Variables in PHP

The syntax for PHP variables is similar to C and most other programming languages. There are three primary differences:

1. Variable names must be preceded by a dollar sign ($).
2. Variables do not need to be declared before being used.
3. Variables are dynamically typed, so you do not need to specify the type (e.g., `char`, `int`, `float`, etc.).

Rules for Naming Variables

A variable can have a short name, like x, or a more descriptive name, like carName. The rules for PHP variable names:

- Variables in PHP starts with a *$ sign*, followed by the name of the variable
- The variable name must begin with a *letter* or the *underscore* character
- A variable name can only contain *alpha-numeric* characters and *underscores* (A-z, 0-9, and _)
- A variable name should *not contain spaces*
- Variable names are *case sensitive* (y and Y are two different variables)

Data Types

PHP has a total of eight data types which we use to construct our variables:

1. **Integers:** are *whole numbers*, without a decimal point, like 4195.
2. **Doubles:** are *floating-point numbers*, like 3.14159 or 49.1.
3. **Booleans:** have only two possible values either *true* or *false*.
4. **NULL:** is a special type that only has one value: *NULL*.
5. **Strings:** are *sequences of characters*, like 'PHP supports string operations'.
6. **Arrays:** are named and *indexed collections* of other values.
7. **Objects:** are instances of *programmer-defined classes*, which can package up both other kinds of values and functions that are specific to the class.
8. **Resources:** are special variables that hold *references to resources external to PHP* (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are *compound* - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

PHP has a useful function named `var_dump()` that prints the current type and value for one or more variables. Arrays and objects are printed recursively with their values indented to show structure.

1. Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type. They correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

2. Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
$many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print(.$many + $many_2 = $few<br>.);
```

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

3. Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
if (TRUE)
 print("This will always print<br>");
else
 print("This will never print<br>");
```

Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

4. NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with `IsSet()` function.

5. Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = "This is a somewhat longer, singly quoted string";
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

```
My $variable will not print!\n
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are:

- `\n` is replaced by the newline character

- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

String as Document

You can assign multiple lines to a single string variable using here document:

```
<?php

$channel =<<<_XML_
<channel>
<title>What's For Dinner<title>
<link>http://www.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
_XML_;

echo <<<END
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
<br />
END;

print $channel;
?>
```

This will produce following result:

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!

<channel>
<title>What's For Dinner<title>
<link>http://www.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
```

6. Arrays

PHP has essentially one type of array – the *associative array* (i.e., hash table). Each element in the array has a ***key*** and a corresponding ***value***. Standard arrays (i.e., *indexed arrays*) can be used in PHP as well; they are simply associative arrays with integer-indexed keys.

There are three ways to populate an array. The *first method* is to use the `array()` function:

```
$proglangs = array("C", "C++", "Perl", "Java");
```

The *second method* is to access the elements directly using the array operator `[]`:

```
$proglangs[0] = "C"; $proglangs[1] = "C++"; $proglangs[2] = "Perl";
$proglangs[3] = "Java";
```

The third method is to use the array operator with no key provided:

```
for ($i=0; $i < 10; $i++)
$nums[] = $i+1;
```

This syntax is used less frequently, and has the effect of appending the given value to the array.

As with Perl, arrays are *heterogeneous*, meaning that the data stored in the array does not need to be of the same type. For example, `$mixedbag` contains a string, floating-point value, and a boolean value. To get the number of elements in an array, use the `count()` function.

Associative Arrays

Associative arrays work much like their indexed counterparts, except that associative arrays can have non-numeric keys (e.g., strings). The same three methods for populating indexed arrays apply, except for the array() function:

```
$file_ext = array(".c" => "C",
                  ".cpp" => "C++",
                   ".pl" => "Perl",
                 ".java" => "Java");
```

The value to the left of the => operator is the *key*, and the content after it is the corresponding *value*.

Multidimensional Arrays

Multidimensional arrays in PHP can be indexed or associative, and are heterogeneous. Consider the following code which constructs a multidimensional array.

```
$proglangs['scripted'] = array("Perl", "Python", "PHP");
$proglangs['compiled'] = array("C", "Java", "FORTRAN");
$proglangs['fun'] = array("PHP" => "Web programming", "Java" => "Nice for
object-oriented code.");
```

The variable $proglangs is a multidimensional associative array. The first two statements create indexed arrays of strings, and the last statement creates another associative array.

7. Objects

Object Oriented Programming (OOP) promotes clean modular design, simplifies debugging and maintenance, and assists with code reuse.

Classes are the unit of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the `class` keyword:

```
class Person
{
var $name = '';

function name ($newname = NULL)
{
if (! is_null($newname))
{
$this->name = $newname;
}
```

```
return $this->name;
}
}
```

Once a class is defined, any number of objects can be made from it with the `new` keyword, and the properties and methods can be accessed with the `->` construct:

```
$ed = new Person;
$ed->name('Edison');
printf("Hello, %s\n", $ed->name);
$tc = new Person;
$tc->name('Crapper');
printf("Look out below %s\n", $tc->name);
Hello, Edison
Look out below Crapper
```

Use the `is_object()` function to test whether a value is an object:

```
if (is_object($x))
{
// $x is an object
}
```

8. Resources

Many modules provide several functions for dealing with the outside world. For example, every database extension has at least a function to connect to the database, a function to send a query to the database, and a function to close the connection to the database. Because you can have multiple database connections open at once, the connect function gives you something by which to identify that connection when you call the query and close functions: a resource.

Resources are really integers under the surface. Their main benefit is that they're garbage collected when no longer in use. When the last reference to a resource value goes away, the extension that created the resource is called to free any memory, close any connection, etc. for that resource:

```
$res = database_connect();  // fictitious function
database_query($res);
$res = "boo";                // database connection automatically closed
```

The benefit of this automatic cleanup is best seen within functions, when the resource is assigned to a local variable. When the function ends, the variable's value is reclaimed by PHP:

```
function search()
{
$res = database_connect();
$database_query($res);
}
```

When there are no more references to the resource, it's automatically shut down.

That said, most extensions provide a specific shutdown or close function, and it's considered good style to call that function explicitly when needed rather than to rely on variable scoping to trigger resource cleanup.

Use the `is_resource()` function to test whether a value is a resource:

```
if (is_resource($x))
{
// $x is a resource
}
```

Variable Variables

You can reference the value of a variable whose name is stored in another variable.

For example:

```
$foo = 'bar';
$$foo = 'baz';
```

After the second statement executes, the variable `$bar` has the value "`baz`".

Variable References

In PHP, references are how you create variable aliases. To make `$black` an alias for the variable `$white`, use:

```
$black =& $white;
```

The old value of `$black` is lost.

Functions can return values by reference (for example, to avoid copying large strings or arrays):

```
function &ret_ref( )  // note the &
{
$var = "PHP";
return $var;
}
$v =& ret_ref( );     // note the &
```