

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 cuales 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)

```

Sí 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

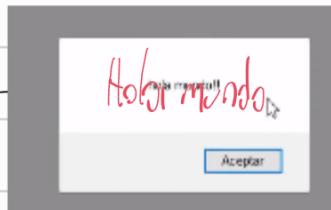
Podemos agregar código utilizando la etiqueta `<script>` en el HTML

```

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 JAVASCRIPT? 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 lineas 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

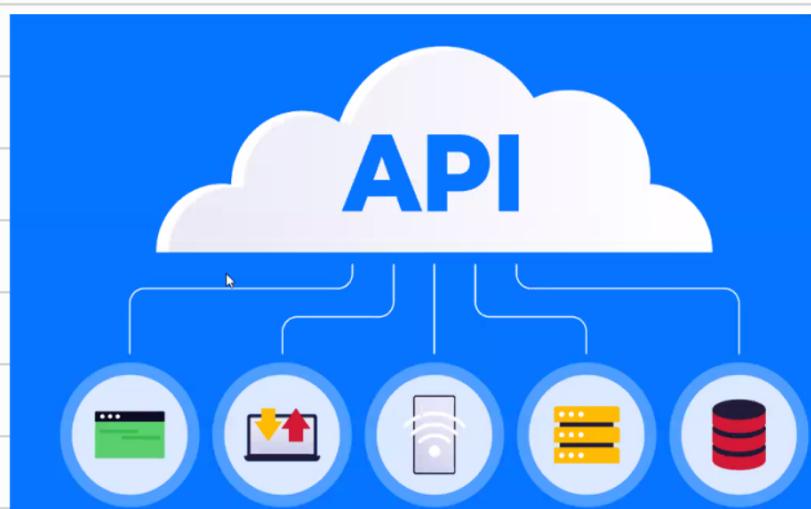
JS puede usarse en muchos lados

Java Enterprise edition

Versión para proys grandes

podemos agregar animaciones con JavaScript

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

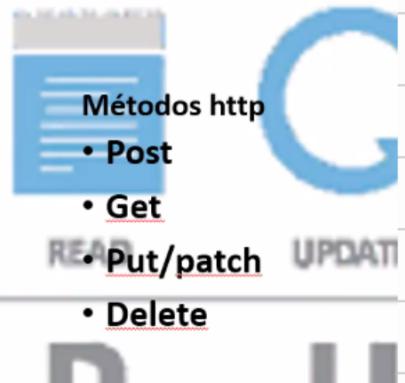


API: Delimita la interacción

que puede tener el JSvar 'o.

Hay APIs para todo

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

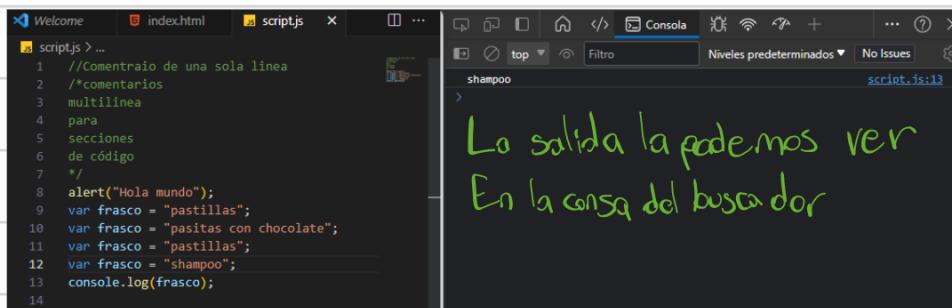
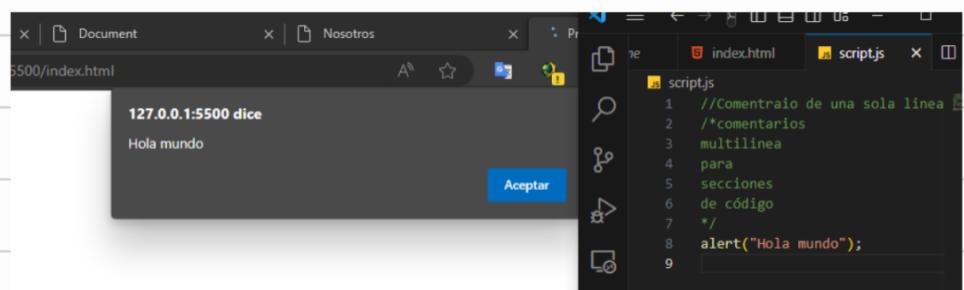


Ejemplos programando

```
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>
```

Hola mundo con JS

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



```
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

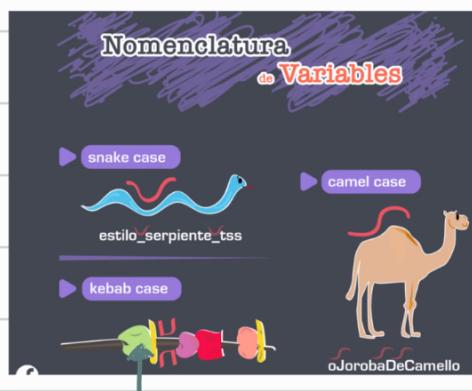
Declarar e invocar funciones

```

//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 pone 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();

```

```

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 farenheit

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