



LECTURER: KRUNAL PATEL
ICCT ENGG. COLLEGE
NEW VALLABH VIDYANAGAR

Why is PHP used?

- PHP : Hypertext pre-processor
- Hypertext pre-processor - Interpreter of PHP code, processes PHP code invoked by web server and send pure HTML code to browser
- Server side scripting language

What Client side scripts can do?

- Scripts can be used to add interesting (Interactive Web Pages) and useful effects to Web pages (Dynamic pages)
- Validations and error checking
- Maintaining state
- Creating Dynamic Forms with Client-Side Scripting.
- Instant Feedback to Users
- Client-Side Scripts Move Processing Tasks back to the Client

Disadvantages of Client Side Scripting

- Browser-Dependent Client-Side Scripts
- Different set of codes for both the browsers
- Secure Source Code of Client-Side Scripts.
- Pages Take Longer to Download
- Program Scope Is Limited to a Single HTML Page
- No Direct Access to System Objects

Which Should I Use? Client- or Server-Side?

- If you want to have dynamic client forms with client-side validation, you must use **client-side** scripting.
- If you want your site to have highly interactive pages, you should use **client-side** scripting.
- If you need to provide your client with advanced functionality that can be created only using ActiveX controls, you must use **client-side** scripting.

Which Should I Use? Client- or Server-Side?

- If you need to track user information across several Web pages to create a "Web application," you must use **server-side** scripting
- If you need to interact with server-side databases, you must use **server-side** scripting.
- If you need to use HTTP server variables or check the capabilities of the user's browser, you must use **server-side** scripting

Which Should I Use? Client- or Server-Side?

Client Side

- Java script
- VB Script

Server Side

- PHP
- JSP
- ASP
- ASP.NET

Why \$php ?

- Today the day has gone when we designed our website with only html. On those days internet is used for only collect information. But now a days we are in web4.0 world of internet.
- Now a days each and every one is on internet. Each one is socializing them selves. More and more companies coming to internet world to sell products or stay with their customers.
- So we can not stay with html to provide a high end site like facebook.com , youtube.com, ebay.com, blogger.com, ibibo.com and etc.

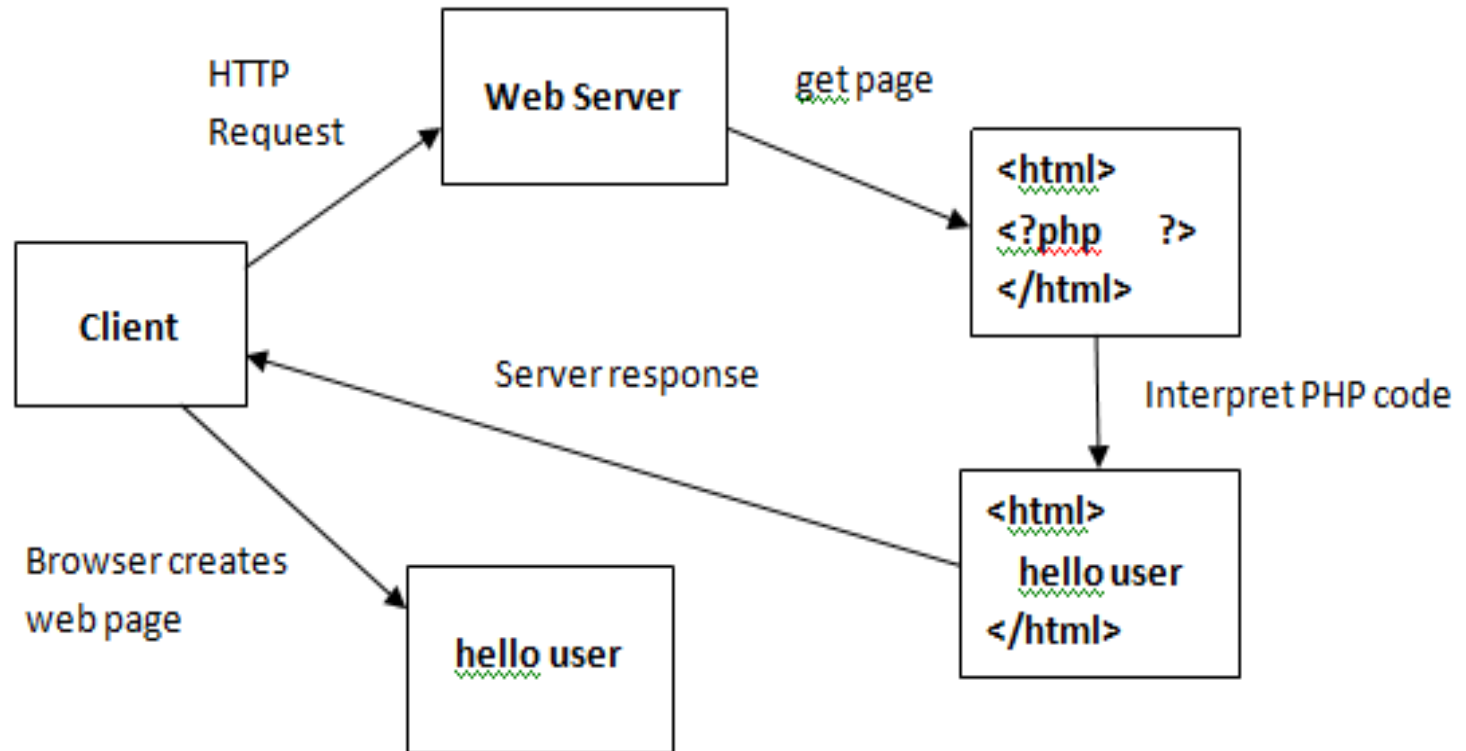
Why is PHP used?

- Free
- Open source
- Platform compatibility
- Simple syntax
- Supports many databases
- Broad functionality for network connectivity, file system support, COM, XML, Macromedia Flash.

What is a PHP File?

- PHP files may contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

How php script works?



HTML in a PHP page

`<script>` block
in `<head>`

```
<head>  
<title>PHP Info</title>  
<?php  
print("PHP Stats on the Server");  
?>  
</head>
```

Language we are
using is PHP

`<script>` block
in `<body>`

```
<body>  
<?php  
phpinfo();  
?>  
And print some more text here!  
</body>  
</html>
```

Generate HTML
“on the fly” with
`print(...);`

Example

```
<html>
```

```
  <head> <title> First Script </title> </head>
```

```
  <body>
```

```
    <?php
```

```
      echo "hello user";
```

```
    ?>
```

```
  </body>
```

```
</html>
```

Example

```
<?php  
    //your php code goes here  
?>
```

OR

```
<?  
//your php code goes here  
?>
```

PHP Comments

```
<html>
```

```
<body>
```

```
<?php
```

```
//This is a single line comment
```

```
/*
```

```
This is a multi-line comment
```

```
*/
```

```
?>
```

```
</body>
```

```
</html>
```

Data Types in PHP

PHP supports all primitive data types

boolean

integer

floating-point number

String

array

Object

- The programmer does not need to specify the type of a variable a variable's type is determined from the context of its usage

Booleans

The boolean data type admits two values

true (case-insensitive)

false (case-insensitive)

Example

```
$itIsRainingToday = true;
```

```
$thePrinterIsBusy = True;
```

```
$theQueueIsEmpty = FALSE;
```

Integers

Integers can be specified in decimal, hexadecimal or octal notation, optionally preceded by a sign

In octal notation, the number must have a leading 0

In hexadecimal notation, the number must have a leading 0x.

Examples

`$a = 1234;`# decimal number

`$a = 0123;`# octal number (i.e., 83 decimal)

`$a = -123;`# a negative number

`$a = 0x1B;`# hexadecimal number (i.e., 27 decimal)

The maximum size of an integer is platform-dependent, but usually it's 32 bits signed – about 2,000,000,000

PHP does not support unsigned integers.

Floating Point Numbers

Specified using any of these forms

$\$a = 1.234;$

$\$a = 1.2e3;$

$\$a = 7E-10;$

The maximum size of a float is platform-dependent, although most support a maximum of about $1.8e308$ with a precision of roughly 14 decimal digits

Strings

Specified in different ways

single quoted

In single-quoted strings, single-quotes and backslashes must be escaped with a preceding backslash

```
echo 'this is a simple string';
```

```
echo 'You can embed newlines in strings,  
just like this.';
```

```
echo 'Douglas MacArthur said "I\'ll be back" when leaving the Phillipines';
```

```
echo 'Are you sure you want to delete C:\\*.?*';
```

double quoted

In double-quoted strings,

variables are interpreted to their values, and

various characters can be escaped

```
\n    linefeed
```

```
\r    carriage return
```

```
\t    horizontal tab
```

```
\\    backslash
```

```
\$    dollar sign
```

```
\"    double quote
```

```
\[0-7]{1,3}  a character in octal notation
```

```
\x[0-9A-Fa-f]{1,2}  a character in hexadecimal notation
```

Variables Names

- In PHP variable starts with a \$
- Followed by letter or underscore
- Can contain letters, numbers, underscores, or dashes
- No spaces
- Case-sensitive

\$student and \$Student are different as it is case sensitive

Assign a value to a variable

The universal form of the assignment statement
PHP uses the single equals sign =

```
$greeting = "Hello World!";
```

```
$balance = 52;
```

```
$isMatch = false;
```

variable *gets* value

greeting gets the value "Hello World!"

balance gets the value 52

isMatch gets the value false

NOTE: The equals sign = is used *differently* in math and programming.

Predefined variables

`$GLOBALS`

`$_SERVER`

`$_GET`

`$_POST`

`$_COOKIE`

`$_FILES`

`$_ENV`

`$_REQUEST`

`$_SESSION`

Variable Scope

- Local – declared Inside a function
- global – declared Outside a function
- static – retains value between calls
- Parameters and variables declared inside a function are available only inside the function
 - They come into existence when the function is called
 - They disappear when the function exits

Variable Scope

- Variables declared outside a function are not available inside the function
- Automatically global
 - `$_POST`, `$_GET`
 - `$_SERVER`, `$_ENV`
- The keyword **global**
 - Makes internal variables available externally
 - Makes external variables available internally
 - `global $fred`

Constants

- A constant is an identifier (name) for a simple value. As the name suggests, that value cannot change during the execution.
- A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

<?php

```
// Valid constant names
define("FOO", "something");
define("FOO2", "something else");
define("FOO_BAR", "something more");
```

```
// Invalid constant names
define("2FOO", "something");
```

```
// This is valid, but should be avoided:
// PHP may one day provide a magical constant
// that will break your script
define("__FOO__", "something");
```

?>

Expressions

- The right-hand side of an assignment statement can be any valid expression
- Expressions are “formulas” saying how to manipulate existing values to compute new values

`$balance = $balance - $transaction;`

`$seconds = 60*$minutes;`

`$message = 'Status code is '.$codeValue;`

Comments

```
<html>
  <head><title>Don't forget your comments</title></head>
  <body>
<!-- Start HTML page here -->
<h2>A few words about comments</h2>
<?php
// Start PHP script here

$string1 = "Welcome";
/*
Another way to add some comments to your PHP code
*/
$string2 = "$string1 stranger!";
?>
...
```

Operators

Addition (+)

Subtraction (-)

Multiplication (*)

Division (/)

Modulus (%)

Negation (-)

String concatenation (.)

Pre- & Post-increment (++)

Pre & Post-decrement (--)

Equality (==)

Identical (===)

Inequality (!= or <>)

Not identical (!==)

Greater than (>)

Less than (<)

Greater than or equal (>=)

Less than or equal (<=)

Logical AND (&&, and)

Logical OR (||, or)

Logical XOR (xor)

Logical negation (!)

Operators (Cont..)

Assignment

Assignment (=)

Plus-equals (+=)

Minus-equals (-=)

Divide-equals (/=)

Multiply-equals (*=)

Modulus-equals (%=)

Miscellaneous

Error suppression (@)

Conditional or Ternary operator (?:)

Control Structures

- if
- else
- elseif
- while
- do-while
- for
- foreach
- break
- continue
- switch

Conditional Statements

In PHP we have the following conditional statements:

if statement - use this statement to execute some code only if a specified condition is true

if...else statement - use this statement to execute some code if a condition is true and another code if the condition is false

if...elseif....else statement - use this statement to select one of several blocks of code to be executed

switch statement - use this statement to select one of many blocks of code to be executed

The if Statement

to execute some code only
if a specified condition is
true

Syntax

if (*condition*)

*code to be executed if
condition is true;*

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

```
<?php
```

```
    $d=date("D");  
    if ($d=="Fri")
```

```
        echo "Have a nice  
weekend!";
```

```
    ?>
```

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

The if...else Statement

Use the if...else statement to execute some code if a condition is true and another code if a condition is false.

Syntax

if (*condition*)

code to be executed if condition is true;

else

code to be executed if condition is false;

Example.....

- `<html>`
`<body>`

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

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

The if...elseif....else Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

Syntax

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;

Example.....

- ```
<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 PHP Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

## Syntax

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

```
<html>
<body>
<?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";
    }
?>
</body>
</html>
```

PHP Loops

- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
- In PHP, we have the following looping statements:
- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified 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

While Loops

```
<?php
/* example 1 */

$i = 1;
while ($i <= 10) {
    echo $i++; /* the printed value would be
                $i before the increment
                (post-increment) */
}

/* example 2 */

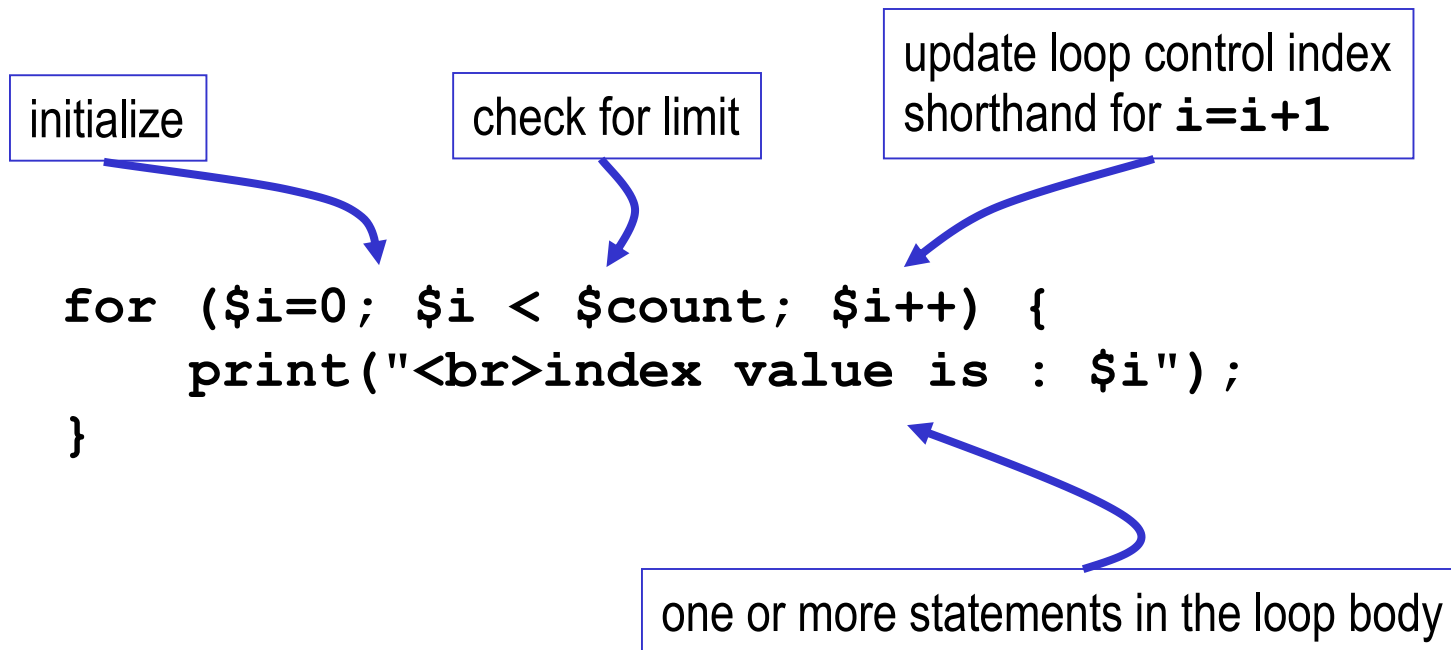
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

do-while

- ***do-while*** loops are very similar to ***while*** loops, except the truth expression is checked at the end of each iteration instead of in the beginning.

```
<?php  
$i = 0;  
do {  
    echo $i++;  
} while ($i > 0);  
?>
```

The for loop



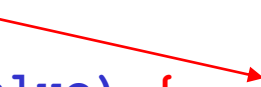
foreach loops

For looping over an array

`foreach (array_expression as $value) statement`

`foreach (array_expression as $key => $value)
statement`

```
<?php  
$arr = array(1, 2, 3, 4);  
foreach ($arr as $key=$value) {  
    echo "$key = $value";  
}
```



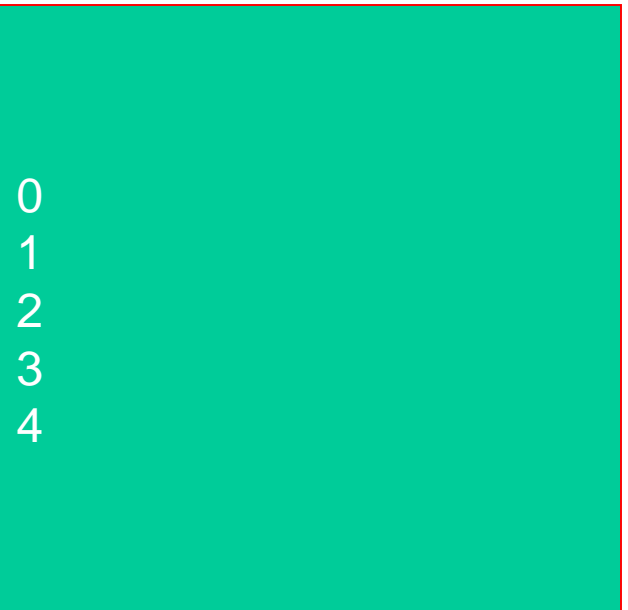
```
0 = 1  
1 = 2  
2 = 3  
3 = 4
```

```
?>
```

break

break ends execution of the current for, foreach, while, do-while or switch structure.

```
<?php
for ($i = 0; $i < 10; ++$i) {
    if ($i == 5)
        break;
    print "$i<br>";
}
?>
```



continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.

```
<?php
for ($i = 0; $i < 6; ++$i) {
    if ($i == 3)
        continue;
    print "$i<br>";
}
?>
```



0
1
2
4
5

switch

```
<?php
    $i=2;
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
    default:
        echo "i is not equal to 0, 1 or 2";
    }
?>
```



i equals 2

PHP functions

What are Functions?

A function structure:

```
<?php
```

```
function myname( $1arg, $2arg,..... $narg)
{
    echo "Example function.\n";
    return $retval;
}
```

```
?>
```


Example For Global

```
<?php
```

```
$a = 1;  
$b = 2;
```

```
function Sum()  
{  
  
    global $a, $b;  
  
    $b = $a + $b;  
}
```

```
Sum();  
echo $b;
```

```
?>
```



b=3

Example For Static

```
<?php
```

```
function Tables()  
{
```

```
    static $count = 0;  
    static $x=0;
```

```
    $y=2;
```

```
    $count++;
```

```
    $x= $x + $y;
```

```
    echo "$x <br>";
```

```
    If ($count < 5)
```

```
    {
```

```
        Tables();
```

```
    }
```

```
}
```

```
Tables();
```

```
?>
```



2
4
6
8
10

Array

- two kinds of arrays in PHP:
 - indexed (The keys of an indexed array are integers, beginning at 0.)
 - associative.(strings as keys and behave more like two-column tables.)

Example

- `// $addresses not defined before this point`
 1. `echo $addresses[0]; // prints nothing`
 2. `echo $addresses; // prints nothing`
 3. `$addresses[0] = 'spam@cyberpromo.net';`
 4. `echo $addresses; // prints "Array"`

Array

- Using simple assignment to initialize an array in your program leads to code like this:
 1. `$addresses[0] = 'spam@cyberpromo.net';`
 2. `$addresses[1] = 'abuse@example.com';`
 3. `$addresses[2] = 'root@example.com';`
- `//` That's an indexed array, with integer indexes beginning at 0.

Array

- `// ... Here's an associative array:`
 1. `$price['Gasket'] = 15.29;`
 2. `$price['Wheel'] = 75.25;`
 3. `$price['Tire'] = 50.00; // ...`
- An easier way to initialize an array is to use the `array()` construct, which builds an array from its arguments:
 - ✓ `$addresses = array('spam@cyberpromo.net', 'abuse@example.com', 'root@example.com');`
- To create an associative array with `array()`, use the `=>` symbol to separate indexes from values:
 - ✓ `$price = array('Gasket' => 15.29, 'Wheel' => 75.25, 'Tire' => 50.00);`

Array

Adding Values to the End of an Array

To insert more values into the end of an existing indexed array, use the `[]` syntax:

1. `$college= array('MBICT', 'BVM');`
2. `$college[] = 'ADIT';`

Getting the Size of an Array

The `count()` and `sizeof()` functions return the number of elements in the array.

1. `$college= array('MBICT', 'BVM');`
2. `$size = count($college);`

Multidimensional Arrays

- The values in an array can themselves be arrays.
 1. `$row_0 = array(1, 2, 3);`
 2. `$row_1 = array(4, 5, 6);`
 3. `$row_2 = array(7, 8, 9);`
 4. `$multi = array($row_0, $row_1, $row_2);`
- You can refer to elements of multidimensional arrays by appending more []s:
 - `$value = $multi[2][0];` // row 2, column 0. `$value = 7`

Extracting Multiple Values

- To copy all of an array's values into variables, use the `list()` construct:
- `list($variable, ...) = $array;`
- example:
 1. `$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');`
 2. `list($n, $a, $w) = $person; // $n is 'Fred', $a is 35, $w is 'Betty'`

What is a function?

- A function is a block of code that can be executed whenever we need it.

There are two distinct aspects to functions:

1. Definition: Before using a function, that function must be defined – i.e. what inputs does it need, and what does it do with them?
2. Calling: When you call a function, you actually execute the code in the function.

Function Definition

- A function accepts any number of input arguments, and returns a SINGLE value.

```
function myfunction($arg1,$arg2,...,$argN)
{
    statements;
    return $return_value;
}
```

A typical function has three components to it

- Starts with the word function
- Has a function name followed by parentheses (). A function name can only start with a letter or a underscore.
- The code block in the function always goes inside curly brackets.

Example

Function to join first and last names together with a space..

```
function make_name($first,$last)
{
    $fullname = $first.' '. $last;
return $fullname;
}
```

Calling functions..

Can be done anywhere..

```
myfunction($arg1,$arg2,...,$argN)
```

or

```
$answer = myfunction($arg1,$arg2,...,$argN)
```

e.g.

```
echo make_name( 'Rob' , 'Tuley' ) ;  
// echoes 'Rob Tuley'
```

Example:

```
<html>
```

```
<body>
```

```
    <?php
```

```
        function add($x,$y) {
```

```
            $total = $x + $y;
```

```
            return $total;
```

```
        }
```

```
        echo "1 + 16 = " . add(1,16);
```

```
    ?>
```

```
</body>
```

```
</html>
```

‘Scope’

- A function executes within its own little protected bubble, or local scope.
- What does this mean? It means that the function can't 'see' any of the variables you have defined apart from those passed in as arguments..
- Each new function call starts a clean slate in terms of internal function variables.

In other words..

- Variables within a function
 - Are local to that function
 - Disappear when function execution ends
- Variables outside a function
 - Are not available within the function
 - Unless set as global
- Remembering variables
 - Not stored between function calls
 - Unless set as static

Global variables..

To access a variable outside the 'local' scope of a function, declare it as a global:

```
function add5toa()
```

```
{
```

```
    global $a;
```

```
    $a = $a + 5;
```

```
}
```

```
$a = 9;
```

```
add5toa();
```

```
echo $a; // 14
```


Static Variables

Local function variable values are not saved between function calls unless they are declared as static:

```
function counter()  
{  
    static $num = 0;  
    return ++$num;  
}  
  
echo counter() ; // 1  
echo counter() ; // 2  
echo counter() ; // 3
```

Default Arguments

Can specify a default value in the function definition which is used only if no value is passed to the function when called..

Defaults must be specified last in the list

function

```
myfunction ($arg1, $arg2= 'blah' ) ...
```

function

```
myfunction ($arg1= 'blah' , $arg2) ...
```

Function argument pass by reference

- By default, function arguments are passed by value (so that if the value of the argument within the function is changed, it does not get changed outside of the function). To allow a function to modify its arguments, they must be passed by reference.
- To have an argument to a function always passed by reference, prepend an ampersand (&) to the argument name in the function definition:

Example : Passing function parameters by reference

```
<?php
function add_some_extra(&$string)
{
    $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str; // outputs 'This is a string, and something extra.'
?>
```

Returning values

- Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called.

Note: If the [return\(\)](#) is omitted the value **NULL** will be returned.

Example #1 Use of [return\(\)](#)

```
<?php
function square($num)
{
    return $num * $num;
}
echo square(4); // outputs '16'.
?>
```

- A function can not return multiple values, but similar results can be obtained by returning an array.

Example #2 Returning an array to get multiple values

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

- To return a reference from a function, use the reference operator & in both the function declaration and when assigning the returned value to a variable:

Example : Returning a reference from a function

```
<?php
function &returns_reference()
{
    return $someref;
}

$newref =& returns_reference();
?>
```

Variable functions

- PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

```
<?php
```

```
function foo() {  
    echo "In foo()<br />\n";  
}
```

```
function bar($arg = "")  
{  
    echo "In bar(); argument  
was '$arg'.<br />\n";  
}
```

```
function echoit($string)  
{  
    echo $string;  
}
```

```
$func = 'foo';  
$func();    // This calls foo()
```

```
$func = 'bar';  
$func('test'); // This calls bar()
```

```
$func = 'echoit';  
$func('test'); // This calls echo  
it()  
?>
```


Forms

- Forms allow us to create interactive websites, by accepting input from users
- Types of input we can collect via forms:
 - Text via text boxes and text areas
 - Selections via radio buttons, check boxes, pulldown menus, and select boxes
- Form actions do all of the work
 - Usually through button clicks

Form Tag

Forms can be classified into three section

- Form header
- Input fields
- Action Buttons

`<form>.....</form>`

Attributes of Form Tag

Action= "URL"

- The action specifies the url to which the Form content is to be sent

Method = "Get" or "Post"

Post – the form content is sent to the server as a message

Get – the content is appended to the URL in a manner identical to query data from an ISINDEX

target="*target*"

Tells where to open the page sent as a result of the request

target= [_blank](#) means open in a new window

target= [_top](#) means use the same window

Input Fields

- Input elements are form elements that can accept input from the user

That elements are

- Text
- Button
- Checkbox
- Image
- Password
- Radio
- Reset
- Submit

Input fields

Syntax

```
<input type = " "      // specifies type of input
      name = " "       // Control name to identify
      value = " "      // Default value to control
      align = " "      // for aligning controls
      size = " "       // horizontal size of textbox
      maxlength = " "  // Max. no. of character
```

>

Text input

A text field:

```
<input type="text" name="textfield" value="with an initial value">
```

A text field:

A multi-line text field

```
<textarea name="textarea" cols="24" rows="2">Hello</textarea>
```

A multi-line text field:

A password field:

```
<input type="password" name="textfield3" value="secret">
```

A password field:

Buttons

A submit button:

```
<input type="submit" name="Submit" value="Submit">
```

A reset button:

```
<input type="reset" name="Submit2" value="Reset">
```

A plain button:

```
<input type="button" name="Submit3" value="Push Me">
```

A submit button: 

submit: send data

A reset button: 

reset: restore all form elements to their initial state

A plain button: 

button: take some action as specified by JavaScript

input

Checkboxes

A checkbox:

```
<input type="checkbox" name="checkbox"
      value="checkbox" checked>
```

A checkbox: ☒

type: "checkbox"

name: used to reference this form element from JavaScript

value: value to be returned when element is checked

Note that there is *no text* associated with the checkbox—you have to supply text in the surrounding HTML

Radio buttons

Radio buttons:


```
<input type="radio" name="radiobutton" value="myValue1">  
male<br>
```

```
<input type="radio" name="radiobutton" value="myValue2" checked>  
female
```

Radio buttons:

☐ male
☒ female

If two or more radio buttons have the same **name**, the user can only select one of them at a time

This is how you make a radio button “group”

If you ask for the value of that **name**, you will get the **value** specified for the selected radio button

As with checkboxes, radio buttons do not contain any text

Drop-down menu or list

A menu or list:

```
<select name="select">  
  <option value="red">red</option>  
  <option value="green">green</option>  
  <option value="BLUE">blue</option>  
</select>
```

A menu or list: 

Additional arguments:

size: the number of items visible in the list (default is "1")

multiple: if set to "true", any number of items may be selected (default is "false")

A complete example

```
<html>
<head>
<title>Get Identity</title>
<meta http-equiv="Content-Type" content="text/html;
      charset=iso-8859-1">
</head>
<body>
<p><b>Who are you?</b></p>
<form method="post" action="">
  <p>Name:
    <input type="text" name="textfield">
  </p>
  <p>Gender:
    <input type="radio" name="gender" value="m">Male
    <input type="radio" name="gender" value="f">Female
  </p>
</form>
</body>
</html>
```

Who are you?

Name:

Gender: ☐ Male ☐ Female

Superglobals

- Superglobals are built-in variables that are always available in all scopes

`$GLOBALS`

`$_SERVER`

`$_GET`

`$_POST`

`$_FILES`

`$_COOKIE`

`$_SESSION`

`$_REQUEST`

`$_ENV`

\$GLOBALS

`$GLOBALS` — References all variables available in global scope

- An associative array containing references to all variables which are currently defined in the global scope of the script.
- The variable names are the keys of the array.

```
<?php
function test() {
    $foo = "local variable";

    echo '$foo in global scope: ' . $GLOBALS["foo"] . "\n";
    echo '$foo in current scope: ' . $foo . "\n";
}

$foo = "Example content";
test();
?>
```

The above example will output something similar to:

```
$foo in global scope: Example content
$foo in current scope: local variable
```

The \$_GET

- ✓ The built-in **\$_GET** is used to collect values from a form sent with method="get".
- ✓ When using method="get" in HTML forms, all variable names and values are displayed in the URL.
- ✓ Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

Note: should not be used when sending passwords or other sensitive information!

- ✓ However, because the variables are displayed in the URL.

Note: The get is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

```
<html >
  <head>
<title>Untitled Document</title>
</head>
<body>
<form name="form1" method="get"
action="checklogin.php" >
```

```
Username: <input name="myusername" type="text"
id="myusername"> <br>
```

```
Password<input name="mypassword" type="password"
id="mypassword" > <br>
```

```
<input type="submit" name="Submit" value="Login">
```

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


checklogin.php

```
<html >
<head>
    <title> get Data</title>
</head>
<body>
    username is "
    <?php echo $_GET["myusername"]; ?>"
    <br />
    password is "
    <?php echo $_GET["mypassword"]; ?> "
</body>
</html>
```

Output:

username is "MBICT".
password is "PHP "

The \$_POST

- ✓ The built-in `$_POST` function is used to collect values from a form sent with `method="post"`.
- ✓ Information sent from a form with the POST method `is invisible to others and has no limits on the amount of information to send`.
- ✓ **Note:** However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the `post_max_size` in the `php.ini` file).

The PHP \$_REQUEST

- ✓ The PHP built-in \$_REQUEST function contains the contents of both \$_GET, \$_POST, and \$_COOKIE.
- ✓ The \$_REQUEST function can be used to collect form data sent with both the GET and POST methods.

Example

- ✓ Welcome <?php echo \$_REQUEST["fname"]; ?>!

You are <?php echo \$_REQUEST["age"]; ?> years old.

```
<form action="checkbox-form.php" method="post"> Which  
buildings do you want access to?<br />  
<input type="checkbox" name="formDoor[]" value="A" />Acorn  
Building<br />  
  <input type="checkbox" name="formDoor[]" value="B"  
/>Brown Hall<br />  
  <input type="checkbox" name="formDoor[]" value="C"  
/>Carnegie Complex<br />  
  <input type="checkbox" name="formDoor[]" value="D"  
/>Drake Commons<br />  
  <input type="checkbox" name="formDoor[]" value="E" />Elliot  
House  
<input type="submit" name="formSubmit" value="Submit" />  
</form>
```

```
<?php
    $aDoor = $_POST['formDoor'];
    if(empty($aDoor))
    {
        echo("You didn't select any buildings.");
    }
    else {
        $N = count($aDoor);
        echo("You selected $N door(s): ");
        for($i=0; $i < $N; $i++) {
            echo($aDoor[$i] . " ");
        }
    }
}
?>
```

```
<form name="form" method="post"
action="listdemo.php">
    <h4>Combo and List Box Form</h4>
    <p>Combo Box<br>
```

```
        <select name="select">
            <option value="Option 1"
selected="true">Option1
        </option>
            <option value="Option
2">Option 2</option>
            <option value="Option
3">Option 3</option>
            <option value="Option
4">Option 4</option>
        </select>
```

<p>List Box - Single Select


```
        <select name="listbox"
size="3">
            <option value="Option 1"
selected="true">Option
1</option>
            <option value="Option
2">Option 2</option>
            <option value="Option
3">Option 3</option>
            <option value="Option
4">Option 4</option>
            <option value="Option
5">Option 5</option>
        </select>
```

List Box - Multiple

Select


```
<option value="Option  
5">Option 5</option>  
</select>
```

```
<select name="listmultiple[]"  
size="3" multiple="true">
```

```
  <option value="Option 1"  
selected="true">Option  
1</option>
```

```
  <option value="Option  
2">Option 2</option>
```

```
  <option value="Option  
3">Option 3</option>
```

```
  <option value="Option  
4">Option 4</option>
```

```
  <input type="submit"  
name="Submit"  
value="Submit">
```

```
  <input type="reset"  
name="Reset" value="Reset">
```

```
</form>
```

```
<body>
```

```
<h3>List Box Form Data</h3>
```

```
<p>Form data passed from the form</p>
```

```
<?php
```

```
    echo "<p>select: " . $_POST['select']."</p>\n";
```

```
    echo "<p>listbox: " . $_POST['listbox'] . "</p>\n";
```

```
    $values = $_POST['listmultiple'];
```

```
    echo "<p>listmultiple: ";
```

```
    foreach ($values as $a){
```

```
        echo $a;
```

```
    }
```

```
    echo "</p>\n";
```

```
?>
```

```
</body>
```


Date function:

```
<?php
```

```
    echo date("Y-m-d");  
    echo date("Y/m/d");  
    echo date("M d, Y");  
    echo date("F d, Y");  
    echo date("D M d, Y");  
    echo date("l F d, Y");  
    echo date("l F d, Y,  
    h:i:s");  
    echo date("l F d, Y, h:i  
    A");  
?>
```

Output

```
2011-02-18  
2011/02/18  
Feb 18, 2011  
February 18, 2011  
Fri Feb 18, 2011  
Friday February 18, 2011  
Friday February 18, 2011, 12:49:17  
Friday February 18, 2011, 12:49 AM
```

```
<html>
```

```
<head>
```

```
<title>PHP Script examples!</title>
```

```
</head>
```

```
<body>
```

```
<font face="Arial" color="#FF00FF">
```

```
<strong>
```

```
<? print(Date("m/j/y")); ?>
```

```
</strong>
```

```
</font>
```

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

Include function

include - includes and evaluates a specific file; failure results in a Warning

Menu.php

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

```
<a  
href="http://www.example.com/index.php">Home</a>
```

```
<a  
href="http://www.example.com/about.php">About Us</a> -
```

```
<a  
href="http://www.example.com/links.php">Links</a> -
```

```
<a  
href="http://www.example.com/contact.php">Contact Us</a>
```

```
<br />
```

Include.php

```
<?php include("menu.php"); ?>
```

```
<p>This is my home page that uses a  
common menu to save me time when I  
add new pages to my website!</p>
```

```
</body>
```

```
</html>
```

Home - About us – Links – Contact
US

This is my home page that uses a
common menu to save me time when

Control Structures – require and include

require – includes and evaluates a specific file; failure results in a Fatal Error

```
<?php  
    require 'header.php';  
?>
```

```
<?php  
    include 'header.php';  
?>
```

Control Structures –require_once and include_once

`require_once` – same as `require` except if the file has already been included, it will not be included again

```
<?php
    require_once 'header.php';
?>
```

`include_once` - same as `include` except if the file has already been included, it will not be included again

```
<?php
    include_once 'header.php';
?>
```

Use when the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.

Exercise:

Change background color based on day of the week
using array

Random text link advertising using predefined arrays

```
<html>
<head>
<title>Background Colors change based on the day of the week</title>
</head>
<?
$today = date("w");
$bgcolor = array( "#FEF0C5", "#FFFFFF", "#FBFFC4", "#FFE0DD", "#E6EDFF",
"#E9FFE6", "#F0F4F1");
?>
<body bgcolor="<?print("$bgcolor[$today]");?>">
<br>This just changes the color of the screen based on the day of the week
</body>
</html>
```

Opening a file

Opening a File

- The fopen() function is used to open files in PHP.
- The first parameter of this function contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened:

```
<html>
```

```
<body>
```

```
<?php
```

```
$file=fopen("welcome.txt","r");
```

```
?>
```

```
</body>
```


Modes	Description
r	Read only. Starts at the beginning of the file
r+	Read/Write. Starts at the beginning of the file
w	Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist
w+	Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist
a	Append. Opens and writes to the end of the file or creates a new file if it doesn't exist
a+	Read/Append. Preserves file content by writing to the end of the file
x	Write only. Creates a new file. Returns FALSE and an error if file already exists
x+	Read/Write. Creates a new file. Returns FALSE and an error if file already exists

Closing a File

The `fclose()` function is used to close an open file:

```
<?php
```

```
$file = fopen("test.txt","r");
```

```
//some code to be executed
```

```
fclose($file);
```

```
?>
```

Check End-of-file

The `feof()` function checks if the "end-of-file" (EOF) has been reached.

The `feof()` function is useful for looping through data of unknown length.

Note: You cannot read from files opened in `w`, `a`, and `x` mode!

```
if (feof($file)) echo "End of file";
```

Reading a File Line by Line

The `fgets()` function is used to read a single line from a file.

Note: After a call to this function the file pointer has moved to the next line.

Example

The example below reads a file line by line, until the end of file is reached:

```
<?php
$file = fopen("welcome.txt", "r") or exit("Unable to open
file!");
//Output a line of the file until the end is reached
while(!feof($file))
{
    echo fgets($file). "<br />";
}
fclose($file);
?>
```

Reading a File Character by Character

The `fgetc()` function is used to read a single character from a file.

Note: After a call to this function the file pointer moves to the next character.

Example

The example below reads a file character by character, until the end of file is reached:

```
<?php
$file=fopen("welcome.txt","r") ;
while (!feof($file))
{
    echo fgetc($file);
}
fclose($file);
?>
```

Create an Upload-File Form

To allow users to upload files from a form can be very useful. Look at the following HTML form for uploading files:

```
<html>
<body>

<form action="upload_file.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" />
<br />
<input type="submit" name="submit" value="Submit" />
</form>

</body>
</html>
```

- The ***enctype*** attribute of the <form> tag specifies which content-type to use when submitting the form.
- “***multipart/form-data***” is used when a form requires binary data, like the contents of a file, to be uploaded
- The ***type="file"*** attribute of the <input> tag specifies that the input should be processed as a file.

For example, when viewed in a browser, there will be a browse-button next to the input field

Create The Upload Script

The "upload_file.php" file contains the code for uploading a file:

```
<?php
if ($_FILES["file"]["error"] > 0)
{
    echo "Error: " . $_FILES["file"]["error"] . "<br />";
}
else
{
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
}
?>
```


`$_FILES["file"]["name"]` - the name of the uploaded file

`$_FILES["file"]["type"]` - the type of the uploaded file

`$_FILES["file"]["size"]` - the size in bytes of the uploaded file

`$_FILES["file"]["tmp_name"]` - the name of the temporary copy of the file stored on the server

`$_FILES["file"]["error"]` - the error code resulting from the file upload

```
<?php
if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
{
    if ($_FILES["file"]["error"] > 0)
    {
        echo "Error: " . $_FILES["file"]["error"] . "<br />";
    }
    else
    {
        echo "Upload: " . $_FILES["file"]["name"] . "<br />";
        echo "Type: " . $_FILES["file"]["type"] . "<br />";
        echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
        echo "Stored in: " . $_FILES["file"]["tmp_name"];
    }
}
else
{
    echo "Invalid file";
}
?>
```

```

<?php
if ((($_FILES["file"]["type"] ==
"image/gif")
|| ($_FILES["file"]["type"] ==
"image/jpeg")
|| ($_FILES["file"]["type"] ==
"image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
{
    if ($_FILES["file"]["error"] > 0)
    {
        echo "Return Code: " .
$_FILES["file"]["error"] . "<br />";
    }
    else
    {
        echo "Upload: " .
$_FILES["file"]["name"] . "<br />";
        echo "Type: " . $_FILES["file"]["type"]
. "<br />";
        echo "Size: " . ($_FILES["file"]["size"] /
1024) . " Kb<br />";
        echo "Temp file: " .
$_FILES["file"]["tmp_name"] . "<br />";

```

```

if (file_exists("upload/" .
$_FILES["file"]["name"]))
{
    echo $_FILES["file"]["name"] . "
already exists. ";
}
else
{
    move_uploaded_file($_FILES["fil
e"]["tmp_name"],
"upload/" .
$_FILES["file"]["name"]);
    echo "Stored in: " . "upload/" .
$_FILES["file"]["name"];
}
}
else
{
    echo "Invalid file";
}
?>

```