# PHP: Hypertext Preprocessor

PHP is a language that was originally designed for web development, but can be used for any number of applications (though we'll be sticking primarily to web development). PHP stands for "PHP: Hypertext Preprocessor" as a recursive acronym.

## Installation:

From the previous lesson on MySQL, you should have already downloaded either **XAMPP** or **MAMP**. If you haven't, please go back to that lesson and follow the download instructions.

### Syntax:

PHP alone looks relatively similar to most languages. To do a basic hello world:

```
echo 'Hello World';
```

This code will print "Hello World" to whatever is calling it (say a command line, or a web page, etc). First thing about the code to note: **PHP REQUIRES SEMICOLONS**. The javascript we've worked with didn't require semicolons (but you could have added them if you wanted to). PHP does require semicolons.

PHP can be rendered in HTML by wrapping the PHP code in an opening tag: `<?php` and a closing tag: `?>`. Additionally, depending on how you wish the code to be rendered, you can write stringified html code in the php that is meant to be rendered (notice the `<p>` tags surrounding the hello world):

```
<!DOCTYPE html>
<html>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Before we get into more difficult syntax, let's review some of the basics.

## Variables:

Just like every other language we've worked with we need a way to store information in memory. To declare a variable in PHP we just need to write:

```php
$variableName = "some variable value";
```

Again, it must be noted that **PHP REQUIRES SEMICOLONS**.

Let's take a look at some variables in actions:

```php
$hi = "Hello ";
$planet = "World ";

echo "$hi $planet";
```

What is nice about PHP is that in the echo (print) statement, we can print our variables directly without having to use any special syntax.

One thing to note: just like many other languages, you cannot declare a variable that starts with numbers:

```php
$4hellos = "Hello Hello Hello Hello";
echo $4hellos
```

## Data Types

Datatypes in PHP are very similar to that of Javascript in that they don't really exist. If you were to write:

```php
$varName = "Hello";
echo $varName;

$varName = 52;
echo $varName;
```

With that out of the way, primitive data types that are supported in other languages are also supported in PHP:

```php
$imAnInteger = 5;
$imAFloatingPoint = 3.14;
$imAString = "Strings are neat!";
```

```
$imABoolean = TRUE;
$imAlsoABoolean = FALSE;


echo "Integers are numbers with no decimal points, like $imAnInteger \n\n";
echo "Floating point numbers have decimals, like pi: $imAFloatingPoint \n\n";
echo "We've already seen strings, but it's good to see them again: $imAString \n\n";
echo "Booleans are either true or false, and that's it! So, $imABoolean and $imAlsoABoolean are the only two possible options! \n\n";


$what = $imAnInteger / 3;
echo "Five divided by three = $what";
```

Did you notice how imABoolean and imAlsoABoolean printed out? In many languages true and false are 1 and 0, in php, true is still 1, but false is just nothing (or 'null') .

### CONSTANTS:

It's not always the case, but sometimes you may need to define a constant. To do that, you'll need to use the `define()` function:

```
define(constantName, constantValue, caseInsensitivityBoolean);
```

So, suppose that we wished to make a constant called PI. We'll define its case insensitivity as true, so that when accessing it, you can only access the data as `PI`, and not `pi`:

```
define('PI', 3.14, true);
echo PI;

// The below will fail:
//echo pi;
```

If we wanted to have a case insensitive `pi`, we can write:

```
define('PI', 3.14, false);
echo pi;
echo PI;
```

## Operators

Most of the traditional operators found in other languages are also found in PHP:

## Maths:

The mathematical operators traditionally found in other languages are also found in PHP. Addition `+`, subtraction `-`, multiplication `*`, division `\`, modulus `%`, and exponents `**` all act the same.

```php
$number = 3;
$otherNumber = 2;

$sumOfTwoNumbers = $number + $otherNumber;
$differenceOfTwoNumbers = $number - $otherNumber;
$productOfTwoNumbers = $number * $otherNumber;
$quotientOfTwoNumbers = $number / $otherNumber;
$remainderOfTwoNumbers = $number % $otherNumber;
$powerOfTwoNumbers = $number ** $otherNumber;



echo "The sum of $number + $otherNumber is $sumOfTwoNumbers \n";
echo "The difference of $number + $otherNumber is $sumOfTwoNumbers \n";
echo "The product of $number + $otherNumber is $sumOfTwoNumbers \n";
echo "The quotient of $number + $otherNumber is $sumOfTwoNumbers \n";
echo "The remainder of $number + $otherNumber is $sumOfTwoNumbers \n";
echo "$number raised to the $otherNumber is $sumOfTwoNumbers \n";


echo "Math is neat :) ";
```

Additionally, increment and decrement operators work the same. If you attempt to pre-decrement/increment, you'll need to include the operators prior to the `$`:

```php
$one = 1;
$two = 2;

echo "Variable one = $one, and variable two = $two \n";

$one--;
$two++;

echo "Now, variable one = $one, and variable two = $two \n";

--$one;
++$two;

```

```php
echo "Now, variable one = $one, and variable two = $two \n";
```

## String Manipulation:

Working with strings in PHP is critical. One of the biggest differences in PHP is string concatenation. In many other languages you've already worked with, string concatenation worked such as `"hello " + "world" = "hello world"`. PHP concatenates with a `.`. To concatenate two sentences in PHP

```php
$string1 = "Friendship";
$string2 = "Is really neat!";

// Unlike javascript, you can't add strings together like this:
// $bothString1AndString2 = $strin1 + $string2;

$bothString1AndString2 = $string1.$string2;

echo $bothString1AndString2;
echo "\n\n\n";
```

In order to check the length of a string, you'll need to use `strlen` (as opposed to the `.length` operator in javascript):

```php
echo "Length of string one is: ";
echo strlen($string1);
echo "\n";

echo "Length of string two is: ";
echo strlen($string2);
echo "\n";

echo "Length of both is: ";
echo strlen($bothString1AndString2);
echo "\n";
```

There are times when you'll need to slice a string apart. In order to do that, you can slice it based on a given delimiter, that is, you can choose how to break up a string:

```php
$stringWithSpaces = "one two three four five six";

// the " " in the explode function is what to separate on!!
$separatedString = explode(" ", $stringWithSpaces);


echo "Our regular string is: $stringWithSpaces \n";
echo "When we separated our string, we separated it on spaces, so now every time we saw a space, we separated it to its own elements:  $separatedString \n\n\n";

for ($i = 0; $i < 5; $i++){
  echo $separatedString[$i];
  echo "\n";
}
```

It is also possible to collapse an array into a string:

```php
$newStringFromArray = implode(" ", $separatedString);
echo $newStringFromArray;
```

## Functions

Functions are an absolute necessity in programming. Functions allow for us to write code just once and reuse it constantly (as opposed to writing it over and over again). Functions in PHP follow a standard formula by declaring the function with the keyword `function`, followed by the funciton name, and a parameter list and a code block:

```php
function funcName($parameter){
  echo $parameter;
}
```

Functions can take parameters. As a note, remember that because parameters are also variables, they need to follow the same rules for syntax and naming:

```php
function partyHard($name){
  echo "Party on $name\n";
}

$wayne = "Wayne";
$garth = "Garth";
```

Functions can also return values (and we don't need to declare a return type):

```php
function getStuff(){
  return "STUFF for the party";
}

$stuffsIGot = getStuff();
$otherSTuffs = getStuff();

echo $stuffsIGot."\n";
echo $otherSTuffs."\n";
```

## Control Flow

### If/Else:

If/Else statements control how a program operates. PHP's if/else statements are almost exactly the same as javascript, for example:

```php
$num = 5;

if ($num > 3) {
  echo "$num is greater than 3";
} else {
  echo "$num is not greater than three";
}
```

What makes if/else statements different from those of javascript and other c-style languages is the elseif statement (i.e. elseif is one word, and not two):

```php
$num = 5;

if ($num > 10) {
  echo "$num is greater than 10";
} elseif ($num > 3) {
  echo "$num is greater than three";
} else {
  echo "$num is not greater than 3 or 10";
}
```

## Switch:

Switch statements work just the same as well. Notice there's no break between `case 'skittles'` and `default`. PHP does allow for fallthrough:

```php
$bestCandy = "skittles";

switch ($bestCandy) {
    case "payday":
        echo "who doesn't love so many peanuts!!";
        break;
    case "twizzlers":
        echo "Pull and peel are the best";
        break;
    case "skittles":
        echo "TASTE THE RAINBOW!";
    default:
        echo "DEFAULT!";
}
```

## Loops:

Loops exist just the same in PHP as other languages.

```
$x = 0;

echo "While loop: ";
while($x <= 5) {
    echo "$x ";
    $x++;
}

echo "\n For loop: ";

for($i = 0; i < $x; $i++){
  echo "$i ";
}
```

## Advanced Data Structures

### Arrays

There are multiple ways to create arrays in PHP. You can do it explicitly with the `array` keyword. To access the elements, the `[]` operators are used:

```
$beastieBoys = array("MikeD", "MCA", "Ad-Rock");
echo "Now here's a little story I've got to tell
About three bad brothers you know so well
It started way back in history
With $beastieBoys[2], $beastieBoys[1] and me $beastieBoys[0]";
```

You can also create arrays implicitly by:

```
$myArray = [1,2,3,4,5];
```

In order to work with arrays, you often need to loop over them. To do so, you'll need to know the length of the array, and since many arrays are of differing sizes, we'll need a way to determine the size of the array. To do that, you'll need to use the `sizeof()` method:

```
$myArray = [1,2,3,4,5];
echo sizeof($myArray);
```

### Arrays and Functions:

Functions can take arrays just like any other language, however, array data are not passed by reference automatically:

```php
 1  function workWithArray($arr){
 2    for($i = 0; $i < 5; $i++){
 3      $arr[$i] = "DUMB";
 4    }
 5    return $arr;
 6  }
 7
 8  function printArray($arr){
 9    for($i = 0; $i < 5; $i++){
10      echo $arr[$i]."\n";
11    }
12  }
13
14  $myArray = [1,2,3,4,5];
15  printArray($myArray);
16
17  $newArray = workWithArray($myArray);
18  echo "the original array is: ";
19  printArray($myArray);
20
21  echo "the newly returned array is: ";
22  printArray($newArray);
```

At line 19, our original array is printed, and it is the exact same as when it was declared. Only by printing the returned array on line 22 do we see the changed data. To pass an array by reference, you'd need to place an ampersand in front of the parameter in the function declaration:

```php
 1  function workWithReferencedArray(& $arr){
 2    for($i = 0; $i < 5; $i++){
 3      $arr[$i] = "DUMB";
 4    }
 5    return $arr;
 6  }
 7
 8  function printArray($arr){
 9    for($i = 0; $i < 5; $i++){
10      echo $arr[$i]."\n";
11    }
12  }
13
14  $myArray = [1,2,3,4,5];
15  printArray($myArray);
```

```php

$newArray = workWithReferencedArray($myArray);
echo "the original array is: ";
printArray($myArray);

echo "the newly returned array is: ";
printArray($newArray);
```

## Dictionaries:

Arrays access data in a variable by using an indexed number. Dictionaries do the same thing, however, instead of using a number, they use what is called a `key`:

```php
$dictionaryExample["someKey"] = "Some Value";
echo $dictionaryExample["someKey"]."\n";

$dictionaryExample["anotherKey"] = 52;
echo $dictionaryExample["anotherKey"]."\n";
```

When accessing data from databases, the data are returned from the database as key value pairings with the key being the column names (we'll see more of how to do this in the next lecture)!

## Classes / Objects

Classes in PHP are declared with the `class` , and then the title of what the class is:

```php
class ObjectName {
  function ObjectName() {
      // Constructor bits.
  }
}
```

Inside the constructor is where class variables are defined. Objects are created by calling a class's constructor such as:

```php
$newObject = ObjectName();
```

To access class variables you use `$objectName->classVariable`, and to access methods `$objectName->classMethod()`. When accessing class variables from within the class, however, you'll need to use the keyword `$this`. So, to access the `classVariable` from within the class (like, in the constructor), you would need to write `$this->classVariable`.

For example, creating a class called spaceship with a constructor that creates a model, name and version for the ship. Additionally, the class has a class function called printData, which prints all of the class data:

```php
class Spaceship {
    function Spaceship() {
        $this->model = "Galaxy-class starship";
        $this->name = "Enterprise";
        $this->version ="D";
    }

    function printData() {
        echo "This $this->model is the $this->name - $this->version";
    }
}

$ncc1701 = new Spaceship();

echo $ncc1701->printData();
echo "\nThis  $ncc1701->model  is the   $ncc1701->name - $ncc1701->version ";
```