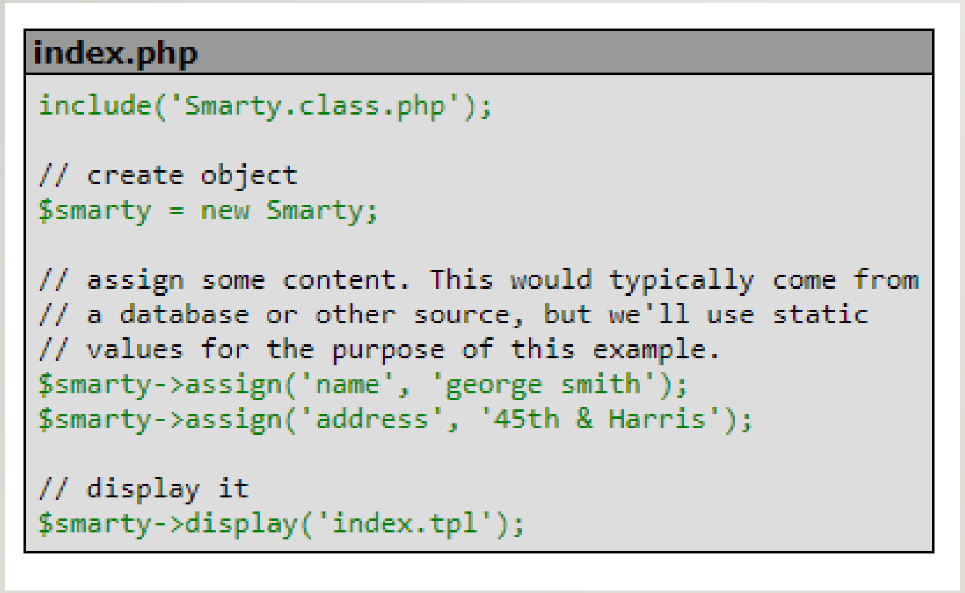
SMARTY TEMPLATES & PROCESSING CAPABILITIES

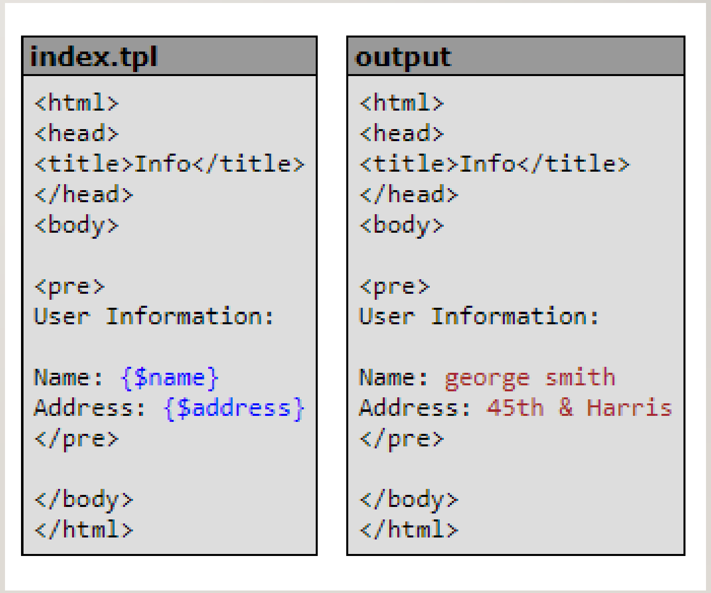
BASIC TEMPLATING IN SMARTY

- Input and Output in Smarty:
  - Assign Content to Smarty as a index.tpl file
  - Presents the output of the template file with the attributes that were given by the user



index.php

```
include('Smarty.class.php');

// create object
$smarty = new Smarty;

// assign some content. This would typically come from
// a database or other source, but we'll use static
// values for the purpose of this example.
$smarty->assign('name', 'george smith');
$smarty->assign('address', '45th & Harris');

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



index.tpl

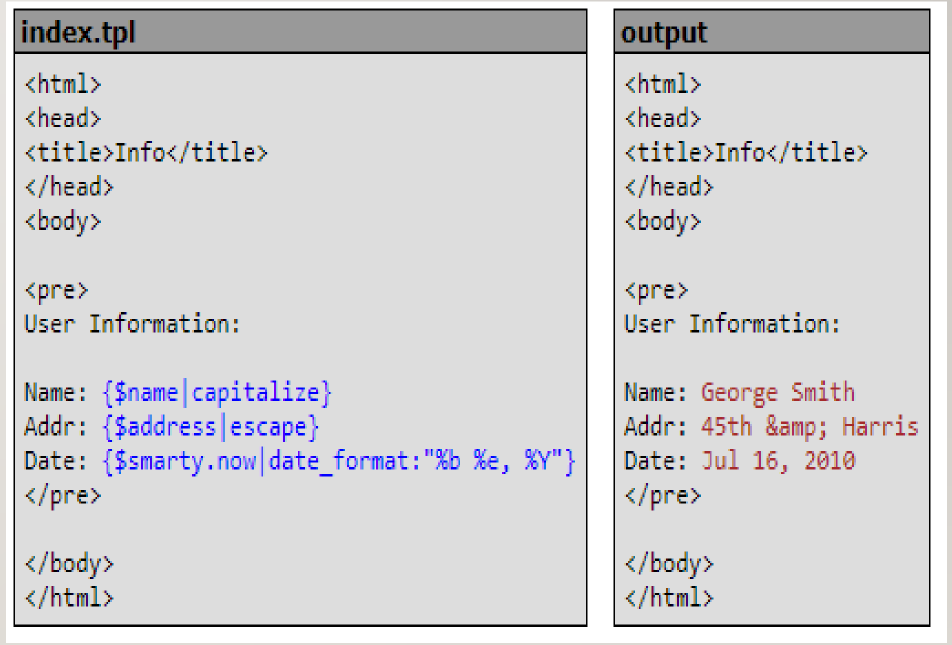
```
<html>
<head>
<title>Info</title>
</head>
<body>

<pre>
User Information:

Name: {$name}
Address: {$address}
</pre>

</body>
</html>
```

output

```
<html>
<head>
<title>Info</title>
</head>
<body>

<pre>
User Information:

Name: george smith
Address: 45th & Harris
</pre>

</body>
</html>
```

ANALYSIS OF THE SMARTY OUTPUT

- Separates the presentation (HTML/CSS) from your application (PHP) code
  - The purpose of Smarty is to help keep the template design (front-end) separated from the application programming role (back-end)
  - Ultimately, this helps us remodel the template (of the content) without altering the application code
    - i.e. If the template defines that the name of the user must be capitalized, then Smarty will capitalize the name that was defined in the PHP application



index.tpl

```
<html>
<head>
<title>Info</title>
</head>
<body>

<pre>
User Information:

Name: {$name|capitalize}
Addr: {$address|escape}
Date: {$smarty.now|date_format:"%b %e, %Y"}
</pre>

</body>
</html>
```

output

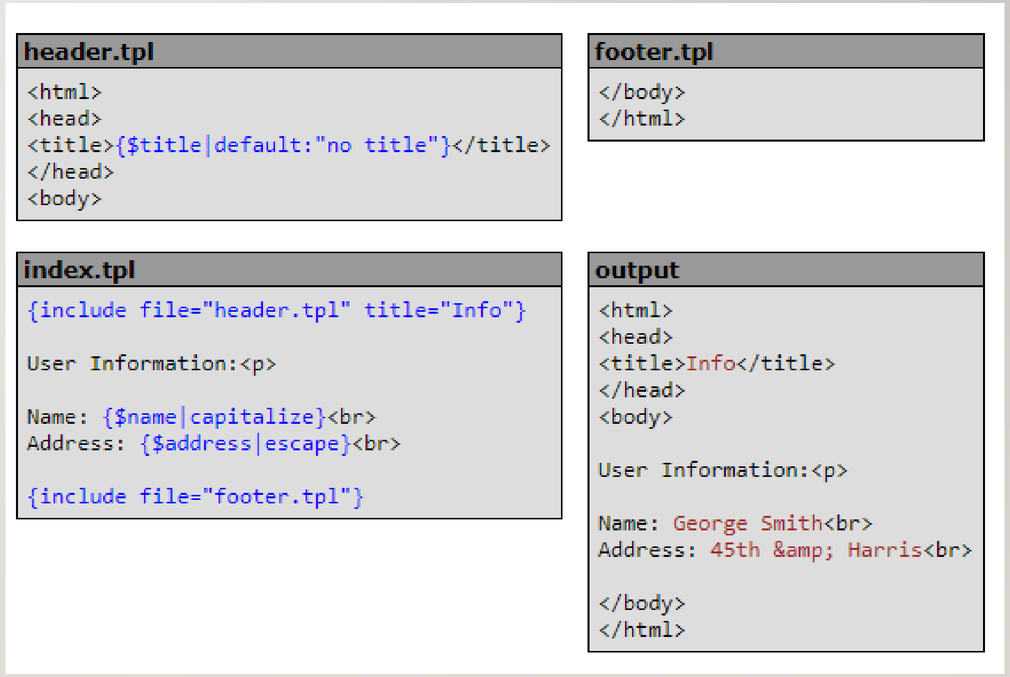
```
<html>
<head>
<title>Info</title>
</head>
<body>

<pre>
User Information:

Name: George Smith
Addr: 45th &amp; Harris
Date: Jul 16, 2010
</pre>

</body>
</html>
```

ANALYSIS OF THE SMARTY OUTPUT

- In the output, we can justify that certain modifiers can be used in a template in order to modify the placeholders that stores the data in a variable (such as name, address, date, and so on)
- This means that the output can be modified in many ways under the discretion of the user in the template
    - On that note, the application programmer can replace an attribute (name, address, data, etc.) to manipulate the data that is displayed in the templates

TEMPLATE FUNCTIONS

- Carry out tasks in the template that is handled by the application programmer
- {include} – Inclusion of other templates created through a header/footer
    - Scope = Locally stored data
        - A template variable will not be assigned directly, but rather be passed as an attribute
    - The variable is available within the scope of the header template, and can be dynamically allocate the data any time when the tpl is included



header.tpl

```
<html>
<head>
<title>{$title|default:"no title"}</title>
</head>
<body>
```

footer.tpl

```
</body>
</html>
```

index.tpl

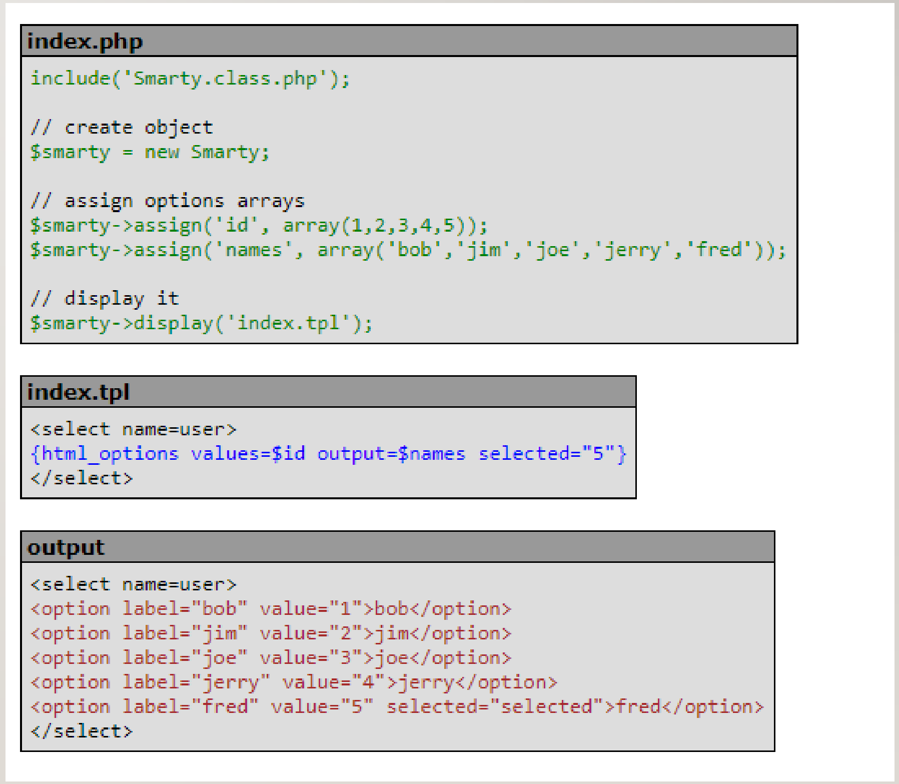
```
{include file="header.tpl" title="Info"}

User Information:<p>

Name: {$name|capitalize}<br>
Address: {$address|escape}<br>

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

output

```
<html>
<head>
<title>Info</title>
</head>
<body>

User Information:<p>

Name: George Smith<br>
Address: 45th &amp; Harris<br>

</body>
</html>
```

TEMPLATE FUNCTIONS

- {html_options} – Helps generate a set of select options
  - A custom function that creates the html <select><option> group with the assigned data
  - Takes care of which attributes are defined as default (as in remaining static throughout the program)



index.php

```
include('Smarty.class.php');

// create object
$smarty = new Smarty;

// assign options arrays
$smarty->assign('id', array(1,2,3,4,5));
$smarty->assign('names', array('bob','jim','joe','jerry','fred'));

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

index.tpl

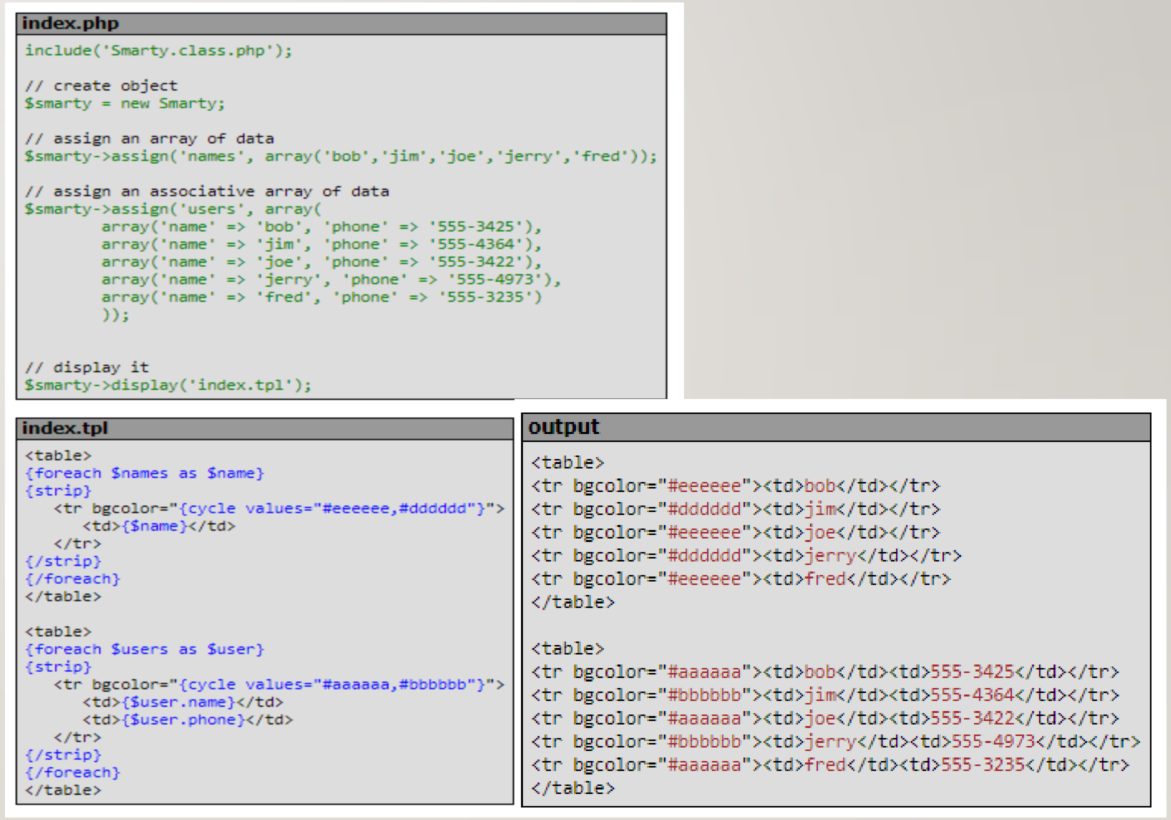
```
<select name=user>
{html_options values=$id output=$names selected="5"}
</select>
```

output

```
<select name=user>
<option label="bob" value="1">bob</option>
<option label="jim" value="2">jim</option>
<option label="joe" value="3">joe</option>
<option label="jerry" value="4">jerry</option>
<option label="fred" value="5" selected="selected">fred</option>
</select>
```

TEMPLATE FUNCTIONS

- {cycle} – Cycle through a set of values
  - Example: Alternate between two or more names in an array
  - Cycling through an array of digits for random generator



index.php

```
include('Smarty.class.php');

// create object
$smarty = new Smarty;

// assign an array of data
$smarty->assign('names', array('bob','jim','joe','jerry','fred'));

// assign an associative array of data
$smarty->assign('users', array(
        array('name' => 'bob', 'phone' => '555-3425'),
        array('name' => 'jim', 'phone' => '555-4364'),
        array('name' => 'joe', 'phone' => '555-3422'),
        array('name' => 'jerry', 'phone' => '555-4973'),
        array('name' => 'fred', 'phone' => '555-3235')
        ));


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

index.tpl

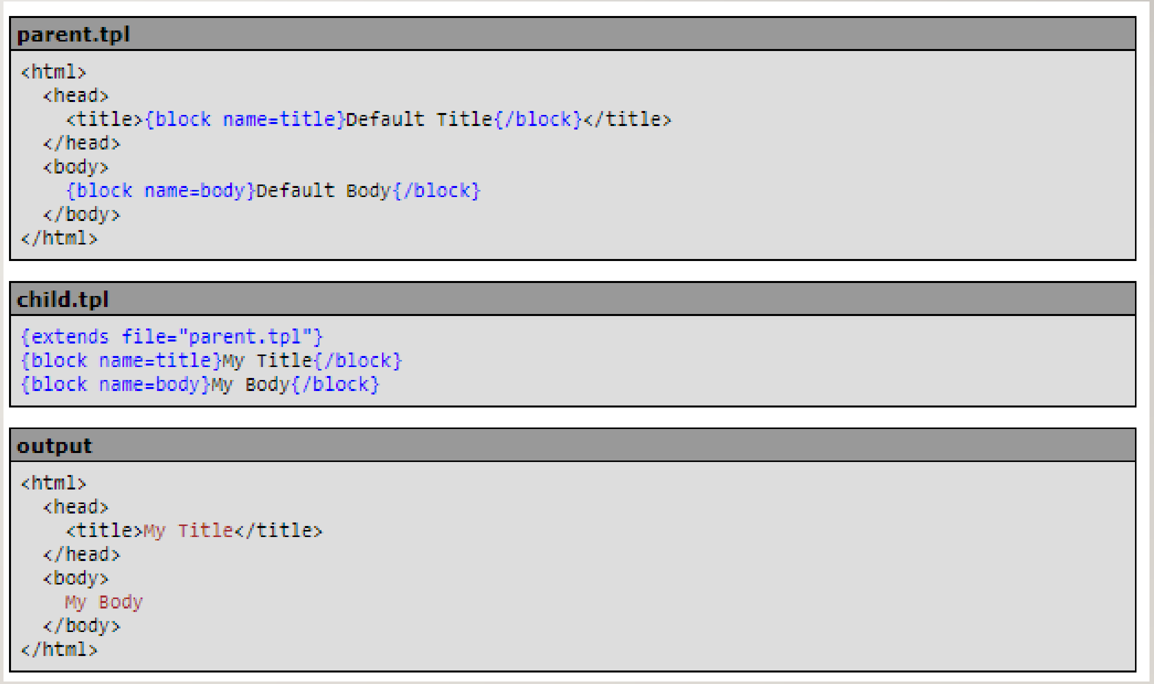
```
<table>
{foreach $names as $name}
{strip}
   <tr bgcolor="{cycle values="#eeeeee,#dddddd"}">
      <td>{$name}</td>
   </tr>
{/strip}
{/foreach}
</table>

<table>
{foreach $users as $user}
{strip}
   <tr bgcolor="{cycle values="#aaaaaa,#bbbbbb"}">
      <td>{$user.name}</td>
      <td>{$user.phone}</td>
   </tr>
{/strip}
{/foreach}
</table>
```

output

```
<table>
<tr bgcolor="#eeeeee"><td>bob</td></tr>
<tr bgcolor="#dddddd"><td>jim</td></tr>
<tr bgcolor="#eeeeee"><td>joe</td></tr>
<tr bgcolor="#dddddd"><td>jerry</td></tr>
<tr bgcolor="#eeeeee"><td>fred</td></tr>
</table>

<table>
<tr bgcolor="#aaaaaa"><td>bob</td><td>555-3425</td></tr>
<tr bgcolor="#bbbbbb"><td>jim</td><td>555-4364</td></tr>
<tr bgcolor="#aaaaaa"><td>joe</td><td>555-3422</td></tr>
<tr bgcolor="#bbbbbb"><td>jerry</td><td>555-4973</td></tr>
<tr bgcolor="#aaaaaa"><td>fred</td><td>555-3235</td></tr>
</table>
```

TEMPLATE INHERITANCE

- Starting with a base, skeleton code (parent template), template inheritance is the method of supporting child templates that can override the placeholders that are provided in a parent template
  - These changes ONLY reflect when {blocks}, which are placeholders in Smarty, are used to create a base template:



parent.tpl

```
<html>
  <head>
    <title>{block name=title}Default Title{/block}</title>
  </head>
  <body>
    {block name=body}Default Body{/block}
  </body>
</html>
```

child.tpl

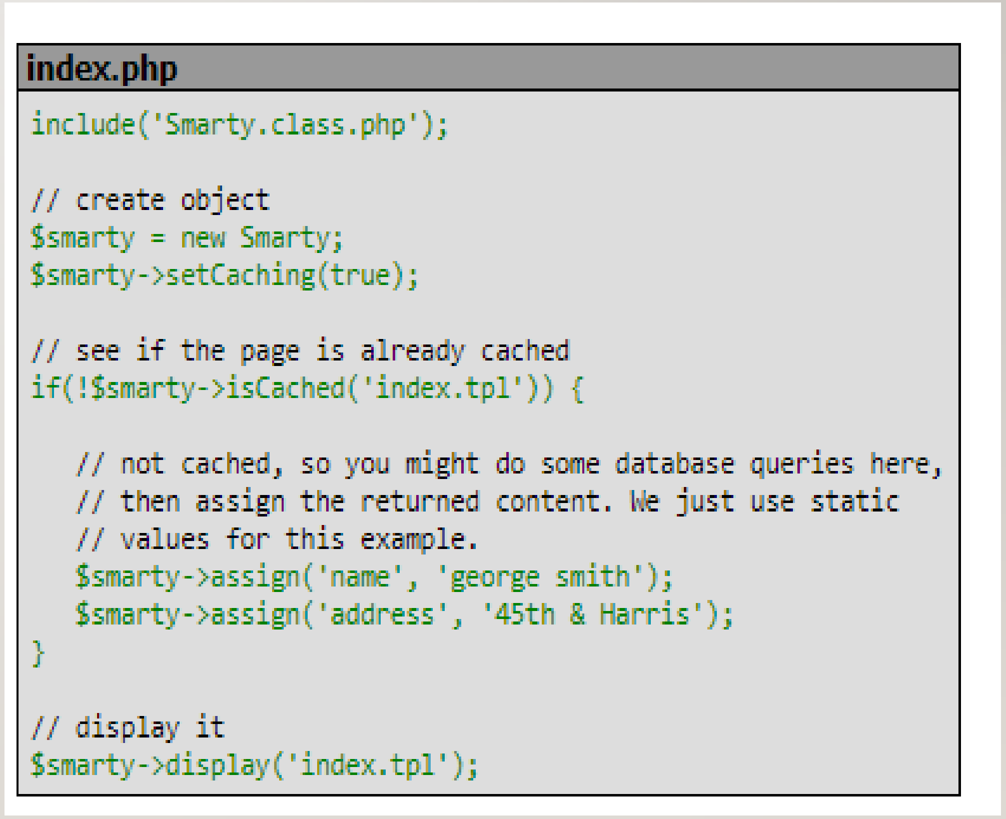
```
{extends file="parent.tpl"}
{block name=title}My Title{/block}
{block name=body}My Body{/block}
```

output

```
<html>
  <head>
    <title>My Title</title>
  </head>
  <body>
    My Body
  </body>
</html>
```

BUILT-IN CACHING CAPABILITIES

- Purpose: To help speed page rendering
- Process:
  - Copy of the template output is stored in a text file
  - The text file is displayed upon subsequent calls to the request instead of dynamically rendering the page each time
- Example: If the page is already cached, then assign the returned content where static values are used to define the name and address attributes

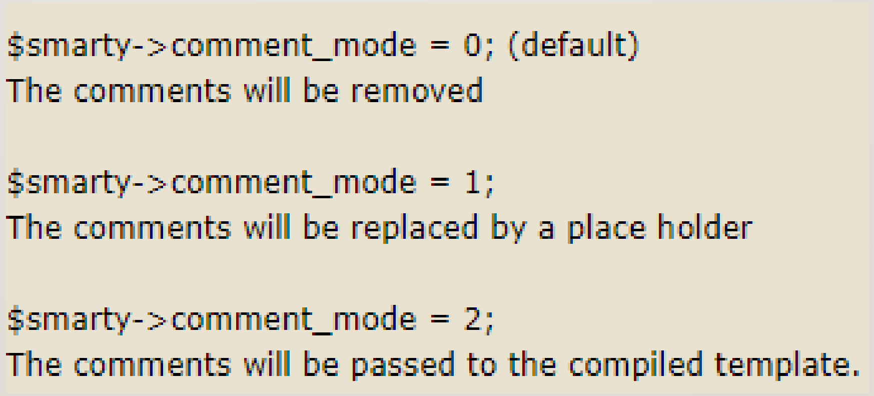


index.php

```
include('Smarty.class.php');

// create object
$smarty = new Smarty;
$smarty->setCaching(true);

// see if the page is already cached
if(!$smarty->isCached('index.tpl')) {

   // not cached, so you might do some database queries here,
   // then assign the returned content. We just use static
   // values for this example.
   $smarty->assign('name', 'george smith');
   $smarty->assign('address', '45th & Harris');
}

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

CONCLUSION

- Overall, I believe that placeholders are changed under the discretion of the template designer (front-end) and application programmer (back-end)
  - I saw a forum post of Smarty, where they mentioned placeholders that appear in comments. Like that idea, it is sufficient to grasp that same logic in terms of php and tpl:
  - Conditions for the logic:



```
$smarty->comment_mode = 0; (default)
The comments will be removed

$smarty->comment_mode = 1;
The comments will be replaced by a place holder

$smarty->comment_mode = 2;
The comments will be passed to the compiled template.
```