WEB 306 – PHP: Databases and Framework

Day 4 – Classes and Objects

1. Font Awesome

1.1 About Font Awesome

Font Awesome is an icon font, SVG graphic, CSS/Sass and JavaScript library which gives you scalable vector icons that can instantly be customized — size, color, drop shadow and anything that can be done with CSS.

1.2 Using Font Awesome

The quickest way to get started with Font Awesome is to go to their Hosting Font Awesome Yourself page, Download Font Awesome Free for the Web, and copy their `all.min.css` file into your CSS folder and the `/webfonts` folder into your project. Then link to their `all.min.css` file in the `<head>` tag of your page.

```
<!-- Font Awesome CSS link to file downloaded from fontawesome.com -->
<link rel="stylesheet" href="css/fontawesome/all.min.css">
```

It used to be even quicker to link to the Font Awesome CDN, where all of the icon font files and CSS files are already stored on Font Awesome's servers and could simply be linked to by copying and pasting a CSS link from their site. However, after Font Awesome updated their site, they made it mandatory to provide them with an email address, receive an email from them, click on a link in the email and register for a Font Awesome account before you can access the CDN link. As a result, it's quicker to download the files to use the latest version of Font Awesome. Here is a CDN link to an older version of Font Awesome which is still functional.

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css" integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt+Y8vEf7N7fWAU" crossorigin="anonymous">
```

Font Awesome tells you to use the `<i>` tag to insert your icons because "i" stands for icon, but I prefer not to do this — it's a misleading practice semantically in my opinion because the `<i>` tag is reserved for italics so I just use the `<span>` tag instead.

To insert an icon just add the class "fa" to the `<span>` tag and then add in the class for your specific icon.

FA Resources

1. Font Awesome Website
2. Font Awesome icons
3. Font Awesome Cheatsheet
4. Start
5. Download
6. Accessibility
7. On the Desktop

```
  <span class="fa fa-smile"></span>
```

For brands, like Facebook or Twitter, you must use the class “fab” instead of “fa”. Font Awesome icons take on all CSS properties of the HTML element which they are inserted in. To find a complete list of all of the Font Awesome icons and their classes view the Font Awesome cheat sheet at fontawesome.com/cheatsheet.

1.3 Font Awesome Accessibility

Font Awesome icons are not accessible by default so if you use them in links or for other functional purposes you must include ARIA attributes and the “sr-only” (screen reader only) class to tell people with visual impairments what the icon represents.

Here is an example of how you would make a Font Awesome icon in a link accessible.

```
  <a href="http://facebook.com" target="_blank">
    <span class="fab fa-facebook" aria-hidden></span>
    <span class="sr-only">Facebook</span>
  </a>
```

You can read more about making Font Awesome icons accessible at fontawesome.com/how-to-use/accessibility.

2. Activity

Find an object and complete the following statements about it.

1. **This object can be classified more generally with other objects as a:**

   ____

2. **This object has the following properties/attributes (i.e. name, color, texture etc.):**

   ____

   ____

   ____

3. **This object is supposed to be used for the following methods/purposes:**

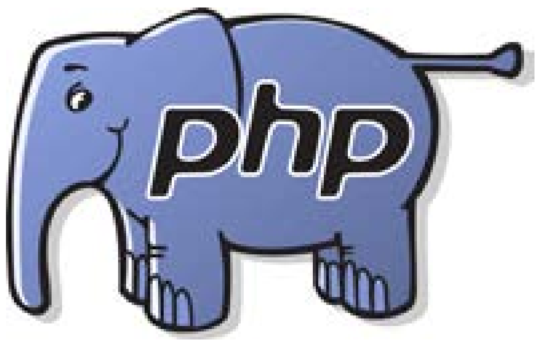
   ____

   ____

   ____

3. Objects

In computer programming an “object” is a representation of a concept or thing which exists in real life as a collection of mapped data and capabilities. For instance, objects are frequently used to represent users in a social media network or products in an ecommerce website.



> **"In object-oriented programming (OOP), objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process. In between, each object is made into a generic class of object and even more generic classes are defined so that objects can share models and reuse the class definitions in their code. Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables. An object is what actually runs in the computer."**
>
> **— From object, TechTarget**

PHP Resources

1. PHP Website
2. XAMPP
3. Documentation
4. W3Schools
5. CSS Tricks
6. Scotch.io
7. Stack Overflow

4. Classes

4.1 About Classes

All objects are defined by a structure called a class. PHP classes are totally different from HTML/CSS classes so don't confuse them. Classes are used to define a type of object and could be very general or very specific. For instance, a class of object could be a `Car`, a `Phone` or a `Product`. Classes could also represent living things like people or animals such as an `Elephant`, a `Dog` or a `Clown`. Think of a class as a blueprint for a category of object which would be given more specific properties at a later point. It is also possible to create child classes of a class which inherit the properties of their parent class

> **In object-oriented programming, a class is a template definition of the methods and variables in a particular kind of object. Thus, an object is a specific instance of a class; it contains real values instead of variables.**
>
> **— From class, TechTarget**

4.2 Defining Classes

PHP class names are frequently capitalized using **UpperCamelCase**. It is convention to give classes a dedicated file and to name the file after the class, using the exact same name as the class, including the casing. For example, if you had a `Phone` class, you would save the file as `Phone.php`.

It wouldn't be an uncommon practice to keep classes in a `/classes` folder but there are many different folder conventions for classes depending on the context. Class files are treated like include files in that they do not require a closing PHP tag because they will only include PHP code aside from

HTML which is stored in strings. To define a class use the `class` keyword, followed by the name of your class.

```
<?php
class Phone {}
```

5. Properties

Objects are used to map data in key/value pairs like associative arrays. However, while associative arrays are used to store similar values, objects are used to store related values which are tied to a specific class. These values are saved as ***variables*** which are defined ***inside of a class*** and are called ***properties***. You ***must*** write a keyword such as `public` before property names to define the scope of the property. I will explain this in more detail later but a ***public property*** is kind of like a ***global variable***.

```
class Phone {
   public name = “Samsung Galaxy A7”;
   public os = “Android 8.0”;
   public weight = 168;
   public memory = 128;
}
```

6. Methods

Another thing that separates an object from an associative array is that they can perform actions which are represented by a group of ***functions*** that are defined ***inside of a class*** and which are called ***methods***. For instance, a user can post a status update or a product can go on sale.

> **In object-oriented programming, a method is a programmed procedure that is defined as part of a class and included in any object of that class. A class (and thus an object) can have more than one method. A method in an object can only have access to the data known to that object, which ensures data integrity among the set of objects in an application. A method can be re-used in multiple objects.**
>
> **— From method, TechTarget**

Just like a regular PHP function, we can run any PHP code in a method including to echo out strings and HTML and it will be used when the method is called at a later point. Some developers consider it a bad practice to echo content inside of methods. This opinion is a bit subjective but it is true that you would not be likely to normally have a reason to use class methods to echo content. However, for the purpose of demonstrating how methods work in a visual way, we will be using methods to echo content during this lesson. Like functions, methods can also have parameters.

```
class Phone {
   name = “Samsung Galaxy A7”;
   os = “Android 8.0”;
   weight = 168;
```

```
    memory = 128;

    function call($contact) {
         echo '<h2>Calling ' . ucwords($contact) . '...</h2>';
         echo '<audio autoplay><source src="audio/call/' .
              strtolower($contact) . '.mp3"></audio>';
         echo '<img src="img/call/' . strtolower($contact) . '.gif"
              alt="' . ucwords($contact) . '">';

    }
}
```

7. Creating New Object Instances

An ***object*** is actually a new ***instance*** of a ***class***. In order to create a new instance of a Phone, we need to `require` our `Phone.php` file which has our `Phone` class.

```
require 'classes/Phone.php';
```

To create a new instance of a class, and therefore an object, create a variable with a value of the class name prefaced with the `new` keyword.

```
my_phone = new Phone;
```

8. Accessing Properties and Methods

To access the properties and methods of an object use the variable name of the object, followed by an arrow `->` and then the name of the property you are trying to access. We can use the `->` arrow to access properties of `$my_phone`.

```
<h1><?php echo $my_phone->name; ?></h1>
<ul>
   <li><?php echo $my_phone->os; ?></li>
   <li><?php echo $my_phone->weight; ?> g</li>
   <li><?php echo $my_phone->memory; ?> GB</li>
</ul>
```

To call a method of an object use the variable name of the object followed by an arrow `->` and then the name of the method including the parentheses, the same way you would call a function.

```
$my_phone->call($_GET['contact']);
```

We are going to call the `->call()` method if the `$_GET` key, `'contact'` is set.

```
if (isset($_GET['contact'])):
   $my_phone->call($_GET['contact']);
endif;
```

We will make links which will change the parameter of `->call()` by changing the value of the `$_GET` key, `'contact'`.

```
<?php if (isset($_GET['contact'])): ?>
   <a href="new-phone.php" class="end">End Call</a>
<?php else: ?>
   <h2><span class="fa fa-user"></span> Recent Contacts</h2>
   <a href="new-phone.php?contact=kermit">Call Kermit</a>
   <a href="new-phone.php?contact=batman">Call Batman</a>
   <a href="new-phone.php?contact=homer">Call Homer</a>
<?php endif; ?>
```

9. Advanced Classes

Most classes would be more complex than the `Phone` class we created and would allow for the properties of the class to be dynamically defined each time an instance of the class is created instead of hardcoding values into the class. Cars are a common example of a class of object because they are very complicated objects with many properties and methods.

```
<?php
class Car {}
```

10. Property and Method Scope

When defining a property, you would only give it a value if it is a ***constant*** value, meaning that it will never change. Otherwise it is just a way of defining the scope and name of the property and the value would be defined at a later point. Properties are given a scope of either `public`, `private` or `protected`.

10.1 Public Scope

A public property is a property which can be accessed ***outside*** of a class. This is kind of like global variable scope. To define a property as public, preface the name of your property with the `public` keyword.

```
class Car {
   // Can be accessed outside of class
   public $name;
   public $driver;
   public $image;
   public $honk;
   public $top_speed;
}
```

10.2 Pivate Scope

A private property is a property which can only be accessed ***inside*** of a class. This is kind of like local variable scope. The purpose of private properties is to prevent users of a class from directly accessing or altering important data without correctly filtering or processing it. To define a property as private, preface the name of your property with the `private` keyword.

```
    // Can be accessed outside of class
    public $top_speed;
    // Can NOT be accessed outside of class
    private $current_speed;
  }
```

10.3 Protected Scope

A protected property is halfway between a public and private property and is only accessible in the ***current class***, as well as ***parent*** and ***child*** classes of the class. To define a property as protected, preface the name of your property with the `protected` keyword.

```
    // Can be accessed outside of class
    public $top_speed;
    // Can NOT be accessed outside of class
    private $current_speed;
    // Can ONLY be accessed in current class & parent/child classes
    protected $gas;
  }
```

10.4 Method Scope

Like properties, methods also have a scope which is defined as either `public`, `private` or `protected` to determine whether they can be accessed outside of a class or not. All of the same rules that apply to property scope apply to method scope.

```
  // Can be accessed outside of class
  public function start() {}
  // Can NOT be accessed outside of class
  private function dashboard() {}
  // Can ONLY be accessed in current class & parent/child classes
  protected function show() {}
```

11. Magic Methods

Magic methods are methods that perform special actions automatically. All magic methods are named with two underscores before the function name so avoid naming your own methods using this practice. There are many types of magic methods but there is one magic method which is more important than the rest.

11.1 The __construct() Method

Most classes will have a type of magic method which is known as a ***"constructor" method*** and is called `__construct()`. This magic method is used to define the properties of a new instance of a class which results in an object. It uses parameters to define the properties of the class and runs automatically whenever an object is created. In this case, the method defines the properties of our `Car`. This class will have a public `__construct()` method which is going to take one parameter which will be defined by an array.

```
class Car {
   public $name;
   public $driver;
   public $image;
   public $honk;
   public $top_speed;
   private $current_speed;
   protected $gas;

   public function __construct($some_array) {}
}
```

11.2 The $this Variable

PHP uses a variable called `$this` which acts as a placeholder for an instance of this class. The `->` arrow is used to define and access properties and methods of `$this`. We are going to define most of our properties using our `$some_array` parameter. We can also set specific values for properties for every instance of this class and all objects will start with those properties until they are redefined.

```
public function __construct($some_array) {
   $this->name = $some_array[‘name’];
   $this->driver = $some_array[‘driver’];
   $this->image = $some_array[‘image’];
   $this->honk = $some_array[‘honk’];
   $this->top_speed = $some_array[‘top_speed’];
   $this->current_speed = 0;
   $this->gas = 100;
}
```

12. Additional Methods

We can also create as many additional methods for our class as we like. These would just represent whatever we need our class to be able to do. We can reference and redefine our class’ properties using `$this` and the `->` arrow inside of its subsequent methods.

This method starts the `Car` by setting the `$current_speed` property, reducing the `$gas` property, displaying the speedometer and fuel level and showing an image of the car driving.

```
public function start($speed) {

   // Set the speed of this car to a value between 1 and $top_speed
   if ($speed > 0 && $speed < $this->top_speed) {
       $this->current_speed = $speed;
   } else if ($speed < 1) {
       $this->current_speed = 1;
   } else if ($speed > $this->top_speed) {
       $this->current_speed = $this->top_speed;
```

```
    }

    // Set the gas to $gas minus 10%
    $this->gas = $this->gas - 10;

    // Display the Car’s current speed and the amount of gas
    echo ‘<p><span class=”fa fa-tachometer-alt”></span> ‘   .
         $this->current_speed . ‘ <span class=”fa fa-battery-half”>
         </span> ‘ . $this->gas . ‘</p>’;

    // Display an image of the Car driving
    echo ‘<p><img src=”img/start/’ . $this->image . ‘” alt=”’ .
         $this->name . ‘”></p>’;
  }
```

This method honks the horn by echoing out an audio tag set to autoplay with a source tag with a src attribute which is set to the value of `$honk`.

```
  public function honk() {
    echo ‘<audio autoplay><source src=”audio/honk/’ . $this->honk . ‘”>
         </audio>’;
  }
```

This method displays an image of the `Car` doing doughnuts.

```
  public function doughnuts() {

    // Set the current speed to the top speed
    $this->current_speed = $this->top_speed;

    // Set the gas to the gas minus 20
    $this->gas = $this->gas - 20;

    // Display the Car’s current speed and the amount of gas
    echo ‘<p><span class=”fa fa-tachometer-alt”></span> ‘   .
         $this->current_speed . ‘ <span class=”fa fa-battery-half”>
         </span> ‘ . $this->gas . ‘</p>’;

    // Display an image of the Car doing doughnuts
    echo ‘<p><img src=”img/doughnuts/’ . $this->image . ‘” alt=”’ .
         $this->name . ‘”></p>’;
  }
```

This method fills up the car’s gas.

```
  public function fill_up() {

    // Stop the car by setting the current speed to 0
    $this->current_speed = 0;

    // Set the gas back to 100%
    $this->gas = 100;
    // Display the Car’s current speed and the amount of gas
    echo ‘<p><span class=”fa fa-tachometer-alt”></span> ‘  .
         $this->current_speed . ‘ <span class=”fa fa-battery-half”>
         </span> ‘ . $this->gas . ‘</p>’;
  }
```

13. Setting Class Properties

Whenever we want to create a new car we would create a new instance of our Car class and define its properties.

In order to create a new instance of an car, we need to require our `Car.php` file which has our `Car` class.

```
  require ‘classes/Car.php’;
```

We can manually define the properties of our class by setting the parameters of the `__construct()` method when we create a new instance of the `Car` class.

Our `__construct()` method takes one parameter, which is an array of properties. Therefore, we will create an array which will define all of our object’s properties.

```
  $my_properties = array(
    ‘name’ => ‘Some Car’,
    ‘driver’ => ‘Some Guy’,
    ‘image’ => ‘car.gif’,
    ‘honk’ => ‘honk.mp3’,
    ‘top_speed’ => 170
  );
```

It is not necessary to explicitly call the `__construct()` method as it is automatically called once the class instance is created. When we create a new instance of a class we can also add parentheses like we would when calling a function and add the parameters of our `__construct()` method.

```
  $my_car = new Car($my_properties);
```

We can use the `->` arrow to access properties of `$my_car`.

```
  <h1><?php echo $my_car->name; ?></h1>
  <h2><?php echo $my_car->driver; ?></h2>
```

We can also use the `->` arrow to call methods of `$my_car`. The `mt_rand()` function is used to generate a random number between two numbers.

```
  $my_car->start(mt_rand(0, $my_car->top_speed));
  $my_car->honk();
  $my_car->doughnuts();
  $my_car->fill_up();
```