Frontend Cheatsheet

Javascript y un poco de html

Pablo Leonel Cristaldo

|  |  |
| --- | --- |
| **Variables** | var num1; // scope global  let num2; // scope de bloque  const num3;// scope de bloque y            // no se puede reasignar valor |
| **Function** | function name(params) {      return dato  } |
| **Operadores Logicos** | +  suma // - resta // \* multiplicacion  // / Division  ++ incremento 1 // -- decremento 1  // % modulo (resto)  == comparacion simple // === comparacionestricta  ! negacion // > < mayor, menor  <= >= menor o igual, mayor o igual  && AND  || OR |
| **If else** | if (condition) {    } else {    } |
| **If ternario** | condition ? "si es true" : "si es false"  Ejemplo  4 > 10 ? "El 4 es mas grande" : "El 10 es mas grande" |
| **Switch**  Compara la key con el valor de cada case, en caso de ser coincidente con el valor del case se ejecuta ese case. | switch (key) {      case value:            break;      default:          break;  } |
| **isNaN()**  Stands for “is Not a Number”, if variable is not a number, it return true, else return false. | if(isNaN(num1)){    /\* si la variable dentro del isNaN no es un    numero se ejecuta esta linea de codigo \*/   }else{    /\* si la variable ES un numero se ejecuta esta. \*/   } |

|  |  |
| --- | --- |
| **Metodos de Strings** | |
| **.indexOf()**  Busca el string que recibe como parametro y retorna el indice donde empieza si es que lo encontro, si no lo encuentra devuelve -1 | let saludo = "Hola, estamos programando"  saludo.indexOf("estamos") //devuelve 6  saludo.indexOf("Estamos") //devuelve -1 CaseSensitive  saludo.indexOf("o")//devuelve 1, la primer "o"  saludo.indexOf("perro") //devuelve -1 |
| **.slice()**  Corta el string y devuelve una parte donde se aplica.  Puede tener inicio y final o solo inicio e ir hasta el final del string. | let frase = "Breaking Bad Rules!"  frase.slice(9,12) // devuelve Bad  frase.slice(13) // devuelve Rules!  frase.slice(-10) // devuelve Bad Rules |
| **.trim()**  Elimina los espacios en blanco del inicio y el final del string. | let nombre = "   pablo cristaldo   "  nombre.trim() // devuelve "pablo cristaldo"  let nombre = "   pablo      cristaldo   "  nombre.trim() // devuelve "pablo      cristaldo" |
| **.split()**  Recibe un string que usara como separador de las partes y no lo agregara al nuevo. Devuelve un array con las partes del string. | let cancion = "And bingo was his name, oh!"  cancion.split(" ")  //devuelve ["And","bingo","was","his","name,","oh!""]  cancion.split(", ")  // devuelve ["And bingo was his name", "oh!"] |
| **.replace()**  Reemplaza una parte del string por otra, recibe la parte a reemplazar y la nueva que lo reemplazara. | let frase = "Aguante Phyton!"  frase.replace("Phyton","JS")//devuelve "Aguante JS!"  frase.replace("Phy","JS")//devuelve "Aguante JSton" |
| **concat()**  se usa para unir dos o más arrays o elementos. Este método no cambia los arrays existentes, sino que devuelve un nuevo array. | const letters = ['a', 'b', 'c'];  const numbers = [1, 2, 3];  letters.concat(numbers);  // result in ['a', 'b', 'c', 1, 2, 3] |

|  |  |
| --- | --- |
| **Array** | array = ["pablo", "leonel"]  // pablo esta en el indice 0  // leonel esta en el indice 1  array[0] // accedemos a "pablo" |
| **For** | for (let index = 0; index < array.length; index++) {      const element = array[index];    } |
| **while** | while (condition) {    } |
| **Do while.** | do {    } while (condition); |
| **For of**  Ejecuta un bloque de codigo para cada elemento de un objeto iterable, como string o array.  Puede declararse un let o const a la varible que sera el elemento del objeto iterado dependiendo de la necesidad de modificarlo o no.  Se puede usar para agregar a nodos del DOM clases o propiedades. | let iterable = [10, 20, 30]  for (let value of iterable) {      value += 1;      console.log(value);    }    // 11    // 21    // 31  let iterable = "boo";  for (let value of iterable) {    console.log(value);  }  // "b"  // "o"  // "o"  // Nota: Esto solo funcionará en plataformas  que tengan implementado  NodeList.prototype[Symbol.iterator]  let parrafosDentroDeArticules =  document.querySelectorAll("article > p");  for (let parrafo of parrafosDentroDeArticules) {    parrafo.classList.add("read");  } |

|  |  |
| --- | --- |
| **Funcion constructora**  Podemos construir objetos con una funcion constructora. El nombre de la funcion lleva la inicial en mayuscula. | function Auto(marca,modelo){      this.marca = marca      this.modelo = modelo  }  let miAuto = new Auto("ford","falcon")  //crea un nuevo objeto  let miOtroAuto = new Auto("chevrolet","corvette")  //crea un nuevo objeto |
| **JSON.parse()**  Convierte un texto con formato JSON al tipo de dato equivalente de JavaScript. | let datosJson = '{"club": "Independiente"}';  let datosConvertidos = JSON.parse(datosJson);  console.log(datosConvertidos)  // se vera en consola un objeto literal  // { club: 'Independiente' } |
| **JSON.stringify()**  Convierte un tipo de dato de javascript en un texto en formato JSON. | let objetoLiteral = { club: 'Independiente' }  let datosConvertidos = JSON.stringify(objetoLiteral)  console.log(datosConvertidos)  //se vera en consola los datos en un string tipo JSON  // {"club":"Independiente"} |
| **Module y require** | let serie = "Friends"  module.export = serie  // la variable serie se exportara y  // se podra acceder a ella desde otro  //archivo  EN OTRO ARCHIVO->  let serie = require("direccion del documento")  let serie = require("./serie/index") // por ejemplo |
| **Fs**  Paquete nativo que trae el **file system** que nos permite acceder a diferentes metodos del mismo, como escribir archivos o leerlos. Por ejemplo un JSON. | const fs = require("fs") |
| **fs.writeFileSync()**  Primer parametro corresponde al archivo donde queremos escribir (si no existe lo crea). El segundo parametro corresponde al contenido que queremos escribir en ese archivo destino. | const fs = require("fs")  fs.writeFileSync("estrenos-2020.txt", "Titanic 2")  SOLO PUEDE SER EN FORMATO STRING, USAR STRINGIFY ANTES SI ES OTRO DATO COMO UN OBJETO. |

|  |  |
| --- | --- |
| **fs.appendFileSync()**  a diferencia de writeFileSync, esto agrega al final lo que estamos escribiendo, mientras que el anterior sobreescribe todo. | const fs = require("fs")  fs.appendFileSync("estrenos-2021.txt","Titanic 3")  ESCRIBE ARCHIVOS EN FORMATO JSON PROBABLEMENTE, USAR METODO STRINGIFY DE JSON PARA CONVERTIRLO. |
| **FS.readFileSync()**  Nos permite leer un archivo, le damos como parametros la ruta y el enconding. | const fs = require("fs")  fs.readFileSync("sitcoms.txt", "utf-8")  LO TRAE EN FORMATO JSON, USAR METODO PARSE DE JSON. |

|  |  |
| --- | --- |
| **Arrow Functions**  Forma acotada de escribir funciones, no necesita un return si es en una sola linea, ni tampoco llaves. De ser mas extensa si los necesita.  A su vez si recibe solo 1 parametro no necesita los parentesis.  Por ultimo si no recibe parametros tenemos que aclararlo poniendo solo unos parentesis sin contenido. | let sumar = (a, b) => a + b;  let sumar = a => a + 2;  let esMultiplo = (a, b) => {      let resto = a % b      return resto == 0  }  const saludo = () => console.log("hola")  saludo() |
| **Callbacks**  Los callbacks van a ser funciones que se van a pasar como argumento de otras funciones.  Entonces podemos usar estructuras de ciertas funciones en beneficio de otras. | let suma = (num1, num2) => num1 + num2  let resta = (num1, num2) => num1 - num2  let multi = (num1, num2) => num1 \* num2  let division = (num1, num2) => num1 / num2  let calcular = (num1, num2, callback) => {      return callback(num1, num2)  }  console.log(calcular(5, 5, suma)) //10  console.log(calcular(5, 5, resta)) //0  console.log(calcular(5, 5, multi)) //25  console.log(calcular(5, 5, division)) //1 |

|  |  |  |
| --- | --- | --- |
| **Metodos de array** | | |
| **.push()**  Agrega lo que pusheamos dentro del array en la ultima posicion. Toma como parametro uno o mas elementos. | | let colores = ["rojo","naranja","azul"]  colores.push("violeta","verde")  console.log(coolores)  //["rojo","naranja","azul","violeta","verde"] |
| **.pop()**  Elimina el ultimo elemento de un array, no recibe parametros | | let colores = ["rojo","naranja","azul"]  colores.pop()  console.log(colores);  //["rojo","naranja"] |
| **.shift()**  Elmina el primer elemento de un array, no recibe parametro.  Devuelve el parametro elminado | | let colores = ["rojo","naranja","azul"]  let eliminado = colores.shift()  console.log(colores);  //["naranja","azul"]  console.log(eliminado)  //["rojo"] |
| **.unshift()**  Agrega uno o mas elementos al principio del array. | | let colores = ["rojo","naranja","azul"]  colores.unshift("verde")  console.log(colores);  // ["verde"","rojo","naranja","azul"] |
| **.join()**  Une los elementos de un array utilizando el separador especificado. Si no lo especificamos se utilizan comas. | | let colores = ["rojo","naranja","azul"]  separadosPorComa = colores.join()  console.log(separadosPorComa)  // "rojo,naranja,azul"  separadosPorGuion = colores.join(" - ")  //"rojo - naranja - azul" |
| **.indexOf()**  Busca en el array la posicion en el indice donde esta ubicado el elemento que le introducimos por parametro.  De no encontrarlo devuelve -1 | | let colores = ["rojo","naranja","azul"]  colores.indexOf("naranja")  // devuelve 1 |
| **.lastIndexOf()**  Similar a indexOf() pero empiez a buscar desde atrás hacia adelante, si no encuentra coincidencia devuelve -1 | | let colores = ["rojo","naranja","azul"]  colores.lastIndexOf("azul")  // devuelve 2 |
| **.includes()**  Busca si hay coincidencia y retorna un booleano (true or false) dependiendo si lo encuentra o no. | | let colores = ["rojo","naranja","azul"]  colores.includes("rojo") // true  colores.includes("violeta") // false |
| **Metodos avanzados de Arrays** | | | |
| **.map()**  Recibe una funcion como parametro (callback), recorre el array y devuelve un nuevo array modificado, con aquellas modificaciones programadas en nuestra funcion de callback. | array.map(function(elemento){      //definimos modificaciones que queremos      //aplicar sobre cada elemento del array.  });  let array = [2,4,6]  let elDoble = array.map(function(num){      return num \* 2  });  console.log(elDoble); //[4,8,12] | | |
| **.filter()**  Nos permite filtrar un array y traer lo que querramos según la condicion. Trae un nuevo array con lo que queremos. | let array = [2,4,6]  array.filter(function(elemento){      return edad > 18  })  // [22,30] | | |
| .forEach()  Itera sobre un array e implementa lo que impongamos sobre cada elemento. No devuelve nada. | var paises = ["argentina","cuba","peru"]  paises.forEach((item)=> console.log(item))  //"argentina"  //"cuba"  //"peru" | | |
| **.reduce(resultado,item,indice,array),inicializador**  Toma un array y los itera obteniendo un unico resultado, operando según indiquemos con cada elemento o item del array.  **IMPORTANTE:** Si vamos a usarlo sobre un array de objetos literales, tenemos que especificar el inicializador en 0, por que si no concatenara como textos las propiedades de los objetos literales. | | | |
| const numeros = [1, 2, 3, 4, 2];  const mayor = numeros.reduce(function (resultado,item, indice, array) {      if (indice < array.length) {          if (resultado < item) resultado = item;      }      return resultado;  });  console.log("Mayor: " + mayor); //Mayor: 4  -----------------------------------------------------------------------   const electronicos = [      { nombre: 'iPad Pro 12"', precioEnUsd: 2200, cantVendidos: 15 },      { nombre: "Macbook Air", precioEnUsd: 1500, cantVendidos: 20 },      { nombre: "Macbook Pro", precioEnUsd: 3000, cantVendidos: 7 },  ];  function gananciaTotal() {      return electronicos.reduce((acum, current) => acum + current.precioEnUsd \* current.cantVendidos, 0);  }  console.log("Ganancia total: ") //Gananacia total:  console.log("$" + gananciaTotal()) //$84000 | | | |

|  |
| --- |
| **.sort()**  Este metodo ordena los elementos de un array en base a su valor en Unicode, si no especificamos su ordenamiento transforma el valor a string.  Por ejemplo mayusculas y minusculas tienen diferente valor , primero ordenara en una lista las mayusculas y luego las minusculas, a menos que le apliquemos un lower o upper case cuando comparamos.  Si solamente necesitamos ordenar palabras con letras en minusculas o mayusculas podemos simplemente hacer un .sort() sin ningun calculo adicional en un if como en el proximo ejemplo. |
| var items = [      {name: 'Edward',value: 21},      {name: 'Sharpe',value: 37},      {name: 'And',value: 45},      {name: 'anna',value: 32},      {name: 'The',value: -12 },      {name: 'Magnetic',value: 13},      {name: 'Zeros',value: 37}  ];  ***//Comparar nombres***  const sortPalabras = items.sort(function (elementoA, elementoB) {      if (elementoA.name > elementoB.name)          return 1;   //el 1 indicara que el elemento ira primero      }      if (elementoA.name < elementoB.name) {          return -1;  //el 2 indicara que el elemento ira despues que el B      }else {return 0}  });  console.log(sortPalabras)  ***//Si tenemos mayusculas y minusculas***  const sortPalabrasMayusMinus=items.sort(function(elementoA,elementoB) {      if (elementoA.name.toLowerCase() > elementoB.name.toLowerCase()) {         return 1;   //el 1 indicara que el elemento ira primero      }      if (elementoA.name.toLowerCase() < elementoB.name.toLowerCase()) {          return -1;  //el 2 indicara que el elemento ira despues que el B      }else {return 0}  });  ***//comparar solo numeros***  const numeros4 = [10, 3, 4, 52, 6, 7];  const ordenado = numeros4.sort((elementoA, elementoB) => {      return elementoA - elementoB}) |

|  |
| --- |
| **.Splice(inicio, cant,item1,item2)**  Remueve o agrega elementos de un array. Recibe 3 parametros.  1) Inicio(donde empieza el camio)  2) cant(opcional) indica cantidad de elementso a elminar, si se omite o es mayor que (array.length – inicio) se elimnaran todos desde el inicio.  3)items(opcional) indica los elementos que se agregaran al array desde inicio. Si se omite splice solo elimina. |
| let numeros = [3,4,5,6,7];  numeros.splice(0,0,2) //inicia en 0, remueve 0, agrega el valor 2 en  donde inicia (0)  console.log(numeros)//[2,3,4,5,6,7]  let numeros = [3,4,5,6,7];  numeros.splice(0,2) //inicia en indice 1 y elimina 2 elementos.  console.log(numeros)//[2,5,6,7] |
| **.Find(elemento, index,array)**  devuelve el valor del primer elemento de un array que cumpla con funcion especificada, necesita declarar el elemento que sera evaluado,cada vez que recorra el array y opcionalmente puede indicarse el index donde se comenzara a buscar (esto ultimo del indice no estoy seguro 100% si para eso es el index). |
| const criptos = [      {nombre: 'Bitcoin', simbolo: 'BTC'},      {nombre: 'Ethereum', simbolo: 'ETH'},      {nombre: 'Cardano', simbolo: 'ADA'}      ];    const encontrarPalabra = nombreBuscado => criptos.find((elemento) => {      return elemento.nombre === nombreBuscado})  console.log(encontrarPalabra("Ethereum")) |

|  |  |
| --- | --- |
| **Front 2** | |
| **Variantes de console.log()**  **.error( )** → Escribe un error en consola.  **.warn( )** → Escribe una advertencia en consola.  **.table( )** → Escribe una tabla en la consola. | console.error("error")  console.warn("advertencia")  console.table(["tabla","tabla2"]) |
| **Alert(“mensaje”)**  Aparece un cartel en el navegador con el mensaje escrito. |  |
| **<Script></Script>**  Etiqueta para poder escribir JS en el documento html, si agregamos un SRC podemos referenciar un documento .js para realizar los script en otro documento. | <body>      <script src="cheatsheet.js"></script>  </body> |
| **Prompt(“cadena de texto”)**  cartel en navegador con texto un espacio para escribir. Convertira el texto pedido en un String. (SIEMPRE ES UN STRING).  Hacer un parseInt o float para convertir en un dato numerico. | let dato = prompt("ingrese un numero");  let dato2 = prompt("ingrese otro numero")  dato = parseInt(dato)  dato2 = parseInt(dato2)  console.log(dato+dato2) // sumara los numeros  -----------------------------------------------  let dato = prompt("ingrese un numero");  let dato2 = prompt("ingrese otro numero")  console.log(dato+dato2)  // concatena y muestra un string de los datos. |
| **Confirm(“cadena de texto”)**  Aparece un cartel con opcion de confirmar o cancelar y devuelve un valor booleano. | let dato3 = confirm("¿Desea un trato personalizado?")  console.log(dato3)  //true si confirma y false si cancela |

|  |  |  |
| --- | --- | --- |
| **Math.random()**  **Math.round()**  **Math.max()**  **Math.ceil()**  **Math.floor()** | Retorna un punto flotante aleatorio entre [0,1)  Retorna valor d’ un numero redondeado al entero mas cercano  Devuelve el mayor de cero o mas numeros.  Devuelve redondeo hacia arriba  Devuelve redondeo hacia abajo | |
| **For in**  Itera un objeto devolviendo sus propiedades, mas no sus valores. Declarando un elemento y diciendole que lo itere dentro del objeto, traera los valores de este. | let personaje = {      nombre: "Luke",      apellido: "Skywalker",      edad: 25  }  for (const iteradora in personaje) {      console.log(iteradora)  }  //"nombre"  //"apellido"  //"edad"  ------------------------------------------  let personaje = {      nombre: "Luke",      apellido: "Skywalker",      edad: 25  }  for (const caracteristica in personaje) {      console.log(personaje[caracteristica])  }  //"Luke"  //"Skywalker"  // 25  // en este ejemplo caracteristica representara como en el ejemplo anterior, el nombre de la propiedad. Y si nosotros llamamos a personaje[nombre] obtendremos el valor de la propiedad nombre por ejemplo. De esta forma se va a iterar y la constante que representa el nombre de cada propiedad ira cambiando en el indice buscado. |
| **For of**  Itera sobre un array dando en cada iteracion la variable iteradora un valor como cada uno de los elementos del array. | let series = ["the mandalorian","the office","brooklyn 99"]  for (const iteradora of series) {      console.log(iteradora);  }  //the mandalorian  //the office  //brooklyn 99 |

|  |  |
| --- | --- |
| **Document Object Model**  Document Object Model o DOM es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML, ​ un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.--------------- | |
| **Window**  Nos permite interactuar con la ventana del navegaodor  Es nativo de JS.  Nos da una interfaz para poder interpretar el html. |  |
| **Document**  Nos permite interactuar con le contenido de la pagina.  Nos da un conjunto de atributos para poder modificar y opearar con lo que ve el usuario.  Es nativo de JS.  Por ejemplo podes usar css desde el document, es decir que podemos alterar el css con JS. |  |

|  |  |
| --- | --- |
| **Query Selectores** | |
| **querySelector()**  Selector que recibe un string que indica el selector CSS del elemento del DOM que estamos buscando. | document.querySelector(".title")  //selecciona el primer elemento del dom  //con una clase "title" |
| **querySelectorAll()**  Selector que recibe un string que indica todos los selector CSS del mismo tipo del elemento del DOM que estamos buscando seleccionar. | document.querySelectorAll(".name")  //selecciona todos los selectores con clase “name”  document.querySelectorAll("div")  //selecciona todos los div  //Trae un array con los elementos. |
| **getElementById()**  selecciona unicamente el elemento con el id introducido. | document.getElementById("marca")  //selecciona el elemento con el id "marca"  Tambien podemos buscar por id con los selectores anteriores, pero deberiamos buscarlos con (“#marca”) por ejemplo. |
| **.innerHTML**  Una vez seleccionada una etiqueta con por ejemplo un querySelector, innerHTML nos permite modificar leer o modificar el contenido de esta etiqueta.  Reemplaza el contenido de la etiqueta, si quneremos agregar informacion deberiamos concatenar.  Es una propiedad y no un metodo. | let titulo = document.querySelector("h1")  //trae la etiqueta  console.log(titulo.innerHTML)  //muestra el contenido de texto  titulo.innerHTML = "Soy un contenido nuevo desde JS"  //modifica el contenido  titulo.innerHTML ="Soy un <strong>contenido</strong>   nuevo desde JS"  //modifica el contenido y permite agregar etiquetas, por que modifica el DOM, a diferencia de .innerText que solo puede agregar texto plano. |
| **.InnerText**  Es similar a .innerHTML, salvo que trabaja con texto plano, por ejemplo no podemos agregar etiquetas dentro del html con esta propiedad. | titulo.innerText = "Soy un <strong>contenido  </strong> nuevo desde JS"  //Soy un <strong>contenido</strong> nuevo  desde JS |

|  |
| --- |
| **Template string // `${ }`**  Son cadenas literales que habilitan el uso de expresiones incrustadas. Con ellas, es posible utilizar cadenas de caracteres de más de una línea, y funcionalidades de interpolación de cadenas de caracteres.  Deben ir entre comillas simples invertidas (` `)  De otra forma si usamos comillas dobles o comillas simples normales js interpretara que todo es un string. |
| Utilizando cadenas de caracteres normales, sería necesario utilizar la siguiente sintaxes para producir cadenas de más de una línea:  console.log('línea 1 de cadena de texto\n' +  '\línea 2 de cadena de texto');  // "línea 1 de cadena de texto  // línea 2 de cadena de texto"  Utilizando plantillas literales, se puede obtener el mismo resultado de la siguiente forma  console.log(`línea 1 de la cadena de texto  línea 2 de la cadena de texto`);  // "línea 1 de la cadena de texto  // línea 2 de la cadena de texto"  ----------------------------------------------------------------------  Para insertar expresiones dentro de cadenas de caracteres normales, se utilizaría la siguiente sintaxis:  let a = 5;  let b = 10;  console.log('Quince es ' + (a + b) + ' y\nno ' +   (2 \* a + b) + '.');  // "Quince es 15 y  // no 20."  Ahora, con las plantillas literales, se pueden utilizar sus nuevas capacidades (es decir, insertar expresiones con ${ } e incluir caracteres  de fin de linea literales dentro de la cadena) para  simplificar la sintaxis:  let a = 5;  let b = 10;  console.log(`Quince es ${a + b} y  no ${2 \* a + b}.`);  // "Quince es 15 y  // no 20." |

|  |  |  |
| --- | --- | --- |
| **.style**  Permite modificar estilos de css mediante JS  De ser una propiedad que en css se escribe con un guion medio para separar dos palabras, en JS se usa camelCase.  Es una propiedad de document. | | titulo.style.background = "cyan";    element.style.color = "red";  //Otros ejemplos:  element.style.fontSize = "15px";  element.style.backgroundColor = "#FFFFFF"; |
| **ClassList** | | |
| **.classList**  Nos trae las clases que tiene un objeto del dom. Nos lo trae en forma de array con cada uno de los objetos, como mencionamos antes, que tengan esa clase.  Tiene propiedades que permiten agregar, quitar y otras acciones con etiquetas del DOM. | let parrafo = document.querySelector("p")  console.log(parrafo.classList) | |
| **classList.add()**  nos permite agregar una clase nueva al elemento que tengamos seleccionado. | let cita = document.querySelector(".cita")  cita.classList.add("italicas")  //esto agregara la clase italicas al elemento  //html que le indiquemos, ene ste caso habiamos  //campturado previamente la primer etiqueta del  //html que tuviese la clase "cita" | |
| **classList.remove()**  nos permite quitarle una clase existente al elemento que tenemos seleccionado. | let cita = document.querySelector(".cita")  cita.classList.remove("cita")  //esto removera del elemento capturado la  //clase "cita | |
| **classList.toggle()**  Revisa si existe una clase en el elemento seleccionado. De ser asi, la remueve, de lo contrario si la clase no existe, la agrega. | let cita = document.querySelector("p")  cita.classList.toggle("cita")  //esto removera o aplicara la clase cita  // del parrafo que esta seleccionado. | |
| **ClassList.contains()**  Nos permite preguntar si un elemento tiene una clase determinada. Devuelve un valor booleano. | let cita = document.querySelector(".italicas")  cita.classList.contains("cita") //false  cita.classList.contains("italicas") //true | |
|  | | |

|  |  |
| --- | --- |
| **Metodos de nodos en el DOM.** | |
| **createElement()**  crea un nodo del tipo elemento según el nombre de la etiqueta de HTML que le indiquemos. | var botonVerMas =  document.createElement("button")  //crea y almacena un boton. |
| **createTextNode()**  crea un nodo de texto explicitado entre comillas. No se visualiza hasta asignarlo a un elemento existente del DOM | var botonTexto =  document.createTextNode("Ver mas")  //crea un nodo de texto que contiene un string  // "Ver mas" |
| **AppendChild()**  Adhiere dentrro del dom un elemento hijo a un elemento padre. Si el elemento padre ya existia en el documento, cambia su posicion hacia el otro elemento padre indicado. Si no existe, lo creamos con el metodo 1. | botonVerMas.appendChild(botonTexto)  //introduce el nodo de texto en el boton  document.body.appendChild(botonVerMas)  //introducimos el boton con el nodo de  //texto incluido en el, dentro del body |

|  |  |
| --- | --- |
| **Atributos dinamicos**  Un atributo es un modificador de un elemento. Es una palabra especial que nos permite controlar un determinado comportamiento en nuestra etiqueta. Como por ejemplo un href dentro de una etiqueta <a>.  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()**  permite preguntar si un elemento tiene cierto atributo, devuelve un valor booleano. | let elemento = document.querySelector("img")  //seleccionamos un elemento del HTML.  elemento.hasAttribute("src") //true  //consultamos si tiene un atributo src |
| **getAttribute()**  consulta el valor del atributo consultado.  Retorna valor si existe, de lo contrario devuelve un texto vacio. | let elemento = document.querySelector("img")  //seleccionamos un elemento del HTML.  elemento.getAttribute("src")  //pedimos el valor del atributo  //imagen\_hat.jpg |
| **removeAttribute()**  borra por completo el atributo y sus valores del elemento. Si no lo encuentra, no hace nada. | let elemento = document.querySelector("img")  //seleccionamos un elemento del HTML.  elemento.removeAttribute("src")  //pedimos el valor del atributo  //remueve el atributo src. |
| **setAttribute()**  permite agregar un atributo con su respectivo valor al elemento seleccionado.  Recibe el nombre del atributo y un valor para el mismo, no retorna ningun valor. | let elemento = document.querySelector("img")  //seleccionamos un elemento del HTML.  elemento.setAttribute("src","image\_hat.jpg")  //pedimos el valor del atributo  //remueve el atributo src. |

|  |  |
| --- | --- |
| **Eventos**  Un evento es una accion que transcurre en el navegador o que es ejecutado por el usuario.  Independientemente de si usamos “On” o “addEventListener”, tenemos que tener en cuenta otros factores. this: esto hace referencia al elemento puntual donde se produce el evento. preventDefault(): evitamos eventos predeterminados/nativos. | |
| **On + algo**  window.onload = function(){      console.log("hola")  }  //solo puedo tener un evento  //si escribo el evento otra vez  //este se sobreescribira o pisara  //y se ejecutara el segundo. | **addEventListener**  window.addEventListener("load",  function(){  })  //con addEventListener, sucederan  //todas en simultaneo. |
| **Onload**  Permite que todo el script se ejecute cuando se haya cargado por completo el objeto **document** dentro del objeto **window.** Se usa para prevenir errores que pueden ocurrir si el documento no esta cargado al momento de ejecucion del script. | window.onload = function(){    console.log("documento listo")  } |
| **Onclick**  Este eveto nos permite ejecutar una accion cuando se haga click sobre el elemento al cual le aplicamos la propiedad. | btn.onclick = function(){      console.log("hiciste clic")  } |
| **addEventListener(“load”,function(){})**  Igual a Onload, pero puede suceder multiples veces y no se iran pisando sucesivamente. | window.addEventListener("load",  function(){    }) |
| **addEventListener(“click”,function(){})**  Igual a Onclick, pero puede suceder multiples veces y no se iran pisando sucesivamente. | window.addEventListener("click",  function(){    }) |

|  |
| --- |
| **preventDefault()**  Permite evitar ejecucion de evento predeterminado o nativo, del elemento al que lo apliquemos.  **Ej:** prevenir que una etiqueta **“a”** se comporte de manera nativa y haga en cambio, otra accion. |
| let hipervinculo = document.querySelector("a")  //atrapamos elemento  hipervinculo.addEventListener("click", function(event){                              //atrapamos el evento        console.log("hiciste click")        event.preventDefault();      //prevenimos la accion nativa de redireccionamiento  }) |
| **Eventos mas usados**   |  |  |  | | --- | --- | --- | | On | addEventListener | evento | | **onclick** | **click** | Cuando el usuario hace click. | | **onblclick** | **blclick** | cuando el usuario hace doble click. | | **onmouseover** | **mouseover** | cuando el mouse se mueve sobre el elemento. | | **onmousemove** | **mousemove** | cuando se mueve el mouse. | | **onscroll** | **scroll** | cuando se hace scroll | | **onkeydown** | **keydown** | cuando se aprieta una tecla | | **onload** | **load** | cuando se carga la pagina | | **onsubmit** | **submit** | cuando se envia un formulario. | |
| **removeEventListener (event, listener)**  If an event is created and there is some activity from the user but you don’t want the element to react to that particular event for some purpose, so to do that we have removeEventListener() method in JavaScript.The removeEventListener() is an inbuilt function in JavaScript which removes an event handler from an element for a attached event.for example, if a button is disabled after one click you can use removeEventListener() to remove a click event listener.  materiasBtn.addEventListener("click",  recorrerListadoYRenderizarTarjetas);  materiasBtn.removeEventListener("click",  recorrerListadoYRenderizarTarjetas  //remueve el evento click creado anteriormente |

|  |  |
| --- | --- |
| **Eventos de teclado** | |
| **onkeydown**  Es lanzado cuando una tecla es presionada, a diferencia de **keypress** y **keydown**, es lanzado para las teclas que pruducen un carácter y tambien para las que no lo producen. | let miImput=document.querySelector("#miInput")  miImput.onkeydown = function(event){  alert("Se presiono la tecla: "+event.onkeydown)      } |
| **onkeyup**  Este evento se dispara unicamente cuando la tecla se suelta. | let miImput = document.querySelector("#miInput")  miImput.onkeyup = function (event) {      alert("Se presiono la tecla: " + event.key)  } |
| **onkeypress**  **(Es el que se usa.)**  Se dispara al finalizar el recorrido de presion y liberacion de la tecla. | let miImput = document.querySelector("#miInput")  miImput.onkeypress = function (event) {      alert("Se presiono la tecla: " + event.key)  } |

|  |  |
| --- | --- |
| **SetTimeout SetInternal, clearInterval** | |
| **setInterval(() => {} , intervalo)**  SetInterval nos permite ejecutar una funcion repetidas veces, el intervalo de tiempo se mide en milisegundos y va como segundo parametro del setInterval | var numero = 0  var div = document.getElementById("contador")  let cont = setInterval(function(){      div.innerHTML = numero      numero++  }, 1000) //intervalo cada 1 segundo  // esta funcion genera un cronometro de //segundos dentro de un div capturado.  setInterval(() => {      console.log(`¡hola, saludo una      vez por segundo!`)  }, 1000); |
| **SetTimeOut()**  Permite retrasar la ejecucion de la funcion que le introduscamos (callback) como primer parametro y el tiempo definido en milisegundos como segundo parametro. | let cronometro = setInterval(() => {         console.log(`Segundo ${contador}`)      contador++  },1000)  let detenedor = () => clearInterval(cronometro)  setTimeout(detenedor,10000) |
| **ClearInterval(variable)**  Nos permite detener la ejecucion de un setInterval, introducimos en los parametros el nombre de la variable o la indicacion de donde esta alojada. | let contador = setInterval(() => {      console.log(`¡hola, saludo una      vez por segundo!`)  }, 1000);  function pararContador(){      clearInterval(contador)  }  //paramos el contador llamando a esta funcion  //no necesariamente tenemos que armar una funcion para usarlo, puede estar suelto, en este caso habria que invocar la funcion para usarlo. |

|  |
| --- |
| **Date**  **Date nos permite trabajar con fechas y horas, tiene muchas propiedades/metodos.**  **Si no proporciona argumentos, el constructor crea un objeto Date con la hora y fecha de hoy según la hora local.**  **Si proporciona algunos argumentos, debe proporcionar al menos 2 argumentos. Los argumentos vacíos se establecen a 0 (ó 1 si falta el día).** |
| horaA = new Date();  // Sentencias que realizan alguna acción.  horaB = new Date();  diferenciaHoras = horaB - horaA;  // diferenciaHoras nos dira el tiempo transcurrido  Date.now()     /\*  Devuelve el valor numérico correspondiente a la hora actual. \*/  Date.parse()     /\*  Transforma una cadena que representa una fecha, y devuelve el número de milisegundos desde el 1 de enero de 1970, hora local 00:00:00. \*/  Date.UTC()      /\* Acepta los mismos parámetros que la forma más larga del constructor, y devuelve el número de milisegundos en un objeto Date desde el 1 de enero de 1970, hora universal 00:00:00.  \*/  // usando métodos estáticos  var inicio = Date.now();  // el evento cuyo tiempo ha transcurrido aquí:  hacerAlgoPorAlgunTiempo();  var fin = Date.now();  var transcurso = fin - inicio; // tiempo en milisegundos |

|  |  |  |
| --- | --- | --- |
| **Formularios**  **Html** | | |
| **<form></form>**  Nos permite crear formularios dentro de ella. sus atributos son action y method. | | <form action="" method="">  /\* contenido del formulario \*/  </form> |
| **Atributo action**  Define la ruta donde va a ser procesada la informacion capturada en los campos del formulario. | | <form name="formA" action="/cgi-bin/test"  method="POST"></form> |
| **Atributo method**  Define como se enviara la informacion. | | **POST:** Corresponde al [método POST](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5) HTTP ; los datos del formulario son incluidos en el cuerpo del formulario y son enviados al servidor.  **GET:** Corresponde al [método GET](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3) HTTP; los datos del formulario son adjuntados a la URI del atributo  action , con un '?' como separador,  y la URI resultante es enviada al servidor. Use este método cuando el formulario no tiene efectos secundarios y contiene solo caracteres ASCII. |
| **<label></label>**  Permite poner texto indicando de forma clara y precisa la informacion que se espera recibir. | | <label for="Name">Click me</label> |
| **<textarea></textarea>**  Sirve para generar un campo donde el usuario puede dejar un mensaje, son las tipicos box de comentarios.  Puede tener atributos diversos para configurar. | | <textarea name="textarea" rows="10" cols="50">Write something here</textarea> |
| **<select>**  **<option></option>**  **<option></option>**  **<option></option>**  **</select>**  Select genera un cajita con opciones qu queremos mostrar, por ejemplo paises. option nos permite generar la opcion dentro del select. | | <select>      <option>Argentina</option>      <option>Brasil</option>  </select> |
| **<button></button>**  Permite generar un boton que enviara la informacion de los campos o borrara los campos dependiendo del atributo type. | /\* type="submit"   enviara los datos  a la ruta especificada en el atributo  **ACTION** de la etiqueta FORM \*/  <button type="submit">Enviar</button>;  /\* type="reset"    reseteara los campos \*/  <button type="reset">Borrar</button>; | | |
| **<fieldset></fieldset>**  Nos permite agrupar los campos para organizar un formulario.  Los campos seran envueltos en un recuadro por default. | <form>   <fieldset>;    <legend>Información Personal</legend>    Nombre: <input name='nombre' type='text' tabindex='1'>    Apellidos: <input name='apellidos' type='text' tabindex='2'>   </fieldset>   <fieldset>    <legend>edad</legend>    <input type='checkbox' tabindex='20'              name='edad' value='20-39' > 20-39    <input type='checkbox' tabindex='21'              name='edad' value='40-59' > 40-59    <input type='checkbox' tabindex='22'              name='edad' value='60-79' > 60-79   </fieldset>  </form> | | |
| **<legend></legend>**  Nos permite crear un titulo para un grupo de campos de **fieldset** |
|  | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Atributos universales**   |  |  | | --- | --- | | Atributo | Descripcion | | **Name = ””** | Permite identificar a cada campo como un elemento en particular | | **Value = “”** | Se introduce en los campos donde no se permite introduccion de texto por parte de los usuarios, le da un valor a la seleccion, por ejemplo en los option.  Es el valor que capturaremos mediante un **document.queryselector(selector).value** | | **Required = “”** | Puede usarse en cualquier campo donde se introduzca texto, en caso de que no se rellene, se indicara una alerta invitando a rellenarlo. | | **Placeholder = “”** | Permite poner texto en el campo a rellenar, se puede usar como guia para indicar que informacion o en que formato rellenar el campo. | |
| **Ejemplo de un form** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **<input>**  Etiqueta multifuncion, tiene diferentes atributos.  Se usa para crear controles interactivos para formularios basados en la web con el fin de recibir datos del usuario.  <input type="submit" value="Submit"></input>  Adicionalmente, necesita un id que tiene que coincidir con el mismo atributo for del label para hacer un enlace.   |  |  | | --- | --- | | TYPE | Descripcion | | **button** | Botón sin un comportamiento específico | | **checkbox** | Casilla de selección. Se debe usar el atributo **value** para definir el valor que se enviará por este elemento. Se usa el atributo **checked** para indicar si el elemento está seleccionado. También se puede usar el atributo **indeterminate** (el cual solo se puede establecer programaticamente) para indicar que la casilla está en un estado indeterminado (en la mayoría de las plataformas, se dibuja una línea horizontal a través de la casilla). | | **radio** | Botón radio. Se debe usar el atributo **value** para definir el valor que se enviará por este elemento. Se usa el atributo **checked** para indicar si el elemento está seleccionado de forma predeterminada. Los botones radio que tengan el mismo valor para su atributo **name** están dentro del mismo "grupo de botones radio". Solo un botón radio dentro de un grupo puede ser seleccionado a la vez. | | **color** | Control para espicificar un color. Una interfaz de selección de color no requiere más funcionalidad que la de aceptar colores simples como texto | | **Date** | Control para introducir una fecha (año, mes y día, sin tiempo). | | **datetime** | Control para introducir fecha y hora, sin zona horaria específica. | | **email** | Campo para introducir una dirección de correo electrónico. El valor introducido se valida para que contenga una cadena vacía o una dirección de correo válida antes de enviarse. Las pseudo-clases :valid y :invalid son aplicadas segun corresponda. | | **file** | Control que permite al usuario seleccionar un archivo. Se puede usar el atributo **accept** para definir los tipos de archivo que el control podrá seleccionar. | | **hidden** | Control que no es mostrado en pantalla, pero cuyo valor es enviado al servidor. | | **image** | Botón de envío de formulario con gráfico. Se debe usar el atributo **src** para definir el origen de la imagen y el atributo **alt** para definir un texto alternativo. Se puede usar los atributos **height** y **width** para definir el tamaño de la imagen en píxeles. | | **number** | Control para introducir un número de punto flotante. | | **password** | Control de línea simple cuyo valor permanece oculto. Se puede usar el atributo **maxlength** para especificar la longitud máxima del valor que se puede introducir. | | **reset** | Botón que restaura los contenidos de un formulario a sus valores predeterminados. | | **submit** | Botón que envía el formulario. | | **tel** | Control para introducir un número telefónico. Los saltos de línea son eliminados automáticamente del valor introducido, pero no hay otra sintaxis forzada. Se pueden usar atributos como pattern y maxlength para restringir los valores introducidos en este control. Las pseudo-clases CSS :valid y :invalid son aplicadas segun corresponda. | | **text** | Campo de texto de línea simple. Los saltos de línea son eliminados automáticamente del valor introducido. | | **time** | Control para introducir un valor de tiempo sin zona horaria específica. | | **url** | Campo para editar una URL. El valor introducido se valida para que contenga una cadena vacía o una ruta URL absoluta antes de enviarse. Los saltos de línea y espacios en blanco al principio o al final del valor son eliminados automáticamente. Se pueden usar atributos como pattern y maxlength para restringir los valores introducidos en el control. Las pseudo-clases :valid y :invalid son aplicadas segun corresponda. | | **Autocomplete** | Podemos darle valor “off” para que no se autocomplete | |

|  |  |
| --- | --- |
| **Mas eventos para formularios.** | |
| **Blur**  Detecta que el usuario quito el cursor de un input del formulario. | addEventListener("blur",function(){  /\* codigio de la funcion \*/  }) |
| **Focus**  Detecta que el usuario situo el cursor sobre un input del formulario | addEventListener("focus",function(){  /\* codigio de la funcion \*/  }) |
| **Change**  Sirve para detectar cambios en el valor de un input. | addEventListener("change",function(){      /\* codigio de la funcion \*/  }) |