
UTN
TÉCNICO UNIVERSITARIO EN PROGRAMACIÓN
LABORATORIO IV

Nosotros

Comisiones y datos de contacto

Prof. Adrián Solimano
adriansolimano@gmail.com

Prof. Nicolás Bertolucci
nicoberto13@gmail.com

Prof. Andrés Gaggini
agaggini@gmail.com

Prof. Juan Azar
juan.azar@gmail.com

Ayte. Sebastián Ribeiro
sarpriest@gmail.com

Ayte. Adrián Solimano
adriansolimano@gmail.com

Ayte. Candela Yarossi
candyarossi@gmail.com

Ayte. Franco Barilatti
francoboca996@gmail.com

RESPETA A TUS COMPAÑEROS

ESTUDIA CON INTENSIDAD

JÚNTATE CON LOS QUE CONSIDERES MEJORES Y MAS EXPERIMENTADOS

“ESTO FUNCIONA” NO ES DONDE TE DETIENES, ES DONDE COMIENZAS

CONÉCTATE CON LOS DEMÁS, TRABAJA EN EQUIPO

APRENDE TÉCNICAS, NO HERRAMIENTAS

PERMANECE CURIOSO

PIDE AYUDA Y OFRECE AYUDA

AMA LO QUE HACES

DIVIÉRTETE!

Sobre esta materia

Cómo aprobarla?

· Aprobar los 2 (dos) parciales o recuperatorios con 6 (seis) o mas puntos en cada uno.

· La metodología del segundo parcial será un Trabajo Práctico.

Cómo son los Recuperatorios?

· Hay una única instancia al final de la cursada.

· Todos los parciales pueden recuperarse en esta instancia.

· En el caso del segundo parcial (Trabajo Práctico), requerimientos adicionales serán solicitados por los profesores.

Cómo es la Aprobación Directa?

· Todos los parciales deben ser aprobados con 8 (ocho) o mas puntos en cada uno.

· 75% de asistencia es requerida (*No aplicable durante situación COVID*).

· El alumno podrá tener sólo 1 (uno) recuperatorio.

Se puede repetir un Parcial?

· Si, solo 1 (una) chance para repetir un Parcial con el objetivo de obtener un puntaje mayor, pero el alumno no deberá tener recuperatorios previos.

· La nota quedará firme aunque ésta sea menor que la de la primera instancia.

Lo que aprenderemos en esta materia

Prepárate para cambiar tu forma de pensar y la manera en la que venías programando.

Durante esta cursada vas a aprender a desarrollar aplicaciones web!

SIN COMPILACIÓN

Tan solo comienza a codificar y ejecuta la aplicación sin necesidad de compilarla

EJECUCIÓN EN EL SERVIDOR

Tu aplicación dependerá ahora de un servidor para ser ejecutada

SIN IDE PARA EJECUCIÓN

Todo lo que necesitas para correr tu aplicación es un Web Browser

QUÉ VAMOS A NECESITAR

Software necesario para trabajar
en esta materia

ENTORNO DE TRABAJO (IDE)

Visual Studio Code
Sublime Text

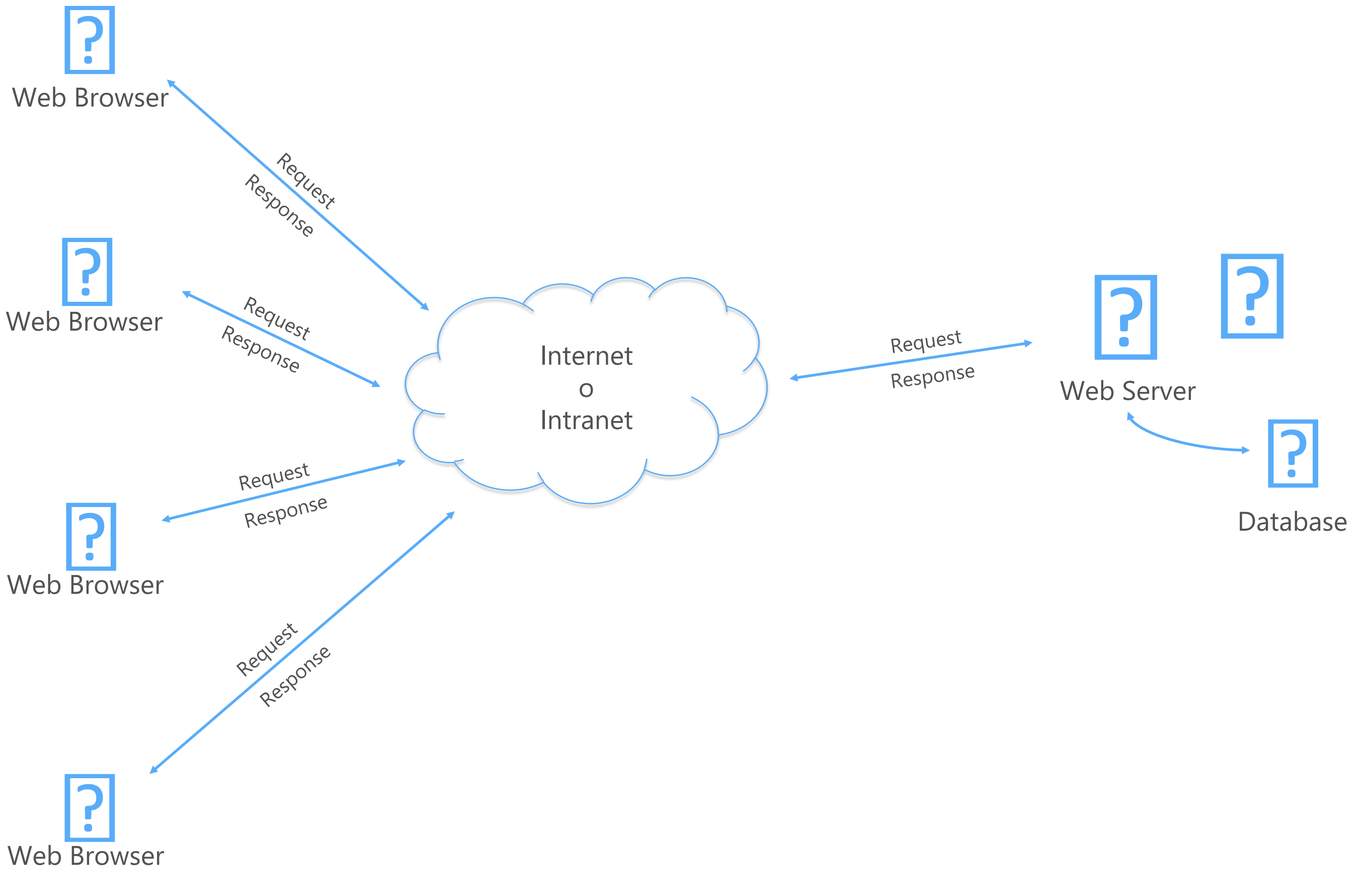
LOCAL SERVER

Wamp Server (Windows)
Xampp Server (Cross Platform)

Arquitectura Web

Aplicación Web es toda aplicación que los usuarios pueden acceder mediante un Web Browser a través de Internet o una Intranet.

Arquitectura Web



HTML Basics

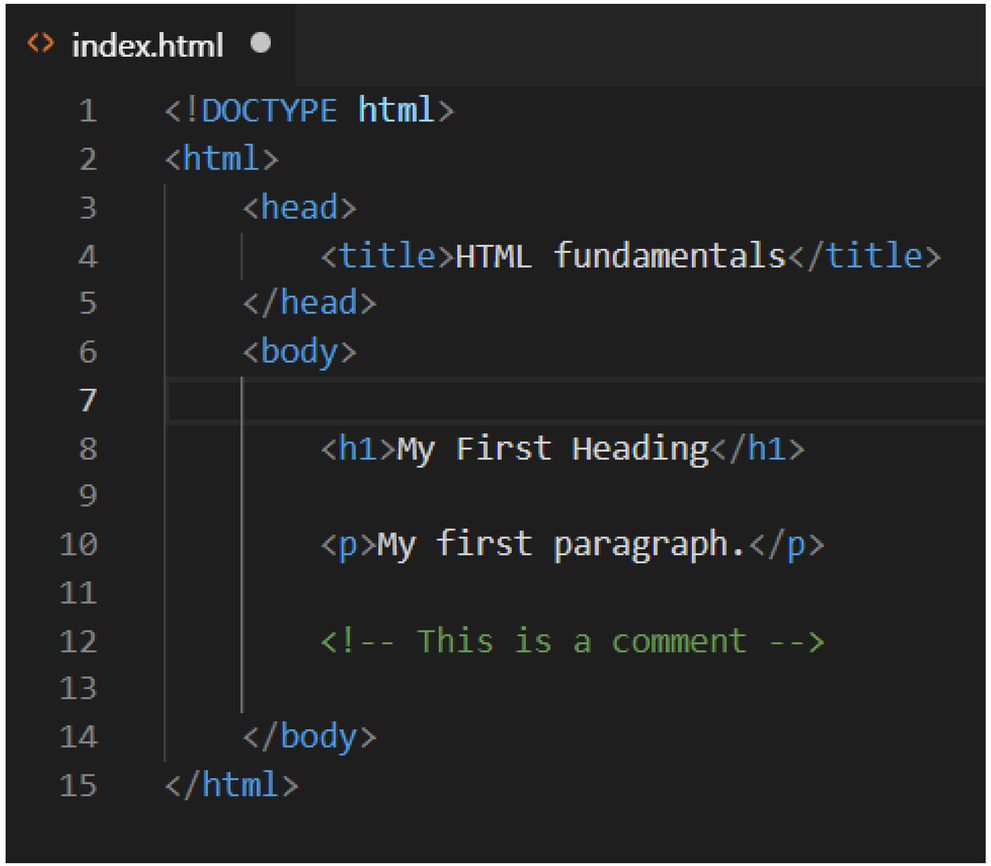
HTML es el lenguaje de etiquetado para Páginas Web.

Con HTML puedes crear tu propio Web Site.

Qué es HTML?

HTML es el lenguaje de etiquetado para la creación de Páginas Web.

- HTML viene de Hyper Text Markup Language
- HTML describe la estructura de una Página Web
- HTML consiste en una serie de elementos
- Los elementos HTML le dicen al browser cómo mostrar el contenido
- Los elementos HTML están representados por etiquetas (**tags**)
- Las HTML tags etiquetan partes del contenido como por ejemplo "heading", "paragraph", "table", etc.
- Los browsers no muestran las tags HTML, pero las usan para renderizar el contenido de la página



Estructura básica de un documento HTML

- **<!DOCTYPE html>** define que el documento es de tipo HTML5
- **<html>** es el elemento raíz de una página HTML
- **<head>** contiene meta información sobre el documento
- **<title>** especifica el título del documento (title bar)
- **<body>** posee el contenido visible de la página
- **<h1>** define un encabezado grande
- ** ** define un párrafo
- **<!–** y **-->** apertura y cierre de código comentado

HTML Tags: **<tagname>**contenido…**</tagname>**

- Generalmente van de a pares, ej.: ** ** y ** **
- Cada una se conoce como *opening tag* y *closing tag*

```html
<body>
    <h1>This is heading 1</h1>
    <h2>This is heading 2</h2>
    <h3>This is heading 3</h3>

    <a href="https://google.com">This is a link to Google</a>

    <br>

    <img
    src="https://raw.githubusercontent.com/JuanAzar/UTN-LabIV/master/Common/Assets/logo-landscape.jpg"
    alt="logo-landscape image" title="UTN Mar del Plata" width="340" height="68">
</body>
```

Estructura básica de un documento HTML

<h1> al **<h6>** definen encabezados de mayor a menor tamaño respectivamente

- **<a>** define un hipervínculo. El atributo **href** indica la URL de destino y entre las tags se especifica el texto a mostrar en el hipervínculo
- **
** salto de línea en HTML, no lleva closing tag
- **<img>** inserta una imagen en el documento, no lleva closing tag y posee los siguientes atributos:

  src origen de la imagen, puede ser un archivo local o un archivo remoto

  alt texto alternativo que se muestra cuando la imagen no logra cargarse

  title texto que se muestra al hacer un **hover over** en la imagen

  width y **height** ancho y alto respectivamente para cambiar el tamaño de forma manual. Si no se especifica ninguno, la imagen se muestra en tamaño original

```html
<body>
    <h2>Unordered List example</h2>
    <ul>
        <li>Coffee</li>
        <li>Tea</li>
        <li>Milk</li>
    </ul>

    <h2>Ordered List example</h2>
    <ol>
        <li>Coffee</li>
        <li>Tea</li>
        <li>Milk</li>
    </ol>

    <h2>Description list example</h2>
    <dl>
        <dt>Coffee</dt>
        <dd>- black hot drink</dd>
        <dt>Milk</dt>
        <dd>- white cold drink</dd>
    </dl>
</body>
```

Trabajando con listas

- **`<ul>`** define una lista desordenada
- **`list-style-type`** es una propiedad CSS que define el tipo de viñeta
- **`<ol>`** define una lista ordenada
- **`type`** es un atributo que define el tipo de numeración
- **`<li>`** corresponde a un list item o elemento de lista
- **`<dl>`** define una lista de descripción
- **`<dt>`** corresponde a un término
- **`<dd>`** corresponde a la descripción de un término
- Las listas pueden anidarse entre sí
- Las listas pueden contener otros elementos HTML
- Se pueden usar las propiedades CSS **`float:left`** o **`display:inline`** para mostrar una lista de manera horizontal

```html
<body>
    <b>This text is bold</b>

    <br>

    <strong>This text is strong</strong>

    <br>

    <i>This text is italic</i>

    <br>

    <em>This text is emphasized</em>

    <h2>HTML <small>Small</small> Formatting</h2>

    <h2>HTML <mark>Marked</mark> Formatting</h2>

    <p>My favorite color is <del>blue</del> red.</p>

    <p>My favorite <ins>color</ins> is red.</p>

    <p>This is <sub>subscripted</sub> text.</p>

    <p>This is <sup>superscripted</sup> text.</p>
</body>
```

Dando formato al texto

- **<b>** define texto en negrita
- **** define texto importante
- **<i>** define texto en cursiva
- **** define texto enfatizado
- **<small>** define texto mas pequeño
- **<mark>** define texto resaltado
- **<del>** define texto tachado
- **<ins>** define texto subrayado
- **<sub>** define texto en subíndice
- **<sup>** define texto en superíndice

```
<body>

    <div>Hello World</div>

    <span>Hello World</span>

</body>
```

Elementos de bloque

- `<div>` define una sección en un documento (**block-level**). Este elemento es utilizado a menudo como contenedor de otros elementos HTML.

- `<span>` define una sección en un documento (**inline**). Este elemento es utilizado a menudo como contenedor de texto.

Nota: Ambos elementos se utilizan con atributos de tipo **style**, **class** y **id**.

Un elemento de tipo **block-level** comienza siempre en una nueva línea y ocupa todo el ancho disponible (se extiende hacia la izquierda y hacia la derecha todo lo que pueda). Ej.: **`<address>` `<article>` `<aside>` `<dd>` `<div>` `<dl>` `<dt>` `<footer>` `<form>` `<h1>-<h6>` `<header>` `<hr>` `<li>` `<main>` `<nav>` `<ol>` `<p>` `<section>` `<table>` `<ul>`**

Un elemento de tipo **inline** no comienza en una nueva línea y ocupa sólo el ancho que sea necesario. Ej.: **`<a>` `<b>` `<br>` `<button>` `<cite>` `<code>` `<em>` `<i>` `<img>` `<input>` `<label>` `<q>` `<script>` `<select>` `<small>` `<span>` `<strong>` `<sub>` `<sup>` `<textarea>`**

```html
<body>
    <table>
        <caption>Employee Information</caption>
        <thead>
            <tr>
                <th>Firstname</th>
                <th>Lastname</th>
                <th>Age</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Jill</td>
                <td>Smith</td>
                <td>50</td>
            </tr>
            <tr>
                <td>Eve</td>
                <td>Jackson</td>
                <td>94</td>
            </tr>
            <tr>
                <td>John</td>
                <td>Doe</td>
                <td>80</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td colspan="3"> </td>
            </tr>
        </tfoot>
    </table>
</body>
```

Tablas

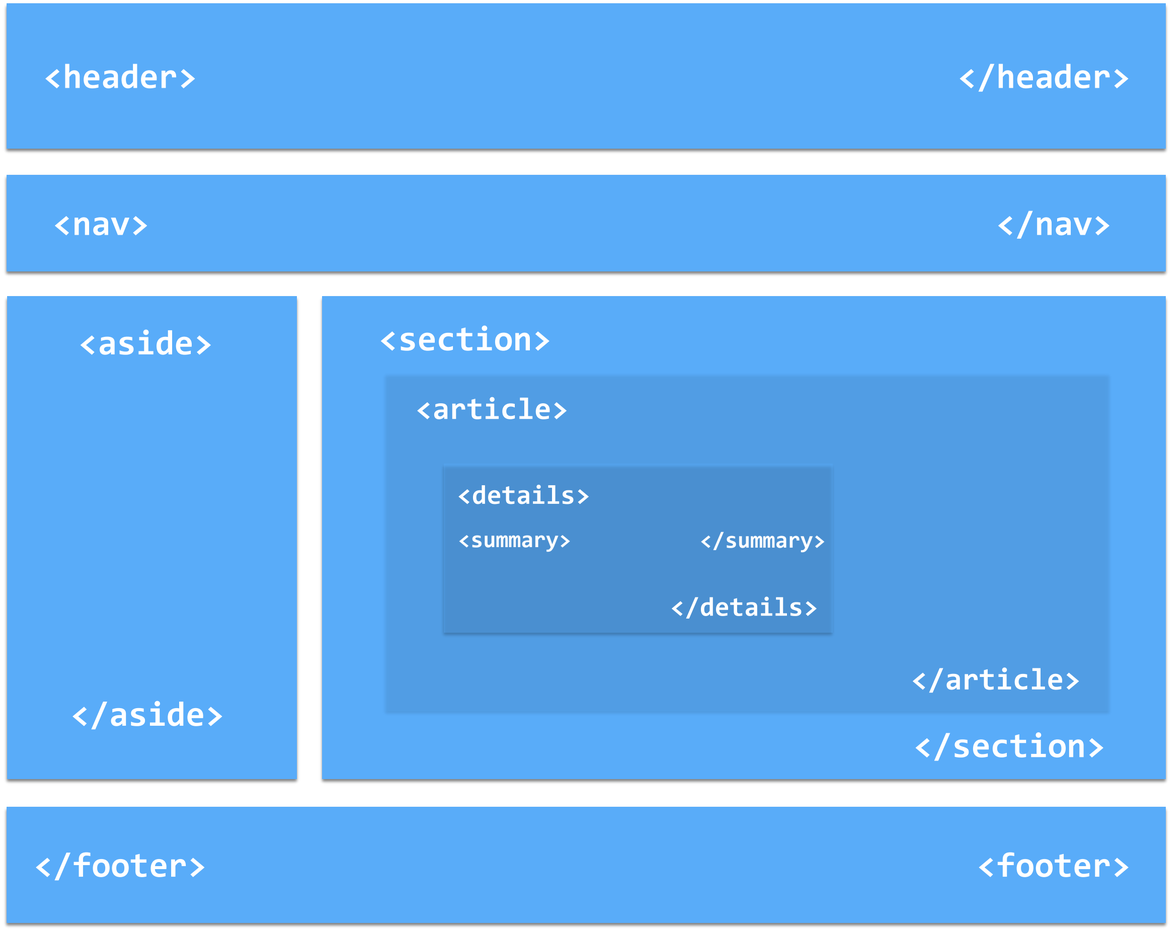
- **`<table>`** define una tabla
- **`<caption>`** define el título de una tabla
- **`<tr>`** define una fila de tabla
- **`<th>`** define una celda de encabezado de tabla
- **`<td>`** define una celda de tabla
- **`<thead>`** agrupa el contenido del **header** en una tabla
- **`<tbody>`** agrupa el contenido del **body** en una tabla
- **`<tfoot>`** agrupa el contenido del **footer** en una tabla

Nota: Los atributos **`colspan`** y **`rowspan`** permiten hacer que una columna o fila abarque mas de una columna o fila respectivamente

```html
<head>
    <title>HTML fundamentals</title>
    <base href="https://raw.githubusercontent.com/JuanAzar/UTN-LabIV/master/Common/Assets/">
    <link rel="stylesheet" href="mystyles.css">
    <meta charset="UTF-8">
    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "Hello JavaScript!";
        }
    </script>
    <style>
        body {
            background-color: powderblue;
        }

        h1 {
            color: red;
        }

        p {
            color: blue;
        }
    </style>
</head>
<body>
    <h1>This is a heading</h1>

    <p id="demo">This is a paragraph</p>

    <button onclick="myFunction()">Click here</button>

    <br><br>

    <img src="logo-portrait.jpg" width="230" height="383">
</body>
```

Head

- **<head>** define información sobre el documento
- **<title>** especifica el título del documento (title bar)
- **<base>** define una dirección/destino por defecto para todos los links relativos de la página
- **<link>** especifica una relación entre el documento y un recurso externo
- **<meta>** define metadata sobre un documento HTML
- **<script>** define un script client-side
- **<style>** especifica estilos para un documento



Layout

- **<header>** define el encabezado para un documento o sección
- **<nav>** define un contenedor para links de navegación (ej.: menú)
- **<section>** define una sección en un documento
- **<article>** define un artículo independiente y auto contenido
- **<aside>** define contenido que no forma parte del contenido principal (ej.: sidebar)
- **<footer>** define el pie de un documento o sección
- **<details>** especifica detalles adicionales. Permite expandir o colapsar su contenido sin necesidad de agregar lógica adicional (*no soportado en IE y Edge*).
- **<summary>** especifica un encabezado para el elemento **<details>**

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark	&apos;	&#39;
á	small letter a with accent	&aacute;	&#225;
é	small letter e with accent	&eacute;	&#233;
í	small letter i with accent	&iacute;	&#237;
ó	small letter o with accent	&oacute;	&#243;
ú	small letter u with accent	&uacute;	&#250;
ñ	n tilde	&ntilde;	&#241;
©	copyright	&copy;	&#169;
®	registered trademark	&red;	&#174;

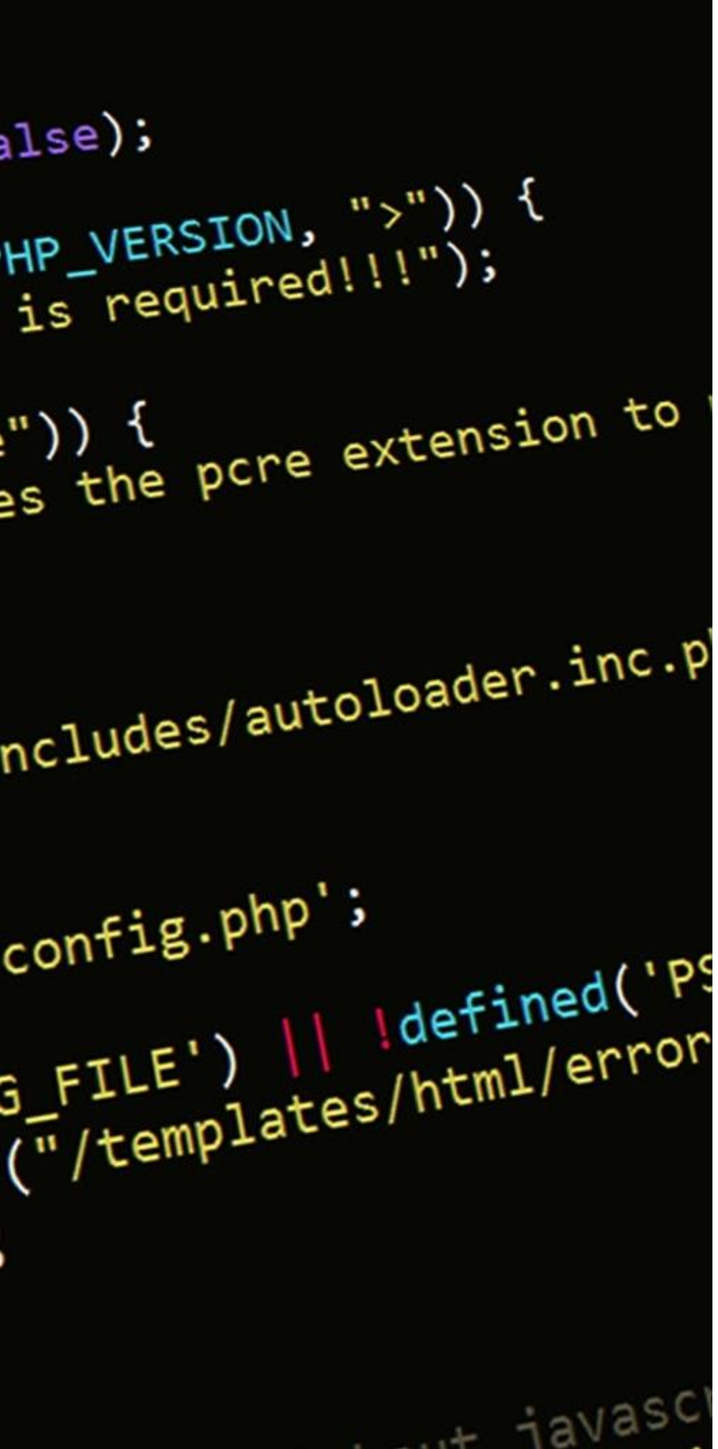
Entidades

- Algunos caracteres son reservados en HTML
- Las entities se utilizan para poder mostrar caracteres reservados en HTML
- Se utiliza **`&entityName`** o **`&#entityNumber`**. Ej.: para mostrar (<) se usa **`<`** o **`&#60;`**
- Ventajas de usar un entity name: Es fácil de recordar
- Desventajas de usar un entity name: Los Browsers pueden no soportar todos los entity names, pero el soporte de entity numbers es muy bueno.

```html
<body>
    <form action="action.php" method="post">
        <input type="text" name="firstName" placeholder="First Name">
        <input type="text" name="lastName" value="Your Last Name">
        <input type="password" name="password" placeholder="Enter password">
        <br>
        <input type="radio" name="answer" value="yes" checked> Yes
        <input type="radio" name="answer" value="no"> No
        <input type="radio" name="answer" value="na"> N/A
        <br>
        <select name="cars">
          <option value="peugeot">Peugeot</option>
          <option value="chevrolet">Chevrolet</option>
          <option value="ford">Ford</option>
          <option value="volkswagen">Volkswagen</option>
        </select>
        <br>
        <textarea name="comments" cols="50" rows="10"></textarea>
        <br>
        <input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>
        <input type="checkbox" name="vehicle2" value="Car"> I have a car
        <br>
        <button type="submit">Send</button>
        <button type="reset">Reset</button>
        <button type="button" onclick="alert('Hello World!')">Say Hello</button>
      </form>
</body>
```

Formularios

- **<form>** define un formulario para recolectar información.
- **<input>** depende del atributo type para determinar cómo se muestra:
  - **text** especifica un input text de una sola línea
  - **password** define un input de tipo password
  - **radio** define un radio button. Aquellos radio buttons con mismo **name** trabajan en conjunto
  - **checkbox** permite elegir cero o mas opciones
- **<select>** define un drop-down list. Contiene varios **<option>**
- **<textarea>** define un input de tipo multilínea
- **<submit>** envía el contenido de los inputs de un form a un **form-handler**
- **<reset>** restablece el contenido de los inputs de un form
- **<button>** botón genérico al cual se le puede añadir acción



PHP

PHP es un lenguaje de programación server-side para crear páginas web dinámicas e interactivas.
PHP es una de las alternativas de lenguajes de programación web como ASP.

Qué es PHP?

PHP es un lenguaje de programación interpretado server-side para la creación de Páginas Web.

- PHP es el acrónimo recursivo de “PHP: Hypertext Preprocessor”
- Los scripts son ejecutados en el servidor (server-side)
- Los archivos PHP pueden contener texto, HTML, CSS, JavaScript y código PHP
- Los archivos PHP llevan la extensión “.php”
- PHP puede manipular archivos (open, close, read, write, delete)
- PHP puede recopilar datos de un form
- PHP permite manipular bases de datos (add, delete, modify, select)
- PHP permite controlar el acceso de usuarios (autenticación y autorización)
- PHP permite encriptar datos
- PHP es cross platform

```php
<html>
    <head>
        <title>PHP fundamentals</title>
    </head>
    <body>
        <?php
            //This is a single-line comment

            #This is also a single-line comment

            /*
                This is a multiple-lines
                comment block
            */

            //PHP Case Sensitivity

            /*NO Keywords, classes, functions and
            user-defined functions are case sensitive */
            echo "Hello World!!";

            ECHO "Hello World!!";

            //*However, variable names are case sensitive */
            $color = "red";

            echo "My car is " . $color . "<br>";
            echo "My house is " . $COLOR . "<br>";
            echo "My boat is " . $coLOR . "<br>";
        ?>
    </body>
</html>
```

Sintaxis básica de PHP

- Un script PHP puede ir en cualquier parte del documento
- Un script PHP comienza con **<?php** y termina con **?>**
- Los archivos PHP llevan extensión ".php"
- Un archivo PHP generalmente contiene HTML tags y código PHP
- El código puede comentarse con **//**, **#** y **/* */**
- La función **echo** muestra texto en la página web
- Los strings pueden ir con comillas dobles o simples
- Las variables se declaran con el símbolo **$** delante
- PHP no es Case Sensitive para palabras reservadas, clases, funciones y funciones definidas por el usuario
- PHP es Case Sensitive para los nombres de variables
- Podemos concatenar texto y variables con el punto (.)

```php
<?php
    $text = "I love PHP!";
    $x = 5;
    $y = 10.5;

    echo "Hey! ".$text . "<br> and this is another line";
    echo "<br>";

    $z = $x + $y;
    echo $z;
    echo "<br>";

    echo $x + $y;

    $a = 5;

    function myGlobalScopeTest() {
        echo "<p>Variable a inside function is: $a<br>";
    }

    myGlobalScopeTest();

    echo "<p>Variable a outside function is: $a<br>";

    function myLocalScopeTest() {
        $b = 5; // local scope
        echo "<p>Variable b inside function is: $b</p>";
    }
    myLocalScopeTest();

    // using x outside the function will generate an error
    echo "<p>Variable b outside function is: $b</p>";

    define("TITLE", "PHP fundamentals");
    echo TITLE;
?>
```

Variables

- Se declaran con el símbolo **$** delante seguido de su nombre
- El nombre debe comenzar con una letra o underscore "_"
- El nombre sólo puede contener letras, números y underscore
- Son case sensitive (**$name** y **$NAME** son diferentes)
- El tipo de dato de la variable depende del valor que se le asigne
- Si se declara **fuera** de una función posee **Global Scope** y sólo puede ser accedida fuera de la función
- Si se declara **dentro** de una función posee **Local Scope** y sólo puede ser accedida dentro de la función

Constantes

- Posee un único valor y no puede cambiarse durante el script
- Se declaran igual que las variables pero sin **$**
- Por convención su nombre va todo en mayúsculas
- Son **Globales** y pueden ser accedidas desde cualquier lado

```php
<?php
    $x = "I Love PHP!";
    $y = 'I Love PHP!';

    echo $x . "<br>";
    echo $y . "<br>";;

    $x = 123456;
    var_dump($x);

    $x = 10.452;
    var_dump($x);

    $x = true;
    $y = false;

    $cars = array("Peugeot", "Chevrolet", "Ford");
    var_dump($cars);

    $x = null;
    var_dump($x);
?>
```

Tipos de datos

- Los **`string`** pueden declararse con comillas simples o dobles
- Los **`integer`** pueden ser negativos o positivos y no deben contener puntos decimales
- Los **`float`** son números con un punto decimal
- Los **`boolean`** pueden ser true o false
- **`NULL`** corresponde a una variable que no tiene ningún valor asignado. Se pueden vaciar las variables asignándoles null
- Los **`array`** pueden contener múltiples valores en una misma variable
- **`Object`** permite guardar objetos. Abordaremos Clases y Objetos más adelante
- **`Resource`** no es un tipo de dato, es más bien la referencia a funciones y recursos externos a PHP, por ejemplo la llamada a una base de datos. Abordaremos este tema mas adelante.

Nota: **`var_dump()`** es una función que permite retornar el tipo de dato y el valor de una variable.

Manejo de Strings

```php
<?php
    echo strlen("I Love PHP!"); // outputs 11
    echo str_word_count("I Love PHP!"); // outputs 3
    echo strrev("I Love PHP!"); //outputs PHP evoL I
    echo strpos("I Love PHP", "PHP"); // outputs 7
    echo str_replace("PHP", "Programming", "I Love PHP!");
    // outputs I Love Programming!
    echo strtoupper("i love php!"); //outputs I LOVE PHP!
    echo strtolower("I LOVE PHP!"); //outputs i love php!
    echo ucfirst("i love php!"); //outputs I love php!
    echo ucwords("i love php!"); //outputs I Love Php!
    echo substr("I Love PHP!", 2); //outputs Love PHP!
    echo substr("I Love PHP!", 2, 4); //outputs Love
    echo trim(" I Love PHP! "); //outputs I Love PHP!
    echo trim("I Love PHP!", "IHP!"); //outputs Love
?>
```

- **`strlen()`** retorna el tamaño de un string
- **`str_word_count()`** retorna la cantidad de palabras de un string
- **`strrev()`** invierte un string
- **`strpos()`** retorna la posición del carácter en el primer match. Si no lo encuentra retorna false
- **`str_replace()`** reemplaza un caracteres dentro de un string
- **`strtoupper()`** convierte el string todo a mayúsculas
- **`strtolower()`** convierte el string todo a minúsculas
- **`ucfirst()`** convierte el primer carácter del string a mayúsculas
- **`ucwords()`** convierte el primer carácter de cada palabra de un string a mayúsculas
- **`substr()`** retorna una parte del string
- **`trim()`** elimina espacios en blanco o algunos caracteres especificados en ambos lados del string.

Operadores Aritméticos

Operador	Nombre	Ejemplo	Resultado
+	Suma	$x + $y	Suma de $x e $y
-	Resta	$x - $y	Diferencia entre $x e $y
*	Multiplicación	$x * $y	Producto de $x e $y
/	División	$x / $y	Cociente de $x dividido $y
%	Módulo	$x % $y	Resto de $x dividido $y
**	Potenciación	$x ** $y	Resultado de elevar $x a la $y potencia

Operadores de Asignación

Asignación	Equivalente	Descripción
x = y	x = y	La variable de la izquierda recibe el valor de la expresión de la derecha
x += y	x = x + y	Suma de $x e $y con asignación en $x
x -= y	x = x - y	Resta entre $x e $y con asignación en $x
x *= y	x = x * y	Multiplicación de $x e $y con asignación en $x
x /= y	x = x / y	Cociente $x dividido $y con asignación en $x
x %= y	x = x % y	Resto de $x dividido $y con asignación en $x

Operadores de Comparación

Operador	Nombre	Ejemplo	Resultado
==	Igualdad	$x == $y	Retorna true si $x es igual a $y
===	Identidad	$x === $y	Retorna true si $x es igual a $y, y además son del mismo tipo
!=	Desigualdad	$x != $y	Retorna true si $x no es igual a $y
<>	Desigualdad	$x <> $y	Retorna true si $x no es igual a $y
!==	No Identidad	$x !== $y	Retorna true si $x no es igual a $y, o si no son del mismo tipo
>	Mayor	$x > $y	Retorna true si $x es mayor que $y
<	Menor	$x < $y	Retorna true si $x es menor que $y
>=	Mayor o igual	$x >= $y	Retorna true si $x es mayor o igual que $y
<=	Menor o igual	$x <= $y	Retorna true si $x es menor o igual que $y

Operadores de Incremento y Decremento

Operador	Nombre	Descripción
++$x	Pre-incremento	Incrementa $x en 1, luego retorna $x
$x++	Post-incremento	Retorna $x, luego incrementa $x en uno
--$x	Pre-decremento	Decrementa $x en 1, luego retorna $x
$x--	Post-decremento	Retorna $x, luego decrementa $x en uno

Operadores Lógicos

Operador	Nombre	Ejemplo	Resultado
`and`	And	`$x and $y`	True si `$x` e `$y` son true
`or`	Or	`$x or $y`	True si `$x` o `$y` es true
`xor`	Or exclusiva	`$x xor $y`	True si `$x` o `$i` es verdadero, pero no ambos
`&&`	And	`$x && $y`	True si `$x` e `$y` son true
`||`	Or	`$x || $y`	True si `$x` o `$y` es true
`!`	Not	`!$x`	True si `$x` no es true

Operadores de Strings

Operador	Nombre	Ejemplo	Resultado
`.`	Concatenación	`$txt1 . $txt2`	Concatenación de `$txt1` y `$txt2`
`.=`	Asignación de concatenación	`$txt1 .= $txt2`	Agrega `$txt2` a `$txt1`

```php
<?php
    date_default_timezone_set("America/Argentina/Buenos_Aires");

    $time = date("H");

    if($time < 10)
        echo "Good morning!";
    elseif($time < 20)
    {
        echo "Have a good day!";
        echo "Work hard too!";
    }
    else
        echo "Have a good night!";


    $favoriteColor = "red";

    switch ($favoriteColor) {
        case "red":
            echo "Your favorite color is red!";
            break;
        case "blue":
            echo "Your favorite color is blue!";
            break;
        case "green":
            echo "Your favorite color is green!";
            echo "This is my favorite color too!";
            break;
        default:
            echo "Your favorite color is neither red, blue, nor green!";
    }
?>
```

Operadores Condicionales

- **`if`** ejecuta código si la condición es true
- **`if...else`** ejecuta código si la condición es true y ejecuta otro código si la condición es false
- **`if...elseif...else`** ejecuta distinto código para más de dos condiciones
- **`switch`** ejecuta diferentes acciones dependiendo diferentes resultados de condiciones

```php
<?php
    $x = 1;

    while($x <= 5)
    {
        echo "The number is: $x <br>";
        $x++;
    }

    $x = 1;

    do
    {
        echo "The number is: " . $x . "<br>";
        $x++;
    }
    while ($x <= 5);

    for ($x = 0; $x < 10; $x++)
    {
        echo "The number is: $x <br>";
    }

    $colors = array("red", "green", "blue", "yellow");

    foreach ($colors as $value)
    {
        echo $value . "<br>";
    }

    foreach ($colors as $key => $value)
    {
        echo $key . " " . $value . "<br>";
    }
?>
```

Loops

- **`while`** ejecuta un bloque de código mientras las condición sea true
- **`do...while`** similar al `while` pero se ejecuta al menos una vez ya que la condición se evalúa luego de ejecutar el código
- **`for`** ejecuta un bloque de código una cantidad de veces específica
- **`foreach`** funciona sólo con *arrays* y se utiliza para iterar sobre pares de tipo *key/value* dentro del *array*

```php
<?php
    function writeMessge()
    {
        echo "I Love PHP!<br>";
    }

    writeMessge();

    function sayHello($name)
    {
        echo "Hello $name.<br>";
    }

    sayHello("John");
    sayHello("Peter");
    sayHello("Eric");

    function setHeight($minheight = 50)
    {
        echo "The height is : $minheight <br>";
    }

    setHeight(350);
    setHeight();

    function sum($x, $y)
    {
        $z = $x + $y;
        return $z;
    }

    echo "5 + 10 = " . sum(5, 10) . "<br>";
?>
```

Funciones

- El nombre de una función debe comenzar con una letra o underscore (no con números)
- Los parámetros se pasan como variables y separados por comas
- Existen parámetros con valor por defecto. Si se llama a la función omitiendo el parámetro, se toma el valor defecto
- Los parámetros con valor por defecto deben indicarse siempre al final del conjunto de parámetros de la función
- Para que una función retorne un valor, se usa el operador **`return`**

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    echo $cars[0] . ", " . $cars[1] . " and " . $cars[2] . "<br>";

    $arrayLength = count($cars);

    echo $arrayLength . "<br>";

    for($x = 0; $x < $arrayLength; $x++) {
        echo $cars[$x]."<br>";
    }

    $age = array("Peter" => "35", "Ben" => "37", "Joe" => "43");

    //or

    $age['Peter'] = "35";
    $age['Ben'] = "37";
    $age['Joe'] = "43";

    $age = array("Peter" => "35", "Ben" => "37", "Joe" => "43");
    echo "Peter is " . $age['Peter'] . " years old<br>";

    foreach($age as $x => $x_value) {
        echo "Key= " . $x . ", Value= " . $x_value . "<br>";
    }

    $cars = array
    (
        array("Volvo", 22, 18),
        array("BMW", 15, 13),
        array("Saab", 5, 2)
    );

    for ($row = 0; $row < 3; $row++) {
        for ($col = 0; $col < 3; $col++) {
          echo $cars[$row][$col];
        }
    }
?>
```

Arrays

- **Arrays indexados:** poseen un índice numérico que siempre comienza en 0 (cero)
- **`count()`** obtiene la cantidad total de elementos del array
- Se puede utilizar **`for`** para recorrer los arrays indexados
- **Array asociativos**: contienen pares de tipo Key/Value.
- Se utiliza **`foreach`** para recorrer los arrays asociativos
- **Arrays multidimensionales:** son arrays que contienen uno o mas arrays. Los mas comunes son los bidimensionales (matrices)
- Se puede utilizar un **`for`** dentro de otro **`for`** para recorrer los arrays multidimensionales

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    sort($cars);

    $numbers = array(4, 6, 2, 22, 11);
    sort($numbers);

    rsort($cars);

    rsort($numbers);

    $age = array("Peter" => "35", "Ben" => "37", "Joe" => "43");
    asort($age);

    ksort($age);

    arsort($age);

    krsort($age);

    array_push($cars,"Peugeot");

    array_shift($cars);

    array_unshift($cars, "Volvo");

    if(in_array("BMW", $cars))
        echo "Match found!";

    if(array_key_exists("Ben", $age))
        echo "Match foound!";

    $keys = array_keys($age);
?>
```

Manejo de Arrays

- **`sort()`** ordena un array de forma ascendente
- **`rsort()`** ordena un array de forma descendente
- **`asort()`** ordena de forma ascendente de acuerdo al value
- **`ksort()`** ordena de forma ascendente de acuerdo a la key
- **`arsort()`** ordena de forma descendente de acuerdo al value
- **`krsort()`** ordena de forma descendente de acuerdo a la key
- **`array_push()`** inserta un elemento al final del array
- **`array_shift()`** remueve y retorna el primer elemento del array
- **`array_unshift()`** inserta un elemento al comienzo del array
- **`in_array()`** true si encuentra el valor en el array, sino false
- **`array_key_exists()`** true si existe la *Key* en el array, sino false
- **`array_keys()`** retorna un array con las *Keys* del array

```php
<body>
    <h1>Additional Functions Example</h1>

    <?php
        include "footer.php";

        require "footer.php";

        include_once "footer.php";

        require_once "footer.php";

        $url = "folder/subFolder/file.php";
        $stringToArray = explode("/", $url);
        var_dump($stringToArray);

        $array = array("Volvo", "BMW", "Toyota");
        $arrayToString = implode("/", $array);
        var_dump($arrayToString);

        function sayHello($firstName, $lastName) {
            echo "Hello $firstName $lastName from PHP!";
        }

        $firstName = "John";
        $lastName = "Doe";

        call_user_func("sayHello", $firstName, $lastName);

        call_user_func_array("sayHello", array($firstName, $lastName));
    ?>
</body>
```

Algunas Funciones de Interés

- **`include`** incluye el contenido de un archivo en el documento actual. Si el archivo no existe, arroja un warning
- **`require`** incluye el contenido de un archivo en el documento actual de forma requerida. Si el archive no existe, arroja un error y se detiene el script
- **`include_once`** igual que `include` pero evita inclusiones repetidas
- **`require_once`** igual que `require` pero evita inclusiones repetidas
- **`explode()`** descompone un string en un array a partir de un delimitador
- **`implode()`** compone un string a partir de un array utilizando un delimitador
- **`call_user_func()`** invoca la función y pasa los parámetros restantes como argumentos
- **`call_user_func_array()`** invoca la función y pasa el contenido de un array como argumentos

HTTP Methods

```html
<body>
    <form action="action.php" method="get">
      <input type="text" name="user" placeholder="Your Username">
      <input type="password" name="password" placeholder="Your Password">
      <button type="submit">Send using GET</button>
    </form>
    <br><br>
    <form action="action.php" method="post">
      <input type="text" name="user" placeholder="Your Username">
      <input type="password" name="password" placeholder="Your Password">
      <button type="submit">Send using POST</button>
    </form>
</body>
```

- El ***Hypertext Transfer Protocol (HTTP)*** está diseñado para habilitar la comunicación entre clientes y servidores
- HTTP trabaja como un protocolo de tipo ***Request-Response*** entre el cliente y el servidor.
- Los HTTP Methods que existen son: **`GET`**, **`POST`**, **`PUT`**, **`HEAD`**, **`DELETE`**, **`PATCH`**, **`OPTIONS`**.
- **`GET`** solicita información. Características: la query es enviada como pares de tipo clave/valor a través de la URL, tiene restricción de tamaño, no debe usarse para enviar información sensible y sólo puede utilizarse para recuperar información (no modificarla)
- **`POST`** envía información al servidor para creación/modificación. Características: la información se envía en el body del HTTP request, no posee restricción de tamaño y puede utilizarse además para recuperar información
- PHP utiliza los superglobals **`$_GET`** y **`$_POST`** para acceder a la información enviada. Esta se guarda como pares *clave/valor* donde clave es el ***name*** otorgado al control de formulario

```php
<?php
    class User
    {
        public $name;
        public $email;

        public function __construct($name, $email)
        {
            $this->name = $name;
            $this->email = $email;
        }
    }

    $user = new User("John", "john@doe.com");
    echo $user->name . " " . $user->email;
    echo "<br>";

    class Student
    {
        private $name;

        public function setName($name)
        {
            $this->name = $name;
        }

        public function getName()
        {
            return $this->name;
        }
    }

    $student = new Student();
    $student->setName("John Doe");

    echo $student->getName();
?>
```

Clases

- Se declaran con la palabra reservada **`class`**
- Los atributos pueden ser **`public`**, **`protected`** o **`private`**
- El constructor se declara como **`function __constructor().`** No es obligatorio declararlo
- Los objetos se instancian con **`new`**
- Respetando la teoría de los 4 pilares de la Programación Orientada a Objetos, todas las clases deberían encapsular sus atributos estableciéndolos como privados/protegidos y exponiendo su funcionalidad a través de métodos **`setters`** y **`getters`**

```php
<?php
    class MyClass
    {
        public $instanceAttribute = 0;
        public static $staticAttribute = 0;

        public function incrementInstanceAttribute()
        {
            $this->instanceAttribute++;
        }

        public static function incrementStaticAttribute()
        {
            MyClass::$staticAttribute++;
        }

        public function getAttributes()
        {
            echo "Instance: " . $this->instanceAttribute . "<br>";
            echo "Static: " . $this::$staticAttribute . "<br>";
        }
    }

    $object1 = new MyClass();
    $object2 = new MyClass();

    $object1->incrementInstanceAttribute();

    $object2->incrementInstanceAttribute();
    $object2->incrementInstanceAttribute();

    $object1::incrementStaticAttribute();

    echo "Object 1: <br>";
    $object1->getAttributes();

    echo "<br>";

    echo "Object 2: <br>";
    $object2->getAttributes();

    echo "<br>";

    echo "Class: <br>";
    echo MyClass::$staticAttribute;
?>
```

Métodos

- Pueden ser de **instancia** o **estáticos**
- Métodos de instancia pueden acceder a atributos estáticos
- Métodos estáticos no pueden acceder a atributos de instancia
- Los miembros de tipo **`static`** se conocen como **de clase**
- En PHP no existe el concepto de **`static class`**
- Un miembro estático puede invocarse desde un objeto concreto o desde la clase, ej.: **`$object->incrementStaticAttribute()`** o **`MyClass::incrementStaticAttribute()`**
- Un miembro de instancia sólo puede ser llamado desde un objeto concreto.

```php
<?php
    abstract class Person
    {
        private $firstName;
        private $lastName;
        private $dni;

        public function getFirstName() { return $this->firstName; }
        public function setFirstName($firstName) { $this->firstName = $firstName; }
        public function getLastName() { return $this->lastName; }
        public function setLastName($lastName) { $this->lastName = $lastName; }
        public function getDni() { return $this->dni; }
        public function setDni($dni) { $this->dni = $dni; }
    }

    class Student extends Person
    {
        private $fileNumber;

        public function getFileNumber() { return $this->fileNumber; }
        public function setFileNumber($fileNumber) { $this->fileNumber = $fileNumber; }
    }

    class Professor extends Person
    {
        private $career;

        public function getCareer() { return $this->career; }
        public function setCareer($career) { $this->career = $career; }
    }

    $student = new Student();
    $professor = new Professor();

    var_dump($student);
    var_dump($professor);
?>
```

Herencia y Clases Abstractas

- Las clases abstractas se declaran anteponiendo **`abstract`**
- Para herencia utilizamos: **`class Child extends Parent`**
- Cada clase debe ir en un archivo específico, ej.: **Person.php**, **Student.php** y **Professor.php**
- Si **Student** y **Professor** extienden de **Person**, cada archivo deberá hacer un **`include/require`** de **Person**
- En PHP sólo tenemos herencia simple

```php
<?php
    class User
    {
        private $email;

        public function getEmail() { return $this->email; }
        public function setEmail($email) { $this->email = $email; }
    }

    interface IUserCollection
    {
        function Add(User $user);
        function GetAll();
    }

    class UserCollection implements IUserCollection
    {
        private $userList;

        public function __construct()
        {
            $this->userList = array();
        }

        public function Add(User $user)
        {
            array_push($this->userList, $user);
        }

        public function GetAll()
        {
            return $this->userList;
        }
    }
?>
```

Interfaces

- Las interfaces se declaran con la instrucción **`interface`**
- Para implementar una interfaz utilizamos: **`class MyClass implements MyInterface`**
- Cada interfaz debe ir en un archivo específico, ej.: **IUserCollection.php**, **IStudentRepository.php** etc.
- Si **UserCollection** implementa **IUSerCollection**, deberá hacer un **`include/require`** de **IUserCollection**
- Se pueden implementar múltiples interfaces
- Debemos usar interfaces en nuestro desarrollo cotidiano para definir contratos de clases que deban ser respetados

```php
<?php
    //Models/Person.php
    namespace Models;

    class Person
    {
        private $firstName;
        private $lastName;

        public function getFirstName()
        {…
        }

        public function setFirstName($firstName)
        {…
        }

        public function getLastName()
        {…
        }

        public function setLastName($lastName)
        {…
        }
    }

    //index.php
    require_once "Models/Person.php";

    use Models\Person as Person;

    $person = new Person();

    $person->setFirstName("John");
    $person->setLastName("Doe");

    var_dump($person);
?>
```

Namespaces

- Un **namespace** o espacio de nombre sirve para encapsular elementos de la misma forma que lo hacen los directorios en un sistema operativo al encapsular archivos
- Para acceder a una clase que se encuentra dentro de un namespace se debe utilizar la palabra reservada **use** seguido de el/los namespaces en que se encuentra la clase. Ej.: **`use Models\Person`** o **`use Parent\Child\Class`**
- Si se referencian dos clases con el mismo nombre, para evitar conflictos se puede utilizar el operador **as** para otorgarle un alias. Ej.: **`use Models\Person as Person`**
- Si el namespace a utilizar se encuentra en un archivo diferente, se requiere que éste sea incluido previamente

```php
<?php
    //Models/Student.php
    namespace Models;

    use Models\Person as Person;

    class Student extends Person
    {
        private $recordId;

        public function getRecordId()
        {
            return $this->recordId;
        }

        public function setRecordId($recordId)
        {
            $this->recordId = $recordId;
        }
    }

    //Autoload.php
    spl_autoload_register(function ($className)
    {
        $fileName = $className.".php";

        require_once($fileName);
    });

    //index.php
    require_once "Config\Autoload.php";

    use Models\Student as Student;

    $student = new Student();

    $student->setFirstName("John");
    $student->setLastName("Doe");
    $student->setRecordId(123456);

    var_dump($student);
?>
```

Autoload

- En una aplicación web real, tendremos muchas clases que utilizar y por lo tanto incluir los archivos uno a uno resultaría engorroso
- La función **`spl_autoload_register()`** provee una manera mas flexible de autocargar clases
- Esta función permite registrar múltiples funciones que PHP colocará en una pila/cola y llamará secuencialmente cuando se declare una nueva clase
- De esta manera invocamos la función una sola vez y dejamos que PHP se encargue de cargar nuestras clases a medida que las utilizamos

```php
class StudentRepository implements IStudentRepository
{
    private $studentList = array();
    private $fileName;

    public function __construct()
    {
        $this->fileName = dirname(__DIR__)."/Data/students.json";
    }

    public function Add(Student $student)
    {
        $this->RetrieveData();
        array_push($this->studentList, $student);
        $this->SaveData();
    }

    public function GetAll()
    {
        $this->RetrieveData();
        return $this->studentList;
    }

    private function SaveData()
    {
        $arrayToEncode = array();

        foreach($this->studentList as $student)
        {
            $valuesArray["recordId"] = $student->getRecordId();
            $valuesArray["firstName"] = $student->getFirstName();
            $valuesArray["lastName"] = $student->getLastName();
            array_push($arrayToEncode, $valuesArray);
        }
        $jsonContent = json_encode($arrayToEncode, JSON_PRETTY_PRINT);
        file_put_contents($this->fileName, $jsonContent);
    }

    private function RetrieveData()
    {
        $this->studentList = array();

        if(file_exists($this->fileName))
        {
            $jsonContent = file_get_contents($this->fileName);
            $arrayToDecode = ($jsonContent) ? json_decode($jsonContent, true) : array();

            foreach($arrayToDecode as $valuesArray)
            {
                $student = new Student();
                $student->setRecordId($valuesArray["recordId"]);
                $student->setFirstName($valuesArray["firstName"]);
                $student->setLastName($valuesArray["lastName"]);
                array_push($this->studentList, $student);
            }
        }
    }
}
```

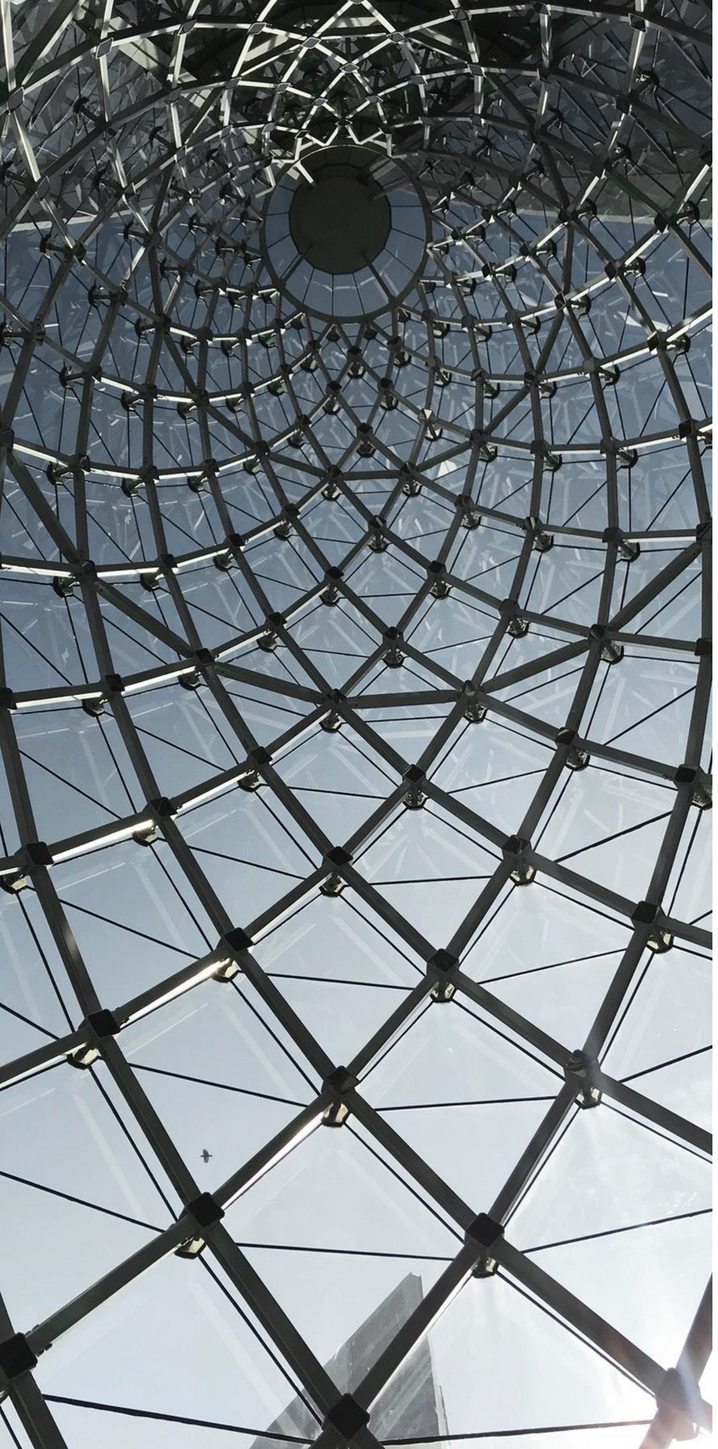
Persistencia con JSON y Archivos

- Codificar y Decodificar nuestras colecciones utilizando JSON y archivos, nos permite persistir información hasta que veamos bases de datos
- **`__DIR__`** es una constante de sistema que devuelve el directorio done se encuentra el archivo que invoca la función
- **`dirname()`** devuelve la ruta de un archivo o directorio dado
- **`json_encode()`** convierte un valor a JSON. El parámetro opcional **`JSON_PRETTY_PRINT`** permite generarlo con estilo indentado para que sea mas legible
- **`json_decode()`** convierte un `string` en formato JSON a una variable en PHP. El parámetro **`true`** indicará que los datos sean devueltos como un `array` asociativo
- **`file_put_contents()`** guarda contenido en un archivo. Si el archivo no existe, lo crea
- **`file_get_contents()`** recupera contenido de un archivo. Debemos comprobar previamente que el archivo exista, lo cual hacemos con **`file_exists()`**

```php
<?php
    //login.php
    require_once("Config/Autoload.php");

    use Models\User as User;

    if($_POST)
    {
        $email = $_POST["email"];
        $password = $_POST["password"];

        if(($email == "john@doe.com") && ($password == "123456"))
        {
            session_start();

            $loggedUser = new User();
            $loggedUser->setEmail($email);
            $loggedUser->setPassword("123456");

            $_SESSION["loggedUser"] = $loggedUser;

            header("location:main.php");
        }
        else
            eader("location:index.php");
    }

    //main.php
    require_once("Config/Autoload.php");

    use Models\User as User;

    session_start();

    if(isset($_SESSION["loggedUser"]))
        $loggedUser = $_SESSION["loggedUser"];
    else
        header("location:index.php");

    //logout.php
    session_start();

    session_destroy();

    header("location:index.php");
?>
```

Session

- Las variables de sesión permiten guardar información para ser utilizada entre múltiples páginas. Dado que HTTP es ***stateless*** no tenemos forma de mantener el estado entre los diferentes *request-response*
- Por defecto las variables de sesión permanecen hasta que se cierre el navegador o bien que se destruya la sesión de forma explícita
- **`session_start()`** función necesaria para crear/abrir una sesión. No puede haber código HTML antes de invocar esta función.
- **`$_SESSION`** es un array asociativo en el cual pueden almacenarse o recuperarse valores
- **`session_destroy()`** destruye una sesión
- **`header()`** envía un encabezado HTTP al cliente. No puede haber código HTML antes de invocar esta función. Ej.: `header("location:index.php")` redirecciona a la página indicada



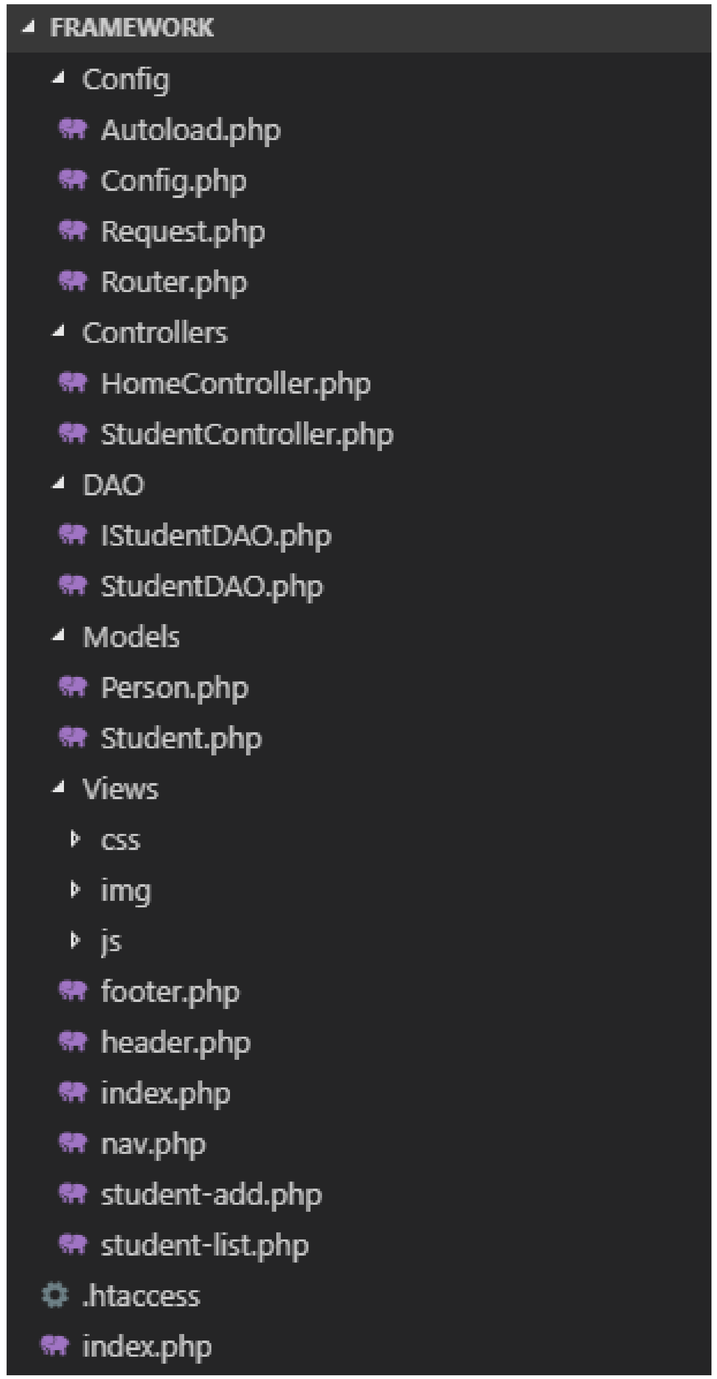
Framework

Un Framework provee una manera estandarizada de crear aplicaciones. Aporta funcionalidad que facilita el desarrollo de las aplicaciones y permite ser modificado o extendido para potenciar su alcance.

Custom Framework

Overview

- Diseñamos un Framework propio para esta asignatura para facilitar el trabajo y permitir el desarrollo en un esquema similar al utilizado en el mercado
- Este Framework trabaja similar a una SPA (*Single Page Application*) donde básicamente tenemos una única página en la que se van a ir *"incrustando"* nuestras vistas
- Toda acción se realizará a través de un formato de URL específico que se traducirá en el llamado a un **`Method`** de un **`Controller`** y pasando parámetros de ser requerido
- Este Controller realizará la acción solicitada y finalmente levantará una vista para devolver información al usuario
- El Framework presentado servirá como soporte para el resto de la materia y es clave fundamental para el **Trabajo Práctico Final** y su integración con **Metodología de Sistemas** y **Bases de Datos**



Arquitectura

- **Config:** Este namespace contiene el core de nuestro framework que se compone de las siguientes clases: **`Autoload`**, **`Request`** y **`Router`**. Además cuenta con un archivo **`Config.php`** donde se definirán constantes comunes

- **Controllers:** Este namespace contiene los Controllers que utilizaremos en nuestro patrón MVC

- **DAO:** namespace que contiene todos los repositorios (DAOs) con sus respectivas interfaces. Aquí también se encontrarán las clases encargadas de gestionar la conexión a la base de datos

- **Models:** namespace que aloja todas las entidades de nuestra aplicación

- **Views:** este folder contiene todas las vistas (*forms*) de nuestra aplicación. Además contiene los **`css`**, **`js`** e **`images`** que sean requeridos

- **Root:** contiene un **`index.php`** que será el punto de partida de nuestra aplicación (*Símil SPA*) y el archivo **`.htaccess`**

- Es importante respetar esta estructura para un correcto funcionamiento de nuestro framework

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l

RewriteRule ^(.*)$ index.php?url=$1 [QSA,L]
```

.htaccess

- Este es un archivo especial que se utiliza en hostings que funcionan con servidores Apache
- Algunas de las utilidades que nos brinda son las siguientes: limitar o bloquear el acceso a determinados directorios, restringir acceso a determinadas direcciones IPs, crear URLs mas amigables o fáciles de reconocer y crear diferentes redirecciones, entre otras.
- No profundizaremos en cómo configurar el .htaccess en esta asignatura, pero básicamente lo que esta configuración hace es detectar cualquier tipo de URL de nuestra aplicación y transformarla en formato: **`index.php?Controller/Action`**
- Si la URL original contiene datos enviados por GET, estos son concatenados al final de la URL
- En conclusión, todos nuestros requests serán redireccionados al index.php de nuestro root folder.

```php
<?php namespace Config;

    class Autoload {

        public static function Start() {
            spl_autoload_register(function($className)
            {
                $classPath = ucwords(str_replace("\\", "/", ROOT.$className).".php");

                include_once($classPath);
            });
        }
    }
?>
```

Autoload

- Nuestro Autoload ahora tendrá algunos cambios. En primera instancia ya no será una función aislada sino que crearemos una **`Class Autoload`**
- El método estático **`Start()`** ejecutará el **`spl_Autoload_register()`**
- Este último ahora, recibirá el $className por parámetro e intentará construir todo el namespace de la clase a ser cargada. Ej.: **`Models\User.php`** o **`DAO\UserDAO.php`**
- Es fundamental respetar el naming convention de los archivos y clases así como el case

```php
<?php namespace Config;

define("ROOT", dirname(__DIR__) . "/");
//Path to your project's root folder
define("FRONT_ROOT", "/UTN/LabIV/Framework/");
define("VIEWS_PATH", "Views/");
define("CSS_PATH", FRONT_ROOT.VIEWS_PATH . "css/");
define("JS_PATH", FRONT_ROOT.VIEWS_PATH . "js/");

?>
```

Config.php

- Este archivo define ciertas constantes importantes para el funcionamiento del framework
- **`ROOT:`** establece la URL absoluta a la carpeta root de nuestra aplicación. Para esto utiliza **`__DIR__`** que es una constante de sistema que devuelve el directorio del archivo donde se invoca y **`dirname()`** que devuelve la ruta de un archivo o directorio dado
- **`FRONT_ROOT:`** indicamos la ruta relativa al directorio root de nuestra aplicación. **Importante!:** Este valor debemos cambiarlo en cada proyecto
- **`VIEWS_PATH:`** Carpeta donde se almacenan nuestras vistas (*forms*)
- **`CSS_PATH:`** Directorio donde se almacenan los archivos CSS
- **`JS_PATH:`** Directorio donde se almacenan los archivos JS

```php
class Request
{
    private $controller;
    private $method;
    private $parameters = array();

    public function __construct()
    {
        $url = filter_input(INPUT_GET, 'url', FILTER_SANITIZE_URL);
        $urlArray = explode("/", $url);
        $urlArray = array_filter($urlArray);

        if(empty($urlArray)) $this->controller = 'Home';
        else $this->controller = ucwords(array_shift($urlArray));

        if(empty($urlArray)) $this->method = 'Index';
        else $this->method = array_shift($urlArray);

        $methodRequest = $this->getMethodRequest();

        if($methodRequest == 'GET')
        {
            unset($_GET["url"]);

            if(!empty($_GET))
            {
                foreach($_GET as $key => $value)
                    array_push($this->parameters, $value);
            }
            else
                $this->parameters = $urlArray;
        }
        elseif ($_POST)
            $this->parameters = $_POST;

        if($_FILES)
        {
            unset($this->parameters["button"]);

            foreach($_FILES as $file)
                array_push($this->parameters, $file);
        }
    }

    private static function getMethodRequest()
    {
        return $_SERVER['REQUEST_METHOD'];
    }
```

Request

- Gestiona el request recibido y lo descompone en un esquema de tipo **`Controller-Method-Parameters`**
- Se recibe un POST o GET con una url y parámetros que pueden ser opcionales. Ej.: **`myapp/Student/GetById/1`** (get) o **`myapp/Student/GetAll`** (get) o **`myapp.com/Student/Add`** (post)
- Request descompone la url recibida de manera tal de obtener el nombre del controller y el method a ejecutar. Ej.: **`myapp/Student/GetAll`** obtiene **`Student`** como controller y **`GetAll`** como Method
- Si no se especifica una url con este formato, por defecto se genera **`Home`** e **`Index`** como controller y method respectivamente
- Dependiendo si se está realizando un POST o GET se obtienen los posibles parámetros enviados ya sea buscando en **`$_POST`** o en **`$_GET`**
- También se controla la posibilidad de recibir archivos (*file upload*) utilizando **`$_FILES`**
- Cada valor obtenido como parámetro se guarda en un `array` asociativo
- Todos estos valores se guardan en properties de la clase

```php
use Config\Request as Request;

class Router
{
    public static function Route(Request $request)
    {
        $controllerName = $request->getcontroller() . 'Controller';

        $methodName = $request->getmethod();

        $methodParameters = $request->getparameters();

        $controllerClassName = "Controllers\\". $controllerName;

        $controller = new $controllerClassName;

        if(!isset($methodParameters))
            call_user_func(array($controller, $methodName));
        else
            call_user_func_array(array($controller, $methodName), $methodParameters);
    }
}
```

Router

- Esta clase tiene como funcionalidad hacer el **routing** en nuestra aplicación
- Posee un método estático **Route** que recibe como parámetro un objeto Request
- Como vimos antes, **Request** nos entrega tres properties (**Controller**, **Method**, **Parameters**) las que utilizaremos para realizar nuestro ruteo. Ej.: **Student**, **GetById**, **1**
- Request concatena “Controller” a nuestro controller y de esta manera quedará: **StudentController**
- Se instancia un objeto de tipo **StudentController**
- Utilizando **call_user_func** o **call_user_func_array** dependiendo si tenemos parámetros o no, se invoca el method del controller. En este caso **GetById** con parámetro **1**
- Conclusión, se ejecuta**: StudentController->GetById(1)**

```php
<?php

    ini_set('display_errors', 1);
    ini_set('display_startup_errors', 1);
    error_reporting(E_ALL);

    require "Config/Autoload.php";
    require "Config/Config.php";

    use Config\Autoload as Autoload;
    use Config\Router   as Router;
    use Config\Request  as Request;

    Autoload::start();

    session_start();

    require_once(VIEWS_PATH."header.php");

    Router::Route(new Request());

    require_once(VIEWS_PATH."footer.php");
?>
```

index.php

- Como vimos anteriormente, el framework utiliza un **`index.php`** que es el punto de partida de la aplicación como si fuese una SPA
- Este archivo contiene instrucciones que indican que queremos mostrar cualquier error para por debuggear con más facilidad
- Se hace un require de **`Autoload`** y **`Config`**
- Se inicializa **`Autoload`**
- Se inicia **`session`**
- Se puede hacer un require de vistas que serán fijas (*cross-application)* como `header.php` y `footer.php` (*opcional*)
- Se invoca **`Router::Route(new Request())`**
- Conclusión: cada vez que proporcionamos una URL el **`.htaccess`** rescribe la url y redirecciona al **`index.php`** el cual utiliza el **`Router`** y **`Request`** para transformar la URL en un esquema **`Controller-Method-Parameters`** y ejecutar la acción solicitada
- El controller instanciado ejecuta el método requerido, realiza las acciones y renderiza la vista correspondiente.

```php
<form action="<?php echo FRONT_ROOT ?>Student/Add" method="post" class="bg-light-alpha p-5">
    <div class="row">
        <div class="col-lg-4">
            <div class="form-group">
                <label for="">Legajo</label>
                <input type="text" name="recordId" value="" class="form-control">
            </div>
        </div>
        <div class="col-lg-4">
            <div class="form-group">
                <label for="">Nombre</label>
                <input type="text" name="firstName" value="" class="form-control">
            </div>
        </div>
        <div class="col-lg-4">
            <div class="form-group">
                <label for="">Apellido</label>
                <input type="text" name="lastName" value="" class="form-control">
            </div>
        </div>
    </div>
    <button type="submit" name="button" class="btn btn-dark ml-auto d-block">Agregar</button>
</form>
```

Views

- Nuestros formularios ahora tendrán un pequeño cambio en su **`action`**
- El `action` deberá anteponer el **`FRONT_ROOT`** para obtener la ruta relativa al root de nuestra aplicación y concatenamos el nombre del `Controller` y el `Method` que queremos ejecutar. Ej.: **`<?php echo FRONT_ROOT ?>Student/Add`**
- En este ejemplo, estaremos ejecutando el Method **`Add`** para el **`StudentController`** y se pasará por POST los parámetros **`recordId`**, **`firstName`** y **`lastName`**, de esta manera la llamada final será: **`$controller->Add($recordId, $firstName, $lastName)`**

```php
namespace Controllers;

use DAO\StudentDAO as StudentDAO;
use Models\Student as Student;

class StudentController
{
    private $studentDAO;

    public function __construct()
    {
        $this->studentDAO = new StudentDAO();
    }

    public function ShowAddView()
    {
        require_once(VIEWS_PATH."student-add.php");
    }

    public function ShowListView()
    {
        $studentList = $this->studentDAO->GetAll();

        require_once(VIEWS_PATH."student-list.php");
    }

    public function Add($recordId, $firstName, $lastName)
    {
        $student = new Student();
        $student->setRecordId($recordId);
        $student->setfirstName($firstName);
        $student->setLastName($lastName);

        $this->studentDAO->Add($student);

        $this->ShowAddView();
    }
}
```

Controllers

- En este ejemplo tenemos el **`StudentController`** que utiliza un **`StudentDAO`** para persistir la información
- El Controller tiene dos métodos **`ShowAddView()`** que renderiza la vista de **`student-add.php`** y el **`ShowListView`**() que renderiza a vista de **`student-list.php`**. Esta última requiere previamente hacer un **`GetAll()`** en el **`StudentDAO`** para traer la información que queremos mostrar
- El método **`Add()`** recibe por parámetro las variables que estaremos enviando desde el formulario en el POST y con esto generaremos un **`Student`** nuevo el cual persistiremos.

Acceso a Bases de Datos

Trabajar con acceso a bases de datos en programación web es crucial para poder desarrollar aplicaciones que puedan persistir información en un lugar seguro y controlado.

PHP permite el acceso a datos utilizando drivers de conexión a la mayoría de Sistemas de Gestión de Base de Datos conocidos en el mercado

En nuestro caso, y como parte de la integración con la asignatura **Bases de Datos**, nos centraremos en al acceso a datos a través de MySQL

Qué es PDO?

Overview

- PDO no solo ofrece soporte para MySQL sino para mas de 12 sistemas de base de datos
- PDO es nativo y está habilitado por defecto a partir de PHP 5.1.0
- PDO nos permite instanciar un objeto con un recurso de conexión, ejecutar las consultas necesarias y la conexión se finalizará automáticamente cuando el script PHP finalice
- Trabaja orientado a objetos para mayor simplicidad
- A continuación veremos cómo utilizar PDO para manejar conexiones a bases de datos MySQL y luego veremos la clase **`Connection`** que formará parte de nuestro Framework y funciona como *wrapper* de PDO para facilitar su uso

PDO – PHP Data Objects

```php
<?php
    define("DB_HOST", "localhost");
    define("DB_NAME", "University");
    define("DB_USER", "root");
    define("DB_PASS", "");

    $recordId = 123;
    $firstName = "John";
    $lastName = "Doe";

    $pdo = new PDO("mysql:host=".DB_HOST."; dbname=".DB_NAME, DB_USER, DB_PASS);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    //Execute SELECT statement
    $selectStatement = $pdo->prepare("SELECT recordId, firstName, lastName FROM students");

    $selectStatement->execute();

    $result = $selectStatement->fetchAll();

    var_dump($result);

    //Execute INSERT statement
    $insertStatement = $pdo->prepare("INSERT INTO students (recordId, firstName, lastName)
                                      VALUES (:recordId, :firstName, :lastName)");

    $insertStatement->bindParam(":recordId", $recordId);
    $insertStatement->bindParam(":firstName", $firstName);
    $insertStatement->bindParam(":lastName", $lastName);

    $insertStatement->execute();
?>
```

- Definimos constantes para nuestros datos de conexión: **`DB_HOST`**, **`DB_NAME`**, **`DB_USER`**, **`DB_PASS`**
- **`new PDO("mysql:host=".DB_HOST."; dbname=".DB_NAME, DB_USER, DB_PASS)`** crea un objeto PDO estableciendo una conexión a la base da datos. Si algún dato es incorrecto se genera un error
- **`$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)`** establece el modo de errores a Excepciones. Cada error que se genera a nivel de PDO lanzará una excepción que podremos manejar en un bloque **`try/catch`**
- **`$pdo->prepare()`** permite ejecutar sentencias SQL de forma repetida utilizando valores no especificados (*parámetros*). Estos se indican como **`:parameterName`** o **`?`** dependiendo si se usa en una query o en un stored procedure
- **`$statement->bindParam()`** enlaza un parámetro a un valor concreto
- **`$statement->execute()`** ejecuta la sentencia SQL
- **`$statement->fetchAll()`** se utiliza para sentencias de tipo **`SELECT`** y retorna un array de resultados, donde cada fila es a su vez un array asociativo

Connection

```php
namespace DAO;

use \PDO as PDO;
use \Exception as Exception;
use DAO\QueryType as QueryType;

class Connection
{
    private $pdo = null;
    private $pdoStatement = null;
    private static $instance = null;

    private function __construct()
    {...
    }

    public static function GetInstance()
    {...
    }

    public function Execute($query, $parameters = array(), $queryType = QueryType::Query)
    {...
    }

    public function ExecuteNonQuery($query, $parameters = array(), $queryType = QueryType::Query)
    {            ...
    }

    private function Prepare($query)
    {...
    }

    private function BindParameters($parameters = array(), $queryType = QueryType::Query)
    {...
    }
}
```

- Wrappea PDO abstrayendo su lógica de los repositories
- **`Constructor():`** Inicializamos un objeto de tipo PDO proporcionando el **`server`**, **`database`**, **`user`** y **`password`** que tenemos en nuestro **`Config.php`**
- **`GetInstance():`** Utilizaremos un patrón **`Singleton`** para nuestra clase Connection evitando múltiples sesiones en un mismo proceso
- **`Exeute($query, $parameters, $queryType):`** Ejecuta una query SQL de tipo **`SELECT`**, recibe parámetros (*opcional*) y un **`$queryType`**. Retorna una matriz de resultados
- **`ExecuteNonQuery($query, $parameters, $queryType):`** Ejecuta una query SQL de tipo **`INSERT`**, **`UPDATE`**, **`DELETE`**, recibe parámetros (*opcional*) y un **`$queryType`**. Retorna la cantidad de filas afectadas
- **`Prepare($query):`** método privado que ejecuta un **`prepare`** interno de PDO para preparar la consulta a ejecutar
- **`BindParameters($parameters, $queryType):`** Dependiendo el **`$queryType`** realiza el armado de los parámetros que serán enviados en la query.
- **`$queryType:`** Indica si la query es **SQL plano** o un **Stored Procedure**

Bibliografía y recursos

HTML Basics, PHP Fundamentals, JavaScript, CSS : https://www.w3schools.com/

PHP Documentation: https://www.php.net/manual/en/

MySQL Documentation: https://dev.mysql.com/doc/

Extras: http://www.htaccess-guide.com/

Campus: http://campus.mdp.utn.edu.ar/course/edit.php?id=169

GitHub Repository: https://github.com/JuanAzar

YouTube Channel: https://www.youtube.com/c/JuanJoseAzar


¿Preguntas?


END
GRACIAS POR ACOMPAÑARNOS
EN ESTE CAMINO