



# ESTRUCTURAS DE DATOS Y ALGORITMOS 1 (Convenio LATAM)

Desarrollo Web

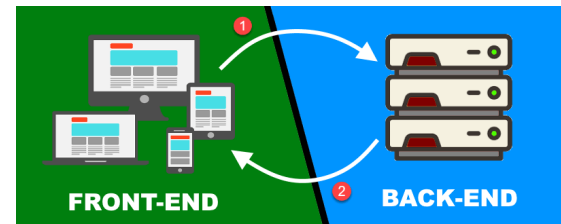
Profesor  
Orlando Arboleda Molina  
(Uso exclusivo de docentes)\*



ACREDITACIÓN  
INSTITUCIONAL  
DE ALTA CALIDAD  
Vigilada MinEduación.  
Pres. No. 16743, 2017-2021.

## DESARROLLO WEB

- El **desarrollo web** es un término que define la creación de **sitios web** para internet o una **intranet**. Para conseguirlo se hace uso de tecnologías **software** del **lado** del **cliente** y del **servidor** (Wikipedia, 2012). Áreas de aplicación:
  - desarrollo de **front-end**
  - desarrollo de **back-end**



## DESARROLLO FRONT-END

- Es el desarrollo de la **GUI** de un **sitio web**, para que los usuarios puedan **ver** e **interactuar** con ese sitio web.

- Se suele usar:



```
Index.html x
Index.html > html > body > main > div > p
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>FrontEnd</title>
8   <link rel="stylesheet" href="Styles/style.css">
9 </head>
10 <body>
11   <main>
12     <div class="coloreado">
13       <h1 id="definicion">¿Qué es Frontend?</h1>
14       <p>Pulsar la imagen para ir a la pagina de referencias
15         <a href="https://assemblerinstitute.com/blog/backend-vs-frontend">
16         </a>
17       </p>
18       <p>Hace referencia a la parte <strong>visible o interfaz de usuario</strong>
19         aquella que verán y interactúan los usuarios que accedan al sitio web.
20       </p>
21       <p>Se encargan de las animaciones y demás elementos que pueden visualizar los usuarios</p>
22     </div>
23   </main>
24 </body>
25 </html>
```

```
style.css x
Styles > # style.css > ...
1 h1 {
2   color: blue;
3   font-size: 30px;
4   font-family: 'Times New Roman', Times, serif;
5 }
6
7 h2 {
8   color: blueviolet;
9   font-size: 20px;
10 }
11
12 img {
13   width: 100px;
14   height: 100px;
15 }
16
17 .coloreado {
18   color: brown;
19 }
20
21
22
23
```

```
JS main.js x
Scripts > JS main.js > ...
14 // obtiene boton2
15 const entradaBoton = document.getElementById("consultaBoton2");
16
17 // se asignan eventos
18 boton1.addEventListener('click', calculaIMC)
19 seleccion.addEventListener('change', mostrarPulsar)
20
21 function calculaIMC(){
22   veces++;
23   const nombre = prompt('Introduce tu nombre');
24   const peso = parseFloat(prompt('Introduce tu peso'));
25   const estatura = parseFloat(prompt('Introduce tu estatura'));
26   const IMC = obtenerIMC(peso, estatura);
27 }
28
```

## HTML

- El **HTML** (HyperText Markup Language) o *lenguaje de marcado de hipertexto*, es el lenguaje para definir el **significado** y **estructura** del contenido web. Utiliza **marcas** para etiquetar **texto**, **imágenes** y **otro contenido** para **mostrarlo** en un **navegador Web**.
- Una **pagina html** se construye usando **etiquetas html**, con el siguiente **formato** (los *atributos* y *contenido* son opcionales, dependiendo de la etiqueta):  
**<etiqueta atributosOpcionales> contenidoOpcional </etiqueta>**
- Es posible **anidar** algunas **etiquetas html**.

**Ejemplo:** se presentan tres ejemplos de etiquetas html.

```

```

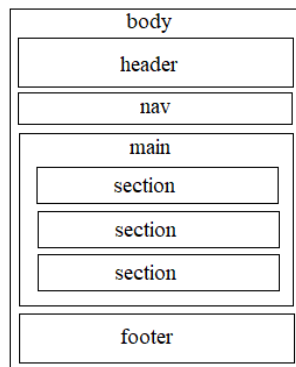
```
<a href="acerca.html">Acerca de</a>
```

```
<li><a href="acerca.html">Acerca de</a></li>
```

## HTML

Ejemplo: a) estructura básica de un archivo html 5 b) la estructura sugerida del *body* de una página html5, una página html de ejemplo y como se visualizaría.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
</body>
</html>
```



```
<header>
  
  <h1>Estructura de Datos y Algoritmos 1</h1>
</header>
<nav>
  <ul>
    <li><a href="#">Inicio</a></li><li><a href="#">Acerca de</a></li>
  </ul>
</nav>
<main>
  <section>
    <h2>Presentación</h2>
    <p>El curso de Estructuras de Datos y Algoritmos 1 contribuye a la formación en el proceso de comprensión de los problemas reales que son eficientes y que pueden ser desplegadas de forma local o en la web.
    <p>La asignatura está ubicada en el ciclo de formación profesional. Su propósito formativo es apoyar la formalización de un proceso de aprendizaje continuo.
  </section>
  <section>
    <h2>Objetivo de la asignatura</h2>
    <p>Seleccionar e implementar algoritmos computacionales correctos, eficientes y desplegados en la web, para resolver problemas reales.
  </section>
  <section>
    <h2>Metodología</h2>
    <p>La tipología de la asignatura es la predominancia del conocimiento procedimental (PCP), ya que se busca desarrollar en los estudiantes una serie de ejercicios cortos con criterio de progresión; es decir, yendo de lo simple a lo complejo, con prácticas que incluyan el uso de herramientas de programación.
  </section>
</main>
<footer>
  <address>
    Autor: Orlando Arboleda Molina<br>Correo: oarboleda@uao.edu.co
  </address>
  <p>Última modificación: 18 de Julio de 2023</p>
</footer>
```

**Estructura de Datos y Algoritmos 1**

- Inicio
- Acerca de

**Presentación**

El curso de Estructuras de Datos y Algoritmos 1 contribuye a la formación en el proceso de comprensión de los problemas reales que son eficientes y que pueden ser desplegadas de forma local o en la web.

La asignatura está ubicada en el ciclo de formación profesional. Su propósito formativo es apoyar la formalización de un proceso de aprendizaje continuo.

**Objetivo de la asignatura**

Seleccionar e implementar algoritmos computacionales correctos, eficientes y desplegados en la web, para resolver problemas reales.

**Metodología**

La tipología de la asignatura es la predominancia del conocimiento procedimental (PCP), ya que se busca desarrollar en los estudiantes una serie de ejercicios cortos con criterio de progresión; es decir, yendo de lo simple a lo complejo, con prácticas que incluyan el uso de herramientas de programación.

Autor: Orlando Arboleda Molina  
Correo: oarboleda@uao.edu.co

## HTML

Ejemplo: se listan algunas de las etiquetas ampliamente usadas.

Documento	Etiqueta	Significado	Observaciones
<head> otras etiquetas </head> ( encabezado )	<title>Texto</title>	Especifica el título de la página web	
	<link rel="stylesheet" href=URL>	Especifica la relación entre el documento actual y un recurso externo	Permite enlazar una hoja de estilos
	<script type="module" src=URL>	Especifica la relación entre el documento actual y un modulo de javascript	Permite enlazar la lógica frontEnd
<body> otras etiquetas </body> ( cuerpo )	<header> otras etiquetas encabezado </header>	Especifica un grupo de ayudas introductorias o de navegación	
	<main> otras etiquetas encabezado </main>	Especifica el contenido principal del documento o aplicación	
	<footer> otras etiquetas pie de pagina </footer>	Especifica un pie de página	
	<nav> otras etiquetas pie de pagina </nav>	Especifica una sección para enlaces de navegación	
	<section> otras etiquetas </section>	Especifica una sección genérica	Esta diseñada para contenidos dependientes, pero diferenciados
	<p>Texto</>	Especifica un párrafo	



## HTML

Ejemplo: se listan algunas de las etiquetas ampliamente usadas.

Documento	Etiqueta	Significado	Observaciones
<body> otras etiquetas </body>	<!--Texto -->	Especifica un comentario	
	<h1>Texto</h1> ... <h6>Texto</h6>	Especifica niveles de encabezado de una sección	Va desde h1, h2, .. h6. Donde h1 es el mas alto y h6 el mas bajo.
	<a href=URL>Texto</p>	Especifica un enlace a otros URLS	Si href=#Id va al componente con dicho id en la pagina
	<img src=URL alt=Texto alternativo>	Especifica una Imagen	
	 	Especifica un salto de línea	
	<ul> Items </ul>	Especifica una lista no ordenada	
	<ol>Items</ol>	Especifica una lista no ordenada	Cada ítem se construye con la etiqueta <li>Texto</li>
	<form action=URI method=metodoHTTP> otras etiquetas input o button </form>	Especifica una sección que contiene controles interactivos para enviar información a un servidor web.	El metodoHTTP puede ser: <b>post</b> (los datos son incluidos en el cuerpo del formulario) o <b>get</b> (los datos son adjuntados a la URI)

## HTML

Ejemplo: se listan algunas de las etiquetas ampliamente usadas.

Documento	Etiqueta	Significado	Observaciones
<body> otras etiquetas </body>	<input type=tipo id=nombre otrosAtributos> Texto </input>	Especifica controles interactivos para formularios basados en la web.	Algunos tipos posibles son button, checkbox, date, time, email, file, number, password, radio, range, text y submit.
	<button name=nombre otrosAtributos> Texto </input>	Especifica un element interactivo activado con el mouse, teclado, etc	
	<table> <caption>texto </caption> <thead><tr> encabezados</tr> .. </thead> <tbody><tr> celdas</tr> .. </tbody> <tfoot><tr> celdas</tr> .. </tfoot> </table>	Especifica una tabla o representación tabular de los datos	Cada encabezado se construye con etiquetas <th>Texto</th> Cada celda se construye con etiquetas <td>Texto</td>
	<textarea otrosAtributos> Texto </textarea>	Especifica un control para la edición multilínea de texto sin formato	
	<fieldset otrosAtributos> <legend> Texto </legend> controles input </fieldset>	Especifica una agrupación de los campos de un formulario	



## Ejercicio de Implementación

- Realizar los ajustes solicitados en el [Caso1\\_DesarrolloWeb\\_FrontEnd](#) a la pagina web. Para esta actividad, se debe disponer de:
  - El entorno de desarrollo [Visual Studio Code](#) (en el cual se debe instalar la extensión [Live Server](#))
  - Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).

## CSS

- Las **CSS** (Cascade Style Sheets) o *hojas de estilo en cascada*, es el lenguaje para definir la **presentación** de un documento escrito en HTML o XML.
- Para **alterar** el **estilo** de los elementos HTML, se aplican reglas, con el siguiente **formato**:

```
selector {  
    propiedad1 : valor1 ;  
    ----  
    propiedadn : valorn ;  
}
```

**Ejemplo:** estilos sobre los selectores html para encabezado de nivel1 y pie de pagina

```
h1 {  
    color: blue;  
    font-size: 30px;  
    font-family: 'Times New Roman', Times, serif;  
}  
  
footer {  
    margin: 20px;  
    background-color: #000099;  
    font-size: large  
    color: white;  
}
```

## CSS

Ejemplo: se listan algunas propiedades ampliamente usadas en CSS.

Propiedad	Significado	Valores Posibles
background-color	Especifica el color de fondo	<ul style="list-style-type: none"> <li>Nombre (ej. blue)</li> <li>RGB en decimal, usando en formato hexadecimal #RGB (ej. #0000FF)</li> <li>RGB en formato rgb(R,G,B) (ej. rgb(0,0,255))</li> </ul>
color	Especifica el color	
width	Especifica el ancho	<ul style="list-style-type: none"> <li>Porcentaje (ej. 80%)</li> <li>Unidad en pixeles (ej. 120px)</li> </ul>
height	Especifica el alto	
border	Especifica el <i>ancho, estilo</i> y <i>color</i> del borde	<p>Algunos estilos posibles son:</p> <ul style="list-style-type: none"> <li>none (oculto)</li> <li>dotted (punteado) y dashed (rayado)</li> <li>solid (línea continua)</li> <li>outset (profundidad hacia afuera)</li> </ul> <p>(ej. 5px dotted blue)</p>
border-radius	Especifica que tan redondeadas están las esquinas del borde	<ul style="list-style-type: none"> <li>Un único valor para todos las 4 esquinas (ej. 10px)</li> <li>Un valor inicial para <i>top-left</i> y <i>bottom-right</i> y el segundo para <i>top-right</i> y <i>bottom-left</i> (ej. 10px 5%)</li> <li>Un valor inicial para <i>top-left</i>, el segundo para <i>top-right</i> y <i>bottom-left</i> y el tercero para <i>bottom-right</i> (ej. 2px 4px 2px)</li> <li>Cuatro valores, en el sentido de las manecillas del reloj, empezando desde <i>top-left</i> (ej. 1px 0 3px 4px)</li> </ul>

## CSS

**Ejemplo:** se listan algunas propiedades ampliamente usadas en CSS.

Propiedad	Significado	Valores Posibles
margin	Especifica el margen para los cuatro lados. Sirve para separar el elemento del elemento que lo contiene	<ul style="list-style-type: none"> <li>• Un único valor para todos los 4 lados (ej. 10px)</li> <li>• Un valor inicial para arriba-abajo y el segundo para izquierda-derecha (ej. 0 auto)</li> <li>• Un valor inicial para arriba, el segundo para izquierda-derecha y el tercero para abajo (ej. 0 auto 5px)</li> <li>• Cuatro valores, en el sentido de las manecillas del reloj, empezando arriba (ej. 0 auto 0 auto)</li> </ul>
padding	Especifica el área de relleno. Sirve para aumentar el espaciado interno (relleno interno)	

## CSS

- Las **reglas** pueden ser aplicadas a los siguientes **selectores**:

Selector	Se utiliza para	Ejemplo
Universal	Seleccionar todos los elementos de la pagina. Se indica con <code>*</code> .	<pre>* {   margin: 0;   padding: 0; }</pre>
Tipo o etiqueta	Seleccionar todos los elementos de la pagina cuya etiqueta HTML coincide con el valor del selector. Las etiquetas se pueden agrupar usando el formato: <code>selector<sub>1</sub> , selector<sub>2</sub> , ... , selector<sub>N</sub></code>	<pre>h1 {   color: red; } h1, h2, h3 {   color: #8A8E27;   font-weight: normal; }</pre>
Descendente	Seleccionar los elementos que se encuentran dentro de otros elementos. Un elemento es descendiente de otro cuando se encuentra entre las etiquetas de apertura y de cierre del otro elemento. Tiene el formato: <code>selector<sub>1</sub> selector<sub>2</sub> ...selector<sub>N</sub></code>	<pre>&lt;p&gt; ... &lt;span&gt;texto1&lt;/span&gt; ... &lt;a href=""&gt;...&lt;span&gt;texto2&lt;/span&gt;&lt;/a&gt; ... &lt;/p&gt;  p span { color: red; }</pre>
Clase	Seleccionar los elementos que tienen el atributo <b>class</b> de HTML.. Tiene el formato: <code>.valorClass</code>	<pre>&lt;body&gt; &lt;p class="destacado"&gt;Lorem ipsum dolor sit amet...&lt;/p&gt; &lt;p&gt;Nunc sed lacus et est adipiscing accumsan...&lt;/p&gt; &lt;/body&gt;  .destacado { color: red; }</pre>
ID	Seleccionar el único elemento que tiene el atributo <b>id</b> de HTML.. Tiene el formato: <code>#valorId</code>	<pre>&lt;p&gt;Primer párrafo&lt;/p&gt; &lt;p id="destacado"&gt;Segundo párrafo&lt;/p&gt; &lt;p&gt;Tercer párrafo&lt;/p&gt;  #destacado { color: red; }</pre>

## CSS

- Se pueden aplicar de las siguientes maneras:

1. Archivos en línea

```
<p>
Esto es un párrafo en varias palabras <span style="color:green">en color verde</span>.
</p>
```

2. Elemento Style

```
<html>
<head>
  <title>Ejemplo de estilos para toda una página</title>
  <style>
    h1 { text-decoration: underline; text-align: center }
    p { font-family: arial,verdana; color: white; background-color: black }
    body { color: black; background-color: #cccccc; text-indent: 1cm }
  </style>
</head>
```

3. Con archivos css  
(así trabajaremos)

```
h1 {
  color: blue;
  font-size: 30px;
}
h2 {
  color: blueviolet;
  font-size: 20px;
}
```

```
<title>FrontEnd</title>
<link rel="stylesheet" href="Styles/style.css">
</head>
```

Suponiendo que el archivo se llama *style.css*  
y está almacenado en la carpeta *styles*

Prioridad de estilos cuando se modifica el mismo elemento es:  
1) Archivo en línea, 2) elemento Style y 3) archivo css

## CSS

**Ejemplo:** se presentan fragmentos de la hoja de estilos aplicada a la pagina web del segundo ejemplo y como se visualizaría.

```
header {
  height: 100px;
  display: flex;
  justify-content: space-around;
}

img {
  width: 200px;
}

section p {
  margin: 0px 20px;
  padding: 5px 20px;
  font-size: 20px;
  background-color: #antiquewhite;
  border-radius: 20px;
}
```

```
footer {
  margin: 20px;
  background-color: #000099;
  font-family: Arial, Helvetica, sans-serif;
  font-size: large;
  color: white;
}
```

```
nav {
  margin: 0 auto;
  background-color: #F00000;
}

nav ul li {
  list-style: none;
  display: inline-block;
  margin: 0 20px;
}

nav ul li a {
  text-decoration: none;
  color: white;
  padding: 15px 20px;
  display: block;
}

nav ul li a:hover {
  background: white;
  color: #000;
}
```





### Estructura de Datos y Algoritmos 1

[Inicio](#)
[Acercas de](#)

#### Presentación

El curso de Estructuras de Datos y Algoritmos 1 contribuye a la formación en el proceso de comprensión de los problemas relacionados con el manejo de información, que permite al estudiante proponer y desarrollar, para aplicativos que implican procesos de ordenamiento y búsqueda, soluciones que son eficientes y que pueden ser desplegadas de forma local o en la web.

La asignatura está ubicada en el ciclo de formación profesional. Su propósito formativo es apoyar la formalización de un proceso de pensamiento metódico, que facilite la resolución de problemas asociados con el manejo de la información y desarrollo web.

#### Objetivo de la asignatura

Seleccionar e implementar algoritmos computacionales correctos, eficientes y desplegados en la web, para resolver problemas asociados con el manejo de información, en procesos de ordenamientos y búsquedas.

#### Metodología

La tipología de la asignatura es la predominancia del **conocimiento procedimental (PCP)**, ya que se busca desarrollar en los estudiantes buenas prácticas algorítmicas, aplicadas en la resolución de problemas que requieran del uso de estructuras de datos y desarrollo web, iniciando con una serie de ejercicios cortos con criterio de progresión; es decir, yendo de lo simple a lo complejo, con prácticas que incluyan el análisis y la síntesis en la resolución de los problemas.

**Autor:** Orlando Arboleda Molina  
**Correo:** oarboleda@uao.edu.co

Última modificación: 18 de Julio de 2023



## Ejercicio de Implementación

- Realizar los ajustes solicitados en el [Caso1\\_DesarrolloWeb\\_FrontEnd](#) a los estilos a aplicar a la página web. Para esta actividad, se debe disponer de:
  - El entorno de desarrollo [Visual Studio Code](#) (en el cual se debe instalar la extensión [Live Server](#))
  - Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).

## JavaScript

- **JavaScript** es un lenguaje de programación de scripting (interpretado) para hacer que las **páginas web** sean **interactivas** (Vara & Granada, 2015).
  - Es una de las principales tecnologías de la web, junto con HTML y CSS
  - Es soportado por todos los navegadores modernos.
- Características:
  - Su **sintaxis** se asemeja a C++ y Java.
  - Es un lenguaje **débilmente tipado** (una variable puede tener valores de tipos diferentes)
  - Una característica **poco deseable**, es que **por defecto, todas las variables son globales**.
  - Está diseñado en un **paradigma** simple **basado en objetos**.

## JavaScript – Variables

Variables	Descripción	Ejemplo
<b>var</b> nombre;	Permite crear la variable <i>nombre</i> .	<i>var peso;</i>
<b>let</b> nombre;	No permite que se vuelva a crear la variable <i>nombre</i> en el bloque	<i>let peso;</i>
<b>const</b> nombre;	Como <i>let</i> y su valor no puede cambiarse con una reasignación	<i>const peso=60.5;</i>

- Las **cadena**s se pueden crear con:
  - Comillas dobles o simples** - usando concatenación (símbolo '+').
  - Comillas o tildes invertidas** - usando plantillas literales las cuales permiten expresiones incrustadas con el formato `${expresion}`.

**Ejemplo:** se indica como crear una nueva cadena con la información de las variables

```
let universidad = 'UAO';
let ciudad = 'CALI'
let cadena1 = 'La universidad es '+universidad+' y esta en '+ciudad;
let cadena2 = `La universidad es ${universidad} y esta en ${ciudad}`;
```

- Aplicando buenas **practica de programación**, se sugiere nombrar las variables usando **camelCase**.

## JavaScript – Operadores

Operadores	Descripción	Ejemplo
Asignación, Aritméticos, Relacionales, Lógicos	Los mismos existente en Java	"3"==3 retorna <i>true</i>
=== y !==	La igualdad y desigualdad estrictas (sin conversión de tipos)	"3"===3 retorna <i>false</i>

Operador	Descripción
. [ ] ( )	Acceso a campos, índice de matrices, llamadas a funciones y agrupamiento de expresiones
++ -- ~ ! delete new typeof void	Incremento +1, decremento -1, negativo, NOT, NOT lógico, borrado, crear objeto, mostrar tipo, indefinido
* / %	Multiplicación, división y resto de la división o módulo
+ - +	Suma, resta, concatenación de cadenas
<< >> >>>	Desplazamiento bit a bit
< <= > >= instanceof	Menor que, menor o igual que, mayor que, mayor o igual que, instanceof
== != === !==	Igualdad, desigualdad, igualdad estricta y desigualdad estricta
&	AND bit a bit
^	XOR bit a bit
	OR bit a bit
&&	AND lógico
	OR lógico
?:	Condicional
= += -= *= /= %=	Asignación, asignación con operación
<<= >>= >>>= &= ^=	
=	
,	Evaluación múltiple

## JavaScript – Condicionales y Ciclos

- **Condicionales:** if, if\_else, if\_else\_if, switch y el operador ternario.

```
if (operacion == "multiplica"){  
    res = a*b;  
}else if (operacion == "suma"){  
    res = a+b;  
}else {  
    res = a/b;  
}
```

```
switch(operacion){  
    case "multiplica":  
        res = a*b;  
        break;  
    case "suma":  
        res = a+b;  
        break;  
    default:  
        res = a/b;  
}
```

```
return a==b ? "Equal" : "Not Equal";
```

- **Ciclos:** while, for, do\_while, for\_in

```
const myArray = [];  
let limite = 5;  
while (limite >= 0){  
    myArray.push(limite);  
    limite--;  
}
```

```
const myArray = [];  
for (let i=1; i<6; i++)  
{  
    myArray.push(i);  
}
```

```
let cantidad = 0;  
for (let data in usersObj){  
    if (data.online){  
        cantidad++;  
    }  
}  
return cantidad;
```

Ejercicios: indicar que hacen los condicionales y ciclos mostrados.

## JavaScript - Arreglos

Tipo de arreglo	Descripción	Ejemplo
<b>var/let/const</b> nombre=[d <sub>0</sub> , d <sub>1</sub> , d <sub>2</sub> ,...];	Permite crear el arreglo unidimensional.	<i>let números=[1,2,3,4];</i>
<b>var/let/const</b> nombre=[Array <sub>0</sub> , Array <sub>1</sub> , ...];	Permite crear el arreglo bidimensional.	<i>let notas=[[1,2,3],[4,5,6]];</i> <i>const cuarto=notas[1][0];</i>

Método	Descripción	Ejemplo	Si data=[1,2,3,4,5]
<b>unshift(dato)</b>	Añadir <i>dato</i> al <b>inicio</b> del arreglo.	<i>data.unshift(0);</i>	<i>data=[0, 1, 2, 3, 4, 5]</i>
<b>push(dato)</b>	Añadir <i>dato</i> al <b>final</b> del arreglo.	<i>data.push(6);</i>	<i>data=[0, 1, 2, 3, 4, 5, 6]</i>
<b>shift()</b>	Recuperar el dato del <b>inicio</b> del arreglo.	<i>let valor=data.shift();</i>	<i>valor=0    data=[1, 2, 3, 4, 5, 6]</i>
<b>pop()</b>	Recuperar el dato del <b>final</b> del arreglo.	<i>let valor=data.pop();</i>	<i>valor=6    data=[1, 2, 3, 4, 5]</i>

**Ejercicio:** crear un arreglo de frutas y hacer uso de todos los métodos indicados.

## JavaScript – Arreglos (2)

Método	Descripción	Ejemplo	Si data=[1,2,3,4,5]
<b>filter</b> (condicion)	Retorna arreglo con todos los elementos que cumplen la <i>condición</i> suministrada.	<code>let pares = data.filter( x =&gt; x%2===0 );</code>	<code>pares=[2, 4]</code>
<b>forEach</b> (funcion)	Ejecuta la <i>función</i> dada por cada elemento.	<code>let res=0 data.forEach( x =&gt; res+=x )</code>	<code>res=15</code>
<b>map</b> (funcion)	Retorna arreglo con el resultante de aplicar la <i>función</i> a cada elemento.	<code>let copia10 = data.map( x=&gt;10*x );</code>	<code>copia10=[10, 20, 30, 40, 50]</code>
<b>join</b> (separador)	Retorna una cadena en la que se concatenan todos los elementos del arreglo: <ul style="list-style-type: none"> <li>• Separados por “,” cuando no se suministra el <i>separador</i>.</li> <li>• Separados por el separador indicado.</li> </ul>	<code>let res1 = elements.join(); console.log(res1);</code>  <code>let res2 = elements.join(""); console.log(res2);</code>  <code>let res3 = elements.join('--'); console.log(res2);</code>	<code>1,2,3,4,5</code>  <code>1*2*3*4*5</code>  <code>1--2--3--4--5</code>

**Ejercicio:** crear un arreglo de números y hacer uso de todos los métodos indicados.



## JavaScript - Funciones

- Existen las **funciones** y las **funciones flecha**.

```
let informacion = [ [1, 2, 3], [2, 3, 4], [3, 4, 5] ];
function filtrarArreglo(arregloB, elemento){
    let newArr=[];

    for (let i=0; i<arregloB.length; i++){
        if (arregloB[i].indexOf(elemento) == -1){
            newArr.push(arregloB[i]);
        }
    }
    return newArr;
}
console.log(filtrarArreglo(informacion,2))
```

```
let informacion = [ [1, 2, 3], [2, 3, 4], [3, 4, 5] ];
let anonima = (arregloB, elemento) => {
    let newArr=[];

    for (let i=0; i<arregloB.length; i++){
        if (arregloB[i].indexOf(elemento) == -1){
            newArr.push(arregloB[i]);
        }
    }
    return newArr;
}
console.log(anonima(informacion,2))
```

Ejercicio: ¿ indicar cuál será el valor obtenido en cada caso ?.

## JavaScript – Objetos predefinidos

- Un **objeto** es una **colección de propiedades** (asociación entre una **clave** y un **valor**). Si el valor de una **propiedad** es una **función**, esta será conocida como un **método**.
- Los **objetos predefinidos** como **no dependen del navegador**, se pueden manipular en **cualquier momento**. Se listan algunas propiedades/métodos del objeto **Math**.

Prop/Métodos	Descripción	Ejemplo
<b>max</b> ( $d_0, d_1, d_2, \dots$ )	Retorna el máximo valor de los datos suministrados.	<code>let res=Math.max(-2.0, 5.0, 3.5, -0.5);</code>
<b>min</b> ( $d_0, d_1, d_2, \dots$ )	Retorna el mínimo valor de los datos suministrados.	<code>let res=Math.min(-2.0, 5.0, 3.5, -0.5);</code>
<b>pow</b> ( <i>base</i> , <i>exponente</i> )	Retorna el valor de elevar la <i>base</i> al <i>exponente</i> suministrado.	<code>let cubo=Math.pow(base, 3.5);</code>
<b>round</b> ( <i>numero</i> )	Retorna el valor entero obtenido al redondear el <i>número</i> suministrado.	<code>let res=Math.round(12.534343);</code> Retorna 13
<b>trunc</b> ( <i>numero</i> )	Retorna el valor entero obtenido al truncar el <i>número</i> suministrado.	<code>let res=Math.trunc(12.534343);</code> Retorna 12

Ejercicio: ¿ indicar cual será el valor obtenido en cada caso ?

## JavaScript – Objetos predefinidos (2)

- Se listan algunas propiedades/métodos del Objeto **String**

Prop/Métodos	Descripción	Ejemplo	
<b>length</b>	Se corresponde a la longitud de la cadena	<code>let data='UAO'; let tamaño=data.length;</code>	<i>tamaño=3</i>
<b>localeCompare(cadena)</b>	Retorna un 0, valor negativo o valor positivo, dependiendo que la cadena se igual, menor o mayor a la <i>cadena2</i> .	<code>let data = 'Uao'; let res1=data.localeCompare('Uao'); let res2=data.localeCompare('Univalle'); let res3=data.localeCompare('Icesi'); let res4=data.localeCompare('uao');</code>	<i>res1 = 0 res2 = -1 res3 = 1 res4 = 1</i>
<b>split(patron)</b>	Retorna un arreglo con las subcadenas existente en la cadena inicial, que se encuentran separadas por dicho patrón.	<code>let data = 'Hola mundo' let res1=data.split(' '); let data = 'UAO**Cali**COL' let res2=data.split('**');</code>	<i>res1= ['Hola', 'mundo'];  res2=['UAO', 'Cali', 'COL'];</i>

**Ejercicio:** crear una cadena larga y hacer uso de todos los métodos indicados.

## JavaScript – Objetos definidos por el usuario

- La creación de **nuevos objetos** es **útil**, cuando no son suficientes las características y funcionalidades de los objetos predefinidos de JavaScript.
- Las **propiedades** de un objeto JavaScript pueden ser **números** o **cadenas**.
- Para **acceder/modificar** las propiedades de un objeto, se puede usar “.” o “[ ]”.

**Ejemplo:** se indica la creación de objeto **persona**, y como acceder/modificar su atributo **equipo**.

```
let persona = {  
  nombre : 'Neymar Jr',  
  equipo : 'PSG',  
  edad : 30,  
  peso : 68.0  
}
```

```
suEquipo = persona.equipo;  
suEquipo = persona['equipo'];
```

```
persona.equipo = 'Barcelona';  
persona['Equipo'] = 'Barcelona';
```

## JavaScript – Objetos definidos por el usuario (2)

- Para **adicionar** nuevas propiedades, solo basta con crear la nueva propiedad.  
**Ejemplo:** se indican instrucciones para adicionar propiedades al objeto **persona**.

```
persona.salario = 40.8;  
persona['estatura'] = 1.75;
```

- Para **eliminar** una propiedad, se usa la instrucción *delete objeto.propiedad*.  
**Ejemplo:** se indican instrucciones para eliminar propiedades al objeto **persona**.

```
delete persona.salario;  
delete persona['peso'];
```

- Para **verificar** si tiene una propiedad, se usa la instrucción *hasOwnProperty(prop)*.  
**Ejemplo:** condicional para desplegar información de su equipo.

```
if (persona.hasOwnProperty('equipo')) {  
    console.log('Su equipo es '+persona.equipo);  
} else {  
    console.log('No se le registró equipo');  
}
```

## JavaScript – Módulos

- Se tiene la posibilidad de **dividir programas** JavaScript en módulos separados. El uso de módulos nativos se realiza con las declaraciones **import** y **export**.
- Para **aplicar** el módulo a una página HTML se debe incluir la etiqueta **script** e indicar que es **type="module"**

**Ejemplo:** archivo **funciones.js**, como se importan sus funciones en el archivo **main.js** y como el archivo **main.js** puede ser invocado en una pagina html.

```
function obtenerIMC(pes, est){
    return pes/(est*est);
}

function calcularCicloVida(años){
    let res = 'Adulto';

    if (años < 18){
        res = 'Adolescente';
    }
    return res;
}

export {obtenerIMC, calcularCicloVida};
```

```
import {obtenerIMC, calcularCicloVida} from './funciones.js';
```

```
<script src="Scripts/main.js" type="module"></script>
</head>
```

## Ejercicio de Implementación

- Realizar los ajustes solicitados en el [Caso1\\_DesarrolloWeb\\_FrontEnd](#) para implementar el módulo del archivo ***computo.js*** e integrarlo a la página web. Para esta actividad, se debe disponer de:
  - El entorno de desarrollo [Visual Studio Code](#) (en el cual se debe instalar la extensión [Live Server](#))
  - Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).



## JavaScript - Interfaz Document

- El **DOM** (Document Object Model), es una **API** para representar e interactuar con cualquier documento HTML o XML.
  - Conecta las páginas web a scripts o lenguajes de programación.
- La interfaz **Document** representa cualquier página web cargada en el navegador y sirve como punto de entrada al contenido de la página web (árbol DOM).
- Algunas propiedades y métodos de la Interfaz Document

Prop/Métodos	Descripción	Ejemplo
<b><i>getElementById(id)</i></b>	Retorna una referencia al elemento con el <i>id</i> suministrado.	<i>let val = document.getElementById('elValor');</i>
<b><i>innerHTML</i></b>	Permite obtener o poner código HTML.	<i>val.innerHTML += '&lt;p&gt;Es un texto&lt;/p&gt;';</i>
<b><i>textContent</i></b>	Permite obtener o poner texto de un nodo y sus descendentes	<i>val.textContent = '2000';</i>

## JavaScript – Entradas y Salida páginas web

1. En la página html se debe **integrar** el módulo donde está la lógica del aplicativo

```
<script type="module" src="Scripts/main.js" ></script>
```

2. En el módulo con la lógica del aplicativo se debe:

- Crear las referencias a los botones de la pagina

```
const boton1 = document.getElementById("consultaBoton");
```

- Adicionar un manejador de eventos a las referencias necesarias

```
boton1.addEventListener('click',calculaIMC)
```

- Implementar cada una de las funciones

```
function calculaIMC(){
  const nombre = document.getElementById("elNombre").value;
  const peso = parseFloat(document.getElementById("elPeso").value);
  const estatura = parseFloat(document.getElementById("laEstatura").value);

  const IMC = obtenerIMC(peso,estatura);

  const res = nombre+' tu IMC es '+IMC;
  document.getElementById("salidaIMC").textContent = res;
  // en algunos casos, tambien se puede usar el value del elemento
  // document.getElementById("salidaIMC").value = res;
}
```

## Otras formas de integración JavaScript y HTML

- En el mismo documento HTML

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1" />
  <title>Ejemplo 1</title>
  <script type="text/javascript">
    alert("Prueba de JavaScript");
  </script>
</head>
<body>
  <h1>Ejemplo 1: código embebido</h1>
</body>
</html>
```

- JavaScript en elementos HTML

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1" />
  <title>Ejemplo 3</title>
</head>
<body>
  <p onclick="alert('Prueba de JavaScript');">
    Ejemplo 3: código en atributos
  </p>
</body>
</html>
```

## Ejercicio de Implementación

- Realizar los ajustes solicitados en el [Caso1\\_DesarrolloWeb\\_FrontEnd](#) para implementar la lógica del aplicativo en el archivo ***index.js*** y finalizar la implementación solicitada. Para esta actividad, se debe disponer de:
  - El entorno de desarrollo [Visual Studio Code](#) (en el cual se debe instalar la extensión [Live Server](#))
  - Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).

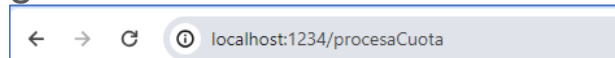
## DESARROLLO BACK-END

- **Proceso** de administrar el almacenamiento de datos y acceder a los datos de una base de datos para mostrarlos en una página web para que los usuarios puedan consumirlos en cualquier dispositivo (Platzi, 2022).
- Se debe tener en cuenta:
  - **Protocolo HTTP**
  - JSON
- Existen muchos lenguajes y plataformas, entre ellas:
  - **Node.js** para programar en **JavaScript** (usando **Express.js**)
  - Php
  - Asp



## PROTOCOLO HTTP

- El **HTTP** (Hypertext Transfer Protocol) es un protocolo que permite la **comunicación** entre los **navegadores** y **servidores web**. Este **define** unos **métodos de petición** (indican la acción que desea que se efectúe sobre el recurso identificado).
- El **método GET** **solicita** una **representación** del **recurso especificado**, generalmente una página. Estas solicitudes solo deben recuperar datos.
- El **método POST** **envía datos** en el cuerpo de la petición, para **que sean procesados** por el recurso identificado en la línea petición. Pensado para crear **nuevos contenidos**.

A screenshot of a web form titled "Simulador de Cuotas". The form has a light blue background. It contains four input fields: "Nombre" with the value "Lina Rios", "Préstamo \$" with the value "6000000", "Interés (%)" with the value "15,5", and "Meses" with the value "12". Below these fields is a large empty rectangular box. At the bottom of the form, there are two buttons: "Calcular Cuota" with a dropdown arrow and "Ejecutar funcionalidad".

## URL

- Una **URL (Uniform Resource Locator)** es una **dirección** dada a un **recurso único en la Web**. Es el mecanismo usado por los navegadores para obtener cualquier recurso publicado en la web.

- Tiene la siguiente estructura:

**protocolo://hostName:puerto/ruta**  
↓  
`http://localhost:1234/procesaCuota`

- **Protocolo** es el método para transferir datos en la red (ej. http, mailto, ftp).
- **Hostname** es el nombre del dominio del servidor web (o *localhost*).
- **Puerto** de acceso para acceder al recurso (por defecto es 80 para http).
- **Ruta** al recurso en el servidor web (es opcional).



## Paginas Html para suministrar datos al servidor

- Se debe incluir un formulario, indicando la **ruta** y el método **post**.
  - Los inputs deben contar con el atributo **name**
  - Debe tener un **botón** de **tipo submit**
  - La ruta será **“/”** si la petición es al hostname, en lugar de una ruta

Ejemplo: se presenta estructura de la pagina que hará la petición post

```

      ruta      petición
      ↓        ↓
<form action="/procesarCuota" method = "post" enctype="application/x-www-form-urlencoded">
  <table>
    <caption>Simulador de Cuotas</caption>
    <tr>
      <td>Nombre</td>
      <td>
        <input type="text" value="Lina Rios" name="elNombre">
      </td>
    </tr>
    <tr>
      <td>Prestamo $</td>
      <td>
        <input type="number" value="6000000" name="elPrestamo">
      </td>
    </tr>
  </table>
  <button id="procesar" type="submit">Ejecutar funcionalidad</button>
</form>
      ↑
    boton de submit

```

## Ejercicio de Implementación

- Realizar los ajustes solicitados en el [Caso2\\_DesarrolloWeb\\_BackEnd](#) al interior de la pagina ***index.html***, para que esta permita realizar las peticiones ***post***, a la ruta indicada.

Para esta actividad, se debe disponer de:

- El entorno de desarrollo [Visual Studio Code](#)
- Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).

## NODE.JS y EXPRESS.JS

- Node.js es un entorno de ejecución de JavaScript que:
  - Usa el motor de JavaScript que impulsa Google Chrome.
- El módulo Express.js es el framework de aplicaciones web más popular de NodeJS. Siendo minimalista y flexible, proporciona características para las aplicaciones web.

Ejemplo: se muestra la lógica de un servidor web en Node.js y Express.js

```
var express = require('express');
var app = express();

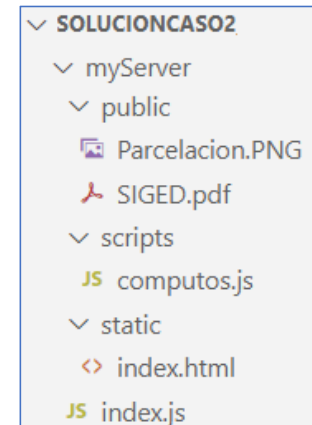
app.get('/', function(req, res) {
  res.send('Hola Mundo!');
});

app.listen(3000, function() {
  console.log('Aplicación ejemplo, escuchando el puerto 3000!');
});
```

## Creando un servidor web en Node.js

**Pasos** para crear el **Back-end** en **NodeJS** del URL <http://localhost:1234/procesaCuota>

1. Descargar e instalar Node.js.
2. Crear estructura del **aplicativo** (ej: **SolucionCaso2**) y el **servidor web** (ej: **myServer**). Al interior del servidor web se sugieren las siguientes subcarpetas:
  - **scripts** – para almacenar los módulos javascript, con las funciones para realizar los cálculos o generar las páginas dinámicas.
  - **static** – para almacenar las páginas web estática.
  - **public** – para almacenar los recursos públicos



## Creando un servidor web en Node.js (2)

3. Construir el aplicativo con **NodeJs** y **Express**. Para ello:
  - Abrir el aplicativo en Visual Studio
  - Dentro de Visual Studio abrir un terminal e ingresar a la carpeta del servidor web (usando el comando `cd`)
  - Crear la **aplicación Node.js** digitando el siguiente comando en el terminal:  
*npm init -y*
  - Instalar el **módulo Express.js** digitando el siguiente comando en el terminal:  
*npm i express*
  - Cuando el servidor web construido, deber ser lanzado digitando el siguiente comando en el terminal:  
*node index.js*

## Creando un servidor web en Node.js (3)

4. Programar la estructura general del servidor web. Para ello, cree el archivo *index.js* al interior de la carpeta del servidor web. Este deberá tener la siguiente estructura:

```
const util = require('./scripts/computos')  ← enlaza a un modulo

const express = require('express')  ← enlaza al modulo express

const app = express()  ← crea aplicativo web
const port = 1234  ← define el puerto en el que será atendido

app.use(express.urlencoded({extended: true}))  ← define indica como serán
suministrados los datos

app.use(express.static("public"));  ← define una ruta con recursos disponibles

app.get('/procesaCuota', (req,res)=> {  ← lógica de las peticiones get
  // lógica
})

app.post('/procesaCuota', (req,res)=> {  ← lógica de las peticiones post
  // lógica
})

app.listen(port, () => {  ← define que hace el servidor cuando es activado
  console.log('Estoy ejecutandome en http://localhost:'+port);
})
```

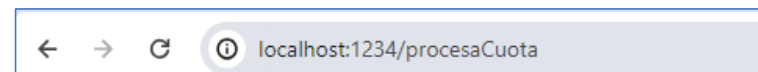
## Creando un servidor web en Node.js (4)

5. Programar la lógica para las **peticiones GET**.
  - Se debe usar el parámetro **res** para retornar las respuestas del servidor web, usando cualquiera de los siguientes métodos:

Prop/Métodos	Descripción	Ejemplo
<b>sendFile</b>	Permite retornar un archivo estático	<code>res.sendFile("index.html")</code>
<b>send</b>	Permite retornar cualquier información que no sea un archivo estático	<code>res.send("HOLA")</code>

**Ejemplo:** se presenta la lógica en el servidor para la petición get y el url que debe invocarse

```
app.get('/procesaCuota', (req,res)=> {
  console.log('en get/procesaCuota')
  res.sendFile(__dirname+"/static/index.html")
})
```



## Creando un servidor web en Node.js (5)

### 6. Programar la lógica para las **peticiones POST**.

- Se debe usar el parámetro **req** para capturar la información enviada en los inputs de un formulario.  
Los inputs deben contar con el atributo **name**.
- Se debe usar el parámetro **res** para retornar las respuestas del servidor web.

**Ejemplo:** se presenta estructura de la lógica en el servidor para la petición put y posible pagina desde la cual se hace la invocación al pulsar el botón de submit

```
app.post('/procesaCuota', (req,res)=> {
  // obtiene los datos suministrados desde el formulario
  const datos = req.body;
  const nombre = datos.elNombre;
  // resto de la lógica
  //obtiene string con la pagina HTML a retornar y la regresa
  const nPage = util.crearPagina(nombre, prestamo, meses, interes, salida);
  res.send(nPage);
})
```

Simulador de Cuotas

Nombre	<input type="text" value="Lina Rios"/>	elNombre
Prestamo \$	<input type="text" value="6000000"/>	elPrestamo
Interes (%)	<input type="text" value="15,5"/>	
Meses	<input type="text" value="12"/>	

Calcular Cuota

Ejecutar funcionalidad

boton a pulsar



## Creando un servidor web en Node.js (7)

### 8. OPCIONAL – definir la ruta con recursos públicos.

- Se debe incluir la siguiente instrucción, ajustando la carpeta donde están los recursos

```
app.use(express.static("public"));
```

carpeta  
↓

- El url para acceder debe contener: **localhost**, **puerto** y **nombre** del recurso

Ejemplo: se presenta la invocación y resultado al acceder al recurso SIGED.pdf



## Ejecución del servidor web en Node.js

Las siguientes son las instrucciones/comandos para interactuar con el **back-end** creado con **NodeJS**:

- Dentro de Visual Studio abrir un terminal e ingresar a la carpeta del servidor web (usando el comando **cd**)
- El servidor web es ejecutado, digitando el siguiente comando en el terminal:  
*node index.js*
- El servidor web es detenido, digitando el siguiente comando en el terminal:  
*Ctrl + C*

**Ejemplo:** se presentan instrucciones para ingresar y ejecutar el servidor creado

```
.\SolucionCaso2> cd .\myServer\  
.\SolucionCaso2\myServer> node .\index.js
```

## Ejercicio de Implementación

- Con relación al [Caso2\\_DesarrolloWeb\\_BackEnd](#):
  - Construir el aplicativo en Node.js
  - Ajustar en el archivo **computos.js**, la lógica de la función **crearPagina**, para que **cadSalida** sea desplegada en el área de texto y que el formulario genere una petición **post** a la ruta indicada
  - Implementar en el archivo **index.js**, la lógica del servidor web para atender las peticiones **get** y **post**, en la ruta indicada y cumpliendo los requerimientos solicitados para el caso.
  - Permitir que la carpeta **public** disponga de recursos públicos

Para esta actividad, se debe disponer de:

- El entorno de desarrollo [Visual Studio Code](#)
- El entorno de ejecución [Node.js](#)
- Un [navegador moderno](#) y actualizado (ej. Chrome, Firefox, etc).

## JSON

- El **JSON** (JavaScript Object Notation), es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript (MSDN WEB DOCS, 2021).
  - Es comúnmente utilizado para intercambiar datos entre aplicaciones.
  - Muchos entornos de programación pueden leer y generar JSON.
  - En el caso de JavaScript se pueden generar las instrucciones **JSON.stringify** y **JSON.parse** para convertir entre objetos de JavaScript y JSON

- Ejemplo:** reglas de un JSON valido y conversiones

1. Esta compuesto por **pares** *clave/valor*.
2. Las *claves* deben ser indicadas entre comillas dobles.
3. Cada *clave* debe tener una asignación.
4. Todos los **pares** se indican entre llaves.
5. La separación entre **pares** deben ser indicadas con comas.
6. Los *valores* pueden ser otros objetos.

```
let data = {persona: 'fernando',
            edad: 35,
            residencia: {ciudad: 'cali', estrato: 3}
            }
```

```
let en_JSON = JSON.stringify(data)
```



```
'{"persona": "fernando", "edad": 35, "residencia": {"ciudad": "cali", "estrato": 3}}'
```

```
let data2 = JSON.parse(en_JSON)
```



```
{persona: 'fernando', edad: 35, residencia: {...}}
```

## REFERENCIAS


MSDN WEB DOCS. (20 de abril de 2021). JavaScript. Recuperado el 22 de junio de 2022 de <https://developer.mozilla.org/es/docs/Learn/JavaScript>

MSDN REFERENCES. (11 de febrero de 2021). Objetos globales. Recuperado el 22 de junio de 2022 de [https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\\_Objects/](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/)

Platzi. (18 de marzo de 2022). Qué es FrontEnd y Backend: diferencias y características. Recuperado el 8 de julio de 2022 de <https://platzi.com/blog/que-es-frontend-y-backend/>

Vara, J. M., Granada, D. (2015). Desarrollo web en entorno cliente. RA-MA Editorial.

Wikipedia. (13 de septiembre de 2012). Desarrollo Web. Recuperado el 22 de junio de 2022 de [https://es.wikipedia.org/wiki/Desarrollo\\_web](https://es.wikipedia.org/wiki/Desarrollo_web)



Universidad Autónoma de Occidente - Cali

BUEN VIENTO Y BUENA MAR !!!