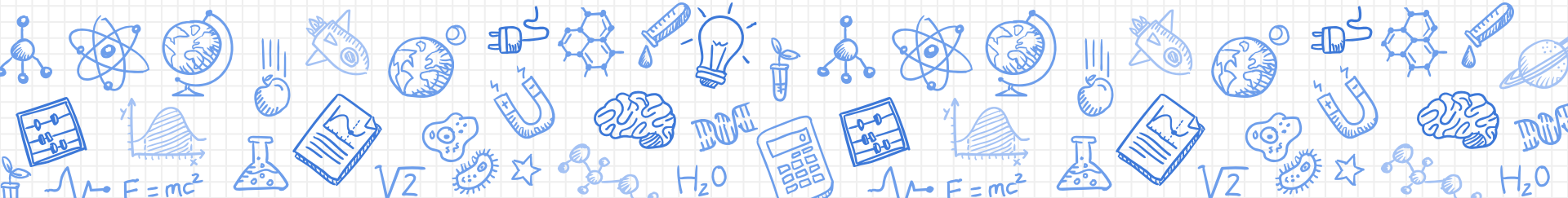
Object-Oriented PHP

Global Variables



Variable scope

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

1. local
2. global
3. static

```php
<?php
    $a = 1; /* global scope */

    function test(){
        echo $a; /* reference to local scope variable */
    }

    test();
?>
```

The global keyword

The global keyword is used to access a global variable from within a function.

PHP also stores all global variables in an array called **$GLOBALS[index]**. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

```php
<?php
$x = 5;
$y = 10;

function myTest() {
    global $x, $y;
    $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

```php
<?php
$x = 5;
$y = 10;

function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

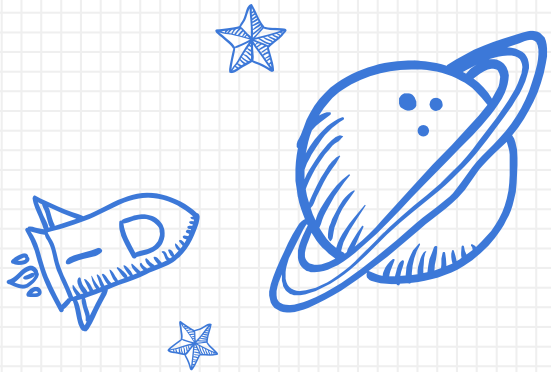
The static Keyword

When a function is completed/executed, all of its variables are deleted.
However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable.

```php
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
myTest();
myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.



A static variable will not lose its value when the function exits and will still hold the value next time the function is called.

Superglobals

Superglobals are built-in variables that are always available in all scopes.

There is no need to do **global** $variable; to access them within functions or methods.

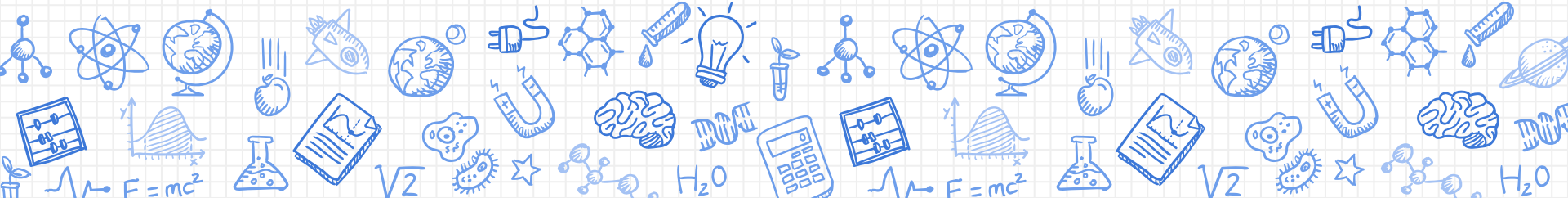
Superglobals

$GLOBALS
$_SERVER
$_GET
$_POST
$_FILES
$_COOKIE
$_SESSION
$_REQUEST
$_ENV

Superglobals

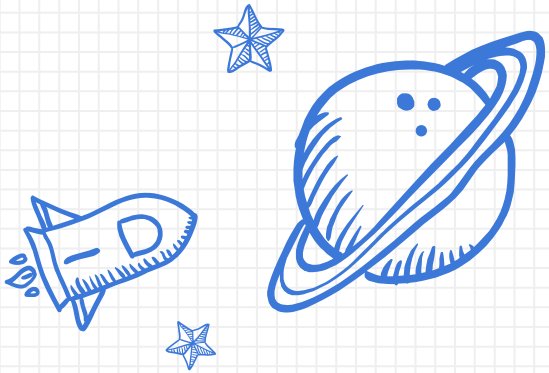
Superglobals	Description
$GLOBALS	An associative array containing references to all variables which are currently defined in the global scope of the script. The variable names are the keys of the array.
$_SERVER	Is an array containing information such as headers, paths, and script locations.
$_GET	An associative array of variables passed to the current script via the URL parameters.
$_POST	An associative array of variables passed to the current script via the HTTP POST method when using application/x-www-form-urlencoded or multipart/form-data as the HTTP Content-Type in the request.

Classes and Objects



Classes

Basic class definitions begin with the keyword class, followed by a class name, followed by a pair of curly braces which enclose the definitions of the properties and methods belonging to the class.



Object-oriented programming is a style of coding that allows developers to group similar tasks into classes.

```php
<?php

        class MyClass{
            // Class properties and methods go here
        }

        $obj = new MyClass();

        var_dump($obj);

?>
```

The syntax to create a class is pretty straightforward: declare a class using the class keyword, followed by the name of the class and a set of curly braces ({}).
After creating the class, a new class can be instantiated and stored in a variable using the new keyword:

Class Properties

Class member variables are called "**properties**". You may also see them referred to using other terms such as "**attributes**" or "**fields**".
They are defined by using one of the keywords **public**, **protected**, or **private**, followed by a normal variable declaration.

```php
<?php

class MyClass{
  public $prop1 = "I'm a class property!";
}

$obj = new MyClass();

var_dump($obj);

echo $obj->prop1; // read class property

?>
```

Because multiple instances of a class can exist, if the individual object is not referenced, the script would be unable to determine which object to read from. The use of the **arrow (->)** is an OOP construct that accesses the contained properties and methods of a given object.

Class Methods

Methods are class-specific functions. Individual actions that an object will be able to perform are defined within the class as methods.

```php
<?php

class MyClass{
  public $prop1 = "I'm a class property!";

  public function setProperty($newval){
      $this->prop1 = $newval;
  }

  public function getProperty(){
      return $this->prop1 . "<br />";
  }
}

$obj = new MyClass();

echo $obj->getProperty(); // Get the property value

$obj->setProperty("I'm a new property value!"); // Set a new one

echo $obj->getProperty(); // Read it out again to show the change

?>
```

$this

The pseudo-variable **$this** is available when a method is called from within an object context.

$this is a reference to the calling object (usually the object to which the method belongs, but possibly another object, if the method is called statically from the context of a secondary object).

The self keyword

Self refers to the class in which it is called, while $this refers to the class of the current object.

Use the self keyword to access static properties and methods.

Constructors and Destructors

Classes which have a constructor method call this method on each newly-created object, so it is suitable for any initialization that the object may need before it is used.

Constructors and Destructors

To call a function when the object is create, the __construct magic method is available.
This is useful for class properties initialization and class config.

To call a function when the object is destroyed, the **__destruct()** magic method is available.
This is useful for class cleanup (closing a database connection, for instance).

```php
<?php

        class MyClass{
                public $prop1 = "I'm a class property!";

                public function __construct(){
                        echo 'The class "', __CLASS__, '" was initiated!<br />';
                }

                public function __destruct(){
                        echo 'The class "', __CLASS__, '" was destroyed.<br />';
                }
        }

        $obj = new MyClass();
?>
```

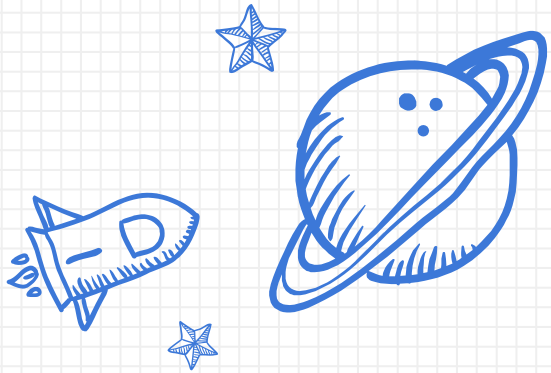
__CLASS__ returns the name of the class in which it is called; this is what is known as a magic constant.

"When the end of a file is reached, PHP automatically releases all resources."

Class Constants

It is possible to define constant values on a per-class basis remaining the same and unchangeable.
Constants differ from normal variables in that you don't use the **$** symbol to declare or use them.
The value must be a constant expression, not (for example) a variable, a property, a result of a mathematical operation, or a function call.

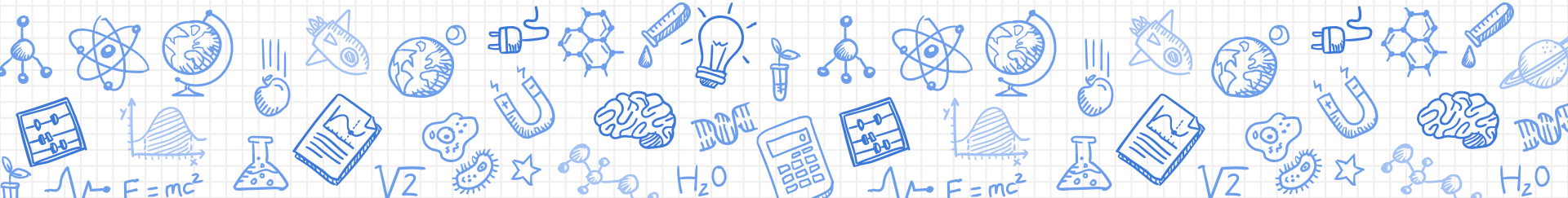
```php
<?php

class MyClass{
    const CONSTANT = 'constant value';

    function showConstant() {
        echo  self::CONSTANT;
    }
}

echo MyClass::CONSTANT;

$class = new MyClass();
$class->showConstant();

echo $class::CONSTANT;

?>
```



Use $this to refer to the current object.
Use self to refer to the current class.

Use $this->member for non-static members.
Use self::$member for static members.

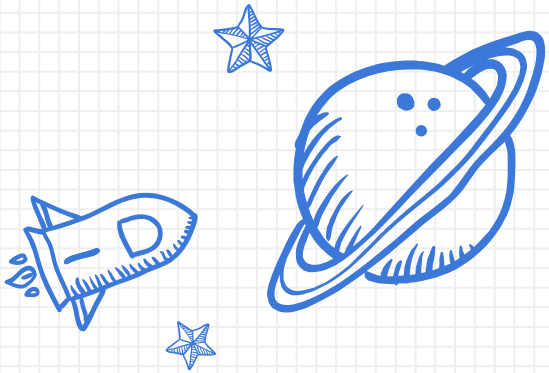
Include & Require



Include & Require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.
The include and require statements are identical, except upon failure:

- ✘ require will produce a fatal error (**E_COMPILE_ERROR**) and stop the script
- ✘ include will only produce a warning (**E_WARNING**) and the script will continue

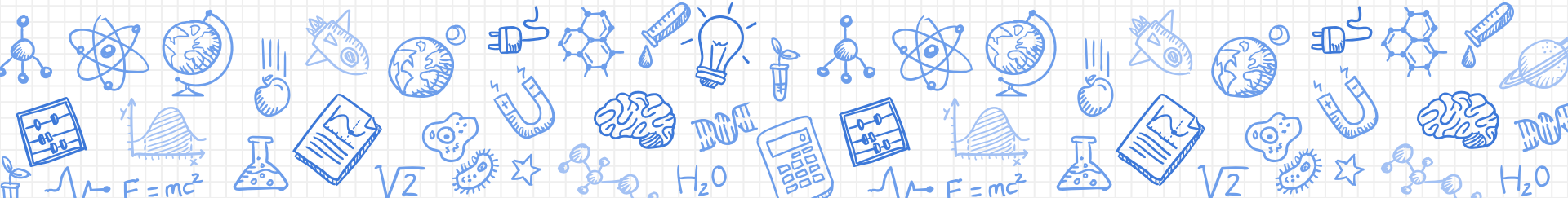


Use require when the file is required by the application.

Use include when the file is not required and application should continue when file is not found.

```php
<?php

        include 'myClass.php';

        //or

        require 'myClass.php';

?>
```

Practice



Create a PHP class

Create a class called Template to define new pages, this class must have a constructor that sets the page title and the contents on the class.
The class must have a method to create new content and also a method to print the page in HTML with all the content.
The class must be in a separate php.
Create a home and a blog pages that must include and use the Template class.

```php
<?php
    class Template{
        public $title;
        public $contents;
        public $footer;

        function __construct($title, $contents){
            $this->title = $title;

            if(is_array($contents)){
                foreach ($contents as $key => $content) {
                    $this->createContent($content);
                }
            }
        }
        public function createContent($content){
            // content logic
        }
        public function printPage(){
            // print logic
        }
    }
?>
```

```php
<?php

require 'TemplateClass.php';

$homePageContents = array(
            'News' => array(
                        'title' => 'News',
                        'content' => 'News Content',
                        'date' => '2015/06/01'
                        ),
            'Blog' => array(
                        'title' => 'How to',
                        'content' => 'How to..',
                        'date' => '2015/06/02',
                        ),
);

$homePage = new Template('Home', $homePageContents);
$homePage->printPage();

?>
```