Table of Contents

Introduction

Origin

PHP was first developed by Rasmus Lerdorf in 1994. Originally, being only a few Perl scripts he used to help manage his Personal Home Page, it was thus dubbed PHP. Later Rasmus rewrote PHP in a much bigger implementation in C. This version was able to communicate with databases, and essentially at the heart, let user create dynamic WebPages, instead of using only static HTML.

Paradigms

At PHP's core it was originally designed as a purely procedural language. Meaning it ran from start to finish picking up assignments and declarations to be used. However, since PHP version 3, it has picked up the object-oriented paradigm. PHP allows for instances of pre-define and user-defined classes to be encapsulated inside of an object. More information on objects in PHP will be covered in the classes section of this paper.

Application Domains

As stated earlier, PHP was originally developed to help manage personal home pages. So it wouldn't be a stretch to say that most if not all of PHP's usefulness lies in web development. PHP was designed in hopes of allowing programmers to develop truly dynamic websites. Some of PHP's hidden power lies within its predefined variables, many of which include server and host information. This information can be used to customize and manipulate pages to user's likings, etc. Additionally, PHP has built-in support for many web technologies, such as MySQL, Apache and more. Many of todays most popular websites, such as Facebook, Wikipedia and Digg use PHP at their core.

Operators

Operator Precedence

Operator precedence in PHP is very similar to many other languages. Parenthesis may be used in order to force a specific order or precedence. If operators are equal in precedence, then they are evaluated from left to right. Below is a list of operators in descending order from highest precedence to lowest precedence.

[, ++ --, !, * / %, + - ., << >>, < <= > >= <>, == != === !==, &, ^, |, &&, ||

Basic Operators

Below is a list of most of the basic operators. Most operators follow the same behavior as many other languages.

-$a - The opposite of $var

$a + $b - Addition

$a - $b - Subtraction

$a * $b - Multiplication

$a / $b - Division

$a % $b - Modulus

$a += 2 - Sets $a equal to $a + 2

$a -= 2 - Sets $a equal to $a – 2

String Operators

$a . $b - Concatenates $b onto the end of $a

$a .= $b - Sets $a equal to $a . $b (as in previous line)

Array Operators

$a + $b - The union of $a and $b

$a == $b - Logical operator for equality of array $a and array $b

$a != $b - Logical not operator for equality of array $a and array $b

Type Operators

The instanceof operator can be used to determine whether an object is of a certain class or not. It is a logical operator and returns a Boolean. It is used like this:

```
class NewClass { }
$a = new NewClass;
print $a instanceof NewClass;
```

This code would return the Boolean value TRUE, since object, $a, is of type NewClass.

Predefined Types

PHP has a total of eight predefined types. Of these eight predefined types there are four scalar types, two compound types, and two special types.

Scalar Types

Boolean – This is the simplest type in PHP. It can have a value of either TRUE or FALSE. The Boolean type is case incentive just like the entire language itself. In PHP, any non-zero value is considered to be TRUE. Values that will evaluate as FALSE include: Integer = 0, Float = 0.0, an empty string, String = "0", NULL, etc.

Integer – PHP has the ability to specify integers using decimal, hexadecimal or octal notation. To use hexadecimal notation you add a "0x" in front of the number, or if you want to use octal notation you add a "0' in front your number. In addition, you can specify positive (+) or negative (-) to any integer value, by adding the symbol to the

beginning of the value. It is also important to note that during any casting, all integer numbers are rounded down.

Float – PHP supports floating-point numbers, also known as “doubles” or “real numbers.” The size and precision of a float is platform-dependent, however a common size is 1.8^308, with 14 decimal precision.

String – To specify a string in PHP, you simply enclose any amount of alpha-numeric characters inside of either single quotes (‘) or double quotes (“). There is no limit to the size of a string type in PHP; the only limit is the available memory on the current machine. It is important to note, that if you enclose your string inside of single quotes, only certain escape characters will be picked up by the engine, where as if you use double quotes, many more escape characters are supported.

Compound Types

Array – Arrays are very dynamic in PHP, and can be treated as an array, a list, a hash table, a dictionary, and more. The simplest way to create an array in PHP is to use the array() function. In the array() function you can choose to specify a key for each value, for example,

```
$arr = array(“key1” => “val1”, “key2” => “val2”, …);
```

However, if you do not wish to specify a unique key for each value in the array, you can just use the array() function like so,

```
$arr = array(“val1”, val2, …);
```

PHP will then automatically assign a numerical value as the key beginning with 0. So if we wanted to access the first element in $arr, we would use $arr[0], and the second element: $arr[1] and so forth. Keys in an array can only be an integer value or a string

value. Even if a string is specified in standard representation of an integer, such as “6”, PHP will then read it at the integer 6, instead of string “6”. Another way of specifying a array type, is to simply use square bracket syntax. Just as if we are trying to access a value in an array, we can assign a value to any location in an array, and an array will automatically be created. For example,

$arr[0] = “This is a string”;

In this instance, a new array will automatically be created with the location at 0, being assigned the value: “This is a string”. There is no need to begin at 0 either, you could start simply by specifying, $arr[20] = “a value here”, and only location 21 in the array will be filled.

Object – To specify an object type, you add new in front of any pre-defined or user-defined class.

Special Types

Resource – A resource is a special variable type in PHP. It basically holds a reference to some external data or resource. There are many functions in the PHP language that return resources. These resources can then be used in certain functions to retrieve data out of. Some examples of resources could be file pointers, database connections, or an image.

NULL – Like in many other languages, NULL represents no value at all in PHP. All variables that have no been set to any other value yet, are initialized as NULL. For example,

$var;

$var2 = “a string”;

In this code segment, $var will have a value of NULL, since it has not been assigned any value yet, where as $var2 will be assigned as a string with value: “a string”. You may also set any already assigned variable to NULL by using the unset() function. NULL in PHP is case-insensitive so can be assigned as null or NULL, etc. Testing for a NULL value can sometimes be important to see if a variable has been set or not. This can be done using the is_null() function.

Control Structures

Basics

Like almost all languages, PHP has control structures for handling: if, else, elseif, while, for, and switch. Most of these control structures are self-explanatory and work in similar fashion as other languages. More unique control structures are provided below:

do-while

A do-while loop is similar to its conter-part, the while loop. The only difference is that the truth expression in a do-while is executed at the end of each iteration instead of at the beginning, such as the case in the while loop. This can be useful in some coding situations, where you want the code inside a loop to be evaluated at least once.

foreach

The foreach loop in PHP gives an easy way for iterating over arrays. A foreach loop is defined like so,

```
foreach ($array as $value) { }
```

In the code above, every element inside of our array called $array will be passed into the $value variable from beginning to end. You can then fill in the code between the curly braces to perform expressions on $value.

include

The include() control operator will open and then evaluate any file specified as an argument. For example, you may wish to evaluate another .php file that contains declarations for useful functions to be used. This can be done very simply:

```
include("functions.php");
```

Once the include statement is evaluated, you may use any functions or variables defined in the functions.php file. However, you cannot reference these functions or variables untill the include statement is reached.

require

The require statement works identically to the include statement, except if the file or resource is not found, then it will produce a fatal error, and exit.

Functions

Definition

In PHP, you define a function simply like so,

```
function tester($arg1, $arg2) {
        echo "This is the tester function.";
        return $return_value;
}
```

In between the curly braces, any valid PHP code may be included. You may even choose to include other function definitions or even class definitions inside of a function. A function in PHP does not need to be defined before it is referenced in code. Essentially, you could put your function definition at the bottom of a .php file, while referencing it at the top.

Function Scoping

All functions in PHP are global. This means that any defined function, may be used in any other .php file once it is declared. Even functions that are defined inside of another function can be accessed anywhere.

Passing by Reference

By default, PHP passes all variables to functions by value. However, you may also choose to pass arguments by reference. This means that the actual reference to the variable will be given to the function instead of just the value being placed inside of a new variable. Therefore, changing the value inside the function will have effect outside of the function on the original variable as well. To do this, you simply add a ampersand (&) in front of the argument. For example,

```
function test(&$byref) { }
```

You may also return a value from a function by reference, by adding & infront of the function name.

```
function &test($byref) { }
```

Classes

Basics

To define a class in PHP, you must specify the “class” keyword, followed by the name of your class. Then inside curly brackets ({ }), you may specify properties and function/ methods. For example let’s create a very basic class,

```
class TestClass {
        public $public_var = “Im public”;
        protected $protected_var = “Im protected”;
```

```
        private $private_var = “Im private”;
        function printVars() {
                echo $this->public_var;
                echo $this->protected_var;
                echo $this->private_var;
        {
}
```

Now that we have a basic test class coded. We need to know how to insantiate a new instance of this TestClass. This is very simple and requires use of the new keyword we learned about earlier. For example, to specify a new object or instance of the TestClass we use:

```
$our_class = new TestClass();
```

This code now creates an instance to TestClass that we can access using our object variable, $our_class. In our class, TestClass, we have one public property and one public function. The public property or variable being, $public_var. This variable can be accessed using $our_class->public_var. We can also access the function inside of our Test Class using the code: $our_class->printVars().

Scoping

Now you are probably wondering what is the difference between public, protected and private in our TestClass. You can use the keyword public, protected, or private in front of any variable or function.

Public – Means you can access this variable/function anywhere in the class or outside of the class.

Protected – You can access this variable/function inside the class or inside any class that inherits this class.

Private – This variable/function may only be accessed inside of this class instance.

Constructors and Destructors

PHP like many other languages supports class constructors as well as class destructors. A class constructor is called upon initiation of a class, and a class destructor is called upon unsetting of a class.

Example:

```
class Person {
        public $person_name, $person_age;
        function __construct($name, $age) {
                $person_name = $name;
                $person_age = $age;
                echo $name . " is " $age . " year(s) old…";
        }
        function __destruct() {
                echo "Destroying " . $name . " now…";
        }
}
```

In the above example, our constructor takes in two arguments: $name and $age. These values are then assigned into our public properties/variables $person_name and $person_age. Whenever any instance of the Person class is destroyed or unset, the __desctruct() function is called and prints out the name of the person. Note it is

impossible to take in arguments or throw an exception in a destructor for obvious reasons.

Extending Classes

In PHP, any class can extend any other class. The child class that extends the parent class will then receive all public and protected functions and variables in the parent class.

Interfaces

Interfaces give you the ability to specify certain functions that an implementing class must implement. All interfaces are defined similarly to classes, except use the interface keyword instead of class. All methods in an interface must be defined as public. For a class to implement a interface, the keyword implement is used followed by the interface name. For example if we wanted to implement the interface called Animal, we would use:

```
class Tiger implements Animal {
        //…
}
```

In this example, our Tiger class would then have to implement all of the methods defined inside of Animal. Interfaces may also define constants. These constants cannot be overridden by the implementing class.

Exception Handling

In PHP, any exception already defined or user-defined may be thrown and/or caught. Once an exception has been thrown, the engine will look for the first available catch block. If no catch block is found, PHP will cause a fatal error with the message, “Uncaught Exception …”. Throwing and catching an error in PHP is very

similar to many other languages, here is a small example:

```
function divide($value, $divideby) {
        if ($divideby == 0) {
                throw new Exception("Division by zero.");
        } else {
                return ($value / $divideby);
        }
}
try {
        divide(18, 0);
} catch (Exception $e) {
        echo "Caught exception with msg: " . $e->getMessage();
}
```

In the above example, we throw our own exception with the message "Division by zero." Upon trying to divide 18 by 0, the engine does not execute any more lines and finds the catch block we have provided printing out the message.

Comments

PHP overall is very similar in syntax to languages such as Perl and C. In my opinion it very easily allows the developer to take simple markup languages like HTML, and maximize use by taking advantage of PHP's objected oriented design. Without the introduction of PHP, it would be hard to imagine many websites that are designed today. PHP truly allows creation of a completely-dynamic and user driven webpage. There really needs to be no static attribute to any standard HTML webpage with PHP. PHP is

also hidden and server-side, meaning it is run before the user has any chance to interact with it. This keeps it relatively secure and powerful at the same time.

Bibliography

Coggeshall, John. "Embedding PHP in HTML." O'Reilly. Retrieved on April 10th from http://www.onlamp.com/pub/a/php/2001/05/03/php_foundations.html.

PHP. "PHP Manual." The PHP Group. Retrieved on April 10th from http://www.php.net/.

Watkins, Thayer. "The Origin and Nature of the Scripting Language and System PHP." Retrieved on April 10th from http://www.applet-magic.com/php.htm.