The PHP Language

CS106E, Young

In this handout, we take a closer look at the PHP language. You may find the most efficient way to use this handout is to skim through it, getting a good sense of what's contained within, and then look things up here as needed as you write your PHP programs for our homework assignments.

Variables

Variables in PHP are untyped and are not declared before use. Variable are indicated with a dollar sign '$' followed by a legal PHP identifier. The rules for PHP identifiers (which can be used for variables, function names, or class names) are:

- They must begin with either a letter or the underscore '_'.
- After the initial letter or underscore, they may be followed by any combinations of letters, numbers, and underscores.

Here are some sample variable examples:[1]

```
$x = 12.7;
$sales_tax = $amount * 0.1;
$salesTax = 15;
```

Variable names starting with an underscore usually indicate system defined variables such as the $_POST and $_GET variables we saw in the previous handout.

In PHP, variable names are case-sensitive so $Example and $example are different variables. However, keywords, functions, and method names are case-insensitive. This mixing of case-sensitive and case-insensitive rules in the same language is quite unusual.

Data Types

PHP supports the following data types (in this class, we will not be covering the types written in italics):

Scalar Types

- boolean
- integer
- float
- string

[1] If we have a name composed of several words such as "Tax Rate", PHP does not have a preference for using underscores or camel case for identifiers – either "tax_rate" or taxRate are fine. If you're not familiar with the term camel case, it refers to creating identifiers from multi-word strings such as "Stanford computer science" by getting rid of the spaces and capitalizing all words past the first as in "stanfordComputerScience". It's called camel case as the capital letters form humps like a camel's back.

Compound Types

- array
- object
- *callable*
- *iterable*

Special Types

- *resource*
- NULL

Boolean

The two boolean literals are TRUE and FALSE. While commonly written in all-caps, they are actually case-insensitive – e.g., you may write TRUE, True, true, or even TrUe.

The integer 0, the float 0.0, the empty string "", empty arrays, and NULL all convert automatically to FALSE. Everything else is considered TRUE.

Boolean Operators

Boolean operators include:

and or **&&** — You may use either the word "and" or the double ampersand for a boolean "and" operator in PHP.[2] Example:

```
if(($age > 18) and ($age < 65)) { … }
```

or

```
if(($age > 18) && ($age < 65)) { … }
```

or or **||** — You may use either the word "or" or use two vertical bars '|' for a boolean "or" operator in PHP

! – The exclamation point acts as a boolean "not" operation. In contrast to "and" and "or" you may not simply write the word "not". Example:

```
if(!($age >= 65)) { … }
```

Comparison Operators

Comparison operators include >, >=, <, <=.

The double equals "==" is used to compare if two values are the same. The exclamation equals "!=" is used to see if two values are not the same. [3] For example:

[2] The && and the "`and`" are almost identical, however, they have different operator precedence. Same for || and the "`or`" operator.

[3] You may also run into a triple equals "===". This operates similarly to "==" but does not allow type conversion. So if the two items being compared are of different types they are automatically not equals. There is a "!==" equivalent for not equals.

```
    if(($university == "Stanford") { … }

    if(($university != "Stanford") { … }
```

PHP also allows “<>” for not equals. This works exactly the same as “!=”.

Integers and Floats

These work pretty much exactly as you’d expect. A few points that may be worth mentioning:

- PHP automatically converts Integers to Floats when they get too big or too small.
- PHP floats are based on the IEEE double precision floating-point standard and are the equivalent of the C, C++, or Java double data type.
- PHP floats support a special value NaN, which stands for “Not a Number”.

Arithmetic operators include:

+ for addition
- for subtraction
* for multiplication
/ for division
% for modulus

There is also a ** operator for exponent. There is also a function `pow` (short for power) which may be used if you’re on an older PHP server that does not support **. Unfortunately, Stanford’s PHP installation is version 5.4 and the exponent operator was added in version 5.6.

In addition, PHP supports C-style increment and decrement operators.

```
    $x++;
```

increments the value of the variable $x by one and

```
    $x--;
```

decrements the value of the variable $x by one.

Strings

PHP Strings are based on 8-bit characters and do not support Unicode. PHP allows use of either single quotes or double quotes when writing string literals, but their mechanics work a bit differently.

Single-Quoted Strings

In a single-quoted string, the only escape sequences supported are \’ for a single quote and \\ for an actual backslash. Other common conventions such as using \t to create a tab character or \n for creating a carriage return (or *newline*) in the middle of the string do not work.

Double-Quoted Strings

With double quotes, standard C/C++/Java escape sequences such as \n for carriage return and \t for tab are supported. PHP double-quoted strings have the rather unusual property that they can actually span multiple lines like this:

```
$myString = "a good
example
string!";
```

In addition, variables that appear in a double-quoted string will be replaced by the variable's value:

```
$university = "Stanford";

$output = "I go to $university";
```

$output will be set to the string "I go to Stanford". You may find it useful to enclose your variable names with curly braces to prevent ambiguity, so you may see the above example written as:[4]

```
$output = "I go to ${university}";
```

Since the $ indicates a variable, if you actually want to place a dollar sign in your string, you should escape it with a slash as in "\$".

String Operators

The concatenation operator is represented by the period '.'. We saw this in our form example in the previous handout:

```
echo "<p>Tax on " . $amount . " is " . $tax . "</p>";
```

Here we form a new string by concatenating together different strings and numbers together. This can be written more compactly using double-quoted strings' variable substitution:

```
echo "<p>Tax on $amount is $tax</p>";
```

If we actually wanted dollar signs to appear in the output (e.g. "Tax on $100 is $10") we would need to use the escape sequence "\$" to indicate that the dollar sign really does represent a dollar sign and not the start of a variable.

```
echo "<p>Tax on \$$amount is \$$tax</p>";
```

Arrays in PHP

The Arrays in PHP aren't actually what we typically think of as arrays. Instead, they are a different data structure that is variously referred to as a Map, an Associative Array, or a Dictionary. An Associative Array allows us to map from one set of values to a different set of values. For example:

[4] PHP will search for the longest matching string name, so if you have a variable named $temp and try to do something like this, "$temporary", it won't find the variable $temp followed by "orary", instead it will look for the non-existent "$temporary" variable. This should be rewritten as "${temp}orary" if you want PHP to find $temp.

```
$capitals = [
    "California" => "Sacramento",
    "Oregon" => "Salem",
    "Washington" => "Olympia",
   ];
```

creates a map allowing us to lookup the capital cities of any of the three states on the West Coast. We refer to the items used to lookup such as “California", "Oregon", and "Washington" as ***keys*** and the items looked up "Sacramento", "Salem", and "Olympia" as ***values***.

We can assign into the array or lookup items from the array using a square bracket notation.

```
$capitals["California"]
```

would return the string "Sacramento".

Keys in PHP Arrays are limited to integers or strings, and can be mixed (so an array could use integers for some keys and strings for other keys). Values may be of any type.

Simulating a Standard Array

You can think of a standard array as a map going from a set of integers used as keys to a set of values. For example, here are the top 5 films of 2017 by Worldwide Box Office earnings:

```
$films = [
       0 => "Star Wars: The Last Jedi",
       1 => "Beauty and the Beast",
       2 => "The Fate of the Furious",
       3 => "Despicable Me 3",
       4 => "Jumanji: Welcome to the Jungle",
     ];
```

PHP does support the short hand of skipping the numbers if we are assigning maps to sequential numbers starting with 0, so I can rewrite the $films assignment using what looks very much like a traditional array:

```
$films = [
       "Star Wars: The Last Jedi",
       "Beauty and the Beast",
       "The Fate of the Furious",
       "Despicable Me 3",
       "Jumanji: Welcome to the Jungle",
    ];
```

Iterating Through Arrays

If you’re iterating through an array that uses numeric indexes like our $films array you can do it using a foreach statement like this:

```
echo "<ol>";
foreach($films as $movie) {
    echo "<li>$movie</li>";
}
echo "</ol>";
```

For associative arrays, such as our capitals array, you have two choices. You may find it simplest to retrieve a traditional array containing the keys by using the array_keys function and iterate through the traditional array just as before. Here's an example using this approach:

```
$states = array_keys($capitals);
echo "<table>";
foreach($states as $currState) {
    $currCapital = $capitals[$currState];
    echo "<tr><td>$currState</td>
              <td>$currCapital</td></tr>";
}
echo "</table>";
```

Or you can use a fancier version of foreach which allows us to retrieve both key and value simultaneously. It looks like this:

```
echo "<table>";
foreach($capitals as $currState => $currCapital) {
    echo "<tr><td>$currState</td>
              <td>$currCapital</td></tr>";
}
echo "</table>";
```

Objects

In this handout we will only show you how to use pre-existing classes. You won't need to define new classes for anything we do in CS106E.

Creating a New Object

As we saw in the last handout, we use the keyword `new` followed by a constructor for the class.

```
$now = new DateTime();
```

A *constructor* is essentially a special function used to create and initialize objects of a particular class. Sometimes constructors take parameters, which is why the object class name has a pair of parentheses after it. Here we've created a date object for the Apollo 11 moon landing:

```
$moon_landing = new DateTime("1969/7/20");
```

Accessing Methods and Properties

Methods and properties are accessed in PHP using what is sometimes referred to as an *arrow operator*. This consists of a dash '-' followed by a greater than sign '>'. As we saw in the previous handout we can output the current date using the DateTime's format method:

```
echo $now->format("H:i:sA");
```

Warning: *Note that the symbol used for associative array definition is a => using the equals sign, whereas the symbol used for objects is a -> using the dash.*

NULL

If you're not familiar with the concept of NULL (also called "nil" in some programming languages), NULL is the value of a variable that normally refers to an object, but has not yet been assigned a value. It's also sometimes returned by functions that usually return an object, but want to indicate that in this particular case no return object exists. For example, if I have a function that returns a reference to an animal, and ask it for a unicorn, the function will return a NULL if no unicorns exist in my system:

```
$animal = getAnimal("unicorn");
```

NULL is treated as FALSE when used as a condition. The literal value is written as NULL (typically written with all capital letters, but as with TRUE and FALSE, it is case-insensitive, so you may also see it written as null or Null).

Control Structures

Control structures in PHP are very similar to those in C, C++, or Java.

Statements

Statements in PHP must be explicitly terminated with a semicolon. However, blocks enclosed in curly braces '{' '}' do not need semicolons.

If-Statements

If statements and if-else statements are very straightforward.

```
if(condition) {
  …statements…
}

if(condition) {
  …statements…
} else {
  …alternative-statements…
}
```

If you want to chain your if-else statements you can either use elseif (as a single keyword) or else if as two separate words.[5]

```
if(condition-1) {
  … statements-1 …
} else if(condition-2) {
  … statements-2 …
} else {
  … statements-3 …
}
```

or

[5] There are some nuances between these two if you use an alternative syntax structure using colons ':' instead of curly braces. But I won't cover that here.

```
if(condition-1) {
  … statements-1 …
} elseif(condition-2) {
  … statements-2 …
} else {
  … statements-3 …
}
```

While Loops

While loops are also very straightforward:

```
while(condition) {
  … statements …
}
```

For Loops

For loops are exactly the same as in C, C++, and Java, but may seem a bit odd to those coming from other languages. In C-based languages such as C++ and Java, a for loop has a control section consisting of three parts, separated by semicolons. The first part sets the initial value for the index variable controlling the loop, the second is a test for stopping the loop, and the third increments the index variable:

```
for(init; test; increment) {
  … statements …
}
```

For example, here I have a loop which has a control variable $i which goes from 0 to 9. Notice the test fails when $i is 10. We increment by 1 each time.

```
for($i = 0; $i < 10 ; $i++) {
  … statements …
}
```

Foreach

We've previously seen the foreach loop in the section on Arrays. The standard form looks like this:

```
foreach($list as $item) {
  … statements …
}
```

If you want to get both keys and values you can use the following variant:

```
foreach($map as $key => $value) {
  … statements …
}
```

See examples in the section on Arrays.

Comments in PHP

PHP supports single line comments with a double slash //.

```
index = index + 2;  // we advance index by even numbers
```

Multi-line comments can be written using /* and */

```
/* This is a multline
   PHP comment */
```

Code Outside of Functions and Classes

In contrast with languages such as C, C++, or Java, code in PHP does not need to be contained within a function or a class. In fact, if you look at the examples from the previous handout, none of the PHP code is in a function. Here's our Tax Calculation code, for example:

```
<?php

// we'll assume a tax rate of 10%
$amount = $_POST["amount"];
$tax = $amount * 0.1;
$total = $amount + $tax;

echo "<p>Tax on " . $amount . " is " . $tax . "</p>";
echo "<p>You owe a total of " . $total . "</p>";

?>
```

Variables that reside outside of a class or function are referred to as ***global variables***. They should be used with caution and can cause problems in larger programs – variables local to functions or classes are much easier to control and manage. However, they make sense in short programs like this that don't actually require any functions or classes.

Functions

For larger, more complex PHP programs, we can of course define functions. While these work similar to the way they do in other languages, there are a few wrinkles.

Declaration Format

PHP uses untyped parameters and function do not have return values. If you're coming from a statically typed language such as Java or C++, the format looks a bit different than you may be used to. We use the keyword return to define our function, instead of the return type listed there in C, C++, and Java. Also the parameters are listed without types.

```
function name(arg1, arg2, …, argN) {
   … statements …
}
```

Here's an example:

```
function interest($amount, $rate, $year) {
  return $amount * (1 + $rate)**$year;
}
```

Note also the use of the keyword return to return a value from the function.

Accessing Global Variables

If you need to access a global variable from a function, you'll need to explicitly indicate this using a special global statement within your function. For example, here I've gotten rid of the $rate parameter from the previous example and am instead accessing a global variable storing the interest rate.

```
$rate = 0.05;

function interest($amount, $year) {
  global $rate;

  return $amount * (1 + $rate)**$year;
}
```

Debugging

Include php.ini File

Just as a reminder, if you do not have the php.ini file in the directory where your PHP file is located, you will receive no error messages. If your PHP file has an error, you'll simply see a blank webpage.

Using print_r

When debugging, you may find it useful to print out the current values of different variables. You can do that with the print_r function. This function will print a human readable version of any value, including the contents of arrays. Here's an example of its use:

```
echo "<p>Printing Array Contents</p>";
print_r($myArray);
```

Learning More

You can find the official PHP language documentation here:

http://php.net/manual/en/langref.php