PROGRAMACIÓN BÁSICA EN JAVASCRIPT

Módulo 2

OTEC PLATAFORMA 5 CHILE

MÓDULO 2

CLASE 5

Introducción

Hoy nos encontraremos con una de las partes más utilizadas de JavaScript, los Arreglos o en inglés Array, que los utilizamos para almacenar colecciones ordenadas.

Lo bueno de los Arreglos es que nos proveen de métodos para manejar el orden de los elementos y así modificar nuestra colección ordenada, según lo necesitemos.

De materia, hay harta cantidad, así que ya arrancamos con nuestro tema del día.

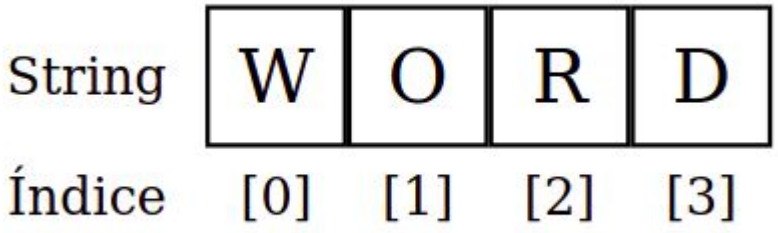
String como colección de caracteres

Los Strings son cadenas de caracteres guardadas en direcciones de memorias continuas. Podemos acceder a cada letra usando corchetes ([]) y un Índice (un número que indica la posición del carácter).

Con la propiedad length nos permite tomar dimensión de la longitud de un String. Es decir, nos ayuda a contar la cantidad de caracteres que hay en esa cadena. Si queremos acceder rápidamente a la propiedad length de un String, debemos escribir directamente la cadena de caracteres (o la Variable que la guarda) seguida por *.length*.

```
> let palabra="word";

  palabra.length
<· 4
```



EL ÍNDICE

El Índice (en inglés, Index) nos revelará cuál es la posición que ocupa cada carácter de un String o de un Arreglo.
El primer Índice de un String es cero (0).

Arreglos

Un Arreglo es un conjunto de datos que se encuentran ordenados. Como vimos previamente, en JavaScript, los datos pueden ser de cualquier tipo (Números, Strings, Funciones, etc.).

Gracias a los Arreglos, podemos reunir en un solo lugar los distintos tipos de datos, sin necesidad de crear una Variable para cada uno.

La sintaxis de un arreglo se declara primero usando la keyword let, luego el nombre de este Array, luego escribimos el signo de declaración = y seguido colocamos corchetes [].

Allí guardaremos todos los datos que necesitemos separados por comas , .

```
//Variables para guardar productos.
let producto1="Televisor";
let producto2="DVD";
let producto3="Computadora";
let producto4="Celular";
let producto5="Aire Acondicionado";

//Variable para guardar grupo de datos.
let productos= ["Televisor","DVD","Computadora", "Celular", "Aire Acondicionado"];
```

Índice de un Arreglo

Los Arreglos nos permiten agrupar datos en una lista ordenada.

Por lo tanto, cada elemento ocupa una posición indexada numéricamente.

¿Para qué se usa la propiedad length en un array?

Podemos usar la propiedad .length para saber cuántos elementos tiene un Arreglo.

Cada vez que agregamos elementos a la lista, el valor de .length se modifica y, de esta manera, es fácil acceder al número total.

```
> let productos= ["Televisor","DVD","Computadora",
  "Celular", "Aire Acondicionado"];
<· undefined
> productos.length
<· 5
```

IMPORTANTE

Recuerde que el primer Índice de un arreglo siempre es 0 (cero). Por lo tanto, el Índice más alto siempre será uno menos que la cantidad total de elementos (length).

Método Push y Pop

Estos métodos trabajan en el final de un arreglo.

Método PUSH

.push() añade un nuevo elemento al final del Arreglo.

Para agregar el valor, debe pasarlo como argumento del método. Para agregar varios elementos dentro del mismo método, debe separar los valores con comas (,).

Método POP

.pop() saca el último elemento del Arreglo y lo retorna.

Ese valor, a su vez, lo podemos guardar para volverlo a usar.

El método pop no lleva Argumentos y solamente saca el último elemento, uno por vez.

Método Unshift y Shift

Estos métodos trabajan en el principio de un arreglo.

Método UNSHIFT

.unshift() añade un nuevo elemento al principio del Arreglo. Para agregar el valor, debe pasarlo como argumento del método.

Para agregar varios elementos dentro del mismo método, debe separar los valores con comas (,).

Método SHIFT

.shift() saca el primer elemento del Arreglo y lo retorna. Ese valor, a su vez, lo podemos guardar para volverlo a usar.

El método shift no lleva Argumentos y solamente saca el último elemento, uno por vez.

```
> let frutas=["uva", "durazno", "mandarina"];
  frutas.unshift("banana", "ananá");
< 5
> frutas
< ▼ (5) ['banana', 'ananá', 'uva', 'durazno', 'mandarina']
    0: "banana"
    1: "ananá"
    2: "uva"
    3: "durazno"
    4: "mandarina"
    length: 5
  ▶ [[Prototype]]: Array(0)
```

```
> frutas
< ▶ (5) ['banana', 'ananá', 'uva', 'durazno', 'mandarina']
> frutas.shift()
< 'banana'
> frutas
< ▶ (4) ['ananá', 'uva', 'durazno', 'mandarina']
> let guardarAnana=frutas.shift();
< undefined
> frutas
< ▶ (3) ['uva', 'durazno', 'mandarina']
> guardarAnana
< 'ananá'
```

Método INDEXOF

.indexOf() verifica la posición de un elemento dentro de un arreglo. En caso de existir nos devuelve su índice. Pero si no encontrara ninguna coincidencia nos retornará -1.

```
> let mascotas=["perro", "gato", "loro", "pez"];
<· undefined
> mascotas.indexOf("loro");
<· 2
> mascotas.indexOf("hamster");
<· -1
```

Método Slice

.slice() se usa para generar una copia de un Arreglo. Esto sirve para trabajar sobre el clon del Arreglo sin afectar su original. Este método trabaja con 3 tipos de formas.

.slice() sin argumento. Esto se declara en una variable para trabajar con el nuevo arreglo.

```
> let almacen=["fideos", "arroz", "aceite"];
< undefined
> let almacenClon=almacen.slice();
< undefined
> almacen
< ▶ (3) ['fideos', 'arroz', 'aceite']
> almacenClon
< ▶ (3) ['fideos', 'arroz', 'aceite']
```

.slice(argumento) .Esto se declara en una variable para trabajar con el nuevo arreglo copiandolo desde el índice pasado como argumento.

```
> let almacenClon=almacen.slice(1);
< undefined
> almacenClon
< ▶ (2) ['arroz', 'aceite']
```

.slice(argumento1, argumento2) .Esto se declara en una variable para trabajar con el nuevo arreglo copiandolo desde el índice pasado como argumento1 hasta el índice del argumento2 sin incluirlo.

```
> let almacenClon=almacen.slice(0,2);
< undefined
> almacenClon
< ▶ (2) ['fideos', 'arroz']
```

Método Splice

.splice() elimina de un Arreglo una cantidad de elementos a partir de una posición dada. Este método es útil ya que podemos eliminar más de un elemento, a diferencia del .shift() y el .pop().

Importe: El método .splice() devuelve los elementos eliminados en un nuevo Arreglo.

```
> let puntajes=[1,5,7,9,10,8];
<· undefined
> puntajes
<· ▶ (6) [1, 5, 7, 9, 10, 8]
> let puntajesEliminados=puntajes.splice(2,3)
<· undefined
> puntajes
<· ▶ (3) [1, 5, 8]
> puntajesEliminados
<· ▶ (3) [7, 9, 10]
```

Método Join

Este método manipula transformando el array.

Método JOIN

.join() convierte un arreglo en una cadena de caracteres.

Importante: Cuando hacemos join() no modificamos el Arreglo original sino que tomamos sus elementos y los manipulamos.

```
> let array=["Hola", "¿cómo", "estas?"];
< undefined
> array
< ▶ (3) ['Hola', '¿cómo', 'estas?']
> array.join()
< 'Hola,¿cómo,estas?'
> //En la declaración de arriba, me junta los elementos y como
  no tiene ningún argumento, me separa los elementos con comas ,

> array
< ▶ (3) ['Hola', '¿cómo', 'estas?']
> array.join(" ");
< 'Hola ¿cómo estas?'
> //En la declaración de arriba, me junta los elementos y me
  los separa con un espacio ya que le pasé como argumento un
  espacio.

> array
< ▶ (3) ['Hola', '¿cómo', 'estas?']
> array.join("15");
< 'Hola15¿cómo15estas?'
> //En la declaración de arriba, me junta los elementos y me lo
  los separa con el 15, ya que fue el argumento pasado.
```

Método Split

Este método modifica el array.

Método SPLIT

.split() separa un String y lo convierte en un Arreglo con sus distintas posiciones.

Importante: Aquello que le pasemos como Argumento le indicará al método dónde debe hacer el corte.

```
> let saludo= "Hola Mario";
<· undefined
> let array= saludo.split(" ");
<· undefined
> saludo
<· 'Hola Mario'
> array
<· ▶ (2) ['Hola', 'Mario']
```

```
> let array= saludo.split("M");
<· undefined
> saludo
<· 'Hola Mario'
> array
<· ▶ (2) ['Hola ', 'ario']
```

Hasta aquí hemos visto todos los métodos con los cuales podemos trabajar con los Arreglos...
Pero nos queda algo super importante:
LA ITERACIÓN.
En palabras más comunes, es la forma de recorrer un arreglo, accediendo a todos sus elementos.
Es decir, determinar mediante Índices las diferentes posiciones. Para hacerlo, usamos una Estructura de Control Repetitiva (un bucle o, en inglés, loop) para recorrer el Arreglo y acceder a cada valor.

Iteración de arreglos con *For Loops*

Una forma de iterar un arreglo es a través un For Loops ya que es muy útil para mostrar todos sus elementos. Veamos cuales son los elementos de la sintaxis;

- Inicialización: Es la Variable que permite iterar el Arreglo. Esta expresión se ejecuta una sola vez.
- Condición: Es una expresión que se evalúa en cada iteración.
- Incremento/Decremento: Es una expresión que actualiza el valor de la Variable después de cada iteración y asegura la condición de corte. Podemos especificar de qué manera queremos que se incremente o decremente el valor (si de uno en uno, de dos en dos, etc.)

Si bien el For Loop no es muy distinto al While Loop, su sintaxis es más simple para manipular arreglos.

```
//Declaramos un arreglo y le colocamos sus elementos.
let arreglo = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

//A través del For Loops lo recorremos. En la condición del for tenemos 3 secciones. En la primera la inicialización, en la segunda la condición y por último la modificación de la inicialización.
for (i = 0; i < arreglo.length; i++) {
 //Aquí declaramos la sentencia que necesita ejecutarse por cada vez que se recorra el arreglo.
 console.log("Este es un elemento del arreglo: " + arreglo[i]);
};
```

¿CUÁNDO USAMOS UN WHILE LOOP Y UN FOR LOOP?
Las buenas prácticas sugieren que uses el While Loop cuando no sepas cuántas iteraciones sucederán hasta que la condición sea false. En cambio, se utiliza el For Loop, sobre todo, con Arreglos en los que se sabe de antemano cuántos elementos contiene. De esta manera, la propiedad .length asegurará la condición de corte.

Iteración de arreglos con *For Each*

Otra forma de iterar un arreglo es utilizar el método For Each que permite ejecutar una Función sobre cada elemento del Arreglo.

El beneficio de este Iterador sobre el resto es que no hace falta ni inicializar una Variable, ni plantear una condición, ni asegurar una acción de corte.

.forEach(), por lo tanto, debería usarse solamente cuando queremos recorrer el Arreglo entero y ejecutar la misma Función para todos los elementos. Su sintaxis está compuesta por las siguientes partes.

nombreArreglo: Es el Arreglo que queremos recorrer.

- .forEach(): Es el iterador que usaremos.
- function(): Es la Función que se ejecutará sobre cada elemento. Para hacerlo, deberás pasarle un Parámetro (dentro del paréntesis).
- elemento: Es el Parámetro que hace referencia a cada elemento del Arreglo (irá cambiando en cada iteración hasta haberlos recorrido completamente).
- acción: Es la Función que queremos que se ejecute sobre cada elemento del Arreglo.

```
//Declaramos un arreglo y le colocamos sus elementos.
let arreglo = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

//Realizamos un recorrido del arreglo que por cada
iteración de los elementos, ejecute una función.
arreglo.forEach(function (numero) {
 console.log(numero);
});
```

MÓDULO 2

GRACIAS

Aquí finaliza la clase n°5 del módulo 2

OTEC PLATAFORMA 5 CHILE