



FULL STACK

**Comenzamos en unos
minutos**

ACADEMY
by NUMEN

Javascript: Clase 4

Arrays (arreglos)

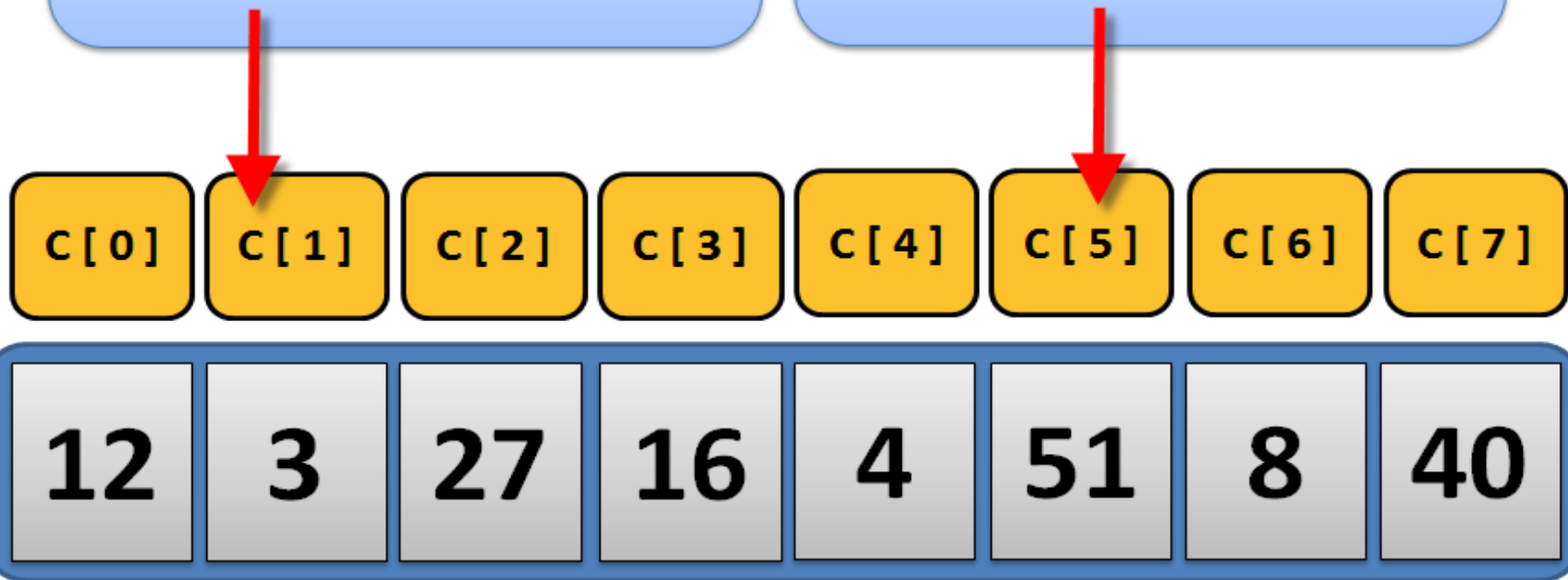
¿Qué es un arreglo?

Características

- Cada elemento en el arreglo contiene un valor
- Los elementos de un arreglo se indexan a partir de 0
- La longitud del arreglo es el número total de elementos que contiene
- Los arreglos pueden ser de una dimensión o de múltiples dimensiones
- Se pueden tener arreglos de arreglos (Jagged Arrays)

Nombre del Arreglo (Obsérvese que todos los elementos de este arreglo tienen el mismo nombre, C)

Índice (o subíndice) del elemento en el arreglo C





Un momento! Espera

Ahora lo veremos en código



Tipos de datos compuestos: Arreglo

En clases anteriores discutimos los 3 tipos de datos básicos (cadenas/strings, números y booleanos) y cómo asignar esos tipos de datos a las variables. Discutimos cómo una variable solo puede apuntar a una sola cadena, número o booleano. Por ejemplo, si quisiéramos guardar los nombres de varios estudiantes, teniendo en cuenta lo visto hasta este punto, deberíamos hacer algo así:

```
let estudiante1 = 'Martin'  
let estudiante2 = 'Fernando'  
let estudiante3 = 'Sara'  
let estudiante4 = 'Samuel'  
let estudiante5 = 'Jorge'  
let estudiante6 = 'Pedro'
```

Sin embargo, en muchos casos queremos poder apuntar a una colección de tipos de datos. Por ejemplo, ¿qué pasaría si quisiéramos hacer un seguimiento del nombre de cada estudiante en esta clase usando una sola variable, `nombresEstudiantes`. Podemos hacer eso usando Arrays. Podemos pensar en los **arrays** como contenedores de almacenamiento para colecciones de datos. Construir un **array o matriz** es simple. Debemos declarar una variable y establecerla en []. Luego podemos agregar al contenedor (separadas por coma) tantas cadenas, números o booleanos como queramos y acceder a esos elementos cuando lo deseemos.

```
const nombresEstudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel',  
    'Jorge', 'Pedro']  
  
console.log(nombresEstudiantes) // ['Martin', 'Fernando', 'Sara', 'Samuel',  
    'Jorge', 'Pedro']
```

Al igual que el tipo de dato String tiene un método incorporado `.length`, también lo hace la matriz/array. De hecho, la matriz/array tiene muchos métodos incorporados útiles (los discutiremos más adelante). Al igual que la cadena `.length` cuenta los caracteres, la matriz `.length` devolverá el número de elementos en una matriz:

```
const nombresEstudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel', 'Jorge', 'Pedro']

console.log(nombresEstudiantes) // ['Martin', 'Fernando', 'Sara', 'Samuel', 'Jorge', 'Pedro']
console.log(nombresEstudiantes.length) // 6
```

Veamos otro ejemplo de arrays, pero esta vez con números en lugar de cadenas/strings:

```
const miPrimerArreglo = [ 0, 1, 2, 3, 4, 5]

console.log(miPrimerArreglo) // [ 0, 1, 2, 3, 4, 5]

console.log(miPrimerArreglo.length) // 6
```

Acceso a elementos en una matriz/array

Para acceder al elemento, escribiremos el nombre o la variable de matriz, seguidos de corchetes que contienen la asignación numérica.

Los elementos reciben una posición numérica (índice) de acuerdo con su ubicación en la matriz, en orden. El orden numérico de una matriz SIEMPRE comienza en 0, por lo que el primer elemento está en el índice 0, el segundo en el índice 1, el tercero en el 2, y así sucesivamente

```
const estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel', 'Jorge',  
                    'Pedro']  
  
                // Posicion:    0           1           2           3           4           5  
  
console.log(estudiantes[0]) // Martin  
console.log(estudiantes[1]) // Fernando  
console.log(estudiantes[2]) // Sara  
console.log(estudiantes[3]) // Samuel  
console.log(estudiantes[4]) // Jorge  
console.log(estudiantes[5]) // Pedro
```

Asignación de valores

Podemos asignar y reasignar cualquier índice en la matriz usando el paréntesis/índice y un "=".

```
const estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel', 'Jorge', 'Pedro']  
                // Posición:  0           1           2           3           4           5  
  
estudiantes[0] = 'Juan';  
  
console.log(estudiantes) // ['Juan', 'Fernando', 'Sara', 'Samuel', 'Jorge', 'Pedro']
```


Métodos push() y pop()

Otros dos métodos de matriz incorporados muy útiles son `.push` y `.pop`. Estos métodos se refieren a la adición y eliminación de elementos de la matriz **después** de su declaración inicial.

```
const estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel'];

estudiantes.push('Patricia');

console.log(estudiantes); // ['Martin', 'Fernando', 'Sara', 'Samuel',
                             'Patricia'];

estudiantes.pop();

console.log(estudiantes); // ['Martin', 'Fernando', 'Sara', 'Samuel']
```

Métodos `shift()` y `unshift()`

`.unshift` y `.shift` son exactamente como `.push` y `.pop`, excepto que operan en el primer elemento de la matriz. `.unshift(item)` colocará un nuevo elemento en la primera posición de la matriz, y `.shift()` eliminará el primer elemento de la matriz.

```
const estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel'];

estudiantes.unshift('Leo');

console.log(estudiantes); // ['Leo', 'Martin', 'Fernando', 'Sara', 'Samuel'];

estudiantes.shift();

console.log(estudiantes); // // ['Martin', 'Fernando', 'Sara', 'Samuel'];
```

Recorriendo arrays con un bucle for



ESTO YA SE HA VISTO

Nunca son

Suficientes bucles

```
function imprimirPosiciones(arreglo) {  
    for (let i = 0; i < arreglo.length; i++) {  
        console.log(arreglo[i])  
    }  
}
```

```
imprimirPosiciones([0,1,2,3,4,5,6,7,8,9,10])
```



Programemos una tabla de multiplicar

```
const tablas = [1,2,3,4,5,6,7,8,9],  
  tablaDelUno = [],  
  tablaDelDos = [],  
  tablaDelTres = [],  
  tablaDelCuatro = [],  
  tablaDelCinco = [],  
  tablaDelSeis = [],  
  tablaDelSiete = [],  
  tablaDelOcho = [],  
  tablaDelNueve = []
```



```
function tablasDeMultiplicar(tablas) {  
  for (let i = 0; i < tablas.length; i++) {  
    tablaDelUno[i] = 1 * tablas[i]  
    tablaDelDos[i] = 2 * tablas[i]  
    tablaDelTres[i] = 3 * tablas[i]  
    tablaDelCuatro[i] = 4 * tablas[i]  
    tablaDelCinco[i] = 5 * tablas[i]  
    tablaDelSeis[i] = 6 * tablas[i]  
    tablaDelSiete[i] = 7 * tablas[i]  
    tablaDelOcho[i] = 8 * tablas[i]  
    tablaDelNueve[i] = 9 * tablas[i]  
  }  
}
```



```
return [  
    tablaDelUno,  
    tablaDelDos,  
    tablaDelTres,  
    tablaDelCuatro,  
    tablaDelCinco,  
    tablaDelSeis,  
    tablaDelSiete,  
    tablaDelOcho,  
    tablaDelNueve  
]  
}  
  
console.log(tablasDeMultiplicar(tablas))
```



Objetos

Tipos de datos compuestos: Objetos

Ya aprendimos sobre **arrays** o matrices. Las matrices son contenedores que sostienen colecciones de datos. En esta lección, introduciremos otro contenedor de datos, el **Objeto**. Los objetos y las matrices son similares en ciertas cosas, y muy diferentes en otras. Mientras que los array pueden contener múltiples elementos relacionados unos con otros, los objetos contienen mucha información sobre una sola cosa. Los objetos se instancian usando llaves (`{ }`).

```
const miPrimerObjeto = {}  
  
console.log(miPrimerObjeto) // {}
```



A diferencia de las matrices que tienen elementos valorados en índices, los objetos usan un concepto llamado pares de **clave : valor**. La clave (`_key_`) es el identificador y el valor (`_value_`) es el valor que queremos guardar en esa clave. La sintaxis es "clave: valor". Los objetos pueden contener muchos pares de clave-valor, deben estar separados por una coma (**importante**: sin punto y coma dentro de un objeto). Las claves son únicas en un objeto, solo puede haber una clave de ese nombre, aunque, varias claves pueden tener el mismo valor. Los valores pueden ser cualquier tipo de dato de Javascript; cadena, número, booleano, matriz, función o incluso otro objeto.

```
const miPrimerObjeto = {  
  nombre: 'Sherlock Holmes',  
  edad: 60,  
  genero: 'Masculino',  
  intereses: ['Violin', 'Boxeo'],  
  
  saludo: function() {  
    alert('Hola, soy ' + this.nombre +  
    '.')  }  
}  
  
miPrimerObjeto.nombre; // 'Sherlock Holmes',  
miPrimerObjeto.saludo() // 'Hola, soy Sherlock  
Holmes'
```


Acceso a elementos en un objeto

```
const miPrimerObjeto = {  
  nombre: 'Sherlock Holmes',  
  edad: 60,  
  genero: 'Masculino',  
  intereses: ['Violin', 'Boxeo'],  
  
  saludo: function() {  
    alert('Hola, soy ' + this.nombre +  
    '.')  }  
}  
  
miPrimerObjeto.nombre; // 'Sherlock Holmes',  
miPrimerObjeto.saludo() // 'Hola, soy Sherlock  
Holmes'
```

Una vez que tengamos los pares clave-valor, podemos acceder a esos valores llamando al nombre del objeto y la clave. Hay dos formas diferentes de hacer esto, usando puntos o usando corchetes.

Con la notación de puntos podemos llamar al nombre del objeto, un punto y el nombre de la clave. Así como llamamos a la propiedad `length` en un array. La propiedad de longitud es un par de clave-valor.

La **notación de corchetes** es como llamar a un elemento en una matriz, aunque con corchetes debemos usar una cadena o número, o una variable que apunte a una cadena o número. Se puede llamar a cada clave envolviéndola con comillas:

```
const miPrimerObjeto = {
  nombre: 'Sherlock Holmes',
  edad: 60,
  genero: 'Masculino',
  intereses: ['Violin', 'Boxeo'],

  saludo: function() {
    alert('Hola, soy ' + this.nombre + '.')
  }
}

const genero = 'genero'

miPrimerObjeto['nombre']; // 'Sherlock Holmes'
miPrimerObjeto['genero ']; // 'Masculino'
miPrimerObjeto.saludo(); // 'Hola, soy Sherlock Holmes'
```

Aclaración: en el caso de funciones dentro de un objeto, solo podemos llamarlas usando la notación de punto.

Asignación de valores

Asignar valores funciona igual que acceder a ellos. Podemos asignarlos, cuando creamos el objeto, o luego de que haya sido creado a través de la notación de puntos o con notación de corchetes:



```
const miPrimerObjeto = {  
  nombre: 'Sherlock Holmes',  
}  
  
var intereses: 'intereses';  
  
miPrimerObjeto.edad = 60;  
miPrimerObjeto['genero '] =  
miPrimerObjeto[intereses] = `['Violin', 'Boxeo'];  
  
console.log(miPrimerObjeto) // {  
  nombre: 'Sherlock Holmes',  
  edad: 60,  
  genero: 'Masculino',  
  intereses: ['Violin',  
    'Boxeo'],  
}
```

Eliminación de valores

Si queremos eliminar una propiedad, podemos hacerlo usando la palabra clave `delete`:

```
const miPrimerObjeto = {  
  nombre: 'Sherlock Holmes',  
}  
  
delete miPrimerObjeto.nombre || miPrimerObjeto.nombre = undefined  
console.log(miPrimerObjeto) // { }
```


Es raro que veamos el uso de la palabra clave `delete`, muchos consideran que la mejor práctica es establecer el valor de una palabra clave en `undefined`. Dependerá de ti cuando llegue el momento.

Palabra Clave THIS

```
const miPrimerObjeto = {  
  nombre: 'Sherlock Holmes',  
  edad: 60,  
  genero: 'Masculino',  
  intereses: ['Violin', 'Boxeo'],  
  
  saludo: function() {  
    alert('Hola, soy ' + this.nombre + '.')  
  }  
}  
  
miPrimerObjeto.nombre; // 'Sherlock Holmes',  
miPrimerObjeto.saludo() // 'Hola, soy Sherlock  
Holmes'
```

Los objetos tienen una palabra clave autorreferencial que se puede aplicar en cada objeto llamado `this`. Cuando se llama dentro de un objeto, se refiere a ese mismo objeto. `this` puede usarse para acceder a otras claves en el mismo objeto, y es especialmente útil en métodos:



A person with dark hair in a ponytail, wearing large white headphones, is seen from the side, working at a desk. They are holding a smartphone in their right hand. On the desk is a laptop displaying a colorful image. In the background, a large monitor shows a video call with a man. The scene is dimly lit, with some green plants visible in the background.

Recorriendo arrays de objetos con un bucle for

Programando tarjetas de productos

```
const productos = [  
  {id: 1, name: "Xayah", precio: "10"},  
  {id: 2, name: "Garen", precio: "20"},  
  {id: 3, name: "Twitch", precio: "30"},  
  {id: 4, name: "Yasuo", precio: "40"},  
  {id: 5, name: "Nasus", precio: "50"}  
]  
  
function crearTarjetas(productos) {  
  for (let i = 0; i < productos.length; i++) {  
    console.log(`  
      ${productos[i].name}  
      ${productos[i].precio}  
      - 0 +  
      AGREGAR  
    `)  
  }  
}
```



ACADEMY
by NUMEN