ARRAYS

¿Qué es un array?

Un array es una estructura de datos que te permite almacenar y organizar múltiples elementos en una sola variable. Los arrays son una de las estructuras de datos más fundamentales y utilizadas en la programación y son útiles para trabajar con conjuntos de datos.

Para crear un array en JavaScript, puedes usar la siguiente sintaxis:

```
// Array vacío
let miArray = [];

// Array con elementos
let miArrayConElementos = [1, 2, 3, 4, 5];
```

Los arrays en JavaScript son dinámicos, lo que significa que pueden cambiar de tamaño y puedes agregar o eliminar elementos en cualquier momento. También pueden contener diferentes tipos de datos en sus elementos, como números, cadenas, objetos y otras arrays.

Acceder a elementos del array

Puedes acceder a los elementos de un array utilizando su índice. El índice de un array comienza desde 0 para el primer elemento y aumenta en 1 para cada elemento subsiguiente. Para acceder a un elemento específico, simplemente proporciona el índice entre corchetes:

```
let miArray = [10, 20, 30, 40, 50];
```

```
console.log(miArray[0]); // Muestra 10
console.log(miArray[3]); // Muestra 40
```

Modificar elementos del array

Para modificar un elemento en el array, simplemente accede a su posición y asigna un nuevo valor:

```
let miArray = [10, 20, 30, 40, 50];

miArray[2] = 35; // Cambia el tercer elemento a 35

console.log(miArray); // Muestra [10, 20, 35, 40, 50]
```

Longitud del array

Puedes obtener la longitud de un array utilizando la propiedad length:

```
let miArray = [10, 20, 30, 40, 50];

console.log(miArray.length); // Muestra 5
```

Métodos comunes de arrays

Así como y trabajos con métodos para los strings (cadenas de caracteres), JavaScript proporciona una variedad de métodos integrados que facilitan la manipulación de arrays.

Algunos de los métodos más comunes son:

Métodos transformadores: llamamos métodos transformadores a los que generan un cambio en el array que estamos afectando.

- **push**(): Agrega uno o más elementos al final del array.
- **pop**(): Elimina el último elemento del array.
- **shift**(): Elimina el primer elemento del array.
- **unshift**(): Agrega uno o más elementos al inicio del array.

- **reverse**(): Se utiliza para revertir el orden de los elementos de un array
- **splice**(): Permite agregar, eliminar o reemplazar elementos en cualquier posición del array.
- **sort**(): Se utiliza para ordenar los elementos de un array. Por defecto, el método sort() ordena los elementos en función de sus representaciones de cadena Unicode (UTF-16). Esto significa que, para los elementos que son cadenas, se realizará una ordenación alfabética. Para los elementos que son números, también se realizará una ordenación de menor a mayor.

Métodos accesores:

- **slice**(): Devuelve una copia de una porción del array original, especificada por los índices de inicio y fin.
- **join**(): Se utiliza para crear una nueva cadena concatenando todos los elementos de un array. Permite especificar un separador opcional que se utilizará entre los elementos mientras se los une en la cadena resultante. Si no se proporciona ningún separador, los elementos se concatenarán sin ningún carácter intermedio.
- **indexOf**(): Busca un elemento y devuelve su índice en el array.
- **includes**(): Verifica si un elemento está presente en el array y devuelve true o false.

Métodos de repetición/iteración:

- **filter**(): Crea un nuevo array con todos los elementos que cumplan con la condición dada.
- **forEach**(): Se utiliza para iterar sobre los elementos de un array y ejecutar una función de devolución de llamada (callback) para cada elemento

Estos son solo algunos de los muchos métodos disponibles para trabajar con arrays en JavaScript.

Ejemplos

✏️ push()

```
// Definimos un array vacío
let miArray = [];

// Agregamos elementos al array usando push()
miArray.push(10);
miArray.push(20);
miArray.push(30);

console.log(miArray); // Muestra: [10, 20, 30]
```

En este ejemplo, primero creamos un array vacío llamado **miArray**. Luego, utilizamos el método **push()** para agregar tres elementos **(10, 20 y 30)** al final del array. Después de ejecutar las operaciones, el array contiene **[10, 20, 30]**.

Puedes usar **push()** para agregar uno o más elementos a un array. Si deseas agregar más de un elemento en una sola llamada, simplemente agrégales como argumentos separados por comas:

```
let miArray = [1, 2, 3];

miArray.push(4, 5, 6);

console.log(miArray); // Muestra: [1, 2, 3, 4, 5, 6]
```

✏️ pop()

```
let miArray = [10, 20, 30, 40, 50];

// Utilizamos pop() para eliminar el último elemento
let ultimoElemento = miArray.pop();

console.log(ultimoElemento); // Muestra 50
console.log(miArray); // Muestra [10, 20, 30, 40]
```

En este ejemplo, el método pop() elimina el último elemento del array miArray, que es 50, y lo asigna a la variable ultimoElemento. Después de la operación, el array miArray ya no contiene 50.

✏️ sort()

```
// Array de ejemplo
const numeros = [4, 2, 8, 1, 5];

// Ordenar el array de manera ascendente (de menor a mayor)
const numerosAscendente = numeros.slice().sort((a, b) => a - b);

// Ordenar el array de manera descendente (de mayor a menor)
const numerosDescendente = numeros.slice().sort((a, b) => b - a);

// Mostrar resultados
console.log("Array original:", numeros);
console.log("Array ordenado ascendente:", numerosAscendente);
console.log("Array ordenado descendente:", numerosDescendente);
```

Resultado en la consola:

```
Array original: [4, 2, 8, 1, 5]
Array ordenado ascendente: [1, 2, 4, 5, 8]
Array ordenado descendente: [8, 5, 4, 2, 1]
```

En este ejemplo, usamos el método **sort()** en dos ocasiones. Primero, creamos una copia del array original mediante **slice()** para evitar modificar el array original. Luego, pasamos una función de comparación como argumento al método **sort()**. Esta función de comparación toma dos elementos del array y devuelve un número negativo si el primer elemento debe aparecer antes que el segundo, cero si ambos elementos son iguales en orden y un número positivo si el segundo elemento debe aparecer antes que el primero.

Cuando restamos **b - a**, **ordena de forma descendente** porque si b es mayor que a, el resultado de la resta será positivo, lo que coloca b antes de a. Por otro lado, **si restamos a - b, el orden es ascendente** porque si a es mayor que b, el resultado de la resta será positivo, lo que coloca a antes de b.

✏️ splice()

Supongamos que tenemos un array que contiene los nombres de algunos colores:

```
let colores = ["rojo", "verde", "azul", "amarillo", "naranja"];
```

Ahora, vamos a realizar algunas operaciones utilizando el método splice():

1. **Agregar un nuevo color al inicio del array:**

```
colores.splice(0, 0, "violeta");

console.log(colores);  //  Muestra  ["violeta",  "rojo",  "verde",  "azul",
"amarillo", "naranja"]
```

En este ejemplo, **splice(0, 0, "violeta")** inserta el color "violeta" en la posición 0 (es decir, al inicio) del array colores, sin eliminar ningún elemento.

2. **Reemplazar un color en una posición específica:**

```
colores.splice(2, 1, "blanco");

console.log(colores); // Muestra ["violeta", "rojo", "blanco", "amarillo",
"naranja"]
```

En este caso, splice(2, 1, "blanco") reemplaza un elemento en la posición 2 (que es "azul") con el color "blanco".

3. **Eliminar elementos desde una posición específica:**

```
colores.splice(3, 2);

console.log(colores); // Muestra ["violeta", "rojo", "blanco"]
```

Aquí, splice(3, 2) elimina dos elementos a partir de la posición 3, eliminando "amarillo" y "naranja" del array.

Recuerda que el método **splice()** modifica el array original y puede realizar operaciones de inserción, eliminación o reemplazo, dependiendo de los argumentos que le proporciones. Su sintaxis es **splice(start, deleteCount, item1, item2, ...)**:

- **start**: Índice donde comenzar a realizar las operaciones.
- **deleteCount**: Número de elementos a eliminar a partir de start.
- **item1**, **item2**, ...: Elementos que se agregarán en lugar de los eliminados (opcional).

💡 ***Aclaración:*** *El método* ***splice()*** *no se utiliza para agregar elementos al final de un array. En su lugar, se utiliza para insertar, eliminar o reemplazar elementos en una posición específica dentro del array. Para agregar un elemento al final de un array, puedes utilizar el método* ***push()****.*

Ejemplos iteración

✏️ filter()

La sintaxis básica de **filter()** es la siguiente:

```
const nuevoArreglo = arregloOriginal.filter(funcionCallback(elemento, index, arreglo) {
  // Condición de filtrado
  // Retornar "true" si se desea incluir el elemento en el nuevo arreglo, o "false" si se desea excluirlo.
});
```

Los parámetros de la función de callback son opcionales:

- **elemento**: El valor del elemento actual del arreglo que se está procesando.
- **index** (opcional): El índice del elemento actual en el arreglo.
- **arreglo** (opcional): El arreglo original al que se está aplicando el método filter().

La función de callback debe retornar un valor booleano (true o false) que indique si el elemento actual cumple o no con la condición de filtrado. Si el valor retornado es true, el elemento se incluirá en el nuevo arreglo; si es false, el elemento se excluye.

Veamos un ejemplo práctico:

```
const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Filtrar solo los números pares
const numerosPares = numeros.filter(numero => numero % 2 === 0);

console.log(numerosPares); // Salida: [2, 4, 6, 8, 10]
```

En este ejemplo, utilizamos filter() para obtener un nuevo arreglo llamado numerosPares, que solo contiene los números pares del arreglo original numeros. La función de callback (numero => numero % 2 === 0) verifica si el número es divisible por 2 (es par), y si es así, retorna true, lo que resulta en la inclusión del número en el nuevo arreglo. Si el número es impar, la función retorna false, y el número se excluye del nuevo arreglo.

filter() es una herramienta útil cuando necesitas seleccionar ciertos elementos de un arreglo que cumplan con ciertas condiciones, sin modificar el arreglo original. Recuerda que **filter**() no altera el arreglo original, sino que crea un nuevo arreglo con los elementos que pasan el filtro.

✏️ forEach()

La sintaxis del método forEach() es la siguiente:

```
array.forEach(callback(elemento, indice, array))
```

Donde array es el array sobre el cual se va a iterar, y callback es una función que se ejecutará una vez por cada elemento del array. La función callback puede aceptar hasta tres argumentos:

- elemento: El valor del elemento actual del array en cada iteración.
- indice (opcional): El índice del elemento actual dentro del array.
- array (opcional): El array sobre el cual se está iterando.

Es importante destacar que **forEach()** no devuelve nada, es decir, devuelve undefined. Se utiliza principalmente para efectuar operaciones en cada elemento del array, como realizar cálculos, modificar valores o mostrar información.

Ejemplo:

```
// Array de ejemplo
const numeros = [1, 2, 3, 4, 5];

// Usar forEach() para mostrar cada número en la consola
numeros.forEach((numero, indice) => {
  console.log(`Elemento ${indice}: ${numero}`);
});
```

Resultado en la consola:

```
Elemento 0: 1
Elemento 1: 2
Elemento 2: 3
Elemento 3: 4
Elemento 4: 5
```

Una ventaja de forEach() es que facilita el acceso a cada elemento del array y su índice, lo que puede ser útil en diversas situaciones.

Objeto Math

Definición y usos

El objeto "Math" en JavaScript es una utilidad incorporada que proporciona propiedades y métodos para realizar operaciones matemáticas. No es necesario crear una instancia de este objeto, ya que está disponible globalmente en el lenguaje.

El objeto "Math" incluye una variedad de constantes y funciones matemáticas útiles que puedes utilizar en tus programas JavaScript. Algunas de las operaciones más comunes que puedes realizar con el objeto "Math" son:

- Operaciones matemáticas básicas:

    - Math.abs(x): Devuelve el valor absoluto de x (el valor sin signo).
    - Math.ceil(x): Redondea un número hacia arriba al número entero más cercano.
    - Math.floor(x): Redondea un número hacia abajo al número entero más cercano.
    - Math.round(x): Redondea un número al número entero más cercano.
    - Math.max(x1, x2, ..., xn): Devuelve el valor más grande de una lista de números.
    - Math.min(x1, x2, ..., xn): Devuelve el valor más pequeño de una lista de números.
    - Math.random(): Devuelve un número pseudoaleatorio entre 0 (inclusive) y 1 (exclusivo).

Los superiores son los métodos más utilizados para

- Funciones trigonométricas:
    - Math.sin(x): Devuelve el seno de x (en radianes).
    - Math.cos(x): Devuelve el coseno de x (en radianes).
    - Math.tan(x): Devuelve la tangente de x (en radianes).
    - Math.atan(x): Devuelve el arco tangente de x (en radianes).
- Funciones exponenciales y logarítmicas:
    - Math.exp(x): Devuelve el valor de e (la base del logaritmo natural) elevado a la potencia x.
    - Math.log(x): Devuelve el logaritmo natural de x.
    - Math.pow(base, exponente): Devuelve la base elevada al exponente especificado.
- Constantes:
    - Math.PI: El valor de π (pi), aproximadamente 3.141592653589793.
    - Math.E: El número de Euler, aproximadamente 2.718281828459045.

Estos son solo algunos ejemplos de las operaciones que puedes realizar utilizando el objeto "Math" en JavaScript. Es una herramienta muy útil para realizar cálculos matemáticos en tus aplicaciones web o programas.

Ejemplos

✏️ métodos Math.xxx()

Te dejamos los siguientes ejemplos para que puedas ver cómo se utiliza el objeto Math.

```
// Calcular la raíz cuadrada
numero = Math.sqrt(25) //5

//Calcular la raíz cúbica
numero = Math.cbrt(27) //3

// Devolver el número más grande
numero = Math.max(25,3,67,89,70) //89

// Devolver el número más chico
numero = Math.min(25,3,67,89,70) //3

// Numero aletorio entre 0 y 1 no incluye el cero ni el 1
numero = Math.random() //0.68345635345

// Numero aletorio entre 0 y 100 redondeado con metodo round
// redondea al enteero más cerecano
numero = Math.round(Math.random()*100) //33

// Redondeado para abajo
numero = Math.floor(4.99) //4

// Elimina los decimales
numero = Math.trunc(4.99) //4
```

Este viaje no termina aquí, te esperamos en la siguiente parte teórica 🚀