

MÓDULO 1

C1A

Origen de JavaScript

- Javascript es uno de los 3 lenguajes principales que se utilizan para desarrollar los sitios webs.
- 1990: La Guerra de los navegadores: Internet Explorer (Microsoft) vs Navigator (Netscape - Marc Andreessen)
- 1995 - Brendan Eich (reclutado por Netscape para crear un lenguaje dinámico para Navigator, con la presión de hacerlo antes que Microsoft) en 10 días crea lenguaje llamado Mocha (sintaxis de **java**, funciones de 1º clase como **Scheme**, tipos dinámicos como **Lisp**, prototipos como **Self**)... Lo lanzaron al mercado como "LiveScript", pero como todos pensaron que Java era lo más asombroso del mundo deciden renombrarlo como JAVASCRIPT (por el mkt de Java, pero los lenguajes no se parecen)
- 1997: Javascript era un éxito, y se decide estandarizar para evitar competencia, se envía al organismo ECMA, quien define el lenguaje... Aquí muchos comienzan a denominarlo ECMAScript.
- 1998: Se lanza ECMAScript 3 version estandarizada con mas funcionalidades.
- 2006: Nace librería jQuery
- 2008: Nace Chrome y su motor V8 para JavaScript
- 2009: Nace Node.js. un entorno que permite ejecutar javascript fuera del navegador, del lado del servidor (backend)

Repaso de Conceptos JavaScript

- Variables:
 - *var*: Declara una variable global o en el ámbito de una función.
 - *let*: Declara una variable en el ámbito de un bloque.
 - *const*: Declara una constante en el ámbito de un bloque.
- Tipos
 - `let myVariable = 'Hello world';` // es un string
 - `let myVariable1 = 22;` // es un number
 - `let myVariable2 = false;` // es un boolean
 - `let myVariable3;` // es un undefined
 - `let myVariable4 = { nombre: 'mi nombre' };` // es un objeto let
 - `myVariable5 = null;` // es un objeto (es un tipo de objeto especial) let
 - `myVariable6 = function() { let doSomething; };` // es una function

// Se pueden comprobar estos tipos mediante el uso de
`typeof myVariable` // -> number

DevTools: Consola

- La consola es una herramienta que tenemos en el navegador para tomar decisiones sobre nuestro proyecto al mismo tiempo que es interpretado por Chrome.
- Tipos de errores:
 - `SyntaxError`: error de sintaxis
 - `TypeError`: error que ocurre cuando una variable o parámetro no es de un tipo válido, es decir, undefined.

JavaScript: Baby Steps

- console.log: mostrar un mensaje por consola, útiles para debuggear nuestro código.
- variantes al console.log → .error() - .warn() - .table()
- alert() → pertenece al objeto Window, son mensajes de alerta que puede ver el usuario.

C1S

Vinculando HTML y JavaScript

- *Interna*: nos permite escribir código JS directo en el archivo html, no es lo mejor

```
{  
  <body>  
    ...  
    <script>  
      console.log("Hola Mundo!");  
    </script>  
  </body>  
}
```

- *Externa*: nos permite linkear el archivo html con un archivo JS ext.

```
{  
  <body>  
    ...  
    <script src="js/main.js"></script>  
  </body>  
}
```

C2A

Capturando datos del Cliente

- 3 acciones para interactuar con el usuario de manera rudimentaria:
 - ALERT() interrumpe la navegación - solo comunica - boton de OK
 - CONFIRM() muestra un cuadro de diálogo con un mensaje opcional y dos botones, "Aceptar" y "Cancelar". En este caso, lo que nos permite es ingresar alguna pregunta o indicación al usuario para que este responda por sí o no únicamente. El valor que nos va a retornar es un booleano indicando true si pulsamos Aceptar y false si elegimos Cancelar. Tmb frena la naveg.
 - PROMPT() muestra un cuadro de diálogo con mensaje opcional, que solicita al usuario que introduzca un texto. Además tiene dos opciones: "Aceptar" o "Cancelar". La funcion me devuelve la rta del usuario en formato string. Si presiona cancelar devuelve null. Si presiona aceptar sin escribir nada nos devuelve string vacio.

Manipulando datos

- *Objeto Math*: incorporado a JS, tiene propiedades y métodos para constantes y funciones matemáticas.

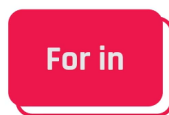
- **Propiedades:** ej: "Math.PI" - Otros: E - LN2 - LN10 - LOG2E - LOG10E - SQRT_2 - SQRT2
- **Métodos:** funciones matematicas que ya conocemos, ejemplo:

Método	Función
Math.random();	Retorna un punto flotante, un número pseudoaleatorio dentro del rango [0, 1).
Math.round();	Retorna el valor de un número redondeado al entero más cercano.
Math.max();	Devuelve el mayor de cero o más números.

- **Parseando**
 - parseInt() → parsea una cadena de texto y devuelve un número entero.
 - parseFloat() → parsea una cadena de texto y devuelve un número decimal.
- **NaN:** nos indica que el valor no es un número (Not A Number), por lo que esto nos produciría un error si queremos realizar alguna operación aritmética con este valor.
→ Tip: Tenemos la función isNaN(), la cual nos devuelve true si el valor dado como parámetro es NaN.

Bucles especificos

- For ... In → solo itera sobre objetos literales.



Propiedades Enumerables

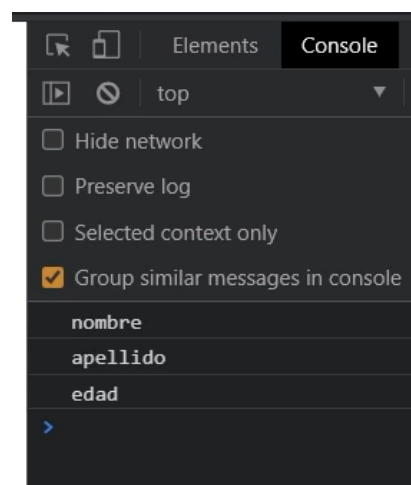
- ☒ ¿La uso con objetos?
- ☐ ¿La uso con Arrays?
- ☐ ¿La uso con Strings?

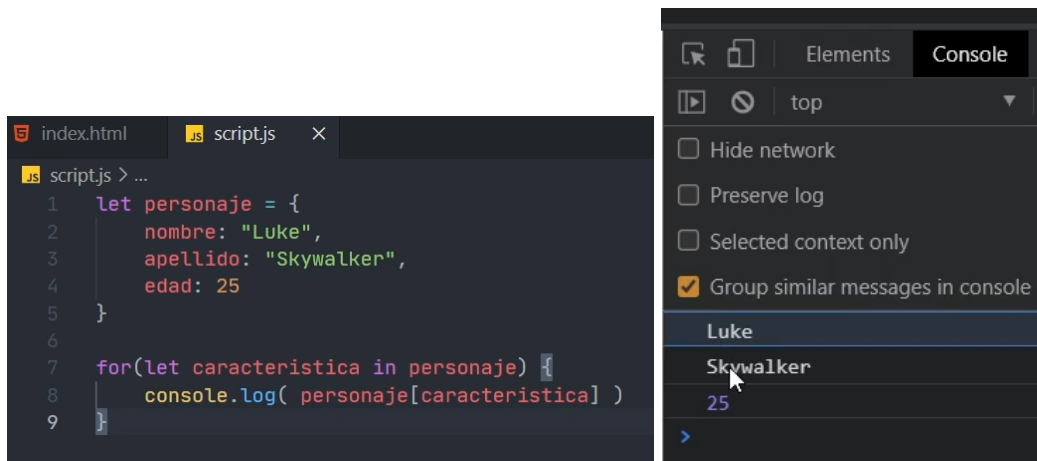
-

```

index.html  JS  script.js  X
JS script.js > ...
1  let personaje = {
2      nombre: "Luke",
3      apellido: "Skywalker",
4      edad: 25
5  }
6
7  for(let iteradora in personaje) {
8      console.log(iteradora)
9  }

```



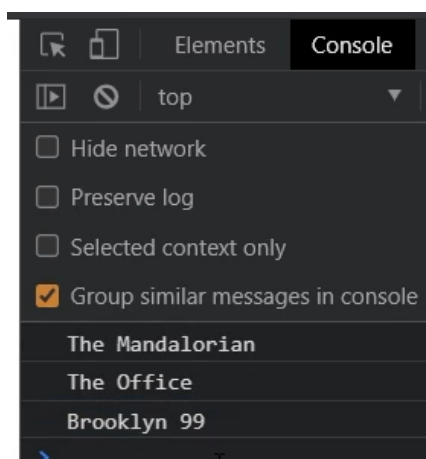


- For ... of → se utiliza para iterar sobre arrays o cadenas de texto



Elementos Iterables

- ☒ ¿La uso con objetos?
- ☒ ¿La uso con Arrays?
- ☒ ¿La uso con Strings?



MÓDULO 2 - Manipulación del DOM

C4A - Introducción al DOM

El Objeto window y document

- El objeto window representa la ventana que contiene al documento y el objeto document representa al DOM (documento HTML) cargado en esa ventana”.
- Hay que recordar que el DOM (document object model) representa al documento que se carga en el navegador como un árbol de nodos, en donde cada nodo representa una parte del documento.
- En resumen, window y document es la manera en la que JavaScript nos da acceso a los elementos presentes en el documento HTML para que a través de estas funcionalidades podamos manipular el contenido según nuestro criterio y necesidades.

Seleccionando elementos

querySelector()

Este selector recibe un string que indica el selector CSS del elemento del DOM que estamos buscando. Por ejemplo:

```
{ } let titulo = document.querySelector('.title');
```

Nos va a retornar el **primer** elemento del HTML que contenga la clase “title”.



Es importante declarar una variable para almacenar el dato que nos traiga el selector, ya que de otra manera lo perderíamos al continuar la ejecución del programa.

querySelectorAll()

Este selector recibe un string que indica el selector CSS del elemento del DOM que estamos buscando. Por ejemplo:

```
{ } let nombres=document.querySelectorAll('.name');
```

Nos va a retornar **un listado** de elementos que coincidan con la búsqueda especificada.



También podemos utilizar los selectores directamente con elementos del documento, por ejemplo:

```
let div=document.querySelectorAll('div');
```

getElementById()

Este selector recibe un string con únicamente el nombre del **id** del elemento del DOM que estamos buscando.

Por ejemplo:

```
{ } let marca=document.getElementById('marca');
```

Nos va a retornar el **elemento cuyo id coincida con el deseado**.



También podemos buscar elementos por su id mediante los selectores anteriores, pero debemos anteponer un # para aclarar que es un id.

```
let marca=document.querySelector('#marca');
```

querySelector()	querySelectorAll()	getElementById()
Retorna el primer elemento del DOM que cumple con la condición que buscamos.	Retorna todos los elementos del DOM que cumplen con la condición que buscamos.	Retorna el elemento del DOM que cumpla con el id que buscamos.

C5A - Modificando elementos con JS

Modificando el DOM

- Inner HTML → para leer o modificar el contenido de la etiqueta HTML (se pueden utilizar texto + etiquetas html)

```
{ } document.querySelector('div.nombre').innerHTML += 'Papitas';
```

- Inner TEXT → para leer o modificar el texto de una etiqueta HTML (solo texto plano)

```
{ } document.querySelector('div.nombre').innerText += 'Messi';
```

Plantillas de texto

- Template strings o template literals → es una de las bases de la programación dinámica en la web

Sintaxis de un template string



Modificando estilos

- A través del DOM, podemos acceder al objeto style que define el estilo de un elemento seleccionado. Por ejemplo, style.color devuelve el color de un elemento

```
EJS index.ejs  EJS contacto.ejs  JS js-front.js •
public > js > JS js-front.js > [e] confirmaCambios
1  let confirmaCambios = confirm('¿Querés cambiar el color del título?');
2
3  if (confirmaCambios) {
4      let titulo = document.querySelector('h1');
5      titulo.innerHTML += ' Soy un contenido nuevo desde JS';
6      titulo.style.color = 'crimson';
7      titulo.style.fontSize = '50px';
8  }
```

Modificando clases

- Conjunto de estilos que quisiéramos agregar de manera constante

.add()	.remove()	.toggle()	.contains()
Agrega la clase al elemento.	Elimina la clase del elemento.	Agrega la clase, si es que no la tiene. En caso de tenerla, la remueve.	Pregunta si el elemento tiene la clase o no. Devuelve un valor booleano.

classList.add()

Nos permite agregar una clase nueva al elemento que tengamos seleccionado.

```
{}  
let cita = document.querySelector('.cita');  
cita.classList.add('italicas');
```

Antes

```
{}  
<p class="cita">El veloz  
murciélago comía feliz  
cardillo y kiwi</p>
```

Después

```
{}  
<p class="cita italicas">El  
veloz murciélago comía  
feliz cardillo y kiwi</p>
```

classList.remove()

Nos permite quitarle una clase existente al elemento que tenemos seleccionado.

```
{}  
let cita = document.querySelector('.cita');  
cita.classList.remove('cita');
```

Antes

```
{}  
<p class="cita">El veloz  
murciélago comía feliz  
cardillo y kiwi</p>
```

Después

```
{}  
<p class="">El veloz  
murciélago comía feliz  
cardillo y kiwi</p>
```

classList.toggle()

Revisa si existe una clase en el elemento seleccionado. De ser así, la remueve, de lo contrario, si la clase no existe, la agrega.

```
{}  
let cita = document.querySelector('p');  
cita.classList.toggle('cita');
```

Antes

```
{}  
<p class="italicas">El  
veloz murciélago comía  
feliz cardillo y kiwi</p>
```

Después

```
{}  
<p class="italicas cita">El  
veloz murciélago comía  
feliz cardillo y kiwi</p>
```


classList.contains()

Nos permite preguntar si un elemento tiene una clase determinada. Devuelve un **valor booleano**.

```
{  
  let cita = document.querySelector('.itálicas');  
  cita.classList.contains('cita'); // false  
}
```

```
{  
  let cita = document.querySelector('.itálicas');  
  cita.classList.contains('itálicas'); // true  
}
```

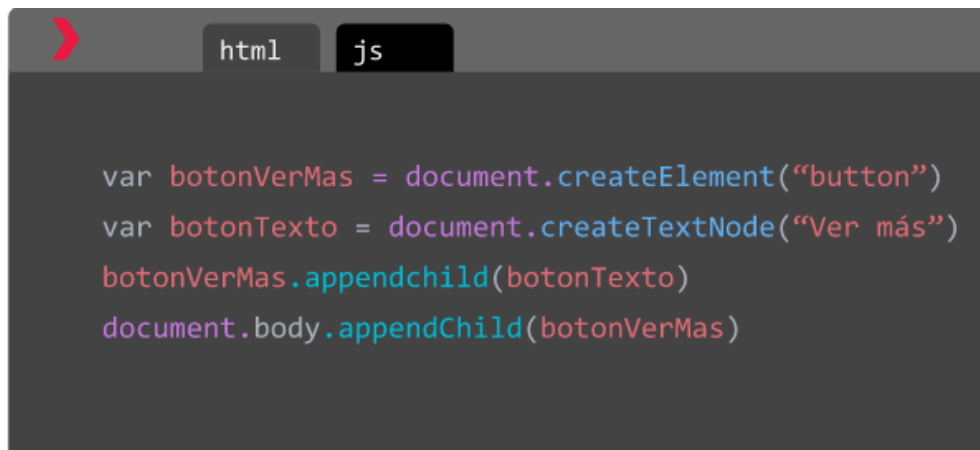
C6A - Trabajando con nodos

Nodos en HTML

- Son elementos o etiquetas del HTML que en conjunto forman un “árbol de nodos” al que llamamos DOM (Document Object Model).
- Entonces, en JavaScript, el nodo objeto principal es el document, y dentro de él, se clasifican estos otros:
 - Todas las etiquetas del HTML que son nodos de elementos.
 - Los nodos de atributos de los elementos.
 - Los nodos de texto.
 - Los nodos de comentarios.
- Cada nodo del árbol es un objeto, es decir, que contienen una colección de propiedades.

Métodos del objeto document

Método	
1. createElement()	Crea un nodo de tipo elemento según el nombre de la etiqueta de HTML que le indiquemos.
2. createTextNode()	Crea un nodo de texto explicitado entre comillas. No se visualiza hasta asignarlo a un elemento existente del DOM.
3. appendChild()	Adhiere dentro del DOM un elemento hijo a un elemento padre. Si el elemento padre ya existía en el documento, cambia su posición hacia el otro elemento padre indicado. Si no existe, lo creamos con el método 1.

A screenshot of a code editor with two tabs: 'html' and 'js'. The 'js' tab is active, showing the following JavaScript code:

```
var botonVerMas = document.createElement("button")
var botonTexto = document.createTextNode("Ver más")
botonVerMas.appendChild(botonTexto)
document.body.appendChild(botonVerMas)
```

Elementos y atributos dinámicos

- Lo dinámico está en manipular completamente los posibles atributos desde nuestro código JavaScript. En el HTML los agregamos de manera estática, pero ahora desde JS podemos leerlos, agregar nuevos o eliminarlos gracias a distintos métodos que veremos a continuación.

hasAttribute()

Este método nos sirve para consultar si el elemento posee o no un determinado atributo. Funciona de la siguiente manera:

- **Recibe** un atributo.
- **Retorna** true si el atributo existe, de lo contrario false.

```
//Seleccionamos un elemento del HTML
let elemento = document.querySelector("#portada");

//Consultamos si tiene un atributo src
elemento.hasAttribute("src"); // true
```

getAttribute()

Este método nos permite obtener el valor de un determinado atributo. Funciona de la siguiente manera:

- **Recibe** el nombre un atributo.
- **Retorna** el valor si existe, de lo contrario nos devuelve **null**.

```
//Seleccionamos un elemento del HTML
let elemento = document.querySelector("#portada");

//Pedimos el valor del atributo
elemento.getAttribute("src"); // imagen_portada.jpg
```

removeAttribute()

Este método borra por completo el atributo y sus valores del elemento. Si no lo encuentra, no hace nada. Funciona de la siguiente manera:

- **Recibe** el nombre un atributo.
- En cualquier caso, **no retorna ningún valor**.

```
//Seleccionamos un elemento del HTML
let elemento = document.querySelector("#portada");

//Pedimos el valor del atributo
elemento.removeAttribute("src");
```



Este caso repercute en el HTML ya que una imagen sin **src** se muestra como rota.

setAttribute()

Este método nos permite agregar un atributo con su respectivo valor al elemento seleccionado. Funciona de la siguiente manera:

- **Recibe** el nombre del atributo y un valor para el mismo.
- En cualquier caso, **no retorna ningún valor**.

```
//Seleccionamos un elemento del HTML
let elemento = document.querySelector("#portada");

//Pedimos el valor del atributo
elemento.setAttribute("src", "imagen_portada.jpg");
```

Módulo 3: Web Reactiva

C8A - Eventos

¿Qué son los eventos?

- Es una acción que transcurre en el navegador o que es ejecutada por el usuario.
- Dos formas de sintaxis: la primera con la función que llega como CB y la segunda con la función adentro del igual. En `onLoad` se ejecuta una vez, se pisa todas las otras veces. En cambio el `addEventListener` se ejecuta todas las veces, no se pisa.

```
terminal Help
</> home.html JS script.js style.css
JS script.js > onload
1  window.addEventListener("load", function() {
2
3  })
4
5  window.onload = function() {
6
7  }
```

- En un evento el objeto `THIS` hace referencia al lugar donde se ejecuta el evento, ejemplo: botón o etiqueta A.

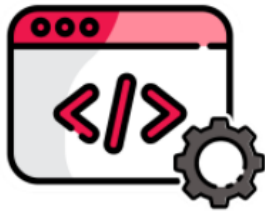
```
Go Debug Terminal Help
</> home.html JS script.js style.css
JS script.js > window.addEventListener("load") callback > aboutButton.addEventListener("click") callback
1  window.addEventListener("load", function() {
2      let homeButton = document.querySelector(".home-button")
3
4      homeButton.addEventListener("click", function() {
5          alert("Tocaste el boton!")
6      })
7
8      let aboutButton = document.querySelector(".about-button");
9
10     aboutButton.addEventListener("click", function(e) {
11         e.preventDefault();
12
13         console.log(this);
14
15         alert("Quisiste saber sobre el about?");
16     })
17 })
18
```

onclick

Este evento nos permite ejecutar una acción cuando se haga **click** sobre el elemento al cual le estamos aplicando la propiedad.

```
btn.onclick = function(){
    console.log('hiciste clic!');
}
```

preventDefault()



Nos permite **evitar** que se ejecute el evento predeterminado —o nativo— del elemento al que se lo estamos aplicando.

Podemos usarlo, por ejemplo, para prevenir que una etiqueta “a” se comporte de manera nativa y que haga otra acción.



Siempre tenemos que tener seleccionado el elemento al que le queremos aplicar el `preventDefault()` mediante los selectores.

Ejemplo

```
let hipervinculo = document.querySelector('a');

hipervinculo.addEventListener('click', function(event){
  console.log('hiciste click');
  event.preventDefault();
});
```

Diagram illustrating the code execution flow:

- Atrapamos el elemento.** (Red arrow pointing to `document.querySelector('a')`)
- Atrapamos el evento.** (Blue arrow pointing to `addEventListener('click', ...)`)
- Prevenimos la acción nativa.** (Orange arrow pointing to `event.preventDefault();`)

Eventos más usados

onclick	Cuando el usuario hace clic.
ondblclick	Cuando el usuario hace doble clic.
onmouseover	Cuando el mouse se mueve sobre el elemento
onmousemove	Cuando se mueve el mouse.
onscroll	Cuando se hace scroll.
onkeydown	Cuando se aprieta una tecla.
onload	Cuando se carga la página
onsubmit	Cuando se envía un formulario.

Eventos de mouse

mouseover



```
{  
  let texto = document.querySelector('.text');  
  texto.onmouseover = function(){  
    console.log('pasaste el mouse');  
  }  
}
```

También podríamos hacer:

```
{  
  texto.addEventListener('mouseover', function(){  
    console.log('pasaste el mouse');  
  });  
}
```

mouseout

```
{  
  let texto = document.querySelector('.text');  
  texto.onmouseout = function(){  
    console.log('quitaste el mouse');  
  }  
}
```

También podríamos hacer:

```
{  
  texto.addEventListener('mouseout', function(){  
    console.log('quitaste el mouse');  
  });  
}
```

Eventos de teclado

- Keydown: se dispara al presionar una tecla



```
{  
  let miInput = document.querySelector('#miInput');  
  miInput.onkeydown = function(event){  
    alert("Se presionó la tecla: "+ event.key);  
  }  
}
```

- Keyup: se dispara al soltar la tecla



```
{  
  let miInput = document.querySelector('#miInput');  
  miInput.onkeyup = function(event){  
    alert("Se soltó la tecla: "+ event.key);  
  }  
}
```

- Keypress: de dispara al finalizar el recorrido completo de presion y liberacion de la tecla.

```

let miInput = document.querySelector('#miInput');
miInput.onkeypress = function(event){
    alert("Se presionó la tecla: "+ event.key);
}

```

Invocando funciones

- El *scope* es el contexto actual de ejecución. Ese contexto se refiere al “ámbito de vida” de las variables. Las mismas que “nacen” en un determinado bloque, “mueren” en ese bloque. También entra en juego la jerarquía de los bloques, esto quiere decir que los scopes secundarios tienen acceso a los ámbitos primarios, pero no al revés.
- *Arrow functions* → otra notación para declarar funciones.
 - Debe ser utilizada con una variable cuyo nombre será el nombre de la función.
 - Si luego de utilizar '=>', no se abren llaves, lo siguiente será devuelto como si se tratara de un return, pero la función no puede tener más de una línea.
 - Es utilizada mayormente cuando pasamos como parámetro una función.
- Funciones como parámetros o Callbacks

C11A - Formularios I

Estructura de un formulario

Elementos de formularios

- **Inputs**
 - Son los elementos más comunes para ingresar datos.
 - Están definidos por la etiqueta llamada de la misma manera (input)
 - Mediante el atributo type definimos el formato de entrada del campo.
 - Para los casos de radio y checkbox son importantes los campos de name y de value, ya que con esto van a definir al grupo al que pertenecen y el valor que se entrega en caso de ser seleccionado respectivamente.

```

// input de texto
<input type="text">
// input que solo admite números
<input type="number">
// input para campos de email
<input type="email">
// input de fecha
<input type="date">
// grupo de opciones de selección única

```

```

<input type="radio" name="miOpcion" value="1">
<input type="radio" name="miOpcion" value="2">
<input type="radio" name="miOpcion" value="3">
// grupo de opciones de selección múltiple
<input type="checkbox" name="miOpcion" value="1">
<input type="checkbox" name="miOpcion" value="2">
<input type="checkbox" name="miOpcion" value="3"></input>

```

- Select:

- Son los campos que permiten seleccionar entre una lista desplegable de opciones.
- Al igual que en el caso de los radio y checkbox, acá también es importante el atributo value para definir dar valor a nuestra opción.

```

<select>
  <option value="opcion1"> nombre de la opción 0</option>
  <option value="opcion1"> nombre de la opción 1</option>
  <option value="opcion1"> nombre de la opción 2</option>
</select>

```

- Textarea:

- Se utilizan en caso de que se necesite ingresar una gran cantidad de texto.
- Generalmente se pueden ver utilizados para tener campos de comentario, mensajes, entre otros.

Obteniendo datos de un formulario

Los formularios web son uno de los principales puntos de interacción entre un usuario y un sitio web o aplicación, ya que permiten a los usuarios la introducción de datos, que generalmente se envían a un servidor web para su procesamiento y almacenamiento.

- Evitar enviar un formulario dos veces: Cuando se pulsa sobre el botón de envío de un formulario, se produce el evento **click** y por lo tanto, se ejecuta el envío de información de este; instrucción que por defecto sucede en todos los formularios.
 - No debemos mandar esa información hasta haber certificado el contenido de dicho formulario: campos obligatorios, formato de mail correcto, etc.
 - Para eso, necesitamos, a través de JavaScript, frenar el envío de datos. Esto lo podemos lograr con el **método preventDefault()**.

```

let formulario = document.querySelector("form");
formulario.addEventListener("submit", function(event) {
  event.preventDefault()
})

```

- Limitar el tamaño de caracteres de un textarea: El valor por defecto de los eventos en JavaScript es true. Si cambiamos esto por false, estaríamos evitando que el evento se produzca, por lo tanto, si lo hacemos con

onkeypress, la tecla presionada no se transforma en ningún carácter dentro del textarea.

De la siguiente manera se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y, por lo tanto, los caracteres adicionales no se añaden al textarea:

```
function limita(maximoCaracteres) {  
    var elemento = document.getElementById("texto");  
    if(elemento.value.length >= maximoCaracteres ) {  
        return false;  
    }  
    else {  
        return true;  
    }  
}
```

- Etiquetas de un form en HTML:
 - Etiquetas: BUTTON, INPUT, OPTION, LI
 - Type: submit, date, radio, ceckbox
 - Attribute: value → obtenemos el dato ingresado por el usuario

INPUT - type: radio

Analicemos este ejemplo

type
Crea un botón de tipo radio.

value
Acá va la información que se enviará si el usuario selecciona este radio-button.

```
html <input type="radio" name="medio" value="Efectivo">  
<input type="radio" name="medio" value="Débito" checked>
```

name
Para que el usuario pueda seleccionar solo una opción tenemos que asignarle el mismo nombre a todos los inputs de tipo radio que sean del mismo grupo.

checked
Es opcional, se encarga de preseleccionar la opción.

Almacenar el valor de un radio button

Tenemos que saber cuál de todos los input de tipo radio se ha seleccionado con la propiedad **checked**. Esta devuelve **true**, si fue seleccionado, y **false**, si no lo está.

```
<label for="">¿Acepta términos y condiciones?</label>  
<input type="radio" name="pregunta" value="si"> Si  
<input type="radio" name="pregunta" value="no"> No
```

Seleccionamos los elementos que tengan el mismo **name** para recorrerlos mediante un ciclo `forEach`. Luego, por cada elemento, mostramos por consola los valores de cada uno y si fue seleccionado.

```
var elementos = document.getElementsByName("pregunta");
elementos.forEach(function(elemento) {
    console.log(`Elementos: ${elemento.value}`)
    console.log(`Seleccionado: ${elemento.checked}`)
})
// Elemento: si Seleccionado: true
// Elemento: no Seleccionado: false
```

INPUT - type: text y number

Almacenar datos de un input text y number

Existen muchas formas que podemos implementar en el código para obtener datos. Vamos a ver unos ejemplos:

```
<input type="text" id="nombre" value="OpcionA">
<input type="number" id="numero" value="OpcionB">
```

```
var nombre = document.getElementById("nombre").value;
console.log(nombre) // OpcionA

var numero = document.getElementById("numero").value;
console.log(numero) // OpcionB
```

INPUT - type: CHECKBOX

Almacenar el valor de un checkbox

En este caso comprobamos cada **checkbox** de forma independiente al resto. Mientras que los **radio button** seleccionan de forma excluyente, los **checkbox** admiten más de una selección

```
<input type="checkbox" id="privacidad" value="privacidad">
He leído la política de privacidad
```

Seleccionamos los elementos que tengan el mismo **name** para recorrerlos mediante un ciclo **for** y ejecutamos que muestre por consola la lista de los valores de cada uno y si fue seleccionado: Seleccionamos cada elemento por su **id** y mostramos si fue seleccionado con **checked**.

```
var privacidad = document.getElementById("privacidad");
console.log(`Elementos: ${privacidad.value}`)
console.log(`Seleccionado: ${privacidad.checked}`)

// Elementos: privacidad
// Seleccionado: false
```

Normalizando datos: métodos de strings

- La normalización de datos es una serie de procesos, reglas o mecanismos que se utilizan para dar un formato común a los datos recolectados en una aplicación, independientemente de quién sea la persona que lo haya ingresado o la manera en lo que lo haya hecho.
- **Método Split()** → nos permite dividir los caracteres de un string sobre la base del criterio que deseemos, obteniendo como resultado un array que contiene cada uno de los substrings generados.

```
const peliculasFavoritas = document.querySelector('#input-peliculas');

console.log(peliculasFavoritas)
//"Harry Potter;Mi Villano Favorito;Avatar"
```

```
const peliculasNormalizadas = peliculasFavoritas.split(';');

console.log(peliculasNormalizadas)
// [ 'Harry Potter', 'Mi Villano Favorito', 'Avatar' ]
```

- Método toLowerCase() → convierte caracteres a minúscula.
- Método toUpperCase() → convierte caracteres a mayúscula.
- Método Concat() → concatenar 2 o más strings en un único valor.
- Método Trim() → elimina los espacios en blanco en ambos extremos del string.
- Método replaceAll() → reemplaza caracteres

```
1 const dni =
2 document.querySelector("#input-dni").
3 value; // "23.345.678"
4
5 console.log(dni.replaceAll(".", ""));
6 // 23345898
7
8
```

Módulo 4: Validación del lado del cliente

C13A - Formularios II ¿Cómo validar?

Eventos de Formulario

- Evento Focus: cuando el usuario ingreso con el cursor dentro un campo input
- Evento Blur: sucede cuando el cursor abandona el campor donde se encuentra
- Evento Change: permite identificar que el valor de un campo cambió. Se puede aplicar en cualquier cambio, incluso sobre el form completo.
- Evento Submit: identifica el momento en que se clickea el botón o un campo input ambos de tipo submit. Acción nativa de html, el navegador va intentar ejecutarla cuando se clickea, para evitar q se envíe el form incluid preventDefault.

-

Objeto Location

- location.href → nos devuelve la URL
- location.reload → recargamos toda la página
- location.search → para obtener el query string completo

```
{} let query = new URLSearchParams(location.search);
if(query.has('search_query')){
    let search = query.get('search_query');
    console.log(search)
};
```

C14A - JSON y Storage

JSON

- JSON es un formato de texto sencillo para el intercambio de datos. Como su nombre lo indica, su implementación deviene de la notación de objetos de JavaScript. Está compuesto por clave valor, únicamente que es el caso de JSON, las propiedades van siempre entre comillas dobles. Esto último es un requisito esencial para su correcto funcionamiento.

JSON es el acrónimo de JavaScript Object Notation, y como su nombre lo indica, es muy similar al objeto literal que ya conocemos. Veamos las diferencias.

Objeto literal	JSON
Admite comillas simples y dobles	Las claves van entre comillas
Las claves del objeto van sin comillas	Sólo se pueden usar comillas dobles
Podemos escribir métodos sin problemas	No admite métodos, sólo propiedades y valores
Se recomienda poner una coma en la última propiedad	No podemos poner una coma en el último elemento

- `JSON.parse()` → recibe JSON por parámetro y retorna un objeto de JS.
- `JSON.stringify()` → recibe un objeto JS y retorna un JSON.

Session storage y Local storage

- Session storage: nos permitirá guardar info en sesión. Es decir, que si usamos esta opción y cerramos el navegador, la info se perderá.

```
{ } sessionStorage.setItem('key', 'value');  
{ } sessionStorage.getItem('key')  
{ } sessionStorage.removeItem('key');
```



Solo podemos almacenar datos en formato **string**.

- Local storage: los datos no tienen fecha de expiración

```
{ } localStorage.setItem('key', 'value');  
{ } localStorage.getItem('key')  
{ } localStorage.removeItem('key');
```



Solo podemos almacenar datos en formato **string**.

C16A - Introducción a asincronismo

AJAX

- AJAX (Asynchronous JavaScript and XML) es un conjunto de tecnologías que se utilizan para crear aplicaciones web asíncronas. Esto las vuelve más rápidas y con mejor respuesta a las acciones del usuario.
 - a. Se produce un evento en una página web (se carga la página, se hace clic en un botón)
 - b. JavaScript crea un objeto XMLHttpRequest
 - c. El objeto XMLHttpRequest envía una solicitud a un servidor web
 - d. El servidor procesa la solicitud.
 - e. El servidor envía una respuesta a la página web.
 - f. La respuesta es leída por JavaScript.
 - g. JavaScript realiza la acción adecuada (como la actualización de la página)

REQUEST Y RESPONSE

- **HTTP:** petición - respuesta → HTTP o Hypertext Transfer Protocol es un protocolo de intercambio de datos en la Web entre cliente y servidor. Los mensajes HTTP forman una estructura como medio para realizar una petición de datos iniciada por el cliente, normalmente un navegador web, en busca de su respuesta ejecutada por el servidor. De esta manera, se resuelve una o más tareas, a través de mecanismos que veremos más adelante.
- El protocolo HTTP es extensible, esto significa que con el tiempo se ha permitido que se implementen más funciones de control y funcionalidad sobre la Web: caché o métodos de identificación o autenticación.
- En versiones anteriores, los mensajes HTTP eran textos planos. En HTTP/2, los mensajes están estructurados en un nuevo formato, lo que contribuye a una mayor legibilidad y debugging más eficiente.
- **Asincronismo** → JavaScript es un lenguaje de programación asíncrono porque es capaz de ejecutar un hilo de tareas o peticiones en las cuales, si la respuesta demora, el hilo de ejecución de JavaScript continuará con las demás tareas que hay en el código.
Existen 2 tipos de asincronismo:
 - *Concurrencia:* cuando las tareas pueden comenzar, ejecutarse y completarse en períodos de tiempo superpuestos, en donde al menos dos hilos están progresando
 - *Paralelismo:* cuando dos o más tareas se ejecutan exactamente al mismo tiempo.
- La diferencia entre la concurrencia y el paralelismo está en que, en el primer caso, no implica que las tareas terminen de ejecutarse al mismo tiempo literalmente como sí ocurre en el segundo caso. Además, decimos que JavaScript es un lenguaje no-bloqueante porque las tareas no se quedan bloqueadas esperando a que finalicen evitando proseguir con el resto de tareas.

Asincronismo



- URI → Identificador de Recursos Uniformes, es un bloque de texto que se escribe en la barra de direcciones de un navegador web y está compuesto por dos partes: la URL y la URN

URL

Indica **dónde** se encuentra el recurso que deseamos obtener y siempre comienza con un **protocolo**. En este caso HTTP.

http://www.digitalhouse.com/preguntas-frecuentes

URN

Es el **nombre exacto** del recurso uniforme. El nombre del dominio y, en ocasiones, el nombre del recurso.

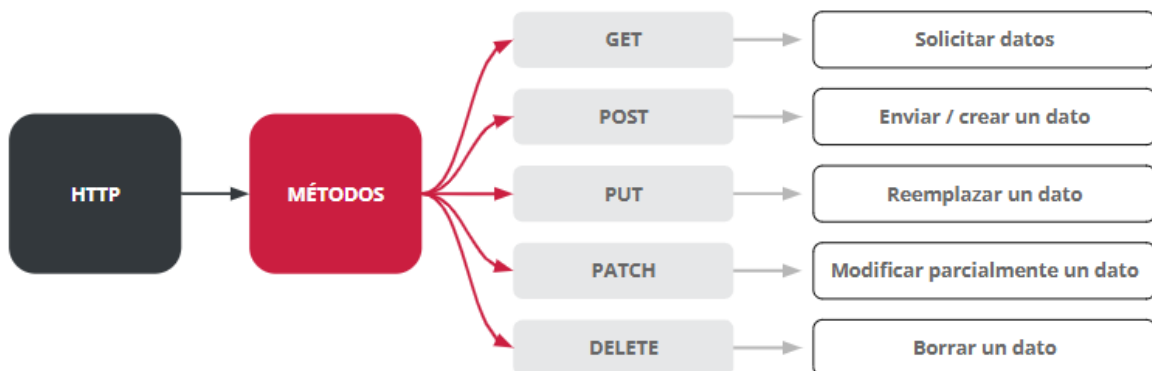
- REQUEST → cada vez que el cliente solicita un recurso al servidor
- RESPONSE → cada vez que el servidor le devuelve una respuesta al cliente
- MÉTODOS DE PETICIÓN → Cada método representa una acción y, si bien comparten algunas características, implementan funcionalidades diferentes entre sí. Los métodos más utilizados por este protocolo son:
 - GET → Se utiliza para pedirle información al servidor de un recurso específico. Cada vez que escribimos una dirección en el navegador o accedemos a un enlace, estamos utilizando el método GET. En caso de querer enviar información al servidor usando este método, la misma viajará a través de la URL.
 - POST → Se utiliza para enviar datos al servidor. Este método es más seguro que GET, ya que la información no viaja a través de la URL.

- DELETE → Borra un recurso presente en el servidor. Cuando eliminamos un posteo en Facebook, por ejemplo, estamos utilizando este método.
- PUT → Es muy parecido a POST. Se usa para reemplazar toda la información actual de un recurso presente en el servidor.
- PATCH → Similar a PUT. Es utilizado para aplicar modificaciones parciales a un recurso en el servidor.



PUT y PATCH suelen ser lo mismo. Elegir entre uno y otro va a depender del contexto y lo que queramos implementar en nuestra aplicación. Al editar un posteo o un perfil estaremos usando alguno de estos dos métodos.

En resumen, tenemos 5 métodos y cada uno de ellos tiene un propósito.



Códigos de estado HTTP → Cada vez que el servidor recibe una petición o request, este emite un código de estado que indica, de forma abreviada, el estado de la respuesta HTTP. El código tiene tres dígitos. El primero representa uno de los 5 tipos de respuesta posibles:

- 1 __ Respuestas informativas
- 2 __ Respuestas exitosas
- 3 __ Redirecciones
- 4 __ Errores del cliente
- 5 __ Errores de servidor

Algunos de los códigos más usados son:

- 200: OK → La petición se realizó con éxito.
- 301: Moved Permanently → El recurso se ha movido.
- 302: Found → El recurso fue encontrado.
- 304: Not Modified → El recurso no cambió, se cargará desde el caché.
- 400: Bad Request → El pedido está mal.
- 401: Unauthorized → No estás autorizado, seguramente debes autenticarte.
- 403: Forbidden → El pedido está prohibido y no debería repetirse.

404: Not Found → El recurso no fue encontrado.
500: Internal Server Error → Hubo un error en el servidor.
503: Service Unavailable → El servicio solicitado no está disponible.
550: Permission denied → Permiso denegado.

- **HTTPS** → es un protocolo mejorado de HTTP. Usando este protocolo, el servidor codifica la sesión con un certificado digital.

C17A - APIs I

API

- API → Application Programming Interface, es una interfaz que permite la comunicación entre 2 aplicaciones.
- Es una URL que devuelve información.
- Cada API tiene su propia información
- Hay públicas (RESTCountries), privadas (netflix) o semipública (twitter)
- ENDPOINT: punto de conexión donde necesitamos apuntar para obtener la info que queremos. Son las URL que debemos utilizar para obtener la información de un servidor a través de una api.

REST

- Un sistema REST busca implementar un esquema o protocolo que le permita a todos los sistemas que se comunican con él entender en qué forma lo tienen que hacer y bajo qué estructura deberán enviar sus peticiones para que sean atendidas. Adentrémonos en el video para conocer un poco más al respecto este tema.
- Es un tipo de arquitectura de servicios que proporciona estándares entre sistemas informáticos para establecer cómo se van a comunicar entre sí.
- REST es una arquitectura del tipo cliente-servidor porque debe permitir que tanto la aplicación del cliente como la aplicación del servidor se desarrollen o escalen sin interferir una con la otra. Es decir, permite integrar con cualquier otra plataforma y tecnología tanto el cliente como el servidor.
- Sin estado (stateless) → REST propone que todas las interacciones entre el cliente y el servidor deben ser tratadas como nuevas y de forma absolutamente independiente sin guardar estado.
Por lo tanto, si quisiéramos —por ejemplo— que el servidor distinga entre usuarios logueados o invitados, debemos mandar toda la información de autenticación necesaria en cada petición que le hagamos a dicho servidor.
- CACHEABLE → En REST, el cacheo de datos es una herramienta muy importante, que se implementa del lado del cliente, para mejorar la performance y reducir la demanda al servidor.
- Formato de envío → Cuando el servidor envía una solicitud, este transfiere una representación del estado del recurso requerido a quien lo haya solicitado. Dicha información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), RAW, XML o texto sin formato, URL-encoded. JSON es el más popular.
 - JSON → Debemos agregar un encabezado en los headers que diga:
 - "Content-Type": "application/json"
 - RAW → Se utiliza para mandar datos con texto sin ningún formato en particular.

- ```
email%3Dcosme%40fulanito.fox%26password%3Dverysecret}
```

## AJAX FETCH - GET

- Fetch → es una función nativa que nos permite hacer pedidos a una API desde nuestro front-end.
- Recibe como primer parámetro la URL del endpoint al cual estamos haciendo el llamado asincrónico. Al no saber cuándo se completa la petición, el servidor devuelve una promesa.
- El primer then será el encargado de recibir un callback y retornará la respuesta de ese llamado asincrónico en formato JSON.
- Una vez que la respuesta de nuestro pedido está en formato JSON, a través de otra promesa, podemos hacer con nuestra respuesta lo que queramos.
- En el caso de haber algún error, el catch() se encargará de atraparlo y luego lo imprimirá por consola.

## TRY, CATCH Y FINALLY

- Manejo de errores:
- Los errores que se producen en un programa pueden ocurrir debido a nuestros descuidos, una entrada inesperada del usuario, una respuesta errónea del servidor, entre otras razones. Por lo general, un script es interrumpido y se detiene cuando esto sucede. Pero podemos evitarlo con try...catch que nos permite “atrapar” errores para que el script pueda funcionar igualmente.
- - La declaración try permite probar un bloque de código en busca de errores.
- - La declaración catch permite manejar el error.
- - La declaración throw permite crear errores personalizados.

- La declaración finally permite ejecutar código, después de intentar y capturar, independientemente del resultado.

## Sintaxis

```
try {
 Block of code to try
}
catch(err) {
 Block of code to handle errors
}
finally {
 Block of code to be executed regardless of the try / catch result
}
```

```
function myFunction() {
 let message, x;
 message = document.getElementById("intro");
 message.innerHTML = "";
 x = document.getElementById("demo").value;
 try { //Ejecutamos un try con condicionales arrojando(throw) un mensaje:

 if(x == "") throw "Contenido vacio";
 if(isNaN(x)) throw "No es un numero";
 x = Number(x);
 if(x > 10) throw "Numero demasiado alto";
 if(x < 5) throw "Numero demasiado bajo";

 }
 catch(err) { //Ejecutamos un catch para manejar el error mostrandolo en
el navegador
 message.innerHTML = "Error: " + err + ".";
 }

 finally { //Ejecutamos la accion que termina con la funcion para
devolver el valor requerido
 document.getElementById("demo").value = "";
 }
}
```

- Tenemos en cuenta que un error puede provenir de valores diferentes:
- RangeError → Se ha producido un número "fuera de rango".
- ReferenceError → Ha ocurrido una referencia ilegal.
- Error de sintaxis → Ha ocurrido un error de sintaxis.
- Error de teclado → Ha ocurrido un error de tipo.
- URIError → Se ha producido un error en encodeURIComponent ().

## C18A - APIs II

### AJAX FETCH - POST

- Tiene un segundo parámetro, el objeto que queremos enviar al servidor.

### POSTMAN

- es una herramienta muy útil a la hora de testear una API. Naveguemos un poco esta gran aplicación para aprender a utilizarla. [Hacé clic para iniciar el recorrido.](#)

## C20A - To-Do App: workflow

### BUENAS PRÁCTICAS

- Principio de responsabilidad única → Como su nombre lo indica, este principio indica que cada clase, función o módulo debe ser responsable de una tarea específica, y dicha tarea debe estar encapsulada dentro de dicha clase, función o módulo.
- Keep it simple Stupid! (KisS) → Este principio, que en español podemos traducir como “mantenlo simple, estúpido!” es tan simple como su nombre lo indica 😊. Básicamente, la idea detrás del mismo es que cualquier sistema o programa, funciona mejor si se mantiene simple que si se hace complejo. Por ello, la idea es evitar agregar cualquier capa de complejidad que no sea estrictamente necesaria para el correcto funcionamiento del sistema.
- Don't repeat yourself (DRY). → Este principio, nos invita a pensar nuestro código de forma tal de evitar repeticiones innecesarias de código. Para ello, debemos pensar nuestro código de manera abstracta, con especial énfasis en las funciones que el mismo debe cumplir independientemente de un caso concreto. Además, en la medida de lo posible, debemos tener en cuenta la normalización de la información, de manera de evitar redundancia.

## C21A - To-Do App: security

### Autenticación y Autorización

- Principio de respon