# Herencia

## Patrón de módulo

{desafío} latam_

***Reutilizar código en JavaScript aplicando el concepto de herencia, polimorfismo y closures para dar solución a una problemática.***

***Reconocer los elementos fundamentales del Document Object Model y los mecanismos para la manipulación de elementos en un documento html.***

{desafío} latam_

- Unidad 1: ES6+ y POO
- Unidad 2: Herencia
- Unidad 2: Callbacks y APIs

***Te encuentras aquí***

# ¿Qué aprenderás en esta sesión?

- *Aplica el patrón de módulo en JavaScript para hacer el código más ordenado, mantenible y reutilizable.*
- *Identifica los elementos fundamentales que componen la jerarquía del Document Object Model.*
- *Utiliza instrucciones Javascript para la manipulación de un objeto utilizando la jerarquía del DOM.*
- *Explica el rol de los eventos dentro del DOM reconociendo los más frecuentes.*
- *Utiliza instrucciones Javascript para la definición de un comportamiento simple ante un evento en el DOM determinado.*

{desafío} latam_

# ¿Tienes alguna consulta sobre la clase anterior?

{desafío} latam_

# /* Funciones Autoejecutables */

# Funciones Autoejecutables

Como hemos visto hasta el momento, JavaScript implementa las funciones para codificar y dar soluciones a muchos problemas de la vida real mediante la programación, ya sean funciones anónimas, expresiones de funciones, funciones anidadas, funciones flechas, funciones constructoras, entre otras. Pero, ninguna de ellas tiene la posibilidad hasta el momento de ejecutarse por sí sola, para lograr esto JavaScript crea y deja a disposición de los programadores las expresiones de función ejecutadas inmediatamente (Immediately-invoked Function Expressions o por sus siglas: IIFE), las cuales, son funciones que se ejecutan tan pronto como se definen. Su sintaxis en ES5 es:

```
(function () {
    ...declaraciones...
})();
```

# Funciones Autoejecutables

La sintaxis mostrada anteriormente, no es más que un patrón de diseño también conocido como función autoejecutable (Self-Executing Anonymous Function) y se caracteriza por estar conformada o compuesta en dos partes. La primera es la función anónima, mientras que la segunda parte crea la expresión de función cuya ejecución es inmediata (). Ahora bien, en ES6, se implementan las funciones flechas y se deja de utilizar la palabra reservada “function”, quedando:

```
(() => {
    ...declaraciones...
})();
```

# Funciones Autoejecutables

Veamos unos ejemplos para comprender el funcionamiento de estas funciones autoejecutables, en este caso utilizaremos la sintaxis de ES5 y dentro de la función declararemos una variable, para mostrarla dentro y fuera de la IIFE, y ver cuál sería el resultado.

```
(function () {
   var interna = "Variable dentro de la IIFE";
   console.log(interna);
})();
console.log(interna);
```

# Funciones Autoejecutables

Veamos unos ejemplos para comprender el funcionamiento de estas funciones autoejecutables, en este caso utilizaremos la sintaxis de ES5 y dentro de la función declararemos una variable, para mostrarla dentro y fuera de la IIFE, y ver cuál sería el resultado.

```
(function () {
   var interna = "Variable dentro de la IIFE";
   console.log(interna);
})();
console.log(interna);
```

Al ejecutar el código anterior en la consola del navegador web, el resultado que encontraremos será el siguiente:

```
Variable dentro de la IIFE
Uncaught ReferenceError: interna is not defined
```

# Funciones Autoejecutables

Como se puede observar en el resultado mostrado anteriormente, al intentar mostrar la variable creada dentro de la función, el resultado será el esperado, es decir, el valor que tiene asignado, pero al intentar mostrar esa variable fuera de la función, el resultado indicará que esta variable no se encuentra definida. Si recordamos del tema anterior, esto se debe a que la variable declarada dentro de una función pasará a tener un scope o ámbito local y no podrá ser utilizada fuera de este ámbito local. Veamos ahora otro ejemplo más completo:

```
var lenguaje= "Ruby";
(function(){
  var lenguaje= "JavaScript";
  console.log(lenguaje + " es un lenguaje de programación");
})();
console.log(lenguaje);
```

# Funciones Autoejecutables

En el código anterior, se puede indicar que una función IIFE crea un nuevo contexto donde podemos declarar variables a salvo de otras variables definidas en el ámbito global. Por ejemplo, si tenemos una variable global denominada “lenguaje”, podemos definir otra variable con el mismo nombre dentro de la IIFE, sin miedo a que ambas sufren una colisión. Si ejecutamos el código anterior en la consola de nuestro navegador web, el resultado será:

```
JavaScript es un lenguaje de programación
Ruby
```

En el resultado mostrado anteriormente, se puede apreciar como la IIFE se ejecuta automáticamente después de ser definida y muestra por la consola dos mensajes. En este caso, dentro de la IIFE la variable “lenguaje” contiene el texto "JavaScript" , sin afectar a la variable lenguaje en su contexto global.

# /* Patrón de Módulo */

# Patrón de Módulo

El patrón módulo consiste en un módulo donde se encapsula toda la lógica de nuestra aplicación o proyecto. Dentro de este módulo estarán declaradas todas las variables o funciones privadas y sólo serán visibles dentro del mismo. Su principal atractivo es que resulta extremadamente útil para conseguir código reusable y sobre todo, “modularizar” nuestro código, para transformarlo en piezas reutilizables de código.

En Javascript el patrón de módulo es utilizado para emular el concepto de clases, de tal manera que podemos utilizar métodos públicos o privados como variables dentro de un objeto, protegiendo así partes de nuestro código de cambios innecesarios, utilizando closures y dejando solo una parte de la información disponible para consultar. Al exponer los datos de esta forma, tendremos una solución limpia para utilizar nuestro código en diversas partes de nuestra aplicación. Por lo tanto, este patrón utiliza funciones invocadas inmediatamente (IIFE, por su sigla en inglés) donde se retorna un objeto.

# Ejercicio guiado: Contador con IIFE


{desafío} latam_

# Ejercicio guiado

## *Contador con IIFE*

Definir un módulo, el cual será una función invocada inmediatamente (IIFE) que dentro del cuerpo de la misma tenga una variable denominada “contador” y retorne dos funciones, una denominada “incrementaContador”, la cual tiene como responsabilidad incrementar el contador en 1 y otra llamada “reseteoContador”, que tiene como responsabilidad darnos el valor del contador, antes de devolverla a 0.

Para llevar esto al código, debemos implementar los siguientes pasos:

Paso 1: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella debes crear un archivo con el nombre de script.js. Seguidamente en el archivo script.js, inicializamos primeramente una constante denominada moduloPrueba, la cual, tendrá la IIFE como valor asignado, quedando el código de la siguiente manera.

```
const moduloPrueba = (() => {})();
```

# Ejercicio guiado

## *Contador con IIFE*

Paso 2: Ahora dentro de la IIFE, lo primero es declarar la variable llamada “contador” e inicializarla en 0. Luego, retornamos dos funciones, una para incrementar el contador y otra para el reinicio del contador. Quedando el código:

```
const moduloPrueba = (() => {
    let contador = 0;
    return {
        incrementaContador: () => {},
        reseteoContador: () => {}
    };
})();
```

# Ejercicio guiado

## *Contador con IIFE*

```
const moduloPrueba = (() => {
    let contador = 0;
    return {
        incrementaContador: () => {
            return contador++;
        },
        reseteoContador: () => {
          console.log(`valor del contador antes de reiniciar:
${contador}`);
            contador = 0;
        }
    };
})();
```

Paso 3: Queda por indicar los procesos dentro de las funciones incrementaContador y reseteoContador, en la primera retornamos el incremento de la variable llamada “contador” y en la segunda el mensaje: `valor del contador antes de reiniciar: `, interpolando la variable contador y posteriormente llevándola a cero:

# Ejercicio guiado

## *Contador con IIFE*

Paso 4: Al código anterior, le agregaremos primeramente el llamado a la variable que contiene la IIFE y luego la función que deseamos ejecutar, en este caso, primeramente ejecutaremos tres veces la función de incremento, luego una vez la función de decremento".

```
moduloPrueba.incrementaContador();
moduloPrueba.incrementaContador();
moduloPrueba.incrementaContador();
moduloPrueba.reseteoContador();
```

Paso 5: Al ejecutar el archivo script.js con ayuda de NodeJS, tendremos lo siguiente.

```
valor del contador antes de reiniciar: 3
```

# /* Recordando el DOM */

# Recordando el DOM

## *El concepto DOM*

El DOM es un estándar adoptado por los navegadores web, creado por el W3C (World Wide Web Consortium), con el fin de ser un medio que permita a los programas y scripts, acceder y gestionar el contenido.

El estándar DOM, contempla 3 tipos de documentos:

**Core DOM**

Modelo estándar para todos los tipos de documentos acogidos.

**XML DOM**


Modelo estándar para los tipos de documentos XML (usado ampliamente en servicios web)

**HTML DOM**


Modelo estándar para los documentos HTML (utilizado en sitios web)

# Recordando el DOM

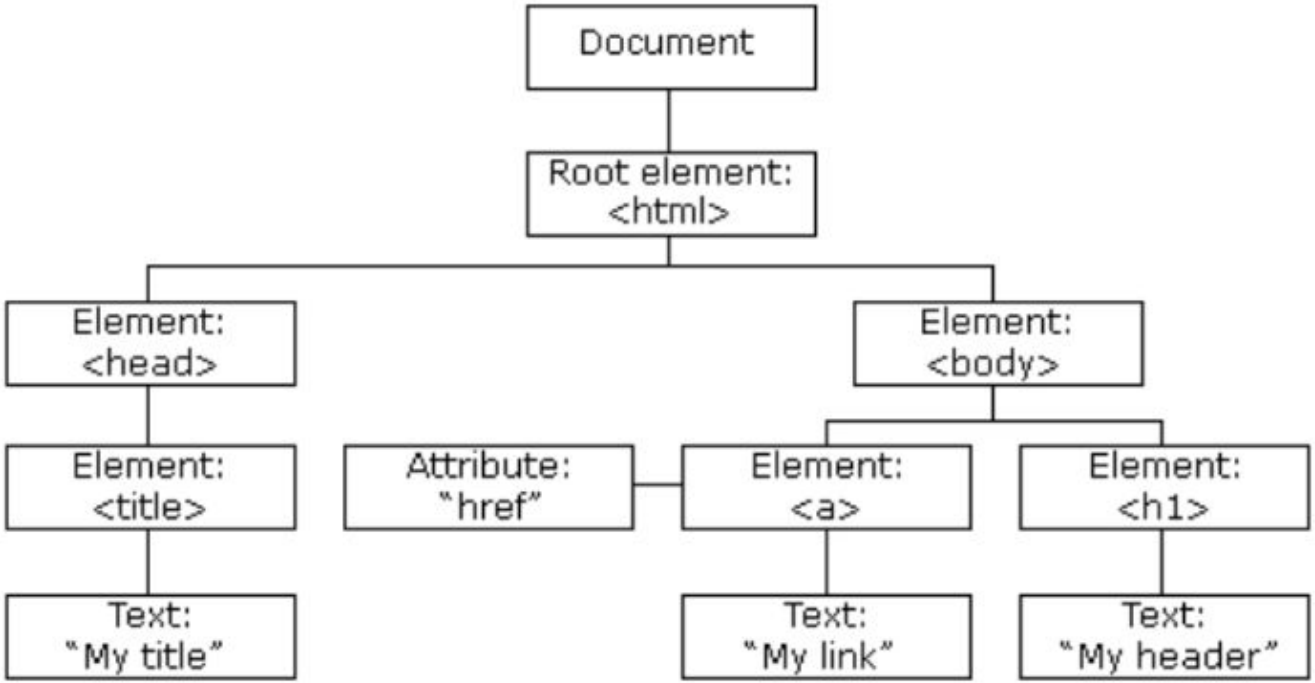
## *HTML DOM*

El estándar HTML DOM está definido principalmente por un modelo de objeto e  interfaz de programación destinado a la gestión de elementos HTML, el cual se  compone de los siguientes elementos:

- **Objetos** que representan elementos HTML.
- **Propiedades** que pueden ser valores manipulables de los elementos HTML.
- **Métodos** para gestionar elementos mediante acciones programadas.
- **Eventos** que reaccionan a alguna acción.

# Recordando el DOM

*Estructura del DOM en HTML*



# Métodos de uso común para manipular el DOM

## *Maneras de acceder a elementos del DOM*

- **Id:** El acceso mediante el atributo id es uno de los más utilizados y de fácil uso, puesto que es un identificador único de un elemento en el DOM.

```
<div id="contenedor">
    <p id="parrafo">Hola soy un párrafo.</p>
</div>
```

```
var parrafo = document.getElementById("parrafo");
```

- **Tag:** El acceso por tag se realiza especificando el tipo de elemento HTML. Por ejemplo, si deseamos buscar todos los elementos de tipo párrafo en el DOM (<p>), podríamos escribir el siguiente código:

```
<div id="contenedor">
    <p id="parrafo">Hola soy un párrafo.</p>
</div>
```

- Usando **getElementsByTagName,** seleccionamos el elemento p.

```
var parrafos = document.getElementsByTagName("p");
```

# Métodos de uso común para manipular el DOM

## *Maneras de acceder a elementos del DOM*

- **Clase:** También podemos buscar un elemento mediante su atributo class en el DOM. Por ejemplo, si deseamos buscar un elemento que contenga la clase botón, podríamos escribir el siguiente código:

```
<div id="contenedor">
  <button type="button"
class="boton">Soy un  botón</button>
</div>
```

```
var boton =
document.getElementsByClassName("boton ");
```

- **Selectores CSS:** Es posible buscar elementos mediante selectores CSS (de la misma forma que se realiza en el código CSS). Por ejemplo, si deseamos buscar todos los elementos <p> con la clase párrafo, podríamos escribir el siguiente código:

```
<div id="contenedor">
    <p class="parrafo">Hola soy un párrafo.</p>
<p class="parrafo">Hola soy otro párrafo.</p>
</div>
```

```
var parrafos =
document.querySelectorAll(".parrafo");
```

# Eventos

## *Listener*

El estándar HTML DOM incluye eventos que nos permiten reaccionar mediante JavaScript, a eventos HTML ya definidos. Esto funciona como una suerte de observador, el cual está pendiente de la ejecución de eventos HTML.

Estos observadores son conocidos como listeners. Para agregar un listener a un elemento del DOM, se debe hacer uso del método addEventListener, que puede recibir como parámetros:

**EVENTO**

**FUNCIÓN**

**PROPAGACIÓN DEL EVENTO**

```
element.addEventListener(evento, funcion, propagacion);
```

{desafío} latam_

# Ejercicio guiado - Login

{desafío}
latam_

# Ejercicio guiado

***Login***

En el sitio web se encuentra un formulario que solicita a los usuarios ingresar el correo electrónico y contraseña.

Con ayuda de listener, vamos a rescatar los valores que escriba el usuario dentro del formulario. Además, se contará con un botón donde el usuario podrá hacer clic para procesar los datos del login y mostrar el texto escrito en el input dentro de una etiqueta <p> con el id denominado resultado. Indicar en el mensaje, por ejemplo: “Bienvenido usuario@usuario.com”.

# Ejercicio guiado

***Login***

Paso 1: Acceder al elemento formulario primeramente y luego le agregamos un listener, en este caso, usaremos el evento "submit". Los submit son eventos propios de los formularios y se utilizan para enviar la información que es introducida en estos, igualmente que en el ejemplo anterior, activaremos una función externa que llamaremos “login”.

```
let form = document.getElementById( "formulario"
); form.addEventListener( "submit", login);
```

# Ejercicio guiado

## *Login*

Paso 2: Ahora en la función login, debemos leer los datos ingresados por el usuario y dar la bienvenida como usuario registrado:

```
function login(){
   var email = document.querySelector("#email").value;
   var password = document.querySelector("#password").value;
document.querySelector("#resultado").innerHTML = `Bienvenido ${email}`;
};
```

# Ejercicio guiado

***Login***

Paso 3: Unir todos los trozos de código en uno solo:

```
function login(){
   var email = document.querySelector(".email").value;
   var password = document.querySelector(".password").value;
   document.querySelector(".resultado").innerHTML =
`Bienvenido ${email}`;
};

let form = document.getElementById( "formulario" );
form.addEventListener( "submit", login);
```

# Ejercicio guiado

***Login***

Paso 4: Al ejecutar el código anterior, agregando un correo, una contraseña y haciendo un clic en el botón de enviar, el cambio ocurre muy rápidamente, debido a que la página se recarga nuevamente ella sola, limpiando los campos con los valores ingresados y desapareciendo el mensaje mostrado en la parte inferior del documento. Esto se debe al comportamiento por defecto de los formularios con el evento submit, que recargan la página donde se encuentran automáticamente.

# Ejercicio guiado

## *Login*

Para evitar el comportamiento por defecto del evento submit y evitar que la página se recargue automáticamente, utilizaremos el método **preventDefault()**.

```
let form = document.getElementById("formulario");
form.addEventListener("submit", function (event) {
  event.preventDefault(); // Evitar que el formulario
se envíe (recargue la página)
  login();
});
```


Bienvenido email@desafiolatam.com

{desafío} latam_

# Comenta con tus palabras, ¿Qué entendiste sobre estos conceptos?

# Resumiendo

- Funciones autoejecutables (Immediately-invoked Function Expressions o por sus siglas IIFE), son funciones que se ejecutan tan pronto como se definen.
- El patrón módulo consiste en un módulo donde se encapsula toda la lógica de nuestra aplicación o proyecto. Dentro de este módulo estarán declaradas todas las variables o funciones privadas y sólo serán visibles dentro del mismo. Su principal atractivo es que resulta extremadamente útil para conseguir código reusable

# Próxima sesión...

- *Desafío evaluado - Sugerencia de videos*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales