BASICS

PHP tag	`<?php code goes here; ?>`

Printing text	`print 'Text to be displayed';`
	`echo 'Text', 'to', 'be', 'displayed';`

Declare a variable	`$varName = value;`

String formatting
(in double-quotes)
	`"Display the variable $varName";`
	`"Display the variable " . $varName;`

Write formatted string to var	`$var = sprintf("You have %d %s left",$amount,$currency);`
Print formatted string	`printf("Value: %s") (String = %s)(Int = %d)(Float = %f)`

Type casting (Explicit)
Type casts:
	`$varName = (dataType)$vartochange;`
	`(int)(double)(float)(bool)(string)(array)(object)(unset)=null`

Global variable (Accessible anywhere in code)	`global $varName;`
	`$GLOBALS['varName'];`

Suppress error message	`@ (Ex: $varName = @$_GET['name'];)`

Create a constant (**case-ins**)	`define(CONSTANT_NAME, "constant_value", [true/false])`
	`const CONTANT_NAME = value;`

Comments	`# single-line // single-line /* multi-line */`

BUILT-IN FUNCTIONS

STRING FUNCTIONS	
Return length of string	`strlen($strName)`
Return number of words	`str_word_count($strName)`
Reverses string	`strrev($strName)`
Searches string, returns index, starting at index	`strpos($strName, "str_to_find", #)`
Replace x with y in str	`str_replace("x", "y", $str)`
NUMERIC FUNCTIONS	
Formats number with comma, optional rounding	`number_format($number, [#])`
Returns random number between min and max	`rand/mt_rand(min, max);`
VARIABLE FUNCTIONS	
Returns information on the variable	`var_dump($var)`
Returns the data type of the variable	`gettype($var)`
Returns true if var is null	`isset($var)`
Returns true if var is null, not set, or an empty string	`empty($var)`
Returns true if var is or can be converted to a number	`is_numeric($var)`
DATE FUNCTION date(format)	
Returns current date in given format (Ex: 16/09/18)	`$date = date('d/m/y')`

OPERATORS

<	Less than	>	Greater than
<=	Less than or equal to	>=	Greater than or equal to
Equality Operators (Type-Coercion)		**Identity Operators (Same type & value)**	
==	Equal value	===	Equal value and type
!= / <>	Not equal to	!==	Not equal value and type
Logical Operators		Example:	
!	Not (Returns opposite Boolean)	`echo !is_numeric($number)`	
&&	And (Both expressions are true)	`$age >= 18 && $score >= 680`	
||	Or (Either expression is true)	`$state == 'CA' || $state == 'NC'`	

SELECTION

if, else if, else

```
if (condition) {
    code to be run if true;
} else if (condition) {
    code to be run if true;
} else {
    to be run if false;
}
```

ternary operator

```
(condition) ? valid_if_true
: value if false;
```

using flags with constants

```
define("_ADMIN_", 1);
define("_STUDENT_", 2);
$role_id = 1;
if($role_id == _ADMIN_) {...
```

switch statement

```
switch (expression) {
 case 'this':
  code to be run if true;
  break;
 case 'this':
  code to be run if true;
  break;
}
```

ITERATION

while loop

```
while (condition) {
    code run while condition = true;
    increment/decrement; } / endwhile
```

do-while loop

```
do {
    code to be executed at least once
} while (condition for code to be run);
```

for loop

```
for (initialization; condition; increment/decrement) {
    code to be run while condition is true;
} / endfor;
```

foreach loop (arrays)

```
foreach ($arrName as $item) {
    echo "$item <br>"; }
```

```
foreach ($arrName as $item):
    …
endforeach;
```

foreach loop (associative arrays)

```
foreach ($arrName as $k => $v)
    echo "Key=" . $k . ", Value=" . $v;
```

```
foreach ($arrName as $k => $v):
    echo "Key=" . $k . ", Value=" . $v;
endforeach;
```

ARRAYS

Creating an array	`$arrName = [];`	`$arrName = array();`
Create array with values	`$arr = ['val', 'val2'];`	`$arr = array('val1', 'val2');`
Assign an item to index	`$arrName[index] = value;`	`$aSarrName['key'] = value;`
Assign an item to the end	`$arrName[count($arrName) - 1]`	
Delete item from array	`unset($arrName[index]);`	`unset($aSarrName['key']);`
Delete entire array	`unset($arrName)`	
Sort array (asc)/(desc)	`sort($arrName); / rsort($arrName);`	
String Formatting	`"First Item: $arrName[0]";`	`"First Item: {$arrName[0]}";`

Array cursor

Get index of current element	`key($arrName);`	Move cursor to last element	`end($arrName);`
Move cursor to next element	`next($arrName);`	Move cursor to first element	`reset($arrName);`

Associative Array & N-Dimensional Arrays

Create & initialize values	`$arrName = array('key' => 'value', 'key2' => 'value2');`
	`$arrName = ['key' => 'value', 'key2' => 'value2'];`	

2-Dimensional	`var_name = [`
	` [r1c1, r1c2, r2c3],`
	` [r2c1, r2c2, r2c3]`
	`];`
		`$ar = array(`
		`  array(r1c1, r1c2, r2c3),`
		`  array(r2c1, r2c2, r2c3)`
		`);`

N-Dimensional	`$arr = [`
	`  [[d1r1c1, d1r1c2],`
	`   [d1r2c1, d1r2c2]],`
	`  [[d2r1c1, d2r1c2],`
	`  [d2r2c1, d2r2c2]]`
	`];`
		`$arr = array(`
		`  array(array(d1r1c1, d1r1c2),`
		`   array(d1r2c1, d1r2c2)),`
		`  array(array(d2r1c1, d2r1c2),`
		`   array(d2r2c1, d2r2c2))`
		`);`

ARRAY FUNCTIONS

Returns number of values	`count($arrName); / sizeof($arrName);`
Delete last item	`$arrName = array_pop($arrName);`
Delete first item	`$arrName = array_shift($arrName);`
Add item to end	`$arrName = array_push($arrName, value);`
Add item to beginning	`$arrName = array_unshift($arrName, value);`
Remove gaps in array	`$arrName = array_values($arrName);`
Concatenate arrays	`$newArray = array_merge($arrayOne, $arrayTwo);`
Array from lo to hi, by step	`$arrName = range($lo, $hi, [$step])`
Slices array from index to len	`$arrName = array_slice($arr, $index, [$len, $keys])`
Replaces arr with new, from i	`array_splice($arr, $i, [$len, $new])`
String to array (sep by del)	`$arrName = explode("del", $strName);`

FUNCTIONS

```
function function_name([optional parameters]) {
    code to be run when function is called;
    optional return statement;
}
```

Variable length parameter

```
function var_list(...$var) {
  return var_dump($var);
}
```

Anonymous function

```
$times_two = function($num) {
    return $num * 2;
};
```

Parameter with default value

```
function add_3($num = 1) {
  return num + 3;
}
```

value & reference

Passed by value: By default, PHP sends a copy of the argument to the function, not the argument itself.

Passed by reference: PHP sends a pointer to the original variable, changing the value. Code a **&**

Argument passed by value

```
function add_1($num) {
  $num += 1;
  echo '<p>Number: ' . $num . '</p>';
}
$number = 1;
add_1($number);    //Displays 2
echo $number       //Displays 1
```

Argument passed by reference

```
function add_1(&$num) {
  $num += 1;
  echo '<p>Number: ' . $num . '</p>';
}
$number = 5;
add_1($number);    //Displays 2
echo $number       //Displays 2
```

variable scope

Global Variable: declare variable outside a function, can be accessed using the **`global`** keyword

Local Variable: declare variable inside a function

A variable with global scope

```
$a = 10;
  function show_a() {
    echo $a;
}
show_a();   //Displays nothing
```

A variable with local scope

```
function show_b() {
  $b = 10;
  echo $b;
}
echo $b   //Outside function, $b is null
```

Accessing a global variable within a function:

```
$c = 10;                   //$c has global scope
function show_c() {
    global $c;             //$c now refers to the global variable $c
    echo $c;
}
show_c();                  //Displays 10
```

FORMS

superglobal variables	
Array sent by the HTTP GET method, to collect values from a form	`$_GET`
Array sent by the HTTP POST method, to collect values from a form	`$_POST`
Used to collect data after submitting an HTML form (Like get/post)	`$_REQUEST`
Holds information about headers, paths, and script locations	`$_SERVER`
Returns the filename of the currently executing script.	`$_SERVER["PHP_SELF"]`
Returns the request method used to access the page (GET or POST)	`$_SERVER["REQUEST_METHOD"]`

example:

`<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">`
`  Name: <input type="text" name="fname">`
`  <input type="submit">`
`</form>`
	Form is run on the same page

`<?php`
`if ($_SERVER["REQUEST_METHOD"] == "POST") {`
`    $name = htmlspecialchars($_REQUEST['fname']);`
`    if (empty($name)) {`
`        echo "Name is empty";`
`    } else {`
`        echo $name;`
	If the server request method is POST:
	the fname property is displayed.
	($_REQUEST could be replaced with $_POST)

PHP FILES

The **`include`** or **`require`** statement copies the content of one file and inserts it into another PHP file.

Inserts and runs the specified file. Shows a warning if it fails.	`include $path/($path)`
Same as include, but makes sure that the file is included only once.	`include_once($path)`
Similar to include, but if it fails, it causes an error that stops the script.	`require($path)`
Same as require, but it makes sure that the file is only required once.	`require_once($path)`
Exits the current script. If $status is included, it is sent to the browser.	`exit([$status])`
Works the same as the exit function	`die([$status])`

Example:

f1.php	f2.php

`<form action="file.php" method="POST">`
` Username: <input type="text" name="username"><br/>`
` Password: <input type="text" name="password"><br/>`
` <input type="submit">`
`</form>`
	`$file = 'f1.php';`
	`include $file;`
	`//include($file);`
	`//include('f1.php');`
	`//include 'f1.php';`

```
*f2.php will run the form from f1.php
```

FILE HANDLING

FILE CREATION & OPENING	
Open, read or create a file	`fopen($filePath, mode);`
Returns reference to open file	`$file = fopen($filePath, 'mode') or die("Error");`
Checks if file exists (returns bool)	`file_exists($filePath);`
Create empty file	`touch($fileName);`

FILE READING	
Reads file and outputs contents	`readfile($filePath);`
Returns contents of file	`file_get_contents($filePath)`
Open file for reading	`$file = fopen($filePath, 'r')`
Reads specified num of bytes (Default: whole file)	`fread($file, [num]);`
Gets line, moves cursor to next line	`fgets($file)`
Gets char	`fgetc($file)`
Save contents to array	

FILE WRITING	

Write to a file (Writes over content)	`$file = fopen($fileName, 'w')`
	`fwrite($file, $stringToWriteToFile);`

Append to a file	`$file = fopen($fileName, 'a')`
	`fwrite($file, $stringToWriteToFile);`

Write content of string to file	`file_put_contents('filename.txt', $content);`

FILE CLOSING	
Rewind file to beginning	`rewind($file);`
Boolean returns end of file	`feof($file);`
Closes the file	`fclose($file);`
Delete a file	`unlink($filePath);`

modes

`r / r+`	Read-only/Read & write (Pointer: beginning of file)
`w / w+`	Write-only/Read & write - Erases or creates if it doesn't exist. (Pointer: beginning of file)
`a / a+`	Write-only/Read & write - Data in file is preserved, or creates file. (Pointer: end of file)
`x / x+`	Create new file write-only/Create new file read & write - Returns false if file exists.

COOKIES

Create/modify cookie	setcookie(name, [*value, expire, path, domain, secure, httponly*]);
Delete a cookie	setcookie("*name*", "", time() - [*negative number*]);
Check if enabled	if(count($_COOKIE) > 0)
Get value	$*varName* = $_COOKIE['*cookieName*'];

Example:

$name = 'userid';	< Name of cookie
$value = 'arusso';	< Value of cookie (default: "")
$expire = strtotime('+1 year');	< Expiration date of cookie (default: 0)
setcookie($name, $value, $expire, '\');	< Create cookie with given values

SESSIONS

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- Session variables are associative arrays that hold information about one single user, and are available to all pages in one application, and last until the user closes the browser.

Start a PHP session	session_start();
Get name of session cookie	$varName = session_name(); //Default: PHPSESSID
Get name of session ID	$id = session_id(); //Example: fj3k3alk49jf30e5g68s3
Set session ID	session_id('IdName');
Set session variable	$_SESSION['name'] = 'value';
Get variable from session	$*varName* = $_SESSION['name'];
Print session variables	print_r($_SESSION); //Returns Array ([*name*] => value)
Modify session variable	$_SESSION['name'] = 'new_value';
Remove a session variable	unset($_SESSION[name]);
Remove all session variables	session_unset(); OR $_SESSION = array();
Destroy the session	session_destroy();

Example:

<?php **session_start();**	< Start the session
$_SESSION["favcolor"] = "green";	< Set session variables
echo "Favorite color is " . **$_SESSION["favcolor"];**	< Get session variable and echo value
session_unset();	< Remove all session variables
session_destroy();	< Destroy the session

OBJECT-ORIENTED

Create class
Create constructor
Add property
Add method
Create destructor
	`class ClassName {`
	`      public function` **`__construct`**`() { … }`
	`      `*`accessLevel`*` $propertyName, $propertyName2…`
	`      `*`accessLevel`*` function functionName() { … }`
	`      public function` **`__destruct`**`() {`
	`}`

Refer to current object	`$this`
Refer to current class	`$self`
Create object	`$objectName = new ClassName();`

SETTER
 	
Using magic method:
Setting using magic method:
	`public function setProperty($newValue) {`
	`        $this->property_name = $newValue;`
	`}`
	
	`function` **`__set`**`($propertyName, $value) { … }`
	`$objName>propertyName = "new value";`

GETTER
 	
Using magic method:
Getting using magic method:
	`public function getPropertyName() {`
	`        return $this->property_name;`
	`}`
	
	`function` **`__get`**`($propertyName) { … }`
	`echo $objName->propertyName;`

Create **static** property/method
Refer to static property:
	`static $propertyName; static methodName() { … }`
	`self::$propertyName;`

Create **constant** property
Refer to constant property:
	`const CONSTNAME = value;`
	`self::CONSTNAME`

classes and relationships

Abstract (Cannot instantiate)
abstract method (Declare signature):
	`abstract`` class `*`ClassName`*` {`
	`  `**`abstract`**` public function `*`funcName`*`();`

Final (Cannot be inherited)
final method (Cannot be overridden):
	`final`` class `*`ClassName`*` {`
	`final`` public function `*`funcName`*`();`

Interface (Cannot instantiate)
Cannot include variables, only const:
Methods (must be public):
Implementing an interface:
	`interface`` `*`InterfaceName`*` {`
	`  `**`const`**` `*`CONSTANTNAME;`*
	`  function `*`funcName`*`();`
	`class `*`ClassName`*` `**`implements`**` InterfaceName {`

Anonymous class	`$`*`objectName`*` = new class {`
	`    //properties and methods };`

Inheritance	`class `**`SubClass`**` extends `**`BaseClass`**
Calling a parent method	`parent::__construct();`

DATABASES

Create database object from PDO	`new PDO($dsn, $username, $password);`
DSN syntax	`mysql:host=host_address;dbname=database_name;`
Example:	`$dsn = 'mysql:host=localhost;dbname=my_guitar_shop';` `$db = new PDO($dsn, 'mgs_usr', 'D39EJKEL34');`
SELECT statements	`$query = 'SELECT * FROM tablename...';` `$result_set = $db->query($query);`
INSERT statements	`$query = "INSERT INTO table(columns) VALUES (values)";` `$insert_count = $db->exec($query);`
UPDATE statements	`$query = "UPDATE table SET col = value WHERE...";` `$update_count = $db->exec($query);`
DELETE statements	`$query = "DELETE FROM table WHERE ...";` `$delete_count = $db->exec($query);`

ERROR HANDLING

Create new exception	`new Exception($message [, $code]);`
Throw statement	`throw $exception;`
Try/Catch	`try { statements }` `catch (ExceptionClass $exceptionName) { statements }`