PHP

Day 01

Prepared by:
Noha Shehab

Open source dept.

ITI

nshehab@iti.gov.eg

What is php?

- PHP stands for PHP: Hypertext Preprocessor
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is php files

- PHP files can contain text, HTML, JavaScript code,
- and PHP code
- PHP code are executed on the server, and the result is returned to the browser **as plain HTML**
- PHP files have a default file extension of ".php"

What PHP can do?

PHP can generate dynamic page content

PHP can create, open, read, write, and close files on the server

PHP can collect form data

PHP can send and receive cookies

PHP can add, delete, modify data in your database

PHP can restrict users to access some pages on your website

PHP can encrypt data

Why PHP?

PHP runs on different platforms (Windows, Linux,

Unix, Mac OS X, etc.)

PHP is compatible with almost all servers used today (Apache, IIS, etc.)

PHP has support for a wide range of databases

PHP is free. Download it from the official PHP resource: www.php.net

PHP is easy to learn and runs efficiently on the server side

Set Up PHP on Your Own PC

- However, if your server does not support PHP, you must: install a web server
  - install PHP
  - install a database, such as MySQL
  - The official PHP website (PHP.net) has installation
  - instructions for PHP: http://php.net/manual/en/install.php

Basic PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>:

```
<?php
    // Welcome  to php group
?>
```

- The default file extension for PHP files is ".php".

Comments

```
<html>
    <body>
        <?php
            //This is a PHP comment line
            /*
            This is a PHP comment
            block
            */
        ?>
    </body>
</html>
```

PHP Variables

- Variable can have short names (like x and y) or more descriptive names (age, carname, totalvolume).
- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name must begin with a letter or the underscore character
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - A variable name should not contain spaces
  - Variable names are case sensitive ($y and $Y are two different variables)
  - A variable can have the same name as a function. This usage is confusing, however, and should be avoided. Also, you cannot create a function with the same name as another function.

Creating (Declaring) PHP Variables

- PHP has no command for declaring a variable.
- A variable is created the moment you first assign a value to it:
  - $txt="Hello world!";
  - $x=5;
- After the execution of the statements above, the variable txt will hold the value Hello world!, and the variable x will hold the value 5.
- Note: When you assign a text value to a variable, put quotes around the value.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In a strongly typed programming language, we will have to declare **(define) the type and name of the variable before using it**.

PHP Variable Scopes

The scope of a variable is the part of the script where the variable can be referenced/used.

- PHP has four different variable scopes:
  - Local
  - Global
  - Static
  - Parameter

Local Scope

A variable declared within a PHP function is local and can only be accessed within that function:

Example

```
<?php
        $x=5; // global scope
        function myTest()
        {
                $x=5;
            echo $x; // local scope
        }
        myTest();
?>
```

Global Scope

A variable that is defined outside of any function, has a global scope.

Global variables can be accessed from any part of the script, EXCEPT from within a function.

To access a global variable from within a function, use the global keyword:

Global scope Example.

```php
<?php
	$x=5; // global scope
	$y=10; // global scope
	function myTest()
	{
		global $x,$y;
		 $y=$x+$y;
	}
	myTest();
	echo $y; // outputs 15
?>
```

Static Scope

When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted.

To do this, use the static keyword when you first declare the variable:

Example

```
function myTest()
{
    static $x=0;
    echo $x;
    $x++;
}
myTest();
myTest();
myTest();
```

Parameter scope

A parameter is a local variable whose value is passed to the function by the calling code.

Parameters are declared in a parameter list as part of the function declaration:

Example :

```
<?php
function myTest($x)
{
    echo $x;
}
myTest(5);
?>
```

Variables scoping summary

The **six** basic scope rules in PHP are as follows:

- Built-in **super global variables** are visible everywhere within a script.
- **Constants,** once declared, are always visible globally; that is, they can be used inside and outside functions.
- **Global variables** declared in a script are visible throughout that script, but not inside functions.
- **Global Variables** inside functions refer to the global variables of the same name.
- **Static variables** created inside functions are **invisible from outside the function** but keep their value between one execution of the function and the next.
- Variables created inside functions are **local** to the function and cease to exist when the function terminates.

Super global

Super global or auto global and can be seen **everywhere,** both inside and outside functions.

The complete list of super global is as follows:

- **$_GET**—An array of variables passed to the script via the GET method.
- **$_POST**—An array of variables passed to the script via the POST method.
- **$_REQUEST**—An array of all user input including the contents of input including $_GET, $_POST & $_COOKIE
- **$_COOKIE**—An array of cookie variables
- **$_FILES**—An array of variables related to file uploads
- $_ENV—An array of environment variables
- **$_SESSION**—An array of session variables

Constants

You can define these constants using the define function:

- define('CONST1', 100);
- One important difference between constants and variables is that when you refer to a constant, it does not have a dollar sign in front of it. If you want to use the value of a constant, use its name only.
- echo CONST1;

Echo & Print

- In PHP there is two basic ways to get output: **echo and print.**
- There are some differences between echo and print:
  - echo - can output one or more strings
  - print - can only output one string, and returns always 1

Tip: echo is marginally faster compared to print as echo does not return any value.

```
<?php
    echo "<h2>PHP is fun!</h2>";
    echo "Hello world!<br>";
    echo "This", " string", " was", " made", " with multiple
    parameters.";
?>
```

Variables datatypes

A variable's type refers to the kind of data stored in it.

•PHP supports the following basic data types:

- Integer—Used for whole numbers
- Float (also called double)—Used for real numbers
- String—Used for strings of characters
- Boolean—Used for true or false values
- Array—Used to store multiple data items
- Object—Used for storing instances of classes

Variable casting

You can pretend that a variable or value is of a different type by using a type cast. You simply put the temporary type in parentheses in front of the variable you want to cast.

- For example, you could have declared the two variables from the preceding section using a cast:
- $var1 = 0;
- $var2 = (float)$var1;

Variable of variable

PHP provides one other type of variable: the variable of variable.

- Variable variables enable you to change the name of a variable dynamically.
- For example, you could set
  - $varname= ‘var1’;
- You can then use $$varnamein place of $var1. For example, you can set the value of $var1 as follows:
  - $$varname= 5;
- This is exactly equivalent to
  - $var1= 5;

Arithmetic operators

- Arithmetic operators are straightforward; they are just the normal mathematical operators.

- With each of these operators, you can store the result of the operation, as in this example:
- $result = $a + $b;

Arithmetic operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

Operators

You can use the string concatenation operator to add two strings and to generate and store a result much as you would use the addition operator to add two numbers:

```
$a = “Hello, “;
$b = “World!”;
$result = $a.$b;
```

The $result variable now contains the string “Hello, World!”

Comparison operators

Expression	Meaning	Example	Illustrate
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7

Combined operators

Combined assignment operators exist for each of the arithmetic operators and for the string concatenation operator.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus
$a.=$b	$a=$a.$b	Concatination

Pre/Post-increment.

- The pre-and post-increment (++) and decrement (--) operators are similar to the +=and -= operators, but with a couple of twists.
- Example

```
$a=4;
echo ++$a;//echo 5 , value of $a = 5
$a=4;
echo $a++;  //echo 4 , value of $a = 5
```

Reference operator

- The reference operator (&, an ampersand) can be used in conjunction with assignment.
  - $a = 5;
  - $b = $a;
- These code lines make a second copy of the value in $a and store it in $b. If you subsequently change the value of $a, $b will not change:
  - $a = 7; // $b will still be 5
- You can avoid making a copy by using the reference operator. For example,
  - $a = 5;
  - $b = &$a;
  - $a = 7; // $a and $b are now both 7

Reference tip.

References can be a bit tricky. Remember that a reference is like an **alias** rather than like a **pointer**. Both $a and $b point to the same piece of memory. You can change this by unsetting one of

Logical operators

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

The logical operators combine the results of logical conditions. $a, is between 0 and 100. using the AND operator, as follows:

- $a >= 0 && $a <=100

Instance-of operator

There is one type operator: instanceof.

This operator is used in object-oriented programming.

- The instanceof operator allows you to check whether an object is an instance of a particular class, as in this example:

```
class sampleClass{};
$myObject= new sampleClass();
if ($myObject instanceof sampleClass)
echo “myObjectis an instance of sampleClass”;
```

Variable functions

- To use gettype()
  - you pass it a variable. It determines the type and returns a string containing the type name: bool, int, double (for floats), string, array, object, resource, or NULL.
  - It returns unknown type if it is not one of the standard types.
- Settype()
  - you pass it a variable for which you want to change the type and a string containing the new type for that variable from the previous list.

Common variables functions

- is_array()—Checks whether the variable is an array.
- is_double(), is_float(), is_real() (All the same function)—Checks whether the variable is a float.
- is_long(), is_int(), is_integer() (All the same function)—Checks whether the variable is an integer.
- is_string()—Checks whether the variable is a string.
- is_bool()—Checks whether the variable is a boolean.

Common variables functions

- is_object()—Checks whether the variable is an object.
- is_resource()—Checks whether the variable is a resource.
- is_null()—Checks whether the variable is null.
- is_scalar()—Checks whether the variable is a scalar, that is, an integer, boolean, string, or float.
- is_numeric()—Checks whether the variable is any kind of number or a numericstring.
- is_callable()—Checks whether the variable is the name of a valid function.

Common variable functions

- **isset()** function takes a variable name as an argument and returns true if it exists and false otherwise. You can also pass in a comma-separated list of variables, and isset() will return true if all the variables are set.
- You can wipe a variable out of existence by using its companion function, unset(), which has the following prototype:

Common variable functions

- **empty()** function checks to see whether a variable exists and has a nonempty, nonzero value; it returns true or false accordingly. It has the following prototype.

Flow control

- If condition
- Switch case
- Foreach
- Break, continue, exit

Flow control (If condition)

- if (*condition*) {
  code to be executed if this condition is true;
  } elseif (*condition*) {
  code to be executed if first condition is false and this condition is true;
  } else {
  code to be executed if all conditions are false;
  }

Flow control (Switch case)

- switch (*n*) {
  case *label1:*
  code to be executed if n=label1;
  break;
  case *label2:*
  code to be executed if n=label2;
  break;
  case *label3:*
  code to be executed if n=label3;
  break;
  …
  default:
  code to be executed if n is different from all labels;
  }

Flow control (foreach)

- foreach ($*array* as $*value*) {
  code to be executed;
  }

Flow control (Break & Continue)

- Use the break statement in a loop, execution of the script will continue at the next line of the script after the loop.

```
for ($i = 0; $i < 5; ++$i) {
        if ($i == 3)
            break;
        print "$i <br>";
    }
```

Flow control (Break & Continue)

- If you want to jump to the next loop iteration, you can instead use the continue statement.

```
for ($i = 0; $i < 5; ++$i) {
        if ($i == 2)
            continue;
        print "$i <br>";
    }
```

Exit

- If you want to finish executing the entire PHP script, you can use exit. This approach is typically useful when you are performing error checking

```
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2)
        exit;
    print "$i <br>";
}
```

Looping

- For
- While
- Do While

Looping (For)

```
for( expression1; condition; expression2){
expression3;
}
```

- expression1 is executed once at the start. Here, you usually set the initial value of a counter.
- The condition expression is tested before each iteration. If the expression returns false, iteration stops. Here, you usually test the counter against a limit.
- expression2 is executed at the end of each iteration. Here, you usually adjust the value of the counter.
- expression3 is executed once per iteration. This expression is usually a block of code and contains the bulk of the loop code.

Looping(while)

- The while loop executes a block of code as long as the specified condition is true.

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

Example

```
$x = 0;
    while($x <= 100) {
        echo " <br> The number is: $x ";
        $x+=10;
     }
```

Looping(do while)

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

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

Example

```
$x = 100000;
    do {
        echo " <br> The number is: $x ";
        $x++;
    } while ($x <= 5);
```

Thank you