Curso Soy Henry

Comandos básicos de Git
pwd imprime el directorio o rutas o carpetas sobre el cual trabajamos actualmente

ls lista los elementos de la carpeta en la que nos encontramos actualmente

cd ayuda a cambiar de directorio o carpeta
cd nombreCarpeta para avanzar o ingresar en una carpeta interna desde donde nos encontramos
cd .. para volver un directorio atrás

clear limpia el terminal es decir todo lo que hicimos y se ve en la pantalla

mkdir permite crear una carpeta vacía
mkdir carpeta1

touch crea un nuevo archivo vacío, es importante
touch archivo4.js

rm elimina el archivo seleccionado
rm archivo4.js

rm -r elimina una carpeta o directorio indicado
rm -r carpeta1

help nos da lista de comandos básicos y podemos leer para que sirve cada uno.

La terminal es la interfaz de linea de comandos *CLI* que nos permite comunicarnos con la computadora para darle ordenes y conseguir un resultado especifico

Un repositorio es un almacén donde se guardan cosas

Git sistema de control de versiones

Configurar credenciales de git para que nos reconozca como propietarios o colaboradores, hacerlo desde git bash

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~ (master)
$ git --version
git version 2.37.0.windows.1
```

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~ (master)
$ git config --global user.email "franco.ferro.it@hotmail.com"

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~ (master)
$ git config --global user.name "Franco Ferro"
```

y ahora ya estamos listos para trabajar con la tecnología git

Ahora vamos a crear un repositorio y hacer nuestro primer commit

Commit es una captura instantánea de los cambios preparados en ese momento del proyecto, es decir es guardar una foto del estado de los archivos y carpetas del repositorio local, al momento de ser tomada

git init en visual studio cuando abrimos la carpeta con el archivo, para poder iniciar el repositorio, poniendo ***new terminal***
Nunca debemos hacer **Git Init** dentro de la carpeta escritorio o de archivos grandes o muchos archivos como un disco duro, ya que iniciaríamos un seguimiento de todo eso.

Ejemplo del git init en el visual studio

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry
(master)
$ git init
Initialized empty Git repository in C:/Users/Franco Ferro/Desktop/PrepCourse
Henry/.git/

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry
(master)
$ ls -a
 ./   ../   .git/  'Mi primer repositorio.txt'

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry
(master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Mi primer repositorio.txt
```

nothing added to commit but untracked files present (use "git add" to track)

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry (master)
$ git add "Mi primer repositorio.txt"

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry (master)
$ git commit -m "Algun mensaje de commit"
[master (root-commit) 6c52773] Algun mensaje de commit
 1 file changed, 3 insertions(+)
 create mode 100644 Mi primer repositorio.txt

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse Henry (master)
$ git log
commit 6c52773f1152cbec24f377d2d672a82754e81804 (HEAD -> master)
Author: Franco Ferro <franco.ferro.it@hotmail.com>
Date: Tue Mar 21 01:16:05 2023 -0300

    Algún mensaje de commit

FIN

ls -a nos permite ver archivos ocultos

Git Status para confirmar nuestro estado actual y que estamos dentro del repositorio

Git Add para agregar el archivo que queramos en el repositorio, como en este caso por ejemplo mi primer repositorio.txt
Git Add . Es un comando que podemos usar para agregar todos los archivos

Git Commit -m "Texto del commit" usaremos para finalmente sacar una foto de lo que agregamos y guardarlo
el mensaje reemplaza al texto que describa las cosas que contiene el commit

Git log es para listar los commits existentes y verlos

Abrimos Git Bash, nos posicionamos en el escritorio y creamos una carpeta1

Creando Repositorio con Github

Los archivos **Readme** son opcionales y permiten darle mas contexto a nuestro trabajo

git clone mas el link del repositorio lo que hacemos es crear una carpeta en el escritorio, carpeta sobre la que vamos a trabajar.

cd PrepCourse-Henry/ usamos para cambiarnos de directorio

code . Para que nuestro vs code se abra y trabaje sobre el repositorio, una vez que se abre podemos ver que contiene el archivo README.md, tenemos que acceder a ese archivo y borrar lo que hay en el para escribir nuestro nombre y un dato que describa nuestro proyecto

Ejemplo de lo que hicimos para crear el repositorio y la carpeta en el escritorio y abrirlo en vs code para poder editar el código

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop (master)
$ git clone https://github.com/FrancoFerro/PrepCourse-Henry.git
Cloning into 'PrepCourse-Henry'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop (master)
$ cd PrepCourse-Henry/

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse-Henry (main)
$ code .
```

FIN *

Seguido de esto y de haber hecho las modificaciones correspondientes desde vs code, vamos a realizar los comandos para crear un nuevo commit desde **GitBash**

git pusch origin main buscar en google, es importante que en este caso aparezca la palabra main ya que es la que figura en el directorio como podemos ver, básicamente esto se hace para poder ver los cambios reflejados en Github pero de todas formas buscar en internet.
Una vez hecho eso ya podemos ir a nuestro repositorio de Github y ver los cambios reflejados

*continuación del Git Bash

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse-Henry (main)
$ git add README.md

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse-Henry (main)
$ git commit -m "Commit de prueba"
[main 6054cce] Commit de prueba
 1 file changed, 1 insertion(+), 2 deletions(-)
```

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/PrepCourse-Henry (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/FrancoFerro/PrepCourse-Henry.git
   a98d320..6054cce  main -> main
```

Ahora si finaliza

Ahora vamos a crear otro repositorio en Github pero sin el README.md, se llamara PrepCourse NoReadme

Una vez creado se puede observar que el repositorio muestra una serie de mensajes debido a que no hay ningún archivo creado, ni siquiera el README.md, entonces debemos agregarle archivos al repositorio.

Para eso tenemos dos opciones presentadas por Github:

1. Dice que debemos hacer un **Git init,** crear un archivo con **Git add,** agregarlo a un commit con **Git commit -m "Nombre de commit"**, luego aparece **Git branch -M main** (modifica el nombre de la rama de trabajo actual) pero de esto hablamos mas adelante, por ahora solo copiamos como lo muestra en Github, seguido de esto debemos usar **Git remote add origin +link del repositorio** (para hacer una conexión remota desde nuestro repositorio local), y finalmente **Git push -u origin main** para realizar la sincronización de este repositorio local con el espacio de trabajo de Github.

   Pasos:

   ```
   git init
   git add NombreArchivo (README.md sugerido por Github)
   git commit -m "Nombre del Commit"
   git branch -M main
   git remote add origin linkRepositorio
   git push -u origin main
   ```

2. Esta segunda opción nos indica que si ya tenemos un repositorio de forma local que ya se encuentra en nuestra maquina podemos realizar la conexión con el comando **Git remote add origin** y demás pasos, ahora vamos a hacer un ejemplo

   Pasos:

   ```
   git remote add origin https://github.com/FrancoFerro/PrepCourse-NoReadme.git
   git branch -M main
   git push -u origin main
   ```

Ejemplo usando esta segunda opción:

no entendí

Usando el repositorio del PrepCourse
Nuevamente ingresamos en git bash y con el comando cd mas el nombre de la carpeta o cd .. nos movemos a la carpeta o lugar deseado para finalmente usar git clone mas link del repositorio y crear una carpeta del repositorio con todos sus archivos listos para poder trabajar.
Una vez hecho eso ingresamos en visual studio code y abrimos esa carpeta con open folder, y ya podemos visualizar todo su contenido en vs code.

Conexión entre repositorio local y github

```
Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop (master)
$ cd RepositorioHenry

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry (main)
$ git add README.md
fatal: pathspec 'README.md' did not match any files

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry (main)
$ cd CarpetaHenry

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry/CarpetaHenry (main)
$ git add primerArchivo.txt

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry/CarpetaHenry (main)
$ git commit -m "primer archivo creado"
[main 58d59b7] primer archivo creado
 2 files changed, 1 deletion(-)
 delete mode 100644 CarpetaHenry
 create mode 100644 CarpetaHenry/primerArchivo.txt

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry/CarpetaHenry (main)
$ git branch -M main

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry/CarpetaHenry (main)
$ git remote add origin
usage: git remote add [<options>] <name> <url>

    -f, --fetch           fetch the remote branches
    --tags                import all tags and associated objects when fetching
                          or do not fetch any tag at all (--no-tags)
    -t, --track <branch>  branch(es) to track
    -m, --master <branch>
                          master branch
    --mirror[=(push|fetch)]
                          set up remote as a mirror to push to or fetch from


Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/RepositorioHenry/CarpetaHenry (main)
```

```
$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Writing objects: 100% (4/4), 308 bytes | 154.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/FrancoFerro/RepositorioHenry.git
   c52a1fb..58d59b7  main -> main
branch 'main' set up to track 'origin/main'.
```

Y estos son los pasos que debemos hacer cada vez que realizamos una modificacion en el archivo, seleccionamos vamos a la carpeta donde se encuentra, volvemos a agregarlo y lo guardamos y hacemos esos pasos anteriores para guardar y mostrar esos cambios en el repositorio de github

MODULO 3

COMENZAMOS CON JAVASCRIPT

Creamos una carpeta en el escritorio donde estara todo lo que practiquemos de javascript
Y desde vs code vamos en la hoja+ y creamos Variables.js o el archivo js que queramos

Con var podemos crear una variable y luego asignarle un valor =

Tipos de datos

Los tipos de datos seran numeros (todos), strings o cadenas de texto (palabras dentro de comillas dobles o simples), booleanos (true or false), undefined (aparece cuando todavia no le asignamos un valor a la variable)

Null, parecido a undefined pero en este caso el programador le da ese valor a tal variable

Hay mas tipos de datos que vamos a ir viendo mas adelante

```
//String

var bootcamp = "Henry";

//Numeros

var numeroEntero = 17;

//Booleanos

var estoyEntendiendo = true;
var estoyAburrido = false;
```

```
//Undefined

var cajon1;
console.log(cajon1);  //Y esto sera Undefined ya que no le asignamos valor a
cajon1

//Null

var sinNada = null;
```

Método Length

Este método solo se puede aplicar a String, si lo intentamos aplicar en otro tipo de dato dara error

```
//Ejemplo Length

"Hola".length   //4
```

Mas tipos de datos y estructuras js

https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures#estructuras_y_tipos_de_datos

Operadores y precedencia

`+ - * / ...`

NOTA: En vs code primero debemos instalar CODE RUNNER para poder trabajar con estos operadores

Y ahora si creamos el archivo operadores.js donde vamos a escribir nuestro código de esta practica

console.log(operación) nos va a permitir imprimir y ver en la consola el resultado de nuestro código

% Sirve para ver si un numero es divisible por otro o no, o simplemente para saber el resto

Precedencia trata por ejemplo sobre que operador se divide por tal operador, o que se hace primero si la multiplicación o la suma como en el ejemplo de acá abajo. Js respeta las reglas de precedencia. En caso de que queramos que por ejemplo primero se haga la suma y luego la multiplicación, tendremos que separar con paréntesis para que lo del paréntesis.

Adición y strings, concatenación

Ejemplo

```
console.log(3 + 5);
```

```
console.log(14 - 7);
console.log(4 * 6);
console.log(30 / 9);

// Modulo o resto de una division

console.log(95 % 4);

// Precedencia de operadores

console.log(3 + 5 * 2 - 8);
console.log((3 + 5) * 2 - 8);

// Adicion y strings

console.log("Hola " + "Franco");

// O lo podemos hacer creando variables
var nombre = "Franco";
var saludo = "Hola " + nombre;
console.log(saludo);
```

Operadores de comparación

== Solo verifica si el valor es el mismo
≡ Verifica el tipo de dato y si el valor es el mismo, es estricta

```
//Operaciones de comparación

console.log (4 < 7);
console.log (4 < 1 );
console.log (4 > 4);
console.log (4 == 7);

//Igualdad vs igualdad estricta

console.log (3 == 3);
console.log (3 === 3);
console.log (3 == "3");
console.log (3 === "3");
console.log (7 == "7");
console.log (7 === "7");


//Asignación y asociatividad

var a = 1;
var b = 2;
var c = (a = b);
```

```
console.log(a);
console.log(b);
console.log(c);
```

Funciones

```
//function sumaTres(x) {
//    console.log(x + 3);
//}

//sumaTres(5);

// Primer Ejemplo para declarar una funcion

function sumaTres(x) {
  return x + 3;
}

// Segunda, guardando la funcion en una variable

var sumaTres = function (x) {
  return x + 3;
};

// Tercera, usando la funcion de flecha

var sumaTres = (x) => {
  return x + 3;
};

-----

// OJO CON CONFUNDIR RETURN CON CONSOLE.LOG

//EJEMPLO ERROR

var animal = "caballo";
console.log(animal) // caballo

// Lo que hace console.log es ayudar al programador a ver el resultado
// Ninguna aplicacion o programa real deberia tener un console.log dentro de su codigo

// Por otro lado tenemos el RETURN, fundamental para toda funcion ya que es la instruccion que va a
// indicar que valor nos debe devolver
```

Ahora hacemos un ejemplo con código y abriendo la terminal gitbash desde vs estudios

```
function cuidadoConElConsoleLog(nombre) {
   console.log(nombre);
   return nombre;
}
```

resultado usando gitbash

Franco Ferro@DESKTOP-DT72A4O MINGW64 ~/Desktop/CursoPreparatorioHenry (master)
$ node
Welcome to Node.js v16.15.1.
Type ".help" for more information.
> function cuidadoConElConsoleLog(nombre) {
...   console.log(nombre);
...   return nombre;
... }
undefined nos da undefined lo cual esta bien
> cuidadoConElConsoleLog("Franco") ahora ejecutamos la funcion con el nombre
Franco nos muestra lo que se imprime
'Franco' El return nos muestra el valor verdadero, por eso es importante
>

Ahora hacemos lo mismo pero le sacamos el return a la funcion

```
function cuidadoConElConsoleLog(nombre) {
   console.log(nombre);
}
```

Usando gitbash

> function cuidadoConElConsoleLog(nombre) {
...   console.log(nombre);
... }
undefined
> cuidadoConElConsoleLog("Franco")
Franco
undefined

Y como podemos ver nos dice que el valor es indefinido, por eso es muy importante incluir el return en la funcion, ya que es la que va a dar el paso al valor real para poder luego usarlo y demas

Siguiendo con el ejemplo anterior para tener en cuenta la importancia del return, a continuacion hacemos otro ejemplo pero creando otra funcion mas que va a usar el valor de la primera funcion

```
function cuidadoConElConsoleLog(nombre) {
  console.log(nombre);
}

function otraFuncion() {
  return (
    "El nombre retornado por la funcion cuidadoConElConsoleLog es = " +
    cuidadoConElConsoleLog("Franco")
  );
}
```

Usando gitbash para comprobar

```
> function otraFuncion() {
...     return (
.....         "El nombre retornado por la funcion cuidadoConElConsoleLog es = " +
.....         cuidadoConElConsoleLog("Franco")  // No va ;
.....     )
... }
undefined
> otraFuncion()
Franco
'El nombre retornado por la funcion cuidadoConElConsoleLog es = undefined'
>
```

Luego si usamos el return el la primera funcion si ya se mostrara el nombre

NOTA, al ejecutar la terminal gitbash debemos si o si pasar por todas las funciones anteriores a las que se incluye en la ultima funcion, en este caso si o si debimos o debemos ejecutar primero la funcion primera, para poder ejecutar la otraFuncion, si no no la tomara, ya que esta ultima incluye a la anterior

Ahora creamos nueva funcion pero con el return primero y el console.log despues
Y lo primero que muestra sera el return pero el console.log no lo muestra, PORQUE?
Porque cuando llega a la linea de un return, la funcion ya termina de ejecutarse, todo lo que se encuentre por debajo del return no se va a ejecutar.

```
function cuidadoConElReturn(nombre) {
   return nombre;
   console.log(nombre);
}
```

Usando gitbash

```
> function cuidadoConElReturn(nombre) {
...     return nombre;
...     console.log(nombre);
... }
undefined
> cuidadoConElReturn("Franco")
'Franco'
>
```

Nomenclatura

3 tipos principales

camelCase, PascalCase, snake_case

Control de flujo (if/else)

```
function viajar (destino) {
   if (destino === "Brasil") {
      console.log("Gire a la IZQUIERDA");
   } else if (destino === "Argentina") {
      console.log("Gire a la DERECHA");
   } else {
      console.log("Nos PERDIMOS");
   }
}

viajar("Brasil");
viajar("Argentina");
```

```
function puedeManejar (edad) {
   if (edad >= 18) {
      console.log(true);
   } else {
      console.log(false);
   }
}

puedeManejar(18);
puedeManejar(17);
```

Objeto Math

- **Math.pow sirve para potenciar un numero, el primer numero es base y el segundo sera la potencia**

Ejemplo

Math.pow(2, 3); //8

Math.round & Math.floor & Math.ceil

- **Math.round redondea el numero decimal al entero mas proximo**
  0,49 → 0
  0,50 → 1

Math.round(0,49)
// 0
Math.round(0,50)
// 1

- **Math.floor redondea el numero decimal al entero de menor valor**
  5,93 → 5

Math.floor(5,93)
// 5

- **Math.ceil redondea el numero al siguiente entero**
  3,27 → 4

Math.ceil(3,27)
// 4

Math.max & Math.min

Nos permiten conocer el valor maximo o minimo de un conjunto de numeros

```
Math.max(1, 2, 3, 4, 5);
// 5

Math.min(1, 2, 3, 4, 5);
// 1
```

Math.random

Nos permite generar un numero aleatorio, siempre sera un numero decimal entre el 0 y el 1

```
Math.random();
// 0,8051654134468465
```

Para finalizar debemos hacer la homework correspondiente al modulo

Para esto vamos a tener que instalar algo en vs code desde la terminal
Abrimos la terminal y escribimos npm install, luego con npm test comprobamos que este todo bien

Si ponemos npm test 04 vamos a estar verificando los ejercicios del modulo 4, y si no hicimos nada o estan mal dira por ejemplo que hay 38 ejercicios mal de 38 totales. Y asi podemos hacer con cada modulo que tenga el test. Con esto verificamos si tenemos todos los ejercicios bien hechos.

NOTA DE JAVA SCRIPT: Las variables var pueden ser modificadas y re-declaradas dentro de su ámbito; las variables let pueden ser modificadas, pero no re-declaradas; las variables const no pueden ser modificadas ni re-declaradas. Todas ellas se elevan a la parte superior de su ámbito

MODULO 3, EJERCICIO 05
NOTA DEL ULTIMO EJERCICIO

toLowerCase() El método toLowerCase() devuelve el valor en minúsculas de la cadena que realiza la llamada

.includes El método includes() determina si una matriz o lista de arreglos incluye un determinado elemento, devuelve true o false según corresponda **y prosigue con el programa según corresponda**

Ejemplo de como queda el codigo usando lo anterior

```
function esVocal(letra) {
  // Escribe una función que reciba una letra y, si es una vocal, muestre el mensaje “Es vocal”.
  // Si el usuario ingresa un string de más de un caracter debes retornar el mensaje: "Dato incorrecto".
  // Si no es vocal, tambien debe retornar "Dato incorrecto".
  // Tu código:
  if(["a", "e", "i", "o", "u"].includes(letra.toLowerCase())){
    return "Es vocal";
   }else {
    return "Dato incorrecto";
   }
}
```

MODULO 4

Operadores Lógicos

Datos de precedencia que se suma a lo visto antes
El operador || (or) precede al operador && (and)

```
function mayorYMenor(num) {
   if (num > 5 && num < 10) console.log(true);
   else console.log(false);
}

mayorYMenor(2);
mayorYMenor(6);
mayorYMenor(11);


function mayorYMenorYPar(num) {
   if (num > 5 && num < 10 && num % 2 === 0) console.log(true);
   else console.log(false);
}

mayorYMenorYPar(7);
mayorYMenorYPar(6);


function operadorOr(str) {
   if (str === "Henry" || str.length < 2) console.log(true);
   else console.log(false);
```

```
}

operadorOr("Henry");
operadorOr("A");
operadorOr("Hola");


function negacion(permiso) {
  if (permiso) console.log("Tiene permiso");  // Es como decir (permiso ===
true)
}

negacion(true);
negacion(false);


function negacion(permiso) {
  if (!permiso) console.log("Tiene permiso");  // Es como decir (permiso !==
true)
}

negacion(true);
negacion(false); // Se mostrara el mensaje


// Funciones mas complejas usando los operadores

function condicionCompleja(num) {
  if (num > 9 && num % 2 === 0 || num === 3) console.log(true);
  else console.log(false);
}

condicionCompleja(12);
condicionCompleja(3);
condicionCompleja(11);
```

VERACIDAD

La forma de procesar la lógica en cualquier lenguaje se basa en un sistema binario. Unos y ceros. Verdaderos y falsos.

Ingresando en cmd y posteriormente ingresando en node.js
C:\Users\Franco Ferro> node ← aca simplemente escribimos node y ya ingresa
Welcome to Node.js v16.15.1.
Type ".help" for more information.

Y posteriormente usamos la funcion Boolean() a la que podemos pasarle ciertos argumentos y nos dara true o false según corresponda

```
1                   // true
0                   // false
-1                  // true
true                // true
false               // false
'string'            // true
null                // false
undefined           // false
[ ]                 // true
```

BUCLES FOR Y WHILE

Diferencias entre For y While

- **For** se usa cuando sabemos la cantidad máxima exacta de pasos que queremos ejecutar
- **While** se usa cuando no sabemos con certeza cuantos pasos necesitaremos hasta finalizar la ejecución

RECURSOS ADICIONALES

Expresion Switch
La declaración switch evalúa una expresión, comparando el valor de esa expresión con una instancia case, y ejecuta declaraciones asociadas a ese case, así como las declaraciones en los case que siguen
Link para saber mas:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/switch

Syntaxis

```
switch (expresión) {
  case valor1:
    //Declaraciones ejecutadas cuando el resultado de expresión coincide con el valor1
    [break;]
  case valor2:
    //Declaraciones ejecutadas cuando el resultado de expresión coincide con el valor2
    [break;]
  ...
  case valorN:
    //Declaraciones ejecutadas cuando el resultado de expresión coincide con valorN
    [break;]
  default:
```

```
    //Declaraciones ejecutadas cuando ninguno de los valores coincide con el valor de la expresión
    [break;]
}
```

Bucle Do-While

El bucle **Do-While** (hacer mientras) ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez.

Link para saber mas:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/do...while

Declaracion Continue

La declaración **continue** se utiliza dentro de los Bucles For o While. Nos permite omitir alguna de las iteraciones si se cumple una condición específica. Y lo que hace es volver al principio de la estructura saltándose todo lo que este por debajo del *continue*

Link para saber mas:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/continue

Syntaxis

```
// Ejemplo usando continue en while

i = 0;
n = 0;
while (i < 5) {
  i++;
  if (i == 3)
  continue;
  n += i;
}

console.log(n); // 12 ya que se saltea todo y vuelve a la estructura desde el principio
```

Break

La declaración **break** se utiliza dentro de los Bucles For. Nos permite "romper" o finalizar el bucle con antelación si se cumple una condición específica

Link para saber mas:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/break