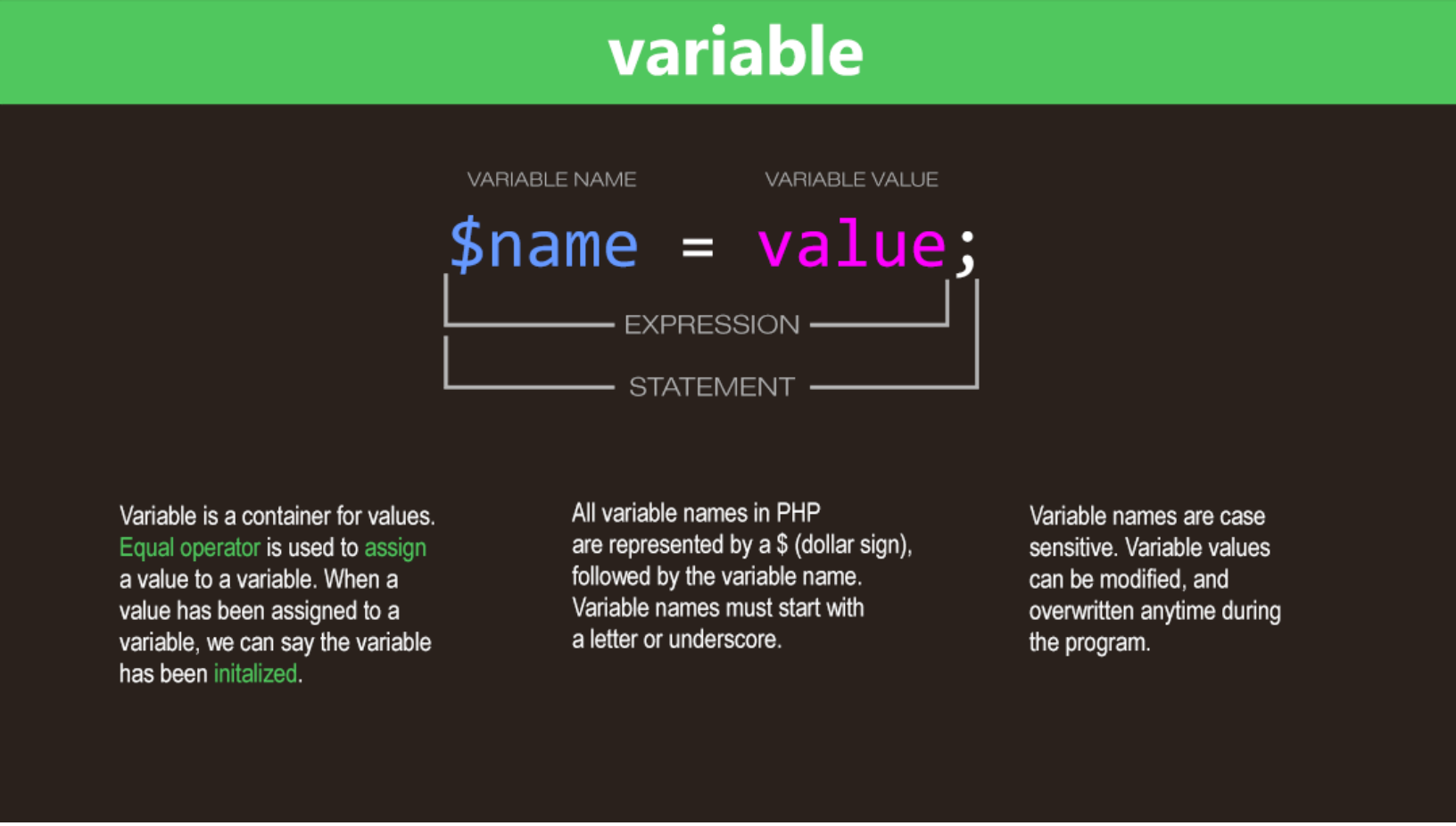

php
programming
CLEVER TECHIE

Preface

Hi everyone! Thank you for downloading the PHP programming book. This book is a work in progress and this is the very first version of it. I'm going to constantly update it, add more content and improve things as times goes on. I just wanted to get this out as soon as possible for you guys so you can start learning PHP right away. This book is mostly intended for beginner PHP programmers but intermediate and advanced coders will also find some useful concepts in it as well. I've personally learned a lot writing this short book and I hope you get you all get a lot out of it. Please write any comments, questions and suggestions to vladi@clevertechie.com.

Clever Techie



PHP Variable

A variable is a container that stores values, values can be of any PHP supported data types.

```php
$movie_name = 'Gladiator'; //string value data type
$release_year = 2000; //integer data type (whole number)
$cost_now = 7.96; //float / double data type (fractional number)
$awesome = true; //boolean data type (true or false only)
```

We can then use variable names to print out the values they hold to the screen using keyword echo (also known as an output statement). When placing variables inside a string, it's important to use double quotes, doing so will print out variable values, which is what we want. Placing variable names inside the

quotes in PHP is called interpolation and is worth learning about from the beginning.

```
echo "The movie $movie_name was released in $release_year and it now costs $$cost_now at amazon.com on blue ray. ";
echo $awesome ? "The movie is awesome!" : "The movie is okay....";
//output: The movie Gladiator was released in 2000 and it now costs $7.96 at amazon.com on blue ray. The movie is awesome!
```

I also used ternary operator ? : in the above example to check if the value of $awesome is equal to true. If it's true "The movie is awesome!" is printed, if not "The movie is okay...." would be printed. A ternary operator is the shortest version of conditional expressions in PHP.

Naming Conventions

Here is a simple example which shows what case sensitive variables mean in PHP:

```
//The following are 3 different variables:
$name = 'Atari';
$Name = 'Sega';
$NAME = 'Nintendo';
echo $Name; //output: Atari
echo $name; //output: Sega
echo $NAME; //output: Nintendo
```

Just make sure you never start a variable name with a special character or a number, you'll get an error, because PHP doesn't allow that, you can use an underscore though:

```
$_valid = "This is okay to do";
```

Also, not a strict rule, but rather good programming practice is to name your variables in a similar style, usually either $CamelCaseFormat, or $under_score_format. As you can see in the examples across this web site I prefer to use the underscore format and use all lower case letters in a variable name.

Modifying Variables

Variable values often change (vary). There are many different functions and operators to assign, modify, compare and check variable values. In the following example, we find out how long ago the movie Gladiator was released by using a subtraction operator.

```
$year_now = date( "Y" ); //gets current year
$years_ago = $year_now - $release_year;
echo "The $movie_name was released $years_ago years ago!";
//output: The Gladiator was released 17 years ago!"
```

A built-in function date() with a string parameter "Y" was called to get the current year. We then used the subtraction operator to subtract current year from the release year which gave us how many years ago the movie was released. In fact

PHP supports all arithmetic operators. Let's find the average of two numbers using a couple of operators:

```
$num1 = 10;
$num2 = 30;
$average = ( $num1 + $num2 ) / 2;
echo $average; //output: 20
```

Pre-Increment & Post-Increment

One very common operator that you will encounter a lot is ++ (plus plus), which simply increments the variable value by 1.

```
$my_age = 33;
echo "Later this year I will be ".++$my_age;
//output: Later this year I will be 34
```

Just in case you noticed that (.) dot in front of ++$my_age, let me explain myself :) The dot in PHP is called "concatenate" which simply means, join the two together. We have to use concatenate whenever there is anything more complex going on like calling a function, using increment/decrement or any other operation beyond simply using a variable name.

In a similar way -- (minus minus) decreases the value of variable by one.

```
echo "Last year I was ".--$my_age;
//output: Last year I was 32
```

Let's also clarify the difference between ++$my_age (pre-increment) and $my_age++ (post-increment). In the case of pre-increment which is the example above, the value of the variable is incremented by one and then returned, so we get 34. In case of $my_age++ the original value of 33 would be returned and printed to the screen, and then it would be incremented by 1. Post-increment is used all the time in PHP For Loops.

Below is an example of a post-increment, notice the difference from earlier pre-increment example, the value of $my_age returned in the first echo output statement is still 33 (the original value is returned), and is only then incremented by one in the second echo statement, giving us 34.

```
$my_age = 33;
echo "I am ".$my_age++." years old now";
//output: I am 33 years old now
echo "On July I'm turning $my_age";
//output: On July I'm turning 34
```

PHP Variable Handling Functions

There are many PHP built-in functions to work with variables, let's look at some of the most useful and common ones.

PHP VARIABLE FUNCTIONS	
`string gettype( $var );`	gets the data type of the variable
`bool isset( $var );`	checks if the variable has been initialized
`void unset( $var );`	destroys variable
`bool is_float( $var );`	checks if variable type is float
`bool is_int( $var );`	checks if variable type is integer
`bool is_null( $var );`	checks if variable type is null
`bool empty( $var );`	check if variable is null or doesn't exist
`bool is_numeric( $var );`	checks if variable value is a number
`string var_dump( $var );`	provides information about a variable
`int intval( $var );`	converts variable value to integer type
`string strval( $var );`	converts variable value to string type
`bool is_array ( $var );`	checks if variable data type is array
`bool settype ( $var, string $type );`	set the type of a variable

I think most of the above functions should be pretty self-explanatory. One thing to note is the keyword just before the function name, colored in purple (string, bool, void, int). This is simply the data type of the value which will be returned by the function after we call it on a variable. Let's use same variables from the first example of this lesson and run some of these functions on them:

```php
$movie_name = 'Gladiator';
$release_year = 2000;
$cost_now = 7.96;
$awesome = true;
echo gettype( $movie_name ); //output: string
echo gettype( $awesome ); //output: bool
echo is_int( $release_year ); //output: 1
echo is_float( $cost_now ); //output: 1

echo isset( $awesome ); //output: 1
echo intval( $cost_now); //output: 7
var_dump( $movie_name ); //output: string(9) "Gladiator"
unset( $movie_name );
echo $movie_name; //Notice: Undefined variable: movie_name
echo empty( $movie_name ); //output: 1
```

As you can see all the value data types in the output are exactly the same as indicated by the return type (keyword in purple before the function name) mentioned earlier.

The function intval(); above converts a fractional (float) number into an integer. Function var_dump(); outputs: string(9) "Gladiator", which is the data type of the variable - string, 9 is the number of characters that string has and finally the content of the string itself - "Gladiator". After using function unset(); the variable $movie_name is destroyed, we try to echo it out and get "undefined variable" notice as expected. We then use empty(); function to check if the variable $movie_name does not exist, in fact it doesn't exist so the output is 1 (true).

Feel free to play around with the rest of the functions!

Variable Scope

So far, all the variables we have defined in this tutorial have global scope, meaning they are accessible anywhere in the program. However, in PHP, variables which are declared inside a function have local scope, meaning they are only accessible within the function where they have been created. Let's take variable scope under the microscope:

```
function microscope()
{
//local scope
$local = 'Hi I\'m $local: I live only inside the function!';
echo $local;
}

//trying to access local variable from a global scope
echo $local; //Notice: Undefined variable: local
microscope(); //output: Hi I'm $local I live only inside the function!
```

Please keep in mind that "global" scope only refers to any area in the program outside of a function. Any variable defined outside of a function which is said to have global scope will still be invisible inside a function:

```
$global = "hola";
function microscope() {
echo $global;
}
microscope(); //output: Notice: undefined variable: global
```

But! There is, of course, a way to create a variable which has access within the function it has been created, as well as outside the scope of the function, we can use keyword "global":

```
function microscope()
{
    global $real_global;
    $real_global = "Hi I\'m all over the place!";
    echo $real_global;
}
echo $real_global; //output: Hi I'm all over the place!
```

Note how I used the keyword global following the variable name $real_global but didn't assign any value to it on the first line, this is for a reason. It's because we first have to use keyword global and only then initialize (assign a value) to the variable.

Static Keyword

Lastly, let's go over static keyword. Whenever you call a function in PHP, any variable that was created when the function was called is destroyed, that is unless we declare that variable using static keyword:

```
function solar_system()
{
static $planets = "Earth, Mercury";
echo $planets;
$planet_one = "Mars";
$planet_two = "Venus";
$planets = $planets.', '.$planet_one.', '.$planet_two;
```

```
}
solar_system(); //output: Earth, Mercury
solar_system(); //output: Earth, Mercury, Mars, Venus
```

By using static, the value of the variable is retained so when we call the function a second time, we see the added values of the other two planets. This concept may be a bit confusing, so let me clarify with a non static example:

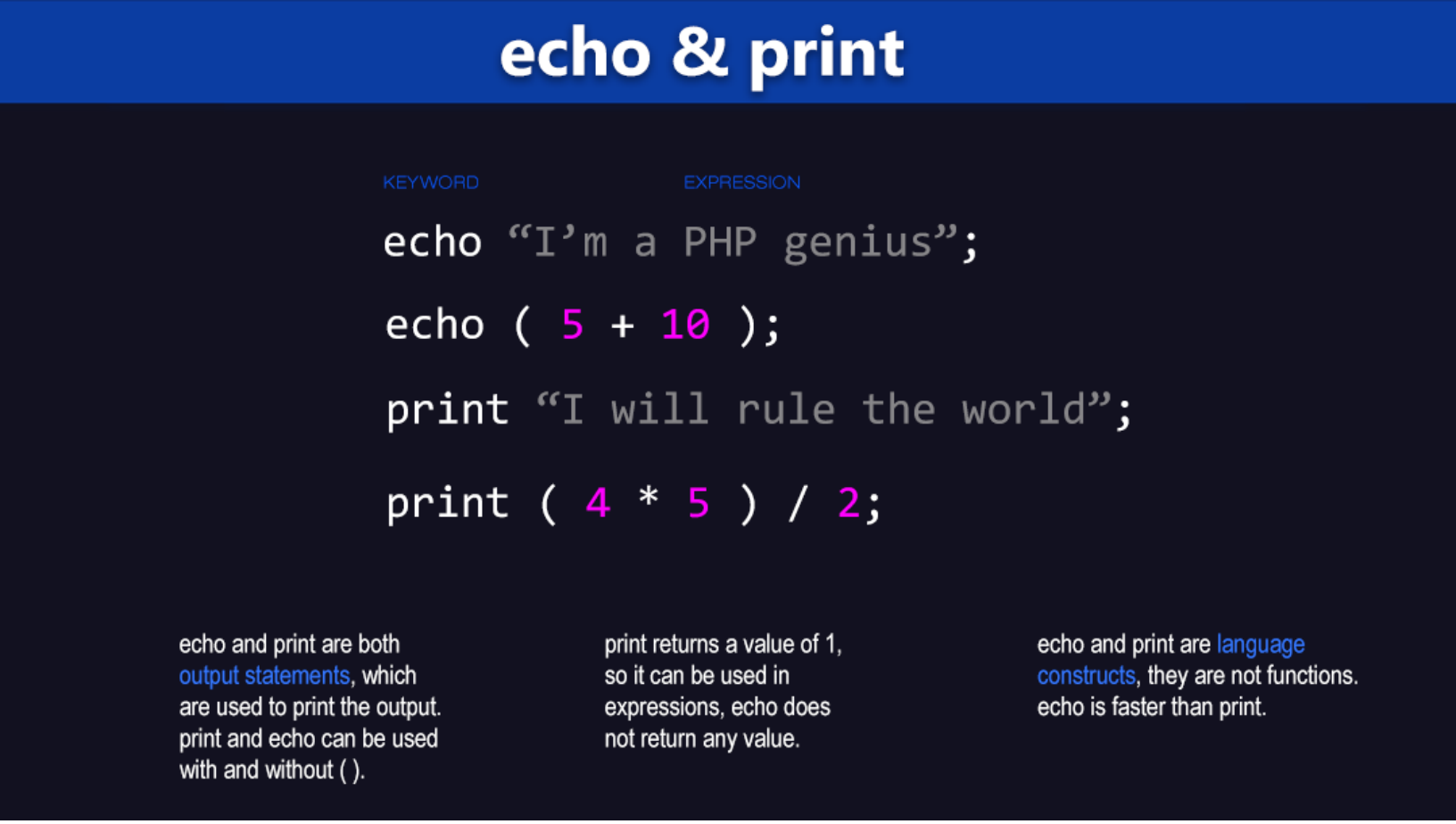
```
function solar_system()
{
$planets = "Earth, Mercury";
echo $planets;
$planet_one = "Mars";
$planet_two = "Venus";
$planets = $planets.', '.$planet_one.', '.$planet_two;
}
solar_system(); //output: Earth, Mercury
solar_system(); //output: Earth, Mercury
```

Any modifications done to the variable after it has been printed with echo have been lost, so the other two planets aren't printed even when calling a function second time, this is how all variables behave within functions without static keyword.

PHP Variable Vocabulary

Here is a list of new vocabulary words that were used in this lesson, I think it's important to learn programming lingo from the beginning because it will help you learn new programming languages and concepts faster and you'll avoid a lot of confusion.

Vocabulary	
variable	a container for values which can and often do vary
value	an expression with any supported data type
equal operator	used to assign a value to a variable
assign	to put the value inside the variable container
initialize	to assign a value to a variable
declare / define	to create a variable, function, class, array
expression	anything that expresses a value
statement	code that performs action, in PHP often ended with ;
data type	a specific data item which the programming language supports
integer	whole number
float	fractional number
string	text (letters stringed together to make a word)
boolean	can only be either 0 (false) or 1 (true)
scope	which part of the program is able to "see" variables
local scope	variables that are defined inside a function
global scope	variables that are defined outside a function
echo	used to output/print data to the screen



PHP echo and print

PHP echo and PHP print are both output statements used to print the output to the screen. Remember that both echo and print are language constructs, they can be used with and without parenthesis.

New Line Breaks

I think it's worth clarifying how PHP new lines work from the beginning to avoid a lot of confusion later on. Most of this confusion comes from the context of where these line breaks are used, I'm going to cover three of them: Outputting to command prompt, writing to files, and printing to web browsers.

Command Prompt New Lines (\n)

When using the echo statement and putting your text on new lines, the new lines will be created in the output as well:

```
echo "This text
extends to multiple lines,
the new lines are output as well.";
```

However, what a lot of beginner coders don't realize, is this is only referring to command prompt output, where PHP output is rarely used:

```
C:\localhost\php_lesson1\lessons>php echo.php
This text
extends to multiple lines,
the new lines are output as well.
```

Another way of writing line breaks to command prompt is to use the "\n" character:

```
echo "This text \nextends to multiple lines,\nthe new lines are output as well.";
```

Which will produce exactly the same result as previous example:

```
C:\localhost\php_lesson1\lessons>php echo.php
This text
extends to multiple lines,
the new lines are output as well.
```

New Lines When Writing to Files (\r\n)

Simply using \n won't work when writing our output to files, and even though it doesn't involve the echo or print statement, I thought it would be a good idea to

include the example here as well. The proper way to include line breaks when writing text to files is to use "\r\n"

```
$output = "This is some other random\r\n
text that will be printed on different lines\r\n
when output in a text file."

//code to write our output in a new text file called newfile.txt
$new_file = fopen( "newfile.txt", "w" ) or die( "Unable to open file!" );
fwrite( $new_file, $output );
fclose( $new_file );
```

Output inside the newfile.txt:

```
This is some other random
text that will be printed on different lines
when output in a text file.
```

New Lines When Printing to Browsers (
)

We finally get to the most common use of line breaks, when printing text in a browser, using the
 HTML tag. None of the previous examples would work in a browser because the browser interprets text a little bit differently and needs tags for formatting. So we would re-write the previous example as follows:

```
<html>
<body>
<?php echo "This is some other random<br />
text that will be printed on different lines<br />
when output in a browser."; ?>
</body>
</html>
```

Output in a browser:

```
This is some other random
text that will be printed on different lines
when output in a browser.
```

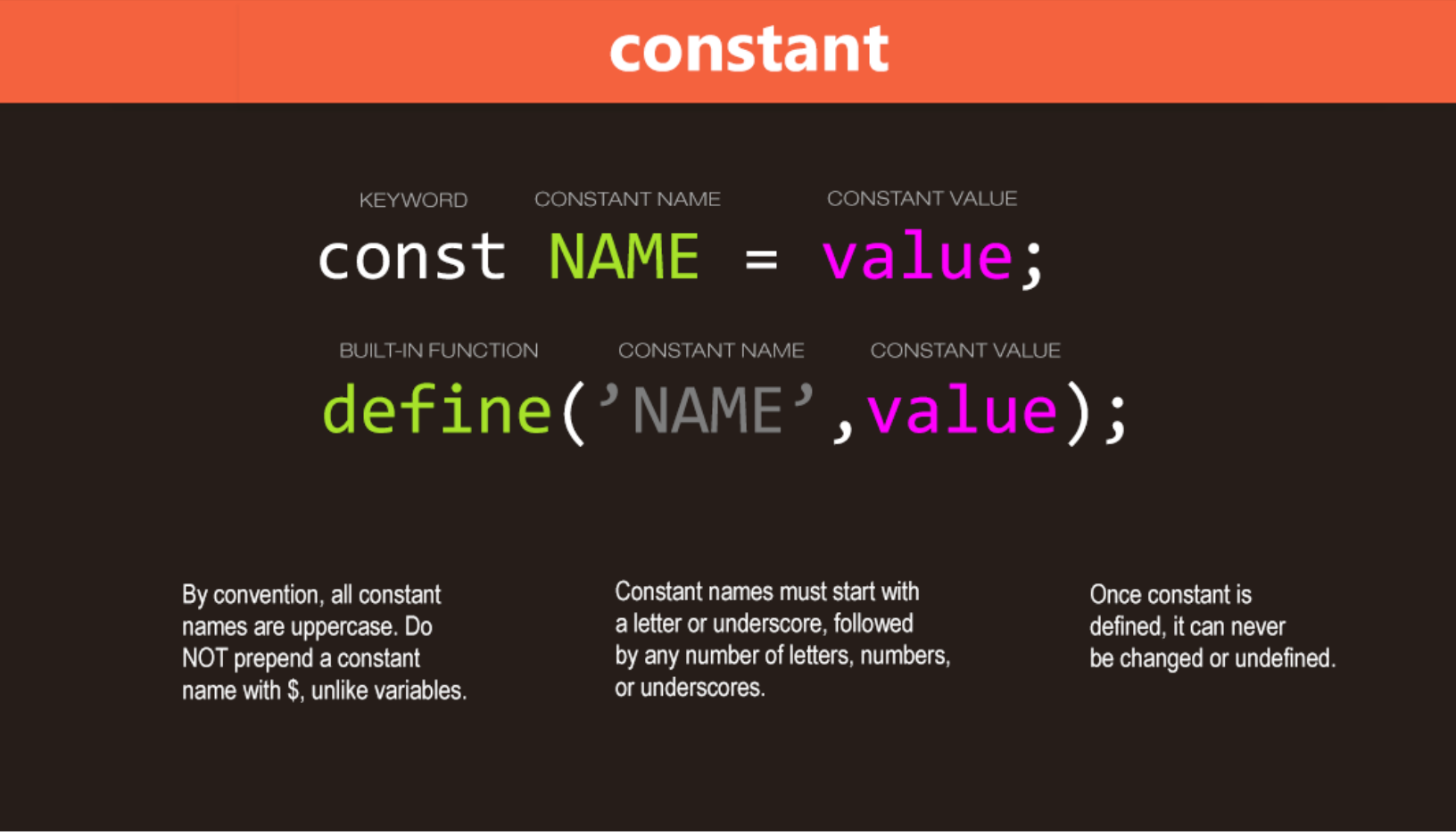
Interpolation

This fancy sounding word refers to the use of variables inside of quotes in PHP:

```php
$cost = 20;
//variable is expanded inside double quotes
echo "This product costs $cost"; //output: This product costs 20
//variable is not expanded inside single quotes
echo 'This product costs $cost'; //output This product costs $cost
```

Arrays can also be printed inside double quotes, though we have to surround it with curly brackets

```php
$animal = array( 'sound' => 'meow' );
echo "Cats {$animal[ 'sound' ]} when they're hungry";
//output: Cats meow when they're hungry
```



PHP Constant

By convention, all constant names are uppercase. Do NOT prepend a constant name with $, unlike variables. Constant's values, once defined, never change.

Const and Define()

Here is one way to create a constant in PHP, using the const construct:

```
const YEAR_BORN = 1983;
echo "I was born in the year ".YEAR_BORN;
//output: I was born the year 1983
```

Another way to create the same constant using the built-in function define();

```php
define('YEAR_BORN', 1983);

echo "I was born in the year ".YEAR_BORN;
//output: I was born the year 1983
```

Trying to put the constant name inside double quotes just like variable names, will not get the desired result because PHP wouldn't know you're referring to a constant and will simply interpret literal text."

```php
const YEAR_BORN = 1983;
echo "I was born in the year YEAR_BORN";
//output: I was born in the year YEAR_BORN
```

So make sure to always concatenate constant names with a dot (.)

Constants in a Class Definition

Though both const and define() are both correct ways to define a constant, using define() within class definition won't work:

```
class Life
{
define ( 'CHANGE', true ); //wrong, only works outside of class definition
const CHANGE = true; //correct, works inside of class definition
}
```

Also, the way to access constant's values when we're dealing with classes is different. Inside of a class definition, we have to use the "self" keyword followed by two semi-colons, followed by the constant name:

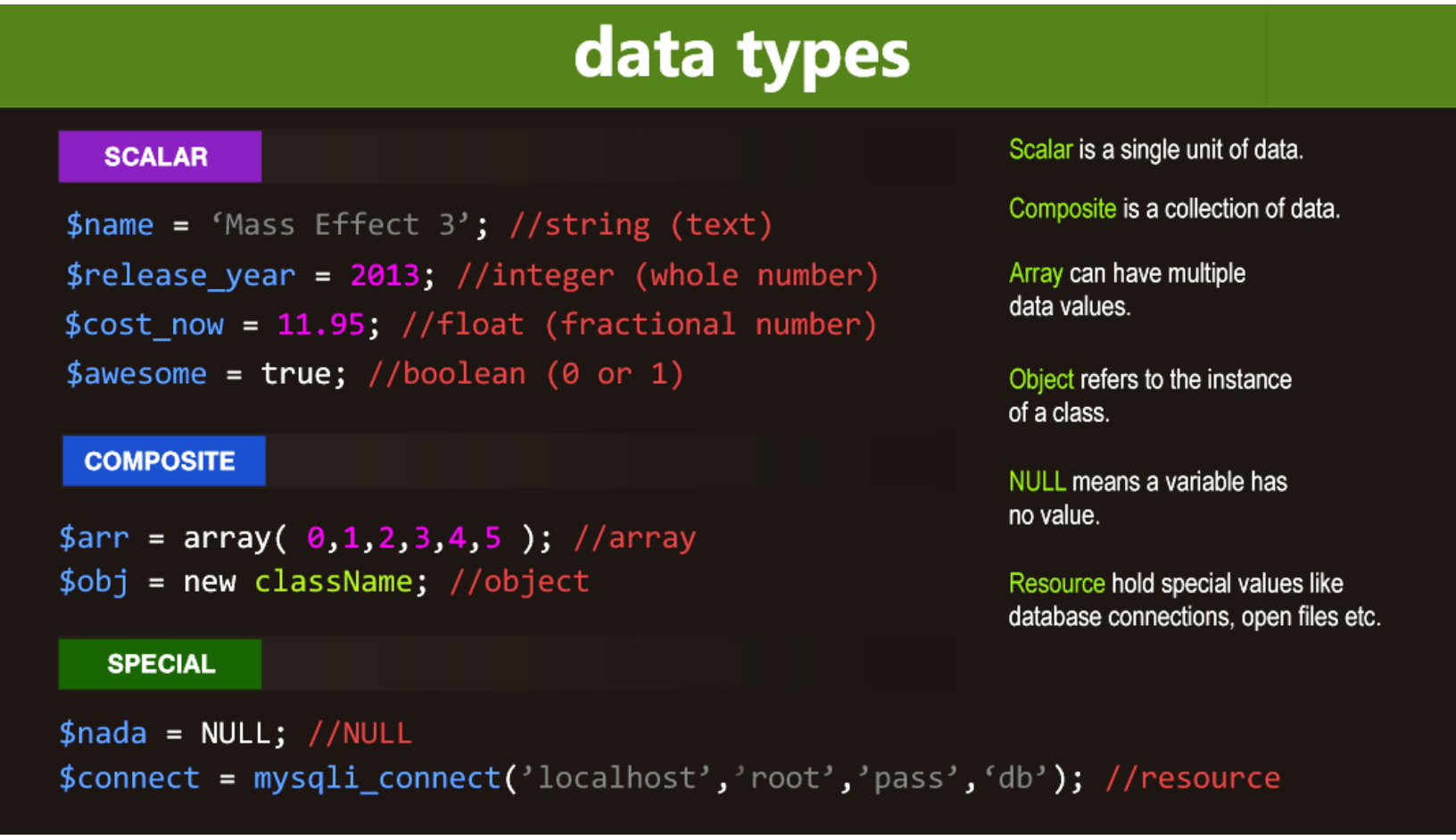
```
class Life
{
const CHANGE = true;
public static function getChange()
{
return self::CHANGE;
}
}
```

Then, if we wanted to access constant's value outside of a class definition, we can use two ways:

```
//1) Type class name, two semi-colons, and constant name exactly
$answer = Life::CHANGE;
//2) Type class name, two semi-colons, and method name with parenthesis
$answer = Life::getChange();
```

We are able to get the value of our constant using getChange() method because it was defined as a static method. All static methods in PHP that are defined in a class are also accessed with double semi-colon just like a constant, though don't forget the parenthesis at the end.

There are no set methods like setChange() within the class definition because once constants are defined and initialized, they can't be changed.



PHP Data Types

PHP supports many different value data types including scalar, composite and special. Scalar refers to just a single unit of data and include most common values: string, integer, float/double and boolean. Composite can be thought of as a collection of data: arrays and objects. Special are other data types like NULL (no value) and resources (eg: mysql connection, open files etc.)

Playing Around with Data Types

Let's create some variables with different data types supported by PHP and then toy around with them by calling all kinds of useful built-in PHP functions designed to handle various data types.

```
$name = 'The Matrix'; //string
$release_year = 1999; //integer
$cost_now = 8.99; //float
$awesome = true; //boolean
$cast = array( 'Keanu Reeves' => 'Neo', 'Laurence Fishburne' => 'Morpheus',
'Carrie-Anne Moss' => 'Trinity', 'Hugo Weaving' => 'Agent Smith' ); //array
class Movie { }
$matrix = new Movie; //object
$nothing = NULL;
$open = fopen( "C:\\movies\\matrix.txt", 'r'); //resource
```

Let's review some of PHP's built-in variable handling functions so we can have some fun with those variable data types we just defined:

PHP VARIABLE FUNCTIONS	
`string gettype( $var );`	gets the data type of the variable
`bool isset( $var );`	checks if the variable has been initialized
`void unset( $var );`	destroys variable
`bool is_float( $var );`	checks if variable type is float
`bool is_int( $var );`	checks if variable type is integer
`bool is_null( $var );`	checks if variable type is null
`bool empty( $var );`	check if variable is null or doesn't exist
`bool is_numeric( $var );`	checks if variable value is a number
`string var_dump( $var );`	provides information about a variable
`int intval( $var );`	converts variable value to integer type
`string strval( $var );`	converts variable value to string type
`bool is_array ( $var );`	checks if variable data type is array
`bool settype ( $var, string $type );`	set the type of a variable

Instead of running these functions individually on each of the variables we have created, I think a more efficient approach would be to loop through them, so let's create one variable array from all the variables we just defined first:

```
//create numeric array with all the variables
$data = array($name, $release_year, $cost_now, $awesome, $cast, $matrix,
$nothing, $open);
```

Now that we have $data with all the variables in it, we can simply create a for loop to go over each variable inside the array and run whatever function we want on those variables like this:

```
for ( $i = 0; $i < count( $data ); $i++ )
{
echo gettype( $data[ $i ] ).'<br />';
}
/*output:
string
integer
double
boolean
array
object
NULL
resource
*/
```

In PHP, a double is the same thing as a float, in case you're wondering. So it turns out I didn't lie about those data types, PHP gets the right data type using gettype() function for all the variables we have declared and you now know this to be true! :)

Detecting Value Types

Let's use a couple of functions from the list, combined with an if statement to print out some values if they match our data type check

```
for ( $i = 0; $i < count( $data ); $i++ )
{
if ( is_array( $data[ $i ] ) )
{
```

```
print_r( $data[ $i ] );
}
elseif ( is_numeric ( $data[ $i ] ) )
{
echo "Numeric value detected: {$data[ $i ]}<br />";
}
}
/*output:
Numeric value detected: 1999
Numeric value detected: 8.99
Array ( [Keanu Reeves] => Neo [Laurence Fishburne] => Morpheus [Carrie-Anne
Moss] => Trinity [Hugo Weaving] => Agent Smith )
*/
```

```
$name = 'Clever Techie'; //string data type

$string1 = "Hello my name is $name"; //output: Hello my name is Clever Techie

$string2 = 'Hello my name is $name'; //output: Hello my name is $name

$array['birth_year'] = 1983;

//array values can be accessed with { } curly brackets inside double quotes

$string3 = "I was born in: { $array['birth_year'] }";

//output: I was born in 1983
```

Variables can be accessed inside double qutoes in PHP.
When placed inside single quotes, however, the PHP
will print out and interpret any string/text as literal values.

PHP Strings

PHP string data types are created by placing text in single quotes (literal) or double quotes. The word "string" comes from "letters stringed together to make up words"

Single vs Double Quotes

Let's look at a quick example to see what the difference between double and single quotes is

```
$website = 'clevertechie.com'; //string data type
$double = "Hello my web site is called $website";
//output: Hello my web site is called clevertechie.com
$single = 'Hello my web site is called $website';
//output: Hello my web site is called $website
```

As you can see the value of the variable was shown in the output for double quotes, just like expected. A proper way to say this is: "a variable has been expanded" according to php.net. However, when using single quotes, the text $website was in the output, which is the literal variable name, hence single quotes are also called literal quotes, because they will interpret any text within literally.

It's a good practice to use single quotes whenever you're sure that you won't be using any variable names within the string.

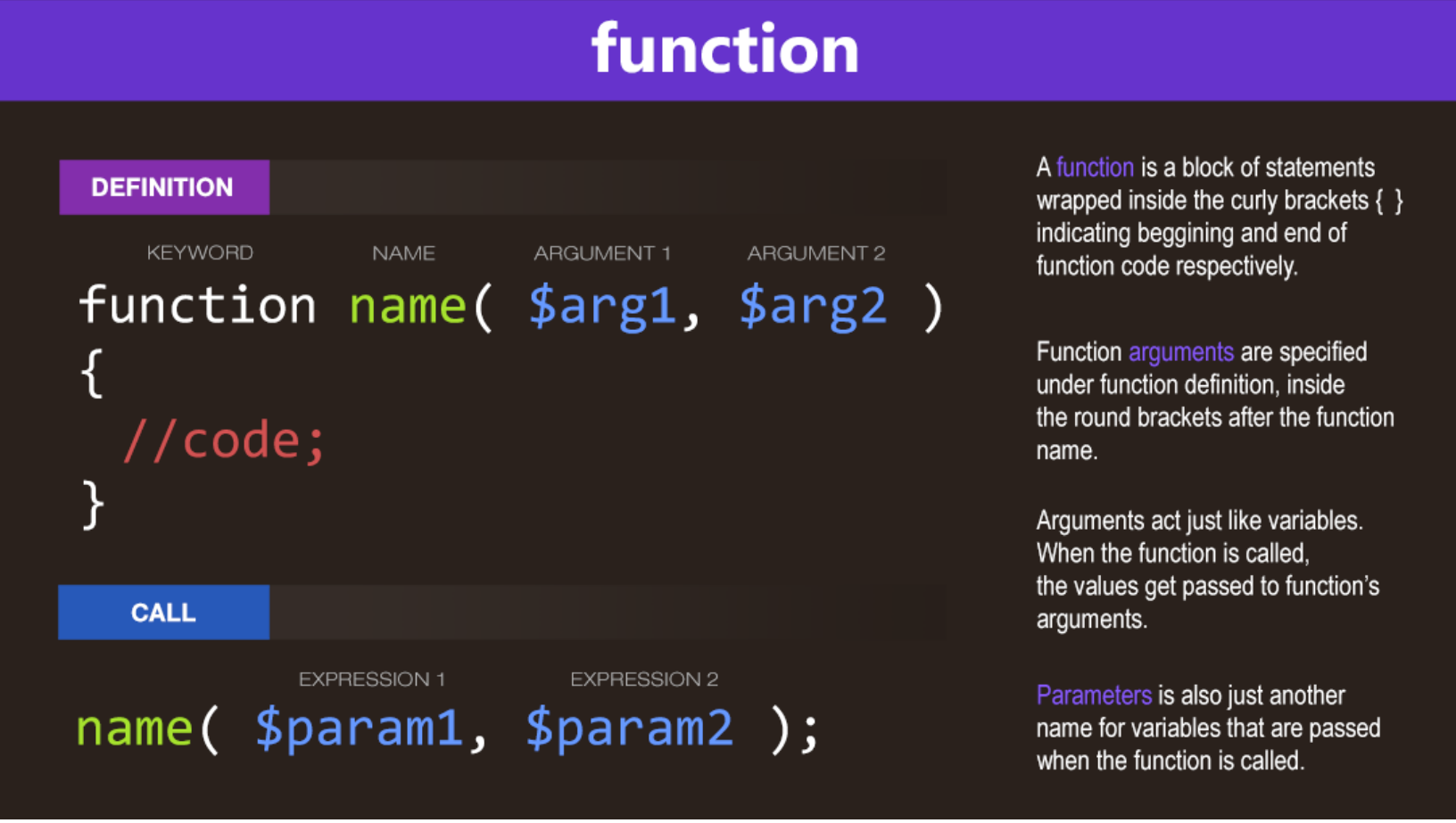
String Delimiters

If you're going to use a single quote character within single quotes, it must be escaped with a backslash delimiter (\ )

```
$message = 'I think we\'re going to make a great team!';
//output: I think we're going to make a great team!
```

And if we wanted to print the backslash inside single quotes, we would escape it as well, creating a double backslash (\\ ).

```
$message = 'Are you sure you want to wipe out
C:\\localhost\\clevertechie.com?';
//output: Are you sure you want to wipe out C:\localhost\clevertechie.com?
```



PHP Function

A function in PHP is created using keyword "function" followed by the function name which must start with a letter or underscore. We then use round brackets after function's name and provide optional arguments within round brackets, separated by comma. Curly brackets follow which indicate the start and end of the function's code. All code that will be performed by the function go inside the curly brackets.

Arguments vs Parameters

Let's clarify a subtle difference between arguments and parameters. Arguments are variables which are future placeholders for values, those values will be assigned when the function is called. Parameters on the other hand are values for those arguments defined in the function. Parameters must have values, so

they can also be named expressions, because in PHP an expression is anything that expresses a value.

Many programmers understandably refer to arguments and parameters as synonyms because they are so similar, however I think it's important to get the lingo right from the beginning when learning a programming language because you'll avoid a lot of confusion in the future.

```
//$one and $two are arguments
function multiply( $one, $two ) //function definition
{
return $one * $two;
}
//5, 5 are called parameters
$multiply_result = multiply( 5, 5 ); //function call
echo $multiply_result;
//output: 25
```

Returning values from functions (using keyword return) can be useful in case we want to store information in a variable or modify it later in the program. Another way is to directly output the result from within the function:

```
function multiply_print( $one, $two ) {
echo $one * $two; //printing values directly
}
multiply_print( 10, 5 );
//output: 50
```

Default Values

Function's arguments can also have default values, which we assign during function's definition:

```
function multiply_default($one = 5, $two = 10) {
echo $one * $two;
}
```

Now multiply_default() function can be called without specifying any parameters

```
multiply_default();
//output: 50
```

Passing parameters to a function which has default argument values will overwrite those values:

```
multiply_default( 4, 4 );
//output: 16
```

Type Declarations

We can set specific data type arguments when writing a function declaration. In the below example, the only value that will be accepted is of data type array.

```php
function print_info(array $info) {
print_r($info);
}
$my_info = array('name' => 'Vladi', 'age' => 33, 'gender' => 'M', 'job' =>
'Clever Techie');
print_info($my_info);
//output: Array ( [name] => Vladi [age] => 33 [gender] => M [job] => Clever
Techie )
```

Arguments by Reference

Whenever we work with data inside the function, that data, will not be available outside of the function, unless the value is returned. However, there is another way of keeping whatever data we have modified inside the function - passing function arguments by reference!

First, this is what would happen if we didn't pass this one argument by reference:

```php
function add_text( $string )
{
//.= means concatenate string or add string to a string
$string .= " and sometimes I like programming";
echo $string; //outputs: "My name is Vladi and sometimes I like programming"
}
$my_text = "My name is Vladi";
add_text( $my_text);
echo $my_text; //outputs: "My name is Vladi";
```

So when we print the value of $string inside the function, we get the expected result, with extra text, but when we try to echo out the text after calling the function, outside of it, then only the original part is printed.

But....but...what if I want that extra text and print it outside the function as too? Well then... you need to pass function arguments by reference of course! All we need to do is add an ampersand & in front of the function argument, it's that easy!

```
function add_text( &$string ) {
$string .= " and sometimes I like programming";
}
$my_text = "My name is Vladi";
add_text( $my_text);
echo $my_text; //outputs: My name is Vladi and sometimes I like programming
```

Recursion

Don't worry...a recursive function is just a function that calls itself, remember to always have an exit when defining such function, otherwise it will run forever (just like an endless loop).

```
function recursion($a)
{
if ($a <= 20) //exits when $a is greater than 20
{
echo "$a\n";
recursion($a + 1);
}
}
```

User-Defined vs Built-in Functions

We have defined many functions throughout this lesson, those are all our precious user-defined functions. There is also a library of functions that are already created and come with PHP called built-in functions. Here are some common built-in functions:

```
print_r ( $array ); //prints out the array
count ( $array ); //counts the number of elements in an array
strtolower ( $string ); //converts a STRING to lower uppercase letters: string
```

Let's Get Practical

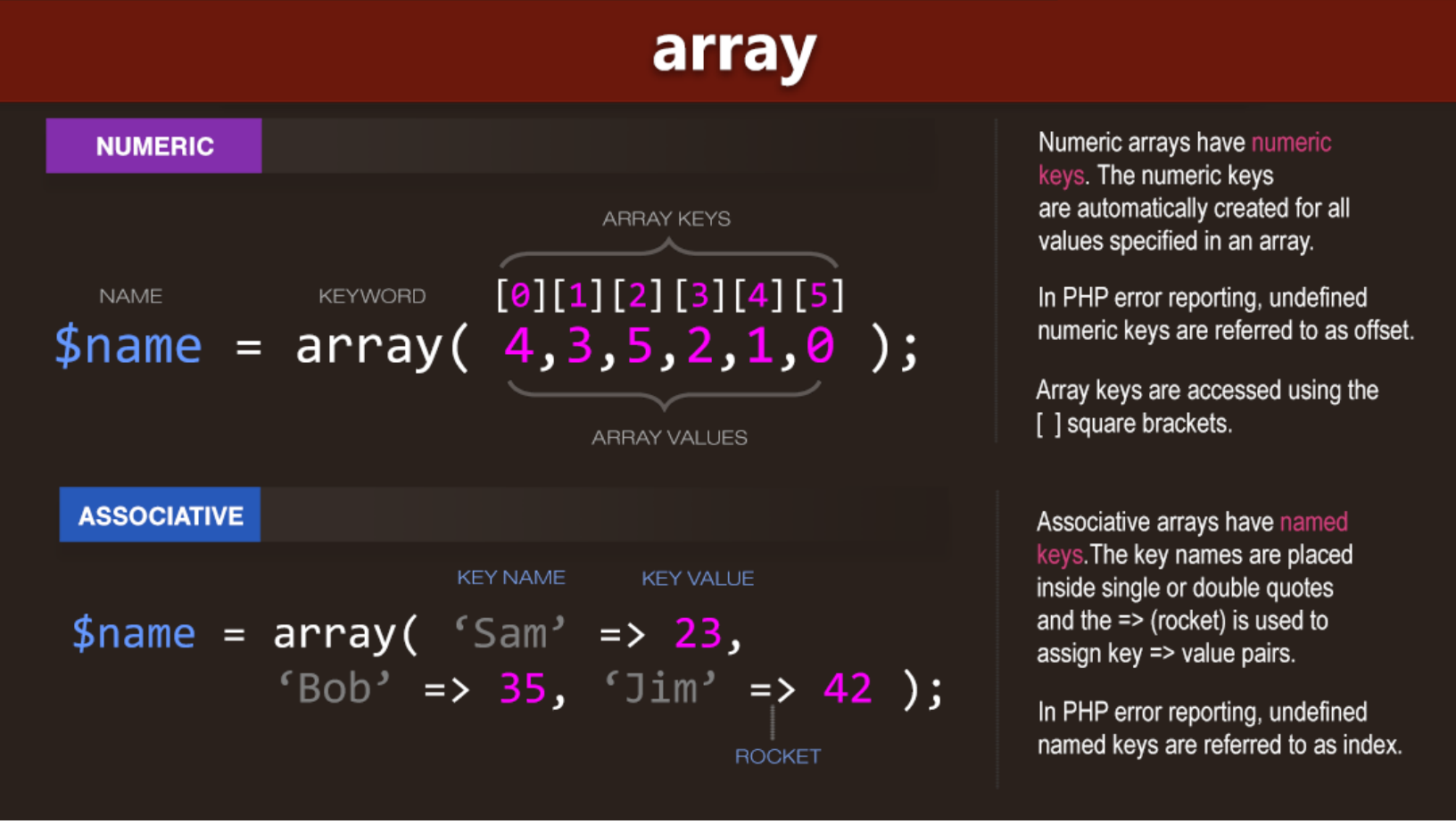
So what would be a good example of a user-defined function that we could use in a real virtual life?

```
function print_file_names_from_directory( $dir_path )
{

    $files = scandir( $dir_path );
    $files = array_diff($files, array('..', '.'));
    $files = array_values($files);

    for ($i = 0; $i < count($files); $i++)
    {
        echo $files[$i].'<br />';
    }
}
```

The above function will take in a directory path and print out all the files from that directory. Notice how I used four different built-in functions: scandir(), array_diff(), array_values(), count() to accomplish the task. Don't worry about what they all mean at this point if you don't understand them.



PHP Array

Let's create an array variable which will hold four different numbers and one string type. Using the syntax from example in the info graphic above, we have:

```php
$numbers = array(42, 12, 1983, 7, "I'm a string");
print_r( $numbers ); //print_r() function is used to print the arrays
/* output: Array
(
[0] => 42
[1] => 12
[2] => 1983
[3] => 7
[4] => I'm a string
) */
```

Each number is identified by a key. The key count starts from zero, so the number 42 is identified by key [0]. If we wanted to print out the value 42, we would type:

```
echo $numbers[0]; //output: 42
```

Having unique key identifiers makes it easier to access array's values, just remember to always refer to the array's key when trying to access a particular value. If we tried to use echo on $numbers we would get a Notice from PHP, because it doesn't know what key/value pair we are trying to access:

```
echo $numbers; //Notice: Array to string conversion
```

We can also create numeric arrays by manually created the index keys and assigning values to them using the => rocket operator, instead of letting PHP automatically create keys for us.

```
//The following is identical to previous example: $numbers = array(42, 12, 1983, 7);
$numbers = array(0 => 42, 1 => 12, 2 => 1983, 3 => 7);
```

Associative Arrays

Associative arrays also known as named arrays have string keys instead of numeric keys. Let's use the example similar to the info graphic with some people's names

and ages as an example. Note the use of => (rocket) to assign array's key to it's value.

```
$people = array('Wallace' => 21, 'Victoria' => 22, 'Roman' => 33, 'Alex' => 35, 'Sarah' => 28);
print_r( $people );
/* output: Array
(
[Wallace] => 21
[Victoria] => 22
[Roman] => 33
[Alex] => 35
[Sarah] => 28
) */
```

And if we wanted to access one of the values, we would use the key name in the same way as numeric arrays, though in this case the key must be inside the quotes:

```
echo $people['Sarah']; //output: 28
```

Mixed Array

The array doesn't have to be either numeric or associative, it can be both:

```
$random = array('Keanu Reeves', 'movie' => 'The Matrix', 1999, 5 => "I'm confused", "Hi there");
print_r( $random );
/* output: Array
(
[0] => Keanu Reeves
[movie] => The Matrix
[1] => 1999
[5] => I'm confused
[6] => Hi there
) */
```

There are some things going on in the above example which may cause confusion. One important thing to remember is that the automatic array key indexing picks up the count from the highest numeric key. So when we used 'movie' as a string named key, the value 1999 go assigned the next number from 0 which is 1. And when we set the count to 5, by using 5 => "I'm confused", the next value "Hi there" got assigned a value of 6 which is obviously the next one after 5.

PHP If Else Statement

PHP if else statement is a conditional statement where specific code is executed if either condition is evaluated to true or false.

The shortest form of an if statement is using only if statement on it's own, when you want to match a condition and perform some action based on it, without capturing any additional conditions

```
$answer = 42;
if ( $answer == 42 )
{
echo "The ultimate answer is 42";
}
//output: The ultimate answer is 42
```

So if the conditional expression ($answer == 42) evaluates to true, the code between the curly brackets of an if statement is executed, and if it evaluates to false, the program continues running without printing anything.

If we wanted to capture the false condition, we would add an else to our if statement.

```
$answer = 50;
if ( $answer == 42 )
{
echo "The ultimate answer is 42";
}
else {
echo "You don't have life figured out yet! Keep trying!";
}
//output: You don't have life figured out yet! Keep trying!
```

Next, if we wanted to expand our if statement further and evaluate an additional condition, we would simply add an elseif.

```
$answer = 13;
if ( $answer == 42 )
{
echo "The ultimate answer is 42";
}
elseif ( $answer == 13 ) {
echo "Today is your lucky day!";
}

else {
```

```
echo "Your answer is just plain wrong!";
}
//output: Today is your lucky day!
```

So the first if condition checks if the $answer is 42, it's false, so it moves to the next elseif condition, this time it matches the number 13, so it prints the lucky message. You can add as many elseif conditions as you want to an if statement.

Additionally, you can add as many if statements within other if statements, in fact you can have infinite nested if statements, giving you a lot of flexibility to execute code based on all sorts of conditions:

```
$answer = 42;
$truth = 1;
if ( $answer == 42 )
{
echo "The ultimate answer is 42. ";
//using if statement in this manner assumes the expression is boolean ( 0 or 1)
if ( $truth ) {
echo "You speak the truth!;"
}
else {
echo "That is a lie!";
}
}
elseif ( $answer == 13 ) {


echo "Today is your lucky day! ";
if ( $truth ) {
echo "You speak the truth!;"
```

```
}
else {
echo "That is a lie!";
}
}
else {
echo "Your answer is just plain wrong!";
}
//output: The ultimate answer is 42. You speak the truth!
```

Alternative If Statement Syntax

There is a slightly different way to write the if statement, as with many PHP statements, which makes it easier to embed in HTML code. We simply remove all the curly brackets and use the colon instead, also an additional keyword endif is used at the end of the if statement.

```
<html>
<title>Welcome to My Web Page</title>
<body>
<?php if (condition): ?>
<!-- html code to run if condition is true -->
<?php elseif (condition): ?>
<!-- html code to run if elseif condition is true -->
<?php else: ?>
<!-- html code to run if all other conditions are false -->

<?php endif ?>
</body>
</html>
```

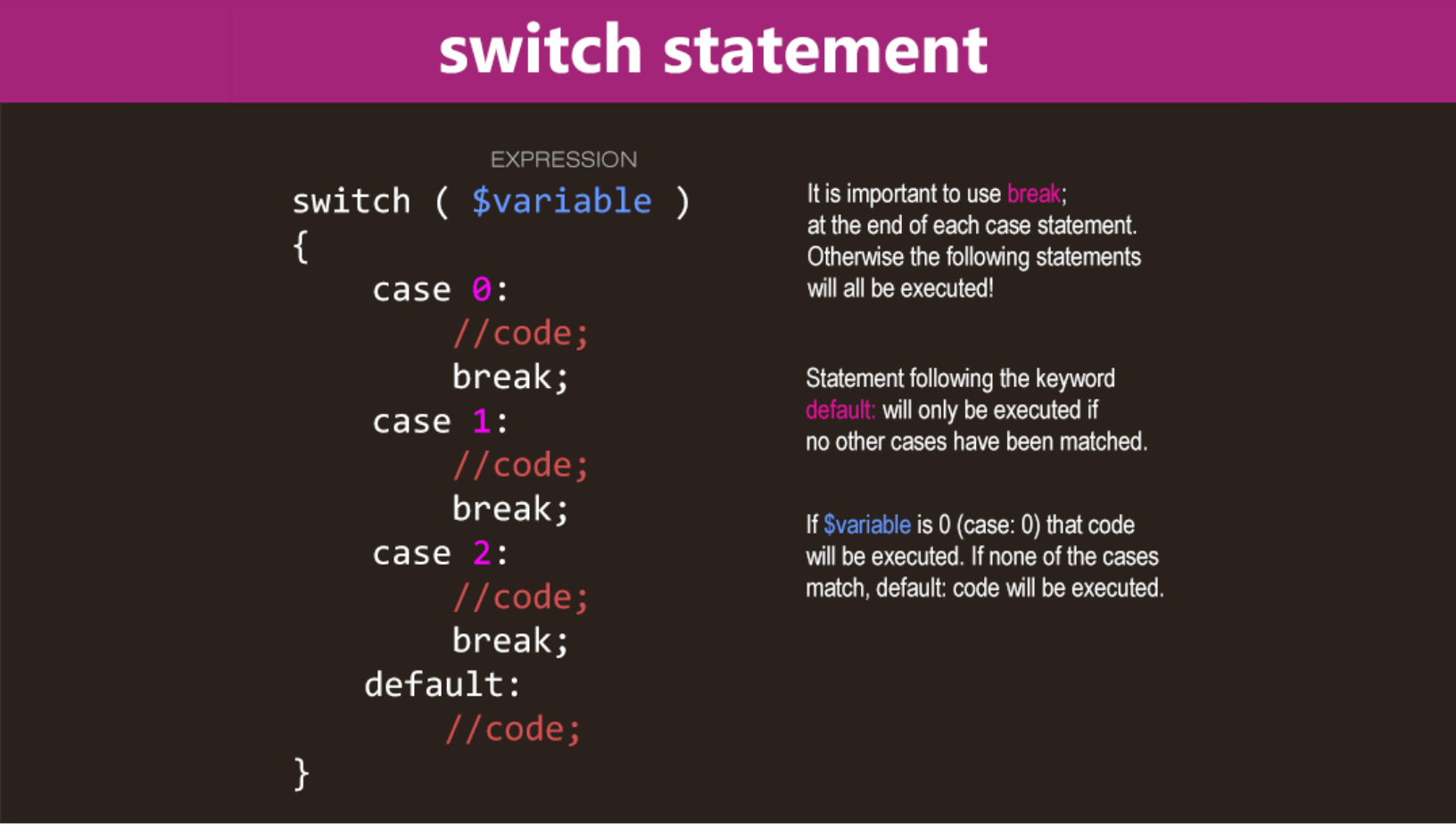
Ternary Operator

A ternary operator is a short version of an if else statement, there are no elseif in the ternary operator, it's just true or false. A good way to memorize this operator is this: Is this true? Yes : No. The statement consists of a question mark and a colon and is written on one line.

```
$are_you_okay = true;
echo $are_you_okay ? "Yeah I'm good" : "Leave me alone";
//output: Yeah I'm good
```

So if the condition is true, the code on the left side of the colon is executed, if it's false, the code on the right side would be run. In the example above, I used the echo statement to print out the result right back to the screen. Alternately, the result can be stored in a variable:

```
$are_you_okay = false;
$message = $are_you_okay ? "Yeah I'm good" : "Leave me alone";
echo $message;
//output: Leave me alone
```



PHP Switch Statement

A PHP switch statement is a conditional statement used to execute different code based on what the conditional expression evaluates to. The keyword "case" followed by the value is used to match the values from the conditional expression.

```
$animals = array('Cat', 'Dog', 'Duck', 'Cow', 'Cricket', 'Hyena',
'Chinchilla');
//conditional expression
switch ( $animals[array_rand( $animals )] )
{
case 'Cat': echo "Meow!";
break;
case 'Dog': echo "Woof!";
break;
case 'Duck': echo 'Quack!';
```

```
break;
case 'Cricket': echo 'Chirp!';
break;
case 'Hyena': echo 'He-he-he-he!';
break;
case 'Chinchilla': echo 'Squeak!';
break;
case 'Cow': echo 'Moooooo!';
break;
}
```

I used array_rand() function above which picks out a random key index within the array. I then put the resulting value returned from array_rand() inside of $animals[ ] array's square brackets, so the final result is a random value from the $animals array. Note that I didn't create another variable holding the random value returned from array_rand() function, but rather put the entire function inside of square brackets right away. This is perfectly okay today in PHP and results in a cleaner, shorter code.

Default Keyword

Switch statement can also have default keyword which will execute the code if no cases have been matched:

```
switch ( $animals[array_rand( $animals )] )
{
case 'Cat': echo "Meow!";
break;
case 'Dog': echo "Woof!";

break;
default: echo "Unknown animal detected! Run for your life!";
```

```
}
```

Switch Without Breaks

You can also omit using break; after the case statement. Doing so will continuing passing the case match to the next one, until it reaches the end of a switch statement:

```
$speed = '65mph';
switch ( $speed )
{
case '65mph': echo "You're going the speed limit, responsible driver!";
case '95mph': echo "Whoa there, slow down, you're way over the speed limit!";
case '135mph': echo "Did you forget about your breaks?";
case '250mph': echo "You must be driving Bugatti";
}
/* output:
You're going the speed limit, responsible driver!
Whoa there, slow down, you're way over the speed limit!
Did you forget about your breaks?
You must be driving Bugatti */
```

Case Grouping

It's possible to group multiple case matches by having an empty case condition, which will pass the control over to the next case:

```
$species = array('Cat','Dog','Cow','Fly','Bee','Ant');
switch ( $species[array_rand( $species )] )
{
case 'Cat':
case 'Dog':
case 'Cow':
 echo 'We are animals!';
break;
case 'Fly':
case 'Bee':
case 'Ant':
 echo 'We are insects!';
break;
}
```

Alternative Switch Case Syntax

PHP also supports an alternative syntax for switch statement, so instead of using the curly brackets we can replace it with a colon (:) and endswitch; which will look like this:

```php
switch ( $i ):
case 0:
echo "i equals 0";
break;
case 1:
echo "i equals 1";
break;
case 2:
echo "i equals 2";
break;
default:
echo "i is not equal to 0, 1 or 2";
endswitch;
```

Switch vs Else If

One important difference to note between switch case and else if is that in the switch case the conditional expression is only evaluated once. On the other hand, in an else if statement the condition keeps being evaluated.

```php
$number = 5;
switch ( $number ) //condition is only evaluated once
{
case 5: echo "Number 5 matched!";
break;
case 3: echo "Number is less than 5!";
break;
case 4: echo "Number is less than 5!";
break;
}
```

```
//output: Number 5 matched!
if ( $number > 5 ) //condition evaluated first time
{
echo "Number is greater than 5!";
}
elseif ( $number < 10 ) //condition evaluated second time
{
echo "Number is less than 10!";
}
//output: Number is less than 10!
```

Practical Switch

The following function uses switch statement and will return the name of the animal based on the year provided.

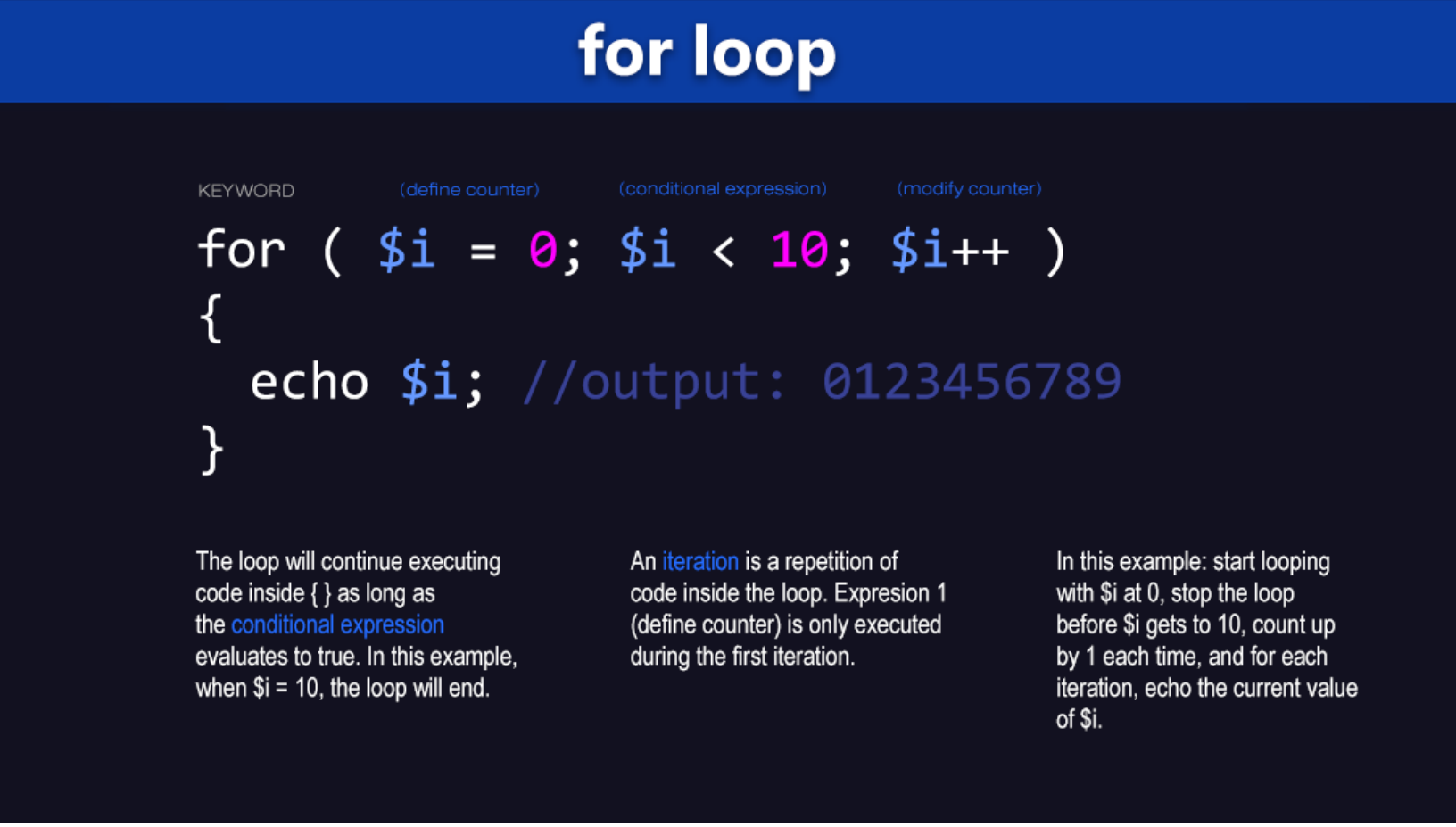
```
//code credit: Anonymous
function getChineseZodiac($year)
{
switch ($year % 12) :
case 0: return 'Monkey'; // Years 0, 12, 1200, 2004...
case 1: return 'Rooster';
case 2: return 'Dog';
case 3: return 'Boar';
case 4: return 'Rat';
case 5: return 'Ox';
case 6: return 'Tiger';
case 7: return 'Rabit';
case 8: return 'Dragon';

case 9: return 'Snake';
case 10: return 'Horse';
```

```
case 11: return 'Lamb';
endswitch;
}
echo getChineseZodiac(2017); //output: Rooster
```

And another cool one for easy Nobel prize winning:

```
//code credit: MaxTheDragon at home dot nl
function winNobelPrizeStartingFromBirth( $subject )
{
switch( $subject )
{
case "peace": echo "You won the Nobel Peace Prize!"; break;
case "physics": echo "You won the Nobel Prize in Physics!"; break;
case "chemistry": echo "You won the Nobel Prize in Chemistry!"; break;
case "medicine": echo "You won the Nobel Prize in Medicine!"; break;
case "literature": echo "You won the Nobel Prize in Literature!"; break;
default: echo "You bought a rusty iron medal from a shady guy who insists it's
a Nobel Prize..."; break;
}
}
winNobelPrizeStartingFromBirth("chemistry");
//output: "You won the Nobel Prize in Chemistry!"
```



PHP For Loop

PHP for loop is the most common type of loop used in PHP which keeps executing code until a conditional expression evaluates to false. The loop usually consists of three expressions: 1) define the counter 2) conditional expression (evaluate the counter) 3) modify the counter - which are separate by a semi-colon.

```
for ( $c = 0; $c <= 10; $c++ )
{
echo $c; //output: 012345678910
}
```

You may be wondering why, $c is not reset to zero, every time this loop runs. That's because that first expression is only run once in a for loop, during the first

iteration. The second and third expressions, however, are run continuously during every iteration.

Looping Through Numeric Arrays

For loop is very useful, if you want to loop through a numeric array:

```
$animals =
array('Lion','Zebra','Moose','Okapi','Panther','Penguin','Siberian
Tiger','Raven');
//count() function returns the number of elements in an array
for ( $i = 0; $i < count( $animals ); $i++ )
{
echo $animals[ $i ].', ';
}
//output: Lion, Zebra, Moose, Okapi, Panther, Penguin, Siberian Tiger,
Raven,
```

Remember that numeric array keys are automatically created when we don't specify them.

We used count() function to get the number of elements in an array, so that we know when to stop our loop.

You may have noticed, there is an unwanted comma at the end of our animal list after Raven. We can use another array function called end() to check if we are at the last array value and combine that with conditional expression to remove the comma from the last animal name:

```
//loop ends when $i equals 8
for ( $i = 0; $i < count( $animals ); $i++ )
{
//are we at the end of $animals array ? Yes print dot : No print the comma
echo $animals[$i] == end($animals) ? $animals[$i] : $animals[$i].', ';
}
//output: Lion, Zebra, Moose, Okapi, Panther, Penguin, Siberian Tiger,
Raven.
```

Using Breaks to End For Loop

The syntax of the for loop is not as strict as it seems. For example, we can choose to leave out the conditional expression and use break; inside the loop instead to let PHP know when to end our loop. The following would be equivalent to previous example:

```
for ( $i = 0; ; $i++ )
{
//loop ends when $i equals 8
if ($i == count( $animals )) {
break;
}
echo $animals[$i] == end($animals) ? $animals[$i] : $animals[$i].', ';
}
```

There is one thing that I think might be confusing about looping through numeric arrays from the examples above, if you're new to this. The numeric array index keys start their count from 0, but the count() function starts its count from 1. So in

our $animals array, we have keys that starts with 0, and the last key is 7. Here is what it would look like if we wrote numeric keys manually:

```
$animals = array(0 => 'Lion', 1 => 'Zebra', 2 => 'Moose', 3 => 'Okapi', 4 => 'Panther', 5 => 'Penguin', 6 => 'Siberian Tiger', 7 => 'Raven');
echo count($animals); //output: 8
```

So the count() function returns 8 because it starts it's count from 1 but we have last array key which is 7. That's why we want to end our loop when $i reaches 8, since there is no key 8 in our array.

Looping Through Letters

We can use for loop to loop through letters as well:

```
for ( $letter = 'a'; $letter != 'z'; $letter++ ) {
    echo $letter.' ';
}
//output: a b c d e f g h i j k l m n o p q r s t u v w x y
```

When looping through letters in this manner, the <= will not produce the desired result, so make sure to use != (not equal) operator instead. Also, I haven't found a way to loop through the alphabet backwards using the for loop, but we can use a foreach loop and range() function to accomplish that easily:

```
foreach (range('z', 'a') as $letter ) {
    echo $letter.' ';
}
//output: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

Optimizing For Loop

Let's re-examine our earlier example where we looped through $animals array and used count() function to know when to end our loop.

```
for ( $i = 0; $i < count( $animals ); $i++ )
{
//print out animals
}
```

How can we optimize this loop to make it faster and more efficient? Well, we already know that second and third expressions are executed during every loop iteration. The second expression which is count($animals); uses count() function which is called repeatedly during every iteration. We can avoid having to run the count() function repeteadly within the loop, by placing it outside the loop and storing our count value in a variable:

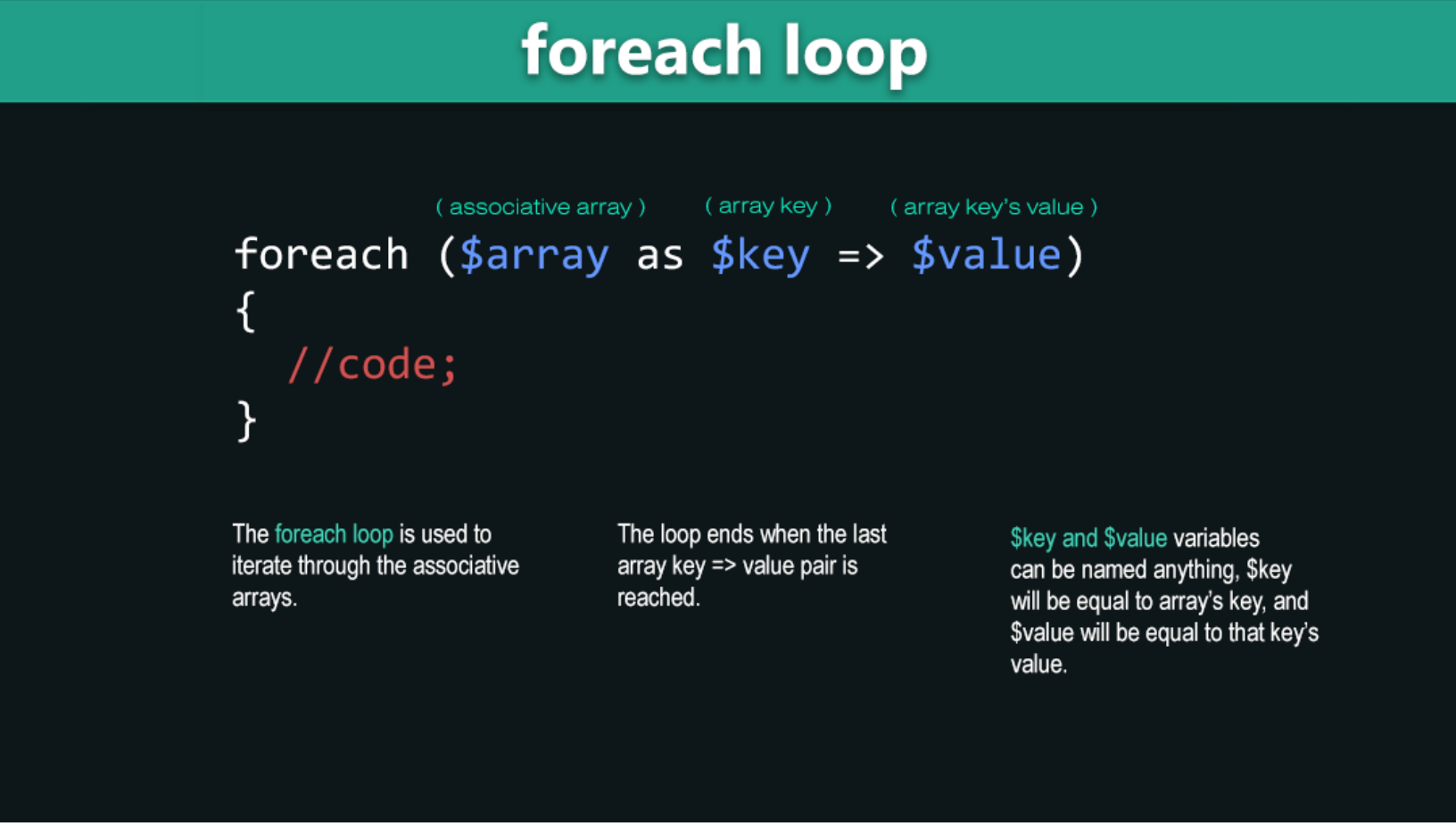
```
$array_count = count( $animals );
for ( $i = 0; $i < $array_count; $i++ )
{
//print out animals
}
```

Doing so will make our code more efficient because count() function is only run once instead of 8 times.

For Loop Practical Example

Here is a function that takes two parameters, $start_date and $end_date and loops through every day using a for loop:

```
function everyday($start_date, $end_date) {
for ($date = strtotime($start_date); $date <= strtotime($end_date); $date =
strtotime("+1 day", $date)) {
echo date("Y-m-d", $date).""; }
}
everyday("2017-01-01","2017-01-09");
/* output:
2017-01-01
2017-01-02
2017-01-03
2017-01-04
2017-01-05
2017-01-06
2017-01-07
2017-01-08
2017-01-09 */
```



PHP Foreach Loop

Foreach loop is used to loop through associative arrays in PHP, the loop ends when the last array key/value pair is reached. Let's create an associative array with animal names as keys and sounds those animals make as values.

```
$animals = array
(
'Antelope' => 'Snorts',
'Bat' => 'Screeches',
'Bear' => 'Roars',
'Dolphin' => 'Clicks',
'Goose' => 'Honks',
'Vulture' => 'Screams'
);
```

We can then create foreach loop to iterate through all the key/value pairs in the array:

```
foreach ( $animals as $name => $sound )
{
//$name is key, $sound is value
echo $name.' '.$sound;
}
/* output:
Antelope Snorts
Bat Screeches
Bear Roars
Dolphin Clicks
Goose Honks
Vulture Screams
*/
```

We can also use $animals[$name] in place of the $sound, since the $animals is our actual array and $name represents the key and array [$key] gets us the value. The following would be the same as previous example:

```
foreach ( $animals as $name => $sound )
{
//$name is key, $sound is value
echo $name.' '.$animals[ $name ];
}
```

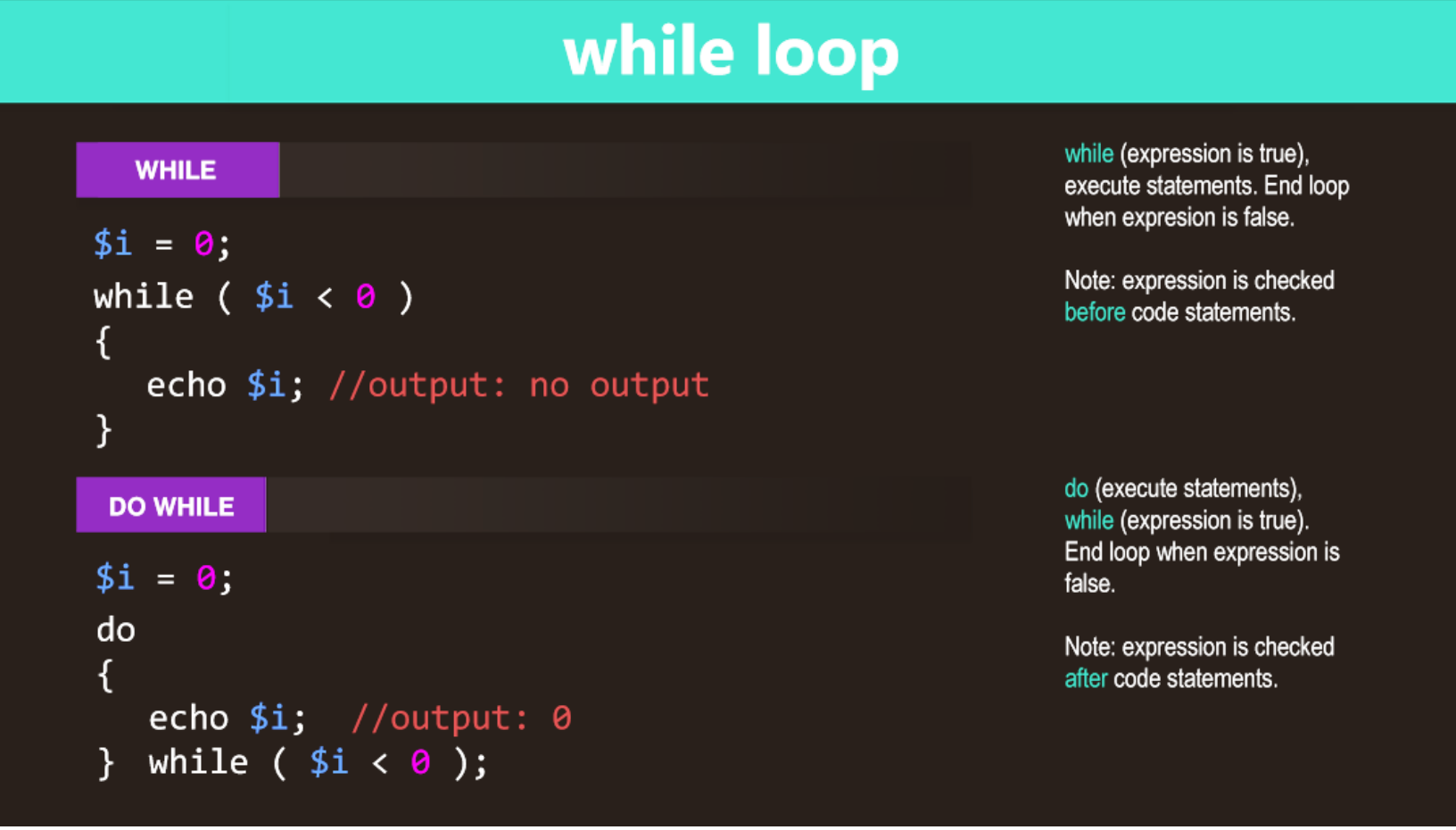
Alternately, we can also omit part of the loop if we were looking for values only, without needing the keys:

```
foreach ( $animals as $value )
{
echo $value;
}
/* output:
Snorts
Screeches
Roars
Clicks
Honks
Screams
*/
```

Alternative Foreach Syntax

As with many other loops and conditional statements, PHP allows for alternative foreach syntax, by removing curly brackets and using colon and endforeach; instead.

```
<html>
<title>Welcome to My Web Page</title>
<body>
<?php foreach ( $animals as $name => $sound ) : ?>
<!-- do something -->
<?php endforeach; ?>
</body>
</html>
```



PHP While Loop

In a PHP while loop, an expression is evaluated before the statements. In a do while loop, however, the expression is evaluated at the end of the loop.

Let's create an array and keep looping until we reach the end of the loop so we can print out all the values of the array:

```
$animals = array('Antelope','Bat','Bear','Dolphin','Goose','Vulture');
$key = 0;
$total = count( $animals );
//using while loop
while ( $key < $total ) {
echo $animals[ $key ].' ';
$key++;
}
```

```php
//output: Antelope Bat Bear Dolphin Goose Vulture
//using do { } while loop
$key = 0;
do {
echo $animals[ $key ].' ';
$key++;
} while ( $key < $total );
//output: Antelope Bat Bear Dolphin Goose Vulture
```

class & object

```php
class className
{
    //variable scope must be provided (public, protected or private)
    public $variableName = value;
    function functionName()
    {
        //code;
    }
}

$obj = new className; //$obj variable becomes object data type
$obj->functionName();
$obj->variableName;
```

The new keyword creates object instance of the class. The object inherits all of classes's variables (properties) and functions (methods).

Classe's variables are called properties and functions are called methods.

Once the object instance of the class is created, we can access properties and methods using -> (dart).

PHP Classes and Objects

PHP class is a template for an object. To give an analogy from a real life example, if a class was "Animal", then we could create an object $tiger from that class, which is an animal that exists in real life.

```
class Animal { }
$tiger = new Animal; //$tiger is an object instance of Animal class
```

So, to continue from the template Animal, we could ask questions like, what does every animal have and what can every animal do? The haves will turn into properties and could be something like animal's prey, and the dos will become methods or actions which every animal can perform like run or make a sound. Knowing some things about animals, we can incorporate them into our Animal class template:

```
class Animal
{
//we don't set any value for the properties because every animal will be different
public $prey;
public $max_speed;
function make_sound( $sound )
{
echo $sound;
}
}
//we can now create the tiger object from class Animal
//and set values that are specific to tiger
$tiger = new Animal;
```

```
$tiger->prey = "deer";
$tiger->max_speed = 40;
echo "I'm an animal that loves to eat $tiger->prey. The fastest I can go is
$tiger->max_speed mph";
$tiger->make_sound("Roar....");
//output: I'm an animal that loves to eat deer. The fastest I can go is 40
mph. Roar....
```

The properties and methods of the class become available to our variable object $tiger when we create the instance of the object with the "new" keyword followed by the class name (Animal), and we are able to access them using the -> (dart) operator, also known as the object operator.

Pseudo Variable $this

Make sure you really understand the difference between a class and an object before moving on, otherwise $this following concept will be confusing.

$this refers to the object instance inside of a class

```
class Animal
{
//set default property value, just for fun
public $sound = "Cows don't growl. ";
function make_sound( )
{
//$this = $cow object ($cow->sound)
echo $this->sound;
}
}
$cow = new Animal;
echo $cow->sound;
$cow->sound = 'Moooooooo';
```

```
$cow->make_sound();
//output: Cows don't growl. Moooooooo
```

Okay...so what happens is we create the object $cow from the Animal class and we set it's sound property to 'Moooooooo'. We are then able to access $cow object from within Animal class using $this pseudo variable. By using $this, we no longer need to pass any parameters into make_sound() method. It's like magic!

It's also perfectly okay to set default property values inside of a class as shown in the above example, which is later overwritten with $cow->sound = 'Moooooooo'; line.

It's also possible to set the value of $this->sound property from within the class method as follows:

```
class Animal
{
public $sound;
function set_sound( $sound )
{
$this->sound = $sound;
}
function make_sound( )
{
echo $this->sound;
}
}
$cat = new Animal;
$cat->set_sound( "Meow" );
//we can then either echo out the $sound property or call make_sound() method
echo $cat->sound;
$cat->make_sound();
```

```
//output: MeowMeow
```

Property Setters and Getters

In a similar way we have "set" the property $sound to a value which was specified outside of the class using the method set_sound(), we can also get the value of the $sound by creating a method called get_sound();

```
class Animal
{
public $sound;
function get_sound( )
{
return $this->sound;
}
function set_sound( $sound )
{
$this->sound = $sound;
}
}
$mouse = new Animal;
$mouse->set_sound("Squeak");
echo $mouse->get_sound();
//output: Squeak
```

Magic PHP Methods Get and Set

Time for some real magic now! The right way to create getters and setters is to use PHP magic methods called __get() and __set() (that's two underscores before method name). Doing so will let us get and set any property, instead of creating

these methods for every property which would be redundant, and redundancy is evil, especially in object-oriented programming! I hope this won't look too scary, if it does, don't worry I'll explain everything.

```
class Animal
{
//private keyword makes properties available only inside this class
private $sound;
private $max_speed;
public function __get($property)
{
//check if property exists
if (property_exists($this, $property)) {
return $this->$property;
}
}
public function __set($property, $value)
{
if (property_exists($this, $property)) {
$this->$property = $value;
}
return $this;
}
}
$monkey = new Animal;
//property name is provided in a string format, without using the dollar sign
$monkey->__set( 'sound', 'Scream' );
echo $monkey->__get( 'sound' ); //output: Scream
$monkey->__set( 'max_speed', 34 );
echo $monkey->__get( 'max_speed' ); //output: 34
//trying to access private property outside the class:
```

```
echo $monkey->sound; //Fatal Error
```

This should look somewhat familiar, though there are a few important differences. First of all, we are declaring our properties using the "private" scope keyword, which makes direct property access available only within the context of the Animal class. The reason for this is good programming design, since we're going to be setting and getting the value of the property using methods, it doesn't make sense to also have direct access to those properties outside of the class.

Second, we are checking if the property exists using an if statement and a self explanatory function called property_exists( ), which takes in two parameters - $this (object) and $property (property name)

Lastly, we're using __get() and __set() PHP magic methods which makes it possible to set and get the value of any property we create in the future.

PHP Session

A session is a way to store information across multiple pages. In PHP, that information is stored in a $_SESSION global variable. session_start(); function is used to create new session that will be used on different pages.

Let's create two php pages and see how we can set and access session variables across multiple pages.

```php
//page-one.php
session_start(); //create new session
//set session variables in a $_SESSION global array variable
$_SESSION[ 'username' ] = 'worker';
$_SESSION[ 'role' ] = 'author';
print_r( $_SESSION );
```

```
//output: Array ( [username] => worker [role] => author )
```

We can then access session variables on another page:

```
//page-two.php
session_start(); //must be used on all pages that use session
print_r( $_SESSION );
//output: Array ( [username] => worker [role] => author )
//change session variables
$_SESSION[ 'username' ] = 'clevertechie';
$_SESSION[ 'role' ] = 'admin';
print_r( $_SESSION );
//output: Array ( [username] => clevertechie [role] => admin )
session_destroy(); //note: this will clear $_SESSION next time it's accessed
```

When we try to print out $_SESSION on the next page, we get an empty array, because the session was destroyed with session_destroy() on previous page.

```
//page-three.php
session_start(); //create new session
print_r( $_SESSION );
//output: Array ( )
```

```
//create mysql connection variables
$host = 'localhost';
$user = 'clevertechie';
$pasword = 'mypass123';
//connect to mysql by creating new object ($connection) from mysqli class
$connection = new mysqli( $host, $user, $password );
//create the database by running sql query
$connection->query("CREATE DATABASE articles");
```

By using new mysqli, we create the new object $connection. We can then use all available methods on the object such as query() to run SQL commands on our database connection.

PHP MySQL

By using new mysqli, we create the new object $connection. We can then use all available methods on the object such as query() to run SQL commands on our database connection.

Create MySQL database

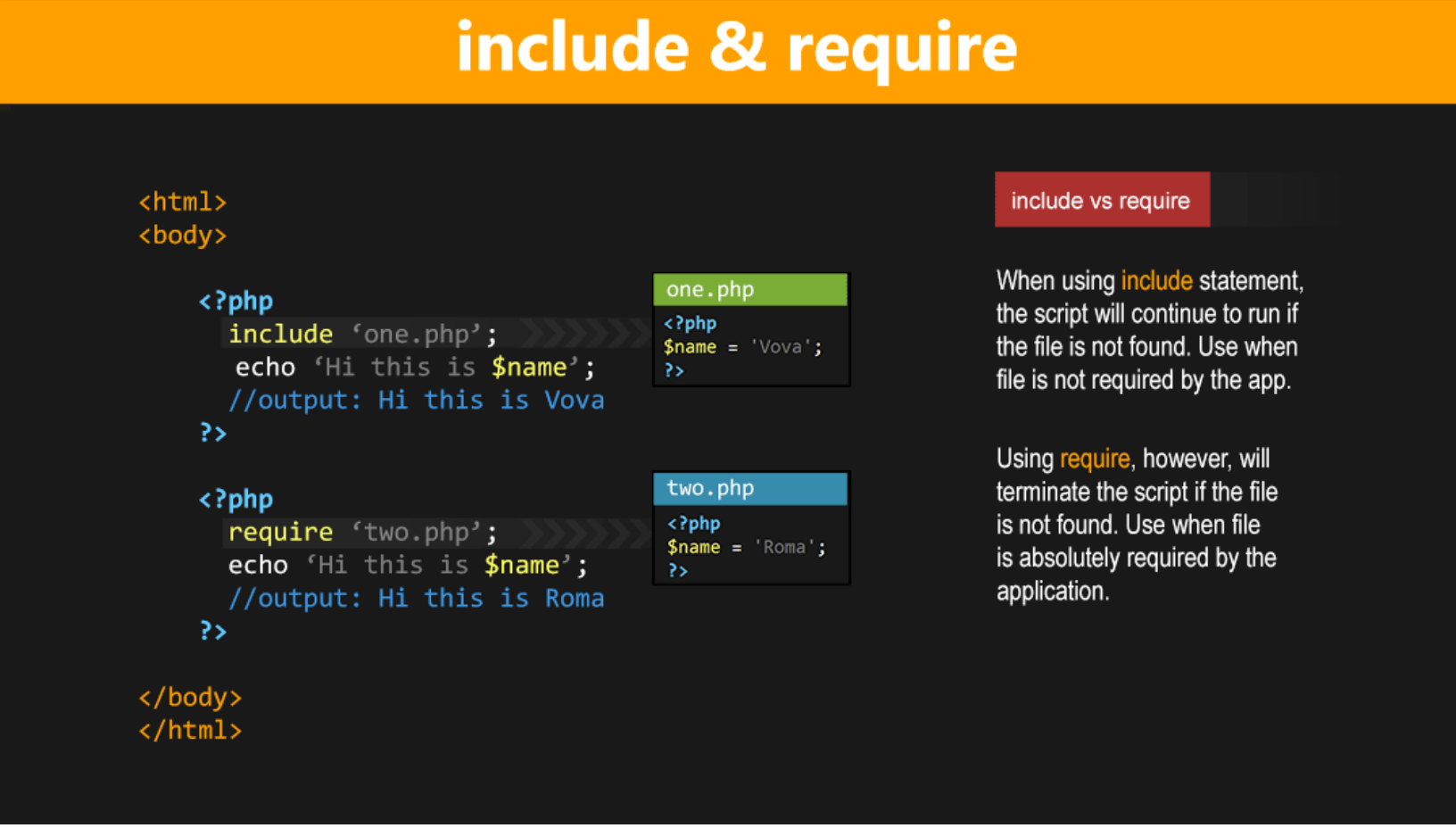
We must pass at least three parameters to mysqli - $host, $user and $password, so we can successfully connect to our database. Let's create the connection:

```
$host = 'localhost';
$user = 'root';
$password = 'pass123';
//$connect becomes object instance type of mysqli class
$connect = new mysqli( $host, $user, $password );
```

```
//we can now run methods on our $connect object using the dart operator
$connect->query("CREATE DATABASE pages");
```

If we then connect to mysql via the command prompt console, and type "show databases;", we should see something that looks similar to the following if everything went well:

```
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| pages |
+--------------------+
1 rows in set (0.00 sec)
```



PHP include and require

When another PHP file is included using include or require keyword, all variables which were declared in another PHP file that is being included will become available from that point forward, from the line where the include statement was used.

```php
//game.php
$name = 'The Witcher 3';
$cost = 35.60;
$developer = 'CD Projekt RED';
//welcome.php
echo "Welcome to our PC video game review web site";
include( 'game.php' );
//game.php's variables become available from this point forward
```

```
echo "Today's featured game is $name, developed by $developer. You may purchase the game at amazon.com for $$cost.";
//output:
//Today's featured game is The Witcher 3, developed by CD Projekt RED. You may purchase the game at amazon.com for $35.60.
```

Including File with Functions and Classes

Unlike variables, where they only become available from the line where they have been included, functions and classes can be accessed anywhere within the file when included because they'll have global scope.

```
//game.php
function print_game( $game ) {
foreach ( $game as $key => $value ) {
echo $key.' : '.$value.'<br />';
}
}
//welcome.php
include ( 'game.php' );
$game = array('name' => 'The Witcher 3', 'cost' => 35.60, 'developer' => 'CD Projekt RED');
print_game ( $game );
/*output:
name : The Witcher 3
cost : 35.6
developer : CD Projekt RED */
```

Including From Within a Function

When a file is included from within a function, all variables that are defined in the file that is being included behave as if though they have been declared inside that function, so they will have local scope of that function, that is unless we use keyword global on the variable that is being included.

```
//game.php
$name = 'The Witcher 3';
$developer = 'CD Projekt RED';
//welcome.php
function welcome ( )
{
global $developer; //will have access outside of function
include 'game.php';
echo "$name by $developer";
}
welcome(); //output: The Witcher 3 by CD Projekt RED
echo "$name by $developer"; //output: by CD Projekt RED
```

Include vs Require

When using include statement, the script will continue to run if the file is not found. Use when file is not required by the app.

```
<html>
<body>
<?php include('file-not-found.php');
//program continues running
```

```
?>
</body>
</html>
```

Using require, however, will terminate the script if the file is not found. Use when file is absolutely required by the application.

```
<html>
<body>
<?php require('file-not-found.php');
//Fatal error, program stops
?>
</body>
</html>
```

Include From Anywhere

As you move your project files around, it may become messy and time consuming, having to change your include paths every time you move your files. For that, there is a simple script which uses the $_SERVER global array and the DOCUMENT_ROOT key which gets the current working root directory:

```
//code credit: snowyurik at gmail dot com
include $_SERVER['DOCUMENT_ROOT']."/lib/sample.lib.php";
```

operators

+$a	Identity	Conversion of $a to int or float as appropriate.
-$a	Negation	Opposite of $a.
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulo	Remainder of $a divided by $b.
$a ** $b	Exponentiation	Result of raising $a to the $b'th power. Introduced in PHP 5.6.

PHP Operators

Most of the operators from above table should be self explanatory, so I'm not going to cover them. Though I do want to go over "Identify" since it may not be so obvious.

Identity Operator

What the identity operator will do is convert a string float or integer value to an actual float or integer value. Here is what I mean by string float or integer:

```
$string_float = '3.5';
var_dump($num); //output: string(3) "3.5"
$string_integer = '100';
var_dump($num); //output: string(3) "100"
```

As you can see both variables from above are of string data type when output using var_dump() function. Let's now use the identity operator on both to see what happens:

```
$string_float = '3.5';
var_dump(+$num); //output: float(3.5)
$string_integer = '100';
var_dump(+$num); //output: int(100)
```

PHP has now converted both strings to their appropriate types: float and integer. This operator is designed for converting valid string values of integer and float which are enclosed in quotes to their appropriate values and there will be situations when it may be useful.