

*Tomasz "Zyx" Jędrzejewski*

# Ciągi tekstowe w PHP

Wersja 2.0 (23.07.2008)



Szczegółowe informacje o licencji znajdują się pod artykułem.

[www.zyxist.com](http://www.zyxist.com)

# Ciągi tekstowe w PHP

Ciągi tekstowe (stringi) są chyba najpopularniejszym typem danych wykorzystywanym w PHP. Umożliwiają bowiem przechowywanie tekstu oraz wszelkich informacji znakowych, a używane są m.in do generowania kodu HTML na stronie. Dlatego też właściwy sposób ich użycia powinien przyswoić sobie każdy programista piszący w tym właśnie języku.

## Czym jest ciąg tekstowy?

Aby utworzyć ciąg tekstowy w języku PHP, używamy apostrof (') bądź cudzysłowów ("). Wszystkie znaki pomiędzy nimi są interpretowane jako dane tekstowe, które potem można wstawić do zmiennej lub wyświetlić na stronie WWW. Ważne jest, aby ciąg kończył się tym samym znakiem, jakim się rozpoczął. Przykłady:

```
"to jest poprawny ciąg"  
'to również jest poprawny ciąg'  
"to nie jest poprawny ciąg"  
'to też nie jest poprawny ciąg"
```

Pierwsze dwa przykłady pokazują poprawnie zdefiniowane ciągi tekstowe w PHP. Dwa kolejne takimi ciągami już nie są, gdyż kończą się apostrofą, zaczynają - cudzysłowiem i na odwrót.

Może się zdarzyć tak, że rozpoczęliśmy ciąg apostrofami, ale chcemy umieścić w nim apostrof. Co wtedy? Sposób taki:

```
'tekst tekst' tekst tekst'
```

nie zadziała i skończy się najwyżej wygenerowaniem komunikatu błędu. Na szczęście jest na to sposób - kiedy poprzedzimy apostrofe/cudzysłów znakiem backslash (\), parser PHP pominie ją i uzna za część tworzonego ciągu tekstowego:

```
'tekst tekst\' tekst tekst'
```

W ogóle należy pamiętać, iż znak backslash jest pomijany podczas parsowania:

```
echo 'przyklad backslasha przyklad backslasha';  
// otrzymamy: przyklad backslasha przyklad backslasha';
```

Jeżeli jednak jest on nam koniecznie potrzebny do szczęścia, stawiamy go podwójnie!

```
echo 'przyklad backslasha \\ przyklad backslasha';  
// otrzymamy: przyklad backslasha przyklad backslasha';
```

## Różnica między apostrofami a cudzysłowami

Wbrew pozorom, różnica między tymi dwoma metodami definiowania ciągów jest dość znaczna i koniecznie należy ją znać. Przyjrzyjmy się najpierw cudzysłowom...

Jak już zapewne zdążyłeś się zorientować z powyższego tekstu, parser PHP musi przejrzeć każdy tworzony ciąg tekstowy w poszukiwaniu różnych specjalnych znaków. Ich cały zestaw będziemy mogli wykorzystać, używając cudzysłowów do ograniczenia ciągu. Przede wszystkim - wstawianie zmiennych. Załóżmy, iż generujesz sobie jakiś raport, w którego ważne fragmenty chcesz powstawić ważne dane. Zamiast pracować zamykając ciąg, wstawiać zmienną, otwierać ciąg ponownie i tak w kółko, zmienne możemy wstawić bezpośrednio w tekst, a parser PHP zamieni je na odpowiednie wartości:

```
<?php
```

```
$wiek = 18;
$ocena = 5;

echo "Wiek ucznia: $wiek. Ocena ucznia: $ocena";
// otrzymamy: Wiek ucznia: 18. Ocena ucznia: 5
?>
```

Jednak tu też kryje się pewien haczyk - otóż za nazwę zmiennej parser bierze wszystkie znaki od "\$" do pierwszego znaku nie będącego literą/cyfrą. Co zatem zrobić, gdy chcemy wyraźnie zaznaczyć, dokąd ciągnie się nazwa naszej zmiennej? Używamy nawiasów klamrowych:

```
<?php
$przedmiot = 'śrubokręt';

echo "Nie używałem $przedmiotu";
// Wyświetli: Nie używałem

echo "Nie używałem {$przedmiot}u";
// Wyświetli: Nie używałem śrubokrętu
?>
```

W podobny sposób można dodawać do ciągów także dane z tablicy:

```
<?php
$tablica[1] = 5;
$tablica[2] = 7;
$tablica['tekst'] = 'Pan Tadeusz, czyli ostatni zajazd na Litwie';

echo "Wartości tablicy: $tablica[1], $tablica[2], $tablica[tekst]";
// Wyświetli: Wartości tablicy: 5, 7, Pan Tadeusz, czyli ostatni zajazd na Litwie
?>
```

Popatrz na przykład. Musimy pamiętać, że przy wstawianiu danych z tablicy asocjacyjnej, nie używamy apostrofu itd. do ograniczenia nazwy indeksu. Robią to za nas kwadratowe nawiasy.

Również klasy poddają się temu zabiegowi - w ich przypadku całość umieszczamy w klamrach:

```
<?php

class klasa{
    var $pole;
}

$klasa = new klasa;
$klasa->pole = 6;

echo "Wartość pola w klasie: {$klasa->pole}";
// Otrzymamy: Wartość pola w klasie: 6
?>
```

Oprócz możliwości wstawiania zmiennych, cudzysłów daje nam całą gamę kodów specjalnych zastępujących wiele znaków. Kody te poprzedzone są zawsze znanym już nam backslashem:

- `\n` - zejście do nowej linii
- `\r` - powrót karetki
- `\t` - tabulator
- `\\` - backslash
- `\$` - znak dolara
- `\"` - cudzysłów
- `\{` - klamra otwierająca

- `\}` - klamra zamykająca

Dobrze, co więc dają apostrofy, zapytasz. Hmm... bardziej pasuje pytanie, czego nie dają, gdyż prawidłowa odpowiedź brzmi: nie dają 95% tego, co dają cudzysłowy. Oto porównanie:

```
<?php
$wartosc = 6;
echo "Wartość: $wartosc\n";

echo 'Wartość: $wartosc\n';
?>
```

Pierwsza instrukcja `echo` wyświetli nam tekst "Wartość", a po nim wstawi wartość zmiennej `$wartosc`. Dodatkowo zakończy tekst zejściem do nowej linii. Drugie `echo`, wyświetlające tekst z apostrof, da nam coś takiego:

```
Wartość: $wartosc\n
```

Oznacza to, iż nie ma tutaj mowy o wstawianiu zmiennych bezpośrednio do ciągu, czy o jakichś znakach specjalnych (poza oczywiście `\'` i `\\`) - wszystko jest interpretowane jako czysty ciąg tekstowy. Pomimo wszystko zalecam jego używanie, a dlaczego - już tłumaczę.

Cudzysłów daje potężne możliwości, lecz ich użycie powoduje całkiem niezłe spowolnienie skryptu. Z moich pomiarów wynika, że wstawienie do ciągu zmiennej przy pomocy kropek działa siedem do ośmiu razy szybciej, niż dzięki właściwościom cudzysłowów. Który wariant wybierzesz – zależy od Ciebie. Na pewno do tworzenia wyrażeń regularnych znacznie lepiej nadają się apostrofy. Ja jednak korzystam z nich zawsze, gdyż dodatkowe możliwości nie są mi potrzebne i w związku z tym dlaczego mają mi przeszkadzać?

## Here document

Istnieje jeszcze jeden sposób tworzenia ciągów, zwany "Here Document". Używa się go najczęściej do tworzenia wielolinijkowych tekstów. Rozpoczynamy go znakami `<<<`, następnie dajemy unikalny identyfikator, który nie pojawi się we wprowadzanym tekście. Kiedy tekst dobiegnie ku końcowi, tworzymy dodatkową linijkę, a w niej wstawiamy nasz identyfikator i średnik. UWAGA: Wraz z nimi NIE MOGĄ znaleźć się ŻADNE inne znaki, w tym spacje i tabulacje!

```
<?php
echo <<<EOF
to jest wielolinijkowy tekst
napisany metoda HereDocument
Jak widać, jest do doskonałe
narzędzie do tego typu
zastosowań
EOF;
?>
```

## Łączenie ciągów

PHP daje nam także możliwość połączenia dwóch ciągów w jeden. Używa do tego celu operatora łączenia (`.`). Ale to nie wszystko. Przy jego pomocy można łączyć nie tylko ciąg z ciągiem, ale też ciąg ze zmienną lub wyrażeniem!

```
<?php
echo 'Operatory łączenia ciągów'.' przydają się najczęściej przy
apostrofach, '.' ale nie tylko';
echo 'Mogą też posłużyć do łączenia ciągów ze zmiennymi: '.$zmienna.' ';
echo 'A także do łączenia dwóch rodzajów ciągów!'. "\n";
echo 'Ponadto jest to doskonały sposób do wstawiania do tekstu wyników
wyrażeń: '.(time() - 3600);
?>
```

Operator łączenia można połączyć z operatorem przypisania, by dodać tekst do już istniejącego w jakiejś zmiennej:

```
<?php
$tekst = 'Litwo, ojczyzna moja! Ty jesteś, jak zdrowie<br/>';
$tekst .= 'Ile Cię trzeba cenić, ten tylko się dowie<br/>';
$tekst .= 'Kto cię stracił. Dziś piękność twą w całej ozdobie';
$tekst .= 'Widzę i opisuję, bo tęsknię po tobie.';
?>
```

## Porównywanie ciągów

Ciągi, podobnie jak inne zmienne, można porównywać. Do dyspozycji mamy operatory porównania: `==`, `===`, a także specjalną funkcję `strcmp()` o większych możliwościach. Omówimy sobie tutaj każdy z tych trzech sposobów. Na początek - operator przypisania...

```
<?php
$tekst1 = 'bimber';
$tekst2 = '2';

if($tekst1 == 'bimber')
{
    echo '1 - TRUE';
}

if($tekst2 == 2)
{
    echo '2 - TRUE';
}
?>
```

Obie instrukcje porównania wyrzucą nam napis TRUE. W pierwszym przypadku - wszystko jest oczywiste. W drugim - porównujemy ciąg tekstowy z liczbą, z tym że nie zazaczyliśmy, iż typy danych także muszą być takie same! Zarówno liczba, jak i ciąg przechowują wartość 2, zatem według tego operatora jest to ta sama wartość.

Trochę inaczej wygląda to w przypadku drugiego operatora:

```
<?php
$tekst1 = 'bimber';
$tekst2 = '2';

if($tekst1 === 'bimber')
{
    echo '1 - TRUE';
}

if($tekst2 === 2)
{
    echo '2 - TRUE';
}
?>
```

Pierwszy tekst (podobnie, jak ostatnio) zwróci nam TRUE, ale drugi nie spowoduje wyświetlenia żadnego tekstu. Dlaczego? Pomimo iż wartości w liczbie i ciągu są takie same, to jednak są to różne typy - tekst i liczba. Operator `===` tymczasem wymaga także obecności tych samych typów.

`strcmp()` jest zapożyczone prosto z języka C, gdzie było jedyną metodą porównywania ciągów tekstowych. Funkcja daje możliwość leksykograficznego porównywania ciągów, przez co znakomicie nadaje się jako porównywarka podczas sortowania. Zwracane wartości to `-1`, jeśli A jest przed B, `0`, jeśli są sobie równe i `1`, gdy B powinno być przed A. Przykład:

```
<?php
$tekst1 = 'aaaaaa';
$tekst2 = 'abcdef';
$tekst3 = 'abcdeg';
```

```

echo strcmp($tekst1, 'abcdef');
echo strcmp($tekst2, 'abcdef');
echo strcmp($tekst3, 'abcdef');

// Porównanie
if(strcmp($tekst2, 'abcdef') == 0){
    echo 'Teksty są takie same';
}
?>

```

W tym przykładzie możemy zobaczyć, co zwraca funkcja `strcmp()` dla różnych kombinacji tekstów, a także jak porównywać je w instrukcji warunkowej. Ponieważ funkcja zwraca 0 w przypadku równości, odmiennie niż większość funkcji, trzeba przyzwyczaić się do podanego zapisu.

## Pobieranie znaków z ciągu

PHP wspomaga także pobieranie pojedynczych znaków z ciągu. Całość jest również bajecznie prosta - po zmiennej tekstowej wstawiamy nawiasy kwadratowe, a w nich numer znaku, który nas interesuje (uwaga: znaki numerowane są od zera!).

```

<?php
$tekst = 'Ala ma kota, a kot ma AIDS';
echo $tekst[0]; // Wyświetli A
echo $tekst[4]; // Wyświetli m
?>

```

**Uwaga:** w PHP 4 i początkowych wersjach PHP 5 na równych prawach funkcjonowały w tym miejscu nawiasy klamrowe. Nie polecamy ich jednak używać, ponieważ w związku z procesem ujednoczenia składni począwszy od wersji PHP 6 "jedyną" słuszną wersją będą właśnie nawiasy kwadratowe.

## Inne funkcje obsługi ciągów

Cały zestaw funkcji dedykowanych ciągom jest opisany w dokumentacji<sup>1</sup>. Tutaj zamieszczę tylko te podstawowe.

Bardzo często przyda nam się długość jakiegoś ciągu. Do jej otrzymania używamy funkcji `strlen()`:

```

<?php
echo strlen('Ala ma kota');
?>

```

Kiedy chcemy wyciąć jakiś duży fragment ciągu, z pomocą przyjdzie nam funkcja `substr()`. Pobiera ona trzy parametry. Pierwszy to ciąg, z którego będziemy wycinać. Następnie podajemy pozycję znaku, od którego będziemy wycinać (tu także znaki numerowane są od zera), a na końcu długość wyciętego ciągu, który otrzymamy jako wynik.

```

<?php
echo substr('Ala ma kota', 7, 4); // wytnie wyraz 'kota'
echo substr('Ala ma kota', 0, 3); // wytnie wyraz 'Ala'
echo substr('Ala ma kota', 5, 2); // wytnie wyraz 'ma'
?>

```

Kiedy jednak chcemy coś wyciąć, a nie wiemy, gdzie się zaczyna? Z pomocą przyjdzie nam funkcja `strpos()`. Pobiera ona dwa ciągi i zwraca początek drugiego ciągu w pierwszym. Gdy nic nie zostanie znalezione, otrzymamy wartość `FALSE`.

```

<?php
if((strpos('Ala ma kota', 'ma') = $pozycja) !== FALSE)
{
    echo '"ma" zaczyna się na pozycji '.$pozycja;
}
else
{
    echo '"ma" nie zostało znalezione';
}

```

```
}  
?>
```

Wspomniana funkcja pobiera także opcjonalny trzeci parametr, dzięki któremu może zacząć ona poszukiwania od żadanego miejsca w ciągu.

## Zakończenie

Ostatni rozdział artykułu postanowiłem poświęcić przede wszystkim praktycznym wskazówkom, które powinieneś stosować, aby nie spotkały Cię przykre niespodzianki. O tym, kiedy używać powinno się cudzysłowów, a kiedy apostrof - wspomniałem. Ale jest jeszcze jedna ważna sprawa, którą można zaliczyć do grzechów głównych początkujących programistów PHP - wykorzystywanie ciągów do... wstawiania parametrów do funkcji. Jest to najgłupszy z możliwych przykładów ilustrujących tezę, iż prostota i elastyczność PHP jest zarówno zaletą, jak i wadą. Oto przykład:

```
<?php  
// Inicjujemy parę zmiennych  
$plik = fopen('jakisplik.php', 'w');  
$wartosc = 15;  
  
// Dobrze  
fwrite($plik, $wartosc);  
  
// Zle  
fwrite("$plik", "$wartosc");  
?>
```

O ile zmienna *\$wartosc* zostanie jeszcze prawidłowo wpisana, poza tym, iż niepotrzebnie wykonuje się tutaj przetwarzanie ciągu, o tyle już wprowadzenie w ten sposób zmiennej *\$plik* spowoduje najwycyżniejsze błędy, bowiem zmienna ta ma typ określany przez PHP jako "Resource", czyli zasób. W efekcie funkcja *fwrite()*, zamiast zasobu z identyfikatorem pliku, do którego ma zapisywać, otrzyma... tekst w stylu "Resource ID #13". Podobne zawirowania wystąpią przy próbie przekazania w ten sposób obiektów i całych tablic. Taki styl pisania powinien być tępiony, ponieważ nie tylko jest bez sensu, ale może być przyczyną poważnych problemów. Pamiętaj - nigdy nie stosuj ciągów do wprowadzania gdziekolwiek wartości jakiegokolwiek zmiennej!

Druga sprawa dotyczy problemów algorytmicznych związanych z ciągami - jeżeli potrzebujesz przeprowadzić na ciągu jakąś operację, np. zamianę liter dużych na małe, sprawdź w dokumentacji, czy PHP nie udostępnia przypadkiem do tego gotowych funkcji. Z pewnością odkrycie, że "o, to do tego jest już funkcja?" jest bardzo interesujące, ale lepiej wiedzieć zawczasu.

**Tomasz "Zyx" Jędrzejewski**

Aktualna wersja artykułu zawsze na [www.zyxist.com](http://www.zyxist.com)

# Licencja

Artykuł rozpowszechniany jest na licencji **Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 2.5 Polska**.

## Wolno:

- kopiować, rozpowszechniać, odtwarzać i wykonywać utwór

## Na następujących warunkach:

1. *Uznanie autorstwa*. Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę\*.
2. *Użycie niekomercyjne*. Nie wolno używać tego utworu do celów komercyjnych.
3. *Bez utworów zależnych*. Nie wolno zmieniać, przekształcać ani tworzyć nowych dzieł na podstawie tego utworu.

W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.

Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Powyższe postanowienia w żaden sposób nie naruszają uprawnień wynikających z dozwolonego użytku ani żadnych innych praw.

Internetowa skrócona wersja licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/>

Pełen tekst licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/legalcode>

---

Podczas publikacji należy podać następujące informacje:

1. Link do internetowej skróconej wersji licencji.
2. Informację o wersji publikowanego tekstu.
3. Informacje o autorze w następujący sposób:

**Tomasz "Zyx" Jędrzejewski**

Aktualna wersja artykułu zawsze na [www.zyxist.com](http://www.zyxist.com)