List of Examples

Chapter 1. Przedmowa

Niewątpliwie jednym z najczęściej zadawanych pytań na forach PHP jest: jak mogę uniezależnić moje skrypty PHP od grafiki? Dopóki PHP zbudowany jest jako „język skryptowy osadzony w HTML-u”, wciąż mieszana jest logika z wyglądem, jednak po napisaniu kilku projektów łączących PHP i HTML w końcu przychodzi nam do głowy pomysł, że odseparowanie formy i zawartości to Dobra Rzecz [TM]. W dodatku w wielu przypadkach role grafików i programistów są od siebie niezależne, co jest koejnym argmentem to rozdzielenia ich pracy. W rezultacie, następuje poszukiwanie systemu szablonowego.

Na przykład w naszej firmie rozwój aplikacji przebiega następująco: Po udokumentowaniu wymagań, projektanci interfejsu tworzą projekt interfejsu i przekazują jego projekt programistą. Programiści implementują logikę PHP i używają projektu interfejsu do utworzenia szkieletu szablonów. Projekt jest wtedy przekazywany w ręce konstruktora strony web, który doprowadza szablony do „pełnego blasku”. Projekt może krążyć tam i z powrotem , pomiędzy projektantem stron web a programistą, nawet kilka razy. Dlatego jest to bardzo ważne aby mieć odpowiednie wsparcie szablonów ponieważ programiści nie chcą mieć nic wspólnego z HTML i nie chcą aby projektanci stron HTML zmieniali coś w kodzie PHP. Projektanci HTML potrzebują wsparcia dla plików konfiguracyjnych, dynamicznych bloków oraz innych wymagań interfejsu, ale nie chcą mieć nic wspólnego z zawiłościami programowania w PHP.

Szukając w wielu systemach szablonowych dostępnych dzisiaj dla PHP, zorienienyujemy sie, że większość z nich dostarcza elementarnych możliwości zamiany zmiennych w szablonach i tworzyć ograniczone funkcjonalności dynamicznych bloków. Ale my potrzebujemy czegoś więcej. Nie chcieliśmy żeby programiści mieli coś wspólnego z wyglądem strony , jednak było to niemal nieuchronne. Na przykład, jeżeli grafik chciał aby kolory tła występowały naprzemiennie w dynamicznych blokach, musiało to być opracowane przez programistę z niezłymi umiejętnościami.;)) Chcieliśmy również aby konstruktorzy mieli możliwość korzystania ze swoich własnych skonfigurowanych plików i przenoszenia z nich zmiennych na szablony.

Rozpoczęliśmy ponowne spisywanie specyfikacji dla silnika szablonów pod koniec 1999 roku. Po ukończeniu specyfikacji, rozpoczęliśmy prace nad silnikiem systemu szablonów pisanym w C co z pewnością byłoby przyjęte w połączeniu z PHP. Podczas naszej pracy napotkaliśmy wiele technicznych przeszkód i niejednokrotnie gorąco debatowaliśmy na temat tego co właściwie taki silnik powinien a czego nie powinien robić. W końcu uznaliśmy, że silnik powinien być napisany w PHP jako klasa, tak aby każdy mógł go używać jak tylko mu pasuje w zależności od potrzeb. Dlatego też napisaliśmy silnik działający w taki właśnie sposób i tak powstał SmartTemplate (ta klasa nigdy nie była zaprezentowana publicznie). Była to klasa, która robiła prawie wszystko co chcieliśmy aby wykonywała: stały czynnik zmian, wspomaganie zawierające inne szablony, integracja ze skonfigurowanymi plikami, osadzenie kodu PHP, ograniczenie zwrotu ‘if’ w funkcjonowaniu i wiele innych silnych dynamicznych bloków, które mogą być różnorako osadzane. Wszystko to oparto na wyrażeniach regularnych przez co kod stał się raczej, można powiedzieć, hermetyczny.. Ponadto był on zauważalnie powolny w dużych aplikacjach dla wszystkich parsowań i wyrażeń regularnych pracujących przy każdym wywołaniu. Największym problemem z punktu widzenia programistów była cała konieczna do wykonania praca w arkuszu PHP aby móc układać i przetwarzać szablony i dynamiczne bloki. Jak zatem uprościliśmy ten proces?

Wtedy pojawiła się wizja czym ostatecznie będzie Smarty. Wiemy jak szybki jest kod PHP bez parsowania szablonu. Wiemy również jak skrupulatny i władczy może wydawać się język PHP dla przeciętnego projektanta, może to być zasłonięte przez dużo prostszy templating synax. Co więc stanie się jeżeli połączymy dwie siły? Tak więc, narodził się Smarty....

1. Czym jest SMARTY

Smarty jest systemem szablonowym dla PHP. Służy ułatwieniu oddzielenia logiki aplikacji od jej wyglądu . Najlepiej opisuje go sytuacja kiedy programista aplikacji i projektant szablonów mają do spełnienia różne role (bo przecież w większości przypadków to nie jest ta sama osoba). Na przykład, powiedzmy, że projektujesz stronę web która wyświetla artykuł z gazety. Nagłówek, stopka autor i treść są elementami zawartymi w artykule, przyporządkowany im numer mówi nam w jaki sposób poszczególne elementy będą prezentowane. Elementy zostają przyporządkowane przez aplikacje do Smarty, wtedy projektant szablonów edytuje szablony używając kombinacji znaczników HTML i znaczników szablonu do formatowania prezentacji tych elementów (tabelki HTML, kolory tła, wielkość czcionek, arkusze stylów, itd.). Pewnego dnia programista potrzebuje zmienić sposób w jaki dane artykułu są pozyskiwane (zmiana logiki aplikacji). Ta zmiana nie interesuje projektanta szablonu, dane artykułu dalej będą dostarczane do szablonu w ten sam sposób. Podobnie, jeżeli projektant szablonu chce totalnie zmienić wygląd szablonu, nie wymaga to zmiany w logice aplikacji. Dlatego programista może zmieniać logikę aplikacji bez potrzeby restrukturyzacji szablonów, a projektant szablonów może modyfikować szablony bez łamania logiki aplikacji.

Teraz kilka słów o tym czego SMARTY nie robi. Smarty nie próbuje całkowicie oddzielić logiki od szablonów. Nie ma problemu z logiką w twoich szablonach pod warunkiem,że ta logika służy tylko do prezentacji. Krótka rada: trzymaj logikę aplikacji z dala od szablonów i logikę prezentacji z dala od aplikacji. To definitywnie utrzyma twoją witrynę jako w pełni i łatwo edytowalną w przyszłości.

Jednym z unikalnych aspektów Smarty jest kompilowanie szablonów. To znaczy, że Smarty czyta plik szablonu i tworzy dla niego skrypt PHP. Kiedy już jest stworzony, szablony uruchamiane są przez ten skrypt. Dlatego nie ma potrzeby parsowania plików szablonów dla każdego żądania, równocześnie każdy szablon jest w pełni obrabialny dal kompilerów PHP takich jak Zend Accelerator (http://www.zend.com) albo PHP Accelerator (http://www.php-accelerator.co.uk).

Niektóre z własności Smarty:

1. Jest ekstremalnie szybki.
2. Jest efektywny odkąd parser PHP przejął "brudną robotę".
3. Nie parsuje szablonu za każdym razem, tylko raz kompiluje.
4. Rekompiluje tylko te szablony które się zmieniły.
5. Język szablonów jest ekstremalnie elastyczny ponieważ możesz tworzyć własne funkcje oraz własne modyfikatory zmiennych.
6. Konfiguralne znaczniki wyrażeń szablonów, możesz używać {}, {{}}, <!—{}--!>, itd.
7. Konstrukcje if/elseif/else/endif są przekazywane do parsera PHP, więc wyrażenie {if...} może być proste lub kompleksowe – jak sobie życzysz.
8. Nieograniczone zagnieżdżanie sekcji, if-ów itd.
9. Jeśli to potrzebne można osadzić kod PHP w plikach szablonów, jednak jest to czynność nie polecana.
10. Wbudowane keszowanie.
11. Mozliwa własna konfiguracja źródeł szablonów.
12. Architektura oparta o wtyczki.
13. Architektura pozwalająca wykorzystywać własne rozszeżenia.

2. Instalacja

2.1. Wymagania

Smarty wymaga serwera sieciowego, na którym zainstalowany jest PHP 4.0.6 lub nowszy.

2.1.1. Podstawowa Instalacja

Zainstaluj pliki bibliotek Smarty znajdujące się w katalogu /libs/ dystrybucji. Są to pliki PHP których NIE POWINENEŚ zmieniać. Są one wydzielone spośród wszystkich aplikacji i zostają zaktualizowane tylko wtedy kiedy zaktualizujesz nową wersje Smarty.

Example 2.1. Pliki bibliotek Smarty

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/internals/*.php (wszystkie)
/plugins/*.php (wszystkie z nich, nalezy je tak zlokalizować a by były
bezpieczne - może twoja strona potrzebuje osobnego katalogu )
```

Smarty używa stałej PHP o nazwie `SMARTY_DIR` która jest ścieżką systemową katalogu z bibliotekami Smarty. Jeżeli twoja aplikacja potrafi znaleźć plik `Smarty.class.php`, zazwyczaj nie musisz ustawiawiać `SMARTY_DIR`. Natomiast, jeżeli Smarty.class.php nie

znajduje się w twoim include_path, lub nie dostarczysz ścieżki absolutnej do jego klasy w twojej aplikacji, wtedy musisz zdefiniować SARTY_DIR ręcznie. SMARTY_DIR musi zawierać ostatni slash.

Oto jak tworzyć przykładowy skrypt PHP ze Smarty:

Example 2.2. Tworzenie instancji Smarty

```
<?php
// NOTE: W Smarty pierwsza litera to duże 'S'
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Spróbuj uruchomić powyższy skrypt. Jeśli w rezultacie zostanie wyświetlony błąd mówiący o tym, że nie można znaleźć pliku Smarty.class.php spróbuj zrobić jedną z następujące rzeczy:

Example 2.3. Ręczne ustawienie SMARTY_DIR

```
<?php
// dla *nix (pamiętaj o dużym 'S')
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

// dla windows
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');

// wersja działająca zarówno na *nix jak i windows
// Przy założeniu że Smarty znajduje się w katalogu 'includes/' dla
konkretnego skryptu
define('SMARTY_DIR',str_replace("\\","/",getcwd()).'/includes/Smarty-
v.e.r/libs/');

require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

Example 2.4. Dostarczenie ścieżki absolutnej do pliku biblioteki

```
<?php
// dla *nix (pamiętaj o dużym 'S')
require_once('/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php');

// dla windows
require_once('c:/webroot/libs/Smarty-v.e.r/libs/Smarty.class.php');

$smarty = new Smarty();
?>
```

Example 2.5. Dodać katalog biblioteki do php_include path

```
<?php
// Zmień swój plik php.ini, dodaj katalog biblioteki Smarty
// do include_path, następnie zrestartuj serwer.
// Wtedy następujący kod powinien zadziałać:
require_once('Smarty.class.php');
```

```
$smarty = new Smarty();
?>
```

Teraz kiedy pliki bibliotek są na swoim miejscu, czas w twoej aplikacji ustawić katalogi Smarty.

Smarty wymaga czterech katalogów, których domyslne nazwy to `'templates/'`, `'templates_c/'`, `'configs/'` i `'cache/'`.

Każda z tych definiowalnych własności klasy Smarty: `$template_dir` , `$compile_dir`, `$config_dir` i `$casche_dir` dokładnie odpowiada każdemu katalogowi. Zalecane jest aby rozdzielić te katalogi dla każdej aplikacji używającej Smarty.

Upewnij się, że znasz twój główny katalog dokumentów na twoim serwerze. W naszym przykładzie, główny katalog to "`/web/www.mydomain.com/docs/`". Katalogi te muszą być dostępne dla biblioteki Smarty, i jednocześnie zawsze niedostępne bezpośrednio z poziomu przeglądarki. Dlatego aby uniknąć obaw związanych z bezpieczeństwem, rekomendowane jest umieszczenie tych katalogów w katalogu poza katalogiem głównym.

Dla przykładu instalacji, będziemy ustawiać otoczenie Smarty dla aplikacji księgi gości. Wybraliśmy tą aplikacje tylko w celu zaproponowania konwencji nazywania katalogów . Możesz używać tego samego otoczenia dla każdej aplikacji zmieniając tylko "guestbook" na nazwę twojej aplikacji. Umieścimy katalogi Smarty w "`/web/www.mydomain.com/smarty/guestbook/`".

Będziesz potrzebował jednego pliku pod katalogiem głównym, i to jest skrypt otwierany przez przeglądarkę. Nazwiemy nasz skrypt "`index.php`" i umieścimy go w podkatalogu katalogu głównego nazwanym "`/guestbook/`" .

> **Note**
>
> Dogodnie jest tak skonfigurować serwer aby "`index.php`" było domyślnym indexem katalogu, więc jeśli wpiszesz "`http://www.mydomain.com/guestbook/`", to skrypt będzie wykonany bez wpisania "index.php" w adresie URL. W Apache możesz ustawić taką opcję przez dodanie "index.php" na końcu ustawień DirectoryIndex (oddzielaj każdy wpis spacją).

Spójrzmy na obecną strukturę plików:

Example 2.6. Przykład struktury plików

```
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Config_File.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/debug.tpl
/usr/local/lib/php/Smarty-v.e.r/libs/internals/*.php
/usr/local/lib/php/Smarty-v.e.r/libs/plugins/*.php

/web/www.example.com/smarty/guestbook/templates/
/web/www.example.com/smarty/guestbook/templates_c/
/web/www.example.com/smarty/guestbook/configs/
```

```
/web/www.example.com/smarty/guestbook/cache/

/web/www.example.com/docs/guestbook/index.php
```

Smarty będzie potrzebował dostępu (zapis) do `$compile_dir` i `$casche_dir`, więc upewnij się, że użytkownik serwera może zapisywać w tych katalogach. Przeważnie jest to użytkownik "nobody" i grupa "nobody". Dla użytkowników OS X, domyślny użytkownik jest nazwany "web" i grupa "web". Jeśli używasz Apache, możesz spojrzeć do pliku httpd.conf (zazwyczaj w `"/usr/local/apache/conf/"`) aby zobaczyć użytkowników i grupy które są aktualnie używane.

Example 2.7. Ustawienia dostępu do plików

```
chown nobody:nobody /web/www.example.com/smarty/guestbook/templates_c/
chmod 770 /web/www.example.com/smarty/guestbook/templates_c/

chown nobody:nobody /web/www.example.com/smarty/guestbook/cache/
chmod 770 /web/www.example.com/smarty/guestbook/cache/
```

> ## Note
>
> chmod 770 zapewni całkowicie szczelną ochronę, pozwala użytkownikom "nobody" i grupą "nobody" na dostęp odczyt/zapis do katalogów. Jeśli chcesz otworzyć dostęp do odczytu dla kogokolwiek (przeważnie dla twojej wygody w przeglądaniu plików), możesz używać 775 zamiast 770.

Musimy stworzyć plik `"index.tpl"` który będzie ładowany przez Smarty. Ten plik będzie umieszczony w `$template_dir`.

Example 2.8. Edytowanie /web/www.mydomain.com/smarty/templates/index.tpl

```
{* Smarty *}

Hello, {$name}!
```

> ## Note
>
> {* Smarty *} jest to komentarz w szablonie. Nie jest wymagany, ale dobrym nawykiem jest zaczynać szablony komentarzem. Czyni to plik łatwiejszy do rozpoznania bez względu na rozszerzenie pliku. Na przykład, edytory tekstu mogły by rozpoznawać plik i włączać podświetlanie specyficznych wyrażeń.

Teraz popracujmy nad `index.php`. Stworzymy przykład Smarty, przydzielimy zmienne i wyświetlimy plik index.tpl. W naszym przykładzie `"/usr/local/lib/php/Smarty"` znajduje się w `include_path`. Upewnij się że zrobiłeś to samo, lub używaj ścieżek absolutnych.

Example 2.9. Edytowanie /web/www.mydomain.com/docs/guestbook/index.php

```
<?php

// ładwanie biblioteki Smarty
```

```
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();

$smarty->template_dir = '/web/www.example.com/smarty/guestbook/templates/';
$smarty->compile_dir =
'/web/www.example.com/smarty/guestbook/templates_c/';
$smarty->config_dir = '/web/www.example.com/smarty/guestbook/configs/';
$smarty->cache_dir = '/web/www.example.com/smarty/guestbook/cache/';

$smarty->assign('name','Ned');

$smarty->display('index.tpl');
?>
```

> **Note**
>
> W naszym przykładzie, ustawiliśmy absolutne ścieżki do wszystkich katalogów Smarty. Jeżeli '`/web/www.mydomain.com/smarty/guestbook/`' znajduje się w PHP `include_path`, to te ustawienia nie są potrzebne. Jednak, ta metoda jest skuteczniejsza i (z doświadczenia) mniej problemogenne. To upewnia Cię że Smarty pobiera pliki z katalogów które sam wprowadziłeś.

Teraz załaduj do przeglądarki plik `index.php` . Powinieneś zobaczyć "Hello, Ned!" Skończyłeś podstawową instalacje Smarty!

2.1.2. Zaawansowana instalacja

To jest kontynuacja podstawowej instalacji, proszę przeczytaj ją najpierw!

Znacznie zręczniejszym, bardziej elastycznym sposobem zainstalowania Smarty jest rozszerzenie klasy o klasę ustawiającą otoczenie dla Smarty. Więc zamiast powtarzać ustawianie ścieżek katalogów, przydzielać te same zmienne itd., możemy to zrobić w jednym miejscu. Stwórzmy nowy katalog "`/php/includes/guestbook/`" i stwórzmy nowy plik "`setup.php`". W naszym przykładzie, "`/php/includes/`" jest w naszych `include_path`. Upewnij się czy też to ustawiłeś, lub używaj ścieżek absolutnych.

Example 2.10. Edytowanie /php/includes/guestbook/setup.php

```
<?php

// Ładowanie biblioteki Smarty
require('Smarty.class.php');

// Plik setup.php jest dobrym miejscem do załadowania
// wymaganych dla aplikacji plików bibliotek, więc możesz
// to zrobic własnie tutaj. Dla przykładu:
// require('guestbook/guestbook.lib.php');

class Smarty_GuestBook extends Smarty {

   function Smarty_GuestBook()
   {

        // Class Constructor.
```

```
        // These automatically get set with each new instance.

        $this->Smarty();

        $this->template_dir =
'/web/www.example.com/smarty/guestbook/templates/';
        $this->compile_dir  =
'/web/www.example.com/smarty/guestbook/templates_c/';
        $this->config_dir   =
'/web/www.example.com/smarty/guestbook/configs/';
        $this->cache_dir    =
'/web/www.example.com/smarty/guestbook/cache/';

        $this->caching = true;
        $this->assign('app_name', 'Guest Book');
   }

}
?>
```

Teraz zmieńmy plik `index.php` tak aby używał `setup.php`:

Example 2.11. Edycja /web/www.mydomain.com/docs/guestbook/index.php

```
require('guestbook/setup.php');

$smarty = new Smarty_GuestBook;

$smarty->assign('name','Ned');

$smarty->display('index.tpl');
```

Teraz widzisz. że to całkiem proste podać przykład Smarty, użyj tylko Smarty_GuestBook który automatycznie wszystko w naszej aplikacji zainicjalizuje.

Chapter 3. Smarty dla projektantów szablonów

Table of Contents

1. Podstawowe wyrażenia

Wszystkie znaczniki w szablonach Smarty są otoczone ogranicznikami. Domyślnie ograniczniki to { i }, ale mogą one zostać zmienione .

Dla następujących przykładów będziemy używać domyślnych znaczników. W Smarty cała treść na zewnątrz ograniczników jest wyświetlana jako treść statyczna, lub niezmieniona. Kiedy Smarty napotyka na znaczniki szablonu, zaczyna je interpretować, i wyświetla zanalizowane dane wyjściowe w ich miejsce.

1.1. Komentarze

Komentarze szablonu są otoczone gwiazdkami, które z kolei otoczone przez ograniczniki np. {*to jest komentarz*}. Komentarze Smarty nie są wyświetlane w końcowym wyjściu szablonu, w odróżnieniu od unlike <!-- komentarzy HTML -->. Komentarze Smarty sa użyteczne do robienia wewnętrznych notatek wewnątrz szablonu.

Example 3.1. Komentarze

```
<body>
{* komentarz w jednej lini *}

{* to jest wieloliniowy
    komentarz, który
   nie zostanie wysłany do przeglądarki
*}

{* tutaj dołączam plik nagłówka *}
{include file="header.tpl"}


{* Nota dev:  $includeFile jest przyporządkowana ze skryptu foo.php *}
<!-- to jest komentaz html, który zostanie wysłany do przeglądarki -->
{include file=$includeFile}

{include file=#includeFile#}

{* zredukowany nlok <select> *}
{*
<select name="company">
  {html_options options=$vals selected=$selected_id}
</select>
*}
</body>
```

1.2. Zmienne

Zmienne szablonu zaczynają sie od znaku $ dollara. Moga zawierać cyfry, litery i podkreślniki - podobnie jak zmienne PHP. Możesz odwoływać sie do tablic indexowanych numerycznie lub nienumerycznie. Możesz również odwoływać sie do własciwości i metod obiektów. Zmienne plików konfiguracyjnych są wyjątkiem od skladni rozpoczynajacej się znakiem dolara. Można się do nich odwoływac przez otoczenie ich #znakamihash# lub specjalnej zmiennej `$smarty.config`.

Example 3.2. Zmienne

```
{$foo}        <-- wyswietla prostą zmienną (nie tablicę/obiekt)
{$foo[4]}     <-- wyswietla piąty element tablicy o indexowaniu
zaczynającym sie od zera
{$foo.bar}    <-- wyswietla wartości tablicy dla indexu "bar", podobnie jak
PHP $foo['bar']
{$foo.$bar}   <-- wyswietla wartości tablicy dla indexu zawartego w
zmiennej, podobnie jak PHP $foo[$bar]
{$foo->bar}   <-- wyswietla właściwość "bar" obiektu "foo"
{$foo->bar()} <-- wyswietla zwróconą wartości przez metodę "bar"
{#foo#}       <-- wyswietla zmienną konfiguracyjną "foo"
{$smarty.config.foo} <-- synonim dla {#foo#}
{$foo[bar]}   <-- składnia poprawna tylko dla pętli sekcji, zobacz
{section}
{assign var=foo value="baa"}{$foo} <--  wyswietla "baa", zobacz {assign}

Dozwolonych jest wiele innychkombinacji:

{$foo.bar.baz}
{$foo.$bar.$baz}
{$foo[4].baz}
{$foo[4].$baz}
{$foo.bar.baz[4]}
{$foo->bar($baz,2,$bar)} <-- przekazywanie parametrów
{"foo"}       <-- dozwolone są statyczne wartości
```

1.3. Funkcje

Każdy znacznik Smarty nawet ten wyświetlający zmienną lub wywołujący jakieś sortowanie zależy od funkcji. Funkcje są wykonywane i wyświetlane przez ograniczenie funkcji i jej atrybutów w ogranicznikach np. {nazwafunkcji attr1="val" attr2="val"}.

Example 3.3. Zapis funkcji

```
{config_load file="colors.conf"}

{include file="header.tpl"}

{if $highlight_name}
    Welcome, <font color="{#fontColor#}">{$name}!</font>
{else}
    Welcome, {$name}!
{/if}

{include file="footer.tpl"}
```

Zarówno wbudowane funkcje i costum functions posiadają takie samo wyrażenie w szablonach.

Funkcje wbudowane działają wewnątrz Smarty, tak jak {if}, {section} i {strip}. Nie mogą być modyfikowane.

Costum functions to dodatkowe funkcje implementowane przez wtyczki. Mogą być modyfikowane do twoich potrzeb, możesz również dodawać nowe. `{html options}` i `{html_select_date}` to przykłady i costum functions.

1.4. Atrybuty

Duża część funkcji pobiera atrybuty aby sprecyzować lub zmodyfikować swoje zachowanie. Atrybuty do funkcji Smarty są podobne do atrybutów HTML. Statyczne wartości nie muszą być zawarte w cudzysłowach, ale jest to zalecane ciągów znaków. Zmienne również mogą być używane i nie powinno sie ich zawierać w cudzysłowy.

Niektóre atrybuty wymagają boolowskich wartości (prawda lub fałsz). Te atrybuty moga być wyspecyfikowane jako inne nie zawierające się w cudzysłowach true, on, i yes, albo false, off, i no.

Example 3.4. Zapis atrybutów funkcji

```
{include file='header.tpl'}

{include file='header.tpl' attrib_name='attrib value'}

{include file=$includeFile}

{include file=#includeFile# title='Smarty is cool'}

{html_select_date display_days=yes}

{mailto address='smarty@example.com'}

<select name='company_id'>
  {html_options options=$companies selected=$company_id}
</select>
```

1.5. Osadzanie zmiennych w podwójnych cudzysłowach

Smarty rozpozna zmienne osadzone w podwójnych cudzysłowach gdy ich nazwy zawierają tylko cyfry, litery, podkreślniki i nawiasy kwadratowe []. W każdym innym przypadku zmienna musi być otoczona przez pojedyncze cudzysłowy (`). Nie mozna w cudzysłowach zamieszczać modyfikatorów muszaznaleźć się one na zewnątrz.

Example 3.5. Osadzanie wyrażeń w cudzysłowach

```
PRZYKŁADY WYRAŻEŃ:
{func var="test $foo test"}       <-- widzi $foo
{func var="test $foo_bar test"}   <-- widzi $foo_bar
{func var="test $foo[0] test"}    <-- widzi $foo[0]
{func var="test $foo[bar] test"}  <-- widzi $foo[bar]
{func var="test $foo.bar test"}   <-- widzi $foo (not $foo.bar)
{func var="test `$foo.bar` test"} <-- widzi $foo.bar

PRAKTYCZNE PRZYKŁADY:
```

```
{include file="subdir/$tpl_name.tpl"} <-- zastąpi $tpl_name jego wartością
{cycle values="one,two,`$smarty.config.myval`"} <-- zastąpi
$smarty.config.myval wartością
```

1.6. Matematyka

Matematyka może zostac zastosowana bezpośrednio do wartości zmiennych.

Example 3.6. Przykłady matematyki w szablonach

```
{$foo+1}

{$foo*$bar}

{* bardziej skomplikowane przykłady *}

{$foo->bar-$bar[1]*$baz->foo->bar()-3*7}

{if ($foo+$bar.test%$baz*134232+10+$b+10)}

{$foo|truncate:"`$fooTruncCount/$barTruncFactor-1`"}

{assign var="foo" value="`$foo+$bar`"}
```

Zobacz również funkcje {math} dla kompleksowych równań.

1.7. Opuszczanie pasowania Smarty

Czasami jest pożądane a nawet niezbedne aby Smarty zignorował sekcjie które normalnie by zostały sparsowane. Klasycznym przykładem jest umieszczanie kodu Javascript lub Css w szablonie. Problem powstaje ponieważ te języki używają znaków { i }, które jednocześnie są dmyślnymi ogranicznikami dla Smarty.

Najpostrzą czynnością jaką można wykonać aby uniknąć tej sytuacji jest przeniesienie kodu Javascript i CSS do osobnych plików używając standardowych metod jezyka HTML aby uzyskać do nich dostęp.

Dołączanie literalnej zawartości jest możliwe przez użycie bloku `{literal}` .. `{/literal}`. Podobnie jak HTMLowskich encjie możesz używać `{ldelim}`, `{rdelim}` lub `{$smarty.ldelim}` aby wyswietlić obecne ograniczniki.

Często wygodnie jest poprostu zmienić `$left_delimiter` and `$right_delimiter` Smartego.

Example 3.7. Pzykład zmiany ograniczników

```
<?php

$smarty = new Smarty;
$smarty->left_delimiter = '<!--{';
$smarty->right_delimiter = '}-->';
$smarty->assign('foo', 'bar');
$smarty->display('example.tpl');
```

```
?>
```

Gdzie `example.tpl`:

Example 3.8.

```
<script language="javascript">
  var foo = <!--{$foo}-->;
  function dosomething() {
    alert("foo is " + foo);
  }
  dosomething();
</script>
```

Zobacz również modyfikator escape.

2. Zmienne

Smarty poziada kilka róznych typów zmiennych. Typ zmiennej zalezy od tego jakim znakiem jest ona zaczeta i jakim znakiem jest zakończona.

Zmienne Smarty moga być bezpośrednio wyświetlone lub uzyte jako argumenty dla atrybutów funkcji i modyfikatorów, wewnątrz instrukcji warunkowych itd. Aby wyświetlić zmienną popostu zamkni ją ogranicznikiem tak iż bedzie ona tylko w ogranicznikach.

Example 3.9.

```
{$Name}

{$Contacts[row].Phone}

<body bgcolor="{#bgcolor#}">
```

2.1. Zmienne przyporządkowane z PHP

Zmienne przyporządkowane z PHP są rozpoznawane przez poprzedzanie ich znakiem dolara $. Zmienne przyporządkowane z wewnątrz szablonu przez funkcje assign również wyglądają w ten sposób.

Example 3.10. Przyporzadkowane zmienne

```
Hello {$firstname}, glad to see you could make it.
<p>
Your last login was on {$lastLoginDate}.

OUTPUT:

Hello Doug, glad to see you could make it.
<p>
Your last login was on January 11th, 2001.
```

2.2. Tablice asocjacyjne

Możesz również odwoływać się do tablic asocjacyjnych przyporządkowanych z PHP poprzez podanie klucza po symbolu '.' (kropka).

Example 3.11. Odwoływanie się do zmiennych w tablicy asocjacyjnej

```
index.php:

$smarty = new Smarty;

$smarty->assign('Contacts',
    array('fax' => '555-222-9876',
          'email' => 'zaphod@slartibartfast.com',
          'phone' => array('home' => '555-444-3333',
                           'cell' => '555-111-1234')));

$smarty->display('index.tpl');

index.tpl:

{$Contacts.fax}<br>
{$Contacts.email}<br>
{* możesz również wyświetlac tablice zawartą w tablicy *}
{$Contacts.phone.home}<br>
{$Contacts.phone.cell}<br>

OUTPUT:

555-222-9876<br>
zaphod@slartibartfast.com<br>
555-444-3333<br>
555-111-1234<br>
```

2.3. Indeksy tablic

Możesz odwoływać się do tablic przez ich indeks, tak samo jak w PHP.

Example 3.12. Odwoływanie się do tablic przez index

```
index.php:

$smarty = new Smarty;
$smarty->assign('Contacts',
    array('555-222-9876',
          'zaphod@slartibartfast.com',
          array('555-444-3333',
                '555-111-1234')));

$smarty->display('index.tpl');

index.tpl:

{$Contacts[0]}<br>
{$Contacts[1]}<br>
{* możesz również wyświetlac tablice zawartą w tablicy *}
{$Contacts[2][0]}<br>
{$Contacts[2][1]}<br>

OUTPUT:
```

```
555-222-9876<br>
zaphod@slartibartfast.com<br>
555-444-3333<br>
555-111-1234<br>
```

2.4. Obiekty

Do własności obiektów przyporządkowanych z PHP można się odwoływać podając nazwę własności po symbolu '->'.

Example 3.13. Odwoływanie się do własności obiektu

```
name: {$person->name}<br>
email: {$person->email}<br>

OUTPUT:

name: Zaphod Beeblebrox<br>
email: zaphod@slartibartfast.com<br>
```

2.5. Zmienne ładowane z plików konfiguracyjnych

Do zmiennych ładowanych z plików konfiguracyjnych można się odwoływać przez zamknięcie ich w znaki hasch (#), albo ze zmiennej Smarty $smarty.config. Drugi sposób jest użyteczny przy osadzaniu w cytowanych wartościach atrybutów.

Example 3.14. Zmienne konfiguracyjne

```
foo.conf:


pageTitle = "This is mine"
bodyBgColor = "#eeeeee"
tableBorderSize = "3"
tableBgColor = "#bbbbbb"
rowBgColor = "#cccccc"

index.tpl:

{config_load file="foo.conf"}
<html>
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
<tr bgcolor="{#rowBgColor#}">
        <td>First</td>
        <td>Last</td>
        <td>Address</td>
</tr>
</table>
</body>
</html>

index.tpl: (alternate syntax)

{config_load file="foo.conf"}
```

```
<html>
<title>{$smarty.config.pageTitle}</title>
<body bgcolor="{$smarty.config.bodyBgColor}">
<table border="{$smarty.config.tableBorderSize}"
bgcolor="{$smarty.config.tableBgColor}">
<tr bgcolor="{$smarty.config.rowBgColor}">
        <td>First</td>
        <td>Last</td>
        <td>Address</td>
</tr>
</table>
</body>
</html>


OUTPUT: (same for both examples)

<html>
<title>This is mine</title>
<body bgcolor="#eeeeee">
<table border="3" bgcolor="#bbbbbb">
<tr bgcolor="#cccccc">
        <td>First</td>
        <td>Last</td>
        <td>Address</td>
</tr>
</table>
</body>
</html>
```

mienne pliku konfiguracyjnego nie mogą być używane dopóki nie zostaną załadowane z pliku konfiguracyjnego. Ta procedura zostanie wyjaśniona później podczas omawiania {config_load}.

2.6. {$smarty} zarezerwowana zmienna

Ta zarezerwowana zmienna {$smarty} może być użyta do dostępu do kilku specjalnych zmiennych szablonu.

2.6.1. Zmienne Request

Do zmiennych Request takie jak get, post, server, enviroment i session można uzyskać dostęp tak jak na przykładzie poniżej.

Example 3.15. Wyświetlanie zmiennych typu Request

```
{* display value of page from URL (GET)
http://www.domain.com/index.php?page=foo *}
{$smarty.get.page}

{* display the variable "page" from a form a form (POST) *}
{$smarty.post.page}

{* display the value of the cookie "username" *}
{$smarty.cookies.username}

{* display the server variable "SERVER_NAME" *}
```

```
{$smarty.server.SERVER_NAME}

{* display the system environment variable "PATH" *}
{$smarty.env.PATH}

{* display the php session variable "id" *}
{$smarty.session.id}

{* display the variable "username" from merged get/post/cookies/server/env
*}
{$smarty.request.username}
```

2.6.2. {$smarty.now}

Dostęp do czasu w danej chwili może być możliwy z {$smarty.now}. Numer ten przedstawia liczbę sekund od 1 stycznia 1970 i może być skierowany do modyfikatora date_format aby wyświetlić go w odpowiedniej formie.

Example 3.16. Użycie {$smarty.now}

```
{* use the date_format modifier to show current date and time *}
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

2.6.3. {$smarty.const}

Możesz uzyskać dostęp do stałych z PHP.

Example 3.17. Użycie {$smarty.const}

```
{$smarty.const._MY_CONST_VAL}
```

2.6.4. {$smarty.capture}

Do wyjścia przechwyconego przez {capture}..{/capture} można uzyskać dostęp przez zmienną {$smarty.capture}.

2.6.5. {$smarty.config}

Zmienna {$smarty} może być używana do kierowania do zmiennych konfiguracyjnych. {$smarty.config.foo} jest synonimem dla {#foo#}.

2.6.6. {$smarty.section}, {$smarty.foreach}

Zmienna {$smarty} może być używana do sterowania pętlami 'section' i 'foreach' .

2.6.7. {$smarty.template}

Ta zmienna zawiera nazwę obecnie przetwarzanego szablonu.

2.7. Modyfikatory zmiennych

Modyfikatory zmiennych mogą być stosowane do zmiennych, zdefiniowanych funkcji albo ciągów znaków. Aby zastosować modyfikator, podaj ziemną za którą umieść znak | (pipe) i potem nazwę modyfikatora. Modyfikator może akceptować dodatkowe parametry określające jego zachowanie. Te parametry podaje się za nazwą modyfikatora i oddzielając je przez znak : (dwukropek)

Example 3.18. Przykład modyfikatorów

```
{* Uppercase the title *}
<h2>{$title|upper}</h2>

{* Truncate the topic to 40 characters use ... at the end *}
Topic: {$topic|truncate:40:"..."}

{* format a literal string *}
{"now"|date_format:"%Y/%m/%d"}

{* apply modifier to a custom function *}
{mailto|upper address="me@domain.dom"}
```

Jeśli zastosujesz modyfikator do tablicy, modyfikator zostanie zastosowany do każdego elementu tablicy. Jeśli chcesz aby modyfikator pracował na tablicy (całej jako jednej zmiennej) musisz poprzedzić nazwę modyfikatora symbolem @ np {$articleTitle|@count}(to wyświetli liczbę elementów w tablicy $articleTitle)

2.7.1. capitalize

Służy do zmieniania pierwszej litery każdego słowa w zmiennej na dużą.

Example 3.19. capitalize

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Police begin campaign to rundown
jaywalkers.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|capitalize}

OUTPUT:

Police begin campaign to rundown jaywalkers.
Police Begin Campaign To Rundown Jaywalkers.
```

2.7.2. count_characters

Używany jest do liczenia liczby znaków w zmiennej.

Example 3.20. count_characters

```
index.php:
```

```
$smarty = new Smarty;
$smarty->assign('articleTitle', 'Cold Wave Linked to Temperatures.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_characters}

OUTPUT:

Cold Wave Linked to Temperatures.
32
```

2.7.3. cat

Ten modyfikator służy dodawaniu ciągu znaków na końcu zmiennej.

Pozycja parametru	Typ	Wymagany	cat	Opis
1	string	nie	pusty	Wartośc która zostanie dodana do podanej zmiennej.

Example 3.21. cat

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Psychics predict world didn't end');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|cat:" yesterday."}

OUTPUT:

Psychics predict world didn't end yesterday.
```

2.7.4. count_characters

Zlicza ilość znaków w podanej zmiennej.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	boolean	nie	false	Decyduje o tym czy spacje wliczane są do całkowitej liczby znaków.

Example 3.22. count_characters

```
index.php:

$smarty->assign('articleTitle', 'Cold Wave Linked to Temperatures.');
```

```
index.tpl:

{$articleTitle}
{$articleTitle|count_characters}
{$articleTitle|count_characters:true}

OUTPUT:

Cold Wave Linked to Temperatures.
29
33
```

2.7.5. count_paragraphs

Zlicza liczbę akapitów w zmiennej.

Example 3.23. count_paragraphs

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'War Dims Hope for Peace. Child's Death
Ruins Couple's Holiday.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_paragraphs}

OUTPUT:

War Dims Hope for Peace. Child's Death Ruins Couple's Holiday.

Man is Fatally Slain. Death Causes Loneliness, Feeling of Isolation.
2
```

2.7.6. count_sentences

Zlicza ilość zdań w zmiennej.

Example 3.24. count_sentences

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Soviet Ships Collide - One Dies.
Enraged Cow Injures Farmer with Axe.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_sentences}

OUTPUT:
```

```
Two Soviet Ships Collide - One Dies. Enraged Cow Injures Farmer with Axe.
2
```

2.7.7. count_words

Zlicza ilość wyrazów w zmiennej.

Example 3.25. count_words

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Dealers Will Hear Car Talk at Noon.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_words}

OUTPUT:

Dealers Will Hear Car Talk at Noon.
7
```

2.7.8. date_format

Formatuje datę i czas do danego według podanego formatu strftime() . Data może zostać przekazana do Smarty jako uniksowe timestamps, mysql timestamps lub przez dowolny ciąg znaków złożony z miesiąca dnia roku (możliwego do sparsowania przez strtotime()). Projektanci mogą używać date_format do zachowania kontroli nad formatowaniem daty. Jeśli data przekazana do date_format jest pusta i drugi parametr jest podany, zostanie on uzyty jako data do sformatowania.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	nie	%b %e, %Y	Format wyjściowy daty.
2	string	nie	brak	Domyślna data, jesli data wejściowa jest pusta.

> **Note**
>
> Od Smarty-2.6.10 wszystkie wartosci numeryczne (z wyjatkiem mysql timestamps) przekazywane do date_format zawsze będą interpretowane jako unix timestamp.

Example 3.26. date_format

```
index.php:

$smarty = new Smarty;
$smarty->assign('yesterday', strtotime('-1 day'));
```

```
$smarty->display('index.tpl');

index.tpl:


{$smarty.now|date_format}
{$smarty.now|date_format:"%A, %B %e, %Y"}
{$smarty.now|date_format:"%H:%M:%S"}
{$yesterday|date_format}
{$yesterday|date_format:"%A, %B %e, %Y"}
{$yesterday|date_format:"%H:%M:%S"}

OUTPUT:

Feb 6, 2001
Tuesday, February 6, 2001
14:33:00
Feb 5, 2001
Monday, February 5, 2001
14:33:00
```

Example 3.27. ddane konwersji datydate_format

- %a - abbreviated weekday name according to the current locale
- %A - full weekday name according to the current locale
- %b - abbreviated month name according to the current locale
- %B - full month name according to the current locale
- %c - preferred date and time representation for the current locale
- %C - century number (the year divided by 100 and truncated to an integer, range 00 to 99)
- %d - day of the month as a decimal number (range 00 to 31)
- %D - same as %m/%d/%y
- %e - day of the month as a decimal number, a single digit is preceded by a space (range 1 to 31)
- %g - Week-based year within century [00,99]
- %G - Week-based year, including the century [0000,9999]
- %h - same as %b
- %H - hour as a decimal number using a 24-hour clock (range 00 to 23)
- %I - hour as a decimal number using a 12-hour clock (range 01 to 12)
- %j - day of the year as a decimal number (range 001 to 366)
- %k - Hour (24-hour clock) single digits are preceded by a blank. (range 0 to 23)
- %l - hour as a decimal number using a 12-hour clock, single digits preceeded by a space (range 1 to 12)
- %m - month as a decimal number (range 01 to 12)
- %M - minute as a decimal number
- %n - newline character
- %p - either `am' or `pm' according to the given time value, or the corresponding strings for the current locale
- %r - time in a.m. and p.m. notation
- %R - time in 24 hour notation
- %S - second as a decimal number
- %t - tab character
- %T - current time, equal to %H:%M:%S
- %u - weekday as a decimal number [1,7], with 1 representing Monday

- %U - week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week
- %V - The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week.
- %w - day of the week as a decimal, Sunday being 0
- %W - week number of the current year as a decimal number, starting with the first Monday as the first day of the first week
- %x - preferred date representation for the current locale without the time
- %X - preferred time representation for the current locale without the date
- %y - year as a decimal number without a century (range 00 to 99)
- %Y - year as a decimal number including the century
- %Z - time zone or name or abbreviation
- %% - a literal `%' character

2.7.9. default

Jest używany do nadania zmiennej wartości domyślnej. Jeśli zmienna jest pusta lub nie zdefiniowana wyświetlana jest wartość domyślna. Default wymaga jednego argumentu.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	nie	pusty	Domyślna wartość wyjścia jeżeli zmienna jest pusta.

Example 3.28. default

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Dealers Will Hear Car Talk at Noon.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|default:"no title"}
{$myTitle|default:"no title"}

OUTPUT:

Dealers Will Hear Car Talk at Noon.
no title
```

2.7.10. escape

Używany jest do html escape, url escape, escape single quotes on a variable not already escaped, hex escape, hexentity or javascript escape. Domyślnie, modyfikator przyjmuje argument html . ???????

Pozycja parametru	Typ	Wymagany	Mozliwe wartości	Domyślnie	Opis

Pozycja parametru	Typ	Wymagany	Mozliwe wartości	Domyślnie	Opis
1	string	nie	html, htmlall, url, quotes, hex, hexentity, javascript	html	Jakiego typu "eskejpowanie" znaków zostanie zastosowane.

Example 3.29. escape

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "'Stiff Opposition Expected to Casketless
Funeral Plan'");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|escape}
{$articleTitle|escape:"html"}    {* escapes  & " ' < > *}
{$articleTitle|escape:"htmlall"} {* escapes ALL html entities *}
{$articleTitle|escape:"url"}
{$articleTitle|escape:"quotes"}
<a
href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity
"}</a>

OUTPUT:

'Stiff Opposition Expected to Casketless Funeral Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff+Opposition+Expected+to+Casketless+Funeral+Plan'
\'Stiff Opposition Expected to Casketless Funeral Plan\'
<a
href="mailto:%62%6f%62%40%6d%65%2e%6e%65%74">&#x62;&#x6f;&#x62;&#x40;&#x6d;
&#x65;&#x2e;&#x6e;&#x65;&#x74;</a>
```

2.7.11. ident

Wcina ciąg znaków w każdej lini, domyślna wartość to 4. Jako opcjonalny drugi argument, możesz podać parametr jakim będzie się posługiwał modyfikator aby stworzyć wycięcie. (Używaj "\t" dla tabulacji.)

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	integer	nie	4	Decyduje o tym ile znaków bedzie posiadało wciecie
2	string	nie	jedna spacja	Znak używany do stworzenia wciecia.

Example 3.30. ident

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'NJ judge to rule on nude beach.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}

{$articleTitle|indent}

{$articleTitle|indent:10}

{$articleTitle|indent:1:"\t"}

OUTPUT:

NJ judge to rule on nude beach.
Sun or rain expected today, dark tonight.
Statistics show that teen pregnancy drops off significantly after 25.

    NJ judge to rule on nude beach.
    Sun or rain expected today, dark tonight.
    Statistics show that teen pregnancy drops off significantly after 25.

          NJ judge to rule on nude beach.
          Sun or rain expected today, dark tonight.
          Statistics show that teen pregnancy drops off significantly after
25.

         NJ judge to rule on nude beach.
         Sun or rain expected today, dark tonight.
         Statistics show that teen pregnancy drops off significantly after
25.
```

2.7.12. lower

Jest używany do zmiany wszystkich liter na małe.

Example 3.31. lower

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Convicts Evade Noose, Jury Hung.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|lower}

OUTPUT:

Two Convicts Evade Noose, Jury Hung.
two convicts evade noose, jury hung.
```

2.7.13. nl2br

W podanej zmiennej wszystkie łamania linii (entery) będą przekształcone do znaczników
. Jest to ekwiwalent funkcji PHP - nl2br().

Example 3.32. nl2br

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Sun or rain expected\ntoday, dark
tonight");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|nl2br}

OUTPUT:

Sun or rain expected<br />today, dark tonight
```

2.7.14. regex_replace

Wyrażenie regularne przeszukujące i zamieniające w zmiennej. Używaj wyrażeń dla preg_replace() z manuala PHP.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	tak	brak	Wyrażenie regularne które będzie zastąpione.
2	string	tak	brak	Tekst do wstawienia w miejsce znalezionych wyrażeń.

Example 3.33. regex_replace

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Infertility unlikely to\nbe passed on,
experts say.");
$smarty->display('index.tpl');

index.tpl:

{* replace each carriage return, tab & new line with a space *}

{$articleTitle}
{$articleTitle|regex_replace:"/[\r\t\n]/":" "}

OUTPUT:

Infertility unlikely to
 be passed on, experts say.
Infertility unlikely to be passed on, experts say.
```

2.7.15. replace

Proste szukanie i zamiana w zmiennej.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	tak	brak	Ciągi znaków które będą zastąpione.
2	string	tak	brak	Tekst do wstawienia w miejsce znalezionych ciągów.

Example 3.34. replace

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Child's Stool Great for Use in Garden.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|replace:"Garden":"Vineyard"}
{$articleTitle|replace:" ":"   "}

OUTPUT:

Child's Stool Great for Use in Garden.
Child's Stool Great for Use in Vineyard.
Child's   Stool   Great   for   Use   in   Garden.
```

2.7.16. spacify

spacify umożliwia wstawienie spacji pomiędzy każdym znakiem w zmiennej. Możesz opcjonalnie podać inny znak (lub ciąg znaków) do wstawienia.

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	nie	jedna spacja	Ten parametr będzie umieszczony pomiędzy każda parą znaków.

Example 3.35. spacify

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Something Went Wrong in Jet Crash, Experts
Say.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|spacify}
{$articleTitle|spacify:"^^"}

OUTPUT:
```

```
Something Went Wrong in Jet Crash, Experts Say.
S o m e t h i n g  W e n t  W r o n g  i n  J e t  C r a s h ,  E x p
e r t s  S a y .
S^^o^^m^^e^^t^^h^^i^^n^^g^^ ^^W^^e^^n^^t^^ ^^W^^r^^o^^n^^g^^ ^^i^^n^^
^^J^^e^^t^^ ^^C^^r^^a^^s^^h^^,^^ ^^E^^x^^p^^e^^r^^t^^s^^ ^^S^^a^^y^^.
```

2.7.17. string_format

Umożliwia formatowanie ciągów znaków takich jak ułamki dziesiętne i podobnych. Do formatowania używaj wyrażeń do sprintf .

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	string	tak	jedna spacja	Format tekstu (sprintf)

Example 3.36. string_format

```
index.php:

$smarty = new Smarty;
$smarty->assign('number', 23.5787446);
$smarty->display('index.tpl');

index.tpl:

{$number}
{$number|string_format:"%.2f"}
{$number|string_format:"%d"}

OUTPUT:

23.5787446
23.58
24
```

2.7.18. strip

Zastępuje wszystkie powtarzające się spacje, łamanie linii i tabulatory pojedynczą spacją, albo podanym ciągiem znaków .

> **Note**
>
> Jeżeli chcesz użyć tego modyfikatora stosunku do bloków tekstu w szablonie, użyj strip function.

Example 3.37. strip

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Grandmother of\neight makes\t    hole in
one.");
$smarty->display('index.tpl');

index.tpl:
```

```
{$articleTitle}
{$articleTitle|strip}
{$articleTitle|strip:" "}

OUTPUT:

Grandmother of
eight makes        hole in one.
Grandmother of eight makes hole in one.
Grandmother of eight makes hole in one.
```

2.7.19. strip_tags

Wycina i oznacza tekst z pomiędzy znaczników, domyślnie spomiędzy < i >.

Example 3.38. strip_tags

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind Woman Gets <font
face=\"helvetica\">New Kidney</font> from Dad she Hasn't Seen in
<b>years</b>.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|strip_tags}

OUTPUT:

Blind Woman Gets <font face="helvetica">New Kidney</font> from Dad she
Hasn't Seen in <b>years</b>.
Blind Woman Gets New Kidney from Dad she Hasn't Seen in years.
```

2.7.20. truncate

Obcina zmienną do podanej długości (domyślnie do 80 znaków). Jako opcjonalny drugi parametr możesz podać ciąg znaków wyświetlanych na końcu obciętej zmiennej. Znaki w podanym ciągu znaków są wliczone do oryginalnej długości przycinania. Domyślnie, obcięcie następuje przy końcu ostatniego słowa. Jeśli chcesz przyciąć dokładnie do takiej długości jaką podałeś podaj opcjonalny trzeci parametr jako true (prawda).

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	integer	nie	80	Decyduje ile ciąg może mieć znaków.
2	string	nie	...	Tekst do wstawienia na końcu wyswietlonego obcietego textu.
3	boolean	nie	fałsz	Decydujeo tym czy pzepuścic ostatni wyraz czy obcinac tekst dokładnie do podanej liczby znaków.

Example 3.39. truncate

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Sisters Reunite after Eighteen Years
at Checkout Counter.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|truncate}
{$articleTitle|truncate:30}
{$articleTitle|truncate:30:""}
{$articleTitle|truncate:30:"---"}
{$articleTitle|truncate:30:"":true}
{$articleTitle|truncate:30:"...":true}

OUTPUT:

Two Sisters Reunite after Eighteen Years at Checkout Counter.
Two Sisters Reunite after Eighteen Years at Checkout Counter.
Two Sisters Reunite after...
Two Sisters Reunite after
Two Sisters Reunite after---
Two Sisters Reunite after Eigh
Two Sisters Reunite after E...
```

2.7.21. upper

Używany jest do zwiększenia wszystkich liter w zmiennej na duże.

Example 3.40. upper

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "If Strike isn't Settled Quickly it may
Last a While.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|upper}

OUTPUT:

If Strike isn't Settled Quickly it may Last a While.
IF STRIKE ISN'T SETTLED QUICKLY IT MAY
```

2.7.22. wordwrap

Pakuje zmienną do kolumny, domyślna szerokość kolumny to 80. Jako opcjonalny drugi parametr, możesz podać ciąg znaków który będzie przenosił tekst do nowej linii (domyślnie jest to return \n). Domyślnie przenoszenie do nowej linii na końcu ostatniego wyrazu

mieszczącego się w linii. Jeśli chcesz przyciąć dokładnie do takiej długości jaką podałeś podaj opcjonalny trzeci parametr jako true (prawda).

Pozycja parametru	Typ	Wymagany	Domyślnie	Opis
1	integer	nie	80	Decyduje co ile znaków wstawiać zdefiniowany tekst.
2	string	nie	\n	Tekst do wstawienia co okreslona ilośc znaków.
3	boolean	nie	fałsz	Decydujeo tym czy przepuścić ostatni wyraz czy wstawić znak dokładnie po podanej liczby znaków.

Example 3.41. wordwrap

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind woman gets new kidney from dad she
hasn't seen in years.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}

{$articleTitle|wordwrap:30}

{$articleTitle|wordwrap:20}

{$articleTitle|wordwrap:30:"<br>\n"}

{$articleTitle|wordwrap:30:"\n":true}

OUTPUT:

Blind woman gets new kidney from dad she hasn't seen in years.

Blind woman gets new kidney
from dad she hasn't seen in
years.

Blind woman gets new
kidney from dad she
hasn't seen in
years.

Blind woman gets new kidney<br>
from dad she hasn't seen in years.

Blind woman gets new kidney fr
om dad she hasn't seen in year
s.
```

2.8. Mieszanie modyfikatorów

Możesz zastosować dowolna liczbę modyfikatorów do zmiennej. Będą wykonywane w kolejności w jakiej zostały wywołane (z lewej na prawą). Muszą być oddzielone znakiem | (pipe).

Example 3.42. mieszanie modyfikatorów

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Smokers are Productive, but Death Cuts
Efficiency.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|upper|spacify}
{$articleTitle|lower|spacify|truncate}
{$articleTitle|lower|truncate:30|spacify}
{$articleTitle|lower|spacify|truncate:30:". . ."}


OUTPUT:

Smokers are Productive, but Death Cuts Efficiency.
S M O K E R S  A R E  P R O D U C T I V E ,  B U T  D E A T H  C U T S
E F F I C I E N C Y .
s m o k e r s  a r e  p r o d u c t i v e ,  b u t  d e a t h  c u t
s...
s m o k e r s  a r e  p r o d u c t i v e ,  b u t . . .
s m o k e r s  a r e  p. . .
s.
```

3. Funkcje wbudowane

Smarty dostarcza kilka wbudowanych funkcji. Funkcje wbudowane są integralną częścią języka szablonowego. Nie możesz stworzyć funkcji o takich samych nazwach, nie możesz modyfikować wbudowanych funkcji.

3.1. {capture}

Jest wykorzystywana do kolekcjonowania danych wyjściowych szablonu do zmiennej zamiast wyświetlania ich. Dowolna treść pomiędzy {capture name="foo"} i {/capture} jest kolekcjonowana do zmiennej podanej jako atrybut name. Przechwycona treść może zostać wykorzystana dzięki specjalnej zmiennej - $smarty.capture.foo gdzie foo jest wartością podaną w atrybucie name. Jeśli nie podasz atrybutu name to wtedy zostanie użyta wartość "default". Wszystkie komendy {capture} muszą zostać zamknięte przez {/capture}. Możesz zagnieżdżać komendy capture .

> **Note**
>
> Smarty 1.4.0 - 1.4.4 umieszczają przechwyconą treść w zmiennej nazwanej $return. Tak jak w 1.4.5, Takie zachowanie zostało zmienione aby używać atrybutu name.

Caution

Bądź ostrożny podczas przechwytywania danych wyjściowych {insert}. Jeśli jest włączone caschowanie a ty masz komendy {insert} po których oczekujesz że będą uruchamiane z zcaschowaną treścią nie przechwytuj tej treści.

Example 3.43. przechwytywanie treści szablonu

```
{* we don't want to print a table row unless content is displayed *}
{capture name=banner}
{include file="get_banner.tpl"}
{/capture}
{if $smarty.capture.banner ne ""}
        <tr>
                <td>
                        {$smarty.capture.banner}
                </td>
        </tr>
{/if}
```

3.2. {config_load}

Ta funkcja jest używana do załadowania zmiennych z pliku konfiguracyjnego do szablonu

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
file	string	tak	brak	Nazwa pliku konfiguracyjnego
section	string	nie	brak	Nazwa sekcji do załadowania
scope	string	nie	local	Decyduje o zasiegu załadowanych zmiennych. local - oznacza że zmienne dostępne są tylko lokalnie, parent - oznacza że zmienne dostepne są dla szablonu który je załadował i dla jego rodzica, global - oznacza że zmienne sa widoczne we wszystkich szablonach.
global	boolean	nie	fałsz	Decyduje czy zmienne są dostepne dla rodzica. Zastapione przesz scope.

Example 3.44. config_load

```
{config_load file="colors.conf"}

<html>
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
        <tr bgcolor="{#rowBgColor#}">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
        </tr>
</table>
</body>
</html>
```

Pliki konfiguracyjne mogą zawierać sekcje. możesz załadować zmienne z danej sekcji przez dodanie atrybutu section.

> **Note**
>
> Sekcje pliku konfiguracyjnego i wbudowana funkcja section nie mają nic wspólnego ze sobą, to tylko podobieństwo w nazwach.

Example 3.45. config_load z sekcjami

```
{config_load file="colors.conf" section="Customer"}

<html>
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
        <tr bgcolor="{#rowBgColor#}">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
        </tr>
</table>
</body>
</html>
```

3.3. {foreach}, {foreachelse}

Pętle foreach są alternatywą dla pętli section. foreach jest używana do pętli na tablicy asocjacyjnej. Wyrażenie dla foreach jest znacznie łatwiejsze niż section, ale za tą cenę może być wykorzystywana tylko na pojedynczej tablicy. Tagi foreach muszą być zakończone tagami /foreach. Wymagane parametry to from i item. Nazwa pętli foreach jest dowolna (zbudowana z liter, cyfr i podkreślników). Pętla foreach może być zagnieżdżona i nazwa zagnieżdżonej foreach musi być unikalna - różnić się od każdej innej nazwy foreach . Zmienna from (zazwyczaj tablica wartości) będzie w pętli foreach dopuki nie skończą się elemęty tablicy. foreachelse jest wykonywana wtedy jeżeli nie ma wartości w zmiennej from .

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
from	string	tak	brak	Nazwa tablicy po której iterujemy.
item	string	tak	brak	Nazwa zmiennej zawierającej obecny element.
key	string	nie	brak	Nazwa zmiennej zawierającej obecny klucz.
name	string	nie	brak	Nazwa pętli uzywana aby dostać się do jej właściwości.

Example 3.46. foreach

```
{* this Przykład will print out all the values of the $custid array *}
{foreach from=$custid item=curr_id}
        id: {$curr_id}<br>
```

```
{/foreach}

OUTPUT:

id: 1000<br>
id: 1001<br>
id: 1002<br>
```

Example 3.47. foreach key

```
{* The key contains the key for each looped value

assignment looks like this:

$smarty->assign("contacts", array(array("phone" => "1", "fax" => "2",
"cell" => "3"),
      array("phone" => "555-4444", "fax" => "555-3333", "cell" => "760-
1234")));

*}

{foreach name=outer item=contact from=$contacts}
  {foreach key=key item=item from=$contact}
    {$key}: {$item}<br>
  {/foreach}
{/foreach}

OUTPUT:

phone: 1<br>
fax: 2<br>
cell: 3<br>
phone: 555-4444<br>
fax: 555-3333<br>
cell: 760-1234<br>
```

3.4. {include}

Include jest wykorzystywana do dołączania innych szablonów do obecnego szablonu. Każda zmienna dostępna w obecnym szablonie jest także dostępna w szablonie dołączanym. Znacznik include musi posiadać atrybut file do którego podawana jest ścieżka do dołączanego pliku. Możesz opcjonalnie podać atrybut assign, w którym podajesz nazwę zmiennej do której ma być załadowany szablon zamiast wyświetlać go.

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
file	string	tak	brak	Nazwa pliku szablonu do dołączenia.
assign	string	nie	brak	Nazwa zmiennej zawierającej wynik(wyjście) dołączonego pliku szablonu.
[var ...]	[var type]	nie	brak	Zmienna do przesłania do dołączanego szablonu.

Example 3.48. include

```
{include file="header.tpl"}

{* body of template goes here *}

{include file="footer.tpl"}
```

Możesz także podać zmienne do dołączanego szablonu jako atrybuty. Dowolna zmienna podana jako atrybut jest widziana jedynie w obrębie dołączanego pliku. Zmienne podane w atrybutach nadpisują dotychczasowe zmienne szablonu na wypadek takich samych nazw.

Example 3.49. Funkcja include z przekazywaniem zmiennych

```
{include file="header.tpl" title="Main Menu" table_bgcolor="#c0c0c0"}

{* body of template goes here *}

{include file="footer.tpl" logo="http://my.domain.com/logo.gif"}
```

Używaj wyrażenia dla template resources do dołączania plików z zewnątrz katalogu $template_dir. ?????

Example 3.50. foreach key

```
{* absolute filepath *}
{include file="/usr/local/include/templates/header.tpl"}

{* absolute filepath (same thing) *}
{include file="file:/usr/local/include/templates/header.tpl"}

{* windows absolute filepath (MUST use "file:" prefix) *}
{include file="file:C:/www/pub/templates/header.tpl"}

{* include from template resource named "db" *}
{include file="db:header.tpl"}
```

3.5. {include_php}

Znaczniki include_php są używane do dołączania skryptów php do szablonu. Jeśli ochrona jest włączona wtedy skrypt php musi być zlokalizowany w zaufanym katalogu o ścieżce $trusted_dir . Znacznik include_php musi posiadać atrybut "file", który zawiera ścieżkę do dołączanego pliku, lub względną do $trusted_dir, albo ścieżkę absolutną. include_php to przyjemny sposób do ??? handle componentized templates, i trzymanie kodu PHP poza plikami szablonów. Powiedzmy że twój szablon wyświetla nawigacje strony, która jest dynamiczne tworzona z bazy danych. Możesz zachować logikę php która pobiera dane z bazy danych w oddzielnym katalogu i dołączanie na początku twojego szablonu. Teraz możesz dołączać ten szablon gdziekolwiek nie martwiąc się czy dane z bazy danych zostały przyporządkowane przez aplikacje. Domyślnie pliki php są dołączane tylko jeden raz nawet jeśli są wywoływane w szablonie kilka razy. Możesz zażądać ładowania pliku za każdym razem jeśli w wywołaniu funkcji Include_php podasz atrybut once. Raz ustawiając once na false skrypt będzie dołączany za każdym razem kiedy jest wywołany w szablonie. Opcjonalnie możesz podać atrybut assign , która zamiast wyświetlać przyporządkuje dane wyjściowe include_php do zmiennej. Obiekt smarty jest dostępny jako $this w skrypcie PHP który dołączamy.

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
file	string	tak	brak	Nazwa pliku php do dołączenia.
once	boolean	nie	prawda	Decyduje o tym czy dołączac plik tylko raz czy wiele razy.
assign	string	nie	brak	Nazwa zmiennej do której zostanie przypisany wynik dziaałania dołączoego pliku

Example 3.51. include_php

```
load_nav.php
-------------

<?php

    // load in variables from a mysql db and assign them to the template
    require_once("MySQL.class.php");
    $sql = new MySQL;
    $sql->query("select * from site_nav_sections order by name",SQL_ALL);
    $this->assign('sections',$sql->record);

?>


index.tpl
---------

{* absolute path, or relative to $trusted_dir *}
{include_php file="/path/to/load_nav.php"}

{foreach item="curr_section" from=$sections}
        <a href="{$curr_section.url}">{$curr_section.name}</a><br>
{/foreach}
```

3.6. {insert}

Znaczniki Insert działają bardzo podobnie jak znaczniki include, z wyjątkiem tego że znaczniki nie są caschowane kiedy włączone jest włączone caschowanie szablonów (caching). Będą wykonywane podczas każdego wywołania szablonu. Powiedzmy że posiada szablon z miejscem na baner na górze strony. Baner może posiadać treść zmieszaną z HTML, obrazów, flash, itp. więc nie możemy usytuować tu statycznego linka, i nie chcemy aby treść była caschowana razem ze stroną. W następującym znaczniku insert szablon zna wartości #banner_location_id# i #site_id# (pobrane z pliku konfiguracyjnego) i potrzebuje wywołać funkcję aby pobrać treść bannera.

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
name	string	tak	brak	Nazwa funkcji.
assign	string	nie	brak	Nazwa zmiennej zawierającej wynik(wyjście) dołączonego pliku szablonu.

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
script	string	nie	brak	Nazwa skryptu php który jest ddołączany i wywoływany przed wywołaniem funkcji insert
[var ...]	[var type]	nie	brak	Zmienna do przesłania do dołączanego szablonu.

Example 3.52. insert

```
{* Przykład of fetching a banner *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

W tym przykładzie używamy nazwy (name) "getBaner" i podajemy parametry #banner_location_id# i #site_id#. Smarty będzie szukał funkcji o nazwie insert_getBanner() w twojej aplikacji PHP, przekazując wartości #banner_location_id# i #site_id# jako pierwszy argument w tablicy asocjacyjnej. W twojej aplikacji wszystkie nazwy funkcji insert muszą posiadać prefix "insert_" aby zażegnać możliwy konflikt nazw. Twoja funkcja insert_getBanner() powinna przetwarzać przekazane dane i zwracać wynik. Wynik jest wyświetlany w szablonie w miejsce znaczników insert. Na przykład, Smarty wywoła funkcje: insert_getBanner(array("lid" => "12345","sid" => "67890")); i wyświetli zwrócony wynik w miejsce znaczników insert. Jeśli podasz argument "assign", de wyjściowe znaczników insert będą przyporządkowane do zmiennej. NOTA: przyporządkowywanie danych wyjściowych do zmiennej nie jest zbyt użyteczne z włączonym caschowaniem. Jeśli podasz argument "script", ten skrypt php będzie dołączony (tylko raz) przed tym jak funkcja insert będzie wykonana. To jest przypadek kiedy funkcja może jeszcze nie istnieć i najpierw musi zostać dołączony skrypt php aby wszystko działało. Ścieżka może być zarówno absolutna jak i względna do $trusted_dir. Kiedy security jest włączone, skrypt musi się znajdować w $trusted_dir. Obiekt Smarty jest przekazywany jako drugi argument. Tym sposobem możesz odwoływać się i modyfikować informacje w obiekcie Smarty z funkcji insert .

> **Note**
>
> Jest możliwe aby porcje szablonu nie były caschowane. Jeśli masz włączone caching , znaczniki insert nie będą caschowane. Będą uruchamiane dynamicznie za każdym razem kiedy strona jest tworzona, nawet w caschowanych stronach. To działa dobrze dla rzeczy takich jak banery, wyniki wyszukiwania, itd.

3.7. {if}, {elseif}, {else}

Wyrażenia if w Smarty mają bardzo podobną składnie jak wyrażenia if w phps, z kilkoma dodanymi rzeczami dla silnika szablonu. Każdy if musi być zakończony przez /if. else i elseif są także dopuszczone. "eq", "ne","neq", "gt", "lt", "lte", "le", "gte" "ge","is even","is odd", "is not even","is not odd","not","mod","div by","even by","odd by","==","!=",">", "<","<=",">=" . Operatory muszą być oddzielone od innych elemętów spacjami.

Example 3.53. wyrażenia warunku

```
{if $name eq "Fred"}
        Welcome Sir.
{elseif $name eq "Wilma"}
        Welcome Ma'am.
{else}
        Welcome, whatever you are.
{/if}

{* an Przykład with "or" logic *}
{if $name eq "Fred" or $name eq "Wilma"}
        ...
{/if}

{* same as above *}
{if $name == "Fred" || $name == "Wilma"}
        ...
{/if}

{* the following syntax will NOT work, conditional qualifiers
   must be separated from surrounding elements by spaces *}
{if $name=="Fred" || $name=="Wilma"}
        ...
{/if}


{* parenthesis are allowed *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
        ...
{/if}

{* you can also embed php function calls *}
{if count($var) gt 0}
        ...
{/if}

{* test if values are even or odd *}
{if $var is even}
        ...
{/if}
{if $var is odd}
        ...
{/if}
{if $var is not odd}
        ...
{/if}

{* test if var is divisible by 4 *}
{if $var is div by 4}
        ...
{/if}

{* test if var is even, grouped by two. i.e.,
0=even, 1=even, 2=odd, 3=odd, 4=even, 5=even, etc. *}
{if $var is even by 2}
        ...
{/if}

{* 0=even, 1=even, 2=even, 3=odd, 4=odd, 5=odd, etc. *}
{if $var is even by 3}
        ...
{/if}
```

3.8. {ldelim}, {rdelim}

ldelim i rdelim są używane do wyświetlenia znaków ograniczających, w naszym przypadku "{" lub "}". Silni szablonów zawsze próbuje zinterpretować znaki ograniczające - więc to jest sposób obejścia tego.

Example 3.54. ldelim, rdelim

```
{* this will print literal delimiters out of the template *}

{ldelim}funcname{rdelim} is how functions look in Smarty!


OUTPUT:

{funcname} is how functions look in Smarty!
```

3.9. {literal}

Znaczniki literal pozwalają blokowi danych na używanie dowolnych znaków, bloki te nie są interpretowane przez silnik Smarty. Jest to użyteczne dla takich rzeczy jak sekcje javascript . Wszystko pomiędzy znacznikami {literal}{/literal} nie jest interpretowane ale wyświetlane takie jakie jest.

Example 3.55. literal

```
{literal}
        <script language=javascript>

                <!--
                        function isblank(field) {
                        if (field.value == '')
                                { return false; }
                        else
                                {
                                document.loginform.submit();
                                return true;
                                }
                        }
                // -->

        </script>
{/literal}
```

3.10. {php}

Znaczniki php pozwalają osadzić PHP w szablonie. They will not be escaped, regardless of the $php_handling setting???. Ta opcja przeznaczona jest tylko dla zaawansowanych użytkowników, normalnie jest niepotrzebna.

Example 3.56. php

```
{php}
                  // including a php script directly
                  // from the template.
```

```
                include("/path/to/display_weather.php");
{/php}
```

3.11. {section}, {sectionelse}

Sekcje szablonów są używane do zapętlenia tablic z danymi. Wszystkie znaczniki section muszą być zakończone przez znaczniki /section. Wymaganymi parametrami są name i loop. Nazwa (name) sekcji może być dowolna, złożona z liter, cyfr i podkreślników. Sekcje mogą być zagnieżdżone i nazwa sekcji zagnieżdżonych musi być unikalna dla każdej sekcji. Zmienna loop (zwykle tablica wartości) decyduje ile razy będzie wykonana pętla. Kiedy wyświetlamy zmienną z sekcji nazwa sekcji musi zostać podana w nawiasach kwadratowych zaraz po nazwie. sectionelse jest wykonywana jeśli w zmiennej loop nie ma wartości.

Nazwa parametru	Typ	Wymagany	Domyślnie	Opis
name	string	tak	brak	Nazwa sekcji.
loop	[$variable_name]	tak	brak	Nazwa zmiennej po której bedziemy iterować.
start	integer	nie	0	Pozycja z której zaczynamy iterowanie po tablicy, Jeśli nie została podana iteracja rozpocznie się od momentu "końca tablicy"
step	integer	nie	1	Krok użyty w iteracji.
max	integer	nie	1	Decyduje o maksymalnej wartości petli które sekcja może wykonać
show	boolean	nie	prawda	decyduje o tym czy wyświetlać tą sekcje.

Example 3.57. section

```
{* this Przykład will print out all the values of the $custid array *}
{section name=customer loop=$custid}
         id: {$custid[customer]}<br>
{/section}

OUTPUT:

id: 1000<br>
id: 1001<br>
id: 1002<br>
```

Example 3.58. zmienna pętli sekcji

```
{* the loop variable only determines the number of times to loop.
   you can access any variable from the template within the section.
   This Przykład assumes that $custid, $name and $address are all
   arrays containing the same number of values *}
{section name=customer loop=$custid}
         id: {$custid[customer]}<br>
         name: {$name[customer]}<br>
         address: {$address[customer]}<br>
```

```
        <p>
{/section}


OUTPUT:

id: 1000<br>
name: John Smith<br>
address: 253 N 45th<br>
<p>
id: 1001<br>
name: Jack Jones<br>
address: 417 Mulberry ln<br>
<p>
id: 1002<br>
name: Jane Munson<br>
address: 5605 apple st<br>
<p>
```

Example 3.59. nazwy sekcji

```
{* the name of the section can be anything you like,
   and it is used to reference the data within the section *}
{section name=mydata loop=$custid}
        id: {$custid[mydata]}<br>
        name: {$name[mydata]}<br>
        address: {$address[mydata]}<br>
        <p>
{/section}
```

Example 3.60. sekcjie zagnieżdżone

```
{* sections can be nested as deep as you like. With nested sections,
   you can access complex data structures, such as multi-dimensional
   arrays. In this example, $contact_type[customer] is an array of
   contact types for the current customer. *}
{section name=customer loop=$custid}
        id: {$custid[customer]}<br>
        name: {$name[customer]}<br>
        address: {$address[customer]}<br>
        {section name=contact loop=$contact_type[customer]}
                {$contact_type[customer][contact]}:
{$contact_info[customer][contact]}<br>
        {/section}
        <p>
{/section}


OUTPUT:

id: 1000<br>
name: John Smith<br>
address: 253 N 45th<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: john@mydomain.com<br>
<p>
id: 1001<br>
name: Jack Jones<br>
address: 417 Mulberry ln<br>
```

```
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jack@mydomain.com<br>
<p>
id: 1002<br>
name: Jane Munson<br>
address: 5605 apple st<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jane@mydomain.com<br>
<p>
```

Example 3.61. sekcje i tablice asocjacyjne

```
{* This is an Przykład of printing an associative array
   of data within a section *}
{section name=customer loop=$contacts}
         name: {$contacts[customer].name}<br>
         home: {$contacts[customer].home}<br>
         cell: {$contacts[customer].cell}<br>
         e-mail: {$contacts[customer].email}<p>
{/section}


OUTPUT:

name: John Smith<br>
home: 555-555-5555<br>
cell: 555-555-5555<br>
e-mail: john@mydomain.com<p>
name: Jack Jones<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jack@mydomain.com<p>
name: Jane Munson<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jane@mydomain.com<p>
```

Example 3.62. sectionelse

```
{* sectionelse will execute if there are no $custid values *}
{section name=customer loop=$custid}
         id: {$custid[customer]}<br>
{sectionelse}
         there are no values in $custid.
{/section}
```

Sekcje mają swoje własne zmienne do uchwytu własności sekcji. Są one identyfikowane jako np: {$smarty.section.sectionname.varname}.

> **Note**
>
> Od Smarty 1.5.0, wyrażenia zmiennych dla własności sekcji zostały zmienione z {%sectionname.varname%} na {$smarty.section.sectionname.varname}. Stare wyrażenia dalej są wspomagane ale w przykładach zobaczysz odwołania do nowych wyrażeń.

3.11.1. index

index jest używany do wyświetlania obecnego indexu pętli, zaczyna od zero (albo podanego atrybutu start) i jest inkrementowany o jeden (albo o wartość podanego atrybutu step)

> **Note**
>
> If the step and start section properties are not modified, then this works the same as the iteration section property, except it starts on 0 instead of 1.

Example 3.63. section index

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{/section}


OUTPUT:

0 id: 1000<br>
1 id: 1001<br>
2 id: 1002<br>
```

3.11.2. index_prev

index_prev jest używany do wyświetlenia poprzedniego indexu pętli, przy pierwszej pętli jest ustawiony na -1.

Example 3.64. section index_prev

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* FYI, $custid[customer.index] and $custid[customer] are
identical in meaning *}
{if $custid[customer.index_prev] ne $custid[customer.index]}
The customer id changed<br>
{/if}
{/section}


OUTPUT:

0 id: 1000<br>
The customer id changed<br>
1 id: 1001<br>
The customer id changed<br>
2 id: 1002<br>
The customer id changed<br>
```

3.11.3. index_next

index_next jest używany do wyświetlenia następnego indexu pętli. Przy ostatniej pętli jest on ciągle większy o jeden od obecnego indexu (uznaje ustawienia atrybutu step, jeśli jest podany.)

Example 3.65. section index_next

```
         {section name=customer loop=$custid}
         {$smarty.section.customer.index} id: {$custid[customer]}<br>
         {* FYI, $custid[customer.index] and $custid[customer] are
identical in meaning *}
         {if $custid[customer.index_next] ne $custid[customer.index]}
        The customer id will change<br>
         {/if}
         {/section}


         OUTPUT:

         0 id: 1000<br>
        The customer id will change<br>
         1 id: 1001<br>
        The customer id will change<br>
         2 id: 1002<br>
        The customer id will change<br>
```

3.11.4. iteration

iteration jest używany do wyświetlania obecnej interakcji w pętli..

> **Note**
>
> Nie jest wypełniony przez właściwości sekcji start, step i max, inaczej niż index property. Iteration także startuje od zamiast 0 tak jak index. rownum to alias do iteration, diałają identycznie.

Example 3.66. section iteration

```
         {section name=customer loop=$custid start=5 step=2}
         current loop iteration: {$smarty.section.customer.iteration}<br>
         {$smarty.section.customer.index} id: {$custid[customer]}<br>
         {* FYI, $custid[customer.index] and $custid[customer] are
identical in meaning *}
         {if $custid[customer.index_next] ne $custid[customer.index]}
        The customer id will change<br>
         {/if}
         {/section}


         OUTPUT:

         current loop iteration: 1
         5 id: 1000<br>
        The customer id will change<br>
         current loop iteration: 2
         7 id: 1001<br>
        The customer id will change<br>
         current loop iteration: 3
         9 id: 1002<br>
        The customer id will change<br>
```

3.11.5. first

first jest ustawiony na true jeżeli obecna interakcja sekcji jest pierwszą.

Example 3.67. section first

```
 {section name=customer loop=$custid}
 {if $smarty.section.customer.first}
<table>
 {/if}

 <tr><td>{$smarty.section.customer.index} id:
        {$custid[customer]}</td></tr>

 {if $smarty.section.customer.last}
</table>
 {/if}
 {/section}


 OUTPUT:

 <table>
 <tr><td>0 id: 1000</td></tr>
 <tr><td>1 id: 1001</td></tr>
 <tr><td>2 id: 1002</td></tr>
 </table>
```

3.11.6. last

last jest ustawiony na true jeżeli obecna interakcja sekcji jest ostatnią.

Example 3.68. section last

```
  {section name=customer loop=$custid}
 {if $smarty.section.customer.first}
<table>
 {/if}

 <tr><td>{$smarty.section.customer.index} id:
        {$custid[customer]}</td></tr>

 {if $smarty.section.customer.last}
</table>
 {/if}
 {/section}


 OUTPUT:

 <table>
 <tr><td>0 id: 1000</td></tr>
 <tr><td>1 id: 1001</td></tr>
 <tr><td>2 id: 1002</td></tr>
 </table>
```

3.11.7. rownum

rownum jest używany do wyświetlania obecnej interakcji w pętli, startuje od jedynki. To alias do iteration, działają identycznie.

Example 3.69. section rownum

```
{section name=customer loop=$custid}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br>
{/section}


OUTPUT:

1 id: 1000<br>
2 id: 1001<br>
3 id: 1002<br>
```

3.11.8. loop

loop jest używana do wyświetlenia ostatniego numeru indexu, który został po ostatniej pętli sekcji. Może być używany wewnątrz lub na zewnątrz sekcji.

Example 3.70. section loop

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{/section}

There were {$smarty.section.customer.loop} customers shown above.

OUTPUT:

0 id: 1000<br>
1 id: 1001<br>
2 id: 1002<br>

There were 3 customers shown above.
```

3.11.9. show

show jest używany jako parametr sekcji, jest to wartość boolowska (true/false). Jeśli parametr jest równy false sekcja nie zostanie wyświetlona. Jeśli jest obecne sectionelse, to zostanie wyświetlony.

Example 3.71. section show

```
{* $show_customer_info may have been passed from the PHP
application, to regulate whether or not this section shows *}
{section name=customer loop=$custid show=$show_customer_info}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br>
{/section}

{if $smarty.section.customer.show}
the section was shown.
{else}
the section was not shown.
{/if}


OUTPUT:
```

```
1 id: 1000<br>
2 id: 1001<br>
3 id: 1002<br>

the section was shown.
```

3.11.10. total

total jest używany do wyświetlenia liczby interakcji które odbyły się podczas gdy sekcja była w pętli. Może zostać użyty na zewnątrz lub w środku sekcji.

Example 3.72. section total

```
{section name=customer loop=$custid step=2}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{/section}

There were {$smarty.section.customer.total} customers shown above.

OUTPUT:

0 id: 1000<br>
2 id: 1001<br>
4 id: 1002<br>

There were 3 customers shown above.
```

3.12. {strip}

ZWiele razy projektanci web próbowali rozwiązać co zrobić ponieważ dodatkowe puste spacje i znaki łamania lini zwracają sztuczne dane wyjściowe z oddanego kodu HTML , musisz 'przysunąć do siebie' wszystkie twoje znaczniki w szablonie aby uzyskać pożądany rezultat. Zazwyczaj kończy się to nieczytelnym i nieedytowalnym szablonem. W Smary wszystko pomiędzy znacznikami {strip}{/strip} jest pozbawione pustych spacji i znaków łamania lini z początku i końca linii zanim zostaną wyświetlone. Tym sposobem możesz utrzymać czytelność szablonów i nie martwić się problemami z pustymi spacjami.

> **Note**
>
> strip}{/strip} nie zmienia treści zmiennych w szablonie.

Example 3.73. strip tags

```
{* the following will be all run into one line upon output *}
{strip}
<table border=0>
        <tr>
                <td>
                        <A HREF="{$url}">
                        <font color="red">This is a test</font>
                        </A>
                </td>
        </tr>
</table>
{/strip}
```

OUTPUT:

```
<table border=0><tr><td><A HREF="http://my.domain.com"><font
color="red">This is a test</font></A></td></tr></table>
```