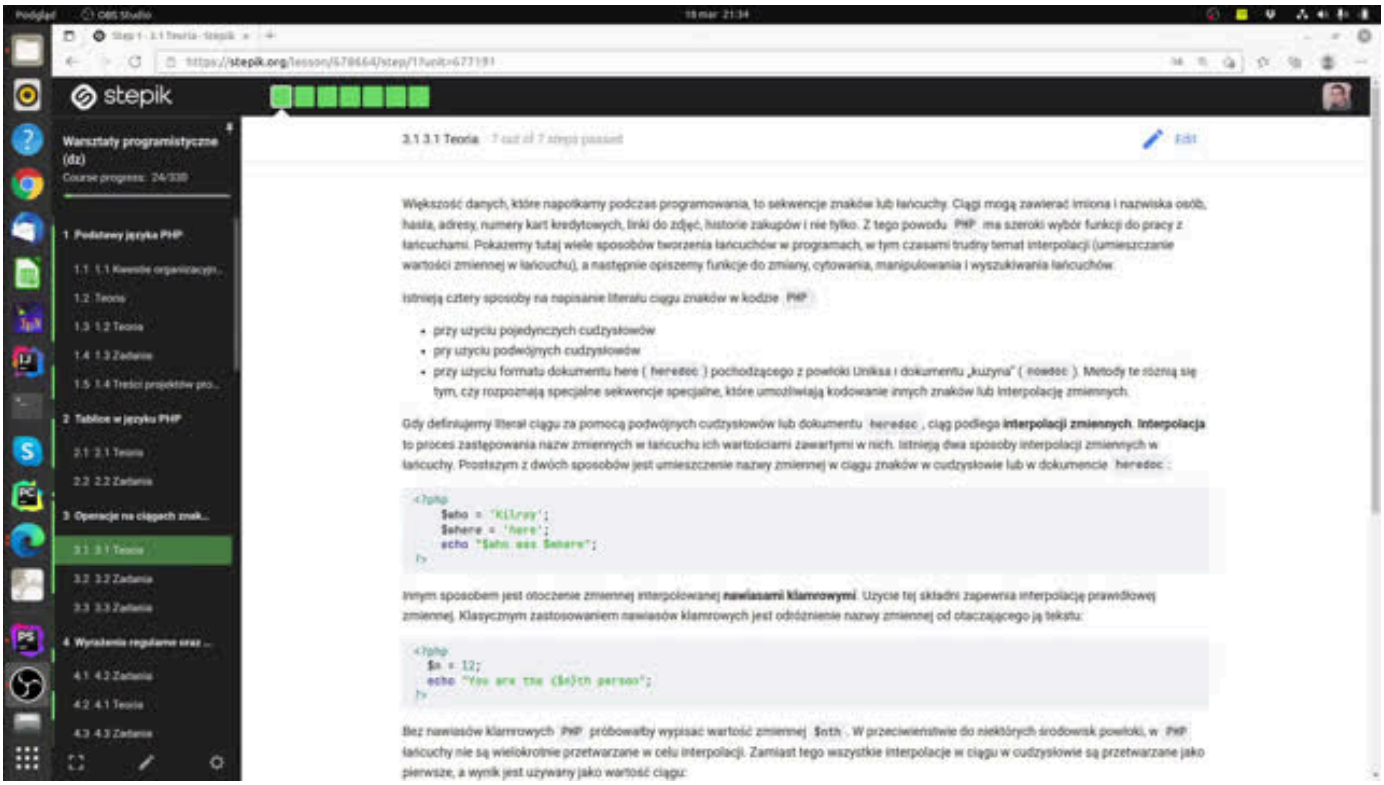
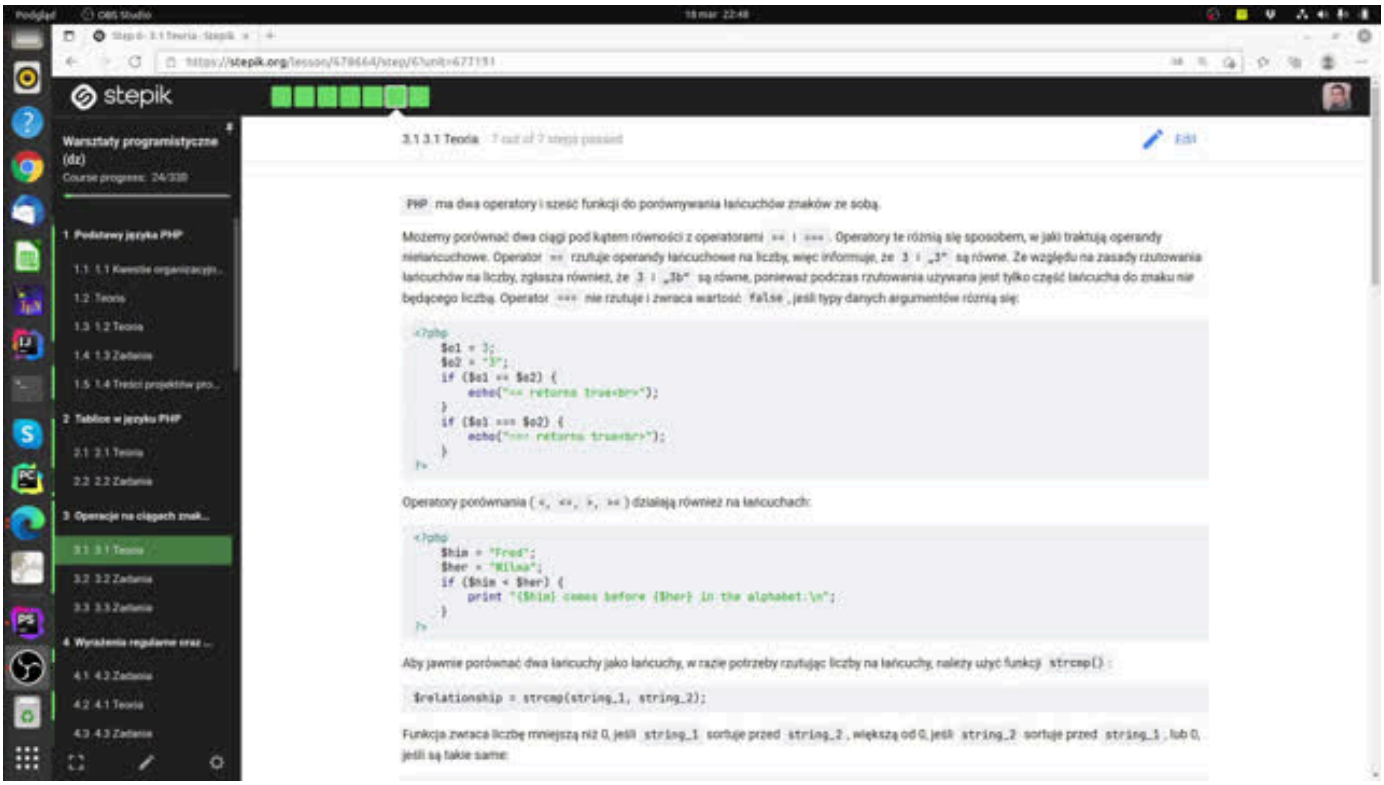
9.1 9.1 Teoria

Step 1



To watch this video please visit https://stepik.org/lesson/678664/step/1

Step 2



To watch this video please visit https://stepik.org/lesson/678664/step/2

Step 3

Większość danych, które napotkamy podczas programowania, to sekwencje znaków lub łańcuchy. Ciągi mogą zawierać imiona i nazwiska osób, hasła, adresy, numery kart kredytowych, linki do zdjęć, historie zakupów i nie tylko. Z tego powodu `PHP` ma szeroki wybór funkcji do pracy z łańcuchami. Pokażemy tutaj wiele sposobów tworzenia łańcuchów w programach, w tym czasami trudny temat interpolacji (umieszczanie wartości zmiennej w łańcuchu), a następnie opiszemy funkcje do zmiany, cytowania, manipulowania i wyszukiwania łańcuchów.

Istnieją cztery sposoby na napisanie literału ciągu znaków w kodzie `PHP`:

- przy użyciu pojedynczych cudzysłowów
- pry użyciu podwójnych cudzysłowów
- przy użyciu formatu dokumentu here (`heredoc`) pochodzącego z powłoki Uniksa i dokumentu „kuzyna” (`nowdoc`). Metody te różnią się tym, czy rozpoznają specjalne sekwencje specjalne, które umożliwiają kodowanie innych znaków lub interpolację zmiennych.

Gdy definiujemy literał ciągu za pomocą podwójnych cudzysłowów lub dokumentu `heredoc`, ciąg podlega **interpolacji zmiennych**. **Interpolacja** to proces zastępowania nazw zmiennych w łańcuchu ich wartościami zawartymi w nich. Istnieją dwa sposoby interpolacji zmiennych w łańcuchy. Prostszym z dwóch sposobów jest umieszczenie nazwy zmiennej w ciągu znaków w cudzysłowie lub w dokumencie `heredoc`:

```
<?php
    $who = 'Kilroy';
    $where = 'here';
    echo "$who was $where";
?>
```

Innym sposobem jest otoczenie zmiennej interpolowanej **nawiasami klamrowymi**. Użycie tej składni zapewnia interpolację prawidłowej zmiennej. Klasycznym zastosowaniem nawiasów klamrowych jest odróżnienie nazwy zmiennej od otaczającego ją tekstu:

```
<?php
  $n = 12;
  echo "You are the {$n}th person";
?>
```

Bez nawiasów klamrowych `PHP` próbowałby wypisać wartość zmiennej `$nth`. W przeciwieństwie do niektórych środowisk powłoki, w `PHP` łańcuchy nie są wielokrotnie przetwarzane w celu interpolacji. Zamiast tego wszystkie interpolacje w ciągu w cudzysłowie są przetwarzane jako pierwsze, a wynik jest używany jako wartość ciągu:

```
<?php
    $bar = 'this is not printed';
    $foo = '$bar';
    print("$foo");
?>
```

Step 4

Ciągi w pojedynczym cudzysłowie i nowdocs nie interpolują zmiennych. Zatem nazwa zmiennej w następującym ciągu nie jest interpretowana, ponieważ literał ciągu, w którym występuje, jest ujęty w pojedynczy cudzysłów:

```
<?php
    $name = 'Fred';
    $str = 'Hello, $name'; // single-quoted
    echo $str;
?>
```

Jedyne sekwencje specjalne, które działają w łańcuchach w pojedynczym cudzysłowie, to `\'`, który umieszcza pojedynczy cudzysłów w pojedynczym cudzysłowie, oraz `\\`, który wstawia odwrotny ukośnik w pojedynczym łańcuchu. Każde inne wystąpienie ukośnika odwrotnego jest interpretowane po prostu jako ukośnik odwrotny:

```
<?php
    $name = 'Matthew O\'Reilly';
    echo $name . "\n";
    $path = 'C:\\WINDOWS';
    echo $path . "\n";
    $nope = '\n';
    echo $nope;
?>
```

Łańcuchy w podwójnych cudzysłowach interpolują zmienne i rozszerzają liczne sekwencje specjalne PHP. Sekwencje specjalne przedstawia poniższa tabela:

Znak specjalny	Reprezentacja znaku specjalnego
`"`	`\"`
Nowa linia	`\n`
Znak powrotu karetki	`\r`
Tabulator	`\t`
`\`	`\\`
`$`	`\$`
`{`	`\{`
`}`	`\}`
`[`	`\[`
`]`	`\]`
Znak ASCII w systemie ósemkowym	`\0` - `\777`
Znak ASCII w systemie szesnastkowym	`\x0` - `\xFF`
Kodowanie UTF-8	`\u`

Jeśli umieścimy znak nie występujący w tabeli powyżej to, zostanie ona zignorowana (jeśli ustawiono poziom ostrzeżenia `E_NOTICE`, dla takich nieznanych sekwencji ucieczki generowane jest ostrzeżenie):

```
<?php
    $str = "What is \c this?";
    echo $str;
?>
```

Możemy łatwo umieścić w swoim programie ciągi wielowierszowe za pomocą dokumentu `heredoc`, w następujący sposób:

```
<?php
$clerihew = <<< EndOfQuote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
EndOfQuote;
echo $clerihew;
?>
```

Token identyfikatora `<<<` informuje parser `PHP`, że piszemy dokument `heredoc`. Możemy wybrać identyfikator (w tym przypadku `EndOfQuote`) i jeśli chcemy, możemy umieścić go w cudzysłowie (np. `„EndOfQuote”`). Następna linia rozpoczyna tekst cytowany przez dokument `heredoc`, który trwa aż do linii zawierającej tylko identyfikator. Aby upewnić się, że cytowany tekst jest wyświetlany w obszarze wyjściowym dokładnie tak, jak go ułożyliśmt, należy włączyć tryb zwykłego tekstu, dodając to polecenie na górze pliku kodu:

```
<?php
header('Content-Type: text/plain;');
$clerihew = <<< EndOfQuote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
EndOfQuote;
echo $clerihew;
?>
```

Alternatywnie, jeśli masz kontrolę nad ustawieniami swojego serwera, możesz ustawić `default_mimetype` na zwykły w pliku `php.ini`:

```
default_mimetype = "text/plain"
```

Nie jest to jednak zalecane, ponieważ wszystkie dane wyjściowe z serwera są umieszczane w trybie zwykłego tekstu, co wpłynęłoby na układ większości kodu internetowego. Jeśli nie ustawimy trybu zwykłego tekstu dla dokumentu `heredoc`, domyślnym jest zwykle tryb `HTML`, który po prostu wyświetla dane wyjściowe w jednym wierszu. Używając `heredoc` dla prostego wyrażenia, możemy umieścić średnik po końcowym identyfikatorze, aby zakończyć instrukcję (jak pokazano w pierwszym przykładzie). Jeśli jednak używamy dokumentu `heredoc` w bardziej złożonym wyrażeniu, musimy kontynuować wyrażenie w następnym wierszu:

```
<?php
header('Content-Type: text/plain;');
printf(<<< Template
%s is %d years old.
Template
    , "Fred", 35);
?>
```

Zachowane są pojedyncze i podwójne cudzysłowy w dokumencie `heredoc`:

```
<?php
$dialogue = <<< NoMore
"It's not going to happen!"
He raised an eyebrow. "Want
NoMore;
echo $dialogue;
?>
```

Podobnie jak białe znaki:

```
<?php
$ws = <<< Enough
 boo
 hoo
Enough;
echo $ws;
?>
```

Nowością w `PHP 7.3` jest wcięcie terminatora `heredoc`. Pozwala to na bardziej czytelne formatowanie w przypadku kodu osadzonego, jak w funkcji:

```
<?php
function sayIt() {
    $ws = <<< "StufftoSay"
    The quick brown fox
    Jumps over the lazy dog.
    StufftoSay;
    return $ws;
}
echo sayIt() ;
?>
```

Nowy wiersz przed końcowym terminatorem jest usuwany, więc poniższe przypisania są identyczne:

```
<?php
$s = 'Foo';
echo "$s\n";
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
echo "$s";
?>
```

Step 5

Istnieją cztery sposoby wysyłania danych wyjściowych do przeglądarki/konsoli. Konstrukcja `echo` umożliwia wydrukowanie wielu wartości naraz, podczas gdy `print()` wypisuje tylko jedną wartość. Funkcja `printf()` buduje sformatowany ciąg przez wstawienie wartości do szablonu. Funkcja `print_r()` jest przydatna do debugowania; drukuje zawartość tablic, obiektów i innych rzeczy w mniej lub bardziej czytelnej dla człowieka formie.

Aby umieścić ciąg w kodzie HTML strony wygenerowanej w `PHP`, można użyć polecenia `echo`. Chociaż wygląda — i w większości zachowuje się — jak funkcja, `echo` jest konstrukcją językową. Oznacza to, że możemy pominąć nawiasy, więc poniższe wyrażenia są równoważne:

```
<?php
    echo "Printy\n";
    echo("Printy");
?>
```

Możemy określić wiele pozycji do wydrukowania, oddzielając je znakiem przecinka:

```
<?php
    echo "First", "second", "third";
?>
```

Używanie nawiasów podczas wywołania polecenia echo dla wielu wartości może wywołać tzw. `Syntax Error`:

```
<?php
    echo("Hello", "world");
?>
```

Ponieważ echo jest poleceniem (nie funkcją), nie można go używać jako części większego wyrażenia:

```
<?php
    if (echo("test")) {
        echo("It worked!");
    }
?>
```

Możesz łatwo naprawić takie błędy, używając funkcji `print()` lub `printf()`.

Funkcja `print()` wysyła jedną wartość (jej argument) do przeglądarki:

```
<?php
    if (print("test\n")) {
        print("It worked!");
    }
?>
```

Funkcja `printf()` wyprowadza ciąg znaków zbudowany przez podstawienie wartości do szablonu (ciąg formatu). Wywodzi się z funkcji o tej samej nazwie w standardowej bibliotece `C`. Pierwszym argumentem `printf()` jest **ciąg formatu**. Pozostałe argumenty to wartości do zastąpienia. Znak `%` w ciągu formatu wskazuje na podstawienie. Każdy znacznik podstawienia w szablonie składa się ze znaku procentu (`%`), po którym mogą występować modyfikatory z poniższej listy i kończy się specyfikatorem typu. (Użycie `%%`, wypisuje pojedynczy znak procentu w wyniku.) Modyfikatory muszą pojawić się w kolejności, w jakiej są tutaj wymienione:

1. Specyfikator dopełnienia określający znak, który ma zostać użyty do wypełnienia wyników do odpowiedniego rozmiaru ciągu. Podanie 0, spacji lub dowolnego znaku poprzedzonego pojedynczym cudzysłowem. Dopełnienie ze spacjami jest domyślne.

2. Znak. Ma to inny wpływ na ciągi niż na liczby. W przypadku ciągów znak minus (`-`) wymusza wyrównanie ciągu do lewej (domyślnie wyrównanie do prawej). W przypadku liczb znak plus (`+`) wymusza wypisanie liczb dodatnich z wiodącym znakiem plus (np. `35` zostanie wydrukowane jako `+35`).
3. Minimalna liczba znaków, jaką powinien zawierać ten element. Jeśli wynik byłby mniejszy niż ta liczba znaków, specyfikator znaku i dopełnienia określa sposób dopełnienia do tej długości.
4. W przypadku liczb zmiennoprzecinkowych specyfikator dokładności składający się z kropki i liczby; mówi, ile cyfr dziesiętnych zostanie wyświetlonych. W przypadku typów innych niż double ten specyfikator jest ignorowany.

Specyfikator typu informuje `printf()` , jaki typ danych jest zastępowany. To determinuje interpretację wcześniej wymienionych modyfikatorów. Istnieje osiem rodzajów wymienionych na liście poniżej:

Specyfikator	Opis specyfikatora
`%`	Znak `%`
`b`	Argument jest liczbą całkowitą wypisywaną jako liczba binarna
`c`	Argument jest liczbą całkowitą wypisywaną jako znak
`d`	Argument jest liczbą całkowitą wypisywaną jako liczba dziesiętna
`e`	Argument jest liczbą zmiennoprzecinkową wyświetlony w notacji naukowej
`E`	Argument jest liczbą zmiennoprzecinkową wyświetlony w notacji naukowej dużymi literami
`g`	Argument jest liczbą zmiennoprzecinkową wyświetlony w notacji naukowej lub jest w formacie `%f`
`G`	Argument jest liczbą zmiennoprzecinkową wyświetlony w notacji naukowej dużymi literami lub jest w formacie `%f`
`o`	Argument jest liczbą całkowitą wypisywaną w postaci ósemkowej
`s`	Argument jest ciągiem znaków
`u`	Argument jest liczbą całkowitą (dodatnią) wyświetlany jako liczba dziesiętna
`x`	Argument jest liczbą całkowitą wyświetlaną w postaci szesnastkowej
`X`	Argument jest liczbą całkowitą wyświetlaną w postaci szesnastkowej (dużymi literami)
`%f`	Argument jest liczbą zmiennoprzecinkową wyświetlany w lokalnym formacie
`%F`	Argument jest liczbą zmiennoprzecinkową

Funkcja `printf()` wydaje się skandalicznie złożona dla ludzi, którzy nie są programistami `C` . Kiedy jednak się do tego przyzwyczai, okaże się, że jest to potężne narzędzie do formatowania.

```
<?php
    printf("%.2f\n", 27.452);
    printf("The hex value of %d is %x\n", 214, 214);
    printf("Bond. James Bond. %03d.\n", 7);
    printf("%02d/%02d/%04d\n", 4, 9, 1991);
    printf("%.2f%% Complete\n", 2.1);
    printf("You\'ve spent $%5.2f so far", 4.1);
?>
```

Funkcja `sprintf()` przyjmuje te same argumenty co `printf()` , ale zwraca wbudowany ciąg zamiast go wypisywać. Pozwala to zapisać ciąg w zmiennej do późniejszego wykorzystania:

```
<?php
    $date = sprintf("%02d/%02d/%04d\n", 4, 9, 1991);
    print($date);
?>
```

Funkcja `print_r()` inteligentnie wyświetla to, co jest do niej przekazywane, zamiast rzutować wszystko na łańcuch, jak robią to `echo` i `print()`. Ciągi i liczby są po prostu drukowane. Tablice są wyświetlane jako ujęte w nawiasy listy kluczy i wartości, poprzedzone przedrostkiem `Array`:

```
<?php
    $a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
    print_r($a);
?>
```

Użycie `print_r()` na tablicy przenosi wewnętrzny iterator na pozycję ostatniego elementu w tablicy.

Kiedy `print_r()` zostanie wywołane na obiekcie, zobaczymy słowo `Object`, po którym następują zainicjowane właściwości obiektu wyświetlane jako tablica:

```
<?php
    class P {
        var $name = 'nat';
    }
    $p = new P;
    print_r($p);
?>
```

Wartości logiczne i `NULL` nie są jednak sensownie wyświetlane przez `print_r()`:

```
<?php
    print_r(true);
    print_r(false);
    print_r(null);
?>
```

Z tego powodu funkcja `var_dump()` jest preferowana w celu debugowania niż `print_r()`. Funkcja `var_dump()` wyświetla dowolną wartość `PHP` w formacie czytelnym dla człowieka:

```
<?php
    var_dump(true);
    var_dump(false);
    var_dump(null);
    var_dump(array('name' => "Fred", 'age' => 35));
    class P {
        var $name = 'Nat';
    }$p = new P;
    var_dump($p);
?>
```

Należy uważać na używanie `print_r()` lub `var_dump()` na strukturze rekurencyjnej, takiej jak `$GLOBALS` (która ma wpis dla `GLOBALS`, który wskazuje na siebie). Funkcja `print_r()` zapętla się w nieskończoność, podczas gdy `var_dump()` wyłącza się po trzykrotnym odwiedzeniu tego samego elementu.

Step 6

Funkcja `strlen()` zwraca liczbę znaków w ciągu:

```
<?php
    $string = 'Hello, world';
    $length = strlen($string);
    print($length);
?>
```

Możemy użyć składni przesunięcia ciągu w ciągu, aby adresować poszczególne znaki:

```
<?php
    $string = 'Hello';
    for ($i=0; $i < strlen($string); $i++) {
        printf("The %dth character is %s\n", $i, $string[$i]);
    }
?>
```

Często ciągi, które otrzymujemy z plików lub użytkowników, muszą zostać oczyszczone, zanim będziemy mogli z nich skorzystać. Dwa typowe problemy z nieprzetworzonymi danymi to obecność dodatkowych białych znaków i nieprawidłowa wielkość liter (wielkie lub małe). Możemy usunąć początkowe lub końcowe białe znaki za pomocą funkcji `trim()`, `ltrim()` i `rtrim()`:

```
$trimmed = trim(string [, charlist ]);
$trimmed = ltrim(string [, charlist ]);
$trimmed = rtrim(string [, charlist ]);
```

Funkcja `trim()` zwraca kopię ciągu z usuniętymi białymi znakami z początku i końca. `ltrim()` (`l` oznacza lewo) robi to samo, ale usuwa białe znaki tylko z początku łańcucha. `rtrim()` (`r` oznacza prawo) usuwa białe znaki tylko z końca łańcucha. Opcjonalny argument `charlist` to łańcuch, który określa wszystkie znaki do usunięcia. Domyślne znaki używane w wyżej wymienionych funkcjach są zestawione w tabeli poniżej:

Znak	Wartość ASCII	Znaczenie
`" "`	`0x20`	Spacja
`"\t"`	`0x09`	Tabulator poziomy
`"\n"`	`0x0A`	Nowa linia
`"\r"`	`0x0D`	Znak powrotu karetki
`"\0"`	`0x00`	Znak NULL
`"\x0B"`	`0x0B`	Tabulator pionowy

```
<?php
    $title = " Programming PHP \n";
    $str1 = ltrim($title);
    $str2 = rtrim($title);
    $str3 = trim($title);
    print($str1);
    print($str2);
    print($str3);
?>
```

Mając wiersz danych oddzielonych tabulatorami, należy użyć argumentu `charlist`, aby usunąć początkowe lub końcowe białe znaki bez usuwania tabulatorów:

```
<?php
    $record = " Fred\tFlintstone\t35\tWilma\t \n";
    print($record);
    $record = trim($record, " \r\n\0\x0B");
    print($record);
?>
```

`PHP` ma kilka funkcji do zmiany wielkości liter: `strtolower()` i `strtoupper()` działają na całych łańcuchach, `ucfirst()` działa tylko na pierwszym znaku ciągu, a `ucwords()` działa na pierwszym znaku każdego słowa w ciągu znaków. Każda funkcja przyjmuje jako argument ciąg, na którym ma działać, i zwraca kopię tego ciągu, odpowiednio zmienioną.

```
<?php
    $string1 = "FRED flintstone\n";
    $string2 = "barney rubble\n";
    print(strtolower($string1));
    print(strtoupper($string1));
    print(ucfirst($string2));
    print(ucwords($string2));
?>
```

Jeśli mamy ciąg znaków o różnej wielkości, który chcemy przekonwertować na format, gdzie pierwsza litera każdego słowa jest wielka, a pozostałe litery małymi (i nie jesteśmy pewien, jaka wielkość liter string jest na początek), należy użyć kombinacji `strtolower()` i `ucwords()`:

```
<?php
    $string1 = "FRED flintstone\n";
    print(ucwords(strtolower($string1)));
?>
```

Step 7

Ponieważ programy `PHP` często wchodzą w interakcję ze stronami `HTML`, adresami internetowymi (`URL`) i bazami danych, istnieją funkcje ułatwiające pracę z tego typu danymi. `HTML`, adresy internetowe i polecenia bazy danych to ciągi znaków, ale każdy z nich wymaga różnych znaków, które mają zostać zmienione na różne sposoby. Na przykład spacja w adresie internetowym musi być zapisana jako `%20`, podczas gdy dosłowny znak mniej niż (`<`) w dokumencie `HTML` musi być zapisany jako `<`. `PHP` ma wiele wbudowanych funkcji do konwersji do i/z tych kodowań.

Znaki specjalne w `HTML` są reprezentowane przez jednostki, takie jak `&` (`&`) i `<` (`<`). Istnieją dwie funkcje `PHP`, które zamieniają znaki specjalne w łańcuchu na ich jednostki: jedna do usuwania znaczników `HTML`, a druga do wyodrębniania tylko znaczników meta.

Funkcja `htmlentities()` zamienia wszystkie znaki z odpowiednikami encji `HTML` na te odpowiedniki (z wyjątkiem znaku spacji). Obejmuje to znak mniej niż (`<`), znak większości (`>`), znak `&` (`&`) i znaki akcentowane.

```
<?php
    $string = htmlentities("Einstürzende Neubauten");
    echo $string;
?>
```

Znak `&uuml;` (widoczne po wyświetleniu źródła) poprawnie wyświetla się jako `ü` na renderowanej stronie internetowej. Jak widać, przestrzeń nie została przekształcona w ` `.

Funkcja `htmlentities()` w rzeczywistości przyjmuje do trzech argumentów:

```
$output = htmlentities(input, flags, encoding);
```

Parametr `encoding`, jeśli został podany, identyfikuje zestaw znaków. Wartość domyślna to `„UTF-8”`. Parametr `flags` kontroluje, czy pojedyncze i podwójne cudzysłowy są przekształcane w ich formy w HTML. `ENT_COMPAT` (domyślnie) konwertuje tylko podwójne cudzysłowy, `ENT_QUOTES` konwertuje oba typy cudzysłowów, a `ENT_NOQUOTES` nie konwertuje żadnego. Nie ma opcji konwertowania tylko pojedynczych cudzysłowów.

```
<?php
$input = <<<End
"Stop pulling my hair!" Jane's eyes flashed.<p>End;
End;
$double = htmlentities($input);
echo "$double\n";
$both = htmlentities($input, ENT_QUOTES);
echo "$both\n";
$neither = htmlentities($input, ENT_NOQUOTES);
echo $neither;
?>
```

Funkcja `htmlspecialchars()` konwertuje najmniejszy zestaw jednostek możliwych do wygenerowania prawidłowego kodu HTML. Konwertowane są następujące elementy:

- Ampersandy (`&`) są konwertowane na `&`
- Podwójne cudzysłowy (`"`) są konwertowane na `&quot;`
- Pojedyncze cudzysłowy (`'`) są konwertowane na `&#039;` (jeśli `ENT_QUOTES` jest włączone, jako opisane dla `htmlentities()`)
- Znaki mniejszości (`<`) są konwertowane na `<`
- Znaki większości (`>`) są konwertowane na `>`

Jeśli mamy aplikację wyświetlającą dane wprowadzone przez użytkownika w formularzu, musimy uruchomić te dane przez `htmlspecialchars()` przed ich wyświetleniem lub zapisaniem. Jeśli tego nie zrobimy, a użytkownik wprowadzi ciąg, taki jak `„kąt < 30”` lub `„sturm & drang”`, przeglądarka uzna, że znaki specjalne to kod HTML, co może doprowadzić do zniekształconej strony.

Podobnie jak `htmlentities()`, `htmlspecialchars()` może przyjmować do trzech argumentów:

```
$output = htmlspecialchars(input, [flags, [encoding]]);
```

`Flag` oraz `encoding` mają to samo znaczenie, co w przypadku `htmlentities()`.

Nie ma funkcji przeznaczonych specjalnie do konwersji z powrotem z jednostek do oryginalnego tekstu, ponieważ jest to rzadko potrzebne. Jest jednak na to stosunkowo prosty sposób. Użycie funkcji `get_html_translation_table()`, pobiera tabelę tłumaczeń używaną przez jedną z tych funkcji w danym stylu cudzysłowu. Na przykład, aby uzyskać tabelę tłumaczeń używaną przez `htmlentities()`, należy wykonać następujący kod:

```
<?php
    $table = get_html_translation_table(HTML_ENTITIES);
    print_r($table);
?>
```

Aby uzyskać tabelę dla `htmlspecialchars()` w trybie `ENT_NOQUOTES`, należy użyć:

```
<?php
    $table = get_html_translation_table(HTML_SPECIALCHARS, ENT_NOQUOTES);
    print_r($table);
?>
```

Fajną sztuczką jest użycie tej tabeli tłumaczeń, odwrócenie jej za pomocą `array_flip()` i przekazanie jej do `strtr()`, aby zastosować ją do łańcucha, tym samym skutecznie wykonując odwrotność `htmlentities()`:

```
<?php
    $str = htmlentities("Einstürzende Neubauten");
    $table = get_html_translation_table(HTML_ENTITIES);
    $revTrans = array_flip($table);
    echo strtr($str, $revTrans);
?>
```

Możemy również pobrać tabelę tłumaczeń, dodać do niej dowolne inne tłumaczenia, a następnie wykonać `strtr()`. Na przykład, jeśli chcemy, aby `htmlentities()` również zakodował każdą spację do ` `, powinniśmy napisać:

```
<?php
    $str = htmlentities("Einstürzende Neubauten");
    $table = get_html_translation_table(HTML_ENTITIES);
    $table[' '] = ' ';
    $encoded = strtr($str, $table);
    echo $encoded;
?>
```

Funkcja `strip_tags()` usuwa znaczniki `HTML` z ciągu:

```
<?php
    $input = '<p>Howdy, &quot;Cowboy&quot;</p>';
    $output = strip_tags($input);
    echo "$input\n";
    echo $output;
?>
```

Funkcja może przyjąć drugi argument, który określa ciąg znaczników do pozostawienia w ciągu. Podajemy tutaj tylko tagi otwierające, automatycznie jego domknięcia zostaną zachowane:

```
<?php
    $input = 'The <b>bold</b> tags will <i>stay</i><p>';
    $output = strip_tags($input, '<b>');
    echo "$input\n";
    echo $output;
?>
```

Atrybuty w zachowanych tagach nie są zmieniane przez `strip_tags()`. Ponieważ atrybuty takie jak `style` i `onmouseover` mogą wpływać na wygląd i zachowanie stron internetowych, zachowanie niektórych tagów za pomocą `strip_tags()` niekoniecznie usunie potencjalne nadużycia.

Funkcja `get_meta_tags()` zwraca tablicę metatagów strony `HTML`, określoną jako lokalna nazwa pliku lub adres URL. Nazwa metatagu (słowa kluczowe, autor, opis itp.) staje się kluczem w tablicy, a zawartość metatagu odpowiada wartością:

```
<?php
    $metaTags = get_meta_tags('https://mmiotk.gitlab.io');
    echo "Web page made by {$metaTags['author']}";
?>
```

Ogólna postać funkcji to:

```
$array = get_meta_tags(filename [, use_include_path]);
```

Wartość `true` użyta dla argumentu `use_include_path` mówi, że `PHP` próbowało otworzyć plik przy użyciu standardowej ścieżki `include`.

`PHP` udostępnia funkcje konwersji do i z kodowania adresów `URL`, co pozwala na tworzenie i dekodowanie adresów `URL`. W rzeczywistości istnieją dwa typy kodowania adresów `URL`, które różnią się sposobem traktowania spacji. Pierwszy (określony w `RFC 3986`) traktuje spację jako kolejny niedozwolony znak w adresie URL i koduje go jako `%20`. Drugi (implementujący system `application/x-www-form-urlencoded`) koduje spację jako `+` i służy do budowania ciągów zapytań. Pełny adres URL, taki jak http://www.example.com/hello, będzie pomijać dwukropki i ukośniki, tworząc:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

Dlatego warto najpierw zakodować częściowe adresy URL i dodać protokół oraz nazwę domeny później.

Aby zakodować ciąg zgodnie z konwencją `URL`, należy użyć `rawurlencode()`:

```
$output = rawurlencode(input);
```

Ta funkcja pobiera ciąg znaków i zwraca kopię z niedozwolonymi znakami `URL` zakodowanymi w konwencji `%dd`. Jeśli dynamicznie generujemy odnośniki hipertekstowe dla linków na stronie, musimy je przekonwertować za pomocą `rawurlencode()`:

```
<?php
    $name = "Programming PHP";
    $output = rawurlencode($name);
    echo "http://localhost/{$output}";
?>
```

Funkcja `rawurldecode()` dekoduje ciągi zakodowane w adresie `URL`:

```
<?php
    $encoded = 'Programming%20PHP';
    echo rawurldecode($encoded);
?>
```

Funkcje `urlencode()` i `urldecode()` różnią się od swoich nieprzetworzonych odpowiedników tylko tym, że kodują spacje jako znaki plus (`+`) zamiast jako sekwencję `%20`. Jest to format tworzenia ciągów zapytań i wartości plików cookie. Funkcje te mogą być przydatne w dostarczaniu podobnych do formularzy adresów `URL` w `HTML`. `PHP` automatycznie dekoduje ciągi zapytań i wartości plików cookie, więc nie musimy używać tych funkcji do przetwarzania tych wartości. Funkcje są przydatne do generowania ciągów zapytań:

```
<?php
    $baseUrl = 'http://www.google.com/q=';
    $query = 'PHP sessions -cookies';
    $url = $baseUrl . urlencode($query);
    echo $url;
?>
```

Większość systemów bazodanowych wymaga, aby w zapytaniach `SQL` literały ciągów były zmieniane. Schemat kodowania `SQL` jest dość prosty — pojedyncze cudzysłowy, podwójne cudzysłowy, bajty `NUL` i odwrotne ukośniki muszą być poprzedzone odwrotnym ukośnikiem. Funkcja `addslashes()` dodaje te ukośniki, a funkcja `stripslashes()` usuwa je:

```
<?php
$string = <<< EOF
"It's never going to work," she cried,
as she hit the backslash (\) key.
EOF;
$string = addslashes($string);
echo "$string\n";
echo stripslashes($string);
?>
```

Funkcja `addcslashes()` unika dowolnych znaków, umieszczając przed nimi odwrotne ukośniki. Z wyjątkiem znaków w Tabeli poniżej, znaki o wartościach `ASCII` mniejszych niż `32` lub powyżej `126` są kodowane z ich wartościami ósemkowymi (np. `„\002”`). Funkcje `addcslashes()` i `stripcslashes()` są używane z niestandardowymi systemami baz danych, które mają własne pomysły na to, które znaki należy zmienić.

Wartość ASCII	Kodowanie
7	`\a`
8	`\b`
9	`\t`
10	`\n`
11	`\v`
12	`\f`
13	`\r`

Wywołanie funkcji `addcslashes()` z dwoma argumentami — ciągiem do zakodowania i znakami które chcemy użyć wygląda następująco:

```
$escaped = addcslashes(string, charset);
```

Możemy również określić zakres znaków za pomocą konstrukcji `„..”`:

```
<?php
    echo addcslashes("hello\tworld\n", "\x00..\x1fz..\xff");
?>
```

Jeżeli określimy znaki jako `„0”, „a”, „b”, „f”, „n”, „r”, „t”` lub `„v”`, to zostaną one zamienione na `„\0” , „\a”` i tak dalej. Zanki te są rozpoznawane przez `C` i `PHP` i mogą powodować zamieszanie.

Funkcja `stripcslashes()` pobiera ciąg znaków i zwraca kopię z rozwiniętymi podanymi znakami:

```
<?php
    $string = stripcslashes('hello\tworld\n');
    echo $string;
?>
```

Step 8

`PHP` ma dwa operatory i sześć funkcji do porównywania łańcuchów znaków ze sobą.

Możemy porównać dwa ciągi pod kątem równości z operatorami `==` i `===`. Operatory te różnią się sposobem, w jaki traktują operandy niełańcuchowe. Operator `==` rzutuje operandy łańcuchowe na liczby, więc informuje, że `3` i `„3”` są równe. Ze względu na zasady rzutowania łańcuchów na liczby, zgłasza również, że `3` i `„3b”` są równe, ponieważ podczas rzutowania używana jest tylko część łańcucha do znaku nie będącego liczbą. Operator `===` nie rzutuje i zwraca wartość `false`, jeśli typy danych argumentów różnią się:

```
<?php
    $o1 = 3;
    $o2 = "3";
    if ($o1 == $o2) {
        echo("== returns true<br>");
    }
    if ($o1 === $o2) {
        echo("=== returns true<br>");
    }
?>
```

Operatory porównania (`<, <=, >, >=`) działają również na łańcuchach:

```
<?php
    $him = "Fred";
    $her = "Wilma";
    if ($him < $her) {
        print "{$him} comes before {$her} in the alphabet.\n";
    }
?>
```

Aby jawnie porównać dwa łańcuchy jako łańcuchy, w razie potrzeby rzutując liczby na łańcuchy, należy użyć funkcji `strcmp()` :

```
$relationship = strcmp(string_1, string_2);
```

Funkcja zwraca liczbę mniejszą niż 0, jeśli `string_1` sortuje przed `string_2` , większą od 0, jeśli `string_2` sortuje przed `string_1` , lub 0, jeśli są takie same:

```
<?php
    $n = strcmp("PHP Rocks", 5);
    echo $n;
?>
```

Odmianą `strcmp()` jest `strcasecmp()` , która konwertuje łańcuchy na małe litery przed ich porównaniem. Jego argumenty i zwracane wartości są takie same jak w przypadku `strcmp()` :

```
<?php
    $n = strcasecmp("Fred", "frED");
    echo $n;
?>
```

Inną odmianą porównywania ciągów jest porównanie tylko kilku pierwszych znaków ciągu. Funkcje `strncmp()` i `strncasecmp()` przyjmują dodatkowy argument, początkową liczbę znaków do użycia w porównaniach:

```
<?php
    $n = strncmp("Alan", "Adam", 1);
    echo $n;
?>
```

Ostatnią odmianą tych funkcji jest porównanie w naturalnym porządku za pomocą `strnatcmp()` i `strnatcasecmp()` , które przyjmują te same argumenty co `strcmp()` i zwracają te same rodzaje wartości. Porównanie w kolejności naturalnej identyfikuje części liczbowe porównywanych ciągów i sortuje części ciągu oddzielnie od części liczbowych.

`PHP` udostępnia kilka funkcji, które pozwalają sprawdzić, czy dwa ciągi znaków są w przybliżeniu równe — `soundex()` , `metaphone()` , `similar_text()` i `levenshtein()` :

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep,
$cost_del ]);
```

Algorytmy `Soundex` i `Metaphone` dają ciąg, który w przybliżeniu przedstawia sposób wymawiania słowa w języku angielskim. Aby sprawdzić, czy dwa ciągi są w przybliżeniu równe tym algorytmom, należy porównać ich wymowę. Wartości `Soundex` można porównywać tylko z wartościami `Soundex` , a wartości `Metaphone` tylko z wartościami `Metaphone` . Algorytm `Metaphone` jest

ogólnie bardziej dokładny, co pokazuje poniższy przykład:

```
<?php
    $known = "Fred";
    $query = "Phred";
    if (soundex($known) == soundex($query)) {
        print "soundex: {$known} sounds like {$query}<br>";
    }
    else {
        print "soundex: {$known} doesn't sound like {$query}<br>";}
    if (metaphone($known) == metaphone($query)) {
        print "metaphone: {$known} sounds like {$query}<br>";
    }
    else {
        print "metaphone: {$known} doesn't sound like {$query}<br>";
    }
?>
```

Funkcja `similar_text()` zwraca liczbę znaków wspólnych dla jej dwóch argumentów, którymi są ciągi znaków. Trzeci argument, jeśli jest obecny, jest zmienną, w której będziemy przechowywać występującą część wspólną jako procent:

```
<?php
    $string1 = "Rasmus Lerdorf";
    $string2 = "Razmus Lehrdorf";
    $common = similar_text($string1, $string2, $percent);
    printf("They have %d chars in common (%.2f%%).", $common, $percent);
?>
```

Algorytm `Levenshteina` oblicza podobieństwo dwóch ciągów na podstawie liczby znaków, które należy dodać, zastąpić lub usunąć, aby były takie same. Na przykład `„cat”` i `„cot”` mają odległość `Levenshteina` równą `1`, ponieważ musimy zmienić tylko jeden znak (`„a”` na `„o”`), aby były takie same:

```
<?php
    $similarity = levenshtein("cat", "cot");
    print $similarity;
?>
```

Ta miara podobieństwa jest zazwyczaj szybsza do obliczenia niż ta używana przez funkcję `similar_text()` . Opcjonalnie możemy przekazać trzy wartości do funkcji `levenshtein()` w celu indywidualnego wstawienia, usunięcia i zastąpienia wagi — na przykład w celu porównania słów wraz z jego skrótem. Ten przykład nadmiernie waży wstawienia podczas porównywania ciągu z jego możliwym skrótem, ponieważ skróty nigdy nie powinny wstawiać znaków:

```
<?php
    echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
?>
```

Step 9

`PHP` ma wiele funkcji do pracy z łańcuchami. Najczęściej używane funkcje do wyszukiwania i modyfikowania ciągów to te, które używają wyrażeń regularnych do opisu danego ciągu. Funkcje opisane w tej sekcji nie używają wyrażeń regularnych — są szybsze niż wyrażenia regularne, ale działają tylko wtedy, gdy szukamy stałego ciągu (na przykład, jeśli szukamy `„12/11/01”` zamiast niż „dowolne liczby oddzielone ukośnikami”).

Jeśli wiemy, że dane, które nas interesują, znajdują się w większym ciągu, możemy je skopiować za pomocą funkcji `substr()` :

```
$piece = substr(string, start [, length ]);
```

Argument `start` jest pozycją w ciągu, od której należy rozpocząć kopiowanie, gdzie `0` oznacza początek ciągu. Argumentem `length` jest liczba znaków do skopiowania (domyślnie kopiuje się do końca ciągu).

```
<?php
    $name = "Fred Flintstone";
    $fluff = substr($name, 6, 4);
    $sound = substr($name, 11);
    echo "$fluff\n";
    echo "$sound";
?>
```

Aby dowiedzieć się, ile razy mniejszy ciąg występuje w większym, należy użyć funkcji `substr_count()`:

```
$number = substr_count(big_string, small_string)
```

```
<?php
$sketch = <<< EndOfSketch
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
?>
```

Funkcja `substr_replace()` pozwala na wiele rodzajów modyfikacji łańcucha:

```
$string = substr_replace(original, new, start [, length ]);
```

Funkcja zastępuje część oryginalną wskazaną przez początek (`0` oznacza początek ciągu) i długość wartości na nowy ciąg. Jeśli nie podano czwartego argumentu, `substr_replace()` usuwa tekst od początku do końca ciągu.

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "bye", 5, 7);
    echo $farewell;
?>
```

Użyj długości 0, aby wstawić bez usuwania:

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "bye", 9, 0);
    echo $farewell;
?>
```

Użyj zamiennika `""`, aby usunąć bez wstawiania:

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "", 9);
    echo $farewell;
?>
```

Oto jak można wstawić element na początku ciągu:

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "now it's time to say ", 0, 0);
    echo $farewell;
?>
```

Wartość ujemna dla `start` wskazuje liczbę znaków od końca ciągu, od którego należy rozpocząć zamianę:

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "riddance", -3);
    echo $farewell;
?>
```

Ujemna długość wskazuje liczbę znaków od końca ciągu, przy której należy zatrzymać usuwanie:

```
<?php
    $greeting = "good morning citizen";
    $farewell = substr_replace($greeting, "", -8, -5);
    echo $farewell;
?>
```

Funkcja `strrev()` pobiera ciąg i zwraca jego odwróconą kopię:

```
<?php
    echo strrev("There is no cabal");
?>
```

Funkcja `str_repeat()` pobiera ciąg znaków oraz liczbę i zwraca nowy ciąg składający się z ciągu podanego jako pierwszy argument, powtórzony tyle razy, ile zostało to podane w drugim argumencie.

```
<?php
    echo str_repeat('_.-.',10);
?>
```

Funkcja `str_pad()` dopełnia jeden ciąg innym. Opcjonalnie możemy powiedzieć czy dopełniać z lewej, prawej strony, czy z obu:

```
<?php
    $string = str_pad('Fred Flintstone', 30);
    echo "{$string}:35:Wilma";
?>
```

Opcjonalnym trzecim argumentem jest ciąg znaków do wypełnienia:

```
<?php
    $string = str_pad('Fred Flintstone', 30, '. ');
    echo "{$string}35";
?>
```

Opcjonalnym czwartym argumentem może być `STR_PAD_RIGHT` (ustawiony domyślnie), `STR_PAD_LEFT` lub `STR_PAD_BOTH` (do środka).

```
<?php
    echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
    echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
?>
```

`PHP` udostępnia kilka funkcji, które umożliwiają rozbicie łańcucha na mniejsze elementy. Są to `explode()`, `strtok()` i `sscanf()`.

Dane często przychodzą w postaci ciągów, które należy rozbić na tablicę wartości. Na przykład możemy chcieć podzielić pola oddzielone przecinkami od ciągu, takiego jak `„Fred,25,Wilma”`. W takich sytuacjach należy użyć funkcji `explode()`:

```
$array = explode(separator, string [, limit]);
```

Pierwszy argument, `separator`, to ciąg znaków zawierający separator pól. Drugi argument, `string`, to łańcuch do podziału. Opcjonalny trzeci argument, `limit`, to maksymalna liczba wartości zwracanych w tablicy. Jeśli limit zostanie osiągnięty, ostatni element tablicy zawiera resztę ciągu:

```
<?php
    $input = 'Fred,25,Wilma';
    $fields = explode(',', $input);
    print_r($fields);
    $fields = explode(',', $input, 2);
    print_r($fields);
?>
```

Funkcja `implode()` działa dokładnie odwrotnie niż `explode()` — tworzy duży ciąg z tablicy mniejszych ciągów:

```
$string = implode(separator, array);
```

Pierwszy argument, `separator`, jest ciągiem znaków, który należy umieścić między elementami drugiego argumentu, tablicy. Aby zrekonstruować prosty ciąg wartości oddzielony przecinkami, po prostu należy napisać:

```
<?php
    $fields = array('Fred', '25', 'Wilma');
    $string = implode(',', $fields);
    echo $string;
?>
```

Funkcja `join()` jest aliasem dla `implode()`.

```
<?php
    $fields = array('Fred', '25', 'Wilma');
    $string = join(',', $fields);
    echo $string;
?>
```

Funkcja `strtok()` pozwala na iterowanie po łańcuchu, otrzymując za każdym razem nowy fragment (`token`). Przy pierwszym wywołaniu musimy przekazać mu dwa argumenty: ciąg do iteracji oraz separator tokenów.

```
$firstChunk = strtok(string, separator);
```

Aby pobrać resztę tokenów, wielokrotnie należy wywołać `strtok()` tylko z separatorem:

```
$nextChunk = strtok(separator);
```

```
<?php
    $string = "Fred,Flintstone,35,Wilma";
    $token = strtok($string, ",");
    while ($token !== false) {
        echo("{$token}\n");
        $token = strtok(",");
    }
?>
```

Funkcja `strtok()` zwraca wartość `false`, gdy nie ma więcej tokenów do zwrócenia. Wywołanie `strtok()` z dwoma argumentami, ponownie zainicjalizuje iterator. Spowoduje to ponowne uruchomienie funkcji od początku ciągu.

Funkcja `sscanf()` rozkłada ciąg znaków zgodnie ze specyfikatorami formatu, używanymi we funkcji `printf()`:

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

W przypadku użycia bez opcjonalnych zmiennych, `sscanf()` zwraca tablicę pól:

```
<?php
    $string = "Fred\tFlintstone (35)";
    $a = sscanf($string, "%s\t%s (%d)");
    print_r($a);
?>
```

Przekazanie odwołania do zmiennych, powoduje, że pola będą przechowywane w tych zmiennych. Funkcja sscanf() zwraca liczbę przypisanych pól:

```
<?php
    $string = "Fred\tFlintstone (35)";
    $n = sscanf($string, "%s\t%s (%d)", $first, $last, $age);
    echo "Matched {$n} fields: {$first} {$last} is {$age} years old";
?>
```

Kilka funkcji znajduje ciąg lub znak w większym ciągu. Występują w trzech rodzinach: `strpos()` i `strrpos()`, które zwracają pozycję; `strstr()`, `strchr()` i `friends`, które zwracają znaleziony ciąg; oraz `strspn()` i `strcspn()`, które zwracają, jaka część początku łańcucha pasuje do maski.

```
<?php
    $large = "my, little pony";
    $pos = strpos($large, ",");
    echo "$pos\n";
?>
```

Wszystkie funkcje wyszukujące ciągi zwracają wartość `false`, jeśli nie mogą znaleźć określonego podciągu. Jeśli podciąg występuje na początku ciągu, funkcje zwracają `0`. Ponieważ `false` rzutuje na wartość `0`, to zawsze należy porównywać zwracaną wartość z operatorem `===` podczas testowania pod kątem niepowodzenia:

```
<?php
    $large = "my, little pony";
    $pos = strpos($large, ",");
    if ($pos === false) {
      echo "False";
    }
    else {
      echo "True";
    }
?>
```

Funkcja `strpos()` znajduje pierwsze wystąpienie małego ciągu w większym ciągu:

```
$position = strpos(large_string, small_string);
```

Jeśli `small_string` nie zostanie znaleziony, `strpos()` zwraca `false`. Funkcja `strrpos()` znajduje ostatnie wystąpienie znaku w ciągu. Przyjmuje te same argumenty i zwraca ten sam typ wartości, co `strpos()`.

```
<?php
    $record = "Fred,Flintstone,35,Wilma";
    $pos = strrpos($record, ",");
    echo("The last comma in the record is at position {$pos}");
?>
```

Funkcja `strstr()` znajduje pierwsze wystąpienie mniejszego ciągu w większym ciągu i zwraca ciąg rozpoczynając od tego mniejszego ciągu:

```
<?php
    $record = "Fred,Flintstone,35,Wilma";
    $rest = strstr($record, ",");
    echo "$rest";
?>
```

Wariacjami funkcji `strstr()` są:

- `stristr()` - jest to wersja case-sensitive `strstr()`

  ```
  <?php
      $record = "Fred,Flintstone,35,Wilma";
      $rest = stristr($record, "f");
      echo "$rest";
  ?>
  ```

- `strchr()` - Jest to alias do `strstr()`

  ```
  <?php
      $record = "Fred,Flintstone,35,Wilma";
      $rest = strchr($record, ",");
      echo "$rest";
  ?>
  ```

- `strrchr()` - Zwraca ostatnie wystąpienie litery w ciągu

```
<?php
    $record = "Fred,Flintstone,35,Wilma";
    $rest = strrchr($record, "F");
    echo "$rest";
?>
```

Podobnie jak w przypadku `strrpos()`, `strrchr()` przeszukuje od końca łańcucha, ale tylko do pojedynczego znaku, a nie całego łańcucha.

Funkcje `strspn()` i `strcspn()` informują, ile znaków na początku ciągu składa się z określonych znaków:

```
$length = strspn(string, charset);
```

Na przykład poniższa funkcja sprawdza, czy ciąg zawiera liczbę ósemkową:

```
<?php
    function isOctal($str)
    {
        return strspn($str, '01234567') == strlen($str);
    }
    var_dump(isOctal("0777"));
    var_dump(isOctal("ABF"));
?>
```

Litera `c` w `strcspn()` oznacza uzupełnienie — mówi, jaka część początku łańcucha nie składa się ze znaków z zestawu znaków. Należy użyć go, gdy liczba interesujących nas znaków jest większa niż liczba nieinteresujących nas znaków. Na przykład poniższa funkcja sprawdza, czy ciąg zawiera bajty `NUL`, tabulatory lub znaki powrotu karetki:

```
<?php
    function hasBadChars($str)
    {
        return strcspn($str, "\n\t\0") != strlen($str);
    }
    var_dump(hasBadChars("ala ma kota"));
    var_dump(hasBadChars("ala\nma\nkota\n"));
?>
```

Funkcja `parse_url()` zwraca tablicę składników adresu `URL`:

```
<?php
    $bits = parse_url("http://me:secret@example.com/cgi-bin/board?user=fred");
    print_r($bits);
?>
```

Możliwe klucze to `scheme`, `host`, `port`, `user`, `pass`, `path`, `query` i `fragment`.