Frontend Cheatsheet

Javascript y un poco de html

Pablo Leonel Cristaldo

**Indice**

**Ctrl+click para usar hipervinculo.**

[**General**](#General)

* Variables
* Function
* Operadores Logicos
* If else
* If ternario
* Switch
* isNan()

[**Metodos de String**](#MetodosDeString)

* indexOf()
* slice()
* trim()
* split()
* replace()
* concat()
* charAt()

[**General II**](#General2)

* Array
* For
* While
* Do while
* For of
* Funcion Constructora
* JSON.parse()
* JSON.stringify()
* Module y Require
* Fs File System
* writeFileSync()
* appendFileSync()
* Arrow Function
* Callbacks

[**Metodos de array**](#MetodosDeArray)

* Push()
* Pop()
* Shift()
* Unshift()
* Join()
* IndexOf()
* LastIndexOf()
* Includes()
* Map()
* Filter()
* ForEach()
* Reduce()
* Sort()
* Splice()
* Find()

[**Front 2**](#Front2)

* Variantes de console.log()
* Alert
* <Script>
* Prompt()
* Confirm()
* Math.algo()
* For in
* For of

[**Document Object model (DOM)**](#DocumentObjectModel)

* Window
* Document
* querySelector()
* querySelectorAll()
* getElementById()
* innerHTML
* innerText
* Template string
* Style
* **classList**
* add()
* remove()
* toggle()
* contaions()

[**Metodos de nodos en el DOM**](#MetodosDeNodosEnElDom)

* createElement()
* createTextNode()
* AppendChild()

[**Atributos dinamicos**](#AtributosDinamicos)

* hastAttribute()
* getAttribute()
* removeAttribute()
* setAttribute()

[**Atributos generales**](#atributosBasicos)

* Atributos basicos
* Atributos de internacionalizacion
* Atributos de foco
* Atributos de evento

[**Eventos**](#Eventos)

* Onload()
* Onclick()
* AddEventListener()
* disabled
* preventDefault()
* Eventos mas usados
* removeEventListener()

[**Eventos de teclado**](#EventosDeTeclado)

* Onkeydown
* Onkeyup
* Onkeypress

[**Intervalos**](#Intervalos)

* setInterval()
* setTimeOut()
* clearInterval()
* Date

[**Formularios**](#Forumlarios)

* Form
* Action
* Method
* Label
* Textarea
* Select
* Option
* Button
* Fieldset
* Legend
* Atributos Universales
* Input
* Type

[**Mas eventos para formulario**](#Forumlarios)

* Blur
* Focus
* Change

[**Object location**](#ObjectLocation)

* Location.href
* Location.search
* Location.reload
* Location.replace()
* New URLSearchParams(location.search)
* Parametros.has(“”)
* Parametros.get(“”)
* localStorage
* sessionStorage
* setItem
* getItem
* removeItem
* clear

[**Expresiones regulares**](#expresionesRegulares)

* Crear un expresion regular
* Métodos que usan expresiones regulares

[**JSON**](#JSON)

* Parse
* Stringify

[**Ajax**](#Ajax)

* Ejemplos practicos de AJAX
* HTTP: peticion – respuesta
* Concepto de asincronismo
* Concurrencia y paralelismo

[**Promesas**](#promesa)

* Garantias
* Encadenamiento

[**APIs**](#APIs)

* REST
* Arquitectura cliente-servidor
* Recursos uniformes
* Formatos
* JSON
* RAW
* TEXT
* URL-encoded
* Resumen
* Ejemplo de un api
* AJAX Fetch
* Resolvemos la promesa
* Obtenemos los datos
* Atrapamos los errores
* Manejo de errores.

**[General](#_top)**

|  |  |  |  |
| --- | --- | --- | --- |
| **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] | |
| **.charAt()**  **Permite obtener una letra de un string indicando el indice donde se encuentra.** | let palabra = "Numero"  palabra.charAt(0) //obtengo N | |
| **General II** |
| **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.exports = 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. |

**Atributos básicos**

Los siguiente cuatro atributos básicos se pueden aplicar prácticamente a todas las etiquetas HTML:

| **Atributo** | **Descripción** |
| --- | --- |
| id = "texto" | Establece un identificador único a cada elemento dentro de una página HTML |
| class = "texto" | Establece la clase CSS que se aplica a los estilos del elemento |
| style = "texto" | Establece de forma directa los estilos CSS de un elemento |
| title = texto" | Establece el título a un elemento (mejora la accesibilidad y los navegadores lo muestran cuando el usuario pasa el ratón por encima del elemento) |

**Atributos para internacionalización**

Estos atributos son útiles para aquellas páginas que muestran sus contenidos en varios idiomas y para aquellas que quieren indicar de forma explítica el idioma de sus contenidos:

| **Atributo** | **Descripción** |
| --- | --- |
| lang = "codigo de idioma" | Indica el idioma del elemento mediante un código predefinido |
| xml:lang = "codigo de idioma" | Indica el idioma del elemento mediante un código predefinido |
| dir | Indica la dirección del texto (útil para los idiomas que escriben de derecha a izquierda) |

**Atributos de foco**

Cuando el usuario selecciona un elemento en una aplicación, se dice que *"el elemento tiene el foco del programa"*. Si por ejemplo un usuario pincha con su ratón sobre un cuadro de texto y comienza a escribir, ese cuadro de texto tiene el foco del programa, llamado *"focus"* en inglés. Si el usuario selecciona después otro elemento, el elemento original pierde el foco y el nuevo elemento es el que tiene el foco del programa.

Los elementos de las páginas web también pueden obtener el foco de la aplicación (en este caso, el foco del navegador) y HTML define algunos atributos específicos para controlar cómo se seleccionan los elementos.

| **Atributo** | **Descripción** |
| --- | --- |
| accesskey = "letra" | Establece una tecla de acceso rápido a un elemento HTML |
| tabindex = "numero" | Establece la posición del elemento en el orden de tabulación de la página. Su valor debe estar comprendido entre 0 y 32.767 |
| onfocus, onblur | Controlan los eventos JavaScript que se ejecutan cuando el elemento obtiene o pierde el foco |

**Atributos de eventos**

Estos atributos sólo se utilizan en las páginas web que incluyen código JavaScript para realizar acciones dinámicas sobre los elementos de la página. Cada vez que el usuario pulsa una tecla, mueve su ratón o pulsa cualquier botón del ratón, se produce un evento dentro del navegador. Utilizando JavaScript y los siguientes atributos, es posible responder de forma adecuada a cada evento.

| **Atributo** | **Descripción** | **Elementos que pueden usarlo** |
| --- | --- | --- |
| onblur | Deseleccionar el elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón *"sale"* del elemento (pasa por encima de otro elemento) | Todos los elementos |
| onmouseover | El ratón *"entra"* en el elemento (pasa por encima del elemento) | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |
| onreset | Inicializar el formulario (borrar todos sus datos) | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

|  |  |
| --- | --- |
| **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(){    }) |
| **.disabled =boolean**  Podemos agregar un .disabled para indicar si el elemento seleccionado esta o no habilitado. | let button = document.querySelector("button")  button.disabled = true; //lo habilita  button.disabled = true; //lo deshabilita |

|  |
| --- |
| **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. | | **Pattern = “”**  **Minlength=””**  **Maxlength=””** |  | |
| **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 \*/  }) |

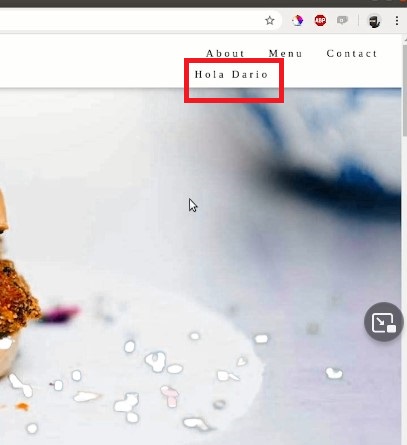
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Object Location**  Acceder al objeto es tan sencillo como escribir location en nuestro archivo de JavaScript. Con objetc location, vienen una serie de metodos y atributos. | | | | |
| **location.href**  nos devuelve al url completa.  Permite cambiar la url y dirigirnos al lugar que le marquemos. |  | | | |
| **location.search**  nos devuelve el query string. |  | | | |
| **location.reload()**  es un metodo que nos permite forzar la recarga de la pagina. | |  | | |
| **Location.replace()**  Reemplaza la url por lo que le ingresemos en los parametros. | |  | | |
| **New URLSearchParams(location.search)**  Podemos crear un objeto del tipo URLSearchParams, lo que le otorga ciertos metodos.  Algunos ejemplos en los siguientes cuadros.  Es una interfaz, en el ejemplo elegimos utilizar una variable para almacenarla con el nombre parametros. | | |  |
| **Parametros.has(“”)**  Nos permite buscar si en el queryString hay algo con lo mismo que introducimos por parametros. | | |  |
| **Parametros.get(“”)**  Nos devuelve lo que el usuario busco, es decir en el query string lo que esta despues del signo de pregunta. | | |  |
| **localStorage** | | | nos permitirá almacenar información por tiempo indeterminado. |
| **sessionStorage**  permite almacenarlo hasta que se cierre el navegador. | | | nos permitirá guardar información en sesión. Eso quiere decir que si usamos esta opción y cerramos el navegador, la información aquí almacenada se perderá. |
| **.setItem(“key”,”value”)**  Este metodo crea nuevos atributos y asigna valores para eso, el primer parametro es el nombre de la clave y el segundo el valor que esta lleve. | | |  |
| **.getItem(“key”)**  Nos devuelve el valor de la clave que le pasemos. | | |  |
| **.removeItem(“key”)**  elimina la clave cuyo nombre recibe por parámetro del almacenamiento | | | sessionStorage.removeItem("nombre")  //No estoy seguro de que sea nombre y no Dario en este caso, probar llegado el caso. |
| **.clear(“”)**  Lipia el historial del storage. | | |  |



De querer Guardar informacion mas compleja como por ejemplo un objeto, primero hay que pasarlo a JSON.

**Usando localStorage para requerir el nombre del usuario solo si, no se encuentra ya en el Storage.**





|  |
| --- |
| **Expresiones regulares**  Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en cadenas. En JavaScript, las expresiones regulares también son objetos. Estos patrones se utilizan con los métodos exec() y test() de RegExp, y con match(), matchAll(), replace(), replaceAll() (en-US), search() y split() métodos de String. |
| **Crear una expresión regular**  Construyes una expresión regular en una de estas dos formas:   * Usando una expresión regular literal, que consiste en un patrón encerrado entre barras, como sigue:     Las expresiones regulares literales proporcionan la compilación de la expresión regular cuando se carga el script. Si la expresión regular permanece constante, su uso puede mejorar el rendimiento.   * llamando a la función constructora del objeto **RegExp**, de la siguiente manera:     El uso de la función constructora proporciona una compilación en tiempo de ejecución de la expresión regular. Usa la función constructora cuando sepas que el patrón de la expresión regular cambiará, o no conoces el patrón y lo obtienes de otra fuente, como la entrada del usuario.  Esta funcion revisa si el mail es valido, conforme se indica en el RegExp cuales son las condicoines. |
| | **Métodos que usan expresiones regulares** | | | --- | --- | | **Método** | **Descripción** | | [exec()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec) | Ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un arreglo de información o null en una discrepancia. | | [test()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test) | Prueba una coincidencia en una cadena. Devuelve true o false. | | [match()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/match) | Devuelve un arreglo que contiene todas las coincidencias, incluidos los grupos de captura, o null si no se encuentra ninguna coincidencia. | | [matchAll()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll) | Devuelve un iterador que contiene todas las coincidencias, incluidos los grupos de captura. | | [search()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/search) | Prueba una coincidencia en una cadena. Devuelve el índice de la coincidencia, o -1 si la búsqueda falla. | | [replace()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/replace) | Ejecuta una búsqueda por una coincidencia en una cadena y reemplaza la subcadena coincidente con una subcadena de reemplazo. | | [replaceAll() (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replaceAll) | Ejecuta una búsqueda de todas las coincidencias en una cadena y reemplaza las subcadenas coincidentes con una subcadena de reemplazo. | | [split()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/split) | Utiliza una expresión regular o una cadena fija para dividir una cadena en un arreglo de subcadenas. | |

**Ejemplo de codigo para validar mails.**

function isValidMail(mail){

let regex =

new RegExp("^(?=.{1,64}@)[A-Za-z0-9\_-]+(\\.[A-Za-z0-9\_-]+)\*@[^-][A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)\*(\\.[A-Za-z]{2,})$")

return regex.test(mail)

}

console.log(isValidMail("hola@gmail"))      //false

console.log(isValidMail("hola@gmail.com"))  //true

console.log(isValidMail("hola"))            //false

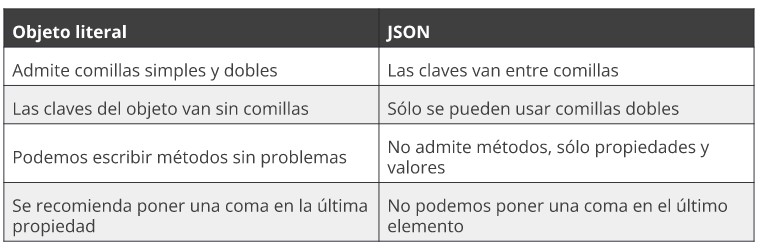
console.log(isValidMail("hola.com"))        //false

console.log(isValidMail("h@g5.com"))        //true

**Debe estar en una sola linea, por esa razon le puse un tamaño de fuente pequeño y la funcion sin indentacion, para evitar que el doc me haga el salto de linea. (hace zoom chicato).**

**JSON**

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

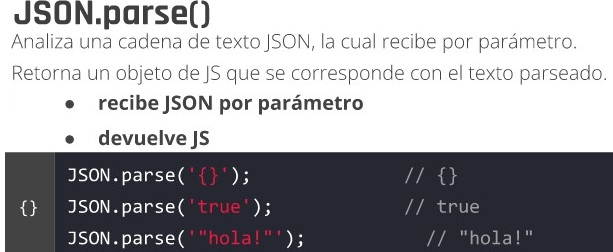


Al momento de trabajar con JSON, desde JavaScript debemos saber interpretar y **parsear** la informacion para poder trabajarla como lo hacemos con los objetos literales.

Tenemos dos procesos que nos facilitan esto:

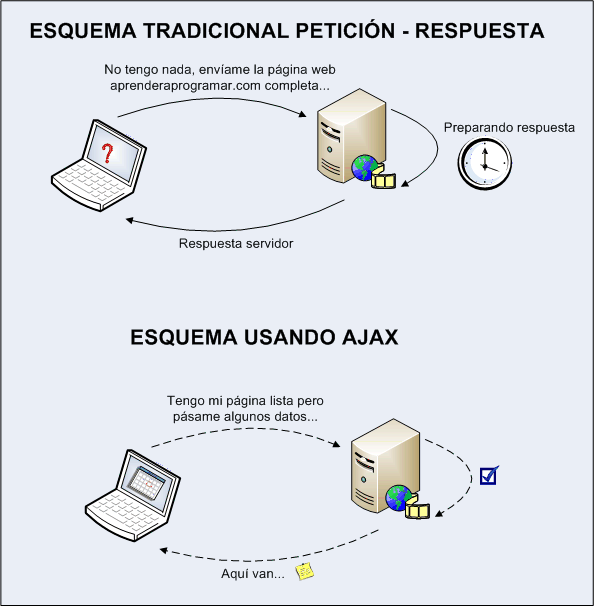
* **JSON.parse()** = JSON -> JS
* **JSON.stringify()** = JS -> JSON





**Ajax**

Asynchronous JavaScript and XML es un conjunto de tecnologias que se utilizan para crear aplicaciones web asincronas. Esto las vuelve mas rapidas y con mejor respuesta a las acciones del usuario.



1. Se produce un evento en una página web(se carga la pagina, se hace clic en un boton)
2. Javascript crea un objeto XMLHttpRequest
3. El objeto XMLHttpRequest envia una solicitud a un servidor web
4. El servidor procesa la solicitud
5. El servidor envia una respuesta a la pagina web
6. La respuesta es leida por JavaScript
7. JavaScript realiza la accion adecuada (como la actualizacion de la pagina).

**Ejemplos prácticos de AJAX**

**Autocompletado de Google**

Esta te ayuda a completar tus palabras clave mientras las escribes. Las palabras clave cambian en tiempo real, sin embargo, la página como tal no cambia.

**Votación o sorteo**

En este se calculan resultados en tiempo real y se muestran los indicadores actualizado

**Chat de atención al cliente**

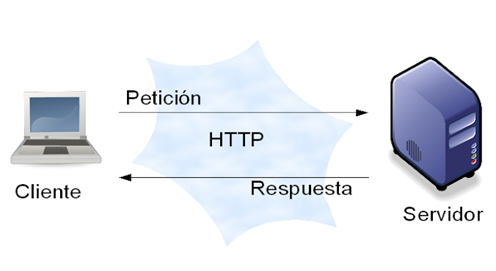
El usuario puede explorar la página al mismo tiempo y que el chat queda abierto sin que se recargue la web con cada nuevo mensaje

**Notificaciones en redes sociales**

Tanto Twitter, Facebook e Instagram, entre otras aplicaciones, reciben actualizaciones constantemente y llegan a ti como mensajes o contenido del inicio que podría interesarte

# 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 mas funciones de control y funcionalidad sobre la Web: cache o metodos de identificacion o autenticacion.

En versiones anteriores, los mensajes HTTP eran textos planos. En HTTP/2, los mensajes estan estructurados en un nuevo formato, lo que contribuye a una mayor legibilidad y debugging mas eficiente.

**Concepto de asincronismo**

Gracias al asincronismo generamos un código más funcional, rápido y eficiente, sin necesidad de recargar la página para actualizar la información. 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.

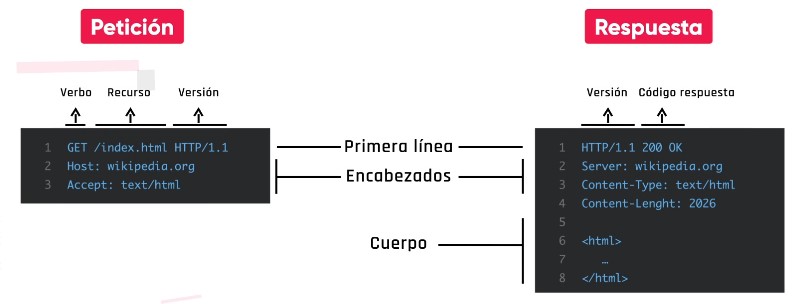
## Concurrencia y paralelismo Existen 2 tipos de asincronismo:

* 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.   
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.



Tantos las peticiones como las respuestas HTTP, comparten una estructura similar de tres campos.



|  |  |
| --- | --- |
| **Peticion** | |
|  |  |

Primero aparece el metodo HTTP con el verbo que se utiliza, solo se utiliza una de las opciones.

**Verbos:**

* Get indica que un archivo debe ser enviado hacia el cliente
* Post indica que datos seran enviados al servidor.
* **PUT** crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la **petición**.

El path es la direccion url completa del recurso, sin indicar el protocolo o nombre del dominio.

Despues la version del protocolo

La cabecera aporta informacion adicional, no es case sensitive. Es opcional y siguen las caracteristicas de unac adena de caracteres.

El cuerpo de mensaje en algun metodo http tambien es opcional.

|  |
| --- |
| **Respuesta** |
|  |

La respuesta lleva:

* Version del protocolo,
* Codigo de estado indicando si la peticion fue exitosa o no
* Mensaje del motivo con una descripcion breve del codigo de estado.
* Cabecera que trae informacion adicional, la cual es opcional.

http

Hay protocolos a cumplir en la trasnferencia de datos. Los cuales se fijan por convenios o tradicion.

El protocolo http hace referencia a hyper text transfer protocol, es un protocolo cliente servidor que gestiona las trasacciones web entre las dos entidades, cliente servidor.

Nos permite navegar hacia sitios web a traves de www y enlaces. Una direccion www comunmente conocida como triple W y formalmente llamada URI (Uniform Resource Identifcator.) y puede ser identificada de dos maneras:

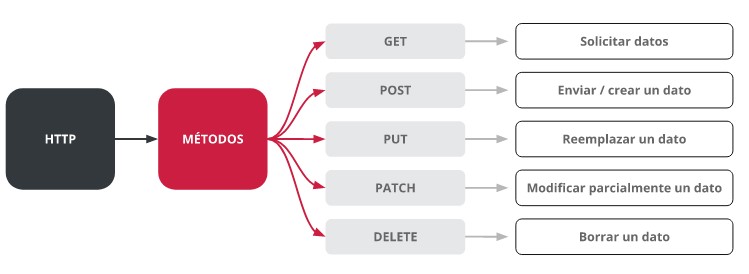
* URL(donde esta el recurso que buscamos obtener) que siempre empieza por un protocolo.
* URN es el nombre exacto del recurso uniforme, el nombre del dominio y en ocaciones el nombre del recurso.

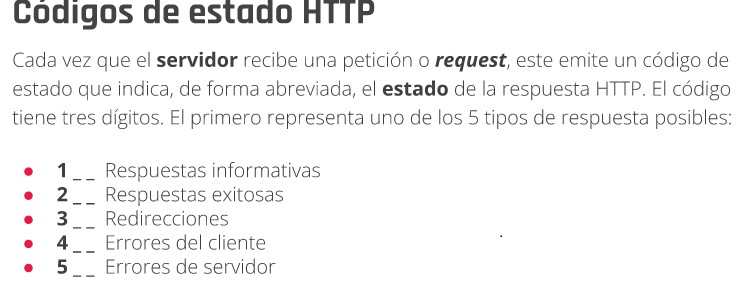


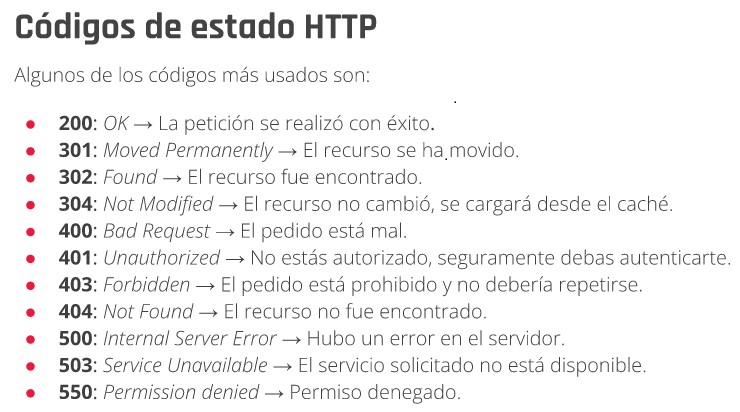
Estas no son tan utlizadas en la actualidad, es importante tener presente que cuando interactuamos con clientes, nos comunicamos con un servidor.

La informacion que se intercambia viaja oculta en los header o encabezados, llevando informacion necesaria tanto para el cliente como para el servidor.

HTTP lleva metodos de peticion, se usan para indicar la accion que se desea utlizar con un recurso determinado





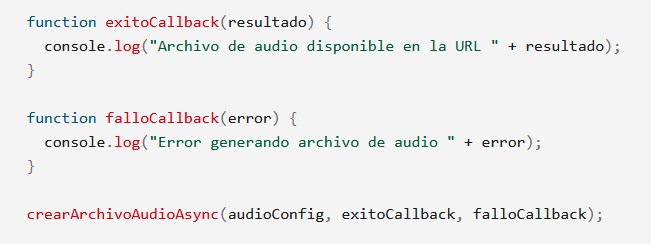


**Promesas**

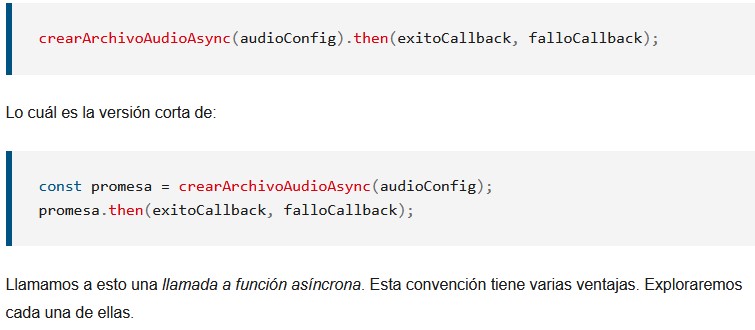
Una [Promise](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise) (promesa en castellano) es un objeto que representa la terminación o el fracaso de una operación asíncrona. Dado que la mayoría de las personas consumen promises ya creadas, esta guía explicará primero cómo consumirlas, y luego cómo crearlas.

Esencialmente, una promesa es un objeto devuelto al cuál se adjuntan funciones callback, en lugar de pasar callbacks a una función.

Considera la función crearArchivoAudioAsync(), el cuál genera de manera asíncrona un archivo de sonido de acuerdo a un archivo de configuración, y dos funciones callback, una que es llamada si el archivo de audio es creado satisfactoriamente, y la otra que es llamada si ocurre un error. El código podría verse de la siguiente forma:



... las funciones modernas devuelven un objeto promise al que puedes adjuntar funciones de retorno (callbacks). Si crearArchivoAudioAsync fuera escrita de manera tal que devuelva un objeto promise, usarla sería tan simple como esto:



[**Garantías**](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises#garantías)

A diferencia de las funciones callback pasadas al "viejo estilo", una promesa viene con algunas garantías:

* Las funciones callback nunca serán llamadas antes de la [terminación de la ejecución actual](https://developer.mozilla.org/es/docs/Web/JavaScript/EventLoop#ejecutar_hasta_completar) del bucle de eventos de JavaScript.
* Las funciones callback añadidas con then() incluso después del éxito o fracaso de la operación asíncrona serán llamadas como se mostró anteriormente.
* Múltiples funciones callback pueden ser añadidas llamando a then() varias veces. Cada una de ellas es ejecutada una seguida de la otra, en el orden en el que fueron insertadas.

Una de las grandes ventajas de usar promises es el encadenamiento, explicado a continuación.

[**Encadenamiento**](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises#encadenamiento)

Una necesidad común es el ejecutar dos o más operaciones asíncronas seguidas, donde cada operación posterior se inicia cuando la operación previa tiene éxito, con el resultado del paso previo. Logramos esto creando una cadena de objetos promises.

Aquí está la magia: la función then() devuelve una promesa nueva, diferente de la original:



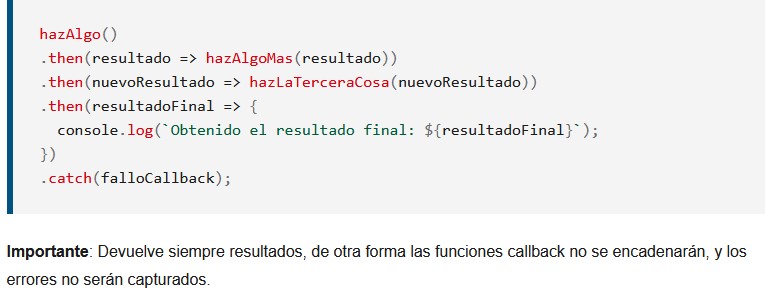
Esta segunda promesa (promesa2) representa no sólo la terminación de hazAlgo(), sino también de exitoCallback o falloCallback que pasaste, las cuales pueden ser otras funciones asíncronas devolviendo una promesa. Cuando ese es el caso, cualquier función callback añadida a promesa2 se queda en cola detrás de la promesa devuelta por exitoCallback o falloCallback.

Básicamente, cada promesa representa la terminación de otro paso (asíncrono on no) en la cadena.

En el pasado, hacer varias operaciones asíncronas en fila conduciría a la clásica pirámide de funciones callback:



Los argumentos a then son opcionales, y catch(falloCallBack) es un atajo para then(null, falloCallBack). Es posible que veas esto expresado con [funciones de flecha](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/Arrow_functions):



**Ejemplo de promise simulando llamado a una API.**

let promise = new Promise((resolve, reject) => {

    const user = {

        status: 'inactivo',

        userName: 'Nombre usuario',

        amount: 5

    }

    // Llamada a la API

    // url/GetUser?userId=Nombre\_usuario

    setTimeout(() => {

        if (user.amount < 1) {

            reject({ mensaje: "Fondos insuficientes" });

        } else if (user.status != 'activo') {

            reject({ mensaje: "La cuenta no está activa" });

        } else {

            resolve({ mensaje: "La transacción se realizó correctamente" });

        }

    }, 2500);

});

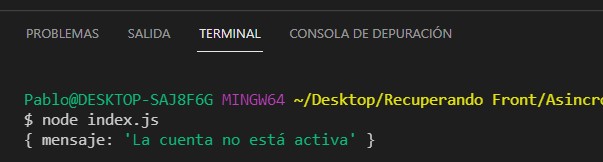
promise.then((respuesta) => {

    console.log(respuesta);

}).catch((err) => {

    console.log(err);

});



**APIs**

Tecnicamente, es una interfaz que permite la comunicación entre 2 aplicaciones. Si tuvieramos que simplificar un poco esta definicion y llevarlo a algo mas simple, imaginemos que es un sitio que en vez de responder algo visual, como HTML con CSS, nos responde informacion.

Los usuaiors probablemente no van a ingresar a este sitio, pero si una aplicación que necesita abastecerse de informacion particular. Por ejemplo, una aplicación que necesite un listado completo de todos los paises del mundo podria consumir una API que le otorgue esa informacion.

**REST**

Transferencia de Estado Representacional

Arquitectura del tipo cliente-servidor, 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.

* Todos los recursos del servidor tienen un nombre en forma de URL o hipervincula.
* Toda la informacion se intercambia a traves del protocolo HTTP.

A esas URL las llamamos **endpoints**, el servidor expone a los clientes a un conjunto de **endpoints** para que este pueda accerder, eeste conjunto de **endpoints** se llama **API**.

Un endpoint esta ligado al recurso solicitado, el cual debe tener solamente un identificador logico, y este proveer acceso a toda la informacion relacionada. Un **endpoint** entonces, tambien es un archivod e nuestro proyecto.

HTTP estandariza el mecanismo de llamada, pero no como es el dialogo entre le cliente y el servidor.

Caracteristicas:

* **Separar la aplicación web en dos:** Por una lado la interfaz de usuario en una aplicación, por otro lado todo lo que la aplicación provee como servicio que la interfaz web consume.

Por ejemplo la interfaz web para hacer pedidos a domicilios, mientras la otra capa hace la logica del negocio, notifica los pagos, interactua con los proveedores, etc.

Es decir que la interface va a seguir funcionando en paralelo mientras la otra parte hace otras cosas. Ambas capas estarian conectadas por medio de APIs.

* **Ubicación de los recursos:** dentro de la comunicación del cliente servidor, surge la necesidad del cliente de acceder a recursos del servidor, imaginemos que un cliente de spotify se comunica con el servidor del mismo y le surjan preguntas como:

- ¿Cuáles son las canciones top 50 del mundo?.

El servidor no sabe necesariamente donde se encuentra, aquí es donde REST dice que si necesitas ese recurso, tendrias que dirigirte a traves de una direccion unica.

* **Stateless o “Sin estado”:** Rest nos propone que el servidor no almacene ningun dato acerca de las peticiones que haga el cliente, de esta manera cada solicitud sera tratada como nueva y de forma independiente, no existe el concepto de sesion de usuario que recuerde cosas.

En cada pedido, el cliente debera enviar toda la informacion necesaria, incluido datos de autentificacion y autorizacion.

* **Cacheable**: Si no existe un estado entre las peticiones, es probable que si vuelvo a pedir el mismo recurso mas tarde, voy a obtener el mismo resultado. Si alguna informacion permanece sin modificacion durante mucho tiempo, el cliente podria recordar esa respuesta para no tener que hacer constantemente el pedido, para no sobrecargar el servidor.

El servidor debe responder con un encabezado **max-age**, que tiene un valor el cual determina el tiempo maximo en segundos que tiene ese recurso, una vez expirado el cliente debe volver a pedir el recurso.

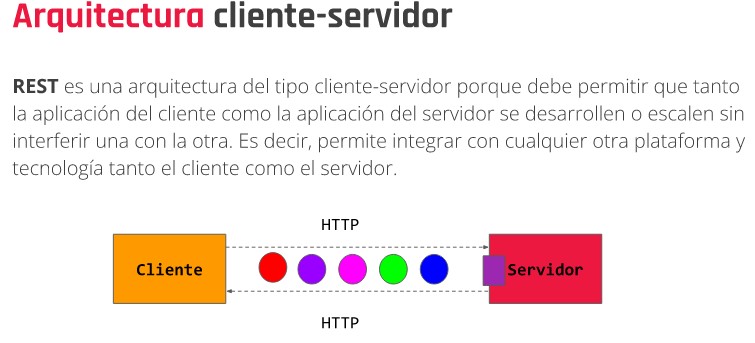
Todos los datos a los que podamos acceder, estaran agrupados bajo nombres, que son sustantivos. Cada uno de ellos los llamaremos Recursos.

Todo lo que resulte util al cliente puede ser un recurso, en cualquier formato. En rest se aconceja un identificador de recursos para cada dato.

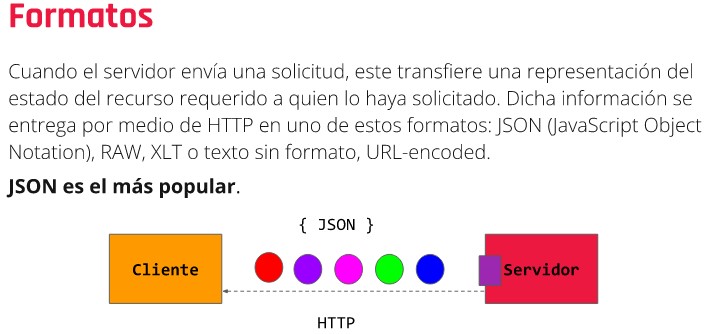
Estos recursos son accesibles mediante URLs. Esto nos da acceso tambien a la informacion relacionada al recurso, ya uqe el servidor y el recurso comparten “recursos”.

* Cuando se envian datos con formato JSON, se agrega en la cabecera un encabezado diciendo el tipo de contenido.
* Raw no se utiliza.
* Text, se puede enviar en forma de texto.
* Url-encoded: envia algo similar a un queryString.

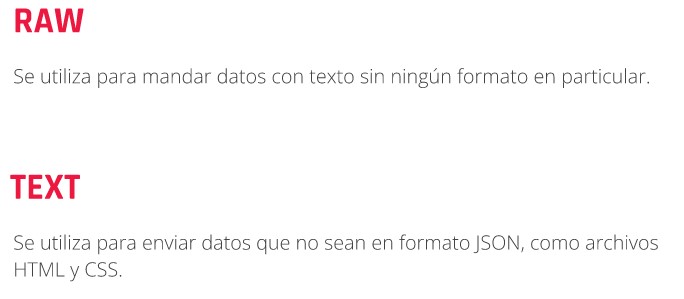
**HEAD:** es un metodo con el cual podemos conocer la fecha de modificacion del recurso, dado que se cumpla con el requisito de no superar el max-age, sabra si tiene o no que volver a pedir los datos actualizados.

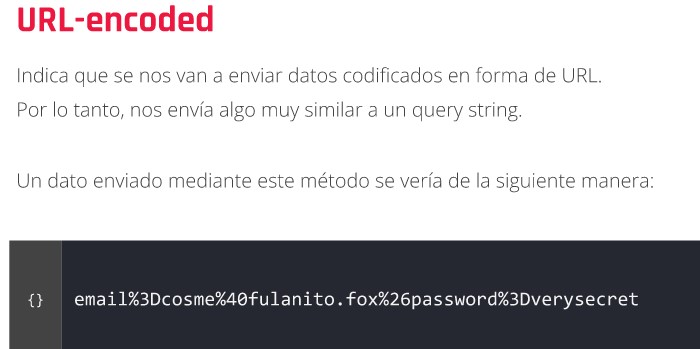


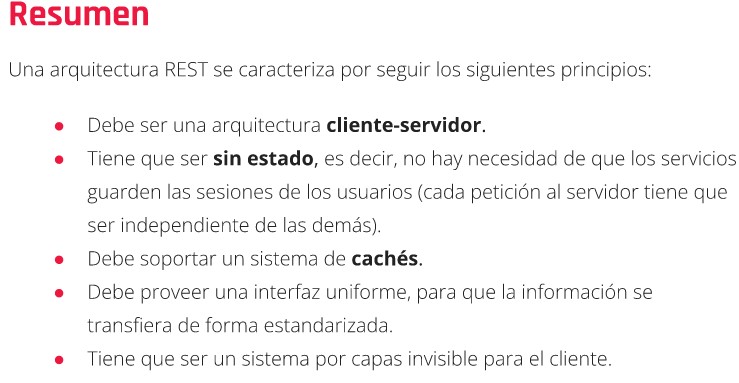










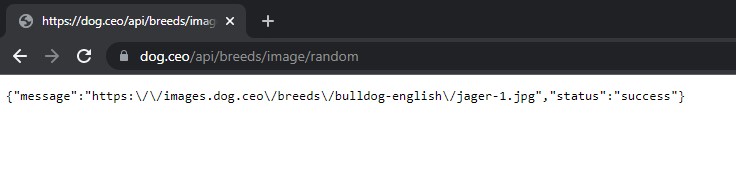


**Ejemplo de un api**

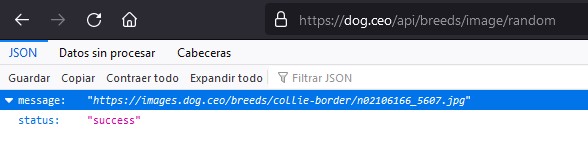
La siguiente direccion es el acceso a una API, nos genera una imagen de un perro de forma aleatoria. Podemos comprobar lo que la API devuelve si pegamos el link en el navegador o llamarla desde JS para obtener el JSON que nos devuelve.

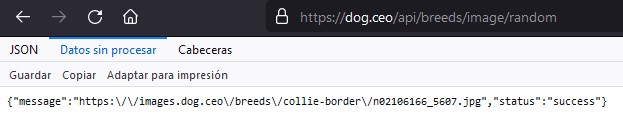
<https://dog.ceo/api/breeds/image/random>

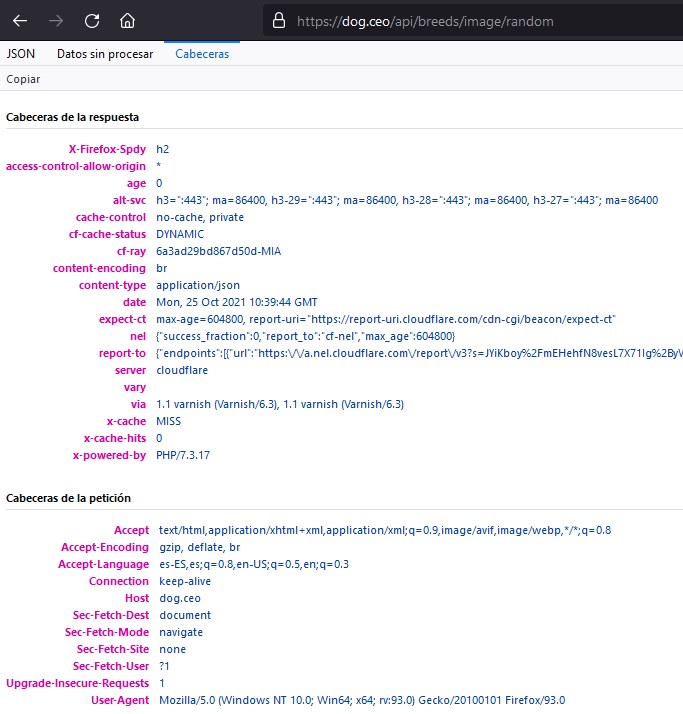
**Google Chrome** nos lleva a esta pagina, donde nos indica lo que nos devolvera.



**Firefox** devuelve tambien un JSON, pero es un poco mas detallado y ordenado, incluyendo la Cabecera del mensaje y el contenido de forma mas legible.

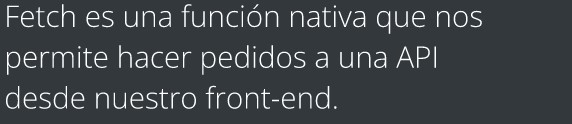






Esta api nos devuelve un JSON con un objeto que tiene 2 atributos, **message** y **status** , message trae una Url con una foto random de un perro y status

**AJAX Fetch**

****

Las solicitudes asincronicas nos permitiran desde el lado del front end, hacer peticiones a distintas APIs y consumir los datos que estas nos proveen de forma dinamica y sin poner en riesgo la carga del resto de nuestra funcionalidad.

JavaScript nos provee las funcionalidades de Fetch las cuales son de gran utilidad, ya que nos permiten de manera sencilla y rapida, establecer una comunicación con un servidor a traves de los distintos endpoints que nos provea su API.

<https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch>







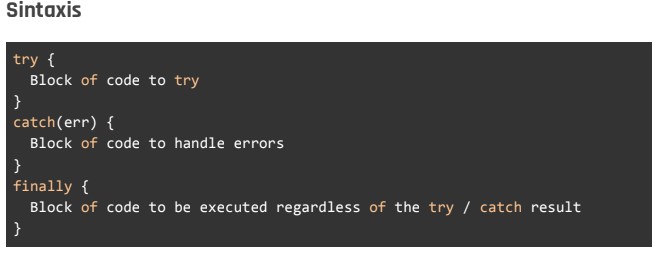


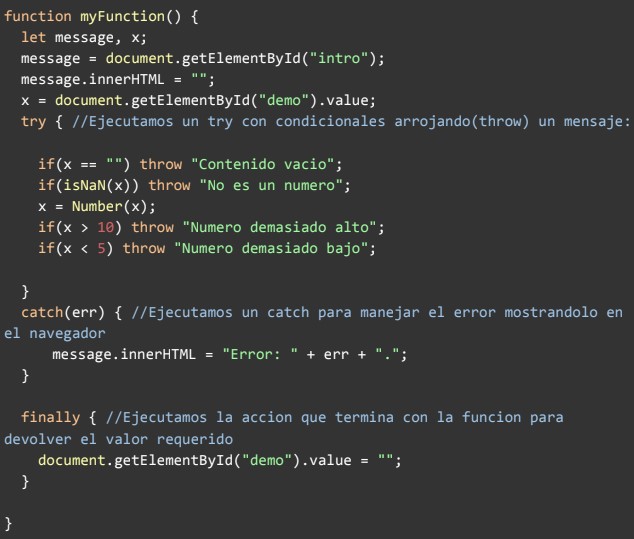
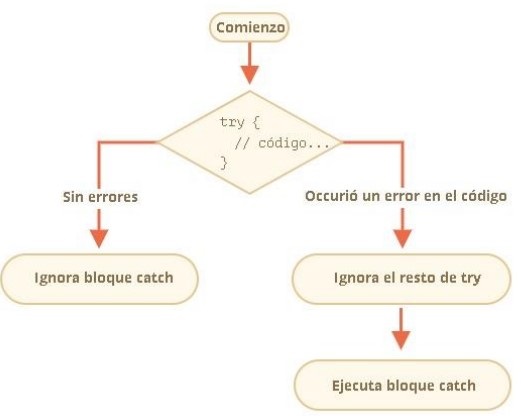
****

**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.

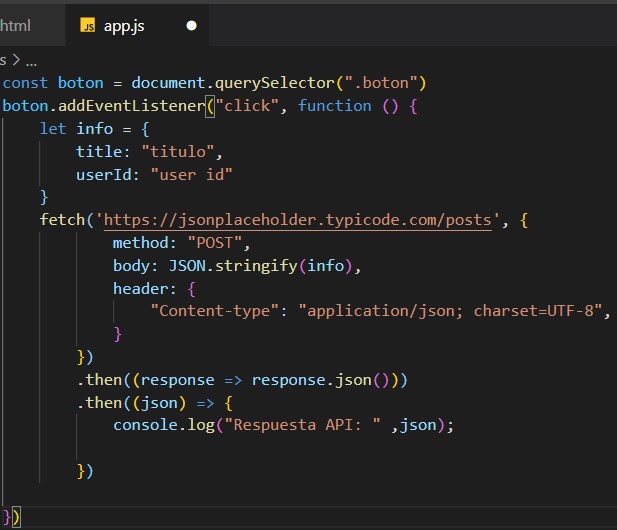
* La declaración try permite probar un bloque de código en busca de errores.
* La declaración catch permite manejar el error.
* La declaración throw permite crear errores personalizados.
* La declaración finally permite ejecutar código, después de intentar y capturar, independientemente del resultado.





**Dataset buscar**

**Ejemplo de una llamada Post a una API borrador.**

****

**Ejecutamos el script desde el html con el boton creado y la API nos devuelve un ID.**

