Laboratorio de Computación II

Unidad 4: JavaScript (JS)

Tecnicatura Universitaria en Programación - UTN

JavaScript - Introducción

Javascript fue creado para “**dar vida a las páginas web**”.

Los programas en este lenguaje se llaman **scripts**. Se pueden escribir directamente en el HTML de una página web y ejecutarse automáticamente a medida que se carga la página.

Los scripts se proporcionan y ejecutan como texto plano. **No necesitan preparación especial o compilación para correr**.

JS es un lenguaje orientado a objetos y también tiene algunas similitudes en sintaxis con el lenguaje de programación Java. Pero JavaScript no está relacionado con Java de ninguna manera.

JS corre tanto del lado del cliente (navegador) como también del lado del servidor (backend).

JavaScript - ¿Qué podemos hacer?

Cuando trabajamos con JavaScript del lado del cliente creando Páginas o Aplicaciones Web, entre muchas otras cosas, podemos hacer lo siguiente:

- Modificar el contenido de una página web agregando o quitando elementos.
- Cambiar el estilo y la posición de los elementos en una página web.
- Monitorear eventos como hacer clic con el mouse, pasar el mouse, etc. y reaccionar ante ellos.
- Realizar y controlar transiciones y animaciones.
- Crear ventanas emergentes de alerta para mostrar información o mensajes de advertencia al usuario.
- Realizar operaciones basadas en las entradas del usuario y mostrar los resultados.
- Validar las entradas del usuario antes de enviarlas al servidor.
- Enviar y recibir información del servidor.
- Etc.

JavaScript - Cómo agregarlo a nuestra página

Existen 3 maneras de agregar JavaScript a nuestra página:

Embebido:
Embebiendo el código JavaScript dentro de las etiquetas *<script></script>*.

```
<script>
    var hello = "Hola desde JavaScript!";
    alert(hello);
</script>
```

Externo:
Creando un archivo .js y referenciandolo desde nuestro documento HMTL de la siguiente manera:

```
<head>
    <script src="js/index.js" defer></script>
</head>
```

En línea:
Poniendo nuestro código JS directamente en propiedades especiales de los elementos HTML, como por ejemplo onclick, onkeypres, onmouseover, etc.

```
<button onclick="alert('Hola!!')">
    Mostrar alerta
</button>
```

JavaScript - Sintaxis

La sintaxis de JavaScript es el conjunto de reglas que definen un programa JavaScript estructurado correctamente.

El siguiente ejemplo muestra cómo se ven las declaraciones de JavaScript:

```
var x = 5;
var y = 10;
var sum = x + y;
console.log(sum); // imprime en consola el valor de la variable
```

Case Sensitivity

JavaScript distingue entre mayúsculas y minúsculas. Esto significa que las variables, las palabras clave del idioma, los nombres de las funciones y otros identificadores siempre deben escribirse con letras mayúsculas coherentes.

Por ejemplo, la variable **myVar** debe escribirse **myVar** no **MyVar** o **myvar**.

Comentarios

Contamos con dos maneras de comentar nuestro código:

En una única línea utilizando: *// Comentario*

En varias líneas utilizando: */* Comentario */*

JavaScript - Variables

Las **variables** son fundamentales para todos los lenguajes de programación. Las variables se utilizan para almacenar datos, como cadenas de texto, números, etc. Los datos o valores almacenados en las variables se pueden configurar, actualizar y recuperar cuando sea necesario.

Puede crear una variable con la palabra clave var, mientras que el operador de asignación (=) se usa para asignar valor a una variable, así:

```
var varName = value;
```

A continuación, podemos ver un ejemplo donde se declaran tres variables diferentes:

```
var name = "Peter Parker";
var age = 21;
var isMarried = false;
```

Consejo: asigne siempre nombres significativos a sus variables. Además, para nombrar las variables que contienen varias palabras, se usa comúnmente camelCase. En esta convención, todas las palabras después de la primera deben tener las primeras letras mayúsculas, p. Ej. myLongVariableName.

JavaScript - Variables

Declaración de variables con “let” y “const”

ES6 introduce dos nuevas palabras clave **let** y **const** para declarar variables.

La palabra clave **const** funciona exactamente igual que **let**, excepto que las variables declaradas usando la palabra clave **const** no se pueden reasignar más adelante en el código.

```
let myLetVar = "Este valor puede cambiar";
const myConstVar = "Este valor NO puede cambiar";
```

Convención de nombres para Variables JS

Estas son las siguientes reglas para nombrar una variable de JavaScript:

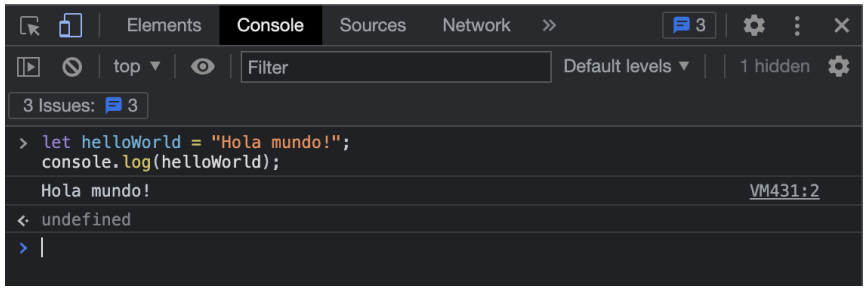
- El nombre de una variable debe comenzar con una letra, un guión bajo (_) o un signo de dólar ($).
- Un nombre de variable no puede comenzar con un número.
- Un nombre de variable solo puede contener caracteres alfanuméricos (A-z, 0-9) y guiones bajos.
- Un nombre de variable no puede contener espacios.
- Un nombre de variable no puede ser una palabra clave o una palabra reservada de JavaScript.

JavaScript - Outputs

En JavaScript, hay varias formas diferentes de generar resultados, incluida la escritura de resultados en la ventana del navegador o la consola del navegador, mostrar los resultados en cuadros de diálogo, escribir los resultados en un elemento HTML, etc.

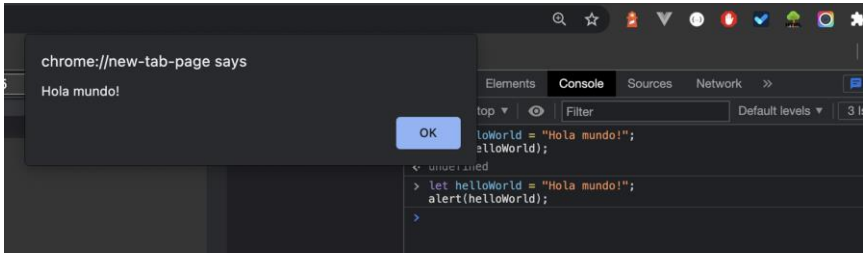
Consola del Browser

Podemos imprimir valores en la consola utilizando *console.log()*



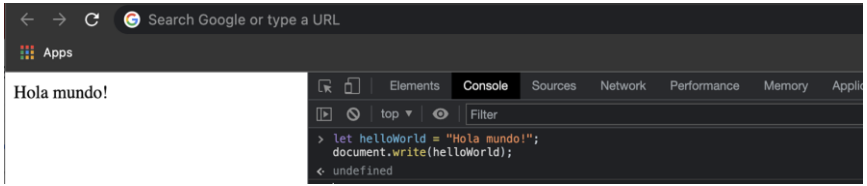
Cuadros de diálogos de alerta

También podemos mostrar resultados o mensajes utilizando *alert(value)*



Escribir la ventana del Navegador

JS nos permite escribir directamente la ventana del navegador utilizando *window.write()*



JavaScript - Outputs

Escribir dentro de un elemento HTML

Para escribir dentro de un elemento HTML, debemos en primer lugar obtener en JS el elemento en cuestión y luego si, escribir lo que nosotros deseamos utilizando la propiedad

```html
<p id="my-paragraph"></p>
```

```
let helloWorld = "Hola mundo desde JS!";
document.getElementById('my-paragraph').innerHTML = helloWorld;
```


```html
<p id="greet"></p>
<p id="result"></p>
```

```
// Escribimos texto dentro de un elemento
document.getElementById("greet").innerHTML =
"Hola Mundo!";
/* Escribimos relsutado almacenado en variable dentro
de un elemento */
var x = 10;
var y = 20;
var sum = x + y;
document.getElementById("result").innerHTML = sum;
```

JavaScript - Tipos de datos

Los tipos de datos (Data type) básicamente especifican qué tipo de datos se pueden almacenar y manipular dentro de un programa.

Hay seis tipos de datos básicos en JavaScript que se pueden dividir en tres categorías principales:

- **Primitivos (o primarios)**: String, Number y Boolean
- **Compuestos (o de referencia)**: Objeto, matriz y función (son todos de tipo Objeto).
- **Especiales**: Undefined y Null.

Los tipos de datos primitivos pueden contener solo un valor a la vez, mientras que los tipos de datos compuestos pueden contener colecciones de valores y entidades más complejas. Analicemos cada uno de ellos en detalle.

Data type: String (cadena de texto)

Utilizado para representar datos textuales (secuencias de caracteres). Se crean utilizando comillas simples o dobles, como se muestra a continuación:

```
var a = 'Esto es un String!';  // comillas simples
var b = "Esto es otro String!";  // comillas dobles
```

Data type: Number

Utilizado para representar números positivos o negativos, con o sin parte decimal o número utilizando notación exponencial, por ejemplo: 1.5e-4 (lo que equivale a 1.5x10-4):

```
var a = -25;      // entero (negativo)
var b = 80.5;     // número flotante, decimal
var c = 4.25e+6; // notación exponencial
```

El tipo de dato Number, también incluye valores especiales como: **Infinite**, **-Infinite** y **NaN**.

JavaScript - Tipos de datos

Data type: Boolean

Utilizado para representar dos valores diferentes, true o false (secuencias de caracteres). Comunmente utilizado para almacenar valores **si (true)** o **no (false)**:

```
var isReading = true;   // Si, estoy leyendo
var isSleeping = false; // No, no estoy durmiendo
```

Data type: Undefined

El tipo de datos indefinido (undefined) sólo puede tener un valor: el valor especial **undefined**. Si una variable ha sido declarada, pero no se le ha asignado un valor, tiene el valor indefinido:

```
var a;
var b = "Hola mundo!"

console.log(a) // Resultado: undefined
console.log(b) // Resultado: Hola mundo!
```

Data type: Null (nulo)

Este es otro tipo de datos especial que solo puede tener un valor: **null**. Un valor nulo significa que no hay ningún valor. No es equivalente a una cadena vacía ("") o 0, simplemente no es nada.

Una variable puede vaciarse explícitamente de su contenido actual asignándole el valor null.

```
var a = null;
console.log(a); // Resultado: null
a = "Hello World!"
console.log(a); // Resultado: Hello World!
a = null;
console.log(a); // Resultado: null
```

JavaScript - Tipos de datos

Data type: Object (objetos)

El objeto es un tipo de datos complejo que le permite almacenar colecciones de datos.

Un objeto contiene propiedades, definidas como un par clave-valor. Una clave de propiedad (nombre) es siempre una cadena, pero el valor puede ser cualquier tipo de datos, como cadenas, números, valores booleanos o tipos de datos complejos como matrices, funciones y otros objetos.

```
var person = {
    name: "Clark",
    surname: "Kent",
    age: 36
};
console.log(person.name); // Resultado: Clark
console.log(person.age); // Resultado: 36
```

Data type: Array (arreglo o matriz)

Un array es un tipo de objeto que se utiliza para almacenar varios valores en una sola variable. Cada valor (también llamado elemento) en un array tiene una posición numérica, conocida como su índice, y puede contener datos de cualquier tipo de datos: números, cadenas, valores booleanos, funciones, objetos e incluso otras matrices. El índice de la matriz comienza desde 0, por lo que el primer elemento de la matriz es [0].

La forma más sencilla de crear una matriz es especificando los elementos de la matriz como una lista separada por comas entre corchetes, como se muestra en el siguiente ejemplo:

```
var colors = ["Rojo", "Amarillo", "Blanco", "Azul"];
var cities = ["Rosario", "Santa Fe", "Venado Tuerto"];
console.log(colors[0]);   // Resultado: Rojo
console.log(cities[2]);   // Output: Venado Tuerto
```

Más información sobre objetos: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/Basics

JavaScript - Tipos de datos

Data type: Function (función)

Las funciones son uno de los bloques de construcción fundamentales en JS. Una función de JS es similar a un procedimiento — un conjunto de instrucciones que realiza una tarea o calcula un valor. En otras palabras, son un bloque de código que puede ser ejecutado desde cualquier lugar de nuestro código.

```
var greeting = function() {
    return "Hola Mundo!";
}
console.log(typeof greeting) // Resultado: function
console.log(greeting()); // Resultado: Hola Mundo!
```

Parámetros: cuando definimos una función podemos indicar la misma reciba información. Los parámetros se colocan entre paréntesis y podemos incluir 0 o N parámetros.

Retornar valores: las funciones pueden o no retornar valores, en caso de necesitarlos debemos colocar "*return* " seguido por el o los valores a devolver.

Definiendo una función:

```
function sayHello(name, lastname) {
    console.log('Hola ' + name + ' ' + lastname + '!!');
}
```

Llamando a una función:

```
sayHello('Richard', 'Stallman');
```

Ejemplo: suma de dos números

```
function getSum(num1, num2) {
    var total = num1 + num2;
    return total;
}

alert(getSum(6, 20)); // Resultado: 26
alert(getSum(-5, 17)); // Resultado: 12
```

Más información sobre funciones: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions

JavaScript - Tipos de datos

JS es de tipo dinámico

Esto significa que al declarar las variables no necesitamos indicar qué tipo de datos almacenaremos en la misma y al mismo tiempo, podemos reemplazar el valor inicial con cualquier otro valor que necesitemos. Como se muestra en el siguiente ejemplo, podemos declarar una variable con el valor nulo, luego asignarle un número entero y luego un String sin ningún tipo de inconvenientes.

```
var myVar = null;

myVar = 5;

myVar = "Cadena de caracteres";
```

La función “typeof”

Esta función incluida por defecto en JS, nos permite obtener el tipo de datos del valor que tiene asignado una función. Veamos algunos ejemplos:

```
var myFunction = function() { };
typeof myFunction // Resultado: 'function'
var myString = "Cadena de caracteres";
typeof myString // Resultado: 'string'
var myNumber = 1;
typeof myNumber // Resultado: 'number'
var myDate = new Date();
typeof myDate // Resultado: 'object'

typeof noExists // Resultado: 'undefined'
```

JavaScript - Scope (ámbito)

Al trabajar con funciones y variables, debemos tener en consideración donde declaramos nuestra variable, ya que si lo hacemos dentro de una función sólo podremos acceder a esa variable en la porción de código de la propia función.
Cuando declaramos una variable fuera de una función, la misma se denomina **GLOBAL**, pues, es “visible” en cualquier parte del código del documento actual.
Por otro lado, cuando declaramos una variable dentro de una función, la misma se denomina **LOCAL**, y este término hace referencia a que sólo se puede acceder a la variable dentro del bloque de código de la función.
Ejemplos:

```
// Definición de la función
function greetWorld() {
    var greet = "Hola Mundo!";
    alert(greet);
}

greetWorld(); // Resultado: Hola Mundo!
alert(greet); // Uncaught ReferenceError: greet is not defined
```

```
function greetWorld() {
    alert(greet);
}
function greetWorld2() {
    var greet = "Hola Mundo 2!!"
    alert(greet)
}

greetWorld();  // Resultado: Hola Mundo!
alert(greet); // Resultado: Hola Mundo!
greetWorld2(); // Resultado: Hola Mundo 2!!
```

Preguntas?

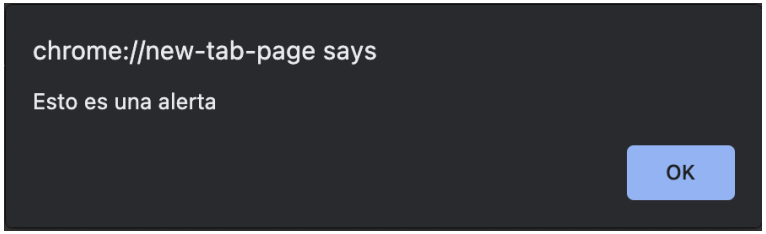
JavaScript - Cuadros de Diálogos

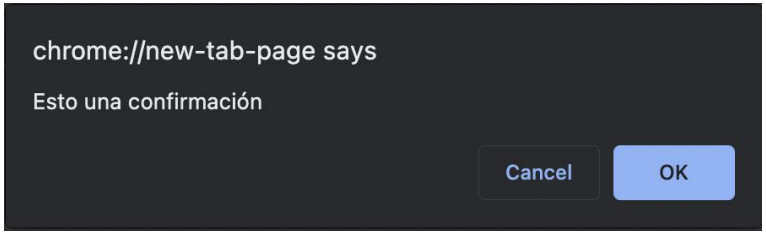
En JS disponemos de cuadros de diálogo para interactuar con el usuario. Puede usarlos para notificar a un usuario o para recibir algún tipo de entrada del usuario antes de continuar.

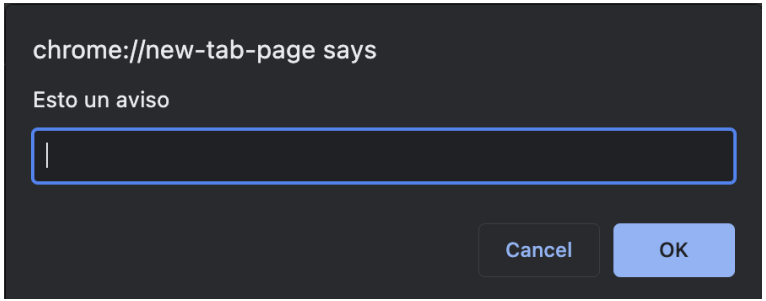
Puede crear tres tipos diferentes de cuadros de diálogo: **alerta (alert)**, **confirmación (confirm)** y **aviso (prompt)**.

La apariencia de estos cuadros de diálogo está determinada por el sistema operativo y/o la configuración del navegador, no se pueden modificar con CSS.

Además, los cuadros de diálogo son **ventanas modales**; cuando se muestra un cuadro de diálogo, la ejecución del código se detiene y se reanuda solo después de que se haya descartado.







JavaScript - Cuadros de Diálogos

alert(*message*)

alert nos permite mostrar un mensaje de texto corto.

```
var message = "Hola! Click OK para continuar.";
alert(message);

/* La siguiente línea no se ejecuta hasta que
    no sea descartado el alert anterior */
alert("Esta es otra alerta");
```

confirm(*message*)

El **confirm** nos permite mostrar un mensaje y capturar la respuesta del usuario (true o false).

```
var result = confirm("¿Desea continuar?");
if(result) {
    console.log("Se clickeo el botón OK");
} else {
    console.log("Se clickeo el botón cancelar");
}
```

prompt(*message*)

El aviso (prompt) nos permite obtener la respuesta de un usuario a través de un input de texto.

```
var name = prompt("Cómo te llamas?");
if(name.length > 0 && name != "null") {
    alert("Hola, " + name);
} else {
    alert("Anónimo!");
}
```

JavaScript - Operadores

JavaScript tiene los siguientes tipos de operadores.

Operadores Aritméticos

Operador	Descripción	Ejemplo
+	Adición	`x + y`
-	Sustracción	`x - y`
*	Multiplicación	`x * y`
/	División	`x / y`
%	Módulo	`x % y`

Operadores de String

Operador	Descripción	Ejemplo
+	Concatenación	`str1 + str2`
+=	Concat. y asignar	`str1 += str2`

Operadores de asignación

Operador	Descripción	Ejemplo	Es lo mismo que
=	Asignación	`x = y`	`x = y`
+=	Suma y asignar	`x += y`	`x = x + y`
-=	Resta y asignar	`x -= y`	`x = x - y`
*=	Multip. y asignar	`x *= y`	`x = x * y`
/=	División y asignar cociente	`x /= y`	`x = x / y`
%=	División y asignar módulo	`x %= y`	`x = x % y`

Operadores de incrementación y decrementación

Operador	Nombre	Efecto
++x	Pre-incrementar	Incrementa x en 1 y devuelve x
x++	Post-incrementar	Devuelve x, luego incrementa x en 1
--x	Pre-decrementar	Decrementa x en 1 y devuelve x
x--	Post-decrementar	Devuelve x, luego decrementa x en 1

JavaScript - Operadores

Operadores Lógicos

Operador	Nombre	Ejemplo	Resultado
&&	And	x && y	True si x e y son true
||	Or	x || y	True si x o y son true
!	Not	!x	True si x no es true (negado)

```
var yearsOld = 22;

// Valido si es mayor de edad
if (yearsOld >= 18) {
    alert("Eres mayor de edad");
} else {
    alert("No eres mayor de edad");
}
```

Operadores de Comparación

Operador	Nombre	Ejemplo	Resultado
==	Igual	x == y	True si x es igual a y
===	Identico	x === y	True si x es igual a y, y ambos son del mismo tipo
!=	Distinto	x != y	True si x no es igual a y
!==	No identicos	x !== y	True si x no es igual a y, o no son del mismo tipo
<	Menor que	x < y	True si x es menor que y
>	Mayor que	x > y	True si x es mayor que y
>=	Mayor o igual que	x >= y	True si x es mayor o igual que y
<=	Menor o igual que	x <= y	True si x es menor o igual que y

```
var x = 25;
var y = 35;
var z = "25";

alert(x == z);  // Resultado: true
alert(x === z); // Resultado: false
alert(x != y);  // Resultado: true
alert(x !== z); // Resultado: true
alert(x < y);   // Resultado: true
alert(x > y);   // Resultado: false
alert(x <= y);  // Resultado: true
alert(x >= y);  // Resultado: false
```

JavaScript - Sentencias condicionales

Una sentencia condicional es un conjunto de comandos que se ejecutan si una condición es verdadera. JS soporta dos tipos de sentencias condicionales, **if...else** y **switch..case**

If...else

Se utiliza **if** para comprobar si una condición es verdadera mientras que **else** se ejecuta se la condición es falsa. Podemos utilizar **else if** si deseamos agregar otras validaciones. Podemos incluso utilizar sólo el **if**.

```
var now = new Date();
var dayOfWeek = now.getDay(); // Domingo - Sábado : 0 - 6
if (dayOfWeek == 5) {
    alert("Buen finde!");
} else if(dayOfWeek == 0) {
    alert("Que tengas lindo Domingo!");
} else {
    alert("Buen día!");
}
```

Switch...case

La sentencia **switch...case** es una alternativa a la sentencia **if...else if...else**. La sentencia **switch...case** prueba una variable o expresión contra una serie de valores hasta que encuentra una coincidencia y luego ejecuta el bloque de código correspondiente a esa coincidencia. Veamos el mismo ejemplo pero con **switch**:

```
var now = new Date();
var dayOfWeek = now.getDay(); // Domingo - Sábado : 0 - 6
switch (dayOfWeek) {
    case 5:
        alert("Buen finde!");
        break;
    case 0:
        alert("Que tengas lindo Domingo!");
        break;
    default:
        alert("Buen día!");
        break;
}
```

JavaScript - Sentencias condicionales y Valores Falsos

Operador Ternario

Al operador ternario lo podemos definir como una forma abreviada de utilización del **if...else**. Como su nombre lo indica, se compone de tres operandos: **condicion ? valor1 : valor2**.

```
var yearsOld = 22;
var message = yearsOld >= 18 ? "Eres mayor de edad" : "No eres mayor de edad";
alert(message);
```

Valores Falsos

Para JavaScript, los siguientes valores son evaluados como **falsos (false)**:

- false
- undefined
- null
- 0
- NaN
- La cadena vacía (“”)

El resto de los valores, incluidos todos los objetos, son evaluados como **verdaderos (true)** cuando son pasados a una sentencia condicional

JavaScript - Bucles e iteraciones

Los bucles e iteraciones son una forma rápida y sencilla de hacer lo mismo repetidas veces.
Principalmente se utilizan para recorrer los elementos (items) de un array y también lo podemos utilizar para recorrer las propiedades de un objeto.

FOR

Puede usar **for** para acceder a cada elemento de una matriz en orden secuencial. Las estructuras **for** se componen de la siguiente manera:

```
for (expresionInicial; condicion;  incremento) {
    sentencia;
}
```

- ***expresionInicial***: si existe, se ejecuta. Aquí declaramos el contador del bucle.
- ***condicion***: si el valor es true, se ejecuta la sentencia del bucle, si es false finaliza.
- **incremento:** aumenta el valor de la exp. inicial y vuelve al paso 2
- **sentencia**: bloque de código a ejecutar cuando las condiciones se cumplen

```
var fruits = ["Manzana", "Banana", "Pera", "Naranja",
"Frutilla"];
// Iteramos sobre el arreglo
for(var i = 0; i < fruits.length; i++) {
    console.log(fruits[i] + ",");
}
// Resultado: Manzana, Banana, Pera, Naranja, Frutilla,
```

Existen otras variantes del **for,** como por ejemplo **for-of** y **for-in**. Vea en detalle esas alternativas y cómo manipular arreglos en la siguiente **página**.

JavaScript - Bucles e iteraciones

WHILE

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia. Las estructuras **while** se componen de la siguiente manera:

```
while (condicion) {
    sentencia;
}
```

- ***condicion***: si el valor es true, se ejecuta la sentencia del while, si es false finaliza.
- **sentencia**: bloque de código a ejecutar cuando las condiciones se cumplen

```
let i = 0;
while (i < 10) {
    console.log("El número es " + i);
    i++;
}
```

El **do...while** es una alternativa al **while.** Vea en detalle esa alternativa en la siguiente **página**.

break;

La palabra reservada “break” nos permite terminar la ejecución de un loop de manera forzada.

```
let i = 0;
while (i < 10) {
    console.log(i);
    if(i === 5) { break; }
    i++;
}
console.log("Fuera del while");
// Resultado: 1 2 3 4 5 Fuera del while
```

Evitar los **bucles infinitos**, asegurarse de que la condición en algún momento llegue a ser falsa (false)

Preguntas?

JavaScript - DOM

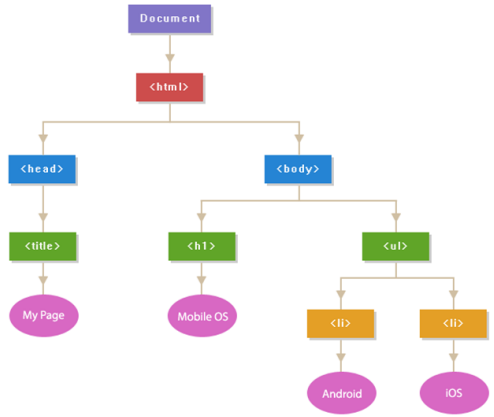
El **DOM** (***Document Object Model - Modelo de objetos de documento***) define la estructura lógica de los documentos y la forma en que un programa de aplicación puede acceder a ellos y manipularlos.
En el DOM, todas las partes del documento, como elementos, atributos, textos, imágenes, etc., están organizadas en una estructura jerárquica en forma de árbol; similar a un árbol genealógico en la vida real que consta de padres e hijos. En la terminología DOM, estas partes individuales del documento se conocen como nodos.
Básicamente, el DOM es el objeto de JavaScript “document” que nos permite manipular los elementos HTML desde el código JS.

Para entender más claramente, consideremos el siguiente ejemplo:

index.html

```html
<!DOCTYPE html>
<html>
    <head>
        <title>My Page</title>
    </head>
    <body>
        <h1>Mobile OS</h1>
        <ul>
            <li>Android</li>
            <li>iOS</li>
        </ul>
    </body>
</html>
```

Representación



index.js

```js
document.title; // Resultado: My Page
document.getElementsByTagName('li')[0].innerText; // Resultado: Android
// Cambio "iOS" por "Firefox OS"
document.getElementsByTagName('li')[1].innerText = "Firefox OS";
```

JavaScript - DOM - Selectores

JavaScript es comúnmente utilizado para modificar los documentos HTML por lo que podemos tomar un elemento de nuestro documento HTML desde JS y cambiarle el contenido, tomar el valor que tiene cualquier input, cambiar los estilos CSS, animarlo o hacerlo aparecer/desaparecer.

Seleccionar un elemento por su ID

Puedes seleccionar un elemento a través de su identificador único (**id**) utilizando el método **getElementById()**.

```
<p id="first-text">Primer párrafo</p>
<p id="second-text">Segundo párrago</p>
<script>
    let firstTextElement = document.getElementById('first-p');
    firstTextElement.innerText = "Texto actualizado desde JS";
</script>
```

Seleccionar elementos por su clase (CLASS)

Puedes seleccionar un elemento a través de su identificador único (**id**) utilizando el método **getElementsByClassName()**.

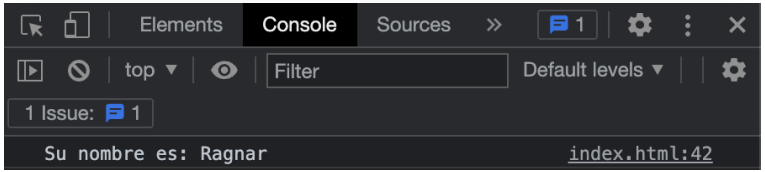
```
<p class="common-text">Primer párrafo</p>
<p class="common-text">Segundo párrafo</p>
<p class="common-text">Tercer párrafo</p>
<script>
    let texts = document.getElementsByClassName('common-text');
    // Aplicamos estilo BOLD al primer elemento
    texts[0].style.fontWeight = "bold";
    // Aplicamos color ROJO al último elemento
    texts[texts.length - 1].style.color = "red";
    // Resaltamos con fondo amarillo todos lo textos
    // y agregamos espacio entre cada uno
    for(let text of texts) {
        text.style.marginTop = "10px";
        text.style.backgroundColor = "yellow";
    }
</script>
```

JavaScript - DOM - Selectores

Seleccionar elementos por su TAG

Puedes seleccionar un elemento a través de su identificador único (**id**) utilizando el método **getElementsByTagName()**.

```
<label for="my-input">Nombre:</label>
<input type="text" name="my-input">
<button onclick="showInputValue()">Mostrar nombre</button>
<script>
    let inputs = document.getElementsByTagName('input');
    function showInputValue() {
        console.log("Su nombre es: " + inputs[0].value);
    }
</script>
```



Seleccionar elementos con Selectores CSS

Puedes seleccionar un elemento a través del selector CSS, así como lo hacemos para aplicar estilos en nuestros archivo .css con el método **querySelectorAll()**

```
<ul>
    <li class="item">Café</li>
    <li class="item">Te</li>
    <li class="item">Mate Cocido</li>
    <li class="item">Mate</li>
</ul>
<script>
let liItems = document.querySelectorAll("ul li.item");
for (let i = 0; i < liItems.length; i++) {
    i % 2 === 0 ?
        liItems[i].style.color = "blue" :
        liItems[i].style.color = "red";
}
</script>
```

JavaScript - DOM - Estilos

Seteando estilos CSS en línea desde JS

Una vez que obtenemos el elemento, es posible modificar o agregar las propiedades CSS accediendo a la propiedad **style** del propio elemento.

```
<p id="intro">Estilos desde el DOM.</p>
<p>Como colocar estilos desde JS</p>
<script>
// Seleccionamos el elemento
var elem = document.getElementById("intro");
// Aplicamos los estilos
elem.style.color = "blue";
elem.style.fontSize = "18px";
elem.style.fontWeight = "bold";
</script>
```

Convención de nombres: en CSS las propiedades se escriben en **kebab-case** (ejemplos: font-size, background-color) mientras que en JS las debemos escribir en **camelCase** (ejemplos: fontSize, backgroundColor)

Obteniendo los estilos en línea de un elemento

Tal como se puede asignar nuevas propiedades, es posible obtener el valor que tiene seteado. Es importante tener en cuenta que sólo obtendremos el valor si el mismo fue seteado en línea.

```
<style>
#intro {
    color: red;
    font-weight: 700;
}
</style>
<p id="intro" style="color: blue">Estilos desde el DOM.</p>
<script>
// Seleccionamos el elemento
var elem = document.getElementById("intro");
console.log(elem.style.color); // Resultado: "blue"
console.log(elem.style.fontWeight); // Resultado: ""
</script>
```

JavaScript - DOM - Estilos

Trabajando con las clases de un elemento

A través de JS y el DOM, podremos agregar, quitar y verificar las clases de un elemento a través de la propiedad **classList**.

Dentro de la propiedad **classList** del elemento, contamos con los siguientes métodos para manipular las clases del mismo:

- **add()**: nos permite agregar una o múltiples clases al mismo tiempo
- **remove()**: nos permite eliminar una o múltiples clases al mismo tiempo.
- **toggle()**: elimina la clase siempre y cuando exista, sino la agrega.
- **contains()**: nos permite chequear si el elemento contiene una clase o no.

```
// Seleccionamos el elemento
var elem = document.getElementById("intro");
// Agregar una clase
elem.classList.add("hide");
// Agregar multiples clases
elem.classList.add("note", "highlight");
// Eliminar una clase
elem.classList.remove("hide"); // Remove a class
// Eliminar múltiples clases
elem.classList.remove("disabled", "note");
// Si la clase existe se elimina, sino se agrega
elem.classList.toggle("visible");
// Determinar si el elemento tiene la clase
if(elem.classList.contains("highlight")) {
    alert("El elemento tiene la clase especificada.");
}
```

JavaScript - DOM - Atributos y Manipulación del DOM

Atributos

Con el método **getAttribute** podemos obtener el valor del atributo, mientras que con **setAttribute** podemos agregar o modificar los atributos del elemento. Al mismo tiempo, con **removeAttribute** podemos eliminar un atributo particular.

```
<a href="https://www.google.com/" id="myLink">Google</a>
<script>
// Seleccionamos el elemento a través de su ID
let link = document.getElementById("myLink");
// Obtenemos el valor del atributo href
let href = link.getAttribute("href");
alert(href); // Resultado: https://www.google.com/
// Seteamos atributos
href.setAttribute("href", "_blank");
href.setAttribute("class", "my-link");
href.setAttribute("disabled", "");
// Eliminamos el atributo target
href.removeAttribute("href");
```

Manipulación del DOM

Desde JS podemos crear elementos, setear contenido a ese nuevo elemento e incluso inyectarlo dentro del documento para hacerlo visible al usuario.

```
<div id="main-content">
    <h1 id="title">Hola Mundo!</h1>
</div>
<script>
// Creamos un elemento <p></p>
var newP = document.createElement("p");
// Creamos un nodo de texto
var newContent = document.createTextNode("Cómo estás?");
// Agregamos el texto dentro del párrafo creado
newP.appendChild(newContent);
// Agregamos el nuevo <p> como hijo del main-content
var currentDiv = document.getElementById("main-content");
currentDiv.appendChild(newP);
</script>
```

En la siguiente **página** podrá profundizar sobre agregar nuevos elementos, eliminar, obtener el contenido, etc..

JavaScript - Eventos

Entendiendo los eventos y controladores de eventos (handlers - listeners)

Un evento es algo que sucede cuando el **usuario interactúa con la página web**, como cuando hace clic en un enlace o botón, ingresa texto en un cuadro de entrada o área de texto, hace una selección en un cuadro de selección, presiona una tecla en el teclado, mueve el puntero del mouse , envía un formulario, etc. También se consideran eventos a aquellos que el **navegador mismo puede desencadenar**, como los eventos de carga y descarga de la página.

Cuando ocurre un evento, puede usar un controlador de eventos de JavaScript (**handler**) o un detector de eventos (**listener**) para detectarlos y realizar una tarea o un conjunto de tareas. Por convención, los nombres de los controladores de eventos siempre comienzan con la palabra "on", por lo que un controlador de eventos para el evento de clic se llama onclick, de manera similar, un controlador de eventos para el evento de carga se llama onload, el controlador de eventos para el evento de desenfoque se llama onblur , etcétera.

Hay varias formas de asignar un controlador de eventos. A continuación, veremos las dos más comunes:

1. Event handler desde atributos del elemento

```
<button type="button" onclick="onClickBtnGreet()">Saludar</button
>
<script>
function onClickBtnGreet() {
    alert("Hola!");
}
</script>
```

2. Event handler desde JavaScript

```
<button type="button" id="my-btn-greet">Saludar</button>
<script>
// Declaro la función
function onClickBtnGreet() {
    alert("Hola!");
}
// Obtengo el elemento Button
let myBtnGreet = document.getElementById("my-btn-greet");
// Asocio función a evento click
myBtnGreet.onclick = onClickBtnGreet;
</script>
```

JavaScript - Eventos

Listado de eventos

A continuación, se listan los eventos más comunes:

Eventos del Mouse

- **onclick**: evento que se dispara cuando se realiza click (izquierdo) sobre el elemento.
- **oncontextmenu**: ocurre cuando se realiza click derecho.
- **onmouseover**: ocurre cuando el usuario posiciona el puntero sobre el elemento.
- **onmouseout**: ocurre cuando el usuario quita el puntero del elemento.

Eventos del Teclado

- **onkeydown**: ocurre cuando el usuario presiona una tecla del teclado.
- **onkeyup**: ocurre cuando el usuario suelta la tecla.
- **onkeypress**: al igual que onkeydown pero sólo para teclas de caracteres (no se dispara si se presiona Ctrl, Shift, etc.).

Eventos de Formulario

- **onfocus**: ocurre cuando el usuario se posiciona sobre un elemento.
- **onblur**: ocurre cuando el usuario quita el foco de un elemento.
- **onchange**: ocurre cuando se produce un cambio en el elemento (conmumente utilizado para los inputs, selects, radios, etc.)
- **onsubmit**: ocurre cuando el usuario presiona el botón de enviar. Se utiliza sobre el elemento <form>.

Eventos del Documento (Document/Window)

- **onload**: ocurre cuando una página se ha cargado por completo en el navegador.
- **onunload**: ocurre cuando el usuario cierra la página.
- **onresize**: ocurre cuando se modifica el tamaño de la ventana.

Vea ejemplos de cada uno de estos eventos en la siguiente **página**

Listado completo de Eventos, **aquí**.

Preguntas?

JavaScript - Window

El objeto **window** representa una ventana que contiene un documento DOM. Una ventana puede ser la ventana principal, un conjunto de marcos o un marco individual, o incluso una nueva ventana creada con JavaScript.

En capítulos anteriores, hemos utilizado el método alert() en nuestros scripts para mostrar mensajes emergentes. Este es un método del objeto ventana.

En los próximos capítulos, veremos una serie de nuevos métodos y propiedades del objeto **window** que nos permite hacer cosas como solicitar información al usuario, confirmar la acción del usuario, abrir nuevas ventanas, etc., lo que le permite agregar más interactividad a sus páginas web.

Obtener el tamaño de pantalla

A través del objeto **window** podemos obtener el alto y ancho de la ventana del navegador en px, puntualmente a través de las propiedades innerWidth y innerHeight del objeto en cuestión. Vea el siguiente ejemplo:

```
<button onclick="windowSize();">
Mostrar tamaño de pantalla</button>
<script>
function windowSize(){
    var w = window.innerWidth;
    var h = window.innerHeight;
    alert("Ancho: " + w + "px, " + "Alto: " + h + "px.");
}
</script>
```

JavaScript - Location

La propiedad **location** del objeto **window** (window.location) nos permite obtener información relacionada a la url actual en la que se encuentra el navegador. En las últimas versiones de JS, ya no es necesario hacer window.location, sino que tenemos disponible la propiedad location de manera global.

Obtener la url actual

A través de **location.href** podemos obtener la url de la página actual.

Dirigir a otro documentos/página

Con location.assign('https://nueva-url.com') podremos redirigir al usuario "programáticamente", es decir, a través del código JS. También se puede cargar/redireccionar con **location.href**.

```
<button onclick="loadGooglePage();">Ir a Google</button>
<script>
function loadGooglePage() {
    location.assign("https://google.com/");
    // Es lo mismo que hacer:
    // location.href = "https://google.com/";
}
</script>
```

Recargar la página

Con **location.reload(true)** podemos recargar la página dinámicamente desde JS. Sería como si el usuario apreta F5.

JavaScript - History

history es una propiedad del objeto **window** y al mismo tiempo podemos acceder a ella como variable global, es decir, sin necesidad de hacer **window.history**.

Obtener el número de páginas visitadas

history.length nos permite saber el número de páginas que han sido visitadas antes de estar en la página actual.

Ir atrás

Para ir atrás, es decir, a la página visitada con anterioridad, contamos con **history.back()**. Es lo mismo que apretar el botón de navegación hacia atrás de nuestro navegador.

Ir adelante

Para ir hacia delante en el historial, **history.forward()**. Es lo mismo que apretar el botón de navegación hacia delante de nuestro navegador.

Ir a punto específico del historial

Para ir a un punto específico del historial, ya sea hacia adelante o hacia atrás podemos utilizar **history.go(N).** Donde N puede ser cualquier número entero negativo o positivo.

```
window.history.go(-2);  // Ir atrás 2 páginas
window.history.go(-1); // Ir atrás 1 página
window.history.go(0);  // Recargar la actual
window.history.go(1);  // Ir adelante una página
window.history.go(2);  // Ir adelante dos páginas
```

Si intentamos dirigirnos a una página que no existe en el historial, la acción será inhabilitada y el usuario permanecerá en la página en la que se encuentra.

JavaScript - Función Callback

Un callback es una función que se pasa a otra función como un argumento, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

Veamos el siguiente ejemplo:

```
function greet(name) {
    alert('Hola ' + name);
}
function processUserAccess(callback) {
    var name = prompt('Por favor ingresa tu nombre.');
    callback(name);
}
processUserAccess(greet);
```

En este ejemplo, el callback es sincrónico, ya que se ejecuta inmediatamente.

Sin embargo, tenga en cuenta que los callbacks a menudo se utilizan para continuar con la ejecución del código después de que se haya completado una operación asincrónica. Por ejemplo, cuando trabajamos con Eventos, o en el siguiente slide veremos ejemplos de callbacks asíncronos.

Se denominan **asíncronos** porque no sabemos exactamente cuándo se devolverán los datos o cuándo el usuario disparará el evento. Veremos este concepto en detalle, en la próxima unidad.

JavaScript - Timers

Un timer es una función que se ejecuta luego de una cantidad X de tiempo. Para ello JavaScript nos ofrece dos métodos:

Ejecutar código luego de una demora

La función **setTimeout(function, time)** recibe dos parámetro, por un lado una función a la cual se llamará cuando pase un determinado tiempo y el otro parámetro (**time**) es el tiempo en cuestión expresado en milisegundos.

```
<button onclick="showMessages();">Mostrar mensajes</
button>
<script>
function showMessages() {
    console.log("Yo aparezco ni bien se hace click");
    setTimeout(function() {
        console.log("Yo 5 segundos después")
    }, 5000);
}
```

Ejecutar código en intervalor regulares

La función **setInterval(function, interval)** ejecuta el código que le indiquemos de manera indeterminada durante el tiempo deseado. Para detener el intervalo, se debe utilizar la función **clearInterval()**

```
<button onclick="startInterval()">Iniciar intervalo</button>
<button onclick="stopInterval()">Detener intervalo</button>
<script>
let myInterval;
function startInterval() {
    myInterval = setInterval(function() {
        console.log("Mensaje mostrado cada 1 segundo")
    }, 1000);
}
function stopInterval() {
    clearInterval(myInterval);
}
</script>
```

Persistencia en el navegador / Almacenamiento de datos

JavaScript nos permite almacenar información del lado del cliente, es decir, en su navegador/computadora y acceder a ellos cuando sea necesario, lo cual tiene como principales usos:

- Personalizar las **preferencias del sitio** (por ejemplo, mostrar la elección respecto de dark/light mode, combinación de colores o tamaño del tipo de letra).
- Persistencia de la **actividad anterior** del sitio (por ejemplo, almacenar el contenido de un carrito de compras de una sesión anterior, recordar si un usuario inició sesión anteriormente).
- Guardar **datos y activos** (assets) localmente para que un sitio sea más rápido (y potencialmente menos costoso) de descargar o se pueda usar sin una conexión de red.

Existen diferentes maneras de persistir información en el navegador, a continuación se listan cada una de ellas:

Cookies (old school)

Desde los primeros días de la web, los sitios han utilizado cookies para almacenar información y personalizar la experiencia del usuario en los sitios web. Son la forma más antigua de almacenamiento de lado del cliente que se usa comúnmente en la web.

Que sea antigua no quiere decir que no sea útil, sólo que en este curso no entraremos en detalle de esta herramienta y nos centraremos en alternativas más modernas y más fáciles de utilizar. Para profundizar sobre Cookies, por aquí puede comenzar.

Persistencia en el navegador / Almacenamiento de datos

LocalStorage

Con localStorage podremos almacenar información del lado del cliente por tiempo indefinido sin importar que el navegador se cierre. La información se debe almacenar en pares clave/valor, y ambos deben ser un string. Se utiliza para almacenar volúmenes pequeños de información.

Guardar Información

Para almacenar información en localStorage, se debe hacer **localStorage.setItem("myVariable", "myValue")**.

Obtener Información

Para obtener cualquier información que hayamos almacenado en localStorage, debemos hacer **localStorage.getItem("myVariable")**.

Eliminar información

A la hora de eliminar información del localStorage, podremos eliminar una variable puntual haciendo **localStorage.removeItem("myVariable")** o bien limpiar toda la información haciendo **localStorage.clear()**.

```
// Guardar información
localStorage.setItem("name", "Jack");
localStorage.setItem("email", "jack@gmail.com");
// Obtener infromación
localStorage.getItem("name"); // Resultado: "Jack"
localStorage.getItem("email");
// Resultado: "jack@gmail.com"
// Elimino una variable
localStorage.removeItem("name");
// Elimino toda la información
localStorage.clear();
```

En caso de que el usuario ingrese en **modo incógnito**, toda la información que almacenemos en **localStorage** se limpiará cuando el usuario cierre la ventana incógnita.

El navegador crea una instancia de localStorage por cada dominio/sitio web, por lo que si en **misitioweb.com** almacena la variable "myVar", en **otrositio.com** no existirá manera de accederla.

Persistencia en el navegador / Almacenamiento de datos

SessionStorage

Funciona de la misma manera que **localStorage**, pero en **sessionStorage** la información que almacenemos se borrará automáticamente cuando el usuario cierre nuestra página.

IndexedDB

IndexedDB es una API de bajo nivel que ofrece almacenamiento en el cliente de cantidades significativas de datos estructurados, incluyendo archivos y blobs.
Es un sistema de base de datos completo disponible en el navegador en el que se puede almacenar datos relacionados, los cuales no se limitan a valores simples como cadenas o números. También, se puede almacenar videos, imágenes y casi cualquier otra cosa en una instancia de **IndexedDB**.

Propiedades principales:

- Permite almacenar grandes cantidades de datos.
- Permite almacenar datos estructurados.
- Cada base de datos tiene múltiples espacios de almacenamiento e índices.
- Permite búsquedas por varias claves e índices eficientemente.
- Soporta transaccionalidad.

Los detalles del funcionamiento de las bases de datos IndexedDB exceden el alcance de este curso. En caso de querer profundizar sobre este tema, puede comenzar por aquí.