Material de lectura

Búsquedas en JavaScript

Existen diferentes métodos que se pueden usar en JavaScript para buscar elementos dentro de un arreglo. El método a elegir depende del caso de uso particular, por ejemplo:

1. Obtener todos los elementos del arreglo que cumplen una condición específica. (**Filter**)
2. Obtener al menos uno de los elementos del arreglo que cumple dicha condición. (**Find**)
3. Obtener si un valor específico es parte del arreglo (**Includes**)
4. Obtener el índice de un valor específico (**IndexOf**).

Array.filter()

Podemos usar el método Array.filter() para encontrar los elementos dentro de un arreglo que cumplan con cierta condición. Por ejemplo, si queremos obtener todos los elementos de un arreglo de números que sean mayores a 10, podemos hacer lo siguiente:

```
let arreglo = [10, 11, 3, 20, 5];

let mayorQueDiez = arreglo.filter(element => element > 10);

console.log(mayorQueDiez) // resultado esperado: [11, 20]
```

Array.find()

Usamos el método Array.find() para encontrar el primer elemento que cumple cierta condición. Tal como el método anterior, toma un Callback como argumento y devuelve el primer elemento que cumpla la condición establecida. Usemos el método find en el arreglo del ejemplo anterior.

```
let arreglo = [10, 11, 3, 20, 5];

let existeElementoMayorQueDiez = arreglo.find(element => element > 10);

console.log(existeElementoMayorQueDiez) // resultado esperado: 11
```

Array.includes()

El método includes() determina si un arreglo incluye un valor específico y devuelve verdadero o falso según corresponda. En el ejemplo anterior, si queremos revisar si 20 es uno de los elementos del arreglo, podemos hacer lo siguiente:

```
let arreglo = [10, 11, 3, 20, 5];
```

```
let incluyeVeinte = arreglo.includes(20);

console.log(incluyeVeinte) // resultado esperado: true
```

Array.indexOf()

El método indexOf() devuelve el primer índice encontrado de un elemento específico. Devuelve -1 si el elemento no existe en el arreglo. Volvamos a nuestro ejemplo y encontremos el índice de 3 en el arreglo.

```
let arreglo = [10, 11, 3, 20, 5];

let indiceDeTres = arreglo.indexOf(3);

console.log(indiceDeTres) // resultado esperado: 2
```

Búsqueda de Máximos y Mínimos

Uno de los problemas académicos más comunes es el de la búsqueda del valor máximo o mínimo dentro de una lista. JavaScript dispone de las funciones Math.max() y Math.min() con las que es posible obtener el máximo y mínimo respectivamente de un conjunto de números, por ejemplo:

```
Math.max(1, 2, 3, 4, 5); // resultado esperado: 5

Math.min(1, 2, 3, 4, 5); // resultado esperado: 1
```

El problema de estas funciones es que no permiten entradas de tipo array, solamente de tipo numérico. Normalmente se puede solucionar empleando diferentes aproximaciones como son los métodos reduce() o apply(). La forma más fácil de aplicar una función a un array es utilizando el método apply(). Simplemente se tiene que aplicar apply() a la función pasando como primer parámetro null y como segundo parámetro el array. Así se puede obtener el máximo o mínimo de un array simplemente con el siguiente código.

```
Math.max.apply(null, values) // resultado esperado: 5

Math.min.apply(null, values) // resultado esperado: 1
```

Búsqueda Secuencial

La búsqueda secuencial se define como la búsqueda en la que se compara elemento por elemento del vector/array con el valor que buscamos. Es decir, un clásico recorrido secuencial (for). De hecho, tenemos varias maneras de implementarlo, pero más allá de eso, el principio es el mismo. Comparar elemento por elemento hasta encontrar el/los que buscamos. Este tipo de búsqueda en el peor de sus casos ejecuta las instrucciones del loop n veces, es decir es la cantidad de elementos del arreglo X👎 . En el mejor de los casos, el primer elemento del arreglo es el elemento que estamos buscando, por eso para ese caso ese elemento pasaría a ser X(1).

Veamos un ejemplo de una función que implementa búsqueda secuencial en un arreglo:

```
// Devolverá el índice donde encontró al elemento. Recibe el valor a buscar y el arreglo donde buscará

function sequentialSearch(element, array){

 for (var i in array){

   if (array[i] == element) return i;

 }

 return -1;

}

 var letters = ["a", "b", "c", "d", "f", "g", "h", "i", "j", "k", "l", "m", "n"];

sequentialSearch("g",letters);
```

Ordenamiento

Javascript provee un método que ordena los elementos de un arreglo localmente y devuelve el arreglo ordenado Array.prototype.sort([compareFunction(a, b)]). El modo de ordenación por defecto responde a la posición del valor del string de acuerdo a su valor en el juego de caracteres Unicode. Cuando se utiliza el método sort(), los elementos se ordenarán en orden ascendente (de la A a la Z) por defecto:

```
const equipos = ['Real Madrid', 'Manchester Utd', 'Bayern Munich', 'Juventus'];

equipos.sort();

// ['Bayern Munich', 'Juventus', 'Manchester Utd', 'Real Madrid']
```

Dada esta característica el ordenar números pasa a ser una tarea no tan simple. Si aplicamos el método sort() directamente a un arreglo de números, veremos un resultado inesperado:

```
const numeros = [3, 23, 12];

numeros.sort(); // --> 12, 23, 3
```

El método sort() puede ordenar valores negativos, cero y positivos en el orden correcto. Cuando compara dos valores, se puede enviar la función compareFunction(a, b), como función de comparación y luego se ordenan los valores de acuerdo al resultado devuelto:

1. Si el resultado es negativo, a se ordena antes que b.
2. Si el resultado es positivo, b se ordena antes de a.
3. Si el resultado es 0, nada cambia.

Por ende si queremos ordenar los números en orden ascendente, esta vez necesitamos restar el primero parámetro (a) del segundo (b):

```
var numbers = [4, 2, 5, 1, 3];

numbers.sort(function(a, b) {

  return a - b;

}); // --> 1, 2, 3, 4, 5
```

Actividad previa

◄ Funciones y estructuras en JavaScript

Siguiente actividad

¿DOM que es? ►