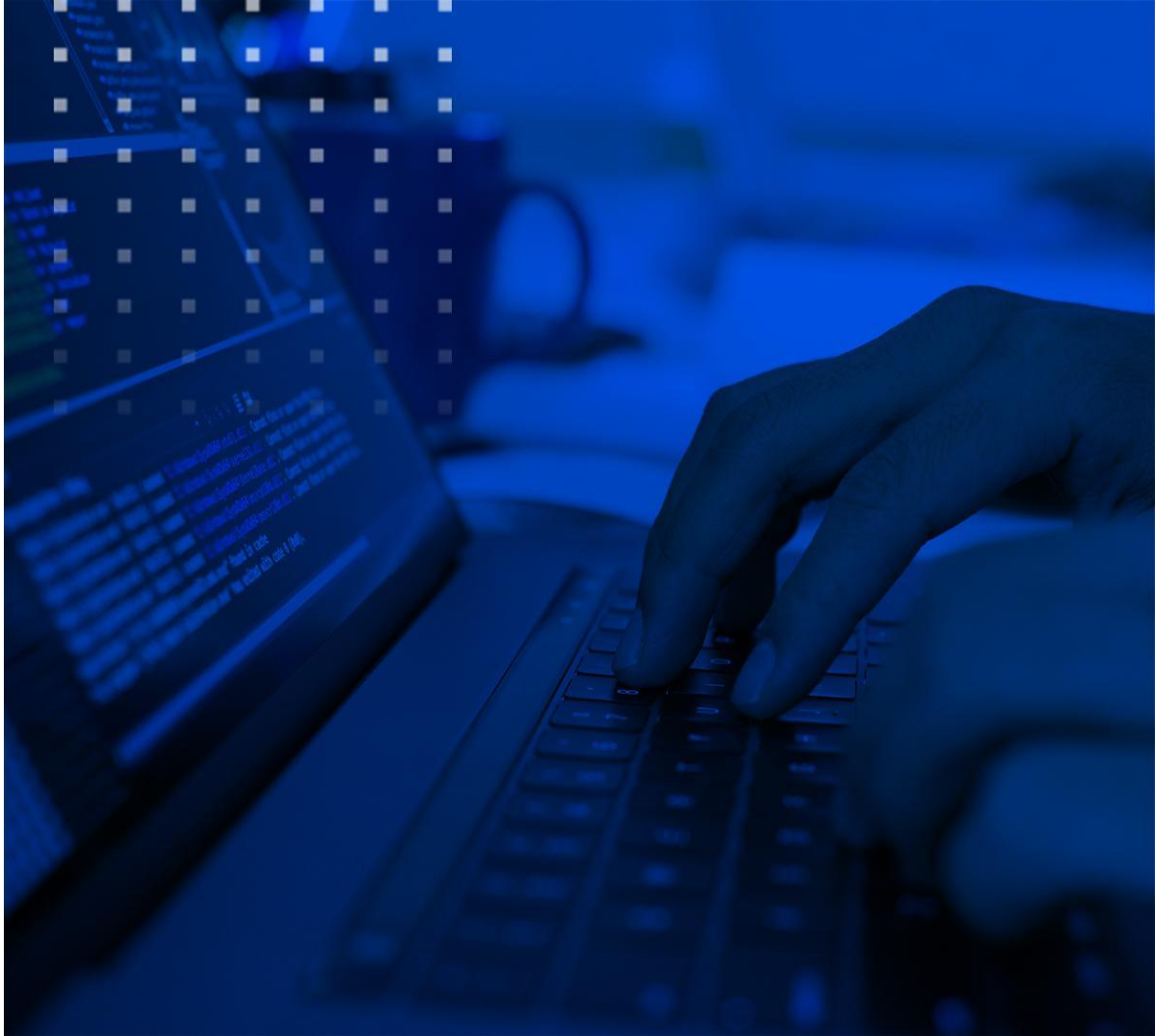
incluyeme.com

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

CURSO

FULL STACK DEVELOPER
NIVEL INICIAL



UNIDAD 6
JavaScript
Bucles e iteraciones

Supongamos que tenemos el siguiente array:

```
let frutas = [
"Manzana",
"Pera",
"Frutilla",
"Pera",
"Mora",
"Limón",
"Kiwi"
];
```

Si quisiera conocer la propiedad length de cada elemento del array frutas. Podría hacer esto:

```
frutas[0].length;
frutas[1].length;
frutas[2].length;
frutas[3].length;
frutas[4].length;
frutas[5].length;
frutas[6].length;
```

Pero no parece muy práctico, ¿no? -> Para esto existen los bucles

BUCLES

BUCLES

Los bucles ofrecen una forma rápida y sencilla de hacer algo repetidamente.

Vamos a ver algunos conceptos básicos:

BUCLES

1. Condición

Al igual que en los if, en los bucles **se va a evaluar una condición** para saber si se debe repetir el bucle o finalizarlo.

BUCLES

2. Iteración

Cada repetición de un bucle se denomina iteración. Por ejemplo, si un bucle repite una acción 10 veces, se dice que tiene 10 iteraciones.

BUCLES

3. Contador

Muchas veces, los bucles tienen una **variable** que se denomina contador, porque **cuenta** el número de repeticiones que ha hecho (esta variable hay que inicializarla antes de comenzar el bucle).

BUCLES

4. Incremento

Cada vez que terminemos un bucle se suele realizar el **incremento (o decremento) de una variable**, generalmente la variable contador.

BUCLES

5. Bucle infinito

Es lo que ocurre si en un bucle se nos olvida incrementar la variable contador o escribimos una condición que nunca se puede dar. El bucle *se queda eternamente repitiéndose y el navegador se queda «colgado»*.

WHILE

WHILE

La estructura *while* permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada.
El funcionamiento del bucle *while* se resume en: *mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle*.

WHILE

```
let i = 0; //  Inicialización de la variable contador
// Condición:  Mientras la variable contador sea menor de 5
while (i < 5) {
  console.log("Valor de i: ", i);
  i = i + 1; //  Incrementamos el valor de i
}
```

El bucle while crea un ciclo que se ejecuta siempre que se evalúe una condición especificada true. El ciclo continuará ejecutándose hasta que la condición se evalúe como false

WHILE

Repasemos:

- ★ Antes de entrar en el bucle, se inicializa la variable i a 0.
- ★ Antes de realizar la primera iteración, comprobamos la condición.
- ★ Si la condición es verdadera, hacemos lo que está dentro del bucle.
- ★ Mostramos en consola el valor de i y luego incrementamos el valor actual de i en 1.
- ★ Volvemos al inicio del bucle para hacer una nueva iteración: comprobamos de nuevo la condición del bucle.
- ★ Cuando la condición sea falsa, salimos del bucle y continuamos el programa.

WHILE

Iteración del bucle	Valor de i	Descripción	Incremento
Antes del bucle	i = undefined	Antes de comenzar el programa.	
Iteración #1	i = 0	¿(0 < 5)? Verdadero. Mostramos 0 por pantalla.	i = 0 + 1
Iteración #2	i = 1	¿(1 < 5)? Verdadero. Mostramos 1 por pantalla.	i = 1 + 1
Iteración #3	i = 2	¿(2 < 5)? Verdadero. Mostramos 2 por pantalla.	i = 2 + 1
Iteración #4	i = 3	¿(3 < 5)? Verdadero. Mostramos 3 por pantalla.	i = 3 + 1
Iteración #5	i = 4	¿(4 < 5)? Verdadero. Mostramos 4 por pantalla.	i = 4 + 1
Iteración #6	i = 5	¿(5 < 5)? Falso. Salimos del bucle.	

WHILE

¡Importante!

Evita los bucles infinitos.
Asegúrate de que la condición en un bucle eventualmente se convierta en false. Si no, el bucle nunca terminará.

```
while (true) {
  console.log("¡Soy un bucle infinito!");
}
```

DO... WHILE

DO... WHILE

```
let i = 0;
do {
  console.log("Valor de i: ", i);
  i = i + 1;
} while (i < 5);
```

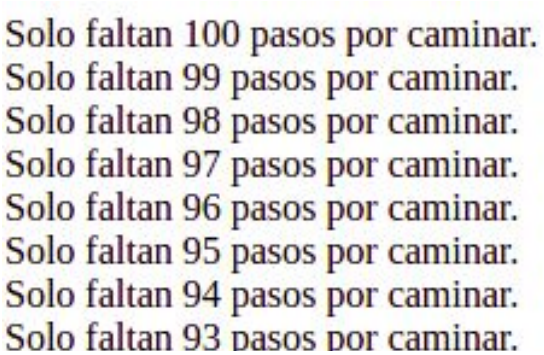
Al igual que while, se repite hasta que una condición especificada se evalúe como falsa. La diferencia es que, la condición se evalúa después de ejecutar la sentencia, esto hace que que la sentencia se **ejecute al menos una vez**.

¡A Practicar!

DO... WHILE

Para llegar a destino tenemos que caminar **100** pasos.
Usando el bucle **while** y el método **document.write()** mostrá por línea cuántos pasos quedan por caminar, partiendo del paso 100 hasta llegar al paso 1.

Debería verse *algo así* ->



FOR

FOR

```
for (inicialización; condición; actualización) {
        //sentencias a ejecutar en cada iteración
}
```

Los bucles for son comúnmente utilizados para **contar un cierto número de iteraciones** para repetir una sentencia. Declara instrucciones de bucle con tres piezas importantes de información separadas por punto y coma (;).

FOR

```
for (inicialización; condición; actualización) {
    //sentencias a ejecutar en cada iteración
}
```

- ★ **Inicialización**: define **dónde comenzar el ciclo** declarando (o haciendo referencia) a la variable iteradora
- ★ **Condición:** determina **cuándo detener** el bucle (cuando la expresión se evalúa como false).
- ★ **Actualización**: **actualiza el iterador** cada vez que se completa el ciclo.

FOR

```
for (let i = 0; i < 4; i++) {
  console.log(i);
}
// Salida en consola: 0, 1, 2, 3
```

El ciclo se repite hasta que la condición especificada (i < 4) se evalúe como false.

FOR

```
for (let i = 0; i < 4; i++) {
  console.log(i);
}
// Salida en consola: 0, 1, 2, 3
```

¡Importante!

El operador **++** agrega uno a su operando:
i++
es lo mismo que
i + 1

El ciclo se repite hasta que la condición especificada (i < 4) se evalúe como false.

EJEMPLO

En el siguiente ejemplo utilizamos un for para contar de 0 a 9.

```
for (let i = 0; i < 10; i++) {
    alert(i);
}
```

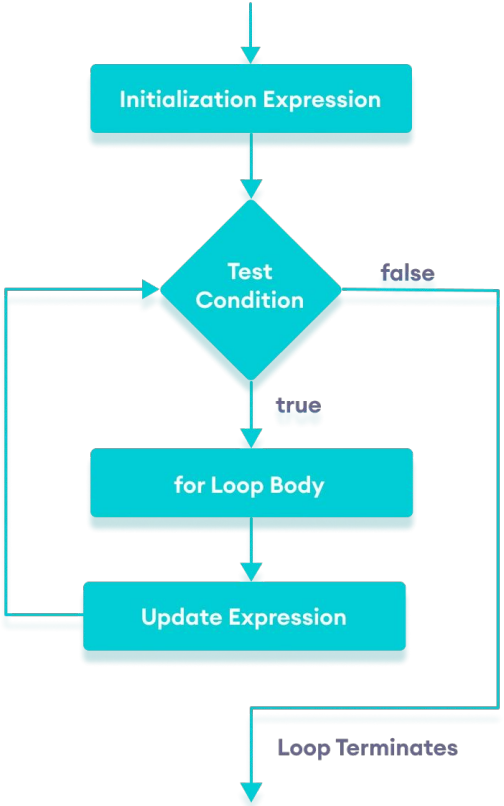
Ahora usamos for para contar de 1 a 10.

```
for (let i = 1; i <= 10; i++) {
    alert(i);
}
```

ALGORITMO PARA CALCULAR LA TABLA DE MULTIPLICAR DE UN NÚMERO

```
// Solicitamos un valor al usuario
let ingresarNumero = parseInt(prompt("Ingresar  Numero"));


//  En cada repetición, calculamos el número ingresado x el número de repetición
(i)
for (let i = 1; i <= 10; i++) {
    let resultado = ingresarNumero * i ;
    alert(ingresarNumero +" X "+ i +" = "+ resultado);
```


Initialization Expression
Test Condition
false
true
for Loop Body
Update Expression
Loop Terminates

¡A Practicar!

Crear una función que tome un **array de arrays de dos números** y me devuelva la suma total de la multiplicación de los dos números.

Nota: usando el **operador de asignación de adición**(**+=**) podemos re-asignar el valor de una variable sumando el valor anterior.

MÉTODOS ARRAYS

FOREACH

.ForEach()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.forEach(funcion);
```

Se usa para **recorrer un array**.
La variable objetos, en este caso, será el array a recorrer. Y la variable funcion es una función que será invocada por cada elemento que exista dentro del array. A esa función se le pasan 3 argumentos: el elemento, su índice, el array.

.ForEach()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.forEach(function(objeto, indice, array) {
    console.log(objeto, indice, array);
});
```

Acá vemos un ejemplo de cómo le pasamos la función con sus tres argumentos para recorrer el array.

Un poco confuso, ¿no?

.ForEach()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.forEach(function(objeto) {
    console.log(objeto);
});
```

No es necesario pasarle los tres argumentos. Podemos pasar uno solo: el elemento (en esta caso objeto).
Ahora es un poco más sencillo, pero...

.ForEach()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.forEach(objeto => {
  console.log(objeto);
});
```

¡Importante!

Las funciones flecha son lo mismo que las funciones sólo que con una sintaxis distinta.

Podemos usar **funciones flecha** para simplificar la sintaxis más todavía y hacer nuestro código más legible.

MAP

.map()

El método map() se utiliza para crear un nuevo Array con todos los elementos del Array original transformados según las operaciones de la función enviada por parámetro. El nuevo Array obtenido tiene la misma cantidad de elementos que el array original pero con los valores modificados.

.map()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.map(objeto => {
  console.log(objeto);
});
```

Al igual que forEach recibe una función con tres argumentos y lo que hace es recorrer el array. La diferencia es que devuelve una lista resultante.

.map()

```
let objetos = ["Varita", "Libro", "Lechuza", "Caldero"];

objetos.map(objeto => {
  return "Objeto: " + objeto;
});
```

Esto quiere decir que podemos guardar el resultado en una nueva variable con la información que quisiéramos añadir. En cambio, con forEach esto no daría undefined.

MÉTODO FIND

MÉTODO FIND

El método find() devuelve el valor del primer elemento del Array que satisface la función de comprobación enviada por parámetro. Si ningún valor satisface la función de comprobación, se devuelve undefined.

```
const numeros     = [1, 2, 3, 4, 5];

//La función parámetro generalmente es una función flecha sin cuerpo.
const encontrado = numeros.find(elemento => elemento > 3);
```

FILTER

.filter()

```
let numeros = [2, 5, 6, 18, 201];

numeros.filter(n => {
  return n > 5
});
```

Al igual que forEach y map recibe una función con tres argumentos. Pero en esta caso, se usa para **filtrar elementos**. En este caso, el resultado serán los elementos mayores a 5.

¡A Practicar!

1. Recorrer un **array de números** con el método **.map()** y crear un nuevo array que indique el número y si el número es par o impar.

2. Crea un array de strings con nombres y luego guarda en una variable **nombresCortos** un nuevo array con los nombres que tengan 5 0 menos letras.


[i]ncluyeme
.com


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES