

# Notas JavaScript

**Var:** genera variables globales que generan un gran lío por lo que

Se implementó **Let** para no romper con la lógica de la programación

No tiene un fuerte tipado, puedes guardar un entero donde tengas un **String** y mas (tipado débil)

Los datos numéricos son sólo **numbers** sin distinción de entero o flotante.

Booleano: **boolean**

Constantes: **const** Usar **Snake uppercase** (EJEMPLO\_DE\_SNAKE)

**null**: alguna variable que tiene un valor que no quieras que interprete

**Undefined**: es el valor de una variable que no se le asignado un valor aún.

## Tipos de datos (Básicos)

```
//String es una cadena de texto siempre  
//debe de ir entre comillas dobles  
var vaso = "leche"  
  
//number, No lleva comillas y se refiere  
//al numero con el cual podemos hacer  
//operaciones matemáticas  
  
var edadUsuario =10;  
  
//boolean es un valor con solo 2 salidas  
true o false  
var usuarioPremium = false;  
  
//undefined variables sin inicializar,  
//existe pero no está definido  
var proximaCita= undefined;  
  
// null, es un valor que aún no tenemos  
//definido  
  
var asistenciaInvitado= null;
```

```
/*Tipos de datos primitivos y no primitivos  
string  
number  
boolean  
undefined  
null  
symbol  
object (No es primitivo )
```

Se recomienda **Lower camel case** para nombrar variables

EjemploLowerCamelCase

EjemploUpperCamelCase

procura que el nombre sea específico

## Declarando listas

Let **myList**=["Hola", "adios", "pepe", 34] Debido al tipado débil  
esta estructura la trabaja como un object se pueden guardar distintos tipos de datos

Existe otro dato llamado **set** el cual es una lista que no admite valores repetidos: Let **mySet**= new set (["Hola", "adios", "pepe", 25])

Si a esta estructura le ingreso un valor repetido va a seguir diciendo que tiene 4 elementos

**Map** es una estructura también conocida como diccionario que está organizado a manera

Clave valor el mapa no permite claves repetidas

Se declara: Let **myMap** = new myMap ([[pepe, 13], [jose, 25], [Angel, 30]])

para agregar un dato **myMap.set("clave", valor)** **myMap.get("clave")** arroja el valor

para agregar un dato a una lista se usa el método **add()** **mySet.add("pepe")**

# Bucles

## Declaración de bucles

```

for (const value of myList) {
    console.log(value)
}

let myCounter = 0

while (myCounter < myList.length) {
    console.log(myList[myCounter])
    myCounter++
}

```

de esta manera daría un undefined por el ==

## Clases

tipo de objeto concreto

```

// Clases

class MyClass {
    constructor(name, age) {
        this.name = name
        this.age = age
    }
}

let myClass = new MyClass("Brais", 36)
console.log(myClass)
console.log(myClass.name)

```

Si no le das valores igual se crea con valor indefinido  
muestra ambos datos

## Enum

```

const MyEnum = [
    DART: "dart",
    PYTHON: "python",
    SWIFT: "swift",
    JAVA: "java",
    KOTLIN: "kotlin",
    JAVASCRIPT: "javascript"
]

const myEnum = MyEnum.JAVASCRIPT
console.log(myEnum)

```

Similar a un mapa le asigna valores a cada clave  
es similar al enum de C

## Trabajando con JavaScript en nuestro código

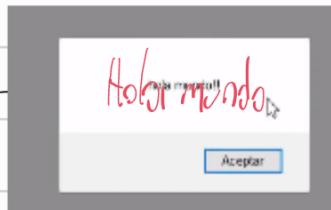
Podemos agregar código utilizando la etiqueta `<script>` en el HTML

```

ReposoJS
1  <!DOCTYPE html>
2  <html lang="es">
3      <head>
4          <meta charset="utf-8" />
5          <title>Reposo JavaScript</title>
6          <script src="main.js" type="text/javascript"></script>
7      </head>
8      <body>
9          <h1>
10             Aprendiendo JavaScript rápido
11         </h1>
12         <p>Hola soy Víctor Robles WEB</p>
13     </body>
14 </html>

```

Osea que carga un script  
Alerta usando alert



EXPLORER

- OPEN EDITORS
- MASTER-FRAMEWORKS-JS
  - ReposoJS
  - index.html
  - main.js

Esta es la carpeta del ...

Reposo JavaScript

Victor Robles190

Aprendiendo JavaScript rápido

Hola soy Victor Robles WEB

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <title>Reposo JavaScript</title>
  </head>
  <body>
    <h1>Aprendiendo JavaScript rápido</h1>
    <p>Hola soy Victor Robles WEB</p>
    <div id="datos"></div>
  </body>
</html>

```

```

index.html   JS main.js x
ReposoJS > JS main.js > ...
1
2 var nombre = "Victor Robles";
3 var altura = 190;
4
5 var concatenacion = nombre + " " + altura;
6
7 var datos = document.getElementById("datos");
8 datos.innerHTML = concatenacion;
9

```

Movemos script al final del body para que cuando se ejecute ya exista el div con el id "datos". Así podemos controlar donde se muestran los datos

```

var datos = document.getElementById("datos");
datos.innerHTML =
  <h1>Soy la caja de datos</h1>
  <h2>Mi nombre es: ${nombre}</h2>
  <h3>Mido: ${altura}</h3>
;

```

puedo agregar líneas de HTML en el script utilizando comillas simples y el método innerHTML

## Aprendiendo JavaScript rápido

Hola soy Victor Robles WEB

Soy la caja de datos

Mi nombre es: Victor Robles

Mido: 190

```

if(altura >= 190){
  datos.innerHTML += '<h1>Eres una persona ALTA</h1>';
} else{
  datos.innerHTML += '<h1>Eres una persona BAJITA</h1>';
}

```

## Aprendiendo JavaScript rápido

Hola soy Victor Robles WEB

Soy la caja de datos

Mi nombre es: Victor Robles

Mido: 189 cm

Eres una persona BAJITA

Si no usamos

+ = y solo usamos

= reescribiríamos el valor "HTML" que hay en datos

Otra manera de hacer  
el bucle Por

```

for(var i = 0; i<=2020; i++){
  // bloque de instrucciones
  datos.innerHTML += "<h2>Estamos en el año: "+i;
}

```

*Llamadas*

```

function MuestraMiNombre(nombre, altura){
  var datos = document.getElementById("datos");
  datos.innerHTML =
    <h1>Soy la caja de datos</h1>
    <h2>Mi nombre es: ${nombre}</h2>
    <h3>Mido: ${altura} cm</h3>
  ;
}

MuestraMiNombre("Victor Robles WEB", 190);

```

*Document.write*  
para mostrar  
en pantalla

```

function MuestraMiNombre(nombre, altura){
    var misDatos = '';
    <h1>Soy la caja de datos</h1>
    <h2>Mi nombre es: ${nombre}</h2>
    <h3>Mido: ${altura} cm</h3>
}

return misDatos;
}

function imprimir(){
    var datos = document.getElementById("datos");
    datos.innerHTML = MuestraMiNombre("Victor Robles WEB", 190);
}

imprimir();

```

Similar a C usa el return

La función tiene un valor de salida  
Puntaje (10) = 9

## JavaScript en Close



### Introducción a javascript

QUE ES JAVASRIPT? PARA QUE UTILIZARLO?  
DIFERENCIAS ENTRE JAVASCRIPT Y JAVA  
COMO FUNCIONA HTML+CSS+JAVASCRIPT  
FORMAS DE UTILIZAR JAVASCRIPT EN NUESTRA PAGINA WEB  
BUENAS PRACTICAS UTILIZADAS EN JAVASCRIPT

podemos usar JS para Front y para Back

Con esto podemos verificar los datos y validarlos

Ejemplo: validar que un hotmail tenga un @

Importante: usar bien nuestras variables

-Ser claros, declarar la variable con el tipo de dato con el que lo vamos a procesar.

Si declaramos un valor como cadena de texto, ya no vamos a poder aplicarle operaciones matemáticas

Más p:ky

Lenguaje compilado, si tienes un error en la linea 16 no corre las 15 anteriores

Amigo cool, jala a todos lados

Lenguaje interpretado

Si tienes un error en la linea 16 sí las 15 líneas anteriores

Solo es back

Se usa para proyectos grandes con demasiada información



VS



Difference Between Java and JavaScript

Back y Front

Compilado: traducción del lenguaje natural al lenguaje máquina

JavaScript nos ayudará a extraer la información de los formularios

Java: nos servirá para manipular estos datos

- La versión que más se usa es la 8 por que los cambios no han sido significativos

- Vamos a usar la versión 17 Java Enterprise JE

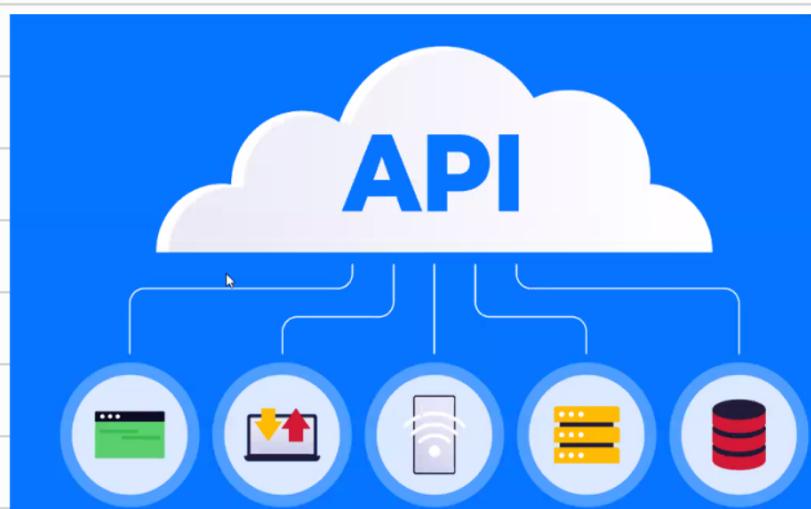
JS puede usarse en  
muchos lados

Java Enterprise edition

Versión  
para proy  
grandes

podemos agregar animaciones con JavaScript

<https://webdesignerwall.com/trends/30-truly-interactive-websites-built-css-javascript>



API: Delimita la interacción

que puede tener el  
JSvar 'o.

Hay APIs para todo

<https://fakestoreapi.com/>  
Web de APIs

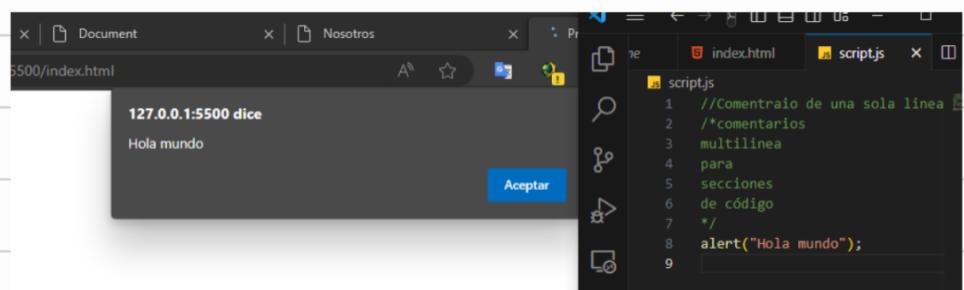


# Ejemplos programando

```
index.html > html > head
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
6      <title>Practicando con JavaScript</title>
7  </head>
8  <body>
9
10
11      <script src="./script.js"></script>
12  </body>
13  </html>
```

La etiqueta script nos permite agregar líneas de Código o llamar un archivo JS

Hola mundo con JS



```
script.js > ...
1 //Comentario de una sola linea
2 /*comentarios
3 multilinea
4 para
5 secciones
6 de código
7 */
8 alert("Hola mundo");
9 var frasco = "pastillas";
10 var frasco = "pastitas con chocolate";
11 var frasco = "pastillas";
12 var frasco = "shampoo";
13 console.log(frasco);
14
```

Lo salida la podemos ver  
En la consola del buscador

```
var peso=80;
console.log(typeof(peso));
var peso=peso.toString();
console.log(typeof(peso));
console.log((peso.toString()));
```

Cambiando variable  
de tipo de dato number → String

```
console.log(typeof(edad));
var edad=parseInt(edad);
console.log(typeof(edad));
```

Cambiando el tipo de dato ↗  
de una variable String → number  
Ejercicio

```
console.log("Cambiando de string a
number");
console.log(typeof(numeroTelefono));
var numeroTelefonoNumber= parseInt
(numeroTelefono);
console.log(typeof(
numeroTelefonoNumber));
console.log("Cambiando de number a string
string");
console.log(typeof(precioExtracción));
var precioExtracciónString =
precioExtracción.toString();
console.log(typeof(
precioExtracciónString));
```

# funciones en JS

## ¿QUÉ ES UNA FUNCIÓN?

Una función es un trozo de código reutilizable en el que hay un conjunto de instrucciones, este código solo se ejecuta cuando se llama a dicha función.



```

Palabra reservada      Nombre de la función      parámetros      Cuerpo de la función
function miFuncion(p1, p2){
  console.log(p1 + p2);
}
  
```

## Terminos

### Scope

es el alcance de una variable en nuestro código global: Se puede utilizar una variable dentro de cualquier parte del código

**Hoisting:** Es cuando puedes invocar alguna función antes de declararla anticipa variables con var y las funciones

## Ventajas de las funciones

Nos permiten a ahorrar líneas de código al reutilizar la sección una y otra vez

También nos permite darle abstracción al código para mantener seguridad

## Declarar e invocar funciones

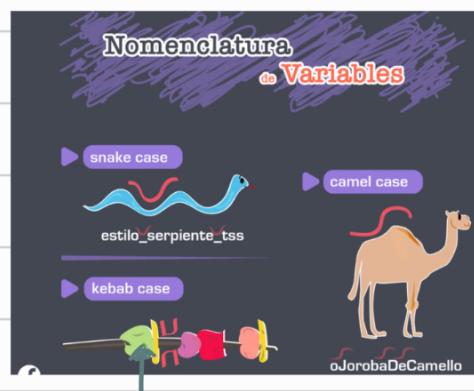
```

//Repaso de funciones
/*
una función en un bloque de
código que realiza una tarea
específica cuando se llama
* cada función de JavaScript es
almacenado como un object function
*/
function saludar(){
  console.log("Hola soy una
  función ");
}

saludar();
/**/
  
```

## Recordatorio:

Una función se almacena como object Function.



Separados por guion medio

\* Las funciones son simplemente objetos especiales con la capacidad adicional de ser invocados (llamados como funciones). Esto significa que puedes asignar funciones a variables, pasárselas como argumentos a otras funciones y devolver funciones desde otras funciones. Esto facilita la creación de funciones de orden superior, como funciones de devolución de llamada y funciones que generan otras funciones.

```

function suma(a,b){
    return a+b;
}

console.log(suma);
var resultado=suma(5,3);
console.log(resultado);

```

**global:** Se puede utilizar una variable dentro de cualquier parte del código.

- 👉 **Var:** no es tan recomendable por el comportamiento de la variable y la poca adaptabilidad
- 👉 **let** preferible y más seguro de utilizar porque además de funcionar de manera global también puede ser utilizado de manera local
- 👉 **Const:** su valor no puede ser reasignado.

## Funciones anónimas

Funciones declaradas sin un nombre y luego asignadas a variables o pasarse como parámetro a otra función

```

//funciones anónimas/
/*
Estas funciones pueden ser declaradas sin un nombre y luego asignadas a variables o pasarse como parámetros en otras funciones la diferencia principal de estas funciones es que declaran sin un nombre
*/
const saludo= function(){
    console.log("Hola vida");
}
saludo();

```

```

HOLA VIDA
>

```

```

const resta= (a,b)=> a-b;
console.log(resta(5,3));
/* esta función flecha es una forma más corta de:
const resta= function(a,b){
    return a-b;
}

```

/\*//Funciones flecha//  
son una forma más concisa de definir funciones anónimas. se le llama flecha debido a su sintaxis que utiliza ( $=>$ )  
\*Básicamente, se trata de reemplazar la palabra function y añadir  $=>$  antes de abrir nuestras llaves

Un **bloque** se define como lo que declaras entre {}

## Tipos de Función:

Tradicionales

Anónimas

Función Flecha

# Control de Flujo

## Poder manipular el orden

Cuando hablamos del orden natural en el que se ejecutan las instrucciones de nuestro programa(arriba/abajo, izq/der) estas instrucciones pueden ser una función un console.log, un ciclo, etc.

la importancia que tiene el control de flujo radica en la posibilidad de poder ejecutar cosas instrucciones dependiendo de una condicional (un elemento o instrucción detonante)

## Condicionales

son palabras reservadas que nos permiten evaluar si una condición se cumple o no, bajos ciertos criterios. las respuestas o salidas esperadas siempre partiran de un true o false

If

nos sirve para validar si una condición se cumple o no. Si se cumple, entonces ejecutamos un bloque de código que está dentro del if. si no se cumple, entonces no hacemos nada.

Nota: es la condicional más sencilla, pero la más limitante porque solo nos da una opción.

```
if(condicion){  
    //bloque de código que se ejecuta si la condición es verdadera()  
    //ejemplo: Felipe va a ir una fiesta y necesita tener su camisa favorita limpia para poder ir  
}  
/*
```

```
if (camisaLimpia=true){  
    console.log("Felipe va a la fiesta");  
}  
/*
```

else (de otro modo, de lo contrario, si no)

```
if(condición detonante){  
    //bloque de código que se ejecuta si la condición es verdadera
```

```
}else{//bloque de código que se ejecuta si la condición es falsa
```

```
}
```

```
*/  
  
if (camisaLimpia=true){  
    console.log("Felipe va a la fiesta");  
}else{ console.log("Felipe se queda en su casa a hacer lloración")}
```

```

}
/*
else if
Esta condición nos ayuda a jugar con más opciones y más detonantes, para tener muchas
posibilidades en el mismo código. para lograr esto vamos a unir el else con el if
if(condición detonante){
//bloque de código si la condición es verdadera
}else if(condición detonante 2){
//bloque de código si la condición 1 es falsa y la condición 2 es verdadera
}else{
//coloque de código si las dos condiciones son falsas
}
*/
//ejemplo2: Felipe tiene hambre, y quiere comer chilaquiles verdes
if (hambre = true){
console.log("Felipe va a comer");
}else if(chiaquiles= verdes){
console.log("Felipe come chilaquiles")
}else{
console.log("Felipe no come y se pone triste");
}

```

Prompt: es la espera de un valor de entrada como un **Getch**  
 O como la versión que es más sencilla como **Readline**  
**El código se detiene hasta que le ingresemos el valor** El dato que ingreso  
lo mete como cadena  
de texto

- Ingresar especialidad
- Validar usuarios registrados
- Verificar cita
- Generar cita
- Registrar pacientes
- \* Buscar información

CUNCIIONES QUE  
VAMOS A REALIZAR

# Operadores lógicos y Condicionales

## Objetivos

Al final de esta sesión seremos capaces de:

- Describir qué son los operadores
- Utilizar los principales operadores de JS: asignación, cadena, lógicos, aritméticos, de comparación e igualdad estricta.
- Definir y leer expresiones.
- Usar operadores para combinar expresiones lógicas.

```
Operador
Un elemento que nos permite realizar una operación entre dos o mas elementos

tipos:
-operadores aritméticos (+, -, *, /)
- operadores de asignación (=, ==, ===)
-operadores de cadenas(toLowerCase,toUpperCase,trim,toString, concat) *tambien conocidos como métodos
operadores lógicos (&&, || , !)
operadores de comparación (<, >, <=, >=, ==, ===)

operadores aritméticos
Son operadores aritméticos que nos permiten tomar valores numéricos como sus operandos y retornar un valor numérico único.
corresponden a operaciones matemáticas

+ suma +
- resta -
* multiplicación
/ división /
% módulo % (a%b te arroja el residuo de la división de a/b )
++ incremento ++ (variable++ incremento en uno en la variable llamada variable)
-- decremento -- (variable-- decremento en uno en la variable llamada variable)

diferencias operador
= igual (asignación de valor a una variable)
== igual (comparación )
=== los números deben de ser totalmente iguales ya que java al comparar 2 variables de tipo de dato diferente hace una conversión

por ejemplo
5=="5" sería igual a true aunque uno sea un número y el otro sea una cadena de texto
5==="5" //false todos deben de ser el mismo tipo y del mismo valor, en ocasiones es más recomendado este para tener un cuidado más estricto de nuestros tipos de datos.

*/
//función para convertir de grados celsius a fahrenheit

function CtoF(gradosC){
    return gradosC*1.8+32;
}
let gradosF=CtoF(12);
console.log("el total de la conversión es: "+ gradosF+ "grados farenheit");
```

Realizamos una conversión de grados Cº a Fº

# add event listener para agregar llamadas a una función a partir de una acción algo así como un callback

```
abrir.addEventListener("click")//define una función que va a ser llamada cada cuando el evento especificado es entregado/le ocurre al objetivo  
los objetivos comunes son elementos o sus hijos, documentos o la ventana, pero el objetivo debe de ser un objeto que soporte la acción  
/*syntax  
addEventListener(type, listener)  
addEventListener(type, listener, options)  
addEventListener(type, listener, useCapture)  
type: un string representando el caso sensible a escuchar (me imagino que si no escucha no actúa)  
listener: el objeto que recibe una notificación (objeto que realiza la interfaz de eventos ) cuando el evento del tipo especificado ocurre  
options (opcional):un objeto que especifica características del oyente del evento las opciones disponibles son  
-capture: un booleano indicando que eventos de este tipo van a ser enviado al oyente registrado antes de ser enviado al Objetivo del evento  
debajo del arbol del DOM. Si no se especifica el valor predeterminado es falso  
-once: un valor booleano que indique que el oyente debe ser invocado como maximo una vez después de ser agregado. si es true, el oyente  
sería removido cuando es invocado si no se especifica es falso  
-passive: Un valor booleano que, si es verdadero, indica que la función especificada por el oyente nunca llamará a preventDefault(). Si un  
oyente pasivo llama a preventDefault(), el agente de usuario no hará nada más que generar una advertencia en la consola.  
-signal: Una señal de aborto. El oyente se eliminará cuando se llame al método abort() del objeto AbortSignal dado. Si no se especifica, no  
se asocia ningún AbortSignal con el oyente  
useCapture: Un valor booleano que indica si los eventos de este tipo se enviarán al oyente registrado antes de enviarse a cualquier EventTarget  
debajo de él en el árbol DOM. Los eventos que se propagan hacia arriba a través del árbol no activarán un oyente designado para utilizar la  
captura. La propagación y captura de eventos son dos formas de propagar eventos que ocurren en un elemento anidado dentro de otro elemento,  
cuando ambos elementos han registrado un identificador para ese evento. El modo de propagación de eventos determina el orden en que los  
elementos reciben el evento. Consulte Eventos de nivel 3 de DOM y Orden de eventos de JavaScript para obtener una explicación detallada.
```

```
outer.addEventListener("click", onceHandler, once);  
outer.addEventListener("click", noneOnceHandler, noneOnce);  
middle.addEventListener("click", captureHandler, capture);  
middle.addEventListener("click", noneCaptureHandler, noneCapture);  
inner1.addEventListener("click", passiveHandler, passive);  
inner2.addEventListener("click", nonePassiveHandler, nonePassive);
```

ejemplos de uso de la sintaxis para la negación de passive o capture o once solo se agrega el none

al exponenciación se realiza utilizando \*\*  
primero las variables después las funciones y al ultimo  
las líneas de código general como recomendación

# Estructuras de control) Ciclos, arreglos y matrices

## Objetivos

- Definir y usar matrices
- Acceder a elementos de una matriz en una posición determinada
- Agregar un nuevo elemento a una matriz
- Eliminar un elemento de una matriz
- Usar los diferentes bucles de JavaScript (While y For)
- Explicar las principales diferencias entre While y For.

//Expresiones

//ciclos matrices y arreglos

/\*

condiciones: alterar el orden natural del código

estructura: Controlan la ejecución del código

-ciclos o bucles (loops)

También conocidos como bucles, son estructuras de control que nos van a permitir ejecutar un bloque de código de forma repetida, hasta que su cumpla una condición específica

es importante usar ciclos para automatizar tareas repetitivas/

ciclos:

-while

-do while

-for

iterar: proceso de repetir una secuencia de instrucciones estructura de control: herramienta especial para crear algoritmos o programas más dinámicos y flexibles

-while: ejecuta un bloque de código mientras se cumpla una condición específica este ciclo se repite mientras las condiciones sea verdadera, y el bloque de código se ejecuta siempre y cuando la condición se mantenga. la condición se evalúa antes de cada interacción

```
while(condición){  
    //código que se ejecuta  
}
```

```
*/  
while (miCruhMeQuiere){  
    console.log("seré feliz")
```

```
//NOTA esto significa peligro  
while(true){  
}
```

Es un ciclo sin fin ya que siempre es verdadero.

```
while(pacientesEnConsultorio<capacidadPacientes)
  console.log("seguimos dando servicio")
  pacientesEnConsultorio++;
}
```

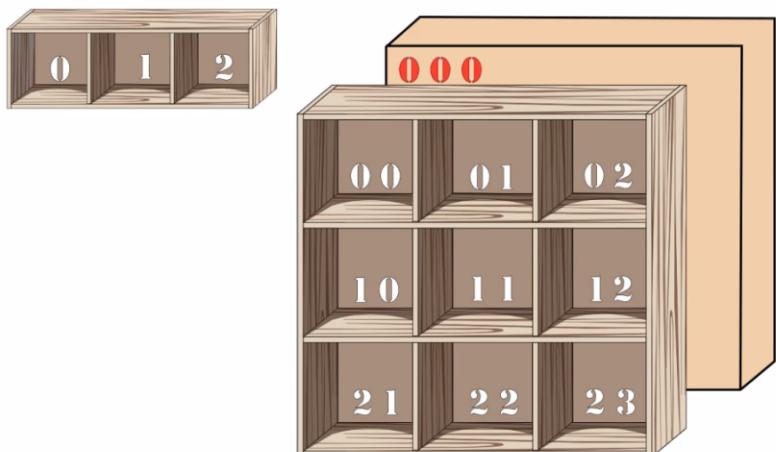
Seguimos dando servicio

Resultado del bucle

muestra el número de veces que se mostró el mensaje

# Arreglos

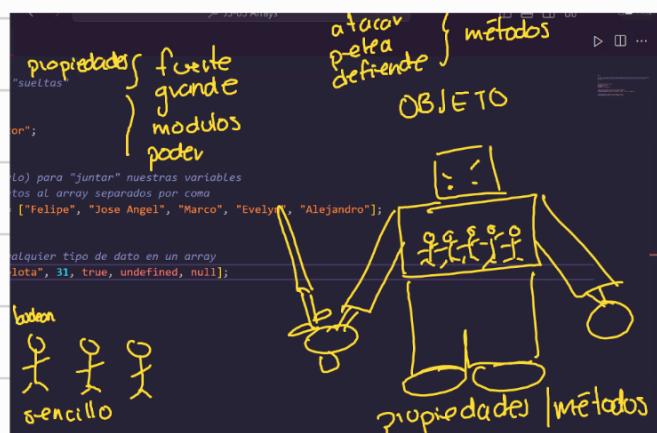
## Arreglos



Los elementos de un arreglo

- tienen algo en común
- objeto
- función

\* tener un array es sistematizar su almacenamiento y ordenamiento

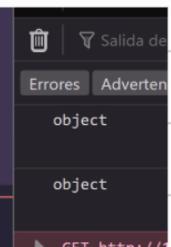


Propiedades: como se ve  
Métodos: lo que pue hacer

Cuando los power rangers(datos primitivos/solitos) se ven superados se unen en algo más grande lo cual les da poderes habilidad (métodos)

```
//Usar un array (arreglo) para "juntar" nuestras variables
//Agregamos los elementos al array separados por coma
let personasDeLaCH31 = ["Felipe", "Jose Angel", "Marco",
"Evelyn", "Alejandro"];

//Podemos almacenar cualquier tipo de dato en un array
let cosasRandom = ["Pelota", 31, true, undefined, null];
console.log(typeof(cosasRandom)); //object
console.log(typeof(personasDeLaCH31));
```



//todos los arreglos son objetos, sin importar si albergan datos del mismo tipo o no

// no es lo mismo el index y la cantidad de elementos, ya que el indice inicia desde el cero ya que para la computadora el 0 es muy importante  
//hay 5 elementos y su indice de posición van del 0 al 4

```
let pacientes=[];
let dentista=["Dr. Smith","Dra. Garcia", "dr.Hous
dentista[3]="similares";
console.log(dentista[3]);
dentista[8]="Dr. strange";
console.log(dentista);
```



se guardan como vacío igual como si quiero mostrar un elemento de un índice mayor al

Si intentamos asignar el valor de una variable que excede el límite de los índices si se asigna y los demás espaciios

numero de elementos puede mostrar un undefined  
Momentos donde no combiene esta caracteristica

```
71 dentistas[3] = "Similares";
72 dentistas[8] = "Dr. Strange"; //undefined por Dr.Strange
73
74
75 console.log(typeof(dentistas[7]));
```

let productos = [0, 1, 3, 4, 5]

NOS MOSTRARIA UN ESPACIO  
VACIO EN NUESTRA PÁGINA

```
//metodos de array
let listaDelSuper = ["gansitos", "Cocas", "Arroz", "atún",
"verduritas"];

//metodo length para conocer la longitud del array
console.log("La cantidad de elementos de mi array es de ", +
listaDelSuper.length);

//Método push() para poner elementos al final de mi array
listaDelSuper.push("jabón para ropa", "jabón para trastes");

console.log(listaDelSuper);
//metodo pop() para eliminar elemento del final del array
listaDelSuper.pop(); //siempre debe de llevar parentesis ya que
es una función
console.log(listaDelSuper);
```



Método push() para agregar  
elementos al final  
y pop para eliminar el último

```
La cantidad de elementos de mi array es de 5
arreglosYBucles.js:56:9
▶ Array(7) [ "gansitos", "Cocas", "Arroz", "atún", "verduritas",
"jabón para ropa", "jabón para trastes" ]
arreglosYBucles.js:61:9
▶ Array(6) [ "gansitos", "Cocas", "Arroz", "atún", "verduritas",
"jabón para ropa" ]
arreglosYBucles.js:64:9
```

## Unshift, shift, indexOf

```
//tenemos métodos para agregar valores al final al inicio o en medio,
pero si agregamos algo en medio o al final todos los índices cambian
para JS(se ejecuta desde el lado del cliente) no le importa los
recursos que eso necesite pero Java no deja hacer eso
// osea que el arreglo inconsistente

//Agregar un elemento al principio del array con método unshift()
listaDelSuper.unshift("sabritones");
console.log(listaDelSuper);

//Eliminar el primer elemento del array shift elimina el elemento al
principio del array
listaDelSuper.shift();
console.log(listaDelSuper);

//Saber la posición de las verduritas
console.log(listaDelSuper.indexOf("verduritas"));
// no es lo mismo Verduritas que verduritas estos métodos son case
sensitive
```

Solida

```
▶ Array(7) [ "sabritones", "gansitos", "Cocas",
"Arroz", "atún", "verduritas", "jabón para ropa" ]
arreglosYBucles.js:70:9
▶ Array(6) [ "gansitos", "Cocas", "Arroz", "atún",
"verduritas", "jabón para ropa" ]
arreglosYBucles.js:74:9
4
arreglosYBucles.js:78:9
```

```

//método splice para agregar, eliminar o reemplazar elementos
//nombreArreglo.splice(inicio, cantidadElementosAEliminar,
elementoAInsertar1, elementoInsertar2)
/*
inicio: a partir de donde va a iniciar a sobre escribir los
elementos
cantidadDeElementosAEliminar: cuantos elementos se elimina

*/
console.log(listaDelSuper);
listaDelSuper.splice(2,0,"Cacahuates", "Manzana");//no borra
nada y solo agrega valores

console.log(listaDelSuper);
//eliminamos y modificamos

listaDelSuper.splice(5,2, "platano");
console.log(listaDelSuper);

```

Salida

```

▶ Array(6) [ "gansitos", "Cocas", "Arroz", "atún",
"verduritas", "jabón para ropa" ]
arreglosYBucles.js:89:9

▶ Array(8) [ "gansitos", "Cocas", "Cacahuates",
"Manzana", "Arroz", "atún", "verduritas", "jabón para
ropa" ]
arreglosYBucles.js:92:9

▶ Array(7) [ "gansitos", "Cocas", "Cacahuates",
"Manzana", "Arroz", "platano", "jabón para ropa" ]
arreglosYBucles.js:96:9

```

```

98 //método .map()
99 let numeros= [1,2,3,4,5];
100
101 //crear otro arreglo con numeros multiplicados x2 [2,4,6,8,10]
102
103 //toma cada valor el arreglo y lo mete en la función
104
105 let numerosPorDos=numeros.map(function(numero){
106   return numero*2;
107 })
108 console.log(numeros);
109
110 console.log(numerosPorDos);

```

Salida

```

▶ Array(5) [ 1, 2, 3, 4, 5 ]
▶ Array(5) [ 2, 4, 6, 8, 10 ]

```

```

let pacientesConsultorio=[{
  nombre:"Felipe",
  edad: 31,
},
{
  nombre: "Fatima",
  edad: 26,
},
{
  nombre: "jesus",
  edad: 51,}];

//Función para agregar el texto "Necesita atención especial"
function agregarEstadoSalud(paciente){
  let estado="Saludable";

  //si el paciente tiene más de 40 años
  if(paciente.edad>40){
    estado="Necesita atención especial";
  }
  return{
    ...paciente,//Copia del paciente
    estadoSalud:estado,//para agregar un nuevo campo o variable (como la edad o el nombre)
  }
}

//Vamos a aplicar la función en cada elemento del arreglo con el map
let pacientesConsultorioConEstado=pacientesConsultorio.map(agregarEstadoSalud);//cuando la mandamos
a llamar a partir del .map no necesitamos poner los parentesis

console.log(pacientesConsultorio);
console.log(pacientesConsultorioConEstado);

```

Salida

```

▼ Array(3) [ ..., ..., ... ]
  ▶ 0: Object { nombre: "Felipe", edad: 31 }
  ▶ 1: Object { nombre: "Fatima", edad: 26 }
  ▶ 2: Object { nombre: "jesus", edad: 51 }
  ▶ length: 3
  ▶ <prototype>: Array []
arreglosYBucles.js:147

▼ Array(3) [ ..., ..., ... ]
  ▶ 0: Object { nombre: "Felipe", edad: 31, estadoSalud: "Saludable" }
  ▶ 1: Object { nombre: "Fatima", edad: 26, estadoSalud: "Saludable" }
  ▶ 2: Object { nombre: "jesus", edad: 51, estadoSalud: "Necesita atención especial" }
  ▶ length: 3
  ▶ <prototype>: Array []

```

## Ejercicio Individual - Carreritas

En una carrera se tienen los siguientes corredores con su respectiva posición:

- Primer lugar: Roberto
- Segundo lugar: Andrea
- Tercer lugar: Jorge
- Cuarto lugar: Ramiro
- Quinto lugar: Sofía

Después de 3 vueltas, Jorge adelanta a Roberto, Ramiro adelanta a Jorge y Roberto se lesiona y sale de la carrera. Además, Andrea baja una posición, Ramiro mantiene su lugar y el quinto lugar es reemplazado por José.

¿Cómo quedan las posiciones después de esas 3 vueltas?

## Pistas

```
//Jorge adelanta a Roberto (cambiamos las posiciones de Jorge y Roberto)
//Ramiro adelanta a Jorge (cambiamos las posiciones de Ramiro y Jorge)
//Roberto se lesionó y sale de la carrera (eliminamos el primer elemento)
//Andrea baja una posición (intercambiamos las posiciones de Andrea y del último corredor)
//El quinto corredor es reemplazado por José (cambiamos las posiciones del último corredor con José)
//Imprimimos arreglo final
```

Tarea 71/Septiembre/2023

## Método forEach

Por Each() ejecuta la función callback una vez por cada elemento del array en orden ascendente. No es invocada para índices que hayan sido eliminados o no inicializado.

## Cuando forEach y cuando map?

Aunque parezcan muy similares, fueron creados para distintas cosas el método map se utiliza para hacerle una transformación o cambio. genera una copia de elemento para hacerlo

(*ForEach se suele utilizar para hacer impresiones del arreglo*)

## Método sort

El método sort es una de las funciones globales que nos permiten manipular los elementos de un array. Sirve para ordenar los elementos según un criterio, criterio que se puede definir con el uso opcional de un callback. Si usamos el método sort sin el uso de un callback este ordenará los elementos en orden ascendente por defecto.

```

14 //la variable paciente está declarada e inicializada (tiene un nombre, y tiene un valor a pesar de que su arreglo esté vacío)
15
16 //Los índices del arreglo están undefined porque no hay elementos pero si hay espacios disponibles
17 let pacientes=[];
18 let citas=[];
19
20
21 function registrarPaciente(nombre,edad){
22     //Crear un array dentro de otro array
23
24     let paciente = {
25         nombre,
26         edad,
27         citas: []
28         //el uso de ":" es porque ya estamos utilizando un = dentro de la misma sentencia (let paciente) se llama escape de caracteres
29     };
30 }
31

```

Lo veremos mejor  
Con objetos y  
objetos JSON.

```

//Crear un array dentro de otro array
let paciente = {
    nombre,
    edad,
    citas : [
        citasManiana = [
            citasManianaAntesDeLas9 = [],
        ],
        citasTarde = [],
    ]
};

}

```

Via vez que ya usamos  
podemos usar cuantos  
quieramos  
||, + =

```

registroCitasYPacientes.js > registrarPaciente
20
21 //Los índices del arreglo están undefined, porque no hay elementos pero si hay espacios disponibles
22 let pacientes = []; //pacientes = definido
23
24
25 // Función para registrar un nuevo paciente
26 function registrarPaciente(nombre, edad){
27
28     //Crear un array dentro de otro array
29     let paciente = {
30         nombre,
31         edad,
32         citas : [] vacío
33     };
34
35     //Agregar paciente al array
36     pacientes.push(paciente);
37
38
39 }
40
41 console.log("Felipe el 'amante numero 1' de los chilaquiles verdes");

```

- ① Pedir datos (invocación)  
- nombre  
- edad
- ② Almacenar en variables { }
- ③ Par de datos se guarde en el arreglo

```

// Función para registrar un nuevo paciente
// Paso 1. Ejecución de mi función. Pido dos datos
function registrarPaciente(nombre, edad){

    //Crear un array dentro de otro array
    //Paso 2. Guardo cada par de datos (nombre y la edad en una variable llamada paciente)
    let paciente = {
        nombre,
        edad,
        citas : []
    };

    //Agregar paciente al array
    //Paso 3. Ya que tengo el par de datos, Le hago un push a mi array para guardar al paciente
    pacientes.push(paciente);
    return paciente;
}

```

no estamos  
mandando la  
variable solo le  
mandamos un dato al ultimo

(let i actúa como global)  
(al no estar dentro  
de algún bloque de  
código)

```

        console.log("Edad del paciente: " , paciente.edad); //31
        console.log("Citas registradas: ");
    }

//Registrar un paciente desde la invocacion de la funcion
registrarPaciente
let paciente1 = registrarPaciente("Felipe", 31);
let paciente2 = registrarPaciente("Naruto", 20);
let paciente3 = registrarPaciente("Dr. Simi", 50);

//Registrar La cita de un paciente desde la invocacion de la
funcion registrarCita
registrarCita(paciente1, "2023-09-12", "1:00 PM");

//Mostrar informacion del paciente
mostrarInfoPaciente(paciente1);
mostrarInfoPaciente(paciente2);
mostrarInfoPaciente(paciente3);

```

Citas registradas:  
Nombre del paciente: Naruto  
Edad del paciente: 20  
Citas registradas:  
Nombre del paciente: Dr. Simi  
Edad del paciente: 50  
Citas registradas:

▶ GET http://127.0.0.1:55... [HTTP/1.1 404 Not Found 0ms]

```

60
61 // Funcion para mostrar La informacion del paciente con su cita
62 function mostrarInfoPaciente(paciente){
63     console.log("Nombre del paciente: " + paciente.nombre);
64     console.log("Edad del paciente: " , paciente.edad); //31
65     console.log("Citas registradas: ");
66
67     //forEach para imprimir citas (parameter) indice: any
68     paciente.citas.forEach((cita, indice) =>{
69         console.log("Indice: " + indice + " Fecha: " + cita.fecha +
70             "Hora: " + cita.hora)
71     });
72
73
74 //Registrar un paciente desde La invocacion de la funcion
75 registrarPaciente
76 let paciente1 = registrarPaciente("Felipe", 31);
77 let paciente2 = registrarPaciente("Naruto", 20);
78 let paciente3 = registrarPaciente("Dr. Simi", 50);
79

```

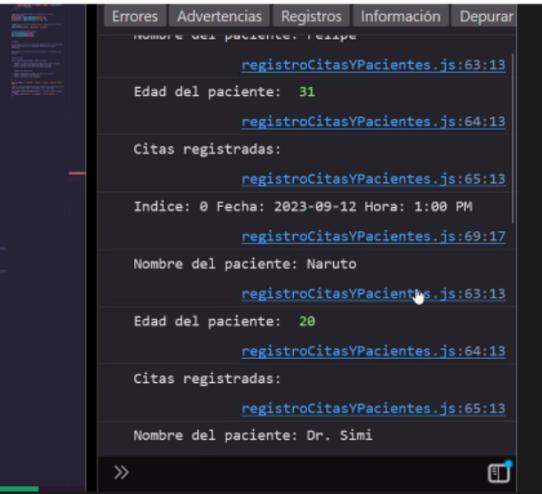
```

// Funcion para mostrar La informacion del paciente con su cita
function mostrarInfoPaciente(paciente){
    console.log("Nombre del paciente: " + paciente.nombre);
    console.log("Edad del paciente: " , paciente.edad); //31
    console.log("Citas registradas: ");

    //forEach para imprimir citas
    paciente.citas.forEach((cita, indice) =>{
        console.log("Indice: " + indice + " Fecha: " + cita.fecha +
            "Hora: " + cita.hora)
    });
}

//Registrar un paciente desde La invocacion de la funcion
registrarPaciente
let paciente1 = registrarPaciente("Felipe", 31);
let paciente2 = registrarPaciente("Naruto", 20);
let paciente3 = registrarPaciente("Dr. Simi", 50);

```



# Manejando el DOM

12 de septiembre 2023

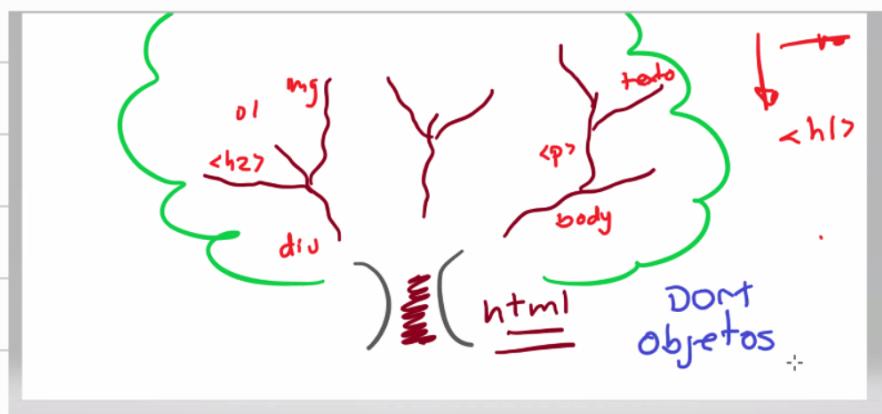
## Objetivos

Al final de la sesión, seré capaz de:

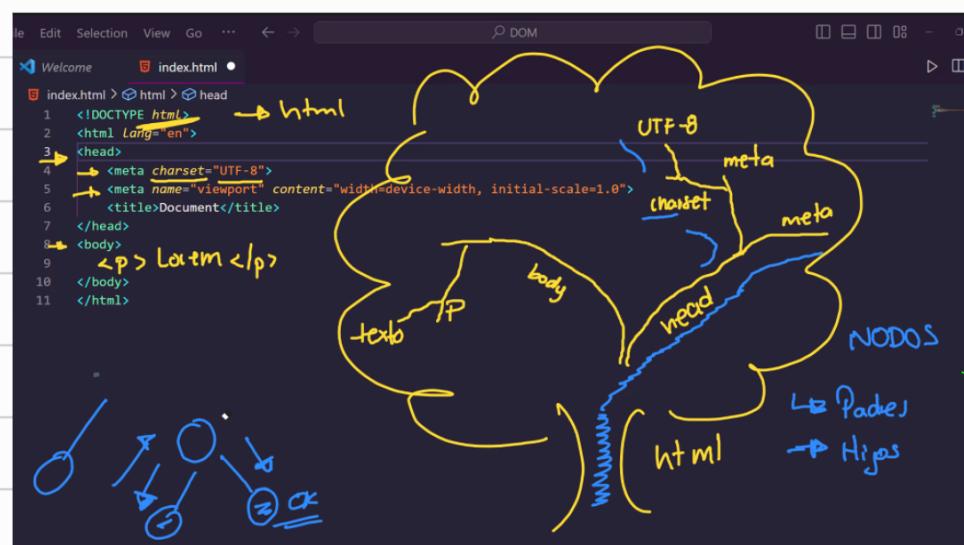
- Explicar el **DOM**
- Usar la **API DOM** para acceder a elementos HTML (`document.getElementById`)
- **Acceder** y **modificar** una clase y atributos de objeto DOM
- **Ocultar** y **mostrar** elementos HTML usando la propiedad de visualización
- **Bonus: Leer** y **validar** datos de **formulario** usando JavaScript

(Como en el video de youtube  
de hace tu propio menú de  
hamburguesa)

Document  
Object  
Model

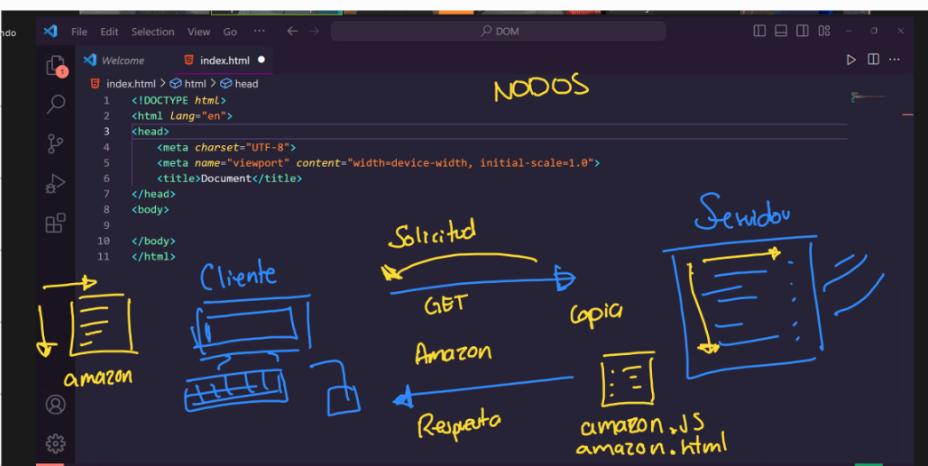


Un modelo  
donde cada ele-  
mento del HTML (La  
Base de todo) es visto  
como un objeto



después podemos hacer  
objetos a partir de  
alguno de sus objetos  
padre.

Tenemos un objeto que alberga  
Otros objetos



En una e-commerce el DOM ayuda a solo tener una plantilla de productos, de imagen de un artista.



## Tarea Tipos de nodo en el DOM

El DOM (Document Object Model) es un concepto que nos permite manipular los elementos de un documento HTML directamente desde nuestro código de JS.

### Nodo en el DOM

El Dom es un punto de conexión o contacto entre varios elementos del archivo html. Un nodo en el DOM es cada una de las etiquetas HTML del árbol jerárquico. Un nodo es una etiqueta HTML sobre la que vamos a poder realizar operaciones de lectura y escritura.

Los principales tipos de nodos son: document, element, text y otros

Document: es el nodo raíz a partir del cual derivan el resto de nodos

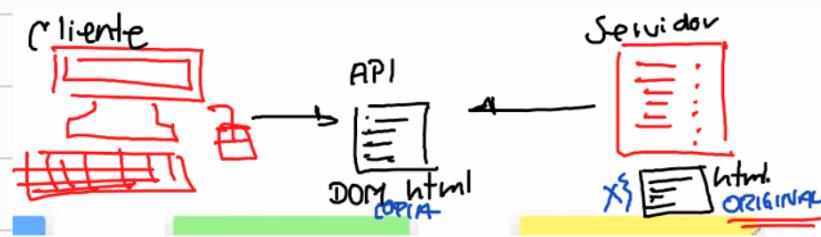
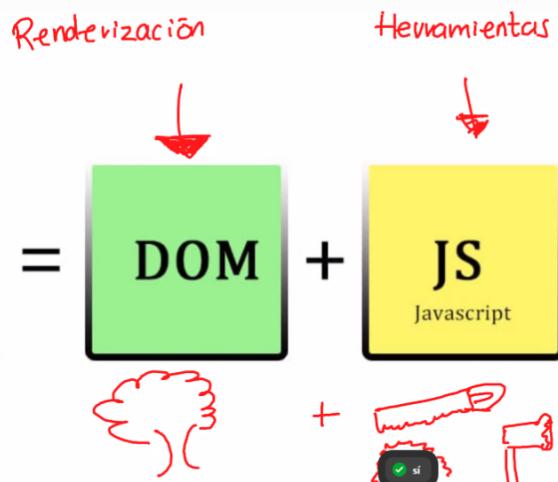
Element: Son los nodos definidos por etiquetas HTML (una etiqueta div genera un nodo)

Text: el texto dentro de un nodo element se considera un nuevo nodo hijo de tipo text (tambien se usan nodos tipo text para referirse a espacios vacíos o saltos de linea)

Attribute: Los atributos de las etiquetas definen nodos. Aunque trabajando con JS se le considerará información asociada al nodo tipo element

**Comentarios y otros:** Los comentarios y otros elementos como las declaraciones doctype en cabecera de los documentos HTML generan nodos.

## API DOM



En cuanto le haces modificaciones con las herramientas pasa de ser un DOM a ser una API

## Herramientas

El dom es el árbol en bruto y JS serán las herramientas para darle forma

No vemos el documento original vemos una copia que podemos manipular y que al refresh volveremos a recibir la misma copia de la original

## Leer nodos

### Métodos tradicionales (se utilizan en versiones más antiguas de JS)

- document.getElementById
- document.getElementsByTagName
- document.getElementsByClassName

### Métodos modernos (más cómodos y prácticos)

- document.querySelector
- document.querySelectorAll

NO confundir con las querys de CSS

```
let columnas= document.getElementsByClassName("col");
console.log(columnas);
```

```
▼ HTMLCollection { 0: div.col, 1: div.col, length: 2 }           manipulacionDOM
  ▶ 0: <div class="col">
  ▶ 1: <div class="col">
  length: 2
  ▶ <prototype>: HTMLCollectionPrototype { item: item(), namedItem: namedItem(), length: Getter, ... }
▶ GET http://127.0.0.1:5500/favicon.ico
```

Con el modelo DOM cada elemento se maneja como un objeto

Usaremos el atributo value para extraer su valor

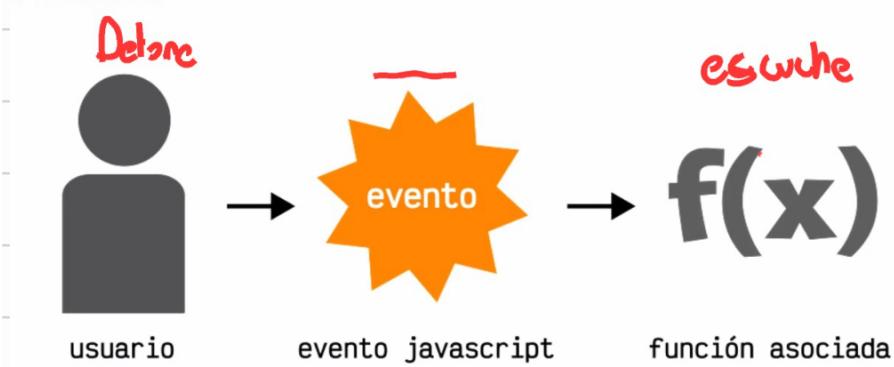
S: usamos const igual podremos hacer cambios en el objeto ya que se recibe una referencia

# Resumiendo lo anterior.

## Manipulación DOM

- Que es el DOM
  - Para que nos sirve
  - Como se el DOM se convierte en API DOM cuando le pongo JS
- 
- Leer elementos (Read)
    - Metodos tradicionales
      - getElementById
      - getElementsByTagName
      - getElementsByClassName
    - Metodos actuales
      - querySelector
      - querySelectorAll
  - Crear elementos (Create)
    - createElement (etiquetas)
    - createTextNode (textos)
    - createComment (comentarios)
  - Poner elementos (append)
    - append
    - appendChild
  - Modificar elementos (Update)
    - Modificaciones sobre el texto
      - innerHTML
      - textContent
  - Borrar elementos (Delete)

## Eventos



# Fetch API

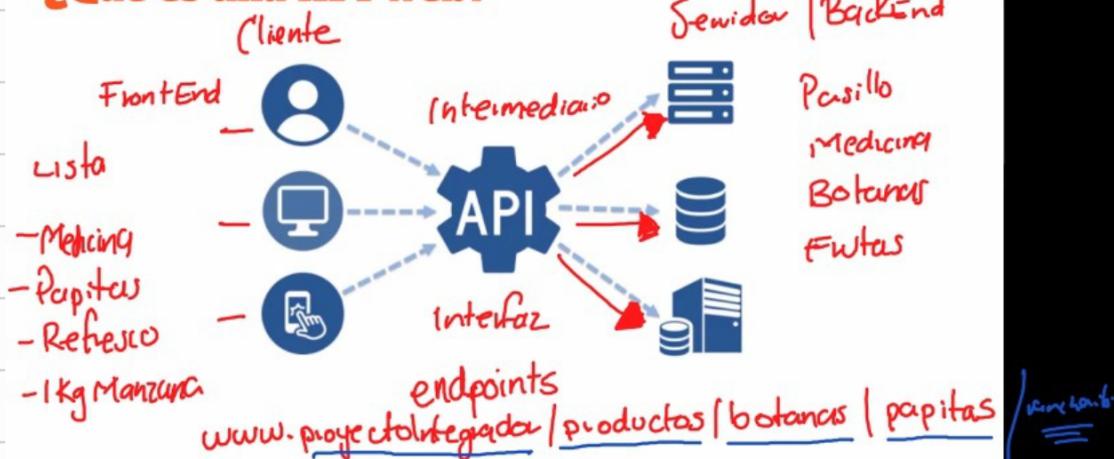
Conectando con servidores

14/Septiembre/2023

## Objetivos

- Explicar la API web de JavaScript
- Explicar la diferencia entre sincrónico y asíncrono
- Leer y usar la documentación de las API web de JavaScript
- Usar la Fetch API para realizar una solicitud GET HTTP.
- Usar la Fetch API para realizar una solicitud POST HTTP.
- Usar las promesas de usuario JavaScript para hacer solicitudes HTTP usando Fetch
- Usar la API de almacenamiento web para guardar datos
- Utilizar las herramientas de desarrollo para verificar los datos de almacenamiento con la API de almacenamiento
- Usar las herramientas de desarrollo para verificar y supervisar la solicitud HTTP

## ¿Qué es una API web?



## Asincrónico vs Sincrónico



se usa para las conexiones a bases de dato para que si no carga la base de datos pueda continuar con las demás tareas

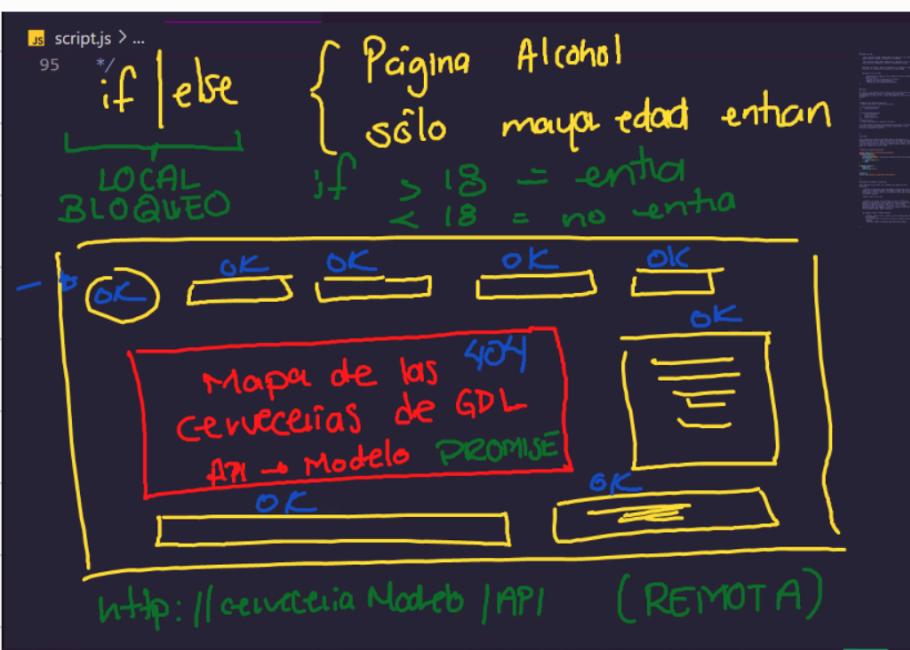
```
/*
console.log("inicia mi operación asíncrona");
//Ejemplo de JavaScript
function dosAsync(){
  console.log("uno");
  setTimeout(function(){// setTime para ejecutar esta función después de 3 segundos
    console.log("dos");
  },3000);
}

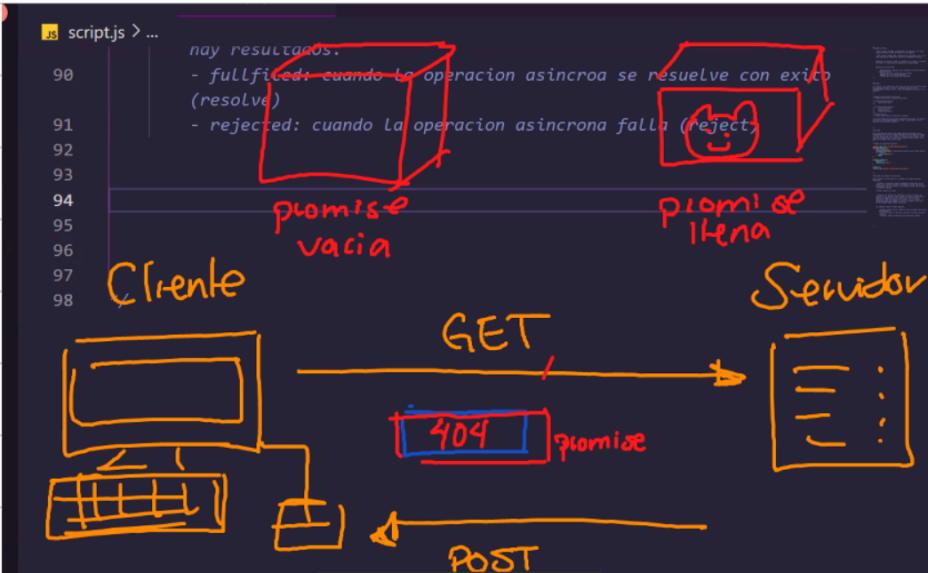
function unoAsync(){
  dosAsync();
  console.log("Tres");
}

unoAsync();
console.log("termina mi operación asíncrona");
```



Se pone a realizar otras operaciones en lo que se cumple el tiempo seteado en `setTimeout` en milisegundos





la promesa es un objeto donde el servidor va a ingresar el objeto que le pidamos

```

// usando promesas
// función especial para consumir APIs y manejar promesas

// instrucción de conexión a servidor
fetch("https://fakstoreapi.com/products/1")

// dos escenarios (obtengo respuesta o no obtengo respuesta)

.then(datos=>{//cuando la promesa se resuelve, ejecuto esta función
    console.log(datos);
    return datos.json(); //convertir la respuesta

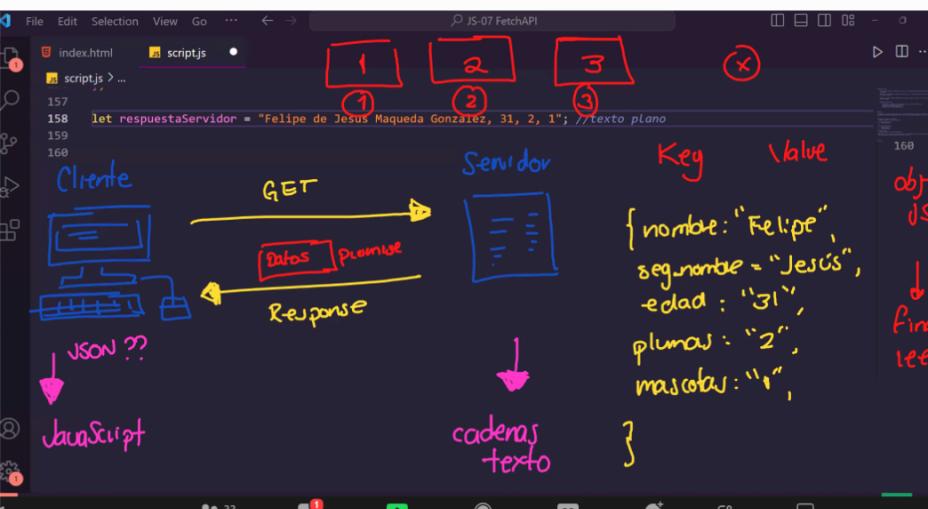
})// si encuentra el dato nos manda la caja con el valor o el objeto
deseado
.catch(error=>{
    console.log("error, no encontramos el producto");
    console.log(error.message);
})// en caso de que no se encuentre te regresa la caja con algún mensaje,
importante que la caja no llegue vacía para saber de donde viene el error

```

El error es un objeto de  
de retorno en caso de error

array  
 ↳ map();  
 ↳ function multiplicar();

promise  
 ↳ then();  
 ↳ function Respuesta();



index.html    JS script.js

script.js > ...  
183

JSON.parse  
POST

JSON  
producto {  
    }  
    }

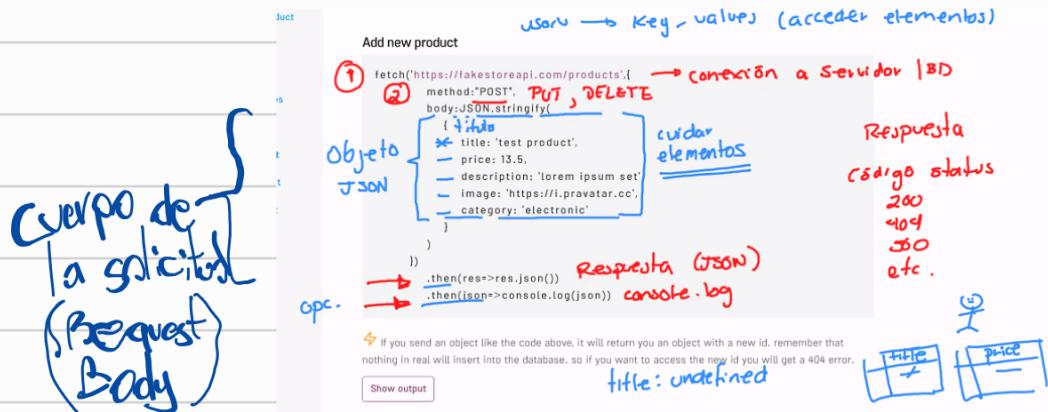
string

JSON.stringify  
producto

1  
2  
...

script.js > let pacienteServidor  
3     console.log(paciente);  
4  
5     //Necesito convertirlo a una cadena de texto para que el servidor  
6     //Lo Lea  
7  
7     let pacienteStringificado = JSON.stringify(paciente);  
8     console.log(pacienteStringificado);  
9  
0  
1     let pacienteServidor = '{"nombre": "Felipe", "edad": 31,  
2        "estatus": "Registrado"}';  
3  
4  
4     let pacienteJSON = JSON.parse(pacienteServidor);  
5     console.log(pacienteJSON);  
6  
7     //Si mando un JSON al servidor, lo stringifico  
8     //Si recibo un string del servidor, lo parseo  
  
                //Objeto JSON que voy a mandar a un servidor  
let paciente = {  
    nombre: "Felipe",  
    edad: 31,  
    estatus: "Registrado"  
}  
  
    console.log(paciente);  
  
    //Necesito convertirlo a una cadena de texto para que el servidor  
    //Lo lea  
  
    let pacienteStringificado = JSON.stringify(paciente);  
    console.log(pacienteStringificado);  
  
    let pacienteServidor = '{"nombre": "Felipe", "edad": 31,  
        "estatus": "Registrado"}';  
  
    let pacienteJSON = JSON.parse(pacienteServidor);  
    console.log(pacienteJSON);

Fetch: sirve para hacer una conexión a un servidor  
como puede tomar una serie de tiempo se pueden utilizar las promesas para recibir el dato despues



cuando queremos hacer un metodo post debemos de especificar que tipo de metodo es (post put delete ) el unico que no se tiene que especificar es get

Cuando se trabaja con un objeto JSON no necesitamo tener comillas en el nombre de los atributos del objeto

## Como va a registrar un producto?

```
// Metodo POST para enviar un nuevo producto a nuestra BD de La
// FakeStoreAPI

buttonEnviarProducto.addEventListener("click", fetch)

fetch('https://fakestoreapi.com/products',{
    method:"PUT", //especificar el tipo de metodo
    body:JSON.stringify( //instruccion para serializar el
        //cuerpo de mi solicitud (para la interpretacion del
        //servidor)
        {
            id: 1,
            title: inputTitulo.value,
            price: inputPrecio.value,
            description: inputDescripcion.value,
            image: inputImagen.value,
            category: inputCategoria.value
        }
    )
    .then(res=>res.json()) //metodo para la respuesta
})
```

# LocalStorage API

hay APIs que pueden guardar información en un entorno local  
(en el pc)

## Almacenamiento de sesión

LocalStorage

- persistente

- carritos de compra
- Formularios de google

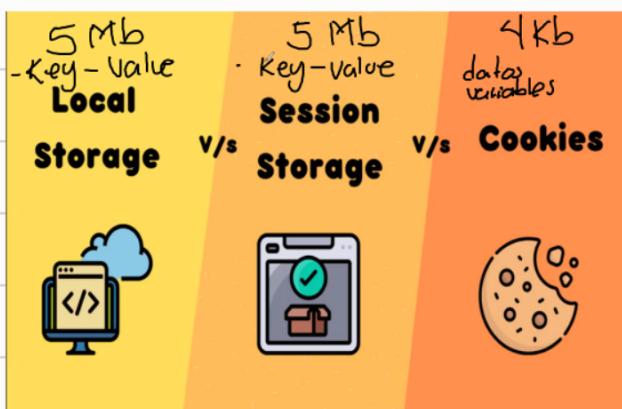
Sesión

- Volátil

La información se guarda en el navegador

Como cuando te conectas a la app del banco que te saca a los minutos de inactividad

## Diferencias entre:

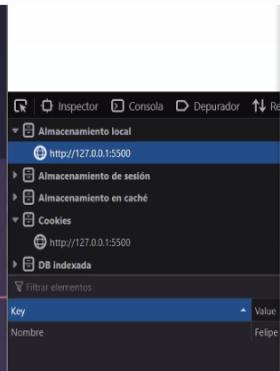


La información se sincroniza a la ligra del buscador

La información se manda en pares de clave valor

La información que pongamos en un almacenamiento Local se guarda indefinidamente hasta que sea eliminada por código o borrada del navegador.

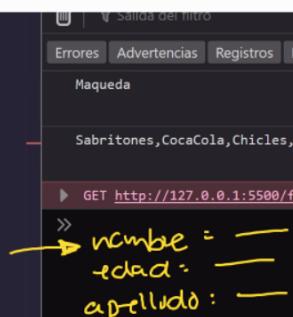
```
/*
//Guardando mi primer dato en localStorage
localStorage.setItem(key, value)
localStorage.setItem("Nombre", "Felipe");
```



① Inicio sesión BD

```
fetch (URL servidor ... (users));
    .then (response)
    .catch (error)
```

② Guardar en local (session Storage)



Se recomienda utilizar el localStorage durante el primer Fetch de la promesa para seguir iterando con el dato

The screenshot shows a browser developer tools window with the title "JS-07 Almacenamiento WEB". The "Console" tab is selected, displaying the following output:

```
34 // Obtener la información almacenada en mi
35 localStorage
36 //localStorage.getItem("Apellido")
37
38 let Apellido= localStorage.getItem("Apellido");
39 console.log(Apellido);
40
41 //Recuperando el carrito de compras
42 let carrito = localStorage.getItem
43 ("CarritoCompras");
44 console.log(carrito);
45 //borrar un elemento del localStorage
46 //localStorage.removeItem(key);
47 localStorage.removeItem("CarritoCompras");//mismo
48 nombre de la llave con la que guardamos el
49 elemento
50
51 //Conocer el largo o longitud del LocalStorage
52 //localStorage.length()
53
54 let longitudLocalStorage=localStorage.length;
55 console.log(longitudLocalStorage);
```

The console also shows a stack trace for a script named "scriptLocalStorage.js" with the following entries:

- scriptLocalStorage.js:39:9
- coca,sabritones,Chicles,Tortillas
- scriptLocalStorage.js:44:9
- scriptLocalStorage.js:53:9

A specific line in the stack trace is highlighted in purple, showing a GET request for "favicon.ico":

- GET http://127.0.0.1:5500/favicon.ico
- [HTTP/1.1 404 Not Found @ms1]

# Unit test JS-09 pruebas unitarias

## Objetivos

Al final de la sesión, seremos capaces de:

- Explicar las pruebas unitarias
- Describir la importancia de las pruebas unitarias para la Garantía de Calidad del Software
- Escribir pruebas unitarias utilizando JavaScript y el marco de pruebas JEST
- Ejecutar pruebas unitarias y verificar los resultados en JavaScript mediante NPM

Son pruebas para testear el código y ver si nuestro programa funciona

Puntaje

Los frameworks más reconocidos son:  
unos sirven para testear desde el servidor  
otros sirven para testear desde el cliente, y otros ambos.

## Frameworks para pruebas unitarias



## Documentación JEST

**TEST**

```

sum.js file:
function sum(a, b) {
  return a + b;
}
module.exports = sum;

① Documento
} función a probar

Then, create a file named sum.test.js. This will contain our actual test:
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});

② Otro documento
— importar la función
③ Palabras reservadas
  - test
  - message (descriptivo)
  - expect (sum(1,2))
  - toBe (3)

```

Running from command line  
Additional Configuration  
Generate a basic configuration file  
Using Babel  
Using webpack  
Using Vite  
Using Parcel  
Using TypeScript

Para poder hablar de Jest tenemos que hablar NPM  
Sirve para poder generar documentación  
por lo tanto podemos encontrar bastante documentación  
Necesitaremos descargar npm para poder utilizar JEST

```
JS-09 pruebas Unitarias > tests > calculadora.test.js > ...
1 const suma= require("../calculadora");// esto es para decir que vas a requerir los datos que exporta el archivo calculadora.js
2
3 console.log(suma);
4
5
6
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
powershell + ...
```

```
at onResult (node_modules/@jest/core/build/TestScheduler.js:133:18)
at node_modules/@jest/core/build/TestScheduler.js:254:19
at node_modules/emittery/index.js:363:13
  at Array.map (<anonymous>)
at Emittery.emit (node_modules/emittery/index.js:361:23)
```

```
Test Suites: 1 failed, 1 total
Tests:       0 total
Snapshots:   0 total
Time:        3.189 s
Ran all test suites.
```

```
PS C:\Users\PP\Desktop\Generation\intro-web 7 JavaScript\JS-09 pruebas Unitarias>
```

Ln 3, Col 1 Spaces: 4 UTF-8 CRLF {} JavaScript Go Live TabNine: ⚡

manda un  
fallido porque no  
hacemos un  
test

File Edit Selection View Go ... ← → ⌂ Untitled (Workspace)

EXPLORER ... calculadora.js package.json calculadora.test.js ●

OPEN EDITORS 1 unsaved

UNTITLED... 📁 🗃 🗃 🗃

JS-09 pruebas Unitarias > tests > calculadora.test.js > ...
19 test('Debe multiplicar numero de a\*b',() => {
20 expect(calculadora.multiplicacion(10,10)).toBe(100); //con expect hacemos la prueba
21 // si es lo esperado, mientras que con el metodo toBe decimos que resultado esperamos
22 }
23 //prueba unitaria division
24 test('Debe dividir numero de a/b',() => {
25 expect(calculadora.division(10,10)).toBe(1); //con expect hacemos la prueba para ver si es lo esperado, mientras que con el metodo toBe decimos que resultado esperamos que sea

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
PS C:\Users\PP\Desktop\Generation\intro-web 7 JavaScript\JS-09 pruebas Unitarias> npm run test
Debugger attached.
```

La linea de codigo que tenemos que utilizar para correr el test es npm run test