PHP

Team 7
Miriam Gutierrez
Gabriel Ortega-Gingrich
Nick Curinga

Overview

- What’s PHP?
- History
- Intro to PHP
- Operators
- Control Structures
- Functions
- Classes
- Exceptions/Error Handling
- Pre-defined Variables
- Supported Protocols/Wrappers

What’s PHP?

- PHP: Hypertext Preprocessor (recursive acronym)
- PHP is a server-side scripting language designed for web-development
- General-purpose programming language
- Could be used to perform command-line scripting
- Evaluate form data sent from a browser
- Build custom web content to serve browser

History

- Created by Rasmus Lerdorf in 1994
- Personal Home Page/Forms Interpreter (PHP/FI)
- PHP Tools

  - Originally used to track visits to his online resume
- Version 1.0

  - provide framework so users could develop simple dynamic web applications

  - database interaction

History cont..

- FI (Forms Interpreter)
  - Perl-like Variables
  - HTML embedded syntax
  - Automatic interpretation of form variables
- Back to PHP/FI
  - included built-in support for DBM, mSQL
  - cookies, user-defined function support
- Version 3.0 - Now known as PHP: Hypertext Preprocessor
  - extensibility features
  - Object Orientated programming support

History cont..

- PHP 4
  - Speed and reliability over PHP3
  - added references, Boolean types
  - output buffering, new array functions
- PHP 5
  - exception handling using try catch structure
  - Filter Library
  - Bette XML tools
  - iterators
- PHP 7
  - PHP next generation (PHPNG)

Intro to PHP

- Delimiters

  <?php   ?>
- Output

  - echo
  - print
- Newline

  - For HTML, add “<br>” or “<p>” tag to the end of string

  - if saving to File, use “\n”
- Concatenation (.)
- ; to end php statements

Intro to PHP- Data Types

PHP supports 8 primitive types

- Boolean
- Integer
- Float/Double
- String
- Objects
- Arrays
- Null
- Resources

Intro to PHP - Variables

- Variables are specific case-sensitive names
- Dynamically typed
- In PHP, variables must always start with a $
- Ex:

```
$myName = John;
$myAge = 23;
```

*Note PHP statements end with semicolon

Variables cont..

- To assign by reference, use & at the beginning of the variable
- Only named variables may be assigned by reference

```
$foo = ‘hi’;
$bar = &$foo;              // references $foo
```

- Variable Variables takes the value of a variable and treats that as the name of a variable. Use two dollar signs on the same variable name.

```
$a = “hello”;
$$a = “world”;
```

- Scalar type declarations - PHP7

Arrays

- An array is actually an ordered map
- When declaring an array, you use the array() language construct

```
$array = array(item1, item2……)
```

- To access an item in a array, we use [] or {}

```
$tens = array(10, 20, 30, 40, 50);
echo $tens[2];
echo $tens{2};    // Both will echo out the number 30
```

- To modify an array, you could assign values to the array with specific key in brackets, or leave the brackets empty and set it a value

```
$tens[1] = 60;
$tens[] = 70;   // Will be added to end of array
```

Arrays cont..

- As stated before, arrays are ordered maps
- Can assign key as an integer or string.
- Useful functions:
- unset() // to remove keys from an array
- array_values() // to reindex an array
- array_diff() //compare arrays
- count() // count number of items in array
- sort() // sort array

Resources

- A special variable, holding a reference to an external resource
- Hold special handles to opened files, database connections, image canvas areas, etc.
- Useful functions:
- Get_resource_type()   //returns the type of the resource
- is_resource()   // returns true if variable is resource
- Many other functions for each type of resource

```
$fp = fopen("foo", "w");
echo get_resource_type($fp);    // prints: stream
```

Operators

- .   //Concatenation
- ===   //identical
- !==   // not identical
- <=>   //spaceship
- @   // error control
- Php does not specify in which order an expression is evaluated

```
Ex: $a = 1;
    echo $a + $a++;         //will either print 2 or 3
```

Control Structures

- If /else statements
- Elseif statements
- While loops
- Do while
- For
- For each
- Switch
- Declare
- Included/ Required

Control Structures - If

If and else:

```
if(condition){
        Statement1
        ..
}else{
        Statement2;
        ….
}
```

Elseif:

Like C but one change….

```
if(condition){
        Statement1
}
elseif(condition){
        Statement2
}
else{
        Statement3
}
```

Control Structures - Loops

While:

```
while(condition){

        Statement

}
```

Do-While:

```
do{

        Statement

}while(condition);
```

For loops:

```
for(expr1;expr2;expr3){
        Statement
}
```

Foreach loop

- Foreach only works on arrays and objects
- Syntax

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

- If you use first form to modify array elements, you need to add ‘&’ before $value as such:

```
foreach ($arr as &$value)
    echo $value;
```

Continue and Break statements

- *Break* ends execution of current loop
- *Continue* is used within looping structures to skip the rest of current loop iteration and continue execution
- Both continue and break also accept a numeric argument
- For break it tells how many nested enclosing structures are to be broken out of
- For Continue it tells how many levels of enclosing loops it should skip to the end of.

Declare

- The declare construct is used to set execution directives for a block of code

```
declare (directive)
    Statement
```

- *Directive* lets the behavior of the declare block be set
- Directives are given as only literals
- 3 directives

  - ticks
  - encoding
  - strict_types

Include, Require

- *Include* and *Require* are both the same in that they include and evaluate the file specified.
- If file is not found, include will emit a warning which allows the script to continue, while require will produce a fatal E_ Compile_Error level error and halt the script
- Wherever include/require is put, any variables available in the calling file will be available within the called file from that point onward.
- Include_once and require_once are the same as include and require, respectively, but the *once* part will make sure that if the file has already been included, it will not include it again

Alternative Syntax for Control Structures

- Can use colon (:) as opening and end the control structure with *endstructure:*
  - endif;, endwhile;, endfor;, endforeach;, endswitch;

Ex:

```
if (condition):
        Statement1
        Statement2
 endif;
```

Functions

- User the keyword “function” in their declaration

Ex:

```
function hello() {
        ...
}

hello();
```

Functions

- You can return values with “return”
- PHP does not require declaring a return type, but it is possible

Ex:

```
function returnFive(): float {
        $ret = 5.0;
        …
        return $ret
}
```

Function arguments

- By default they are passed by value, but can be passed by reference if prefixed with “&”
- Type declarations are not required, but can be used similarly to other languages

Ex:

```
function append(string &$string1, string $string2, $string3) {
        ...
}
```

Argument lists

- Functions can accept a variable number of arguments using the “...” token
- You can pass as many arguments as you want and use some built in functions to access those

Ex:

```
function varArgs(...$strings) {
        foreach($strings as str) { …}
        $arglist = func_get_args();
        $numArgs = func_num_args();
}
```

Default argument values

- In function definitions, you can set default argument values
- If not all arguments have default values, the ones with default values must come at the end

Ex:

```
function defaultArg($arg1, $arg2=5) {
        ...
}
```

Variable functions

- If a variable name is used with parentheses, it will call a function with the same name as the variable’s value

Ex:

```
function func() { …}

$varFunction = “func”;

$varFunc();
```

Anonymous functions

- Functions created with no given name
- Can be used as callback parameters or with anonymous function variables

Ex:

```
$func = function() {...};
$func();
```

Classes

- Syntactically, class definitions are not too different from Java
- Properties and methods can be public (default), private, or protected

Ex:

```
class Sample {
        private $prvtVar = “default”;
        …

        function method1(...) {...}
        static function staticMethod(...) {...}
}
```

Method Overloading and Overriding

- Method overloading is not allowed in PHP
- You can sort of get around this by taking advantage of variable argument lists
- Method overriding is similar to other languages
- Parent methods can be accessed with the scope resolution operator "::"

Ex:

```
class A {function func() {...}}
class B extends A {function func() {parent::func(); ...}}
```

Accessing properties and methods

- Non-static:
    - “->”
- Static:
    - “::”

Ex:

```
$a = new Class();
echo $a->name;
$a->echoName();
Class::staticMethod()
```

Constructors and Destructors

- Constructors:
  - function __construct(...) {...}
  - Essentially works like constructors in other languages
- Destructors:
  - function __destruct(void) {...}
  - Get called automatically when there are no more references to the object

Inheritance

- Done with the “extends” keyword
- Classes must be defined before they are extended
- Multiple inheritance is not allowed, but multi-level inheritance is

Ex:

```
class Parent {...}

class Child extends Parent {...}
```

Abstract Classes and Interfaces

- Abstract classes:
  - Have at least one abstract method
  - Must be extended with abstract methods overridden
- Interfaces:
  - Can only have abstract methods and class constants
  - Must be implemented with abstract methods overridden
  - Multiple interfaces can be implemented

Ex:

```
interface TestInterface {...}
abstract class TestAbstractClass implements TestInterface {...}
```

Namespaces

- Similar to packages in Java, but uses “\” instead of “.”
  - “\” alone means global space
- A way of encapsulating items
- Useful for avoiding name collisions

Ex:

```
namespace mynamespace\subnamespace;
```

Exceptions

The thrown object must be an instance of the Exception class or a subclass of Exception. Throwing an object that is not will result in a PHP Fatal Error.

Try - If the exception does not trigger, the code will continue as normal. However if the exception triggers, that exception is "thrown"

Throw - Each "throw" must have at least one "catch"

Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Exceptions

```
<?php
function check($number) {
 if($number = 0) {
   throw new Exception("Value Cannot be Zero");
 }
 return true;
}
try {
 checkNum(0);
}
catch(Exception $e) {
 echo   $e->getMessage();
}
?>
```

Error Handling

```
<?php
$file=fopen("File.txt","r");
?>
```

If the file does not exist this:

```
Warning: fopen(File.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\webfolder\test.php on line 10
```

Error Handling

To fix use die method: die()

The die() function prints a message and exits the current script.

To handle potential error:

```
<?php
if(!file_exists("File.txt")) {
 die("File not found");
} else {
 $file=fopen("File.txt","r");
}
?>
```

Predefined Global Variable

Superglobal variables you can access regardless of scope.

$GLOBALS

$_SERVER

$_POST

$_GET

$GLOBALS

Access global variables from anywhere in the PHP script.

Superglobal - is accessible regardless of scope.

```
<?php
$a = 2;
$b = 2;
function addGlob() {
        $GLOBALS['print'] = $GLOBALS['a'] + $GLOBALS['b];
}
addition();
echo $print;
?>
```

$_SERVER

Holds information about headers, paths, and script locations.

$_SERVER[‘PHP_SELF’] - Returns filename of script that is executing

$_SERVER[‘SERVER_ADDR’]- Returns IP address of host server.

$_SERVER[‘SERVER_NAME’]- Returns name of host server.

$_SERVER[‘SERVER_PROTOCOL’] - Returns the name of information protocol

$_POST

Mainly used to collect data after the submission of an HTML form.

HTML File

```
<form action="aFile.php" method="post">

Name:  <input type="text" name="username" /><br />

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

</form>
```

PHP file

```
echo $_POST['username'];
```

$_GET

HTML

```
<form action="file.php" method="get">
<select name="a">
...
<input name="b" type="text" />
```

PHP

```
$a= $_GET['a'];
$b = $_GET['b'];
```

GET Vs POST

Both GET and POST create an array. The array holds key/value pairs, where keys are the names of the form controls and values that are the input data from the user.

Both GET and POST are treated as $_GET and $_POST.
These are superglobals - always accessible, regardless of scope and you can access them from any function, class or file without having to do anything.

$_GET is an array of variables passed to the current script via the URL parameters.
$_POST is an array of variables passed to the current script via the HTTP POST method.

PHP File

fopen() - A method to open files.

```
$myfile = fopen("MyFile.txt", "r")
```

fread()- A method to read from open files.

```
fread($myfile,filesize("MyFile.txt"));
```

fwrite() - A method to write to open files

```
fwrite($myfile,$str);
```

fclose() - A method to close an open file.

```
fclose($myfile);
```

File modes

R mode - Open a file for read only.

W mode - Open a file for write only, Erases file contents or creates new file.

A mode - Writes to existing file, creates file if does not exist.

Supported Protocols/Wrappers

File://: is the default wrapper used with PHP and represents the local filesystem. When a relative path is specified, the path provided will be applied against the current working directory. In many cases this is the directory in which the script resides unless it has been changed.

Http:// and Https://: The HTTP functions let you manipulate information sent to the browser by the Web server, before any other output has been sent.

Ftp:// and Ftps://: The FTP functions give client access to file servers through the File Transfer Protocol (FTP).The FTP functions are used to open, login and close connections, as well as upload, download, rename, delete, and get information on files from file servers. Not all of the FTP functions will work with every server or return the same results.

php://: provides a number of I/O streams that allow access to PHP's own input and output streams.