/*La asignación de las variable dependerá del uso que se

le darán en el desarrollo del proyecto. Se usara la convencion (camellCase),

se inicia la variable con la primera letra en minuscula y la palabra sigiente sin

espasio con la primera letra en Mayuscula.

Cabe destaca que hay tres altenativas para inicializar variables

var

let

const

OPERADOR DE ASIGNACIONES

var a; var b = 2 La variable (a) esta basia en console.log dara

error undefind

la variable (b) mostrara la asignacion, La variable (a) ya esta declarada

en el proyecto, para agregarle un valor usamos el signo de asignacion =

También podemos asignar valores de una variable a otra variable.

Ej. a = 5, b = a veremos que el valor de las variables son iguale.

Otra alternativa de cómo podemos asignar el valor de una variable a otra. Declaramos

nuevamente las variables a y le asignamos un valor en este ej. vamos a cambiar

la segunda línea y declaramos la variable (b) bacía y la asignación la podemos hacer

en otra línea distinta del proyecto, es decir cuando la necesitemos.

Para incrementar una variable solamente en 1 la sintaxis más concisa seria

Variable→ a = 25; a = ++; lo mismo sucede cuando restamos 1, a --; pero si

tenemos que realizar operaciones con valores mayores que uno, en ese caso sería,

variable→ a = 23 y aplicamos la siguiente regla (variable operador asignación valor)

a += 5; a pasa a valer 28. Esto se aplica a cualquier valor, entero o decimales EJ.

var ventasDiarias = 13567.34; ventasDiarias += 345.67; ventasDiarias pasa

se incremento y pasa a valer 13913.01. Asi como podemos incrementar el valor

de una variable tambien podemos reducir ese valor, mutiplicar o dividir sigiendo la regla nombrada → (variable operador asignación valor) Ej. de * y /

Var salario = 45000 lo incrementamos salario *=5 y lo reducimos a la mitad

salario /= 2;

Variables con cadenas de caracteres. Las cadenas de caracteres se definen rodeandolas con comillas var miNombre = "Manuel"; ¿Como podemos incluir parte de la cadena entre comillas? Eso se denomina secuencias de Escape: hay dos formas para lograr eso, 1) se antepone una barra invertida antes de cada comilla dentro de la cadena Ej1→ var = "Manuel \"Manu\""; Ej2→ Se inicia la cadena con comillas simles''y el texto a destacar con comillas dobles""Ej2→ var = ´Manuel "Manu"´; Otras secuencias de escapes muy usadas son: Mostrar la barra invertida en la cadena→ \\ o lograr un salro de linea en la cadena→ \n. Concatenar cadenas de caracteres, para eso se usa el signo + Ej→ miNombre = "Manuel" + " Orozco"

Concatenar variables con cadena de caracteres:Ej→ var verbo = "programar";

var mensaje = "Estoy aprendiendo a " + verbo; Tambien se puede segir concatenando la cadena despues de la variable. Ej. sigiente linea,

mensaje = "Estoy aprendiendo a " + verbo + " con freeCodeCamp";

Tambien podemos concatenar dos variables de una forma mas concisa con +=

var mensajeFinal = "El invierno llego"; Var = " con un frío muy cruel";

esto podria ser en una situacion real que esa parte del mensaje se personalisada o que el usuario tenga que ingresar algun valor para tener la cadena completa.

Longitud de una CdC Esto es muy comun en programacion Ej→ var miCadena = "A";

Para aberiguar cuantos caracteres tiene una cadena con console.log() seria, console.log(miCadena.length) Esta propiedad→lenght nos va a permitir saber cuantos caracteres tiene la cadena.

Nota: Los espacios se cuentan como un caracter.

Notación de corchetes en JS, Esta es una notación que nos permite acceder a los caracteres de una cadena. Definimos var lenguajeDeProgramacion; lenguajeDeProgramacion = “Javascript”; Esa cadena tiene una estructura interna que nos va a permitir acceder a sus caracteres, veamos un diagrama de EJ.

Cadena: → J A V A S C R I P T

Índices:→ 0 1 2 3 4 5 6 7 8 9

c/u de esos caracteres tiene un índice asociado y podemos usar ese índice para acceder a ese carácter usando corchetes en console.log(variable[0]);

en este caso es 0 por que es el primer índice.

Inmutabilidad de cadenas→ no se pueden cambiar, es solo lectura.

Nota: En una cadena el último índice es longitud -1 Ej. var miCadena;

miCadena = "JavaScript" consol.log(miCadena.length -1); se muestra el numero del ultimo índice por que no está envuelta por corchetes. Si agregamos los corchetes se mostrara la última letra. E j→ variable[variable.length -1]

También se puede inicializar una variable para agregarla a continuación de length

Var n; n = 4 variable[variable.length variable]

Palabras en Blanco: Concatenar cadenas compuestas de caracteres y variables con espacios en blanco. Declaramos las variables.

```
var miSustantivo  = "perro";
var miAdgetivo =  "negro";
var miVerbo  = "corrió";
var miAdverbio = "rápidamente";
var miEspacio " ";
oracion  =  "El " + miSustantivo + miEspacio + miAdgetivo + miEspacio + miVerbo + miEspacio + miAdverbio + " a la tienda";
```

resultado→ El perro negro corrió rápidamente a la tienda

Siguiente tema: Arreglos (Arrays) Los arrays son estructura de datos que nos permiten agregar múltiples valores en una misma estructura. A diferencia de las cadenas de la caracteres que son Inmutabes, cabe destacar que los arrays si se pueden modificar sus índices

los array son mutables.

Para crear un arrays escribimos corchetes [] y dentro de los corchetes tendríamos los elementos de ese arrays Ej. definimos una variable var miArrays; miArrays = [“Daniel”, 24];

0: "Daniel" El nombre esta en el índice cero

1: 24 La Edad esta en el índice uno

length: 2 El largo de l arrays es de dos

Los arrays son estructuras muy poderosas para trabajar con valores relacionados

Arrays anidados: Algo muy interesante de los arrays es que pueden contener cualquier tipo de datos, caracteres, números, decimales y los podemos anidar usando una variable. Ej.

var listaDeEstudiantes;

listaDeEstudiantes = [["Nora", 18], ["Manuel", 24]];

console.log(listaDeEstudiantes);

var listaDeProductos = [["Camisas", 15.25, "s-32"], ["Zapatos", 30.45, "s-11"], ["Pantalones", 28.50, "s-33"]];

console.log(listaDeProductos); Ahora pasamos al siguiente tema:

Acceder a los elementos de un Arrays:

Definimos una var miAcceso = [10, 20, 30];

Diagrama

Arreglos: 10, 20, 30

Índices: 0 1 2

El diagrama nos muestra que igual que en las CdC teníamos un índice

para c/carácter en el diagrama tenemos un índice para cada elemento. Acceder a los elementos de un array es parte fundamental de la programación. EJ. inicializamos una variable var suma;

suma = miAcceso[0] + miAcceso[1] + miAcceso[2] ;

console.log(suma) //resultado 60

Modificar los elementos de un array: Es otra parte importante en programación

var miModificacion; miModificacion = [10, 20, 30]; accedemos al elemento a modificar de la misma manera. Escribimos la variable, enlazamos entre corchetes con el número del índice del elemento a modificar. EJ→ miModificacion [0] seguido del signo de asignación = seguido del nuevo valor. miModificacion [0] = 40; También podemos acceder al índice [1] y modificar su elemento por una cadena de caracteres. Ej→ miModificacion [1] = "Manuel" Otra cosa que se permite en JS es tomar un elemento del arrays, digamos el índice [3] y asignarle elementos anidados. El→ miModificacion [2] = [1, 2, 3];

Acceder arrays multidimencionale:

Creamos la, var accesosArraysAnidados; para asignar un arreglo multidimensional. Ej.

accesosArraysAnidados = [[1 ,2, 3], [4, 5, 6], [7, 8. 9]];

Para acceder a los elementos mostramos el siguiente diagrama.

Acceso : [[1 , 2, 3], [4, 5, 6], [7, 8. 9]];
↑ ↑ ↑

Índices: 0 1 2
↑ ↑ ↑

Índices internos: [0 1 2] [0 1 2] [0 1 2]

La explicación grafica nos muestra como accedemos al arrays multidimensional

para eso tendremos que usar dos índices. ¿Cómo accedemos al elemento 8?

Primero usamos el índice 2 que nos permite acceder al elemento 3→[7, 8. 9]

y seguidamente el índice 1 que deja el puntero en el num. 8 para tratarlo.

El codigo seria:

```
var accesosArraysAnidados;

accesosArraysAnidados = [[1 ,2, 3], [4, 5, 6], [7, 8, 9] ];

// se muestra el acceso a cada elmento

console.log(accesosArraysAnidados[0]);

console.log(accesosArraysAnidados[1]);

console.log(accesosArraysAnidados[2]);
```

Para acceder algunos de los elementos anidados en c/u de esos array usaremos

el segundo índice que vimos en el diagrama. Digamos que queremos acceder al

elemento 6 de segundo array [4, 5, 6] accesosArraysAnidados[1] [2];

console.log(accesosArraysAnidados[1] [2]); //vemos el elemento 6

Métodos:

Los arrays son estructuras de datos muy poderosas , no solo podemos acceder y

modificar los elementos de un array si no que podemos agregar elementos.

Digamos que tenemos una lista de cadenas de caracteres en este caso son las

estaciones del año a la cual le falta el verano.

var estaciones = ["invierno"; "otoño"; "primavera"];

¿Cómo podemos agregar esa cadena de caracteres al arreglo? Con algo llamado método,

El método es como una función que podemos llamar para hacer algo con un elemento

.push() este método nos permite añadir un elemento al final del arrays

variable→ estaciones.push("verano") //["invierno"; "otoño"; "primavera"; "verano"];

.pop() este metodo nos permite remover el último elemento de un array que nos

entrega un retorno que podemos asignarlo a una variable para su

taratamiento en el proyecto.

var estación = estaciones.pop() → ["invierno"; "otoño"; "primavera"] se extrajo "verano"

.shift() este metodo nos permite remover el primer elemento de un array

estaciones.shift();

["invierno"; "otoño"; "primavera";"verano"] En este array se extrae "invierno"

Lista de compra: (mini proyecto) de una lista de compra.

miListaDeCompras = [["Arroz" 1], ["Aceite" 2], ["Fideos" 3], ["Café" 1]];

CODIGO:

```
miListaDeCompras = [["Arroz", 1], ["Aceite", 2], ["Fideos", 3], ["Café", 1]];
console.log("Voy a comprar " + miListaDeCompras[0][1] + " kg de " +
miListaDeCompras[0][0] + " tambien " + miListaDeCompras[1][1] + " Litros de "
```

```
+ miListaDeCompras[1][0] + " y " + miListaDeCompras[2][1] + " paquetes de " +
miListaDeCompras[2][0] + " por ultimo " + miListaDeCompras[3][1] +
" un paquetes de 1/2 kg de " + miListaDeCompras[3][0]);
```

Resultado en pantalla:

Voy a comprar 1 kg de Arroz también 2 Litros de Aceite y 3 paquetes de Fideos por ultimo 1 un paquetes de 1/2 kg de Café

FUNCIONES:

Con las funciones podemos escribir código que podemos reutilizar en nuestro programa

Declaración → function mostrarMensaje()

```
{
        console.log("Hola Manu);
}
mostrarMensaje();
mostrarMensaje();
```

ARGUMENTOS:

Lo primero que necesitamos saber para una operación, es que operando vamos a usar y que valores vamos a tener.→ Function sumar(**a, b**) lo que envuelven los paréntesis se los denominan **parámetros,** son cualquier nombre que tú le quieras asignar a un valor que se va a recibir o a tomar cuando llamemos a la función. La parte de la función que va entre las llaves se denomina cuerpo. Si vamos a sumar a+b que vamos hacer en el cuerpo de la ¿function? Vamos a tomar los valores de **a** y de **b** esos valores que en estos momentos no existen en el cuerpo de la fuction declaramos la variable suma y le asignamos los parámetros a+ b.

```
function sumar(a, b )
{
  var suma = a + b;
 console.log("El resultado de  " + a " + " b + " es  " + suma);
}
```

Ahora vamos a llamar a la function. Sabemos que tenemos dos parámetros, tenemos que pasar un valor para cada uno. Los valores se asignan en el mismo orden en que aparecen en la lista de parámetros en la definición de la function. Para a es 5 y para b es 3

sumar(5, 3);

Para ilustrar por que las funciones nos ayudan a escribir código que es más conciso y fácil de leer de modo tal que evita la repetición de código. Podemos llamar a esta function varias veces.

sumar(57, 3);

sumar(31, 83);

sumar(11, 43);

Estos valores que estamos pasando se denominan argumentos. Los argumentos son los valores que asignamos a los parámetros. Esos dos conceptos trabajan juntos para permitir que las function sea la herramienta poderosísima que es en programación.

Veamos otra variación que podemos tener cuando llamamos a una function En lugar de escribir los valores directamente en la lista de argumentos →(5, 3)← también podemos asignarles a variables y pasarlas como argumentos. Ej. var x = 5; var y = 3; sumar(x, y); y el resultado sería el mismo. No solo podemos pasar números. Vamos a definir otra función para un ej. de CdC

```
function concatenarTresCadenas(cadena1, cadena2, cadena 3)
   {
     console.log(cadena1 + "  "   + cadena2 + "  " + cadena3);
   }
```

Luego cuando llamemos a la function y debemos pasar un valor para cada una de las cadenas de lo contrario tendríamos un error

```
   concatenarTresCadenas("Estoy", "aprendiendo", "a programar");
```

Resultado en pantalla→ Estoy aprendiendo a programar

Ámbito Global:

Algo muy importante e JS es que no todas las variables pueden usarcé en todas las distintas partes del programa. Tenemos dos tipos de variables GLOBALES y LOCALES.

Una variable con ámbito **global** está definida dentro del programa y no dentro de la function.

Var miVariableGlobal = 5; global por que la definimos dentro del programa y no dentro de una function. Si necesitamos acceder a su valor, usarla en el programa podemos hacerlo.

```
console.log(miVariableGlobal);
```

Y también podemos usarla dentro de una function

```
function miFuncion()

{

console.log(miVariableGlobal);

 }
```

Y también va a estar definida luego de la definición de la function, básicamente en cualquier lugar del programa. Para comprobar que si está definida debemos llamar a la function.

```
function miFuncion();

console.log(miVariableGlobal);
```

AMBITO LOCAL:

Ahora veamos cómo funcionan las variables locales. Estos tipos de variable las definimos dentro de una function que solo se pueden utilizar dentro de esa function.

```
Function miFunction()

{

Var miVariableLocal =  4;

console.loc(miVariableLocal );

}
```

Vamos a llamar a la function para ver que esa variable si está definida

```
miFunction();
```

Pero luego veamos que queremos acceder a esa variable local fuera de la function.

Esto nos dará un error de no definición de la variable

Si declaras una variable dentro de una function solo se podrá usar solo dentro del cuerpo de la function.

Ámbito Local vs Ámbito Global:

Veamos que sucede cuando tenemos una v/local y una v /global con el mismo nombre.

```
 var miNombre = "Nora"
```

```
function mostrarMiNombre()

{

  var miNombre = "Ricardo";

  console.log(miNombre);

}
```

Cuando intentamos verificar la variable llamando la function, vemos que predomina la

v/local sobre la v/global

```
mostrarMiNombre(); //resultado Ricardo
```

Pero si tratamos de mostrar la variable global en el programa, lo podemos hacer y la podemos usar en cualquier lugar del proyecto.

```
console.log(miNombre);
```

RETORNAR UN VALOR:

Las function pueden retornar un valor además de recibir valores con propiedades especiales para poder interactuar con el programa principal.

```
function sumar(a, b)

{

  return a + b; // no lleva coma

}

//si nosotros llamamos a la function

sumar(5, 3); //no se muestra nada, para eso usamos

console.log(sumar(5, 3)); //resultado retorna 8
```

UNDEFINED:

Así cómo podemos retornar un valor específico en una function con returna, también podemos omitir esa sentencia pero aun así la function va a retornar un valor por defecto ese valor va a ser UNDEFINED. Si definimos una function que en vez de retornar el lugar de la suma lo vamos a mostrar en la consola

```
function sumar(a, b)
```

```
{
  Console.log(a + b);
   //return a + b; // no lleva coma
}
```

Como no estamos retornando ningún valor explícitamente

```
console.log(5, +3); // vemos aquí UNDEFINED
```

ASIGNAR UN VALOR RETORNANDO

Cuando una fuction retorna un valor podemos asignarle ese valor a una variable

```
function sumar(a, b)
{
   return a + b; // no lleva coma
}
var resultado  = Sumar(5, 3); //resultado ahora vale 8
console.log(resultado); //se muestra 8
```

Otro ejemplo, podemos definir una function con una cadena de caracteres con nuestra meta.

¿Cuál va a ser nuestra meta? Vamos a retornar una cadena de caracteres

```
 function crearCadenaConMeta(lenguajeDeProgramacion)
{
    return "Mi meta es aprender " + lenguajeDeProgramacion;
}
```

Si llamamos a la function y pasamos la cadena de caracteres “JavaScript” el parámetro va a tener ese valor cuando llamemos a la function y lo asignamos a una variable para su tratamiento

```
 var miMeta = crearCadenaConMeta("JavaScript");
 console.log(miMeta); //resultado: Mi meta es aprender JavaScript
```

Permanece en Fila

Vamos hacer una variación de una cola (queu) es una estructura de datos que nos va a permitir agregar elementos a una fila. Vamos a tener una estructura de datos abstracta, en la cual los elementos se mantienen en orden. Los nuevos elementos se pueden añadir al final de la cola y los elementos previos se retiran al principio de la cola. Vamos a definir una function que se llame próximo en la fila, esa function debe tomar un arreglo, que se va a llamar arreglo←ese va a ser el nombre del parámetro y va a tomar un número que se llamara →elemento ese va a ser el parámetro, ambos van a ser los argumentos. La tarea es agregar el numero al final del argumento →arreglo. Y luego la function debe eliminar l primer elemento del arreglo. Es un ciclo vamos agregarlo al final y si se agrega un elemento significa que el primero debe ser eliminado. La function proximoEnLaFila(){ } debe retornar el elemento que fue removido.

¿Qué es lo primero que queremos hacer cuando llamamos a la function? Agregar el número al parámetro elemento al final del array con el método .push . Luego que agregamos al parámetro del elemento en el array. Debemos eliminar el primer elemento con el método .shitf() no toma ningún argumento los paréntesis vacíos. Y la parte final de lo que debe cumplir la function, es que debe retornar el elemento que fue removido. Recordemos que el método .shift también retorna ese elemento es ahí donde anteponemos return

```
function proximoEnLaFila(arreglo, elemento)
{
        arreglo.push(elemento);//agregar al final del array
        return arreglo.shift() ; //shift es el que va a remover el primer elemento del array
}
```

Definimos una variable

```
var miArreglo = [1, 2, 3, 4, 5,]
```

Queremos ver el estatus de nuestro arreglo antes de llamar a la function. Nota:JSON.stringify() es una funcion muy util que nos muestra los array en la consola en un formato especifico

```
console.log("Antes: " + JSON.stringify(miArreglo));
```

Llamamos a la function, y el primer elemento que toma es el arreglo y ¿cuál argumento pasamos? La variable miArreglo como argumento → se cambia a [1, 2, 3, 4, 5,] y luego que tenemos ese arreglo queremos agregar un elemento que será → 6 El efecto de llamar a esta function es que se elimina el primer elemento →1 y se agrega el último elemento →6

[2, 3, 4, 5, 6]

console.log(proximoEnLaFila(miArreglo, 6));// aquí realizamos el cambio

console.log(proximoEnLaFila(miArreglo, 6));// aquí realizamos el cambio

Luego de esto mostramos después de hacer el cambio

console.log("Después: " + JSON.stringify(miArreglo));

VALORES BOOLEANOS

Verdadero Falso

console.log(true) console.log(false)

Operadores de igualdad

Esto que tenemos entre parentesis se denomina una exprecion, va a evaluar

true o false

console.log(5 == 5) true / consol.loge(6 == 5) false

De igual forma podemos comparar cadenas de caracteres la comparacion tambien

se hace en base de mayusculas o minusculas

console.log("Hola" == "Hola") true / console.log("Hola" == "hola")false

Es importante saber que no se deve comparar array con este operador

console.log([1,2,3]==[1,2,3]); El resultado va a ser false, ¿por qué? porque

no compara los elemento de los array sino que compara si los arreglos en la memoria

representan el mismo objeto.

Operador de igualdad estricta:

Este operador nos permite comparar si ambos tipos de datos son los mismos.

Veamos la diferencia ente igualdad = = e igualdad estricta. = = =

console.log(9 = = 9); // Resultado true

Pero si tenemos dos valores que son de distinto tipos de datos pero que representan el mismo número el resultado también va a ser true. ¿Qué ocurre con este operador?

Antes de realizar la comparación ambos valores se convierten a un tipo de dato común

Por eso es que el resultado nos dice true

console.log(9 = = “9”); // Resultado true

Pero si queremos que la comparación también determine que el tipo de dato es

el mismo o no, es ahí donde usamos el operador de igualdad estricta

console.log(9 = = = “9”); // Resultado es false

Operador de desigualdad

Este operador compara dos valores y retorna true si su valor es distinto y retorna

false si son iguales

console.log(9 != 6)//Resultado true

console.log(9 != 9)//Resultado false

De igual manera sucede con cadena de caracteres, pero en el caso de los array

Podemos intentar comparar array con este operador

console.log([1,2,3] != [1,2,3]);//Resultado es true

pero no los va a comparar en base a sus elementos, sino en base a como están

representados como objetos en la memoria del dispositivo que es algo relacionado a como se almacenan en la memoria

Operador de desigualdad estricta:

En el tema anterior con el operador de desigualdad si nosotros comparábamos si los siguientes valores (1 != “1”) eran distinto el resultado sería **false** porque **!=** ←este operador convierte ambos valores a un tipo de datos común antes de hacer la comparación y detecta que ambos números son iguales. Pero que ocurre con el operador != = ←de desigualdad estricta (1 != = “1”); ahora el resultado sería **true** porque también está comparando si ambos valores son del mismo tipo de datos, no lo convierte a un tipo de dato común.

console.log(1 != "1");// true está mal

console.log(1 !== "1");// false está bien

Operador mayor que:

En javaScipt además de comparar valore si son iguales o diferentes también podemos comparar el valor relativo de los valores. Si un valor es mayor o menor que el otro o menor igual o mayor igual.

Operador mayor que > Digamos que queremos saber si (6 > 5); ←true / (3 > 10); ←false

También podemos comparar cadena de caracteres. El criterio con que JavaScript compara las cadenas de caracteres es según el orden alfabético. console.log("B" > "A"); true ←Estamos diciendo que si la letra “B” es mayor que la letra “A” según el orden alfabético como estamos usando este operador mayor que > Cuando aplicamos a cadena actúa como si estuviera más allá adentro del diccionario. Si tuviéramos otra cadena console.log("A,C,B" > "A","B","C"); //true ←

Como javascript compara esta cadena. Compara cada uno de los caracteres en su secuencia.

A =A / C=B compara **C** viene después que **B** sí. Ahí se determina el orden de la cadena porque el primer par de caracteres desiguale que encuentra y compara define el resultado, de manera tal que es true. Si una cadena tiene más caracteres que otra verificamos→ ("AB" > "A");true← porque tiene más caracteres. Por orden alfabético también podemos comparar palabras ("Mundo" > "Hola");true← Por orden alfabético **M** viene después que la letra **H** en el alfabeto asi a modo de ejemplo comparamos ("M" > "H");true← Otro dato interesante es que podemos usar estos datos de comparación con variables. var az = 15; / var by = 8;

console.log(az > by);//true / console.log(by > az);//false

Mayor o igual que:

Este operador >= nos permite incluir el caso en que ambos valores sea iguales,

(5 > 5); //false (5 >= 5); //true En el caso de cadenas de caracteres también aplica

cuando las cadena sean iguales el valor va a ser true

Menor que: Este operador < nos permite verificar (5 < 6); //true, si el número a la

izquierda es mayor (10 < 8);//false. También se puede usar para cadena de caracteres

siempre la comparación de cadena será por orden alfabético ("A" < "B");true porque

A esta antes que B en el alfabeto. En este ejemplo ("A ,C, B" > "A","B","C");//true←

el proceso va a ser igual que en ejemplos anteriores. También podemos comparar los valores

de variables

Operador menor o igual que: basicamente funciona igual que menor que <

(5 < 5); //false / (5 >= 5); //true.

Operadores lógicos "and"

Estos operadores lógicos nos permiten expresar distintas expresiones para formar

condiciones un poco más elaboradas o más complejas

Tabla de verdad de operador AND←→&&

| x | z | x && tenemos dos expresiones x y z y los unimos x && z

↓ ↓ ↓

↓ ↓ ↓

| T | T | T |Si x es true y z es true toda esa expresión es true

| T | F | F |En cambio si alguno de ellos es false o ambos

| F | T | F |entonces la expresión va a ser

| F | F | F |←false

La operación solo es verdadera cuando ambos operadores son verdaderos.

Operadores lógicos "and" Estos operadores lógicos nos permiten expresar distintas

expresiones para formar condiciones un poco más elaboradas o más complejas

Tabla de verdad de operador Y→AND←→&&←en JS se expresa con &&

| x | z | x && Z |tenemos dos expresiones x y z y los unimos x && z

| ↓ ↓ ↓ |

├----------┤

| ↓ ↓ ↓ |

| T | T | T |Si x es true y z es true toda esa expresión es true

| T | F | F |En cambio si alguno de ellos es false o ambos

| F | T | F |entonces la expresión va a ser

| F | F | F |←false

La operación solo es verdadera cuando ambos operadores son verdaderos.

Operador logico "or"

Tabla de verdad del operador O→OR←→||←en JS se expresa con ||

| x | z | x && Z |tenemos dos expresiones x y z y los unimos x || z

| ↓ ↓ ↓ |

├----------┤

| ↓ ↓ ↓ |En el operador OR si cualquiera de los operando es true entonces

| T | T | T |el resultado es verdadero

| T | F | T |←true

| F | T | T |←true

| F | F | F |←false

La operación solo es verdadera si alguno de los dos operando o ambos son true

Operador logico "not"

Tabla de verdad del operador NO→NOT←→!←en JS se expresa con !

Para: !x

| x | !x|

| ↓ ↓ |

├-------┤

| ↓ ↓ |

| T | F | Si la expresión es true agregando el operador not la convertimos en false

| F | T |Si la expresión es false agregando el operador not la convertimos en true

Setencias Condicionales:

Las condiciónale determinan que rumbo va a continuar nuestro código, en el caso de (if) o si en español siguiendo el ejemplo seria if(condición, la cual si es verdadera){ejecutara lo que esta ente las llaves} o cuerpo de la condicional. También se pueden usar los operadores lógicos→ && || !← para condiciones más elaboradas.

var cond = 5

if (cond > 2) // la condición se cumple

```
{
   console.log("La condición es verdadera"); // ejecuta el código
 }
if((cond > 2) && (cond < 10)); //condiciones más complejas
{
   console.log("La condición es verdadera"); // ejecuta el codigo
 }
var estación = "invierno"
 if(estación == "invierno") // la condición se cumple
{
console.log("¡Si! Me encanta el invierno");// ejecuta el código
}
if (cond < 2) // la condición NO se cumple
 {
    console.log("La condición es verdadera"); // NO ejecuta el código
 }
 console.log("La condición no es verdadera"); // Continua la ejecución saltando el if
```

Clausula "else"

También podemos expandir nuestras condicionales si la condición de la condicional es false

```
if (false){
  console.log("si la condición es true se ejecuta este código");
} else{
console.log("si la condición es false ignora el if y se ejecuta else con este código");
}
```

Clausula else if:

Los condicionales también pueden verificar y manejar varias condiciones, con la cláusula else if que nos permite manejar exactamente eso, manejar condiciones alternativa y decidir por la primera condición que encuentre y sea verdadera→ true. Para el ejemplo es conveniente trabajar con funciones: Esta función va a tomar un

valor como un parámetro y va a determinar si el valor es divisible entre 2 o 3

o si no es divisible entre ninguno.

```
function clasificarValor(valor)
{
  if(valor % 2 == 0)//genera un resto igual a cero es true
  {
    console.log("Divisible entre 2");
  }
    else if(valor % 3 == 0)
  {
    console.log("Divisible entre 3");
  }
  else
  {
    console.log("NO es divisible entre las opciones");
  }
 }
  clasificarValor(2) //caso 1 se ejecuta porque es el primer true que encuentra
  clasificarValor(15)//caso 2 se ejecuta porque caso 1 es false
  clasificarValor(7) //caso 3 se ejecuta porque caso 1 y 2 son false
 var elcasoUno = "caso 1 se ejecuta porque es el primer true que encuentra"
  var elCasoDos = "caso 2 se ejecuta porque caso 1 es false"
  var elCasoTres = "caso 3 se ejecuta porque caso 1 y 2 son false"
```

```
  console.log(elcasoUno);
  console.log(elCasoDos);
  console.log(elCasoTres);
```

Condicionales Orden lógico, se llama a la function y se ejecuta según el caso

```
clasificarValor(2) //caso 1 se ejecuta por que es el primer true que encuentra
clasificarValor(15)//caso 2 se ejecuta por que caso 1 es false
clasificarValor(7) //caso 3 se ejecuta por que caso 1 y 2 son false
var elcasoUno = "caso 1 se ejecuta por que es el primer true que encuentra"
var elCasoDos = "caso 2 se ejecuta por que caso 1 es false"
var elCasoTres = "caso 3 se ejecuta por que caso 1 y 2 son false"
console.log(elcasoUno);
console.log(elCasoDos);
console.log(elCasoTres);
```

Encadenar sentencias "if else"

Vamos a ver un ejemplo con dos clausulas else if creando una funcion

```
function interpretarIMC(indiceDeMasaCorporal)
{
  if(indiceDeMasaCorporal < 18.5)
  {
    console.log("Bajo peso");
  }
  else if(indiceDeMasaCorporal <= 24.9 )
  {
    console.log("peso Normal");
  }
  else if(indiceDeMasaCorporal <= 29.9)
```

```
  {
    console.log("Sobrepeso");
  }
  else
  {
    console.log("Obeso");
  }
}
interpretarIMC(17.8);//caso 1 se ejecuta porque es el primer true que encuentra
interpretarIMC(22.2);//caso 2 se ejecuta porque caso 1 es false
interpretarIMC(28.5);//caso 3 se ejecuta porque  caso 1 caso 2 es false
interpretarIMC(32.2);//caso 4 se ejecuta porque no encuentra ningún true
var casoIF = "Caso 1 se ejecuta porque es el primer true que encuentra"
var casoPrimerElseIf = "Caso 2 se ejecuta porque  caso 1 es false"
var casoSegundoElseIf = "Caso 3 se ejecuta porque caso 1 caso 2 es false"
var casoElse = "Caso 4 se ejecuta porque no encuentra ningún true"
console.log(casiIF);
console.log(casoPrimerElseIf);
console.log(casoSegundoElseIf);
console.log(casoElse)
```

Código de golf

Definimos una función que va a llevar como parámetro→par que es el numero promedio de goles esperados y→golpes que es el número de golpes ejecutados.

para proseguir con este mini proyecto deberemos usar condicionales para implementar esta tabla.

Golpes Retornar

---

1 "Hole-in-one!"

<= par - 2 "Eagle"

par - 1 "Birdie"

par "Par"

par + 1 "Bogey"

par + 2 "Doble Bogey"

>= par + 3 "Go Home!"

Nota: par y golpes siempre serán numéricos y positivos, el parámetro par es→4

```
function PuntageDeGolf(par, golpes)
  {
    if(golpes == 1)
    {
      return "Hole-in-one!";
    }
    else if(golpes <= par - 2)
    {
      return "Eagle";
    }
    else if(golpes == par - 1)
    {
      return "Birdie";
    }
    else if(golpes == par)
```

```
 {
   return "Par";
 }
 else if(golpes == par + 1)
 {
   return "Bogey";
 }
 else if(golpes == par + 2)
 {
   return "Double Bogey";
 }
 else if(>= par + 3)
 {
   return "Go Home!";
 }
}
console.log(puntajeDeGolf(4, 1)); // par  4  retorna 1 se muestra  "Hole-in-one!"
console.log(puntajeDeGolf(4, 2)); //par  4 <= par -2 retorna 2  se muestra  "Eagle"
console.log(puntajeDeGolf(4, 3)); // par  4 par -1 retorna 3 se muestra  "Birdie"
console.log(puntajeDeGolf(4, 4)); // par  4 par  retorna  4 se muestra  "Par"
console.log(puntajeDeGolf(4, 5)); //par  4  par + 1  retorna  5 se muestra  "Bogey"
console.log(puntajeDeGolf(4, 6)); // par  4 par +2 retorna  6 se muestra  "Doble Bogey"
console.log(puntajeDeGolf(4, 8)); // par 4 >= par + 3  retorna 7 se muestra "Go Home!"
```

Sentencia Switch:

Ahora vamos a ver otra estructura que en cierta forma remplaza los condicionales en JS

Definimos una función con un parámetro →valor, este parámetro va a ser un numero entre

1 y 4 cada uno de esos valores va a estar asociados a una cadena específica. A una cadena de caracteres, la función va a retornar esa clasificación, esa cadena a partir del número en lugar de condicionales vamos a usar la sentencia switch que nos permite escribir código que es muy similar a una condicional. Pero swifch nos permite a partir de un valor decidir que va a pasar en base a ese valor.

```
function clasificarValor(valor)
{
  var respuesta;//variable local
  switch (valor)
 {
   case 1:
     respuesta = "alpha";
     break;
   case 2:
     respuesta = "beta";
     break;
   case 3:
     respuesta = "gamma";
     break;
   case 4:
     respuesta = "delta";
     break;
 }
    return respuesta;
```

```
  }
```

Y finalmente vamos a retornar el valor respuesta. return se coloca fuera de las llaves porque no es parte del switch . Finalmente llamamos a la función para pasarle los valore que podemos asignarle a una variable o para mostrarlo en pantalla

```
console.log(clasificarValor(1));//valor=1 se ejecuta caso1 respuesta vale→"alpha"

console.log(clasificarValor(2));//valor=2 se ejecuta caso2 respuesta vale→"beta"

console.log(clasificarValor(3));//valor=3 se ejecuta caso3 respuesta vale→"gamma"

console.log(clasificarValor(4));//valor=4 se ejecuta caso4 respuesta vale→"delta"
```

Ahora vamos a usar switch con una cadena de caracteres solo declarando una variable

```
var producto = "helado"
  switch (producto)
  {
    case  "pizza":
      console.log("La pizza basica cuesta $550");
      break;
    case "hambueguesa":
      console.log("Las hamburguesas cuestan $250");
      break;
    case "helado":
      console.log("El helado cuesta $95");
      break;
    }
    //Como la variable valor se asignó "helado" se muestra→ ("El helado cuesta $95")
  }
    console.log("Luego del switch continua ejecutando el código");
```

Sentencia Switch opción predeterminada

También podemos agregar una opción predeterminada a la sentencia switch que es el equivalente a else en una condicional if. Esa opción se va a ejecutar si ningunos de los valores es el valor de la variable o de la expresión. Esta función va a seleccionar el idioma que corresponda a un valor especifico.

```
function seleccionarIdioma(valor)
  {
    var idioma;
    switch(valor)//dentro de la sentencia agregamos el parametro
    {
      case 1:
        idioma = "Español";
      break;
      case 2:
        idioma = "Frances";
        break;
      case 3:
        idioma = "Italiano";
        break;
      default:
        idioma = "Ingles";
        break;// es opcional pero por norma lo usamos
    }
    return idioma;
  }
  console.log(seleccionarIdioma(1));//valor=1 se ejecuta caso1 var idioma es "Español"
  console.log(seleccionarIdioma(2));//valor=1 se ejecuta caso1 var idioma es "Frances"
  console.log(seleccionarIdioma(3));//valor=1 se ejecuta caso1 var idioma es "Italiano"
```

```
console.log(seleccionarIdioma(4));//se ejecuta por defecto var es "Ingles"
```

Sentencia switch múltiples casos:

En las sentencias switch también podemos optar ejecutar un código específico para varios valores vamos a tener números entre 1 y 6 que van a representar los valores en el dispositivo. El parámetro se va a llamar valor, el parámetro que va a representar el volumen que va ser una cadena de caracteres que va a definir la intensidad del volumen. Para eso usaremos la sentencia switch , esa sentencia va a depender del valor del parámetro. Pero si queremos que el volumen intermedio sea el caso 2 o el 3 lo podemos hacer dentro de un solo break dos valores pueden realizar la misma acción. También podemos tener más de dos casos manejado de la misma forma

caso 4, 5 y 6 y si es valor esta fuera de rango, digamos 7 se ejecuta por defecto

```
function clacificarVolumen(valor)
 {
   var volumen;
   switch(valor)
    {
      case 1:
        volumen = "Bajo";
      break;
      case 2:
      case 3:
        volumen = "Intermedio";
      break;
      case 4:
      case 5:
      case 6:
        volumen = "Alto";
      break;
      default:
```

```
      volumen = "Sin volumen";
     break;
   }
   return volumen;
}
console.log(clacificarVolumen(1));//valor=1 se ejecuta caso1 var volumen "Bajo"
console.log(clacificarVolumen(2));//valor=2 se ejecuta caso2,3 var volumen "Intermedio"
console.log(clacificarVolumen(4));//valor=4 se ejecuta caso4,5,6 var volumen "Alto"
console.log(clacificarVolumen(7));//valor=7 se ejecuta default var volumen "Sin volumen"
```

Reemplazar "if… else" por "switch"

En ciertos casos podemos remplazar condicionales con la sentencia switch en el ejemplo vemos que el nombre de la función y el parámetro al igual que la variable no cambia al igual que en el final el retorno el mismo

```
function seleccionarIdioma(valor)
 {
   var idioma;
   switch(valor)
   {
     case 1:
       idioma = "Español";
     break;
     case 2:
       idioma = "Frances";
       break;
     case 3:
       idioma = "Italiano";
```

```
if(valor == 1
{
   idioma = "Español";
 } else if (valor == 2) {
   idioma = "Frances";
} else if(valor == 3) {
   idioma = "Italiano";
 } else {
   idioma  = "ingles";
 }
```

```
        break;
       default:
        idioma = "Ingles";
        break;// es opcional pero por norma lo usamos
     }
     return idioma;
  }
```

Retornar valores Vooleanos:

Como retornar valores booleano de una forma muy concisa desde una función, inicialmente lo podíamos iniciar con un condicional que funciona y va a cumplir su propósito pero lo mismo lo podemos hacer en una sola línea simplemente retornar el valor de una comparación.

→return a < b; ←nota la diferencia de lo conciso que es en comparación de usar condicional

```
function esMenorQue(a, b)
{
   If(a < b)
  {
    return  true;
  }
  Else
  {
   return false;
  }
}
```

Patrón de retorno anticipado

Cuando retornamos el valor de una función, en ese momento que retornamos el valor la función se detiene completamente. Cualquier línea que esta luego del return no se va a ejecutar. En el ejemplo vamos a intentar mostrar otro mensaje después de return→Esa línea nunca se va a ejecutar y la función se detiene en el return. Eso es algo importante porque ese patrón anticipado es muy útil, específicamente cuando necesitamos detener la función.

```
function miFuncion()
 {
  console.log("Hola")
  return "Mundo";
  console.log("Adios");//unreachable code detecte (7027) ←codigo inalcansable detectado
 }
 console.log(miFuncion());// aquí nos muestra "hola" "Mundo"
```

Patron de valor anticipado

A continuación en este ejemplo definimos una función y le damos un parámetro→ num, La raíz cuadrada de un número negativo no existe, así que si esa condición es verdadera num<0 no podemos calcular la raíz cuadrada. En ese caso lo que hacemos es detener la función y retornamos un valor que sería→ udefefined En el caso de que el valor sea valido→ num>0. Digamos que pasamos 25, la condición del if no se cumple la raíz cuadrada seria 5 pero si pasamos un numero negativo la condición if es true y se ejecuta retrun undefined

```
 function raizCuadrada(num)
 {
  if(num < 0)//num no es menor que 0 y el resultado es 5 pero si pasamos -5 ahi entra al if
  // y se ejecuta return undefined
  {
   return undefined//aqui se detiene la funcion si la condicion es true
  }
  return Math.sqrt(num);//el parametro num calculamos la raiz cuadrada
 }
 console.log(raizCuadrada(25));//pasamos a num el valor 25 que nos devueve su raiz 5
 console.log(raizCuadrada(-5))//pasamos a num el valor -5 que nos devueve undefined
```

```
*/
```