# Intro to PHP

Alex S.[*]

## 1 Introduction

PHP Hypertext Preprocessor is just a language like any other. Learning it is no different than learning C, or Java, etc. This document will describe the basic PHP syntax.

## 2 Comments

PHP comments are basically exactly the same as C++ comments. You can use either:

```
// some comment here
```

Or

```
/* some comment here */
```

They work exactly like in C++; one is for single line comments and another for multi-line comments.

## 3 Variables

PHP variables are sort of similar to Perl variables. They start with a `$` and are virtually type-free. You also don't need to declare variables before using them. So you could for example say:

```
$blah = 7;
```

Or

```
$blah = "Hello World";
```

And PHP would just figure out what value `$blah` has.

[*] alex@theparticle.com

## 4 Conditional Statements

PHP has the usual conditional statements, like `if`, `switch`, etc. They have a pretty obvious syntax, like:

```
if($a < 5){
    // ... stuff goes here...
}
```

Or

```
switch($a){
    case 1:
        // do stuff
        break;
    case 2:
        // do stuff
        break;
    default:
        // do stuff
}
```

There are also one-liners

```
$a = $b < 0 ? -$b : $b;
```

## 5 Loops

There are the general purpose `for` and `while` loops, with obvious formats. Ie:

```
for($i=0;$i<100;$i++){
    print $i."\n";
}
```

And:

```
$i=0;
while($i<100){
    print $i."\n";
    $i++;
}
```

## 6 Strings

Strings act just like you'd expect strings to act. In other words, when you do:

```
$blah = "Hello World";
```

You get a string. To concatenate strings you can use the "dot" operator:

```
$blah = "Hello"." "."World".$a." ".$blah;
```

## 7 Arrays

One of the most useful data structures is a plain array. Unfortunately the way PHP treats arrays is a bit confusing—until you're used to it.

### 7.1 Creating

You create arrays via the `array()` keyword. For example, to create an array of values, you can do:

```
$arr = array("gah","beh","d0h","g0h",1,3.14);
```

You can also declare an empty array via:

```
$arr = array();
```

To access array elements, you can just do:

```
$a = $arr[1];
```

Arrays indexes start at 0 (more on this later).

### 7.2 Appending

You can just set values in the array by assigning them. For example:

```
$arr[100] = "blah";
```

It doesn't matter if the array isn't that big (it will grow).
Sometimes it is useful to "append" (or "push") a value onto the end of the array. You can do that by:

```
$arr[] = "glah";
```

## 7.3 Hashes

It turns out that in PHP, arrays and hashes (maps) are the same. So for example, you can use integer indexes to access values, but you can also use string keys. For example:

```
$arr["name"] = "John Doe";
```

How does this work exactly? Well, arrays in PHP are just hashes. So when you use an integer index, that number becomes the ‘key’.

To create a ‘hash’, you can also use the `array` keyword:

```
$arr = array("name" => "John Doe", "age" => 32);
```

## 7.4 Working with Arrays

There are basically three array functions in PHP, they are:

1. `list()`
2. `each()`
3. `count()`

### 7.4.1 `list()`

The `list()` function presents an “lvalue” of an array—meaning that you can assign things to it. So for example, let’s say you have an array:

```
$arr = array("gah","beh","d0h","g0h",1,3.14);
```

Then you can assign it to values via:

```
list($a,$b,$c) = $arr;
```

The values `$a`, `$b`, `$c` will grab the first, second, and third values, respectively. This comes in handy with the use of `each()`.

### 7.4.2 `each()`

The function `each()` returns `key => value` pairs. So for example, even any array, we can print out its contents via:

```
while(list($key,$value) = each($arr)){
    print("$key => $value\n");
}
```

There’s also the `reset()` function that resets the internal counter used by `each()`. So for example, if you wanted to start to loop through the list once again, you should call `reset($arr)`, which would put you right at the beginning of the array.

### 7.4.3 count()

The `count()` does the obvious thing: it counts the number of elements in the array. This may seem trivial, but you must remember that we are talking about associative arrays—where indexes may not be numeric.

For example, imagine you have an array like:

```
$arr = array(0=>"gah",2=>"beh",4=>"d0h",100=>"g0h");
```

If you run `count($arr)`, it will return a 4. However, you cannot just loop through the array using a `for` loop like:

```
for($i=0;$i<count($arr);$i++){
    print($arr[$i]."\n");
}
```

The above will not work because `$i` will get values: 0, 1, 2, and 3, which aren't the proper indexes for that array.

## 8 Functions

PHP has functions, which are basically just chunks of code that you can execute whenever you want. The general format is something like this:

```
function blah(){
    // do stuff here.
}
```

Functions can also take parameters and return values, i.e.:

```
function blah($a,$b,$c){
    // do stuff here, using $a,$b,$c.
    return $glah;
}
```

One of the more useful features is defaults. You can specify a default value for some parameter if nothing is specified. For example:

```
function blah($a = 0.5,$b = 100,$c = 5){
    // do stuff here, using $a,$b,$c.
    return $glah;
}
```

Now if someone just calls `blah()`, the parameter values will be set to those defaults.

## 8.1 Call by Reference

You can also pass parameters by reference as opposed to by value. You just add an `&` in front of their name. For example:

```
function blah(&$a,$b,$c){
    $a = $b + $c;
}
```

The above will change the value of `$a`, as far as the caller is concerned.

## 8.2 `global`

Whenever you create variables in a function, they're local to that function. So for example, if you have code like the following:

```
$a = 3.14;
function blah(){
    print $a."\n";
}
```

The value of `$a` in the function will be undefined. It's a new variable (as far as the function is concerned). If you want to access the 'global' variable, you need to specify that before you use it in the function. I.e:

```
$a = 3.14;
function blah(){
    global $a;
    print $a."\n";
}
```

This comes in handy in PHP, since your script gets a lot of parameters from the user via the global variables, such as `$_GET`, `$_POST`, `$_REQUEST`, `$_SESSION`, etc.

## 8.3 Variable Arguments

Your functions can take any number of parameters. You can use the:

```
$arr = func_get_args();
```

To retrieve an array of arguments your function got. For example,

```
function blah(){
    $arr = func_get_args();
    for($i=0;$i<count($arr);$i++){
        print($i.": ".$arr[$i]."\n");
    }
}
```

If you call it via: `blah(1,2,"g0h")`, it will display all the arguments (and if you call it with a different number of arguments, it will display all of those).

## 9 Objects

PHP has objects, which are basically enclosures or structures, that can have "methods". To define a class, you just do something like the following:

```
class User {
    var $id;
    var $username;
    var $password;
    var $name;
    var $email;
}
```

You also generally include some functions, like a constructor, or some other functions:

```
class User {
    var $id;
    var $username;
    var $password;
    var $name;
    var $email;

    function User ($arr){
        $this->id=$arr['id'];
        $this->username=$arr['username'];
        $this->password=$arr['password'];
        $this->name=$arr['name'];
        $this->email=$arr['email'];
    }

    function getEmailLink(){
        return "<a href='mailto:".$this->email."'>".$this->name."</a>";
    }
}
```

The constructor has the same name as the class, and obviously you can have other functions, etc.

You create 'new' objects via the 'new' operator:

```
$user = new User();
```

You can also pass an array to set the class attributes to some particular value. You can always access the values directly, via:

```
print $user->name;
```

Or call a function via:

```
echo( $user->getEmailLink() );
```

## 9.1 Inheritance

You can `extend` classes to accomplish “inheritance”.

```
class SomeUser extends User {
    function getEmail(){
        return $this->email;
    }
}
```

Now you have a new class, `SomeUser`, that has all the methods that `User` has, as well as an additional method.

## 10 Meta Information

PHP lets you find information about the program itself. There are many useful functions that you should read up on:

- `get_class($object)`: returns the name of the class of the object as string.
- `get_parent_class($object)`: returns the name of parent class.
- `method_exists($object,$method)`: returns true/false if a method exists.
- `class_exists($class)`: returns true/false if a class exists.
- `is_subclass_of($object,$class)`: returns true/false if object is subclass of class.

## 11 Including Files

You are encouraged to modularize your PHP code into separate files. You can include files in other files via the `include` command. Ie:

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

## 12 PHP Superglobals

This section is taken right out of the PHP documentation.

- `$GLOBALS`: Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.
- `$_SERVER`: Variables set by the web server or otherwise directly related to the execution environment of the current script.
- `$_GET`: Variables provided to the script via URL query string.
- `$_POST`: Variables provided to the script via HTTP POST.
- `$_COOKIE`: Variables provided to the script via HTTP cookies.
- `$_FILES`: Variables provided to the script via HTTP post file uploads.
- `$_ENV`: Variables provided to the script via the environment.
- `$_REQUEST`: Variables provided to the script via the GET, POST, and COOKIE input mechanisms, and which therefore cannot be trusted.
- `$_SESSION`: Variables which are currently registered to a script's session.

## 13 Conclusion

Well, that's about all there is to PHP, language wise.