Conceptos básicos de lógica

Conceptos básicos de lógica

En tu proceso de aprendizaje, ten muy presente los siguientes conceptos para aplicarlos en los ejercicios de lógica de programación o desarrollo.

CONCEPTO	EJEMPLO

Arreglo de dimensión 1 o unidimensional: Son aquellos cuyos elementos no son arreglos, se usa un solo índice para acceder a sus elementos	`let arregloUnidimensional = [2, 5, 7, true, false, "ww"];`
	`console.log(arregloUnidimensional[0]);`
	`//imprime: 2`

Arreglo de dimensión 2 o bidimensional: Es aquel cuyos elementos son arreglos. Es un arreglo que contiene arreglos. Se usan dos índices para acceder a sus elementos más internos.	`let arregloBidimensional = [[1, 3, 60], ["w", "e", "q"], [true, false, true]];`
	`console.log(arregloBidimensional[0][1]);//imprime: 3`
	`console.log(arregloBidimensional[1][2]);//imprime: q`

Arreglo de dimensión superior(dimensión 3 o más): Para acceder a los elementos de dimensión superior, se usan tantos índices como tenga la dimensión, por ejemplo, si el arreglo es de dimensión 3 se usan tres índices, si es de dimensión 4 se usan 4 índices.	`let arregloDimension3 = [ [[1, 5, 8], [true]], [[true, false, true], [true]] ];`
	`console.log(arregloDimension3[0][1][0]);//imprime: true`
	`console.log(arregloDimension3[1][0][1]);//imprime: false`

Escritura y eliminación de elementos en arreglos multidimensionales: Para escribir o eliminar elementos en un arreglo multidimensional, primero se accede al elemento que se quiere escribir o eliminar(tal cual se explica en la primera parte de esta guía), luego usamos el operador de asignación para escribir o usamos splice para eliminar el elemento.	`let arregloDimension3 = [ [[1, 5, 8], [true]], [[true, false, true], [true]] ];`
	`//ESCRITURA`
	`arregloDimension3[1][0][2] = false;`
	`console.log(arregloDimension3[1][0][2])//imprime: false`
	`//ELIMINACIÓN`
	`arregloDimension3[0][0].splice(1, 1);`
	`console.log(arregloDimension3);//imprime: [ [ [ 1, 8 ], [ true ] ], [ [ true, false, false //], [ true ] ] ]`

Recorrido de un arreglo unidimensional con for: El bucle for tradicional permite iterar sobre los elementos del arreglo. Aquí, i es el índice que cambia en cada iteración, permitiendo acceder a cada elemento del arreglo con array[i].	`const array = [1, 2, 3, 4, 5];`
	
	`for (let i = 0; i < array.length; i++) {`
	`  console.log(array[i]);`
	`}`

Recorrido de un arreglo unidimensional con for each: El método forEach ejecuta una función para cada elemento del arreglo. La función recibe como parámetros el elemento actual, el índice y el arreglo original (aunque el índice y el arreglo son opcionales).	`const array = [1, 2, 3, 4, 5];`
	
	`array.forEach((element, index) => {`
	`  console.log(element);`
	`  // Si necesitas el índice o el arreglo completo, puedes usarlos así:`
	`  // console.log(`Índice: ${index}, Elemento: ${element}`);`
	`});`

Método para obtener el tamaño de un arreglo: para obtener el tamaño (o la cantidad de elementos) de un arreglo, se usa la propiedad .length. Esta propiedad devuelve un número que representa la cantidad de elementos en el arreglo.	`const numeros = [1, 2, 3, 4, 5];`
	`const tamaño = numeros.length;`
	`console.log(tamaño); // Imprime: 5`

Métodos de arreglos en javascript

A Continuación se mostraran algunos de los métodos que contienen los arreglos en javascript los cuales serán de gran utilidad al momento de usarlos.

- **push()**

Agrega uno o más elementos al final del arreglo.

```
let array = [1, 2, 3];
array.push(4); // array ahora es [1, 2, 3, 4]
```

- **pop()**

Elimina el último elemento del arreglo y lo retorna.

```
let array = [1, 2, 3, 4];
let lastElement = array.pop(); // lastElement es 4 y array ahora es [1, 2, 3]
```

- **shift()**

Elimina el primer elemento de un arreglo y lo retorna.

```
let array = [1, 2, 3];
let firstElement = array.shift(); // firstElement es 1 y array ahora es [2,
   3]
```

- **unshift()**

Agrega uno o más elementos al principio del arreglo.

```
let array = [2, 3];
array.unshift(1); // array ahora es [1, 2, 3]
```

- **splice()**

Cambio el contenido del arreglo eliminandolo, reemplazando o agregando nuevos elementos.

```
let array = [1, 2, 3, 4];
array.splice(1, 2, 'a', 'b'); // array ahora es [1, 'a', 'b', 4]
```

- **slice()**

Retorna una copia de una parte del arreglo dentro de un nuevo arreglo.

```
let array = [1, 2, 3, 4];
let newArray = array.slice(1, 3); // newArray es [2, 3]
```

- **forEach()**

Ejecuta una función sobre cada elemento del arreglo.

```
let array = [1, 2, 3];
array.forEach(element => console.log(element * 2)); // Imprime 2, 4, 6
```

- **map()**

Crea un nuevo arreglo con los resultados de la llamada a una función aplicada a cada uno de sus elementos.

```
let array = [1, 2, 3];
let newArray = array.map(element => element * 2); // newArray es [2, 4, 6]
```

- **filter()**

Crea un nuevo arreglo con todos los elementos que pasen la prueba implementada por la función dada.

```
let array = [1, 2, 3, 4];
let newArray = array.filter(element => element > 2); // newArray es [3, 4]
```

- **reduce()**

Aplica una función a un acumulador a cada elemento del arreglo (De izquierda a derecha) para reducirlo a un único valor.

```
let array = [1, 2, 3, 4];
let sum = array.reduce((accumulator, currentValue) => accumulator +
  currentValue, 0); //sum es 10
```

- **find()**

Retorna el primer elemento del arreglo que cumpla con la condición dada en la función.

```
let array = [1, 2, 3, 4];
let found = array.find(element => element > 2); // found es 3
```

- **findIndex()**

Retorna el índice del primer elemento que cumpla con la condición dada por la función.

```
let array = [1, 2, 3, 4];
let index = array.findIndex(element => element > 2); // index es 2
```

- **includes()**

Determina si un arreglo incluye un determinado elemento.

```
let array = [1, 2, 3];
let hasTwo = array.includes(2); // hasTwo es true
```

- **join()**

Une todos los elementos de un arreglo en una cadena y la retorna.

```
let array = [1, 2, 3];
let str = array.join('-'); // str es '1-2-3'
```

- **sort()**

Ordena los elementos del arreglo y retorna el arreglo ordenado.

```
let array = [3, 1, 4, 1, 2];
array.sort(); // array ahora es [1, 1, 2, 3, 4]
```

- **reverse()**

Invierte el orden de los elementos del arreglo.

```
let array = [1, 2, 3];
array.reverse(); // array ahora es [3, 2, 1]
```

Formas de recorrer un arreglo

A Continuación se mostraran algunas formas de recorrer arreglos, ya sea unidimensionales, bidimensionales o multidimensionales.

Usando ciclo for

Para recorrer un arreglo multidimensional, por ejemplo uno bidimensional con ciclo for, se usan dos ciclos anidados y dos índices, i y j, de tal manera que el ciclo for externo recorra los elementos más externos, los cuales son arreglos, y el ciclo for interno recorra los elementos más internos, los cuales no son arreglos.

```
let arregloBidimensional = [[20, 80, 30], [10, 20, 50],[70, 60, 40]];
//El siguiente ciclo recorre los elementos que conforman el arreglo
//arregloBidimensional es decir, [20, 80, 30], [10, 20, 50],[70, 60, 40]
for (let i = 0; i < arregloBidimensional.length; i++) {
 //El siguiente ciclo recorre los elementos de cada elemento del arreglo
 //arregloBidimensional, es decir 20, 80, 30, 10, 20, 50 , 70, 60, 40
 for (let j = 0; j < arregloBidimensional[i].length; j++) {
   //Acá se imprime cada elemento más interno del arreglo
   console.log(arregloBidimensional[i][j]);
 } //La salida seria: 20, 80, 30, 10, 20, 50, 70, 60, 40
}
```

El código anterior es una fórmula que siempre va a funcionar para recorrer un arreglo bidimensional con ciclo for. Se puede usar simplemente reemplazando en él el nombre del arreglo bidimensional a recorrer.

Usando ciclo forEach

Para recorrer un arreglo multidimensional, por ejemplo uno bidimensional con ciclo forEach, se usan dos ciclos anidados, de tal manera que el ciclo forEach externo recorra los elementos más externos, los cuales son arreglos, y el ciclo forEach interno recorra los elementos más internos, los cuales no son arreglos.

```
let arregloBidimensional = [[20, 80, 30], [10, 20, 50],[70, 60, 40]];
//El siguiente ciclo recorre los elementos que conforman el arreglo
//arregloBidimensional es decir, [20, 80, 30], [10, 20, 50],[70, 60, 40]
arregloBidimensional.forEach(i => {
 //El siguiente ciclo recorre los elementos de cada elemento del arreglo
 //arregloBidimensional, es decir 20, 80, 30, 10, 20, 50 , 70, 60, 40
 i.forEach(j => {
   //Acá se imprime cada elemento más interno del arreglo
   console.log(j);
 })// salida: 20, 80, 30, 10, 20, 50, 70, 60, 40
})
```

El código anterior es una fórmula que siempre va a funcionar para recorrer un arreglo bidimensional con ciclo forEach. Se puede usar simplemente reemplazando en él el nombre del arreglo bidimensional a recorrer.

A tener en cuenta:
En general, para recorrer un arreglo multidimensional, primero nos fijamos en la dimensión del arreglo, si el arreglo es de dimensión 2 y lo queremos recorrer con un ciclo for, entonces se usan DOS ciclos for anidados y DOS índices, si lo queremos recorrer con forEach, entonces se usan DOS ciclos forEach anidados. Si el arreglo que queremos recorrer es de dimensión 3 y lo queremos recorrer con un ciclo for, entonces se usan TRES ciclos for anidados y TRES índices, si lo queremos recorrer con forEach, entonces se usan TRES ciclos forEach anidados…