji `strcmp()` (opisanej poniżej), otrzymujemy zawsze prawidłowe wyniki.

Podstawową funkcją do porównywania ciągów jest `strcmp()`. Posiada dwa ciągi jako argumenty, porównuje je znak po znaku aż do znalezienia różnicy. Funkcja zwraca wartość ujemną, jeżeli pierwszy ciąg jest „mniejszy" od drugiego, wartość dodatnią, gdy drugi ciąg jest mniejszy, oraz `0`, jeżeli ciągi są identyczne.

Funkcja `strcasecmp()` działa w taki sam sposób, z tą różnicą, że nie bierze pod uwagę wielkości liter podczas porównywania ciągów. Wywołanie funkcji: `strcasecmp("hej!", "HEJ!")` powinno zwrócić `0`.

Przeszukiwanie

Funkcje porównujące ciągi sprawdzają, czy ciągi są identyczne. Aby sprawdzić, czy jeden ciąg zawiera się w drugim, używamy funkcji `strpos()` (opiszemy ją później) lub `strstr()` (ewentualnie jednej z jej odmian).

Funkcja `strstr()` oczekuje w pierwszym parametrze ciągu do przeszukania, a w drugim ciągu do odszukania. Jeżeli operacja się powiedzie, funkcja zwróci fragment przeszukiwanego ciągu rozpoczynający się od pierwszego miejsca wystąpienia poszukiwanego fragmentu. Jeżeli nic nie zostanie odnalezione, zwracana jest wartość `FALSE`. W poniższym przykładzie zamieściliśmy jedno udane i jedno nieudane wywołanie funkcji.

```
$ciag_do_przeszukania = "pierwszaoladrugaola";
$ciag_do_odszukania = "ola";
print("Wynik szukania ciągu $ciag_do_odszukania: " .
       strstr($ciag_do_przeszukania, $ciag_do_odszukania) . "<br>");
$ciag_do_odszukania = "ula";
print("Wynik szukania ciągu $ciag_do_odszukania: " .
       strstr($ciag_do_przeszukania, $ciag_do_odszukania));
```

Wykonanie tego fragmentu da następujący wynik:

```
Wynik szukania ciągu ola: oladrugaola
Wynik szukania ciągu ula:
```

Puste miejsce za dwukropkiem w drugim wierszu wyniku jest wynikiem próby wydrukowania wartości `FALSE`, która została skonwertowana na pusty ciąg. Funkcja `strstr()` posiada alternatywną nazwę `strchr()`. Można używać dowolnej z nazw tej funkcji, wynik działania będzie identyczny. Podobnie jak `strcmp()`, `strstr()` posiada odmianę, która przy szukaniu ciągów identycznie traktuje małe i wielkie litery — `stristr()`. Funkcja ta działa identycznie jak `strstr()`, nie rozróżnia jednak wielkości liter. Dotychczas opisane funkcje operujące na ciągach zebraliśmy w tabeli 8.1.

Tabela 8.1. *Funkcje porównujące, przeszukujące i badające ciągi*

Funkcja	Opis
`strlen()`	Zwraca długość ciągu podanego jako argument wywołania i zwraca ją w postaci liczby całkowitej.
`strpos()`	Posiada dwa argumenty: ciąg do przeszukania i ciąg do znalezienia. Zwraca pozycję (licząc od zera) początku pierwszego miejsca wystąpienia ciągu lub wartość `FALSE`, jeżeli ciąg nie został odszukany. Można podać trzeci argument określający pozycję, od której ma rozpocząć szukanie.
`strrpos()`	Jak `strpos()`, ale przeszukuje ciąg od tyłu. Poszukiwany ciąg może składać się tylko z jednej litery. Nie ma opcjonalnych argumentów.
`strcmp()`	Spodziewa się dwóch ciągów jako argumentów. Zwraca `0`, jeżeli ciągi te są identyczne. Jeżeli funkcja znajdzie różnicę, zwraca wartość ujemną, jeżeli kod ASCII pierwszego różniącego się znaku jest mniejszy w pierwszym ciągu, lub wartość dodatnią, jeżeli mniejszy kod ASCII znajduje się w drugim ciągu.
`strcasecmp()`	Działa tak jak `strcmp()`, traktując identycznie małe i wielkie litery.
`strstr()`	Przeszukuje pierwszy ciąg, szukając w nim drugiego. Zwraca fragment pierwszego ciągu rozpoczynający się od poszukiwanego fragmentu. Jeżeli nie ma drugiego ciągu w pierwszym, zwraca `FALSE`.
`strchr()`	Działa tak samo jak `strstr()`.
`stristr()`	Działa tak samo jak `strstr()`, nie rozróżniając wielkości liter.

Wycinanie podciągu

Wiele z funkcji operujących na ciągach ma na celu wybranie określonego podciągu lub modyfikację go w ciągu oryginalnym. Warto wiedzieć, że większość funkcji modyfikujących ciąg nie zmienia oryginalnego ciągu, ale zwraca zmienioną kopię, pozostawiając oryginał nietknięty.

Podstawową metodą wycięcia fragmentu ciągu jest użycie funkcji `substr()`, która zwraca ciąg będący określonym fragmentem ciągu przekazanego jako argument. Oprócz ciągu, na którym operuje, funkcja wymaga podania liczby określającej początek wycinanego ciągu. Trzeci argument jest opcjonalny i określa długość wynikowego podciągu. Jeżeli nie zostanie on podany, funkcja zwraca fragment od podanej pozycji (za pomocą drugiego argumentu) do końca ciągu. Należy pamiętać, że pierwszy znak ciągu znajduje się na pozycji 0.

Spójrzmy na następujący przykład:

```
echo (substr("Wycinanie ciągów w PHP jest proste", 23));
```

Zwróci on w wyniku „jest proste”; wyrażenie:

```
echo (substr("Wycinanie ciągów w PHP jest proste", 10, 6));
```

wypisze słowo „ciągów” — sześcioznakowy ciąg rozpoczynający się od pozycji 10.

Oba argumenty numeryczne, pozycja początkowa i długość mogą być ujemne, lecz w obydwu przypadkach znaczenie ujemnego argumentu jest inne. Jeżeli pozycja początkowa jest ujemna, oznacza to, że początek podciągu jest określany względem końca ciągu. Pozycja `-1` oznacza początek ciągu na ostatnim znaku, `-2` na przedostatnim itd.

Można oczekiwać, że podanie ujemnej długości zinterpretowane będzie analogicznie do pozycji początkowej i podciąg będzie wyznaczany, odliczając wstecz od początkowego znaku ciągu. Nie jest to do końca prawdziwe. Znak określony przez indeks początkowy będzie pierwszym znakiem podciągu (a nie ostatnim), a jego długość określona będzie odliczeniem podanej liczby znaków od końca ciągu.

Spójrz na kilka przykładów, w których użyte zostały dodatnie i ujemne wartości:

```
$alfabet = "abcdefghijklmnop";
print ("3: ". substr( $alfabet, 3). "<BR>");
print ("-3: ". substr( $alfabet, -3). "<BR>");
print ("3,5: ". substr( $alfabet, 3, 5). "<BR>");
print ("3,-5: ". substr( $alfabet, 3, -5). "<BR>");
print ("-3,-5: ". substr( $alfabet, -3, -5). "<BR>");
print ("-3, 5: ". substr( $alfabet, -3, 5). "<BR>");
```

W wyniku otrzymujemy:

```
3: defghijklmnop
-3: nop
3,5: defgh
3,-5: defghijk
-3,-5:
-3, 5: nop
```

W przykładzie z pozycją początkową `-3` i długością `-5` pozycja końcowa znajduje się przed pozycją początkową, co jest sytuacją określaną jako „ciąg o ujemnej długości". W podręczniku, dostępnym pod adresem *http://www.php.net/manual*, jest napisane, że w takiej sytuacji funkcja `substr()` zwróci ciąg zawierający jeden znak znajdujący się na pozycji początkowej. Zamiast tego widzimy, że PHP5 zwraca pusty ciąg. Należy szczególnie zwrócić uwagę na takie sytuacje.

Zauważmy, że pomiędzy funkcjami `substr()`, `strstr()` i `strpos()` występuje bliski związek. Funkcja `substr()` wycina podciąg, bazując na pozycji, `strstr()` wycina podciąg na podstawie zawartości, a `strpos()` znajduje położenie podciągu. Jeżeli jesteśmy pewni, że `$ciag` zawiera `$podciag`, wtedy wyrażenie:

```
strstr($ciag, $podciag)
```

jest równoważne wyrażeniu:

```
substr($ciag, strpos($ciag, $podciag))
```

Funkcje porządkujące

Mimo że funkcje `chop()`, `ltrim()` i `trim()` są zwykłymi funkcjami wycinającymi podciąg, używa się ich do porządkowania ciągów. Funkcje te wycinają znaki odstępów z początku, końca lub początku i końca ciągu. A oto przykład:

```
$oryginal = " To przechodzi ludzkie pojęcie   ";
$f_chop = chop($oryginal);
$f_ltrim = ltrim($oryginal);
$f_trim = trim($oryginal);
print("Ciąg oryginalny: '$oryginal'<BR>");
print("Długość ciągu: ". strlen($oryginal). "<BR>");
print("Po funkcji chop: '$f_chop' <BR>");
print("Długość ciągu: ". strlen($f_chop). "<BR>");
print("Po funkcji ltrim: '$f_ltrim' <BR>");
print("Długość ciągu: ". strlen($f_ltrim). "<BR>");
print("Po funkcji trim: '$f_trim' <BR>");
print("Długość ciągu: ". strlen($f_trim). "<BR>");
```

W przeglądarce dostajemy:

```
Ciąg oryginalny: ' To przechodzi ludzkie pojęcie '
Długość ciągu: 33
Po funkcji chop: ' To przechodzi ludzkie pojęcie'
Długość ciągu: 30
Po funkcji ltrim: 'To przechodzi ludzkie pojęcie '
Długość ciągu: 32
Po funkcji trim: 'To przechodzi ludzkie pojęcie'
Długość ciągu: 29
```

Ciąg oryginalny ma trzy spacje na końcu (usuwane przez `chop()` i `trim()`) oraz jedną na początku (usuwana przez `ltrim()` i `trim()`). Dokładniej opiszemy zawartość okna przeglądarki po wykonaniu tego fragmentu. Powtarzające się odstępy zostały przez przeglądarkę połączone w jeden. Jeżeli jednak zajrzysz do źródła strony, to na końcu ciągu nadal znajdują się trzy odstępy.

Oprócz spacji funkcje te usuwają znaki zapisane jako sekwencje sterujące `\n`, `\r`, `\t` oraz `\0` (znaki końca wiersza, tabulatory oraz znak końca ciągu używany w programach napisanych w C).

Gdyby nazewnictwo funkcji było spójne, funkcja `chop()` powinna nazywać się `rtrim()`. Mimo że nazwa funkcji `chop()` (ang. *rąbać*) sugeruje destrukcyjne działanie, nie wpływa ona na ciąg przekazany jako argument. Po wykonaniu funkcji ciąg nie zmienia się.

Zastępowanie ciągów

Funkcje operujące na ciągach, które omówiliśmy do tej pory, zwracały fragment ciągu przekazanego jako argument, zamiast tworzyć całkowicie nowy ciąg. Teraz zajmiemy się funkcjami `str_replace()` i `substr_replace()`.

Funkcja `str_replace()` pozwala zamienić wszystkie miejsca wystąpienia podanego fragmentu ciągu na inny. Funkcja oczekuje trzech argumentów: ciągu do odszukania, ciągu, który zamieni szukany fragment, oraz ciągu źródłowego. Na przykład:

```
$pierwszy = "Birma jest nieco podobna do Rodezji.";
$drugi = str_replace("Rodezji", "Zimbabwe", $pierwszy);
$trzeci = str_replace("Birma", "Panama", $drugi);
print($trzeci);
```

Wynikiem tych operacji jest zdanie:

```
Panama jest nieco podobna do Zimbabwe.
```

Zastępowane są wszystkie miejsca wystąpienia podanego fragmentu. Jeżeli do ciągu PHP uda się wtłoczyć całą encyklopedię, można ją będzie zaktualizować jednym wywołaniem.

Musisz zwrócić uwagę na sytuację, gdy wystąpienia poszukiwanego ciągu zachodzą na siebie. Fragment programu:

```
$ciag = "ABA jest częścią ABABA";
$wynik = str_replace("ABA", "DEF", $ciag);
print("Wynik zamiany to '$wynik'<BR>");
```

wyświetli ciąg:

```
Wynik zamiany to 'DEF jest częścią DEFBA'
```

Jest to chyba najrozsądniejsze działanie.

Funkcja `str_replace()` odszukuje fragmenty ciągu do wymiany, porównując ciąg źródłowy z fragmentem podanym jako parametr; funkcja `substr_replace()` zamienia fragment ciągu znajdujący się na określonej pozycji. Można ją wywołać maksymalnie z czterema argumentami: ciągiem, na którym wykonuje się operacja, ciągiem, który jest wstawiany, początkową pozycją oraz parametrem opcjonalnym — długością fragmentu do wymiany. Na przykład:

```
print(substr_replace("ABCDEFG", "-", 2, 3);
```

daje w wyniku:

```
AB-FG
```

Fragment `CDE` został zamieniony na pojedynczy znak minus. Możemy więc zastępować fragmenty ciągu ciągami o innej długości. Jeżeli nie zostanie podana długość fragmentu, wymieniony zostanie fragment ciągu od pozycji startowej do końca ciągu.

Funkcja `substr_replace()` pozwala również na używanie ujemnych wartości argumentów numerycznych. Są one traktowane identycznie jak ujemne wartości argumentów w funkcji `substr()`; opisaliśmy je w części „Wycinanie podciągu”. Należy pamiętać, że funkcje `str_replace()` i `substr_replace()` nie zmieniają w żaden sposób oryginalnych ciągów.

Mamy również kilka rzadziej używanych funkcji, które tworzą nowy ciąg na podstawie podanego jako parametr. Funkcja `strrev()` zwraca ciąg podany jako argument, ale pisany wspak. Funkcja `str_repeat()` tworzy ciąg zawierający ciąg źródłowy powtórzony daną liczbę razy. Przykładowo:

```
print(str_repeat("witaj ", 3));
```

daje w wyniku:

```
witaj witaj witaj
```

Funkcje przeszukujące i zamieniające ciąg zebrane zostały w tabeli 8.2.

Tabela 8.2. *Funkcje przeszukujące i zamieniające ciąg*

Funkcja	Działanie
`substr()`	Zwraca fragment ciągu opisany przez drugi argument określający pozycję początkową i opcjonalny trzeci argument określający długość. Fragment rozpoczyna się na pozycji startowej i ma długość podaną w trzecim argumencie, a jeżeli nie został podany trzeci argument, obejmuje ciąg do końca. Ujemna wartość pozycji startowej oznacza, że jest określana od końca ciągu, ujemny parametr określający długość powoduje, że koniec podciągu jest wyznaczany przez podaną liczbę znaków od końca ciągu.
`chop()` lub `rtrim()`	Zwraca ciąg podany jako argument bez końcowych znaków odstępu. Znakami odstępu są znaki `" "`, `\n`, `\r`, `\t` i `\0`.
`ltrim()`	Zwraca ciąg podany jako argument bez początkowych znaków odstępu.
`trim()`	Zwraca ciąg podany jako argument bez początkowych i końcowych znaków odstępu.
`str_replace()`	Zamienia podany fragment ciągu na inny. Funkcja ma trzy argumenty: ciąg do odszukania, ciąg, na który jest on zamieniany, oraz ciąg bazowy. Zwraca kopię z zamienionymi na drugi argument wszystkimi miejscami wystąpienia pierwszego argumentu.
`substr_replace()`	Wstawia fragment podany jako argument na pozycję określoną przez parametry numeryczne. Funkcja posiada cztery argumenty: ciąg bazowy, fragment wstawiany, pozycję początkową i liczbę znaków do wymiany. Zwraca kopię ciągu podanego jako pierwszy argument, z ciągiem do zamiany wstawionym na określonej pozycji. Jeżeli opuszczony zostanie czwarty argument, koniec ciągu zostanie zamieniony na fragment przekazany w argumencie. Ujemne wartości parametrów traktowane są identycznie jak w funkcji `substr()`.

Funkcje zmiany wielkości liter

Funkcje poniższe zmieniają wielkość liter z małych na duże, i odwrotnie. Pierwsze dwie operują na całym ciągu, kolejne — tylko na pierwszych literach.

strtolower()

Funkcja `strtolower()` zwraca ciąg ze wszystkimi małymi literami. Nie ma znaczenia, czy na początku cały ciąg był zapisany dużymi literami, czy duże i małe litery były wymieszane.

```
<?php
$oryginalny = "On NIE wiE, że KRZYCZY";
$male = strtolower($oryginalny);
echo $male;
?>
```

Ten fragment programu zwróci ciąg: `"on nie wie, że krzyczy"`.

Jeśli już wcześniej zmagałeś się z formularzem, który wymagał zaimplementowania rozbudowanych mechanizmów weryfikacji poprawności danych, być może zauważyłeś, że funkcja `strtolower()` okazuje się niezwykle użyteczna dla osób trwających w przekonaniu, iż ich adresy pocztowe zawierają wielkie litery. Równie użyteczne są funkcje opisywane dalej w tym punkcie.

strtoupper()

Funkcja `strtoupper()` zwraca ciąg z wszystkimi dużymi literami. Nie ma znaczenia, czy na początku cały ciąg był zapisany małymi literami, czy duże i małe litery były wymieszane.

```
<?php
$oryginalny = "napisz to wyraźnie";
echo("<B>".strtoupper($oryginalny)."</B>");
?>
```

ucfirst()

Funkcja `ucfirst()` zmienia pierwszą literę ciągu na dużą.

```
<?php
$oryginalny = "przykładowe zdanie.";
echo(ucfirst($oryginalny));
?>
```

ucwords()

Funkcja `ucwords()` zamienia pierwsze litery wyrazów w ciągu na duże.

```
<?php
$oryginalny = "miasto nowy jork";
echo( ucwords($oryginalny));
?>
```

Zarówno `ucwords()`, jak i `ucfirst()` nie konwertują liter na małe. Zmieniają tylko właściwe początkowe litery na duże. Jeżeli w środku ciągu wystąpią duże litery, nie zostaną zamienione.

Funkcje znaków sterujących

Jedną z charakterystycznych cech PHP jest to, że potrafi się on komunikować niemal ze wszystkimi programami. PHP działa jako język łączący strony WWW z bazami danych, serwerami LDAP, połączeniami poprzez gniazda sieciowe czy protokół HTTP. Zwykle dialog taki jest realizowany dwuetapowo. Najpierw tworzony jest ciąg komunikatu (na przykład zapytanie do bazy danych), następnie jest wysyłany do programu, z którym realizowana jest komunikacja. Często program ten w specjalny sposób interpretuje niektóre znaki. Aby były one potraktowane jako część literału, a nie kod sterujący, należy je specjalnie oznaczyć.

Wielu użytkowników uaktywnia opcję *magic-quotes*, dzięki czemu znaki apostrofów i cudzysłowów automatycznie poprzedzane są znakiem ukośnika przed wstawieniem ciągów do bazy danych. Jeśli rozwiązanie takie okaże się jednak niewygodne lub niepożądane, istnieją też funkcje usuwające i dodające znaki `\`. Funkcja `addslashes()` oznacza apostrofy, cudzysłowy, lewe ukośniki oraz znaki `NULL` za pomocą lewego ukośnika. Zwykle jest to konieczne przy wysyłaniu znaków do bazy danych.

```
<?php
$escapedstring = addslashes("Po angielsku zdanie - 'I'm a dog.' znaczy 'Jestem psem.'");
$query = "INSERT INTO test (quote) values ('$escapedstring')";
$result = mysql_query($query) or die(mysql_error());
?>
```

Dzięki temu serwer SQL nie potraktuje apostrofu przed `"I"` jak zakończenia wyrażenia. Jeżeli odczytujesz dane z bazy, musisz użyć funkcji `stripslashes()`, aby usunąć znaki sterujące.

```
<?php
$query = "SELECT quote FROM test WHERE ID = 1";
$result = mysql_query($query) or die (mysql_error());
$new_row = mysql_fetch_array($result);
$quote = stripslashes($new_row[0]);

echo $quote;
?>
```

Funkcja `quotemeta()` przetwarza wszystkie znaki specjalnego znaczenia w poleceniach powłoki Unix: `'.'`, `'\'`, `'+'`, `'*'`, `'?'`, `'['`, `'^'`, `']'`, `'('`, `'$'`, `')'`.

Na przykład:

```
$literal= 'Znaki ($, *) mają dla mnie szczególne znaczenie \n<BR>';
$qm = quotemeta($literal);
echo $qm;
```

Wykonanie tego fragmentu da w wyniku:

```
Znaki \($, \*\) mają dla mnie szczególne znaczenie \\n
```

Funkcje sterujące specyficzne dla HTML opisano w punkcie „Zaawansowane funkcje obsługi ciągów” w rozdziale 22.

Formatowanie danych

Podstawowymi konstrukcjami używanymi do drukowania danych są `print` i `echo`, które zostały opisane w rozdziale 5. Zwykle wartości zmiennych są wypisywane poprzez wbudowywanie zmiennych w ciągi otoczone cudzysłowami (i są zamieniane na wartości) i przekazywanie takiego ciągu do instrukcji `echo` lub `print`.

Jeżeli potrzebujesz dokładniej sformatowanego tekstu, PHP udostępnia funkcje `printf()` i `sprintf()`, które działają analogiczne do funkcji o tych samych nazwach w C. Obie funkcje mają takie same argumenty: specjalny ciąg formatujący (opisany poniżej), a następnie dowolną liczbę argumentów, które zostaną wklejone we właściwe miejsca ciągu formatującego.

Jedyną różnicą pomiędzy `printf()` i `sprintf()` jest to, że `printf()` wysyła wynik bezpośrednio na wyjście, a `sprintf()` zwraca wynik jako ciąg.

Dla programistów C: funkcja `sprintf()` różni się od jej wersji z C jeszcze jednym szczegółem. W C należy utworzyć ciąg wynikowy, PHP sam utworzy ten ciąg.

Najbardziej skomplikowaną częścią tych funkcji jest ciąg formatujący. Każdy znak, jaki umieścisz w tym ciągu, pojawi się w ciągu wynikowym, oprócz sekwencji znaków rozpoczynających się od `%`. Znak `%` oznacza początek „specyfikacji konwersji", która wskazuje sposób wydruku argumentu odpowiadającego tej specyfikacji.

Po znaku `%` można umieścić pięć elementów określających sposób konwersji (niektóre opcjonalnie): wypełnienie, wyrównanie, minimalna szerokość, dokładność oraz typ.

- Pojedynczy (opcjonalny) znak wypełnienia może być zerem lub spacją. Znak ten jest używany do wypełniania nieużywanego miejsca, które zarezerwowano, określając minimalną szerokość. Jeżeli znak ten nie zostanie podany, wolne miejsce zostanie wypełnione spacjami.
- Opcjonalny znak *wyrównania* (`-`) wskazuje, czy wyświetlana wartość powinna być wyrównywana do strony prawej, czy do lewej. Jeżeli został umieszczony, wartość jest wyrównana do lewej strony; jeżeli go nie ma, wartość jest wyrównana do prawej strony.
- Opcjonalna liczba określająca *minimalną szerokość*; wartość zajmuje co najmniej tyle miejsca. Jeżeli będzie potrzebne więcej miejsca do wydrukowania wartości, przekroczona zostanie szerokość pola.
- Opcjonalne określenie szerokości części ułamkowej. Składa się z kropki i liczby określającej dokładność drukowania liczb rzeczywistych (dla innych wartości nie ma znaczenia).
- Pojedynczy znak określający sposób, w jaki ma być interpretowany typ zmiennej. Znak `f` oznacza drukowanie liczby rzeczywistej, `s` oznacza drukowanie ciągu, reszta znaków (`b`, `c`, `d`, `o`, `x`, `X`) oznacza, że wartość jest traktowana jak liczba całkowita i drukowana w różnych formatach. Formaty te to: format binarny dla litery `b`, `c` oznacza drukowanie znaku o odpowiednim kodzie ASCII, `o` oznacza liczbę oktalną, `x` to liczba heksagonalna (pisana małymi literami), a `X` to liczba heksagonalna pisana dużymi literami.

Przykładem może być drukowanie tej samej liczby rzeczywistej na różne sposoby:

```
<pre>
<?php
$wartosc = 3.14159;
printf("%f, %10f, %-010f, %2.2f\n", $wartosc, $wartosc, $wartosc, $wartosc);
?>
</pre>
```

Wynik działania tego fragmentu:

```
3.141590,          3.141590, 3.141590000000000,  3.14
```

Konstrukcja `<pre></pre>` w HTML wskazuje przeglądarce, aby nie formatowała tak oznaczonego bloku (nie usuwała wielokrotnych odstępów itp.).

Rozszerzony przykład: kalkulator do ćwiczeń

W niniejszym punkcie rozszerzymy przykładowy kalkulator do ćwiczeń utworzony w rozdziale 5., wykorzystując szereg funkcji operujących na ciągach do przetwarzania ciągów wysyłanych z formularza użytkownika. W poprzednim przykładzie udało nam się jedynie przekazać zmienną będącą ciągiem znaków z formularza HTML do strony PHP mającej za zadanie odebrać i obsłużyć ten ciąg. W tym punkcie zaimplementujemy mechanizmy analizy otrzymanego ciągu. (W końcowej części niniejszego punktu dowiesz się, dlaczego nasz przykład wymaga dalszego rozszerzania w następnych rozdziałach).

Listing 8.1 prezentuje kod formularza, w którym użytkownik ma wpisać wykonywane ćwiczenie. Jest on bardzo podobny do formularza zaprezentowanego w rozdziale 7. — zmianie uległa jedynie docelowa strona z kodem obsługującym formularz.

Listing 8.1. *Formularz początkowy*

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=150>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Kalkulator do ćwiczeń (przekazywanie ciągu znaków)</H1>
<P>Wpisz rozdzaj ćwiczenia, a kalkulator powie ci, jak długo powinieneś je wykonywać,
<BR>by spalić kilogram tłuszczu.</P>
<FORM METHOD="post" ACTION="wc_handler_str.php">
<INPUT TYPE="text" SIZE=50 NAME="exercise">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Spalaj się!">
</FORM>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Na listingu 8.2 przedstawiono zmienioną stronę obsługującą formularz, która wyświetla odpowiednie statystyki w zależności od ćwiczenia wpisanego przez użytkownika.

W poniższym kodzie celowo użyliśmy superglobalnej tablicy $_POST do odczytania wartości ciągu znaków przekazanego z formularza. Oznacza to jednak, że kod z listingu 8.2 nie będzie działać w PHP w wersji wcześniejszej niż 4.1. Aby przystosować skrypt do wcześniejszej wersji języka, należy zamiast $_POST użyć $HTTP_POST_VARS, pamiętając jednak, że obecnie długie nazwy zmiennych są oficjalnie nieważne i w którymś momencie przestaną być w ogóle obsługiwane.

Listing 8.2. *Skrypt obsługi formularza przy użyciu funkcji ciągów*

```
<?php

$excercise_str = $_POST['exercise'];    // Uwaga! Działa tylko w PHP 4.1+
// Upewniamy się, że użytkownik przestrzega reguł
if (strlen($excercise_str) > 50) {
  echo "Nie stosujesz się do reguł. Wstydź się!";
  exit;
}
// Próba przetworzenia ciągu wejściowego
// -------------------------------------
// Usunięcie ewentualnych spacji na początku i na końcu ciągu
$excercise_str = trim($excercise_str);
// Przekształcenie wszystkich liter w małe, by ułatwić porównywanie ciągów
$excercise_str = strtolower($excercise_str);

// Przypisanie liczby godzin, przez jaką trzeba wykonywać dane ćwiczenie,
// by spalić kilogram tłuszczu
if ($excercise_str == 'jazda na rowerze' || $excercise_str == 'jazda na rowerze górskim') {
  $hours = '5 godzin i 40 minut';
} elseif ($excercise_str == 'bieg' ||
          $excercise_str == 'jogging') {
  $hours = '4 godziny i 30 minut';
} elseif ($excercise_str == 'piłka nożna' ||
          $excercise_str == 'futbol') {
  $hours = '4 godziny i 30 minut';
} elseif ($exercise_str == 'aerobik') {
  $hours = '7 godzin i 30 minut';
} else {
  // Wyzerowanie wszystkich innych ćwiczeń
  $excercise_str = '';
  $hours = '';
}

// Utworzenie zdania
// -----------------

if ($excercise_str != "" && $hours != "") {
  $message = 'Jeden kilogram tłuszczu można spalić, gdy ' . $excercise_str . ' jest
uprawiany(a) ' .
             'przez ' . $hours . '.';
} elseif ($excercise_str == "" && $hours == "") {
  // Jeżeli ćwiczenia nie ma na naszej liście, wyświetlany jest
  // domyślny komunikat.
  $message = 'Brak danych dla podanego ćwiczenia.';
} else {

  // Istnieją jeszcze dwie możliwości
  // 1. Rozpoznajemy ćwiczenie, lecz nie mamy dla niego statystyki
  // 2. Nie rozpoznajemy ćwiczenia, ale jakimś cudem mamy dla niego statystyki
  // Żadna z nich nie powinna się zdarzyć, ale na wszelki wypadek...
  $message = 'Coś poszło bardzo źle!';
}
// Utworzenie strony
// -----------------
$page_str = <<< EOPAGE
<HTML>
<HEAD>
<STYLE TYPE="text/css">
```

```
<!--
BODY, P   {color: black; font-family: verdana; font-size: 10 pt}
H1        {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=150>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Obsługa kalkulatora do ćwiczeń - część 2</H1>
<P>Kalkulator informuje: "$message"</P>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
EOPAGE;

echo $page_str;

?>
```

Kod z listingu 8.2 wykonuje następujące czynności:

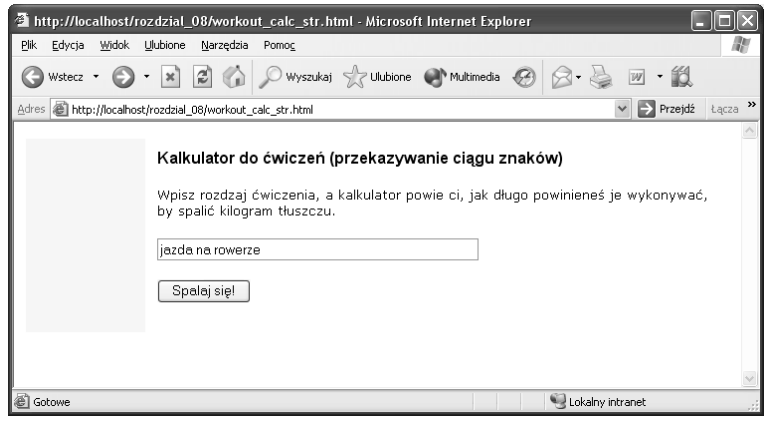
1. Odczytuje ciąg znaków przekazany z formularza HTML.
2. Próbuje doprowadzić otrzymany ciąg znaków do standardowej postaci, usuwając znaki spacji z początku i końca, przekształcając wszystkie litery na małe i rozpoznając niektóre synonimy.
3. Na podstawie oczyszczonego i odpowiednio przekształconego ciągu znaków wyszukuje statystyki dla podanego ćwiczenia.
4. Tworzy stronę z odpowiedzią przy użyciu składni heredoc.

Rysunki 8.1 i 8.2 przedstawiają efekt działania kalkulatora, gdy użytkownik wpisze jazdę na rowerze.

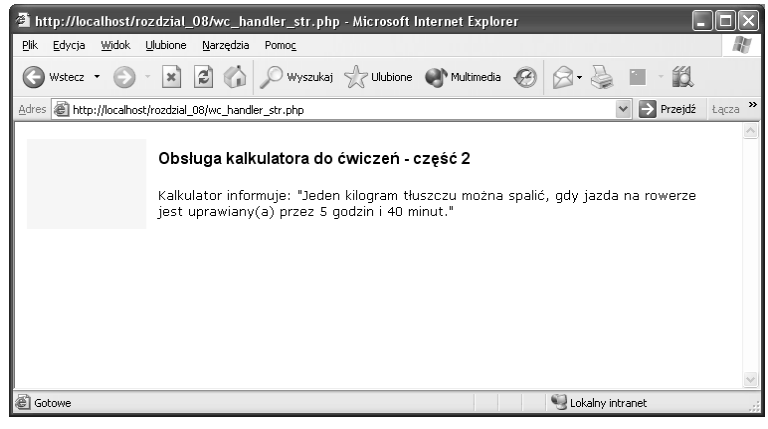
Nie znudził Ci się jeszcze kalkulator do ćwiczeń? Główny problem polega na tym, że nasz skrypt najprawdopodobniej nie będzie wiedział, jak obsłużyć różne dyscypliny sportu wpisywane przez użytkowników w sposób inny, niż wyświetlanie komunikatu domyślnego. Ponadto jeśli podniesiesz poziom raportowania błędów do `E_ALL`, wyświetlane będą ostrzeżenia informujące o braku inicjacji zmiennych. Użytkownik nie otrzymuje również żadnych wskazówek odnośnie dyscyplin, jakie rozpoznaje kalkulator.

Podsumowując, dopóki nie utworzymy systemu (takiego jak wyszukiwarka), który będzie umiał sobie radzić z dowolnymi tekstami, generowanie przez kod wyników w zależności od wartości wpisanych przez użytkowników mija się w praktyce z celem. Jeśli użytkownik ma dokonywać wyborów, należy zakres dostępnych wartości odpowiednio ograniczyć. Tekstu wpisywanego przez użytkownika można używać wówczas, gdy ma on być przeglądany przez innego użytkownika (np. zapisywany w tym celu do bazy danych lub przesyłany pocztą elektroniczną).

Rysunek 8.1. *Formularz początkowy*



Rysunek 8.2. *Odpowiedź*



Aby elegancko obsługiwać wartości, które może wybierać użytkownik, powinno się używać innych elementów interfejsu HTML niż pola tekstowe — np. pól wyboru, pól opcji czy list rozwijanych. Z kolei aby móc korzystać z tych elementów interfejsu po stronie klienta, w większym stopniu trzeba zacząć używać tablic. W tym właśnie kierunki pójdziemy w rozdziale 9.

Podsumowanie

Ciągi są sekwencjami znaków. Jest to jeden z sześciu podstawowych typów danych w PHP. W przeciwieństwie do niektórych innych języków, nie ma tu osobnego typu znakowego, pojedynczy znak zachowuje się jak ciąg o długości 1. Ciągi w kodzie są otaczane apostrofami (') lub cudzysłowami ("). Ciągi otoczone apostrofami są interpretowane prawie dosłownie, a ciągi otoczone cudzysłowami interpretują kilka sekwencji sterujących i automatycznie wstawiają wartości zmiennych.

Podstawowym operatorem ciągów jest ' . ', który łączy dwa ciągi. Istnieje dodatkowo spora gama funkcji, które umożliwiają sprawdzanie, porównywanie, szukanie, wycinanie, zamianę zawartości ciągu. PHP udostępnia także wyrażenia regularne, zgodne ze standardem POSIX i Perl (i opisane w rozdziale 22.), do zaawansowanych zastosowań.