MAGIC CONSTANTS

- `__LINE__` The current line number of the file.
- `__FILE__` The full path and filename of the file with symlinks resolved. If used inside an include, the name of the included file is returned.
- `__DIR__` The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to dirname(__FILE__). This directory name does not have a trailing slash unless it is the root directory.
- `__FUNCTION__` The function name.
- `__CLASS__` The class name. The class name includes the namespace it was declared in (e.g. Foo\Bar).
- `__TRAIT__` The trait name. The trait name includes the namespace it was declared in (e.g. Foo\Bar).
- `__METHOD__` The class method name.
- `__NAMESPACE__` The name of the current namespace.
- `ClassName::class` The fully qualified class name. See also ::class.

BITWISE OPERATORS

- `$a & $b` - **And** - Bits that are set in both $a and $b are set.
- `$a | $b` - **Or (inclusive or)** - Bits that are set in either $a or $b are set.
- `$a ^ $b` - **Xor (exclusive or)** - Bits that are set in $a or $b but not both are set.
- `~ $a` - **Not** - Bits that are set in $a are not set, and vice versa.
- `$a << $b` - **Shift left** - Shift the bits of $a $b steps to the left (each step means "multiply by two")
- `$a >> $b` - **Shift right** - Shift the bits of $a $b steps to the right (each step means "divide by two")

EXECUTION OPERATORS

```php
<?php

/* PHP supports one execution operator: backticks (``).
Note that these are not single-quotes! PHP will attempt to execute the contents of the
backticks as a shell command;
the output will be returned (i.e., it won't simply be dumped to output; it can be
assigned to a variable).
Use of the backtick operator is identical to shell_exec(). */

$output = `ls -al`;
echo "<pre>$output</pre>";
```

TYPE OPERATORS

```php
<?php

// instanceof is used to determine whether a PHP variable is an instantiated object of
a certain class

class MyClass
{
}

class NotMyClass
{
}
$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
```

ANONYMUS FUNCTIONS

```php
<?php

/* Anonymous functions, also known as closures, allow the creation of functions which
have no specified name. They are most useful as the value of callback parameters, but
they have many other uses. */

$greet = function($name) {
    printf("Hello %s\r\n", $name);
};

$greet('World');
$greet('PHP');
```

VARIABLE FUNCTIONS

```php
<?php

/* PHP supports the concept of variable functions. This means that if a variable name has
parentheses appended to it, PHP will look for a function with the same name as whatever
the variable evaluates to, and will attempt to execute it. Among other things, this can
be used to implement callbacks, function tables, and so forth. */

class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // This calls the Bar() method
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // This calls $foo->Variable()
```

REFERENCES

```php
<?php

// assign by reference
$a =& $b; // $a and $b are completely equal here. $a is not pointing to $b or vice versa. $a and $b are pointing to the same place.


// pass by reference
function foo(&$var) {
    $var++;
}
$a = 5;
foo($a);
echo $a; // outputs 6

// returning references
class Fruit {
    private $color = "red";

    public function &getColor() {
        return $this->color;
    }
}
$fruit = new Fruit;
$color = &$fruit->getColorByRef();
echo "Fruit's color is $color\n";
$color = "green"; // now this changes the actual property of $fruit
$color = $fruit->getColor();
echo "Fruit's color is $color\n";
```

VARIABLE VARIABLES

- Sometimes it is convenient to be able to have variable variable names.
- That is, a variable name which can be set and used dynamically.

VARIABLE VARIABLES EXAMPLE

```php
<?php

$a = 'hello';

$$a = 'world';

echo "$a ${$a}";

echo "$a $hello"; // output same as above
```

GENERATORS

- Generators provide an easy way to implement simple iterators without the overhead or complexity of implementing a class that implements the `Iterator` interface.
- A generator allows you to write code that uses foreach to iterate over a set of data without needing to build an array in memory, which may cause you to exceed a memory limit, or require a considerable amount of processing time to generate.
- Instead, you can write a generator function, which is the same as a normal function, except that instead of returning once, a generator can yield as many times as it needs to in order to provide the values to be iterated over.

GENERATORS EXAMPLE

```php
<?php

function xrange($start, $limit, $step = 1) {
    if ($start < $limit) {
        if ($step <= 0) {
            throw new LogicException('Step must be +ve');
        }

        for ($i = $start; $i <= $limit; $i += $step) {
            yield $i;
        }
    } else {
        if ($step >= 0) {
            throw new LogicException('Step must be -ve');
        }

        for ($i = $start; $i >= $limit; $i += $step) {
            yield $i;
        }
    }
}


echo 'Single digit odd numbers from range():  ';
foreach (range(1, 9, 2) as $number) {
    echo "$number ";
}
echo "\n";

echo 'Single digit odd numbers from xrange(): ';
foreach (xrange(1, 9, 2) as $number) {
    echo "$number ";
}
```

ITERABLES

- Iterable is a pseudo-type introduced in PHP 7.1. It accepts any array or object implementing the `Traversable` interface.
- Both of these types are iterable using foreach and can be used with `yield` from within a generator.
- `Traversible` is an interface to detect if a class is traversable using foreach.

ITERABLES EXAMPLE

```php
<?php

function foo(iterable $iterable) {
    foreach ($iterable as $value) {
        // ...
    }
}


var_dump(is_iterable([1, 2, 3])); // bool(true)
var_dump(is_iterable(new ArrayIterator([1, 2, 3]))); // bool(true)
var_dump(is_iterable((function () { yield 1; })())); // bool(true)
var_dump(is_iterable(1)); // bool(false)
var_dump(is_iterable(new stdClass())); // bool(false)
```

CALLBACKS / CALLABLES

- Some functions like `call_user_func()` or `usort()` accept user-defined callback functions as a parameter.
- Callback functions can not only be simple functions, but also object methods, including static class methods.
- A PHP function is passed by its name as a string.
- Any built-in or user-defined function can be used, except language constructs such as: `array()`, `echo`, `empty()`, `eval()`, `exit()`, `isset()`, `list()`, `print` or `unset()`.

CALLBACKS / CALLABLES EXAMPLE

```php
<?php

// An example callback function
function my_callback_function() {
    echo 'hello world!';
}

// An example callback method
class MyClass {
    static function myCallbackMethod() {
        echo 'Hello World!';
    }
}

// Type 1: Simple callback
call_user_func('my_callback_function');

// Type 2: Static class method call
call_user_func(array('MyClass', 'myCallbackMethod'));

// Type 3: Object method call
$obj = new MyClass();
call_user_func(array($obj, 'myCallbackMethod'));


// Our closure
$double = function($a) {
    return $a * 2;
};

// This is our range of numbers
$numbers = range(1, 5);

// Use the closure as a callback here to double the size of each element in our range
$new_numbers = array_map($double, $numbers);
print implode(' ', $new_numbers);
```

PHP & MYSQL

```php
<?php

/* Connect to a MySQL database using driver invocation */
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';

try {
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}
$dbh->exec("set names utf8");

/* Delete all rows from the FRUIT table */
$count = $dbh->exec("DELETE FROM fruit WHERE colour = 'red'");

$sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
foreach ($conn->query($sql) as $row) {
    print $row['name'] . "\t" . $row['color'] . "\t" . $row['calories'] . "\n";
}
```

PHP & MYSQL – TRANSACTIONS & PREPARE STATEMENTS

```php
<?php

/* Begin a transaction, turning off autocommit */
$dbh->beginTransaction();

/* Insert multiple records on an all-or-nothing basis */
$sql = 'INSERT INTO fruit (name, colour, calories) VALUES (?, ?, ?)';

$sth = $dbh->prepare($sql);

foreach ($fruits as $fruit) {
    $sth->execute(array(
        $fruit->name,
        $fruit->colour,
        $fruit->calories,
    ));
}

/* Commit the changes */
$dbh->commit();

/* Database connection is now back in autocommit mode */
```

PHP & CURL

```php
<?php

// A very simple PHP example that sends a HTTP POST to a remote site

$ch = curl_init();

curl_setopt($ch, CURLOPT_URL,"http://www.example.com/tester.phtml");
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, "postvar1=value1&postvar2=value2&postvar3=value3");

// in real life you should use something like:
// curl_setopt($ch, CURLOPT_POSTFIELDS,
//          http_build_query(array('postvar1' => 'value1')));

// receive server response ...
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$server_output = curl_exec ($ch);

curl_close ($ch);

// further processing ....
if ($server_output == "OK") { ... } else { ... }
```

PHP & XML

```php
<?php

$xml = new SimpleXMLElement('<xml/>');

for ($i = 1; $i <= 8; ++$i) {
    $track = $xml->addChild('track');
    $track->addChild('path', "song$i.mp3");
    $track->addChild('title', "Track $i - Track Title");
}

header('Content-type: text/xml');
print($xml->asXML());
```

PHP & XML

```php
<?php

$mysongs = simplexml_load_file('songs.xml');

echo $mysongs->song[0]->artist;
echo $mysongs->song[1]['dateplayed'];


$string = <<<XML
<?xml version='1.0'?>
<document>
 <title>Forty What?</title>
 <from>Joe</from>
 <to>Jane</to>
 <body>
  I know that's the answer -- but what's the question?
 </body>
</document>
XML;

$xml = simplexml_load_string($string);
```

PHP & JSON

```php
<?php

$a = array('<foo>',"'bar'",'"baz"','&blong&', "\xc3\xa9");

echo "Normal: ",  json_encode($a), "\n";
echo "Tags: ",    json_encode($a, JSON_HEX_TAG), "\n";
echo "Apos: ",    json_encode($a, JSON_HEX_APOS), "\n";
echo "Quot: ",    json_encode($a, JSON_HEX_QUOT), "\n";
echo "Amp: ",     json_encode($a, JSON_HEX_AMP), "\n";
echo "Unicode: ", json_encode($a, JSON_UNESCAPED_UNICODE), "\n";
echo "All: ",     json_encode($a, JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT |
JSON_HEX_AMP | JSON_UNESCAPED_UNICODE), "\n\n";


$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

var_dump(json_decode($json)); // outputs object
var_dump(json_decode($json, true)); // outputs associative array
```

PHP FILTERS

```php
<?php

$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;

$int = 100;
if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}

$int = 0;
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
```

PHP FILTERS

```php
<?php

$ip = "127.0.0.1";
if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}

$email = "john.doe@example.com";
$email = filter_var($email, FILTER_SANITIZE_EMAIL); // Remove all illegal characters from email
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}

$url = "https://www.w3schools.com";
$url = filter_var($url, FILTER_SANITIZE_URL); // Remove all illegal characters from a url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
```

PHP FORM VALIDATION

```php
<?php

// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

PHP FILE UPLOAD

```php
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);

if(isset($_POST["submit"])) { // Check if image file is a actual image or fake image
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if ($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }

    if (file_exists($target_file)) { // Check if file already exists
        echo "Sorry, file already exists.";
        $uploadOk = 0;
    }
    if ($_FILES["fileToUpload"]["size"] > 500000) { // Check file size
        echo "Sorry, your file is too large.";
        $uploadOk = 0;
    }
    if ($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
        && $imageFileType != "gif"
    ) {
        echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
        $uploadOk = 0;
    } // Allow certain file formats

    if ($uploadOk == 0) { // Check if $uploadOk is set to 0 by an error
        echo "Sorry, your file was not uploaded.";
    } else { // if everything is ok, try to upload file
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
            echo "The file " . basename($_FILES["fileToUpload"]["name"]) . " has been uploaded.";
        } else {
            echo "Sorry, there was an error uploading your file.";
        }
    }
}
```