

SWE 363: Web Engineering & Development

Module 7-1

Server-Side Programming. (Intro to **PHP**)



Objectives

- ❑ Learn how to configure and prepare to implement scripts for the server-side
- ❑ Learn how to create a simple PHP web page
- ❑ Learn the Basics of PHP

- ☐ Client-server model of the Web
- ☐ Server-Side Scripts
- ☐ Server-side technologies
- ☐ Why PHP?
- ☐ Applications of PHP Scripts
- ☐ How to start & install PHP
- ☐ PHP Basics
- ☐ Conditional & Looping statements
- ☐ PHP Functions
- ☐ Strings
- ☐ Arrays
- ☐ Including files

Introduction

- ❑ Server-side development is much more than web hosting: it involves the use of a programming technology (like PHP) to create scripts that dynamically generate content
- ❑ When developing server-side scripts, you are writing software, like C or Java, however with the major distinction
 - (1) your software runs on a web server and
 - (2) uses the HTTP request-response loop for most interactions with the clients
- ❑ >> This distinction is significant, since it invalidates many classic software development patterns, and requires different thinking for many seemingly simple software principles like data storage and memory management.

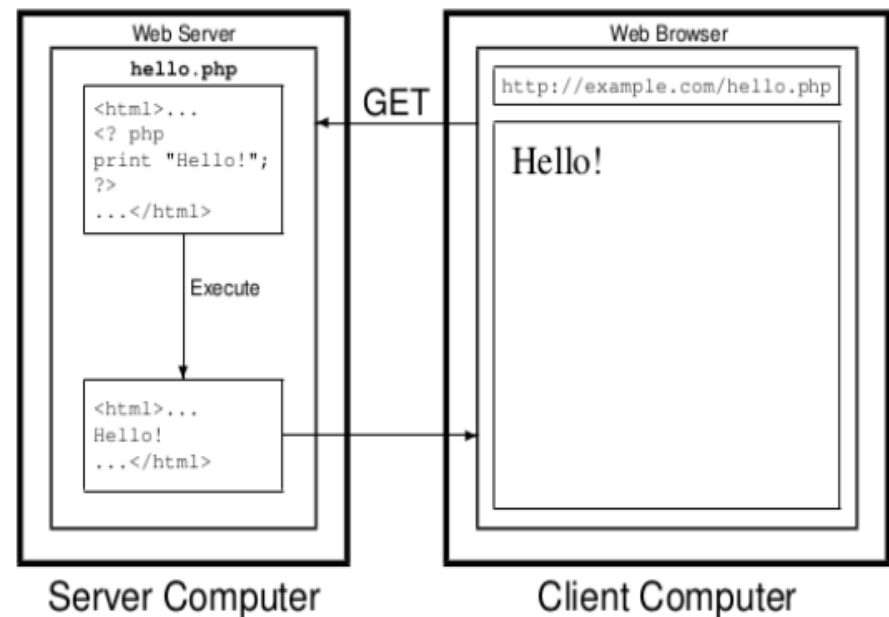
Client-server model of the Web

□ Request ↔ Response

□ A **web server** responds to client requests (typically from a web browser) by providing resources such as HTML documents. e.g.

- When a user **enters** a URL address into a web browser, he's **requesting** a specific document from a web server
- The web server **maps** the URL to a resource on the server (or to a file on the server's network) and **returns** the requested resource to the client.

□ When given a web page URL, a web browser uses HTTP **to request** and **display** the web page found at that address.



What could be visible for the user?

Server-Side Scripts

- ❑ The power of the **web resides not only in serving content to users**, but **also in**
 - responding to requests from users (form processing) and
 - generating web pages with dynamic content.

- ❑ **Server-side scripting**
 - have a wider range of **programmatic capabilities** than their client-side equivalents.
 - Executes on server (e.g. PHP)
 - Scripting is transparent to client: content received is normal HTML
 - Code is hidden from user

- ❑ **Server-side Advantages**
 1. Ability to interact with a relational **database**
 2. Perform file **manipulations** on the **server**
 3. Generate **responses** based on users' **requests**
 4. **Security**: has access to server's private data; client can't see source code
 5. Script **execution** is hidden from the user
 6. Accessibility : Server-side code is **browser independent**
 7. **Scalability** -- Web-based 3-tier architecture can scale out

Server-side technologies

- ❑ **ASP (Active Server Pages)**- This was Microsoft's first server-side technology. ASP code (using the VBScript programming language) can be embedded within the HTML.
 - ASP programming code is **interpreted at run time**, hence it can be slow in comparison to other technologies.
- ❑ **ASP.NET**- This replaced Microsoft's older ASP technology. ASP.NET is part of Microsoft's .NET Framework and **can use any .NET programming language** (though C# is the most commonly used).
 - ASP.NET is essentially limited to Windows servers.
- ❑ **JSP (Java Server Pages)**- JSP uses Java as its programming language and like ASP.NET it **uses an explicit object-oriented** approach and is used in large enterprise web systems and is integrated into the J2EE environment.
 - JSP uses the Java Runtime Engine, thus it also uses a Just-In-Time (JIT) compiler for fast execution time and is cross-platform.

Server-side technologies..

- ❑ **Node.js-** This is a more recent server environment that uses JavaScript on the server side.
 - Node.js has its own web server software, thus eliminating the need for Apache, IIS, or some other web server software.
- ❑ **Python-** is an object-oriented programming language that has many uses including being used to create web applications.
 - It is also used in a variety of web development frameworks such as Django and Pyramid.
- ❑ **Ruby on Rails-** This is a web development framework that uses the Ruby programming language.
 - It integrates features such as templates and engines that aim to reduce the amount of development work required in the creation of a new site.

Server-side technologies..

- ❑ **PHP** - stands for “**PHP: Hypertext Preprocessor**”.
- ❑ **PHP** is a dynamically typed language that can be embedded directly within the HTML, though it now supports most common **object-oriented** features, such as classes and inheritance.
 - By default, PHP pages are compiled into an intermediary representation called **opcodes**.
- ❑ **PHP** is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language.

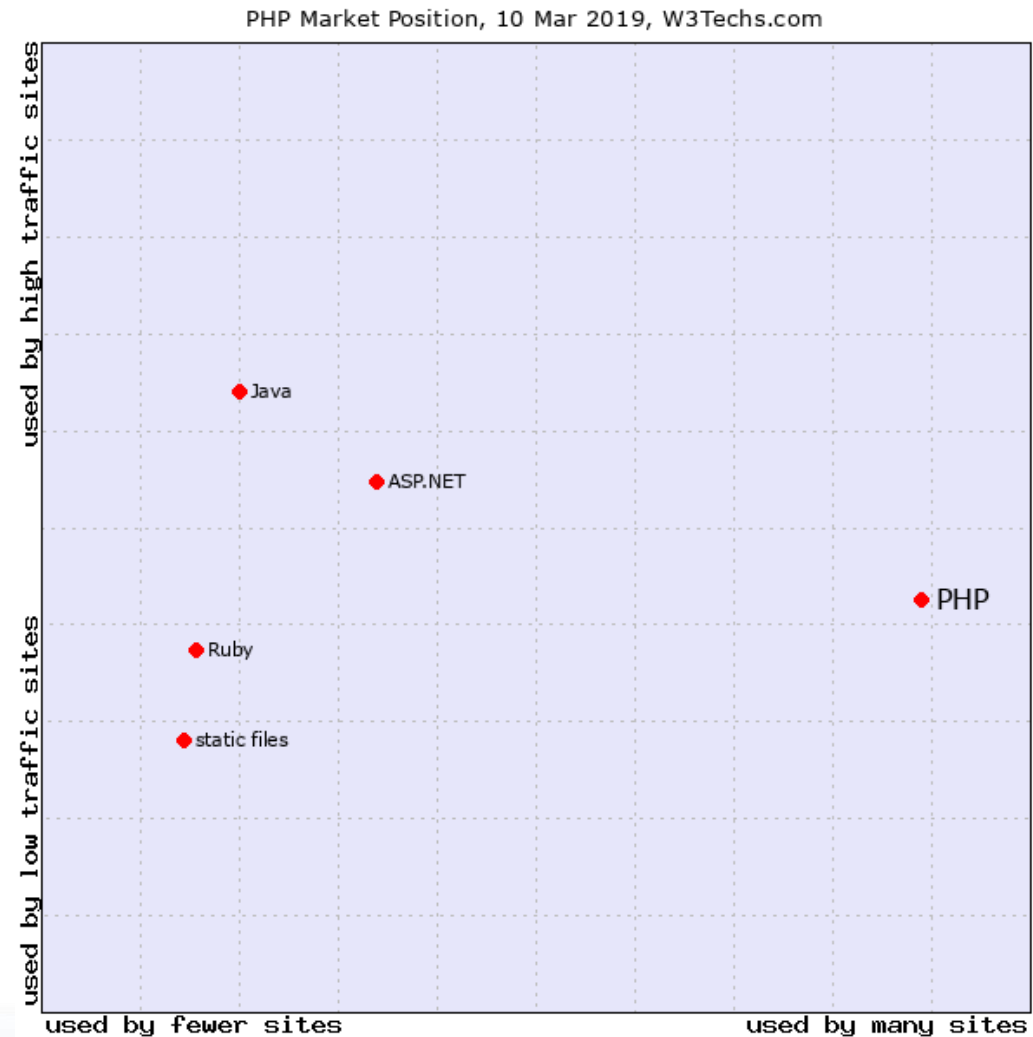
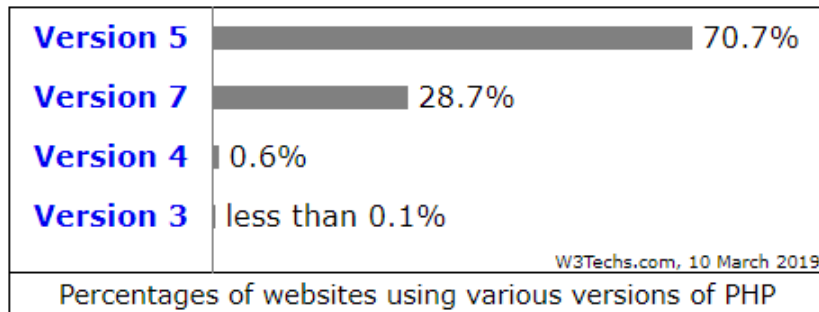
Most popular server-side programming languages

© W3Techs.com	usage	change since 1 September 2019
1. PHP	79.0%	-0.1%
2. ASP.NET	10.8%	-0.1%
3. Java	3.8%	
4. Ruby	2.7%	
5. static files	2.0%	

percentages of sites

Note: a website may use more than one server-side programming language

Usage statistics and market share of PHP for websites



Comparison of the usage of PHP vs. Python for websites

Popular sites using PHP

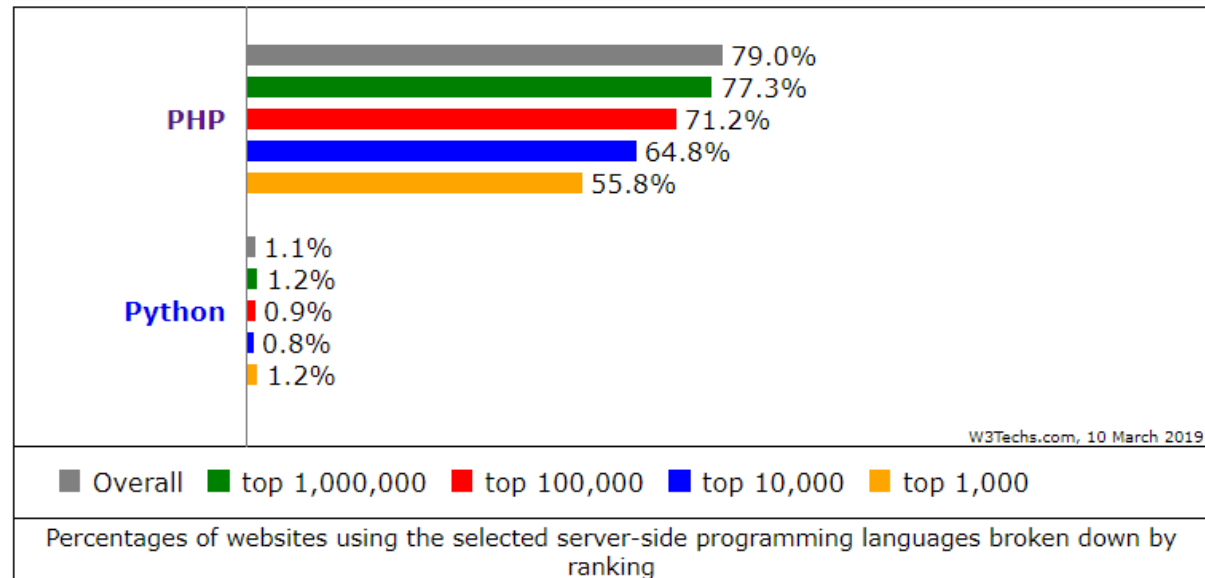
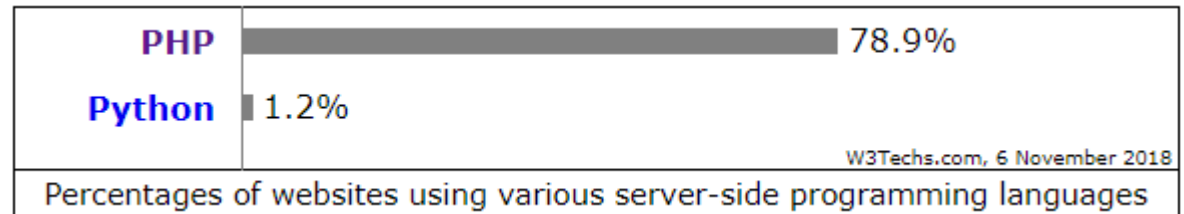
- Facebook.com
- Wikipedia.org
- 360.cn
- Sina.com.cn
- Vk.com
- Babytree.com
- Wordpress.com
- Pinterest.com
- Ettoday.net
- Mama.cn

Random selection of sites using PHP

- Polovniautomobili.com
- Tapsell.ir
- Disruptarian.com
- Incentient.com
- Evolife.cn

Sites using PHP only recently

- Youth.cn
- Ali213.net
- Free.fr
- Motor1.com
- Wayfair.com



Why PHP?

- ❑ **Popularity**- As a result, lots of documentation, books, and web tutorials.
- ❑ **PHP is easy to learn and runs efficiently on the server side**
- ❑ **PHP is free, open source and a platform independent**- implementations exist for all major UNIX, Linux, Mac and Windows operating systems.
- ❑ **PHP supports a wide range of databases** - Support for MySQL, Oracle, dbm, DB2, PostgreSQL. It can connect to any database which provides an ODBC driver (Open Database Connectivity Standard) – e.g. MS Access.
- ❑ **Existing Libraries**- PHP was originally designed for web use – lots of functions for common web-development tasks (e.g. Sending email, XML parsing, etc.)
- ❑ **Object-Oriented Programming** - Similar syntax and features as C++ and Java – inheritance, attribute visibility (private, protected), abstract classes/methods, constructors and destructors, etc.
- ❑ **PHP code can be embedded into HTML markup**, or it can be **used in combination** with various web template systems, web content management systems and web frameworks.

The official PHP resource: www.php.net

Applications of PHP Scripts

Three main areas where PHP scripts are used:

❑ Server-side scripting.

- Server side scripting is the first purpose of PHP. All you need to start working on a desktop PC with PHP is a PHP Parser, a webserver (such as Apache) and a web browser like Google Chrome.

❑ Command line scripting.

- If you want to use PHP on Linux or task scheduler on Windows, then you don't really need a web server, but only a PHP Parser. This is called “command line scripting”.

❑ Desktop applications.

- Although, PHP is not a suitable language for development of desktop applications, but it supports some advanced features like [PHP-GTK](#) to write such programs.
- Use [WinBinder](#) is a (Windows only) alternative to PHP-GTK

How to start & install PHP

❑ Server-Side Scripting Language

- Must have a **web server** and the **PHP interpreter** installed.
- PHP interpreter processes pages before they are served to clients.

❑ Before starting PHP, you need a **web host** with **PHP** and **MYSQL**. For this, you should also install a **web server** such as **Apache**.

- **Apache HTTP Server**, maintained by the *Apache Software Foundation*, is currently the **most popular web server**. It's open source software that runs on UNIX, Linux, Mac OS X, Windows and numerous other platforms.
- **MySQL** is the **most popular open-source database management system**. It runs on Linux, Mac OS X and Windows.

How to start & install PHP..

- ❑ To do it locally on your PC, you can download [Apache, MYSQL and PHP](#) in your machine. They can be downloaded separately but this also requires additional configuration on your part.

- ❑ Thus, [you can download them one package](#):

- LAMP : Linux, Apache, MYSQL, PHP
- MAMP: Mac, Apache, MYSQL, PHP
- WAMP: Windows, Apache, MYSQL, PHP
- [XAMPP: X-OS, Apache, MYSQL, PHP , Perl](#)

All these are used for serving php websites and acts as the **local server** so that you can see your working website without uploading it first.

❑ **XAMPP**

- Combines an Apache web server, PHP, and MySQL into one simple installation service.
- Very little configuration required by the user to get an initial system up and running.

How to start & install PHP..

- ❑ We will use the **XAMPP** integrated installer provided by the Apache Friends website (www.apachefriends.org)
 - To **download**: <https://www.apachefriends.org/index.html>
- ❑ Choose the installer for your platform. Carefully follow the provided installation instructions and be sure to read the entire installation page for your platform!
- ❑ Also, it is recommended to a source code editor to help writing the code, organizing and displaying the files.
 - Sublime Text is recommended for this purpose
 - To **download** <https://www.sublimetext.com/3>



More explanation in PDF file posted in the blackboard...

How to run the code..

- ❑ With assuming you are using XAMPP:
- ❑ Now that the Apache HTTP Server is running on your computer, you can put your PHP files into the server's web space (XAMPP's **htdocs folder**).
 - **C:\xampp\htdocs**
- ❑ URL of files will have localhost as address portion.
 - save your file to C:\xampp\htdocs\name_file.php
 - Then Visit it at: **http://localhost/name_file.php**
- ❑ You can create folders also to organize your work, but it **should be inside hotdocs** folder..

>> **Use Sublime text (or any other editor)** to create the file and locate them in your folder in the local host

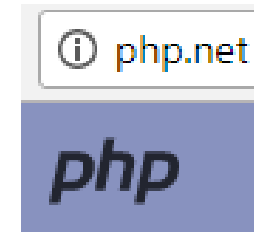
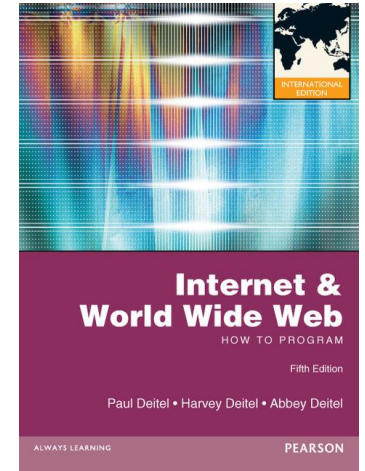
Useful Resources

- ❑ “Internet & World Wide Web: How to Program 4th and 5th editions”

© Pearson Education

- ❑ Lots of resources are available at <http://www.php.net>

- Documentation: manual
 - <http://www.php.net/manual/en/>
 - <http://us2.php.net/manual/en/index.php>
- Tutorials
 - <http://php.net/manual/en/tutorial.php>
- Documented PHP functions
 - <http://us2.php.net/quickref.php>



- ❑ W3schools tutorial

<http://www.w3schools.com/php/default.asp>

w3schools.com

PHP basics

- ❑ PHP is **NOT case-sensitive** except **variables** that are case-sensitive
- ❑ PHP, **like JavaScript**, is a dynamically typed language that can be embedded directly within the HTML.
- ❑ **Unlike JavaScript** it uses classes and functions in a way consistent with other object-oriented languages such as C++, C#, and Java.
- ❑ The syntax for loops, conditionals, and assignment is **identical to JavaScript**, only differing when you get to **functions, classes, and in how you define variables**.
- ❑ The **most important fact** about PHP is that the **programming code can be embedded directly within an HTML file**.
 - However, instead of having an **.html** extension, a PHP file will usually have the extension **.php**
 - **>> A PHP file normally contains HTML tags, and some PHP scripting code.**

PHP basics..

- ❑ A PHP script can be placed anywhere in the document.
- ❑ To differentiate it from the HTML, PHP script **must be contained** within an opening `<?php` tag and a matching closing `?>` tag
- ❑ **A PHP script** is executed on the server, and the plain HTML result is sent back to the browser.



❑ Comments:

- `//` single-line comment
- `#` single-line comment
- `/*` multi-line comment `*/`

```
<?php

# single-line comment

/*
This is a multiline comment.
They are a good way to document functions or complicated blocks of code
*/

$artist = readDatabase(); // end-of-line comment

?>
```

Variables and Data types

- ❑ Variables in PHP are **dynamically typed**
- ❑ Variables are also **loosely typed** in that a variable can be assigned different data types over time.
- ❑ To declare a variable you must preface the variable name with the **\$** symbol.
 - Whenever you use that variable, you must also include the **\$** symbol with it.
- ❑ You should note that in **PHP the name of a variable is case-sensitive**,
 - **\$count** and **\$Count** are references to **two different variables**.
- ❑ While PHP is **loosely typed**, it **still does have data types**, which describe the type of content that a variable can contain.

- ❑ Built-in types:
 - `int` or `integer` : integer
 - `float` or `double` : real number
 - `bool` or `boolean` : logical (true/false)
 - `string` : text string
 - `NULL` : variable has no value
 - `Resource` ?
- ❑ The `type can change` if value is changed.
- ❑ In `mixed expressions`, type is converted automatically.
 - `Can also cast`, e.g. `(int)$x`

Converting Between Data Types

- ❑ Type conversions can be performed using function **settype**.

```
<?php
$foo = "5bar"; // string
$bar = true;   // boolean

settype($foo, "integer"); // $foo is now 5 (integer)
settype($bar, "string");  // $bar is now "1" (string)
?>
```

- Variables are typed based on the values assigned to them.

- ❑ Function **gettype** returns the current type of its argument.

```
<?php
$data = array(1, 1., NULL, new stdClass, 'foo');

foreach ($data as $value) {
    echo gettype($value), "\n";
}
?>
```

The output

```
integer
double
NULL
object
String
```


Converting Between Data Types

- ❑ Another option for conversion between types is **casting** (or **type casting**). Casting does not change a variable's content—it creates a temporary copy of a variable's value in memory.

```
<?php
$foo = 10; // $foo is an integer
$bar = (boolean) $foo; // $bar is a boolean
?>
```

The casts allowed are:

- (int), (integer) - cast to **integer**
- (bool), (boolean) - cast to **boolean**
- (float), (double), (real) - cast to **float**
- (string) - cast to **string**
- (array) - cast to **array**
- (object) - cast to **object**
- (unset) - cast to **NULL**

Constants

- ❑ A constant can be defined anywhere but is typically defined near the top of a PHP file via the `define()` function (NO \$ sign before the constant name).
- ❑ Unlike variables, constants are automatically **global** across the entire script.
 - `define(name, value, case-insensitive)` *case-insensitive*: Default is false

```
<!DOCTYPE html>
<html> <body>
<?php
define("GREETING1", "Welcome to PHP!");
define("GREETING2", "Welcome to PHP again!", true);

function myTest() {
    echo GREETING1;
    echo "<br/>" . greeting2;
}
myTest();
?>
</body> </html>
```

```
Welcome to PHP!
Welcome to PHP again!
```

echo and print Statements

- ❑ In PHP there are **two basic ways to get output**: **echo** and **print**.
- ❑ Almost the same, but:
 - **echo** has no return value while **print** has a return value of 1 so it can be used in expressions.
 - **echo** can take multiple parameters while **print** can take one argument.
 - **echo** is marginally faster than **print**.
- ❑ The **echo** statement can be used with or without parentheses: `echo` or `echo()`.

```
<html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>

</body>
</html>
```

PHP is Fun!

Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.

Notice that the text can contain HTML markup

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

echo and print Statements..

- ❑ how to output text and variables with the echo statement:

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

Learn PHP

Study PHP at W3Schools.com
9

- ❑ The print statement can be used with or without parentheses: print or print().

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>
```

Learn PHP

Study PHP at W3Schools.com
9

Operators

❑ PHP operators for arithmetic:

- `+` : addition
- `-` : subtraction
- `*` : multiplication
- `/` : division
- `%` : remainder
- `++`, `--` : increment, decrement

❑ Other operators:

- `=` : assignment, yields value assigned
- `+=`, `-=`, etc.: operate and assign
- `.` (period): string **concatenation**

❑ Beware: unlike JavaScript, `+` is **only addition**. `"foo" + "bar"` = 0.

Logical operators

- ❑ Operators for combining boolean values:
 - `&&` : AND `||` : OR `!` : NOT
- ❑ Boolean literals are `FALSE` and `TRUE`
- ❑ Any expression can be converted to boolean:
 - 0, empty string, empty array, NULL, unset variable evaluate to FALSE
 - any other values evaluate to TRUE
- ❑ Comparison operators:
 - `==` equal to
 - `!=` not equal to
 - `<` less than
 - `<=` less than or equal
 - `>` greater than
 - `>=` greater than or equal

PHP Conditional Statements

❑ There are **several statements** in PHP that you can use to make decisions:

- The if statement
- The if...else statement
- The if...elseif....else statement
- The switch...case statement

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

❑ PHP allows an **alternative form** for control statements

- Replace opening brace by **colon :**
- Replace closing brace by **endif** keyword;

```
if( test) :
    // statements
endif;
```

Looping Statements

- ❑ The While Loop
- ❑ The Do...While Loop
- ❑ The For Loop
- ❑ The Foreach Loop
- ❑ Break and Continue Statements
 - The **while** loop and the **do . . . while** loop are quite similar.
 - The **for** loop in PHP has the same syntax as the for loop in JavaScript

```
$count = 0;
while ($count < 10)
{
    echo $count;
    $count++;
}

$count = 0;
do
{
    echo $count;
    $count++;
} while ($count < 10);
```

```
for ($count=0; $count < 10; $count++)
{
    echo $count;
}
```


example

```
<html>
<body>
  <h1>Fibonacci Numbers Less than 100</h1>
  <p>
    <?php
      /* Computes the famous Fibonacci sequence using a loop.
       * The recurrence is  $F_{n+1} = F_n + F_{n-1}$ 
       */
      $fib1 = 1;
      $fib2 = $fib1;
      # Need to print the first number before loop starts
      print $fib1;

      while( $fib2 < 100 ) {
        print ", " . $fib2;
        $fib3 = $fib1 + $fib2; // compute next Fibonacci number
        $fib1 = $fib2;         // shift so $fib1, $fib2 are latest two
        $fib2 = $fib3;
      }
    ?>
  </p>
</body>
</html>
```

```
<html>
<body>
  <h1>Fibonacci Numbers Less than 100</h1>
  <p>
    1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89  </p>
  </body>
</html>
```

Fibonacci Numbers Less than 100

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

Alternate syntax for Control structures

- ❑ PHP has an **alternative syntax for most of its control structures** (namely, the if, while, for, foreach, and switch statements).
- ❑ the **colon (:)** replaces the opening curly bracket, while the closing brace is replaced with **endif; , endwhile; , endfor; , endforeach; , or endswitch;**

```
<?php if ($userStatus == "loggedin") : ?>
    <a href="account.php">Account</a>
    <a href="logout.php">Logout</a>
<?php else : ?>
    <a href="login.php">Login</a>
    <a href="register.php">Register</a>
<?php endif; ?>
```

Functions

❑ In PHP there are **two types of functions**:

- A **user-defined function** is one that you the programmer define.
- A **built-in function** is one of the functions that come with the PHP environment

- Declaring a function:

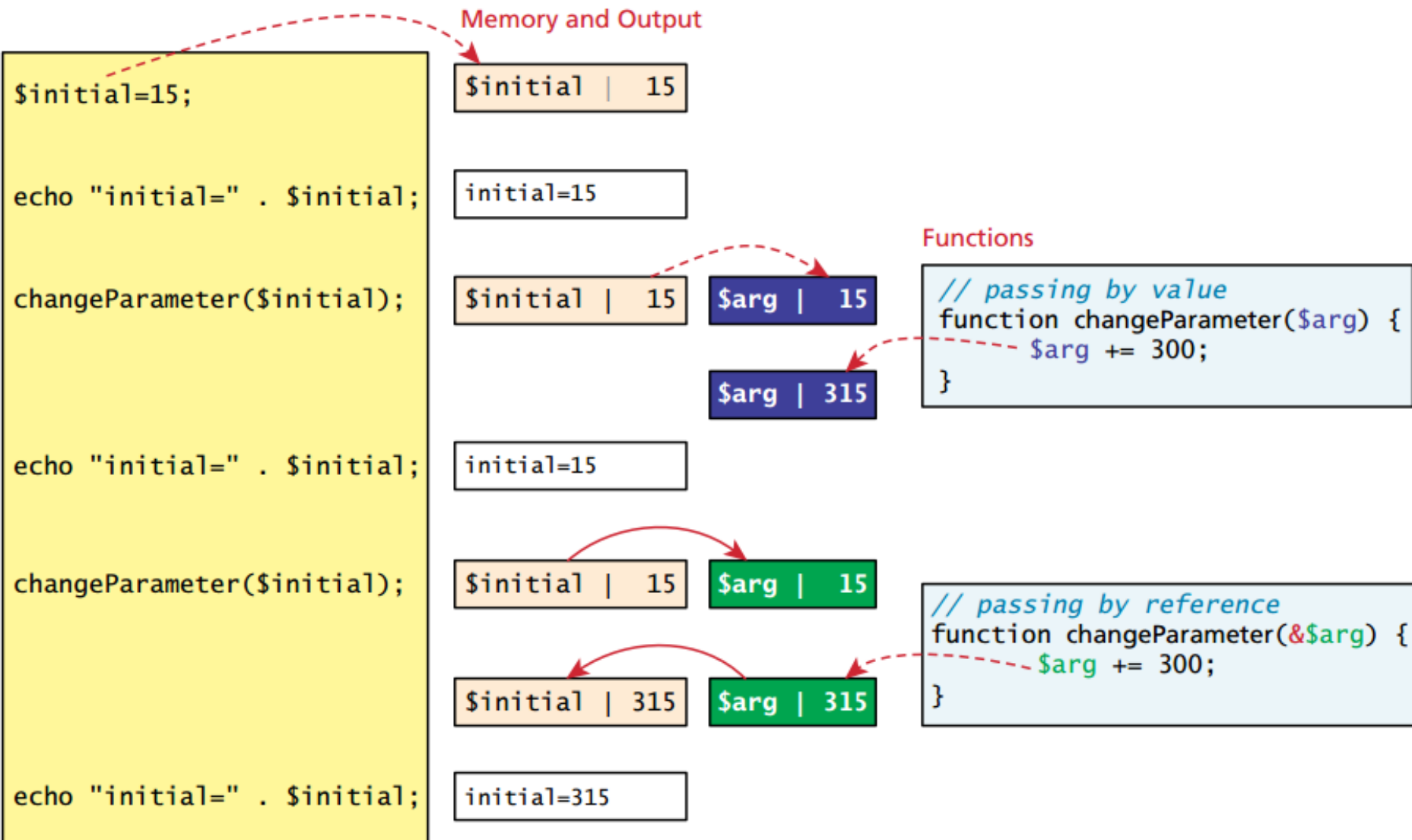
```
function function-name ( $arg1, $arg2, ... )  
{  
    // statements  
}
```

- Argument can be given default value by listing as **\$arg=value**
- Function may return a value using **return** expression

```
<?php  
function add_salutation( $name, $title = "Mr." ) {  
    return "Dear $title $name";  
}  
?>  
  
<!DOCTYPE html>  
<html lang="en">  
    <head>  
        <title>Function example</title>  
        <meta charset="utf-8">  
    </head>  
    <body>  
        <?php  
        print add_salutation("Jones");  
        print "<br />";  
        print add_salutation("Smith", "Ms.");  
        ?>  
    </body>  
</html>
```

Dear Mr. Jones
Dear Ms. Smith

Pass by value versus pass by reference



Example

```
<?php
/* This function returns twice its argument, but leaves
 * original value unchanged since call is by value.
 */
function twice( $x ) { // passing by value
    $x *= 2;
    return $x;
}

/* This function returns no value. It uses call by reference
 * to double its argument.
 */
function double( &$x ) { // passing by reference
    $x *= 2;
}
?>

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Function example</title>
        <meta charset="utf-8">
    </head>
    <body>
        <?php
        $x = 3;                                // original value of x
        print "x = $x<br />";                  // print it to see
        print "2x=" . twice($x) . "<br />";    // print 2x but leave x unchanged
        print "x = $x<br />";                  // show that x is unchanged
        double($x);                             // now double x for real
        print "x = $x<br />";                  // show that x is doubled
        ?>
    </body>
</html>
```

- By default, function arguments are **passed by value**.
 - they are copies of the originals
 - may be **changed** by the function without affecting the originals.
- For **call by reference**, precede name by &

```
x = 3
2x=6
x = 3
x = 6
```

Math functions

- ❑ PHP has many mathematical functions.
 - `abs(x)` : magnitude of x
 - `min(x,y)`, `max(x,y)` : smaller, larger of x, y
 - `sqrt(x)` : \sqrt{x}
 - `round(x)` : round x to nearest integer
 - `round(x,d)` : round x to d digits after decimal point
- ❑ Unlike JavaScript, **no Math. prefix.**

```
<body>
<h1>Table of square roots</h1>
<h2>rounded to 4 decimal places</h2>
  <?php
    for( $i=1; $i <= 25; $i++) {
      print "sqrt(" . $i . ")=" . round(sqrt($i),4) . "<br/>";
    }
  ?>
</body>
```

- ❑ String values may be enclosed in either:
 - single quotes 'This is a string' or
 - double quotes "Another string"
 - As usual, straight quotes only!

- ❑ **Difference:** variables are expanded inside double quotes but not single quotes.
 - `$x = 'Joe';`
 - `$y = "Hello, $x!"; // $y = 'Hello, Joe!'`
 - `$z = 'Hello, $x!'; // $z = 'Hello, $x!'`

- ❑ Strings may **span multiple lines** between opening and closing quotes.
 - `$str = "Line 1
Line 2"`

String functions

- ❑ There are many **built-in functions for working with strings**:
 - `strlen(string)` : length of string
 - `trim(string)` : remove leading & trailing spaces
 - `str_word_count()`: counts the number of words in a string
 - `strrev()`: reverses a string
 - `strpos()` : searches for a specific text within a string
 - `str_replace()`: replaces some characters with some other characters in a string.
 - and many more...
- See more functions
https://www.w3schools.com/php/php_ref_string.asp

example

explode() function

- ❑ The `explode()` function breaks a string into an array.

`explode(separator, string, limit);` // limit is optional, Specifies the number of array elements to return.

```
<!DOCTYPE html>
<html> <body>

<?php
$str = 'one,two,three,four';

// zero limit
print_r(explode(',',$str,0));
print "<br>";

// positive limit
print_r(explode(',',$str,2));
print "<br>";

// negative limit
print_r(explode(',',$str,-1));
?>
</body> </html>
```

limit:

- **Greater than 0** - Returns an array with a maximum of *limit* element(s)
- **Less than 0** - Returns an array except for the last *-limit* elements()
- **0** - Returns an array with one element

```
Array ( [0] => one,two,three,four )
Array ( [0] => one [1] => two,three,four )
Array ( [0] => one [1] => two [2] => three )
```

- ❑ Arrays in PHP are very **similar to those in JavaScript**
 - Can contain elements of differing types
 - Can grow or shrink dynamically
 - Can have holes (undefined elements)
 - Can be indexed by integer or keyword string

- ❑ **Declaring an array:**
 - `$a = array();` // empty array
 - `$a = array(value1, value2, ...);` // initialized array
 - Accessing an array element, numbering starts at 0: **e.g. `$a[1]`** // is value2

- ❑ In PHP, there are three types of arrays:
 - **Indexed** arrays - Arrays with a numeric index
 - **Associative** arrays - Arrays with named keys
 - **Multidimensional** arrays - Arrays containing one or more arrays

example

- ❑ To see contents of array (or other objects):
 - `print_r(array);`

```
<?php
$colors = array( "Red", "Blue", "Green" );
$colors[20] = "White"; // create a gap
?>
```

```
<?php
$arlen = count($colors);
for($x = 0; $x < $arlen; $x++) {
    echo $colors[$x];
    echo "<br>";
}
?>
```

Red
Blue
Green

Notice: Undefined offset: 3 in C:\xampp\htdocs\PHP-SWE363\helloPHP.php

```
<ul>
    <?php
        foreach( $colors as $set )
        { print "<li>$set</li>\n"; }
    ?>
</ul>
```

- Red
- Blue
- Green
- White

```
<p>Using <code>print_r</code>:</p>
<?php
    print_r($colors);
?>

<pre>
<?php
    print_r($colors);
?>
</pre>
```

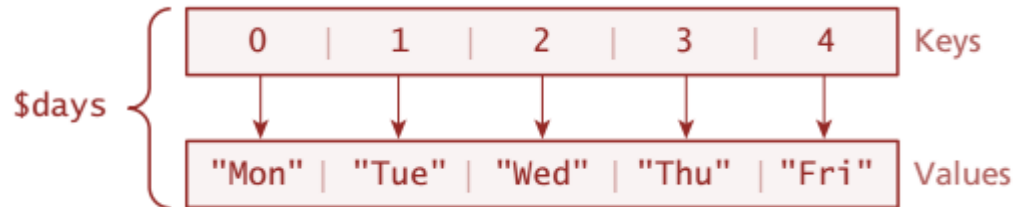
Using print_r:

Array ([0] => Red [1] => Blue [2] => Green [20] => White)

```
    Array
(
    [0] => Red
    [1] => Blue
    [2] => Green
    [20] => White
)
```

PHP Associative Arrays

- ❑ The keys assigned to **values** can be arbitrary and user defined strings.



- ❑ Declaring an associative array:

- `$a = array(key1 => value1, key2 => value2, ...);`

- ❑ **Array keys** in most programming languages are limited to integers, start at 0, and go up by 1.

- >> In PHP, keys *must* be either **integers** or **strings** and need not be sequential.

- ❑ **Array values**, unlike keys, are not restricted to integers and strings. They can be **any object, type, or primitive supported in PHP**. You can even have objects of your own types, so long as the keys in the array are integers and strings.

PHP Associative Arrays

- ❑ There are two ways to create an associative array:

```
<?php
    // Define an associative array
    $ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
?>
```

```
<?php
    $ages["Peter"] = "22";
    $ages["Clark"] = "32";
    $ages["John"] = "28";
?>
```

- ❑ In PHP, **arrays are dynamic**, that is, they can grow or shrink in size. An element can be added to an array simply by **using a key/index** that hasn't been used, as shown below:

```
<?php
    $ages["Tom"] = "32";
?>
```

Will be added to the end of array.

- ❑ You can also create “**gaps**” by explicitly deleting array elements using the **unset()** function

Arrays functions

- ❑ There are many built-in sort functions, which sort by key or by value.
 - `sort()` - sort arrays in ascending order
 - `rsort()` - sort arrays in descending order
 - `asort()` - sort associative arrays in ascending order, according to the value
 - `ksort()` - sort associative arrays in ascending order, according to the key
 - `arsort()` - sort associative arrays in descending order, according to the value
 - `krsort()` - sort associative arrays in descending order, according to the key

- ❑ `array_rand()` function returns a random key from an array
- ❑ `array_reverse()` function returns an array in the reverse order.
- ❑ `array_walk()` function runs each array element in a user-defined function. The array's keys and values are parameters in the function.
- ❑ `in_array()` function searches an array for a specific value.
- ❑ `shuffle()` function randomizes the order of the elements in the array.
`count` : number of elements in array -- Only counts defined elements: skips holes
- ❑ `array_push`, `array_pop` : add/remove elements from end of array
- ❑ `array_shift`, `array_unshift` : add/remove elements from front of array

Arrays functions...

- ❑ `array_walk()` function runs each array element in a user-defined function. The array's keys and values are parameters in the function.

```
<!DOCTYPE html>
<html>
<body>

<?php
function myfunction($value,$key)
{
echo "The key $key has the value $value<br>";
}
$a=array("a"=>"red","b"=>"green","c"=>"blue");
array_walk($a,"myfunction");
?>

</body>
</html>
```

```
The key a has the value red
The key b has the value green
The key c has the value blue
```

Multidimensional Array

- ❑ An array in which **each element can also be an array** and **each element in the sub-array can be an array** or further contain array within itself and so on.

```
<?php
// Define a multidimensional array
$contacts = array(
    array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    array(
        "name" => "Clark Kent",
        "email" => "clarkkent@mail.com",
    ),
    array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
// Access nested value
echo "Peter Parker's Email-id is: " . $contacts[0]["email"];
?>
```


Viewing Array Structure and Values

- ❑ by using one of two statements — `var_dump()` or `print_r()`.
 - The `print_r()` statement, however, gives somewhat less information.

```
<?php
// Define array
$cities = array("London", "Paris", "New York");

// Display the cities array
Print_r($cities);
?>
```

```
Array ( [0] => London [1] => Paris [2] => New York )
```

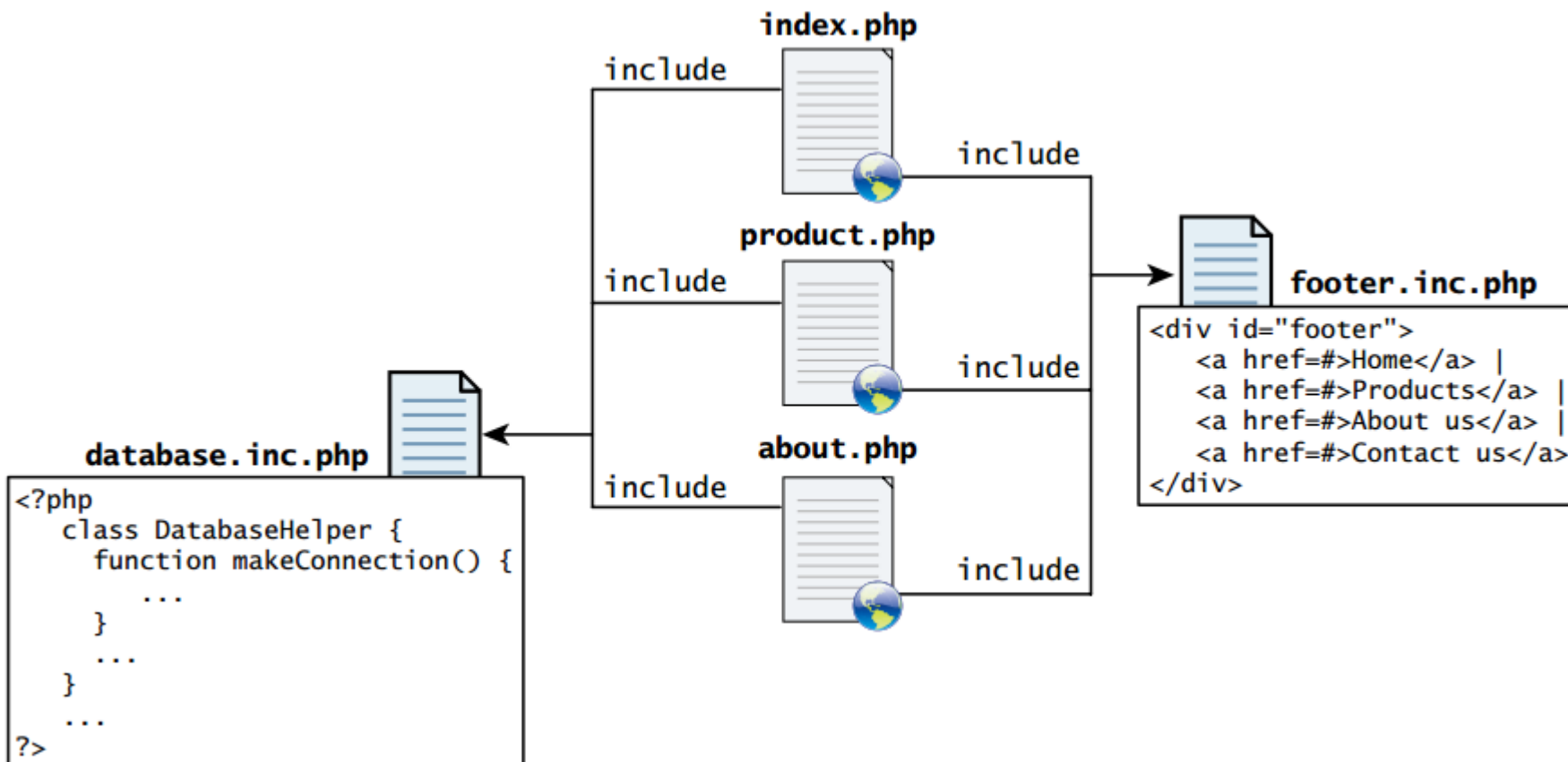
```
<?php
// Define array
$cities = array("London", "Paris", "New York");

// Display the cities array
var_dump($cities);
?>
```

```
array(3) { [0]=> string(6) "London" [1]=> string(5) "Paris" [2]=> string(8) "New York" }
```

Including files

- ❑ Include files provide a mechanism for reusing both markup and PHP code
- ❑ It produces the same result as **copying the script from the file specified and pasted into the location** where it is called.



Including files..

- ❑ The `include()` and `require()` statement allow you to include the code contained in a PHP file within another PHP file.

```
include("path/to/filename"); Or- include "path/to/filename";  
require("path/to/filename"); Or- require "path/to/filename";
```

- **File path is relative to script location** (does not refer to URL space)
- ❑ A typical example is including the header, footer and menu file in all the pages of a website.

```
<body>  
<?php include "header.php"; ?>  
<?php include "menu.php"; ?>  
    <h1>Welcome to Our Website!</h1>  
    <p>Here you will find lots of useful resources.</p>  
<?php include "footer.php"; ?>  
</body>
```

Tutorial Republic

[Home](#) | [About](#) | [Contact](#)

Welcome to Our Website!

Here you will find lots of useful resources.

Copyright © 2014 Tutorial Republic

Including files..

- ❑ The difference between `include` and `require` lies in what happens when the specified file cannot be included (generally because it doesn't exist or the server doesn't have permission to access it).
 - With `include`, a warning is displayed and then execution continues.
 - With `require`, an error is displayed and execution stops.

- ❑ `include_once` and `require_once` : same as above, but prevent including same file twice
 - It is not uncommon for a PHP page to include a file that includes other files that may include other files, and in such an environment the `include_once` and `require_once` statements are certainly recommended.

