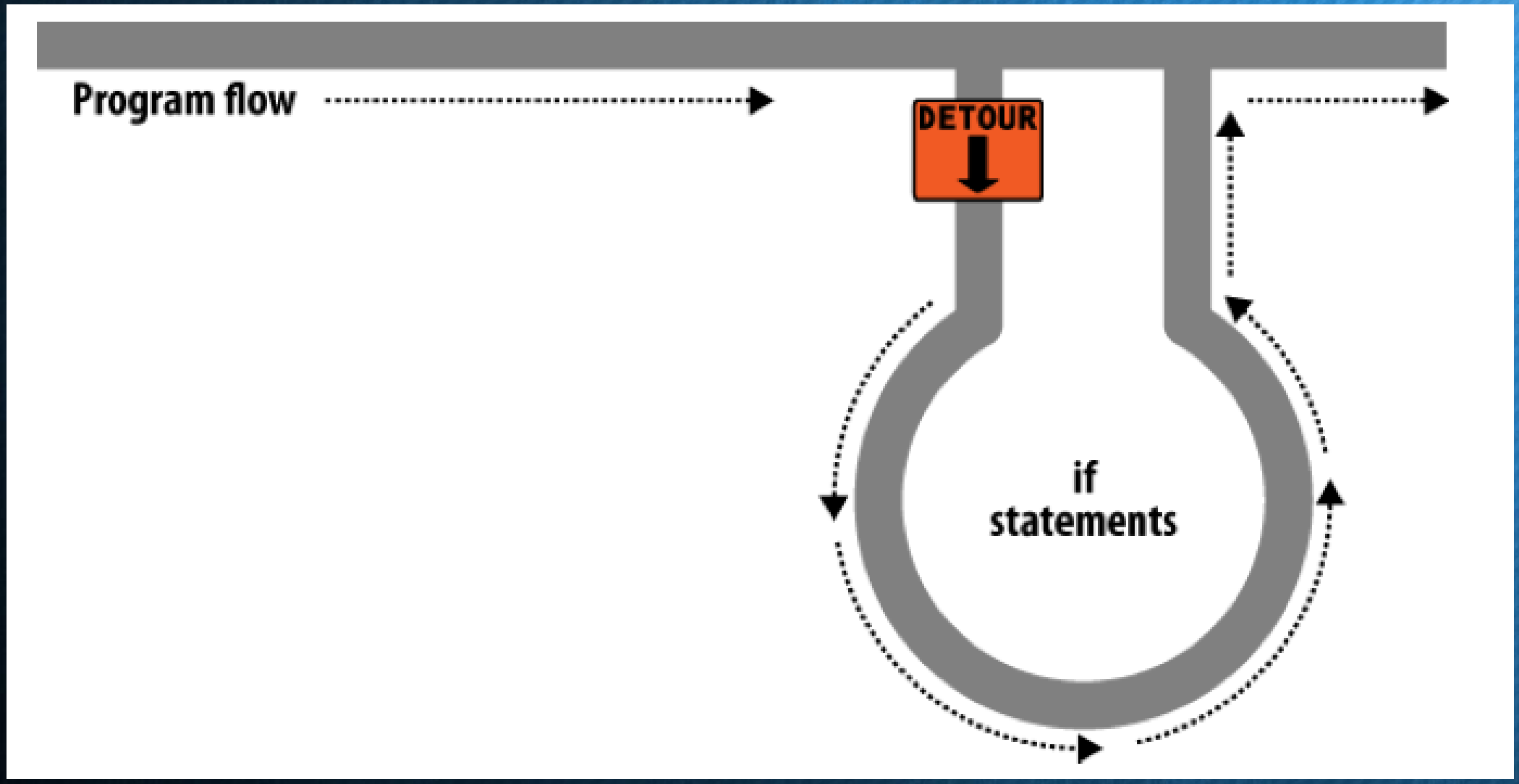
Basic PHP 2

Control Structures



Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
  - if...else statement - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
  - elseif statement - is used with the if...else statement to execute a set of code if one of several condition are true

The If...Else Statement - Syntax

- If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

```
if (condition)
code to be executed if condition is true;
else
code to be executed if condition is
false;
```

The If...Else Statement - Example

- The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":.

```
<html>
<body>
<?php
$d=date("D");
if  ($d=="Fri")
echo   "Have   a  nice  weekend!";
else
echo  "Have   a  nice   day!";
?>
</body>
</html>
```

The If...Else Statement - Example

- If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
<body>
<?php
$d=date("D");
if  ($d=="Fri")
{
echo  "Hello!<br  />";
echo  "Have  a  nice  weekend!";
echo  "See  you  on  Monday!";
}
?>
</body>
</html
```

The Elseif Statement - Syntax

- If you want to execute some code if one of several conditions are true use the elseif statement.

```
if (condition)
code to be executed if condition is true;
elseif (condition)
code to be executed if condition is true;
else
code to be executed if condition is
false;
```

The Elseif Statement - Example

- The following example will output "Have a nice weekend!" if the current day is Friday,
- and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if  ($d=="Fri")
echo  "Have  a  nice  weekend!";
elseif  ($d=="Sun")
echo  "Have  a  nice  Sunday!";
else
echo  "Have  a  nice  day!";
?>
</body>
</html>
```

The Switch Statement - Syntax

- If you want to select one of many blocks of code to be executed, use the Switch statement.
- The switch statement is used to avoid long blocks of if..elseif..else code.

```
switch  (expression)
{
case  label1:
code  to  be  executed  if  expression  =  label1;
break;
case  label2:
code  to  be  executed  if  expression  =  label2;
break;
default:
code  to  be  executed
if  expression  is  different
from  both  label1  and  label2;
}
```

The Switch Statement - Example

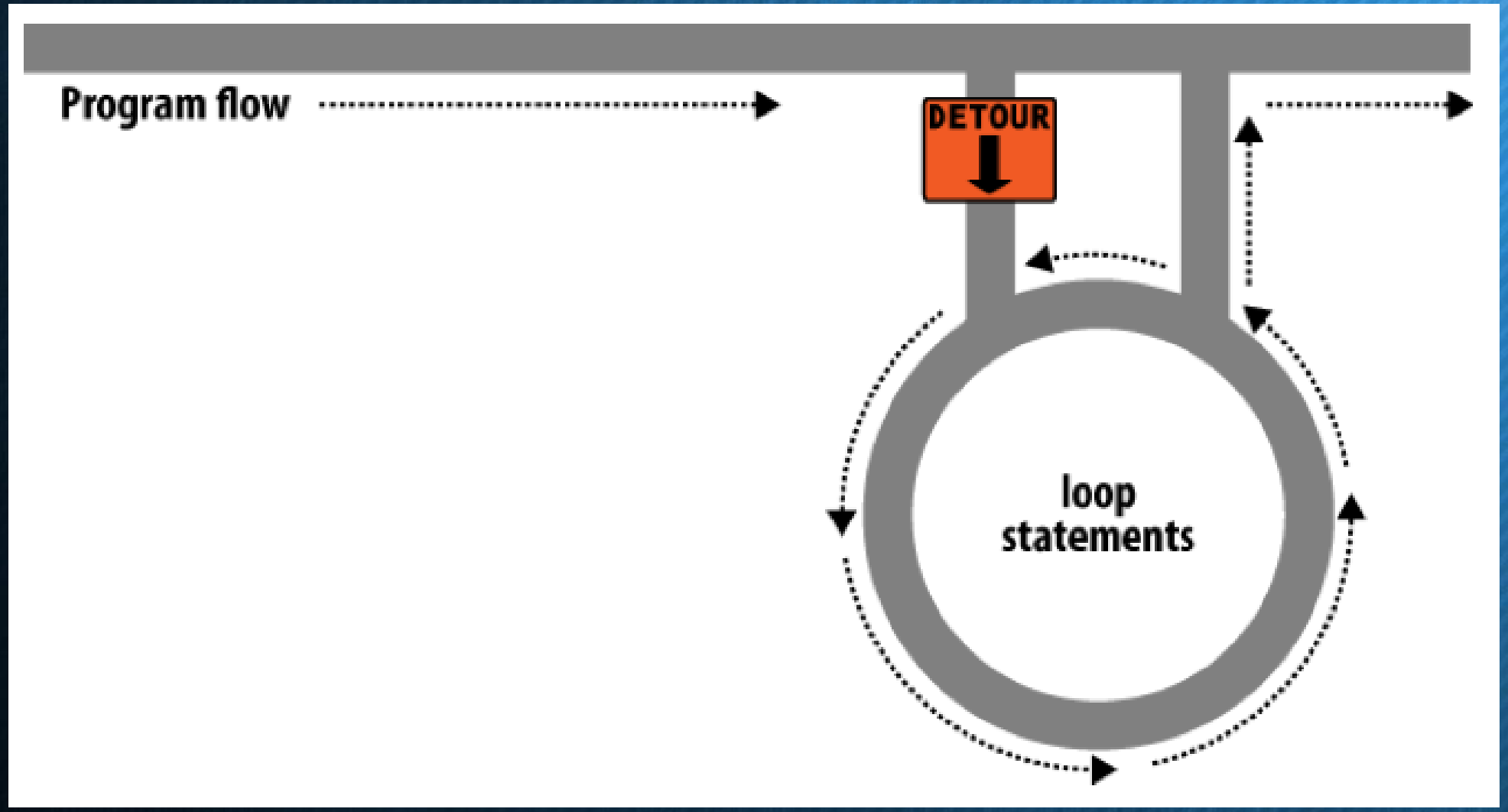
This is how it works:

- A single expression (most often a variable) is evaluated once
- The value of the expression is compared with the values for each case in the structure
- If there is a match, the code associated with that case is executed
- After a code is executed, break is used to stop the code from running into the next case
- The default statement is used if none of the cases are true

The Switch Statement - Example

```
<?php
switch($x)
{
case 1:
echo "Number 1";
break;
case 2:
echo "Number 2";
break;
case 3:
echo "Number 3";
break;
default:
echo  "No number between 1 and 3";
}
?>
```

Looping Structures



PHP Looping

- Looping statements in PHP are used to execute the same block of code a specified number of times.
- Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.
- In PHP we have the following looping statements:
  - while - loops through a block of code if and as long as a specified condition is true
  - do...while - loops through a block of code once, and then repeats the loop as long as a special condition is true
  - for - loops through a block of code a specified number of times
  - foreach - loops through a block of code for each element in an array

The while Statement - Syntax

- The while statement will execute a block of code if and as long as a condition is true.

```
while (condition)
code to be executed;
```

The while Statement - Example

- The following example demonstrates a loop that willcontinue to run as long as the variable i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while($i<=5)
{
Echo "The number is  " . $i . "<br/>";
$i++;
}
?>
</body>
</html>
```

The do…while Statement - Syntax

- The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

```
do
{
code to be executed;
}
while (condition);
```

The do…while Statement - Example

- The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
{
$i++;
echo "The number is " . $i . "<br/>";
}
While ($i<5);
?>
</body>
</html>
```

The for Statement - Syntax

- The for statement is used when you know how many times you want to execute a statement or a list of statements.

```
for   (initialization;  condition;  increment)
{
code  to  be  executed;
}
```

The for Statement - Example

- The following example prints the text "Hello World!" five times:

```
html>
<body>
<?php
for   ($i=1;   $i<=5;  $i++)
{
echo   "Hello  World!<br  />";
}
?>
</body>
</html>
```

The foreach Statement - Syntax

- The foreach statement is used to loop through arrays.
- For every loop, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.

```
foreach (array as value)
{
code to be executed;
}
```

While Loop Looping Through an Array

- This method is commonly used when working with database results in the style of arrays but at the same time, are a completely feasible way to read non-database result-arrays.

```
$ShoppingArray = ["Eggs", "Bacon", "HashBrowns",
"Beans", "Bread", "RedSauce"];
$arrayLength = count($ShoppingArray);
    $i = 0;
    while ($i < $arrayLength)
    {
        echo $ShoppingArray[$i] ."<br />";
        $i++;
    }
```

For Loop Looping Through an Array

- Three parameters are needed
  - An initial counter set to a certain value, usually zero.
  - A Boolean test, usually involving the initial counter.
  - A counter increment eg: counter++.

```
$ShoppingArray = ["Eggs", "Bacon", "HashBrowns",
"Beans", "Bread", "RedSauce"];
for ($i = 0; $i < count($ ShoppingArray); $i++)  {
                echo $ ShoppingArray[$i] ."<br />";
            }
```

For Each Looping Through an Array

- Designed to read array, there is no need for a boolean test, just simply pass in the array and do what you want with it.
- There isn't a mandatory rule to use a numeric index to pick out data values, the foreach loop essentially rids this concept for you.

```
$ShoppingArray = ["Eggs", "Bacon", "HashBrowns",
"Beans", "Bread", "RedSauce"];
foreach ($ShoppingArray as $food)  {
            echo $food ."<br />";
        }       }
```

Array Iterator Looping Through an Array

- It is a more advanced method of looping over arrays, It's part of a wider class that exposes many accessible variables and functions.

```
$ShoppingArray = ["Eggs", "Bacon", "HashBrowns",
"Beans", "Bread", "RedSauce"];
$arrObject = new ArrayObject($ ShoppingArray);
$arrayIterator = $arrObject->getIterator();

while( $arrayIterator->valid() )
{
      echo $arrayIterator->current()  . "<br />";
      $arrayIterator->next();
}
```

Loop Control

- Like many C-inspired languages, PHP has two control structures that work within a loop
  - break - exit the loop immediately
  - continue - finish the current iteration and jump to the next iteration, starting at the top of the loop

PHP Break

- We have already seen the break statement used in an earlier example. It was used to "jump out" of a switch statement.
- The break statement can also be used to jump out of a loop.

```
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
}
?>
```

PHP Continue

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 4:

```
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    continue;
  }
  echo "The number is: $x <br>";
}
?>
```

Conversion / Casting

- As PHP evaluates expressions, at times values in the expression need to be converted from one type to another as the computations are done.
- PHP does aggressive implicit type conversion (casting)
- You can also make type conversion (casting) explicit with casting operators.

Casting

```
$a = 56; $b = 12;
$c = $a / $b;
echo "C: $c\n";
$d = "100" + 36.25 + TRUE;
echo "D: ". $d . "\n";
echo "D2: ". (string) $d . "\n";
$e = (int) 9.9 - 1;
echo "E: $e\n";
$f = "sam" + 25;
echo "F: $f\n";
$g = "sam" . 25;
echo "G: $g\n"
```

In PHP, division forces operands to be floating point. PHP converts expression values silently and aggressively

C: 4.66666666667
D: 137.25
D2: 137.25
E: 8
F: 25
G: sam25

Explicit Casting

Cast type	Description
`(int)` `(integer)`	Cast to an integer by dropping the decimal portion
`(bool)` `(boolean)`	Cast to a Boolean
`(float)` `(double)` `(real)`	Cast to a floating-point number
`(string)`	Cast to a string
`(array)`	Cast to an array
`(object)`	Cast to an object

PHP Functions

- In this lesson we will show you how to create your own functions.

- PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.
- Please check out our PHP reference for a complete overview of the built-in functions, please visit the W3C functions list,

http://w3schools.com/php/default.asp

Create a PHP Function

- A function is a block of code that can be executed whenever we need it.
- Creating PHP functions:
  - All functions start with the word "function()“
  - Name the function - It should be possible to understand what the function does by its name. The name can start with a
  - letter or underscore (not a number)
  - Add a "{" - The function code starts after the opening curly brace
  - Insert the function code
  - Add a "}" - The function is finished by a closing curly brace

Create a PHP Function

- A simple function that writes my name when it is called:

```
<html>
<body>
<?php
function  writeMyName()
{
echo   "Cindy  Liu";
}
writeMyName();
?>
</body>
```

Use a PHP Function

- Now we will use the function in a PHP script:

```
<?php
function  writeMyName()
{
echo   “Cindy  Liu";
}
echo   "Hello  world!<br  />";
echo   "My  name  is  ";
writeMyName();
echo   ".<br  />That's  right,  ";
writeMyName();
echo   "  is   my   name.";
?>
```

Use a PHP Function

- The output of the code will be:

Hello world!

My name is Cindy Liu.

That's right, Cindy Liu is my name.

PHP Functions - Adding parameters

- Our first function writeMyName() is a very simple function. It only writes a static string.

- To add more functionality to a function, we can add parameters. A parameter is just like a variable.

- You may have noticed the parentheses (brackets!) after the function name, like: writeMyName(). The parameters are specified inside the parentheses.

PHP Functions - Adding parameters

- The following example will write different first names, but the same last name:

```
<?php
function writeMyName($fname)
{
echo $fname . " Smith.<br />";
}
echo "My name is ";
writeMyName("John");
echo "My name is ";
writeMyName("Sarah");
echo "My name is ";
writeMyName("Smith");
?>
```

PHP Functions - Adding parameters

- The output of the code will be:

My name is John smith.

My name is Sarah Smith.

My name is Smith Smith.

PHP Functions - Adding parameters

- The following function has two parameters:

```
<?php
function writeMyName($fname,$punctuation)
{
echo $fname . " Smith" . $punctuation .
"<br />";
}
echo "My name is ";
writeMyName("John",".");
echo "My name is ";
writeMyName("Sarah","!");
echo "My name is ";
writeMyName("Smith","...");
?>
```

PHP Functions - Adding parameters

- The output of the code will be:

My name is John smith.

My name is Sarah Smith!

My name is Smith Smith…

PHP Functions - Return values

- Functions can also be used to return values.

```
<html>
<body>
<?php
function  add($x,$y)
{
$total  =  $x  +  $y;
return  $total;
}
echo  "1  +  16  =  "  .  add(1,16);
?>
</body>
</html>
```

PHP Functions - Return values

- The output of the code will be:

```
1  +  16  =  17
```

Variable Scope

- In general, variable names used inside of function code, do not mix with the variables outside of the function. They are walled-off from the rest of the code. This is done because you want to avoid "unexpected" side effects if two programmers use the same variable name in different parts of the code.

- We call this "name spacing" the variables. The function variables are in one "name space" whilst the main variables are in another "name space“

- Like little padded cells of names - like silos to keep things separate.

PHP Global Variable - Superglobals

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

- The PHP superglobal variables are:
  - $GLOBALS
  - $_SERVER
  - $_REQUEST
  - $_POST
  - $_GET
  - $_FILES
  - $_ENV
  - $_COOKIE
  - $_SESSION

PHP $Globals

- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable.

```
<?php
$x = 75;
$y = 25;
function addition() {
        $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z; ?>
```

PHP $_SERVER

- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo   $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

Normal Scope (isolated)

```
function tryzap() {
   $val = 100;
}

$val = 10;
tryzap();
echo "TryZap = $val\n";
```

TryZap = 10

Global Scope (common)

```
function dozap() {
global $val;
$val = 100;
}
$val = 10;
dozap();
echo "DoZap = $val\n";
```

DoZap = 100

Programming in Multiple Files

- When your programs get large enough, you may want to break them into multiple files to allow some common bits to be reused in many different files.

```
<!DOCTYPE html>
<head>
<?php include("header.php"); ?>
</head>
<body>
<?php include("nav.php"); ?>
<div id="main">
.
.
.
```

Including files in PHP

- include "header.php"; - Pull the file in here
- include_once "header.php"; - Pull the file in here unless it has already been pulled in before
- require "header.php"; - Pull in the file here and die if it is missing
- require_once "header.php"; - You can guess what this means...
- These can look like functions - require_once("header.php");

Coping with Missing Bits

- Sometimes depending on the version or configuration of a particular PHP instance, some functions may be missing. We can check that.

```
if ( function_exists("array_combine"))
{
echo "Function exists";
}
else
{
echo "Function does not exist";
}
```