CHECKPOINT 7 – EJERCICIO TEÓRICO

01 - ¿Qué hace que Javascript sea diferente de cualquier otro lenguaje de programación?

JavaScript es el único lenguaje que un navegador puede entender sin necesidad de ejecutarlo desde un servidor o tener que instalar ningún software extra en el equipo. Esto significa que se puede escribir código JavaScript, simplemente abrirlo en el navegador y este puede analizar el código, interpretarlo y luego ejecutar el programa.

Algunas de las aplicaciones más potentes del mundo incorporan JavaScript, como, por ejemplo; Gmail, Twitter y Facebook, los cuales utilizan JavaScript de forma extensiva. Google y Facebook incluso han cogido el lenguaje JavaScript y han construido su propio marco de trabajo, sus propias capas sobre él.

JavaScript tiene una amplia variedad de bibliotecas (jQuery, AngularJS, React, Node.js, etc) que ayudan con el desarrollo para ahorrar tiempo y líneas de código.

JavaScript se integra a la perfección con HTML y CSS, lo que permite añadir sin esfuerzo comportamiento dinámico e interactividad a las páginas web.

Es posible usar Javascript para crear aplicaciones para dispositivos móviles, tanto para iOS como Android, sin necesidad de usar los lenguajes específicos para cada sistema. Asimismo, puedes acceder a la cámara, a la localización y otros servicios del dispositivo para la creación de la aplicación.

JavaScript es una de las herramientas más potentes que puedes tener en tu arsenal para automatizar tu flujo de trabajo diario. Como el código JavaScript se puede ejecutar directamente en el navegador, puedes realizar tareas como: si imaginas que tienes una página web grande a la que accedes y necesitas hacer clic en un botón cientos de veces. Digamos que quieres actualizar y seguir a un grupo de personas en Instagram o en Linkedin.

De hecho, puedes crear un script de JavaScript muy simple que se ejecutará y lo automatizará. De esta forma, en lugar de tener que hacer clic manualmente en "Seguir" y continuar con cada botón, puedes escribir un script y, en una fracción de segundo, este puede hacer todo el trabajo por ti. Es ideal para trabajar con la automatización, por lo cual podemos decir que Javascript permite la automatización de flujos de trabajo para que de esta forma pueda hacer tareas repetitivas más rápido. Además, Javascript es un lenguaje asíncrono, lo que significa que se pueden ejecutar múltiples tareas simultáneamente sin afectar al rendimiento de la página. Esta característica abre la puerta a muchas posibilidades a la hora de crear

aplicaciones web, ya que permite ejecutar procesos en segundo plano mientras el usuario interactúa con otros elementos de la página.

02 - ¿Cuáles son algunos tipos de datos JS?

Existen 8 tipos de datos:

Datos primitivos:

- Undefined: consiste exactamente en un valor: undefined. Es lo que se obtiene cuando algo se declara, pero no se asigna.
  Es una variable vacía y que no tiene ningún valor, pero a diferencia del tipo “null”, las variables “undefined" son creadas sin valor y quedan a la espera de asignárselo.

  Ejemplo 1: Obtendremos undefined si intentamos llamar una variable que hemos declarado pero no hemos asignado.

  ```
  let myUndefined;
  ```

  Ejemplo 2: Tambien se puede asignar especificando undefined.

  ```
  let myUndefined = undefined;
  ```

- Boolean: Solo puede contener dos valores, true o false. Los valores booleanos suelen utilizarse para operaciones condicionales.
  Es importante escribir las palabras clave true y false en minúscula, ya que de lo contrario JavaScript no las reconocerá:

  ```
  let myBoolean = true
  ```

- Number: Contiene números positivos, negativos, enteros y decimales con una limitación de 64 bits, esto quiere decir que tiene una precisión limitada en cuanto a cálculos de coma flotante y números muy largos.

  ```
  let myNumber = 80;
  ```

- String: Contiene una cadena de caracteres que pueden ser letras, números y símbolos. Definimos el tipo String delimitándolo con comillas dobles " " o comillas simples ' '.

Una variable string puede consistir en cualquier conjunto de caracteres, desde un nombre hasta todo el código Html de un sitio web.

```
let myString = ‘Esto es una cadena de texto, con números (123) y
simbolos (#€=)’;
```

- Array: Contiene una lista ordenada de elementos que pueden ser de diferentes tipos de datos. Su longitud es variable y se pueden eliminar y añadir elementos en cualquier momento.

```
let myArray = [‘Dato1’, ‘Dato2’];
```

- BigInt: es una primitiva numérica en JavaScript que puede representar enteros de magnitud arbitraria. Contiene números sin limitación en cuanto a su tamaño, pero a diferencia del tipo “Number” solo admite números enteros, no decimales o de coma flotante.
  Es importante remarcar que no se pueden hacer operaciones entre variables tipo “bigint” y “number”, las variables “bigint” solo se pueden operar con otras variables “bigint”. Un BigInt se crea añadiendo n al final de un entero o llamando a la función BigInt().

```
let myBigint = 12345678901234567890123456789n;
```

- Symbol: Es una variable utilizada para datos únicos e inmutables, no hay dos variables de este tipo iguales, ni siquiera dando el mismo valor, esto hace que puedan ser usadas como identificadores o claves de objetos. La inclusión de un identificador es opcional y no afecta a su uso. En algunos lenguajes de programación, los Symbols se denominan "atoms". Su propósito es crear claves de propiedad únicas que garanticen no chocar con claves de otro código.
  Ejemplo 1:

```
const mySymbol = symbol(‘dato’);
```

  Ejemplo 2:

```
Const mySymbol = symbol();
```

- Function: Este tipo de variable contiene una función que puede ser ejecutada en cualquier momento.

```
const sumar = function() {}
```

Otros tipos de datos:

- Null: Este tipo de variable representa la ausencia intencional de cualquier valor, un valor nulo o «vacío.

```
Ejemplo let myNull = null;
```

- Object: Es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor.
  Contiene una lista con índices para identificar a cada uno de los elementos. Al igual que los “Arrays” su longitud es variable y se pueden eliminar y añadir elementos en cualquier momento, además también pueden ser elementos de diferentes tipos.
  En Javascript los objetos son mutables, lo que significa que pueden ser cambiados sin crear un valor completamente nuevo

```
let myObject = {Nombre1: “Valor de Nombre1”,Nombre2 “Valor de Nombre 2”};
```

03 - ¿Qué son las tres funciones de cadena JS?

Hay muchas operaciones útiles que podemos realizar en strings mediante métodos integrados:

- charAt(): Sirve para devolver el carácter de la posición especificada dentro del paréntesis, es decir, entre paréntesis debemos informar de la posición en la que estemos interesados y nos devolverá el carácter que este en dicha posición.

```
let miTexto = ‘Texto prueba’;
console.log(miTexto.charAt(7));
```

- length: da la longitud de un string en caracteres. length no es una función (no se llama con paréntesis al final), sino un atributo.

```
const refran = 'A quien madruga Dios le ayuda';
console.log(refran.length);
```

- indexOf(): Sirve para devolver la posición de un elemento en una cadena, es decir, hará una búsqueda en una cadena de lo que se le informe entre los paréntesis y te dirá la posición en la que se encuentra. Si hay más de una coincidencia, solo nos devuelve la primera.

```
let miTexto = ‘El equipo de Futbol es de Londres;
```

```
console.log(miTexto.indexOf(‘de’));
```

- lastIndexOf(): Sirve para devolver la posición de un elemento en una cadena, es decir, hará una búsqueda en una cadena de lo que se le informe entre los paréntesis y te dirá la posición en la que se encuentra mirando de atrás adelante el String facilitado. Si hay más de una coincidencia, solo nos devuelve la última, la primera que encuentra mirando de atrás adelante.

```
let miTexto = ‘El equipo de Futbol es de Londres;
console.log(miTexto.lastIndexOf(‘de’));
```

- concat(): La función concatena el valor que pasemos como argumento al string. Esto no cambia permanentemente el valor de la variable original, sólo cambia el valor que se devuelve al llamar a esa función. Para almacenar este nuevo valor, podemos reasignar la variable original o guardarlo en una nueva variable.

```
let miTexto = ‘Texto original’;
miTexto = miTexto.concat(‘ y texto añadido);
console.log(miTexto);
```

- repeat(): toma como argumento cuántas veces queremos repetir el string. Esto no altera el string original, solo retorna el valor repetido.

```
const animo = '¡Bravo!';
console.log(animo.repeat(4));
```

- includes(): Sirve para comprobar si una cadena se encuentra dentro de otra, devuelve un valor booleano por lo que obtendremos “true” si encuentra la cadena y “false” si no la encuentra.

```
let miTexto = 'A quien madruga Dios le ayuda';
console.log(miTexto.includes(‘gracias’));
```

- replace(): sirve para reemplazar valores en un string. Toma dos argumentos, primero el elemento que estás buscando, y segundo con qué quieres reemplazarlo. En este caso, la función tampoco cambia la variable original.

```
const cuestion = '¿Quieres queso?';
console.log(cuestion.replace('queso', 'jamon'));
```

04 - ¿Qué es un condicional?

Las condicionales son estructuras de comprobación que nos permiten realizar diferentes acciones dependiendo de si una condición se cumple o no. Las condicionales nos permiten tomar decisiones y ejecutar bloques de código diferentes en base a esas decisiones.

La sintaxis básica consiste en la palabra clave if, la condición a cumplir entre paréntesis y llaves {} que recogerán el output del condicional.

Existen tres niveles a la hora de crear condicionales, if, else y else if.

- if: Es la condicional básica que nos permite ejecutar un bloque de código si una condición es verdadera.
  Ejemplo:

```
let edad = 18;
if (edad >= 18) {
console.log("Tienes suficiente edad para alquilar un vehículo");
}
```

  En este ejemplo, si la variable edad es mayor o igual a 18, nos devolverá el mensaje "Tienes suficiente edad para alquilar un vehículo".

- else: Se puede combinar con la condicional if para ejecutar un bloque de código para que además de querer ejecutar un código si nuestra condición se cumple, también queramos ejecutar otro código cuando no se cumple. Para esto utilizamos la palabra clave else, y la añadimos justo después de cerrar la llave de nuestra condición if.
  Siguiendo el ejemplo anterior, vamos a establecer una condición else. El código comprobará si la primera condición se cumple, y si no es así, pasará a ejecutar el código establecido en else.
  Ejemplo:

```
let edad = 15;
if (edad >= 18) {
console.log("Tienes suficiente edad para alquilar un vehículo");
} else {
console.log("No tienes edad para alquilar un vehículo");
 }
```

  En este ejemplo, si la variable edad es menor de 18, nos devolverá el mensaje " No tienes edad para alquilar un vehículo ".

- else if: Nos permite agregar condicionales adicionales en el caso de que la primera condición no se cumpla.

Se pueden añadir tantos “else if” como sean necesarios y a diferencia de “else”, con “else if” si se deben añadir condiciones que se deban cumplir.

Ejemplo:

```
let edad = 25;
if (edad < 18) {
 console.log("No tienes edad para alquilar un vehículo");
}
else if (edad >= 18 && edad < 65) {
 console.log("Tienes suficiente edad para alquilar un vehículo");
}
else {
 console.log("Tienes demasiada edad para alquilar un vehículo");
}
```

En este ejemplo, se comprueban tres condiciones distintas, una dentro del “if” inicial y dos dentro del “else if”.

Las condicionales dan la posibilidad de poder anidarse una dentro de otra para poder tener más opciones dependiendo de lo que necesitemos que haga nuestro código y que información vaya a tener. Se pueden anidar tantas condicionales como se necesiten.

05 - ¿Qué es un operador ternario?

Un operador ternario es una forma abreviada de escribir una condicional en menos espacio que con la estructura habitual, es una declaración if-else que se define en una sola línea.

Es recomendable usarlo para condiciones cortas, y es especialmente útil para almacenar el output de la condicional en una variable, o para entornos en los que debemos introducir el código en una sola línea.

Si tu operador ternario es tan largo que comienza a requerir múltiples líneas, lo más recomendable será que utilices la sintaxis tradicional. Asimismo, es importante comprobar que el operador ternario sea legible y fácil de comprender.

Remarcar que se puede utilizar para múltiples condiciones, pero como hemos mencionado anteriormente requeriría múltiples líneas dificultando la lectura del código, por lo que no es la opción más recomendable.

Veamos el ejemplo de esta condicional con sintaxis clásica:

```
role = 'guest'


if role == 'admin':
  auth = 'can access'
else:
  auth = 'cannot access'


print(auth)
```

Y este sería el operador ternario equivalente:

```
role = 'guest'


auth = 'can access' if role == 'admin' else 'cannot access'
print(auth)
```

06 - ¿Cuál es la diferencia entre una declaración de función y una expresión de función?

En JavaScript tenemos dos maneras principales de definir funciones, como una declaración o como una expresión.

- Declaración de función

Es probablemente la más utilizada y la más fácil de recordar. Consiste en declarar la función con un nombre y sus parámetros de entrada entre paréntesis.

Las declaraciones de funciones pueden ser ejecutadas antes de su definición desde cualquier parte del código sin necesidad de tener que hacer la llamada por debajo de la propia función gracias al Hoisting.

Las declaraciones de funciones se realizan a la derecha y no deben utilizarse dentro de un bloque (por ejemplo, dentro de las llaves {} en una condicional). Técnicamente funcionan, pero pueden derivar en comportamientos inesperados.

```
function suma1(a, b) {
  return a + b
}
```

- Expresión de función

Consiste básicamente en guardar una función en una variable, para así ejecutar la variable como si fuera una función, declarándose a la izquierda del código.

Este nuevo recurso ha dado pie a las funciones anónimas ya que a las declaraciones de funciones se les asigna un nombre, pero no a las funciones de expresión.

Con las expresiones de función, puede utilizar una función inmediatamente después de definirla, por lo tanto, deben ser declaradas antes de ser llamadas y, hay que esperar a que se haya analizado todo el script.

Además, las expresiones de función pueden asignarse como valores a variables, pasar como argumentos a otras funciones, etc.

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

07 - ¿Cuál es la palabra clave 'this'?

En JavaScript, la palabra clave 'this' se refiere al contexto o ámbito actual dentro del cual se está ejecutando el código. Su valor viene determinado por cómo se llama a una función, y puede cambiar dinámicamente en función del contexto de invocación.

Hay cuatro maneras principales en las que se puede utilizar la palabra clave "this":

1. Dentro de un método de un objeto. Cuando se utiliza "this" dentro de un método de un objeto, hace referencia al objeto que contiene ese método.

2. En una función normal. Si se utiliza "this" dentro de una función normal, su valor será el objeto global.

3. En un constructor de objetos (usando la palabra clave "new"): Cuando se crea un objeto usando la palabra clave "new", el valor de "this" dentro del constructor hace referencia al objeto recién creado.

4. En un evento: Cuando se utiliza "this" dentro de un controlador de eventos, hace referencia al elemento que desencadenó el evento.