listas(JSON) y arrays

Las listas y los arrays son estructuras de datos que te permiten almacenar una colección de elementos en una sola variable. Sin embargo, hay algunas diferencias importantes entre ellos.

Un array en JavaScript es una colección ordenada de elementos que se almacenan en una variable. Cada elemento del array tiene una posición numérica única, llamada índice, que comienza en cero. Puedes acceder a cada elemento del array utilizando su índice. Aquí te dejo un ejemplo:

```
// Crear un array con algunos elementos
let miArray = ["manzana", "banana", "naranja", "pera"];

// Acceder al primer elemento del array
console.log(miArray[0]); // Output: "manzana"

// Acceder al tercer elemento del array
console.log(miArray[2]); // Output: "naranja"
```

Las listas en JavaScript son similares a los arrays, pero no tienen índices numéricos fijos. En su lugar, los elementos se almacenan en una estructura de datos llamada objeto. Cada elemento de la lista se almacena en un par clave-valor, donde la clave es un identificador único para el elemento y el valor es el elemento en sí. Aquí tienes un ejemplo:

```
// Crear una lista con algunos elementos
let miLista = {
  "fruta1": "manzana",
  "fruta2": "banana",
  "fruta3": "naranja",
  "fruta4": "pera"
};
// Acceder al primer elemento de la lista
console.log(miLista.fruta1); // Output: "manzana"
// Acceder al tercer elemento de la lista
console.log(miLista.fruta3); // Output: "naranja"
```

Una de las principales ventajas de las listas es que te permiten asignar identificadores más descriptivos a cada elemento, en lugar de tener que recordar los índices numéricos.

En resumen, las listas y los arrays son estructuras de datos comunes en JavaScript que te permiten almacenar colecciones de elementos. Los arrays son una colección ordenada de elementos que se almacenan en una variable y se acceden a través de índices numéricos, mientras que las listas son una colección de elementos que se almacenan en una estructura de objeto y se acceden a través de identificadores descriptivos.

Ciclos

Los ciclos son una herramienta muy útil en JavaScript para trabajar con listas y arrays. Aquí te muestro cómo puedes utilizarlos para recorrer una lista o un array en JavaScript.

Ciclos for en Arrays:
Para recorrer un array, puedes utilizar un ciclo for. Un ciclo for se utiliza para iterar a través de una lista de elementos. En JavaScript, puedes utilizar un ciclo for para recorrer un array y realizar alguna acción en cada uno de sus elementos. Aquí tienes un ejemplo:

```
// Crear un array con algunos elementos
let miArray = ["manzana", "banana", "naranja", "pera"];

// Recorrer el array utilizando un ciclo for
for (let i = 0; i < miArray.length; i++) {
  console.log(miArray[i]);
}
// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

Este código recorre el array miArray utilizando un ciclo for y utiliza el índice i para acceder a cada elemento del array.

Ciclos for-in en Listas:
Para recorrer una lista en JavaScript, puedes utilizar un ciclo for-in. Un ciclo for-in se utiliza para iterar a través de las propiedades de un objeto. En JavaScript, puedes utilizar un ciclo for-in para recorrer una lista y realizar alguna acción en cada uno de sus elementos. Aquí tienes un ejemplo:

```
// Crear una lista con algunos elementos
let miLista = {
  "fruta1": "manzana",
  "fruta2": "banana",
  "fruta3": "naranja",
  "fruta4": "pera"
};

// Recorrer la lista utilizando un ciclo for-in
for (let key in miLista) {
  console.log(miLista[key]);
}
// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

Este código recorre la lista miLista utilizando un ciclo for-in y utiliza la clave key para acceder a cada elemento de la lista.

Ciclos forEach en Arrays:
Otra forma de recorrer un array en JavaScript es utilizando el método forEach. El método forEach se utiliza para recorrer un array y realizar alguna acción en cada uno de sus elementos. Aquí tienes un ejemplo:

```
// Crear un array con algunos elementos
let miArray = ["manzana", "banana", "naranja", "pera"];

// Recorrer el array utilizando el método forEach
miArray.forEach(function(elemento) {
  console.log(elemento);
});
// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

Este código recorre el array miArray utilizando el método forEach y utiliza la función anónima para acceder a cada elemento del array.

Ciclo while en Arrays:
El ciclo while se utiliza para repetir una acción mientras se cumpla una condición. En JavaScript, puedes utilizar un ciclo while para recorrer un array y realizar alguna acción en cada uno de sus elementos. Aquí tienes un ejemplo:

```
// Crear un array con algunos elementos
let miArray = ["manzana", "banana", "naranja", "pera"];

// Recorrer el array utilizando un ciclo while
let i = 0;
while (i < miArray.length) {
  console.log(miArray[i]);
  i++;
}
```

```
// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

Este código recorre el array miArray utilizando un ciclo while y utiliza el índice i para acceder a cada elemento del array.

Ciclo while en Listas:
El ciclo while se puede utilizar para recorrer una lista en JavaScript. A diferencia de un array, una lista en JavaScript no es un tipo de datos incorporado, pero puedes utilizar un objeto para crear una lista. Por ejemplo:

```
// Crear una lista con algunos elementos
let miLista = {
  "fruta1": "manzana",
  "fruta2": "banana",
  "fruta3": "naranja",
  "fruta4": "pera"
};

// Recorrer la lista utilizando un ciclo while
let keys = Object.keys(miLista);
let i = 0;
while (i < keys.length) {
  console.log(miLista[keys[i]]);
  i++;
}
// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

En este ejemplo, creamos una lista miLista utilizando un objeto, donde cada clave representa el índice y el valor representa el elemento. Luego, utilizamos Object.keys() para obtener un array con las claves de la lista y lo almacenamos en la variable keys. Finalmente, utilizamos un ciclo while para recorrer el array de claves y acceder a cada elemento de la lista utilizando la sintaxis miLista[keys[i]].

Es importante destacar que en una lista, el orden de los elementos no está garantizado, ya que JavaScript no define un orden específico para las propiedades de un objeto. Por lo tanto, cuando se utiliza un ciclo while para recorrer una lista, es posible que los elementos no se muestren en el mismo orden en el que se agregaron a la lista.

Ciclo do-while en Arrays:

El ciclo do-while en JavaScript es una variante del ciclo while. La diferencia es que el código dentro del bloque de un ciclo do-while se ejecuta al menos una vez, sin importar si la condición del ciclo es verdadera o falsa. Después de la primera ejecución, la condición se evalúa y si es verdadera, el bloque se ejecuta nuevamente y esto continúa hasta que la condición sea falsa.

El ciclo do-while es útil cuando se quiere ejecutar una tarea al menos una vez, y luego repetir la tarea mientras se cumple una determinada condición. Por ejemplo, se puede utilizar para recorrer un array en JavaScript:

```
let miArray = [1, 2, 3, 4, 5];
let i = 0;

do {
  console.log(miArray[i]);
  i++;
} while (i < miArray.length);
```

```
// Output:
// 1
// 2
// 3
// 4
// 5
```

En este ejemplo, creamos un array miArray con algunos elementos y utilizamos un ciclo do-while para recorrer el array. La variable i se inicializa en 0, lo que significa que el ciclo comenzará por el primer elemento del array. El código dentro del bloque de do se ejecuta al menos una vez, lo que significa que se mostrará el primer elemento del array. Luego, la variable i se incrementa en 1, lo que significa que el ciclo volverá a ejecutar el bloque mientras i sea menor que el tamaño del array (miArray.length).

El ciclo do-while es útil cuando se quiere garantizar que un bloque de código se ejecute al menos una vez, antes de que se evalúe la condición de salida. En este caso, el ciclo do-while es más apropiado que el ciclo while regular.

Ciclo do-while en Listas:
El ciclo do-while es similar al ciclo while, pero se asegura de que la acción se ejecute al menos una vez antes de comprobar la condición. En JavaScript, puedes utilizar un ciclo do-while para recorrer una lista y realizar alguna acción en cada uno de sus elementos. Aquí tienes un ejemplo:

```
// Crear una lista con algunos elementos
let miLista = {
  "fruta1": "manzana",
  "fruta2": "banana",
  "fruta3": "naranja",
  "fruta4": "pera"
};
```

```
// Recorrer la lista utilizando un ciclo do-while
let keys = Object.keys(miLista);
let i = 0;
do {
  console.log(miLista[keys[i]]);
  i++;
} while (i < keys.length);


// Output:
// "manzana"
// "banana"
// "naranja"
// "pera"
```

Este código recorre la lista miLista utilizando un ciclo do-while y utiliza la clave keys para acceder a cada elemento de la lista.

EJERCICIOS

1. Crea un programa que solicite al usuario un número y luego imprima la suma de todos los números impares entre 0 y el número ingresado.

2. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine el número más grande y el más pequeño de la lista.

3. Crea un programa que solicite al usuario un número y luego imprima todos los números primos menores o iguales al número ingresado.

4. Crea un programa que solicite al usuario una lista de palabras separadas por coma y luego imprima la palabra más larga.

5. Crea un programa que solicite al usuario dos números y luego calcule el mínimo común múltiplo (mcm) de los dos números.

6. Crea un programa que solicite al usuario un número y luego determine si es un número perfecto o no. Un número es perfecto si la suma de sus divisores propios es igual al número.

7. Crea un programa que solicite al usuario un número y luego determine si es un número Armstrong o no. Un número Armstrong es aquel cuya suma de las potencias de sus dígitos es igual al número.

8. Crea un programa que solicite al usuario un número y luego imprima su tabla de multiplicar hasta el 12.

9. Crea un programa que solicite al usuario una palabra y luego determine si es un palíndromo o no. Una palabra es un palíndromo si se lee igual de derecha a izquierda que de izquierda a derecha.

10. Crea un programa que solicite al usuario un número y luego imprima el número en reversa. Por ejemplo, si el usuario ingresa el número 1234, el programa deberá imprimir 4321.

11. Crea un programa que solicite al usuario un Array de números y luego determine si estos son números primos o no.

12. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine cuántos de ellos son pares.

13. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine la suma de los números que están en posiciones pares de la lista.

14. Crea un programa que solicite al usuario un número y luego determine si es un número perfecto o no. Un número es perfecto si la suma de sus divisores propios es igual al número.

15. Crea un programa que solicite al usuario un número y luego determine si es un número feliz o no. Un número es feliz si la suma de los cuadrados de sus dígitos, iterada hasta que el resultado es 1, es igual a 1.

16. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine la mediana de la lista.

17. Crea un programa que solicite al usuario un número y luego determine si es un número de Fibonacci o no. Un número es de Fibonacci si es igual a la suma de los dos números anteriores en la secuencia de Fibonacci.

18. Crea un programa que solicite al usuario una lista de palabras separadas por coma y luego determine cuántas de ellas son palíndromos.
19. Crea un programa que solicite al usuario un número y luego determine si es un número feliz o no. Un número es feliz si la suma de los cuadrados de sus dígitos, iterada hasta que el resultado es 1, es igual a 1.

20. Crea un programa que solicite al usuario un número y luego determine si es un número de Harshad o no. Un número de Harshad es aquel que es divisible por la suma de sus dígitos.

21. Crea un programa que solicite al usuario una palabra y luego determine si es un palíndromo o no. Un palíndromo es una palabra que se lee igual de izquierda a derecha que de derecha a izquierda.

22. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine la media geométrica de la lista. La media geométrica se calcula como la raíz n-ésima del producto de los n números.

23. Crea un programa que solicite al usuario una lista de números separados por coma y luego determine el número de números que son divisibles por la suma de los dígitos del número.

24. Crea un programa que solicite al usuario un número y luego determine la cantidad de números primos que hay entre 1 y el número ingresado.

25. Crea un programa que solicite al usuario un número y luego determine cuántos dígitos tiene el número. Luego, el programa deberá solicitar al usuario una lista de números separados por coma, y solo permitirá ingresar números que tengan la misma cantidad de dígitos que el número ingresado inicialmente. Finalmente, el programa deberá determinar cuántos de los números ingresados son múltiplos del número inicial.


SENA