1. Ciclos: Repetir… repetir… repetir

Las estructuras repetitivas permiten repetir una serie de instrucciones desarrolladas anteriormente y que es necesario ejecutarla más de una vez, evitando escribir la misma instrucción reiteradas veces evitando un código repetitivo, difícil de entender y sucio.

1.1. While

(Ver Archivo)

while se utiliza cuando la cantidad de veces que se tiene que repetir una acción no es fija o se pueda definir dentro de las instrucciones que se están realizando, y también es útil para cuando la condición, que indica cuándo continuar ejecutando o repitiendo las acciones, puede cambiar dentro del bloque brindado.

```
while(variable condición) {
        instrucciones;
        cambio;
};
```

- Variable de control:

Es una variable inicializada (tiene un valor en su declaración) que se utiliza para evaluar la cantidad de veces que se repite una serie de instrucciones, es decir, cuántas iteraciones se realizarán.

- Condición:

Evalúa el valor del contador para seguir ejecutando o no las instrucciones.

- Cambio de la variable contadora:

Instrucción dentro del bloque while que cambia el valor del contador en cada iteración.

El flujo consiste en que el código lee la variable de control y luego ejecuta el ciclo **while**. Una vez que termina la ejecución de la instrucción, vuelve a evaluar la variable y su condición, si esta sigue siendo verdadera, vuelve a ejecutar las instrucciones dentro del **while**. Vuelve a repetir el ciclo de lectura del contador junto a su condición hasta que esta sea falsa.

Ciclo repetitivo o bucle infinito: sucede cuando la condición nunca llega a ser falsa, por lo que el ciclo se repite indefinidamente, de ahí la importancia de siempre evaluar una condición.

1.2. Do While

(Ver Archivo)

Es Familiar de la estructura **while**, con la diferencia en la forma en que se evalúan las condiciones. **do while** primero itera una vez y luego valida la condición.

```
do {
        instrucciones;
        cambio;
} while(variable condición);
```

Contiene los mismos elementos de la estructura **while** (variable de control, condiciones y el cambio en la variable de control), solo que las ejecuta de otra manera. La diferencia radica en que siempre ejecuta al menos una instrucción aunque la condición sea falsa.

1.3. For loop

(Ver Archivo)

También permite repetir instrucciones con la característica de que se suele utilizar cuando se conoce de antemano la cantidad de repeticiones que se requieran realizar. En la estructura **for**, los elementos necesarios se declaran dentro de los paréntesis **()**.

```
for(inicialización ; condición; cambio) {
        instrucciones;
}
```

La variable de control puede inicializarse o declararse tanto dentro como fuera de la sintaxis del **for**.

Primero ejecuta la inicialización, luego consulta la condición, y finalmente realiza el cambio. Dadas esas condiciones, ejecuta las instrucciones, luego vuelve a iterar hasta que se cumple la condición.

2. Strings y arrays: Trabajando con colecciones

2.1. Intro arrays

(Ver Archivo)

Los arrays o arreglos son una colección ordenada de elementos, lo que significa que, bajo una sola variable, se pueden cargar distintos valores y mantenerlos siempre ordenados en la misma dirección de memoria.

Los arreglos se guardan en fila o pila, ordenados en función a un índice, los cuales comienzan en 0, es decir, el primer elemento se guarda en la posición o índice 0, el segundo bajo el índice 1 y así sucesivamente. Pueden guardar diferentes tipos de datos.

Los arreglos no son variables primitivas, sino que son más complejas, y se declaran con corchetes, y cada dato dentro del array se separa con comas:

```
let variable = [dato0, dato1, dato2, …];
```

Para acceder a un dato específico del array, se llama a la variable y se marca la posición o índice del elemento al cual se quiere acceder entre corchetes:

```
variable[índice];
```

Para obtener la cantidad de elementos o longitud que tiene un arreglo se utiliza la propiedad **length**:

```
array.length;
```

2.2. Algoritmos clásicos con arrays

(Ver Archivo)

Se puede modificar el valor una posición del arreglo accediendo a su índice y asignando otro valor:

```
array[indice] = nuevo valor;
```

Para agregar valores accediendo también a la cantidad de elementos que hay en el arreglo, como lenght obtiene la cantidad total, esta hace referencia al último índice más uno:
variable[variable.length] = valor;

2.3. Métodos de arrays

(Ver Archivo)
Un arreglo es un tipo de dato que proviene de una estructura llamada *objeto*. Al trabajar con objetos se cuenta con distintos métodos que son acciones que permiten trabajar sobre esa información.Estos métodos son funciones que están previamente definidas para manipular estas estructuras:

array.push(valor, valor, …);
Agrega uno o varios elementos al final del arreglo. Permite una forma más dinámica de poder agregar elementos. Puede recibir uno o más elementos como parámetros. Retorna la nueva longitud del array.

array.pop();
Elimina el último elemento de un arreglo. Retorna el elemento eliminado.

array.shift();
Elimina el primer elemento de un arreglo. Retorna el elemento eliminado.

array.unshift();
Agrega elementos al principio del arreglo. Retorna la cantidad de elementos luego del **unshift**.

array.join(separador);
Une los elementos de un array utilizando el separador que se defina como parámetro, o con comas si no se define ninguno. Retorna un *string* con todos los elementos del array unidos.

array.indexOf(elemento);
Método que busca en el arreglo el elemento que se le pase como parámetro. Retorna el índice del elemento que se le pase como parámetro y si no lo encuentra, retorna **-1**, debido a que como los arreglos tienen índices de 0 hacia arriba, un número negativo no puede ser un índice, indicando que no pudo localizarlo. En caso de haber repetidos, devuelve la posición del primero que encuentre.

array.lastIndexOf(elemento);
También busca elementos en el arreglo, con la diferencia que empieza buscando desde atrás hacia adelante, es decir, desde el último elemento hasta el primero. En caso de no encontrarlo, devuelve **-1**, y en caso de haber repetidos, devuelve la posición del primero que encuentre.

array.includes(elemento);
Otro método para encontrar elementos. Retorna un booleano, por lo que devuelve **true** si lo encuentra, y **false** si el elemento no existe.

2.4. Métodos de strings

(Ver Archivo)

Las cadenas de texto o strings son muy similares a los arreglos, es decir, cada carácter dentro de la cadena también tiene un índice o una posición o una longitud y pueden ser accedidos o manipulados como los arreglos. Al trabajar con strings, lo que se hace es cargar una colección o conjunto de caracteres a una variable, que tienen un orden específico que se debe respetar para poder formar una palabra. Un string es un arreglo de caracteres.

string[posición];
Devuelve el carácter definido en la posición. Si la posición no existe, retorna **undefined**.

string.length;
Propiedad que retorna la cantidad de caracteres que contiene el string. Incluye los espacios.

string.indexOf(valor);
Busca el string que se le pase como parámetro y retorna el índice o posición del primer valor que encuentre, sino devuelve **-1**. En estos casos, si se le pasa un solo carácter retorna el índice de la primera coincidencia. Si se le pasa una subcadena que exista, devuelve el índice a partir del cual esa subcadena comienza.

array.slice(incio, corte);
Corta la cadena de texto y retorna una parte del string en donde se le haya indicado. Recibe como primer parámetro obligatorio un índice que indica dónde iniciar el corte y como segundo parámetro opcional otro índice que indica dónde finalizar el corte. Considerar que el valor del segundo parámetro no lo retorna, es decir que devuelve hasta un valor antes del índice de corte. Por otro lado, si no se declara el parámetro de corte, retorna desde el inicio hasta el final de la cadena. Finalmente, se pueden pasar valores negativos, en este caso, empieza a contar desde atrás hacia adelante, siendo **-1**, el último valor de la cadena.

array.trim();
Elimina espacios al inicio y al final de un string, aunque mantiene espacios que tenga la cadena entre subcadenas.

array.split(separador);
Divide un string en distintas partes. Recibe por parámetro un **string** que lo utilizará como separador y retorna un arreglo con las distintas partes de la cadena, separando la cadena entre lo que identifique en el separador. Si el separador no se encuentra en la cadena, devuelve el arreglo con el string completo.

array.replace(string, reemplazo);
Reemplaza una parte del string por otra. Recibe como primer parámetro el string dentro de la cadena que se quiere reemplazar, y como segundo parámetro otro string que va a ser el reemplazo del primero. Retorna un nuevo string con la cadena reemplazada. No modifica el string original por si solo. Si no se encuentra el primer parámetro, se devuelve la cadena original.

3. Modelando la vida real
    3.1. Objetos Literales

(Ver Archivo)

Los objetos literales son la representación en código de elementos de la vida real que no se pueden representar con tipos de datos primitivos. Es una estructura que contiene varios datos representados bajo propiedades y/o métodos. Las propiedades son todas aquellas características que comparten los objetos modelados y los métodos son todas las acciones que el objeto pueda realizar. Cualquier propiedad o método pueden ser guardados en variables.

```
let objeto = {
        propiedad: valor,
        …,
        método: function() {instrucción},
        …,
};
```

El modelado de objetos consiste en que tanto las propiedades como los métodos se declaran con un nombre o clave (**key**) que los identifique, y luego con dos puntos (**:**) se define el valor de cada uno, separándolos por una coma (**,**). Las propiedades pueden guardar cualquier tipo de objeto sean primitivos, arreglos y hasta otros objetos.

objeto.propiedad/método();
Accede al valor de la propiedad o método modelado. Si no existe, devuelve **undefined**.

objeto.propiedad/método() = nuevoValor;
Accede al valor de la propiedad definida y modifica su valor.

objeto.nuevaPropiedad/nuevoMetodo() = valor;
Crea una nueva propiedad o método al objeto.

delete objeto.propiedad;
Elimina la propiedad o método del objeto.

this.propiedad/método();
Palabra reservada de los objetos que permite acceder a propiedades de un objeto dentro del mismo objeto.

4. Integración de contenidos
    4.1. Desafio práctico

- Unir dos arrays:

Crea una función llamada ‘unirArrays’ que tome dos arrays como parámetros y los una en uno solo. La función debe devolver el nuevo array.Para esto, averiguar qué hace el método .concat().

array.concat(array);
Une los elementos del arreglo con los elementos del arreglo definido como parámetro en un nuevo arreglo.

- Eliminar primer elemento:

Crea una función llamada ‘eliminarPrimerElemento’ que tome un array como parámetro y elimine el primer elemento. La función debe devolver el elemento eliminado.

- Convertir texto:

Crea una función que reciba un string por parámetro y lo convierta a mayúsculas y minúsculas respectivamente. La función deberá mostrar los resultados por consola. Para esto, averiguar qué hacen los métodos de Strings: toUpperCase() y toLowerCase().

string.toUpperCase();
Devuelve el valor de la cadena convertida a mayúsculas. No modifica la cadena original.

string.toLowerCase();
Devuelve el valor de la cadena convertida a minúsculas. No modifica la cadena original.

- Manipulación de objetos:

Crea un objeto llamado ‘persona’ que contenga como propiedades: nombre, edad y ocupación. Crea una función que muestre cada propiedad de dicho objeto por consola, e invocarla para ver sus valores. Luego, modifica el valor de ‘ocupación’ y agrega la propiedad ‘habilidad’. Llama nuevamente a la función creada para visualizar el cambio efectuado.

- Objetos anidados:

Agrega una nueva propiedad al objeto creado en el ejercicio anterior llamada ‘ubicación’, donde deberás guardar un objeto interno con las propiedades: codigoPostal, ciudad, calle y número. Luego, crea una función que reciba dicho objeto por parámetro y le permita al usuario eliminar solo una de las propiedades del objeto.

4.2. Resolución de desafío práctico

(Ver Archivo)

5. Desarrollo y Aplicación de Competencias

5.1. Inicio de proyecto integrador

(Ver Archivo)

array.splice(índice, cantidad, elemento);
Cambia el contenido de un array eliminando elementos existentes y/o agregando nuevos elementos. El primer parámetro es el índice del elemento que se quiere eliminar. El segundo parámetro hace referencia a la cantidad de elementos que se quieren eliminar luego del índice. Puede recibir un tercer parámetro que es un elemento que se reemplaza por el elemento eliminado. Retorna un arreglo con los elementos eliminados.

5.2. Continuación proyecto integrador

(Ver Archivo)

6. Checkpoint de conocimientos

- ¿Cuál es la diferencia principal entre while y do-while?

“do - while” siempre se ejecuta al menos una vez, mientras que “while” no.

- ¿Cómo se accede al primer elemento de un arreglo en JavaScript?

arreglo[0]

- ¿Qué método nos servirá para eliminar el último elemento de un arreglo en JavaScript?

.pop()

- ¿Qué método de cadena se usa para convertir una cadena a minúsculas?

.toLowerCase()

- ¿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 numbers = [1, 2, 3, 4, 5];
let total = 0;

for (let i = 0; i < numbers.length; i++) {
    total += numbers[i];
}
```

Suma todos los números en el arreglo numbers

- ¿Qué hace el siguiente código?

```
let count = 0;

while (count < 5) {
    console.log(count);
    count++;
}
```

Muestra los números del 0 al 4, la variable “count” termina valiendo 5.

- ¿Qué hace el siguiente código en JavaScript?

```
let colors = ['red', 'green', 'blue'];

colors.push('yellow');
```

Agrega un nuevo color al final del arreglo “colors”.

- ¿Qué valor tendrá typeOfAnimal después de ejecutar este código?

```
let animal = {
    type: 'Dog',
    age: 3,
    breed: 'Labrador'
};

let typeOfAnimal = animal.type;
```

'Dog'

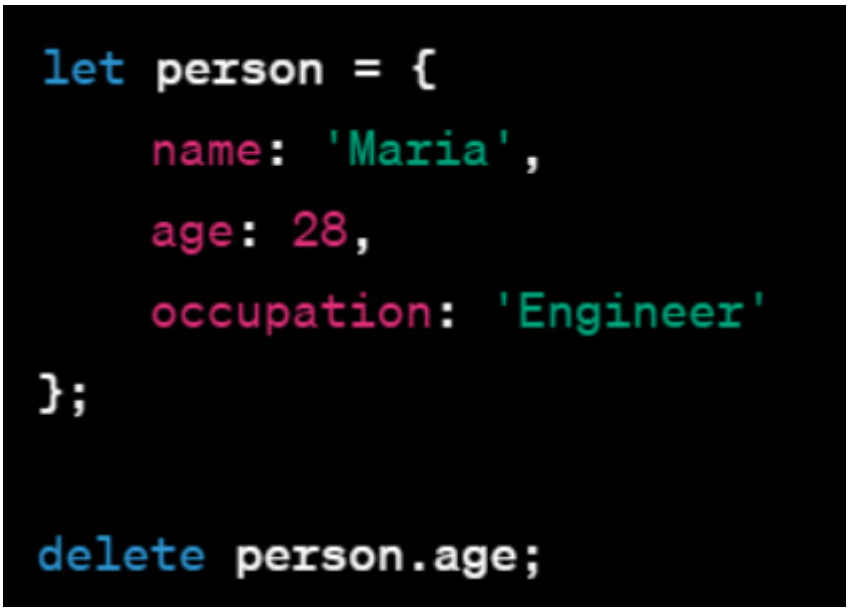
- ¿Qué se imprimirá en la consola al ejecutar este código?

```
let x = 10;

while (x > 5) {
    console.log(x);
    x -= 2;
}
```

Se imprimirá: 10, 8, 6

- ¿Qué hace el siguiente código?



```
let person = {
    name: 'Maria',
    age: 28,
    occupation: 'Engineer'
};

delete person.age;
```

Elimina la propiedad 'age' del objeto person.