**PHP**

**UNIT 1:**WELCOME TO PHP!

<body>

<p>*<?php*

*echo "I'm learning PHP!";*

*//* The echo function outputs strings.

*?>*</p>

</body>

The concatenation operator is just a dot (.).

<?php

echo "Hello," . " " . "world" . "!";

// PHP requires **semicolons** at the end of each **statement!**

?>

You might have noticed that our main file is now **index.php** instead of index.html. This is important! It tells the PHP interpreter that there's PHP code in the file to evaluate.

<?php

echo 5 \* 7;

$myName = "Beyonce";

$myAge = 32; // I'm a comment.

// All variable names in PHP start with a dollar sign ( $ ).

?>

**UNIT 2:**CONTROL FLOW: IF/ELSE

List of comparison operators:

* > Greater than
* < Less than
* <= Less than or equal to
* >= Greater than or equal to
* == Equal to
* != Not equal to

<?php

$name = "Edgar";

if ($name == "Simon") {

print "I know you!";

}

else {

print "Who are you?";

} ?>

**UNIT 3:** CONTROL FLOW: SWITCH

$myNum = 2;

switch ($myNum) {

case 1:

echo "1";

break;

// use break to exit the switch statement

// otherwise, the next case block runs

case 2:

echo "2";

break;

default:

echo "None of the above";

}

This is called falling through:

case 1:

case 2:

case 3:

echo '$i is somewhere between 1 and 3.';

break;

Alternative syntax:

switch ($i):

…

endswitch;

**UNIT 4:** ARRAYS

$array = array("Egg", "Tomato", "Beans" );

echo $tens[2]; // => “Beans”

// we can also use: echo $tens{2};

$tens[2] = “Apple”;

echo $tens{2}; // => “Apple”

Finally, we can remove elements using unset:

$array = array("red", "blue", "green");

unset($array[2]);

We can even delete the whole array:

unset($array);

**UNIT 5:** LOOPS: FOR AND FOREACH

<?php

for ($i = 0; $i < 10; $i++) {

echo $i;

}

// echoes 0123456789

?>

The foreach loop is used to iterate over each element of an object—which makes it perfect for use with arrays!

$langs = array("JavaScript",

"HTML/CSS", "PHP",

"Python", "Ruby");

foreach ($langs as $lang) {

echo "<li>$lang</li>";

}

**UNIT 6:** LOOPS: WHILE AND DO-WHILE

while(cond) {

// looped statements go here

}

Alternative syntax:

while(cond):

// looped statements go here

endwhile;

Do/while loop:

$i = 0;

do {

echo $i;

} while ($i > 0);

**UNIT 7:** FUNCTIONS IN PHP, PART I

**strlen()** returns the number of characters in string:

$length = strlen("david");

print $length; // => 5

**substr()** allows to return a substring (piece of) of string:

$myname = "David";

$partial = substr($myname, 0, 3);

print $partial; // prints "dav"

**strtoupper()** and **strtolower()** make entire string UPPERCASE or lowercase:

$uppercase = strtoupper($myname);

print $uppercase; // prints "DAVID"

$lowercase = strtolower($uppercase);

print $lowercase; // prints "david"

**strpos()** find the position of the first occurrence of a substring in a string.

strpos("emily", "e"); // 0

strpos("emily", "i"); // 2

strpos("emily", "ily"); // 2

strpos("emily", "zxc"); // false

**round()** rounds floating point numbers (numbers with decimal points in them):

// Round pi down from 3.1416...

$round = round(M\_PI);

print $round; // prints 3

// This time, round pi to 4 places

$round\_decimal = round(M\_PI, 4);

print $round\_decimal; // prints 3.1416

NOTE: M\_PI is a PHP constant that is equal to pi.

**rand()** returns a random number between two numbers:

// prints a number between 0 and 32767

print rand();

// prints a number between 1 and 10

print rand(1,10);

**array\_push()** takes two arguments: an array, and an element to add to the end of that array:

$fav\_bands = array();

array\_push($fav\_bands, "Maroon 5");

array\_push($fav\_bands, "Bruno Mars");

**count()** will return the number of elements in that array:

print count($fav\_bands); // prints 2

**sort()**

$array = array(5, 3, 7, 1);

sort($array);

print join(", ", $array); // prints "1, 3, 5, 7"

**rsort()**

$array = array(5, 3, 7 ,1);

rsort($array);

print join(":", $array); // prints "7:5:3:1"

Lastly, we use **join(glue, array)** so we can easily print out the representations of our sorted arrays in this exercise. For this exercise, we'll use a comma (,) as the glue.

**UNIT 8:** FUNCTIONS IN PHP, PART II

The typical structure of a function is as follows:

function name(parameters) {

statement;

}

1. The keyword function indicates that the code following it will be a user-defined function.
2. name indicates the name of a function, which is case insensitive. The name of a function can contain numbers, letters, underscores or dashes.
3. The arguments, or parameters, will be the optional input a function uses to perform calculations.
4. And of course, the statements themselves will be the code the function executes when it is called.

**UNIT 9:** OBJECT-ORIENTED PROGRAMMING, PART I

The basic class syntax looks like the following:

class Classname {

…

}

And we can create new instances of this class using the following syntax:

$obj1 = new Classname();

class Fruit {

public $count = 3;

public $type;

}

$apple = new Fruit();

$apple->type = "apple";

print $apple->count; // 3

print $apple->type; // apple

1. In the example above, we first create a new class called Fruit.
2. Then we add a property, $count, and set its value to 3.
3. Next, we add a property, $type, but don't store anything in it yet.
4. After the class definition, we create new instance of Fruit and store it in $apple.
5. Then we set the $type property of $apple to the string "apple".
6. Finally, we print out the two properties of $apple.

We have to create a constructor to create different objects. This constructor is also a method, but you don't need to worry about this fact just yet.

public function \_\_construct($prop1, $prop2) {

$this->prop1 = $prop1;

$this->prop2 = $prop2;

}

1. We are creating a function bound to a class (a method).
2. The constructor method has to be called \_\_construct().
3. Finally, the weird way to assign the values: $this->prop1 = $prop1 means that the value we pass in the \_\_construct() function via the new keyword is assigned to $this, which represents the object we are dealing with, and ->prop1 is the actual property of the object.
4. By creating a new instance using the new keyword, we actually call this \_\_construct() method, which constructs the object. And that's why we have to pass in some arguments when we create an instance of a class, since this is how the properties get set!

Methods—functions bundled into objects—have the following syntax:

public function funcname($optionalParameter) {

// Do something

}

If we want a method to return a sentence containing the firstname, we would have to use $this->firstname. (As you see, there is no $ when you access a property in a class.)

$obj1 -> meth1();

Class example

<?php

// The code below creates the class

class Person {

// Creating some properties (variables tied to an object)

public $isAlive = true;

public $firstname;

public $lastname;

public $age;

// Assigning the values

public function \_\_construct($firstname, $lastname, $age) {

$this->firstname = $firstname;

$this->lastname = $lastname;

$this->age = $age;

}

// Creating a method (function tied to an object)

public function greet() {

return "Hello, my name is " . $this->firstname . " " . $this->lastname . ". Nice to meet you! :-)";

}

}

// Creating a new person called "boring 12345", who is 12345 years old ;-)

$me = new Person('boring', '12345', 12345);

// Printing out, what the greet method returns

echo $me->greet();

?>

**UNIT 10:** OBJECT-ORIENTED PROGRAMMING, PART II

**is\_a()** we use to find out if a particular object is an instance of a given class; **property\_exists()**, to see if an object has a given property; and **method\_exists()**, to see if an object has a given method.

class Person {

public $isAlive = true;

function \_\_construct($name) {

$this->name = $name;

}

public function dance() {

return "I'm dancing!";

}

}

$me = new Person("Shane");

if (is\_a($me, "Person")) {

echo "I'm a person, ";

}

if (property\_exists($me, "name")) {

echo "I have a name, ";

}

if (method\_exists($me, "dance")) {

echo "and I know how to dance!";

}

Inheritance

class Shape {

public $hasSides = true;

}

class Square extends Shape { }

}

Sometimes we want a child class (or **subclass**) to be able to override a property or method of its parent class (or **superclass**).

class Shape {

$sides = true;

}

class Square extends Shape {

$sides = 4;

}

In PHP, a parent class can prevent its methods from being overridden by its children with the **final** keyword.

class Vehicle {

final public function drive() {

return "I'm drivin' here!";

}

}

We've talked a lot about changing variables in PHP, but sometimes we want variables that don't change. These are prefixed with the **const** keyword (short for constant).

PHP lets us set constants on a class-by-class basis! Each class has its own scope, which is the context in which its variables can be used.

class Immortal extends Person {

// Immortals never die!

const alive = true;

}

// If true...

if (Immortal::alive) {

echo "I live forever!";

}

// echoes "I live forever!"

In the example above, we use :: to access the alive constant inside the Immortal class.

Note that constants do not start with $.

The static keyword lets you use a class' property or method without having to create an instance of that class. It works like this:

class Person {

public static $isAlive = "Yep!"

public static function greet() {

echo "Hello there!";

}

}

echo Person::$isAlive;

// prints "Yep!"

Person::greet();

// prints "Hello there!"

**UNIT 11:** ADVANCED ARRAYS AND MAPS

An **associative array** (also called a **map**) makes use of (key => value) pairs.

$myAssocArray = array('year' => 2012,

'colour' => 'blue',

'doors' => 5);

echo $myAssocArray['colour']; // “blue”

Loop through an associative array using "foreach":

$salad = array('lettuce' => 'with',

'tomato' => 'without',

'onions' => 'with');

foreach ($salad as $ingredient=>$include) {

echo $include . ' ' . $ingredient . '<br />';

}

Multidimensional Array:

$deck = array(array('2 of Diamonds', 2),

array('5 of Diamonds', 5),

array('7 of Diamonds', 7),

array('8 of Diamonds', 8);

If we want the "7 of Diamonds" string, we would simply use $deck[2][0];