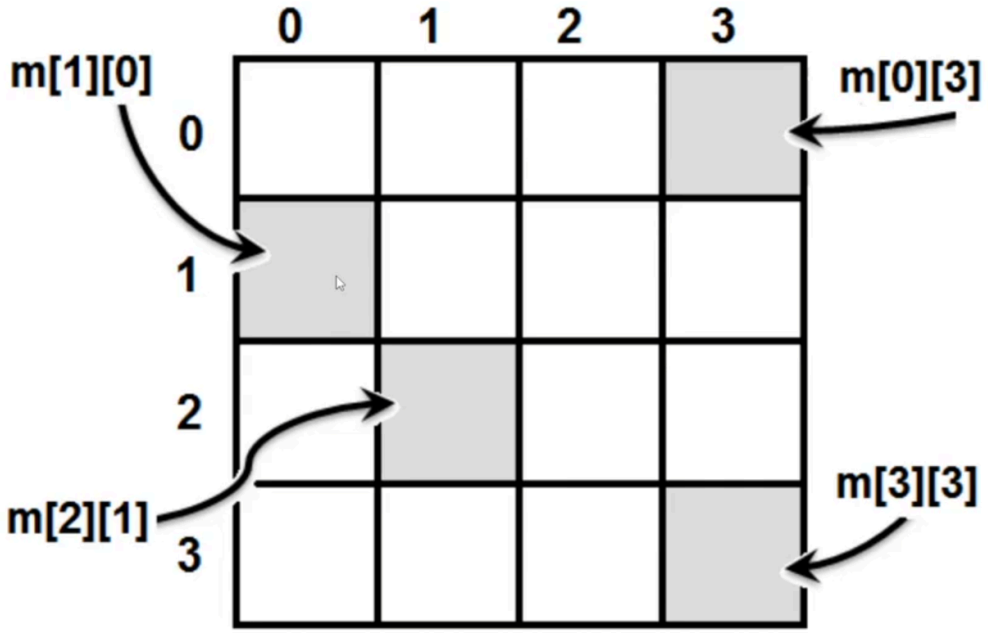
1. Matrices
    1.1. Introducción a matrices

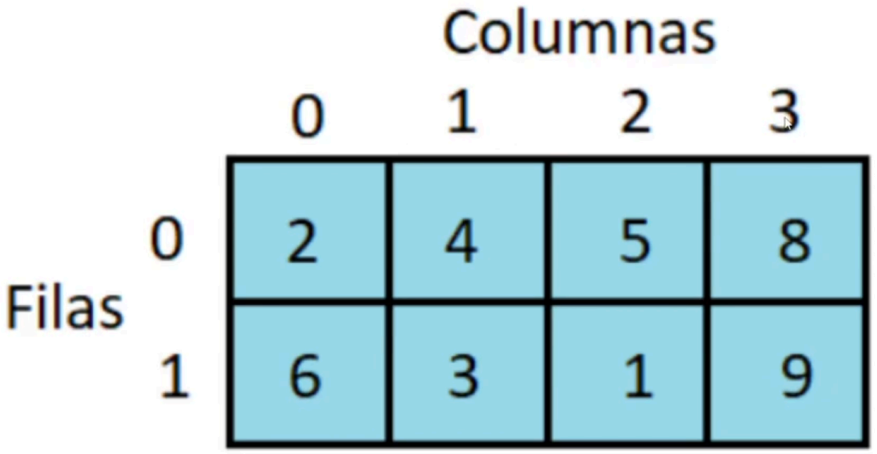
(Ver Archivo)
Suele ser información que se puede plasmar en forma de tabla.



Es una variable que contiene un cierto conjunto de datos. Tiene una estructura matricial donde se cargan distintos elementos ubicados en lugares específicos relacionados a un número de fila y un número de columna determinado. Es decir que existe una relación entre la ubicación de un dato con la información que la fila y la columna estén proporcionando. De esta manera, al trabajar con matrices, se va a contar con una cierta cantidad de filas y columnas.

Las matrices son arreglos dentro de arreglos, contando con un arreglo padre o superior que en cada una de sus posiciones contiene otros subarreglos que van a armar cada fila.

Para poder acceder a un dato, se deberá proporcionar un número de fila y un número de columna específicos haciendo uso de un doble corchete, siendo el primero el número de fila, y el segundo el número de la columna.



let matriz = [[primera fila], [segunda fila], …];
Manera de crear una matriz, cada subarreglo dentro del arreglo hace referencia a una fila.

console.table(matriz);
Recibe una matriz como parámetro, aunque puede recibir cualquier dato. Loguea por consola datos en forma de tabla.

matriz[índice fila][índice columna];
Para acceder a datos de una matriz se lo hace desde afuera hacia adentro: primero la matriz, luego el doble corchete donde se ingresa, primero el índice correspondiente a la fila, y el segundo índice se refiere al número de columnas.

matriz[índice fila][índice columna] = valor;
Cambiar datos específicos.

1.2. Recorrer matrices

(Var Archivo)

1.3. Filtrar dentro de matrices

(Ver Archivo)

2. Avanzando con funciones

2.1. Tipos de funciones

(Ver Archivo)

- Funciones declaradas:

Es la forma clásica y normal de crear funciones. Pueden ser invocadas antes de su declaración, es decir, antes de crearlas, debido a un comportamiento llamado **hoisting**.

El **hoisting** (o elevación), que permite a ciertos elementos de JavaScript (funciones declaradas y variables **var**) se eleven al principio del ámbito en el que están definidas.

Además proporcionan mayor claridad a la hora de la lectura del código.

```
function nombre(parámetros) {
        instrucciones;
};
```

- Funciones expresadas:

Se asignan a una variable o se utilizan como parte de una expresión, lo que facilita la posibilidad de pasarlas como argumento a otras funciones o es necesario almacenarlas como un valor. Además facilita el control de la visibilidad, ya que su alcance se limita al contexto en el que se definen evitando “contaminar” el ámbito global.

```
let nombre = function(parametros) {
        instrucciones;
};
```

Tienen la característica de que no se pueden invocar antes de su definición, ya que siguen las reglas normales de asignación de variables.

- Arrow functions:

Las funciones flecha son una forma más breve y concisa de escribir funciones. Ofrecen una forma más breve de expresarlas.

```
let nombre = (parámetros) => {
        instrucciones;
};
```

No tienen su propio contexto **this**. Tampoco se elevan o tienen **hoisting**.
La propiedad o elemento **this** está determinado por la forma en que la función es llamada, por ejemplo, si la función es llamada como método de un objeto, this hace referencia a ese objeto.

Las funciones flecha heredan el objeto **this** del contexto donde están definidas, por lo que si se las usan como método de un objeto, no accederán al contexto del objeto si no al **this** de su ámbito global o del ámbito en que estén definidas.

2.2. Callbacks

(Ver Archivo)
Son funciones que se envían como parámetros de otras funciones y se ejecutan luego de que finalice la ejecución de una instrucción asincrónica.
La utilidad que brinda utilizar callbacks se relaciona principalmente con la reutilización de código y la flexibilidad para realizar diferentes operaciones sobre los elementos, por ejemplo, de un array, sin alterar la función que llama a los callbacks.
Al utilizar un callback dentro de una función, genera que esa función se vuelva más genérica y versátil, ya que no se limita a realizar una acción específica, lo que significa que se puede pasar por parametro diferentes métodos de callback para realizar diferentes operaciones sin tener que modificar la lógica interna de la función.
Proporcionan flexibilidad y modularidad al código, permitiendo que una función pueda realizar diferentes acciones sin necesidad de modificar su estructura interna, haciendo un código fácil de extender y mantenible.

3. Arrays y más arrays

3.1. Métodos de arrays avanzados

(Ver Archivo)

array.slice(inicio, fin);
Retorna una copia superficial de una porción de un array existente desde el índice indicado como primer parámetro, hasta el índice indicado como segundo parámetro opcional.
Si solo se indica un parámetro, considera desde el inicio del arreglo, hasta el final.
Si los parámetros indican valores mayores a la longitud del arreglo, devuelve uno vacío.
El segundo parámetro de fin no contempla el elemento incluido.
Valores negativos, empieza a contar desde atrás hacia adelante, manteniendo la ubicación original del arreglo.

array.splice(incio, cantidad, elemento, elemento, …);
Combina el contenido de un array eliminando o reemplazando elementos existentes y/o, opcionalmente, agregando nuevos elementos en su lugar.
El parámetro de inicio indica desde donde se van a realizar los cambios.
El segundo parámetro es la cantidad de elementos a eliminar o reemplazar a partir del parámetro de inicio. Si se define en 0, no se va a eliminar ningún elemento y simplemente se agregarán los valores de los parámetros subsiguientes.
Los demás parámetros, opcionales, son los valores que se van a cargar en el array, luego de establecer el inicio desde donde se va a reemplazar hasta la cantidad de elementos a reemplazar.

array.sort();
Ordena los elementos de un array retornando un arreglo ordenado.
El ordenamiento funciona mediante el Código ASCII, que es la representación que tiene cada símbolo, que se puede ingresar en el código, a un valor numérico por el cual la computadora lo procesa, por ejemplo, **a** tiene el valor decimal de 97, y **z** el valor decimal de 122, por lo que **sort()**, al considerar esos números, ordena de esa manera los caracteres.

array.find(función(elemento));
Retorna el primer elemento que cumpla con una condición dada en una función de prueba.

3.2. Más métodos de arrays

(Ver Archivo)

array.map(callback);
Recibe una función como parámetro, y lo que hace es recorrer un arreglo y devolver un nuevo arreglo con las modificaciones indicadas en la función de callback.
La función de callback recibe un parámetro que hace referencia a cada uno de los elementos del arreglo y un segundo parámetro opcional que hace referencia al índice o posición del elemento.

array.filter(callback);
Recorre un arreglo y filtra según la condición indicada en el callback, retornando un nuevo arreglo.

array.reduce(callback(acumulador, elemento));
También recorre todos los elementos del arreglo, retornando un único valor entero, en función de callback definido como parámetro. Reduce los elementos del arreglo a uno solo según el criterio del callback.
El callback recibe dos valores como parámetros: el primero es un acumulador y el segundo son cada uno de los elementos del arreglo.

array.forEach(callback);
Método para iterar sobre un array, similar a un **for**. A diferencia de los métodos anteriores, **forEach** no retorna nada y el comportamiento dado se lo define en el callback. El primer parámetro refiere al elemento, y un segundo parámetro opcional que refiere al índice o posición del elemento.

4. Integración de contenidos
    4.1. Desafío Práctico

Matrices

Se está realizando el desarrollo de una aplicación para control de gastos. Cada día, el usuario ingresa sus gastos cotidianos. La idea es solo registrar el total de los gastos, al finalizar la jornada. Para simplificar, vamos a considerar que todos los meses tienen cuatro semanas. Los gastos estarán en una matriz de 4x7, cada fila representa una semana y cada columna un día. Es decir fila 0, semana 1, fila 1, semana 2, etc. Columna 0, dia 1, columna 1, dia 2, etcétera.

- Gastos de toda la semana:

Crear una función que nos sirva para obtener el total de gastos de una semana específica. Recordemos que cada fila representa una semana. El número de semana deberá recibirse por parámetro.

- Gastos de un día de la semana:

Crea una función que nos sirva para obtener el total de gastos de un día específico de la semana. Por ejemplo, del gasto de todos los martes del mes, o el gasto de todos los sábados del mes. El número correspondiente al día deberá recibirse por parámetro. Recuerda que el número de día es determinado por las columnas.

- Gasto total:

Crea una función que nos sirva para obtener el valor total de gastos de todo un mes. La función deberá retornar dicho valor.

Callbacks

Gastos por semana:

Crea una función que calcule el total de gastos de cada semana y retorne un array con estos totales. En la posición 0 estará el gasto de la semana 0, en la posición 1 el gasto de la semana 1, etc. Será obligatorio el uso de callbacks para esta función.

    4.2. Resolución de desafío práctico

(Ver Archivo)

5. Implementación avanzada de conocimientos I
    5.1. Avanzando el proyecto integrador - Parte I

(Ver Archivo)

    5.2. Avanzando el proyecto integrador - Parte II

(Ver Archivo)

    5.3. Avanzando el proyecto integrador - Parte III

(Ver Archivo)

6. Checkpoint de conocimientos

- ¿Cuál es la sintaxis correcta de una función de flecha que toma dos parámetros y devuelve su suma?

let sum = (a, b) => { return a + b; }

- ¿Cuál es la principal diferencia entre una función tradicional y una función de flecha en JavaScript?

Las funciones de flecha no tienen acceso al objeto this.

- ¿Qué son los callbacks en JavaScript?

Una función que se pasa como argumento a otra función y se ejecuta después.

- ¿Qué es una matriz en JavaScript?

Una lista ordenada de elementos del mismo tipo que se almacenan bajo un solo nombre.

- ¿Cómo se accede a un valor de una propiedad dentro de un objeto en JavaScript?

objeto.valor

- ¿Cómo se agrega una nueva propiedad a un objeto literal existente?

objeto.nuevaPropiedad = valor;

- ¿Qué hace el siguiente código?

```
let numeros = [2,4,6];

let resultado = numeros.map( function(num){
    return num * 2;
});

console.log(resultado);
```

Guarda en la variable ‘resultado’ un array con los elementos del array ‘numeros’ multiplicados por dos.

- ¿Qué hace el siguiente código?

```
let numeros = [2,4,6];

let resultado = numeros.reduce( function(acum, num){
    return acum + num;
});

console.log(resultado);
```

Guarda en la variable ‘resultado’ la suma de todos los elementos del array ‘numeros’.