

ARREGLOS Y CICLOS

Arreglos

Los *arrays* son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un *array* son variables en general estas características son cómodas, pero de igual manera se puede considerar el uso de *arrays* con tipo.

Creación de arreglos

Para crear un arreglo se debe definir al igual que cualquier variable con las consideraciones de que se deben usar los paréntesis cuadrados para su correcta inicialización, en el ejemplo a continuación se muestra la creación de un arreglo.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code displayed is `let array = ["Manzana", "Pera"]` in a light blue monospace font.

Acceder a los elementos de arreglos

Para acceder a un arreglo es necesario entender que el arreglo es una estructura de datos manejada por un índice, es decir, que este índice va a ser la referencia para obtener y/o modificar el valor de un elemento dentro del arreglo. Es necesario recalcar que los índices de los arreglos en JavaScript **comienzan en cero**, esto quiere decir que para acceder al primer elemento de un arreglo se debe hacer referencia al índice cero. En el siguiente ejemplo se muestra cómo se accede a los distintos elementos de un arreglo.

```
let arreglo= ["Elemento 1", "Elemento 2" "Elemento 3"]

let primero = arreglo[0]
// Elemento 1

let ultimo = frutas[2]
// Elemento 3
```

```
let arreglo = ["Elemento 1","Elemento 2","Elemento 3"];
let primero = arreglo[0];
let ultimo = arreglo[2];
```

Contar los elementos de un arreglo

Para saber la cantidad de elementos que tiene un arreglo, se debe saber que los arreglos en JavaScript poseen propiedades y métodos que ayudaran a manipularlos, para obtener la cantidad de elementos que tiene un arreglo se utilizará la propiedad **length** la cual nos indica la cantidad exacta de elementos que tiene un arreglo.

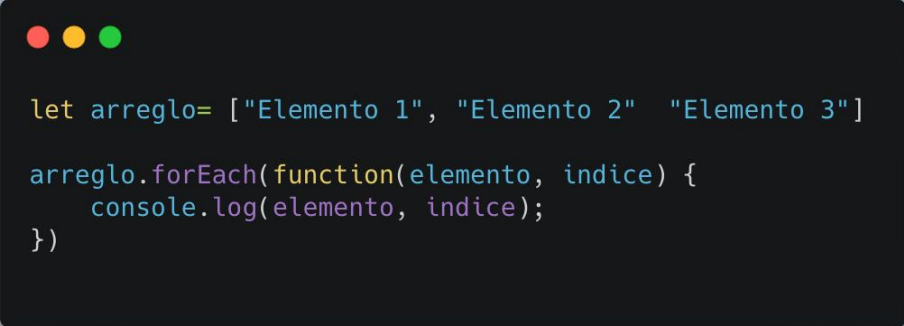
```
let arreglo= ["Elemento 1", "Elemento 2" "Elemento 3"]

let cantidadElementos = arreglo.length
// 3
```

Iterar en los elementos de un arreglo

Para iterar o recorrer sobre los elementos de un arreglo, existen muchas maneras, sin embargo, los arreglos nos proveen de métodos para poder manipularlos y el método que se utiliza para recorrer o iterar en los elementos de un arreglo es el método **forEach**.

En el ejemplo a continuación se muestra el uso de este método.



```
let arreglo= ["Elemento 1", "Elemento 2" "Elemento 3"]

arreglo.forEach(function(elemento, indice) {
  console.log(elemento, indice);
})
```

El método `forEach` de los arreglos recibe como parámetro una función la cual se ejecutará por cada uno de los elementos contenidos en el arreglo, esta función a su vez recibe como parámetros el elemento que se está iterando y el índice correspondiente. Más adelante se verán las funciones y se ahondará más en este tema.

Insertar y eliminar elementos de un arreglo.

De igual modo que en apartados anteriores, para insertar y eliminar elementos de un arreglo se deben utilizar las funciones que se proveen para ese objetivo.

Para insertar un elemento se puede hacer tanto al final usando el método `push` o al principio usando el método `unshift`.

Operaciones con arreglos.

Los arreglos al igual que los otros tipos de variables en JavaScript permiten realizar distintos tipos de operaciones entre ellos, en el presente apartado vamos a estudiar las operaciones más comunes entre los arreglos.

Unión o concatenación de arreglos, para unir dos o más arreglos se debe usar el método concat, este se ejecuta sobre el arreglo principal y recibe como parámetro el arreglo que queremos añadir.

```
const array1 = ['a', 'b', 'c'];  
const array2 = ['d', 'e', 'f'];  
const array3 = array1.concat(array2);  
  
console.log(array3);  
//Salida esperada: Array ["a", "b", "c", "d", "e", "f"]
```

Matrices

Las matrices no son más que un arreglo en el que cada uno de sus elementos son a su vez arreglos, y generalmente se usan para representar los datos de manera tabulada, es decir su estructura será del tipo fila columna.


Por ejemplo para representar la siguiente tabla en forma de matriz se debería hacer de la siguiente manera:

1	2	3
4	5	6
7	8	9

```
let matriz = [  
  [1,2,3],  
  [4,5,6],  
  [7,8,9],  
]
```

Arrays Asociativos

Un array asociativo es un array cuyos índices no son numéricos sino cadenas de texto (claves). En JavaScript no existen realmente arrays asociativos pero se pueden simular creando objetos y accediendo a sus propiedades



```
var coche = new Array();  
coche["color"] = "rojo";  
coche["marca"] = "seat";  
coche["modelo"] = "leon";
```

Ahora además del índice se puede utilizar la clave para acceder al valor, por ejemplo para acceder a la marca podremos usar **coche.marca**

Ciclos iterativos

Ciclos

Los ciclos o bucles son parte primordial en la programación en general y por supuesto en JavaScript.

Los ciclos se utilizan para repetir una instrucción una determinada cantidad de veces, en JavaScript existen muchos tipos de ciclos siendo los principales los que se describen a continuación.

While

El primer ciclo que se revisará es el más básico de todos, el ciclo While ejecuta una o varias instrucciones tantas veces como se cumpla una determinada condición.

En el siguiente ejemplo se muestra el uso de un ciclo while para imprimir por la consola la suma de los 10 primeros números naturales.

CHE
MSA
Middle States Commission
on Higher Education
3631 Market Street
Philadelphia, PA 19104-3680
UNIVERSIDAD ACREDITADA 2015 - 2020

Do While

El ciclo do while, al igual que el ciclo while ejecuta una o varias instrucciones dependiendo de una condición de salida, la única diferencia con respecto al ciclo while es que independiente de que si la condición de salida se cumpla o no se ejecutará al menos una vez.

En el siguiente ejemplo se ilustrará cómo funciona el ciclo do while, enfatizando en que siempre se ejecuta a lo menos una vez independiente de la condición de salida.



```
var x = 11

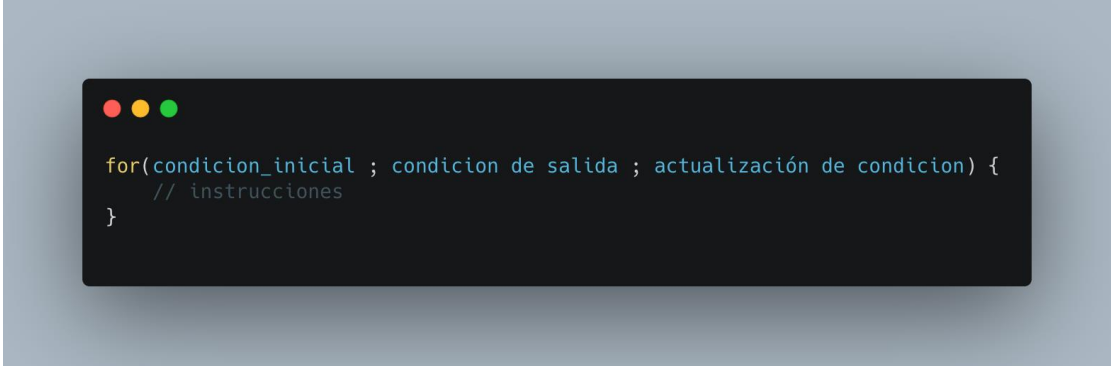
do {
  console.log(x)
  x++;
}while(x < 10 )
```

En el ejemplo se puede verificar que la variable x que es la que controla la condición de salida está inicializada con el valor de 11, también se puede verificar que la condición de salida se cumple cuando el valor de x es menor que 10, sin embargo como el ciclo que se está usando es un ciclo do while se imprimirá 11 en la consola ya que independiente de que la condición de salida del ciclo no se cumple primero se ejecutará el bloque de instrucciones “do” y luego se evaluará la condición de salida en el while.

For

El ciclo for es un poco más complicado, ya que tiene la inicialización de la o las variables de control, la condición de salida y el incremento de la variable de control en la misma definición del ciclo, generalmente se usa para recorrer listas de datos, arreglos o matrices, entre muchas otras utilidades.

El ciclo for tiene la siguiente estructura:



```
for(condicion_inicial ; condicion de salida ; actualización de condicion) {  
    // instrucciones  
}
```

Cuando se ejecuta un bucle for, ocurre lo siguiente:

1. Se ejecuta la expresión de condición Inicial, si existe. Esta expresión normalmente inicia uno o más contadores del ciclo, pero la sintaxis permite una expresión de cualquier grado de complejidad. Esta expresión también puede declarar variables.
2. Se evalúa la expresión la condición de salida. Si el valor de la condición de salida es verdadero, se ejecutan las instrucciones del ciclo. Si el valor de condición es falso, el bucle for termina.
3. Se ejecutan las instrucciones.
4. Si está presente, se ejecuta la actualización de la condición, y finalmente el control regresa al evaluar la condición de salida

Estos pasos se ejecutan hasta que la condición de salida se cumpla.

Ciclos anidados

Anidar un **ciclo o bucle** consiste en meter ese bucle dentro de otro. La anidación de bucles es necesaria para hacer determinados procesamiento un poco más complejos que los que hemos visto en los ejemplos anteriores. Si en vuestra experiencia como programadores los habéis anidado un bucle todavía, tener certeza que más tarde o temprano os encontraréis con esa necesidad.

Un bucle anidado tiene una estructura como la que sigue. Vamos a tratar de explicarlo a la vista de estas líneas:

```
for (i=0;i<10;i++){  
    for (j=0;j<10;j++) {  
        document.write(i + "-" + j)  
    }  
}
```

La ejecución funcionará de la siguiente manera. Para empezar se inicializa el primer bucle, con lo que la variable *i* valdrá 0 y a continuación se inicializa el segundo bucle, con lo que la variable *j* valdrá también 0. En cada iteración se imprime el valor de la variable *i*, un guión ("-") y el valor de la variable *j*, como las dos variables valen 0, se imprimirá el texto "0-0" en la página web. Debido al flujo del programa en esquemas de anidación como el que hemos visto, el bucle que está anidado (más hacia dentro) es el que más veces se ejecuta. En este ejemplo, para cada iteración del bucle más externo el bucle anidado se ejecutará por completo una vez, es decir, hará sus 10 iteraciones. En la página web se escribirían estos valores, en la primera iteración del bucle externo y desde el principio:

0-0
0-1
0-2
0-3
0-4
0-5
0-6
0-7
0-8
0-9

Para cada iteración del bucle externo se ejecutarán las 10 iteraciones del bucle interno o anidado. Hemos visto la primera iteración, ahora vamos a ver las siguientes iteraciones del bucle externo. En cada una acumula una unidad en la variable i, con lo que saldrían estos valores.

1-0
1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
1-9

Y luego estos:

2-0
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
2-9

Así hasta que se terminen los dos bucles, que sería cuando se alcanzase el valor 9-9.

Veamos un ejemplo muy parecido al anterior, aunque un poco más útil. Se trata de imprimir en la página las todas las tablas de multiplicar. Del 1 al 9, es decir, la tabla del 1, la del 2, del 3...

```
for (i=1;i<10;i++){  
    document.write("<br><b>La tabla del " + i + ":</b><br>")  
    for (j=1;j<10;j++) {  
        document.write(i + " x " + j + ": ")  
        document.write(i*j)  
        document.write("<br>")  
    }  
}
```

Con el primer bucle controlamos la tabla actual y con el segundo bucle la desarrollamos. En el primer bucle escribimos una cabecera, en negrita, indicando la tabla que estamos escribiendo, primero la del 1 y luego las demás en orden ascendente hasta el 9. Con el segundo bucle escribo cada uno de los valores de cada tabla.

Combinación de ciclos con instrucciones if/else:

La estructura if / else lo que hace es ejecutar una acción si el resultado de la evaluación de la expresión es verdadero y otra acción si el resultado de la evaluación es falso.

La diferencia con utilizar sólo la estructura if es que si la expresión evaluada es verdadera sólo en ese caso se ejecuta una acción de otro modo se pasa de largo.

En cambio en la estructura if / else si la expresión es falsa entonces se ejecuta otra acción.

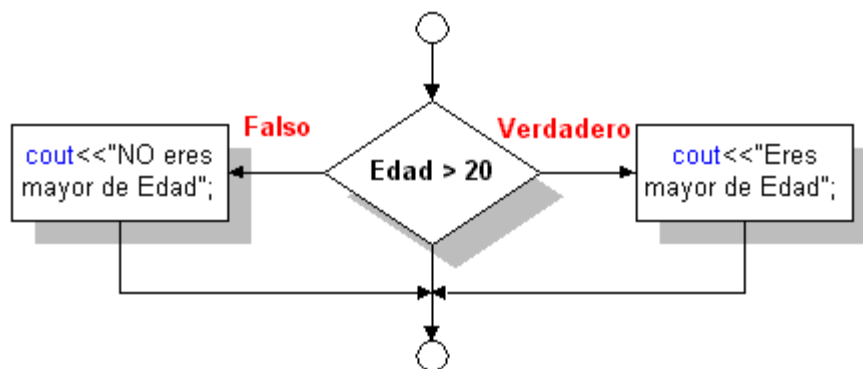
```
if (expresión(es))  
else (sentencias)
```

En síntesis lo que hace esta estructura es realizar una acción si la expresión es verdadera y otra si es falsa.

Ejemplo:

```
#include <iostream>
using namespace std;
int main()
{
    if ( edad > 20 )
        cout<<"Eres mayor de edad" ;
    else
        cout<<"No eres mayor de edad";
}
```

El diagrama de flujo correspondiente a esta estructura es el siguiente:



Estilos, convenciones y buenas prácticas de codificación

¿Qué es código limpio?

Clean Code, o Código Limpio, es una filosofía de desarrollo de software que consiste en aplicar técnicas simples que facilitan la escritura y lectura de un código, volviéndolo más fácil de entender.

El objetivo del código limpio es permitir modificar de manera fácil el mismo; dado que será entendible y evitará la inyección de errores y mayor productividad en el desarrollo.

- **Let:** Las variables declaradas como let tienen un alcance limitado a solo el bloque donde fue definido, como en cualquier bloque interno.

```
// Ejemplo de definiciones con var

function varTest() {
  var x = 10;

  if (true) {
    var x = 20;    // Es la misma variable
    console.log(x); // Imprime '20'
  }

  console.log(x);  // Imprime '20'

// Ejemplo de definiciones con let

function letTest() {
  let x = 10;

  if (true) {
    let x = 20;    // Es una variable diferente
    console.log(x); // Imprime '20'
  }

  console.log(x);  // Imprime '10'
}
```

- **Const:** Las variables declaradas como const presentan un ámbito de bloque (Block Scope) similar a las variables definidas como let, pero estas en particular son de sobre lectura, es decir no cambian con la reasignación de un valor en la variable.

```
const x = 10;

console.log(x);    // Imprime '10'

x = 20;            // Lanza un error de tipo "TypeError": Asignación a variable constante
```

Pero si creamos un objeto de tipo de const e intentamos cambiar una de sus propiedades es posible hacerlo:

```
const x = { value: 10 };

console.log(x.value);    // Imprime '10'

x.value = 20;            // Imprime '20'
```


Debido a esto existe confusión con el uso de las variables declaradas como const.

Usar nombre adecuado según el tipo de dato:

- **Arrays:** Los arrays son una lista iterable de elementos, generalmente del mismo tipo. Por eso es importante pluralizar los nombres de las variables.

```
// bad
const country = [ 'Colombia', 'Brazil', 'Argentina', 'México' ];

// regular
const countryList = [ 'Colombia', 'Brazil', 'Argentina', 'México' ];

// good
const countries = [ 'Colombia', 'Brazil', 'Argentina', 'México' ];

// better
const countryNames = [ 'Colombia', 'Brazil', 'Argentina', 'México' ];
```

- **Boleanos:** Las variables de tipo booleano solo devuelven dos valores, “true” o “false”. Dado esto, el uso de prefijos como “js”, “has”, “can” ayudara a mejorar la legibilidad de nuestro código.

```
// bad
const open = true;
const write = true;
const fruit = true;

// good
const isOpen = true;
const canWrite = true;
const hasFruit = true;
```

- **Números:**

Para las variables de tipo numérico es interesante escoger palabras que describan números, como “min”, “max”, “total”:

```
// bad
const fruits = 3;

// better
const maxFruits = 5;
const minFruits = 1;
const totalFruits = 3;
```

Referencias:

- **Bucles e iteración**
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Loops_and_iteration
- **Bucles anidados en JavaScript**
<https://desarrolloweb.com/articulos/619.php>
- **Ciclos for, do while, switch case, if else**
<https://sites.google.com/site/herramientasinformaticasdaniel/ciclos-for-do-while-switch-case-if-else>
- **Arrays**
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array