educación IT
CODE YOUR FUTURE

JavaScript desde cero

Módulo 3

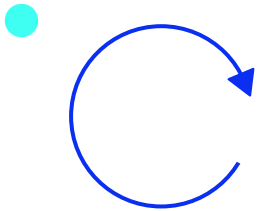
Ciclos de iteración (Bucles)

Ciclos de iteración (Bucles)

JavaScript cuenta cláusulas denominadas ***ciclos de iteración***. Permiten **generar tareas de forma repetitiva**, de acuerdo a nuestra necesidad.

Los ciclos de iteración, se dividen en dos:

- Ciclos de iteración **por conteo**.
- Ciclos de iteración **por repetición**.



Ciclos de iteración por conteo (*for* y *for...of*)

Ciclos de iteración por conteo (*for*)

Este es **el ciclo más conocido** por todos: el ciclo **for**. Este es ideal para **poder recorrer un arreglo, o *array*, y realizar diferentes operaciones** sobre los datos, objetos, elementos o cualquier otro conjunto de características que conformen ese arreglo.

Tener presente que, el ciclo **for**, no nos limita a utilizarlo solo para recorrer arreglos. **También se puede implementar para iterar dentro de un rango de valores estáticos, inicial y final**, previamente definidos.

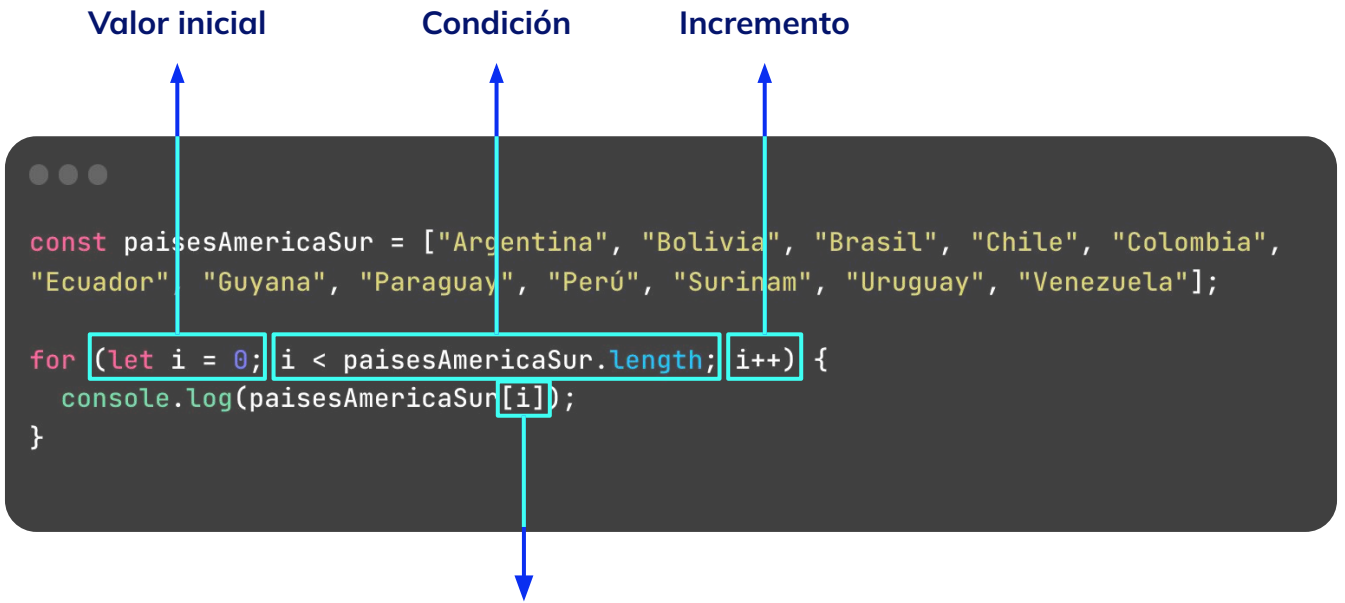
Si bien, el uso del ciclo **for** se puede aprovechar muy bien con *arrays*, estos cuentan con otros ciclos y métodos más modernos que permiten recorrerlos mucho más rápido.

Estructura paso a paso

El ciclo `for` cuenta con una estructura configurable a través de sus **paréntesis**, en la cual recibe **tres parámetros**:

1. ***Valor inicial***.
2. ***Condición***.
3. ***Incremento***.

Veamos un ejemplo siguiente diapositiva.



El valor cambiante de **i**, nos permite acceder a cada elemento del *array*.

1. **Valor inicial:** se declara una variable, usualmente `i`, con el valor inicial `0` (**cero**).

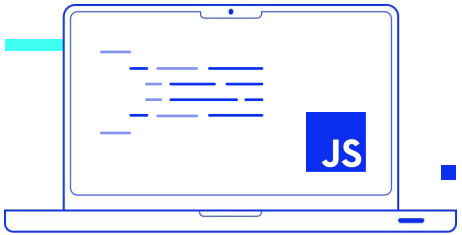
   Si utilizamos el ciclo **`for`** para iterar un *array*, la variable `i` será la que tendrá un valor numérico, cambiante, que podremos utilizar para acceder a los diferentes elementos del *array* que estamos iterando.

2. **Análisis de una condición:** en el segundo parámetro, se evalúa una condición; en el ejemplo del *slide* anterior, se valida que el valor que posee `i`, sea menor al total de elementos del *array*. **Solo si esta condición se cumple, se ejecutará el código** que se defina internamente **dentro de las llaves de bloque del ciclo `for`**.

Se utiliza la propiedad `.length`. Usar esta propiedad (cuando se define un ciclo `for` sobre un arreglo) **es clave**, ya que ayuda a **prevenir que, si el arreglo incrementa o decrece su contenido, el ciclo sigue siendo 100% funcional** y lo puede continuar iterando sin problema.

3. **Incremento:** el tercer parámetro, **incrementa en un dígito el valor que posea `i`**. Esto **reinicia la iteración** sobre el arreglo, debiendo previamente volver a validar la condición definida como segundo parámetro.

   Si la condición del segundo parámetro se sigue cumpliendo, seguimos iterando el *array* y trabajando con el siguiente elemento de este.



Interrupción del ciclo

Cláusula *break*

La cláusula **break**, permite **interrumpir el ciclo de iteración**. Por ejemplo, podemos evaluar una condición determinada y, si esta se cumple, interrumpirlo.

En nuestro ejemplo, interrumpimos la iteración si el valor de **i**, es igual a **4**.

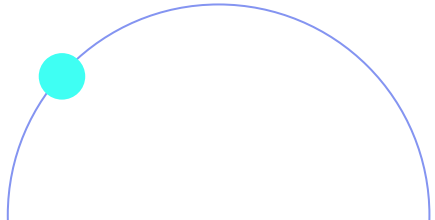
```
for (let i = 0; i < paisesAmericaSur.length; i++) {
  console.log(paisesAmericaSur[i]);
  if (i === 4) {
    break
  }
}
```

Cláusula *continue*

La cláusula **`continue`**, permite **saltear un paso del ciclo de iteración**.

De igual forma, se puede evaluar una condición determinada y si, esta se cumple, aplicar `continue`.

```
for (let i = 0; i < paisesAmericaSur.length; i++) {
  console.log(paisesAmericaSur[i]);
  if (i === 4) {
    continue
  }
}
```



Ciclo *for...of*

El ciclo **`for...of`** ofrece una **forma de iterar más simple** que la utilizada con el ciclo **`for`** convencional.

```
const paises = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia", "Ecuador",
"Guyana", "Paraguay", "Perú", "Surinam", "Uruguay", "Venezuela"]

for (let pais of paises) {
    console.log(pais)  //mostrará cada elemento contenido en el array
}
```

Funcionamiento

En cada iteración, le asigna el valor del elemento actual a una variable, lo que hace que sea más sencillo recorrer y trabajar con los elementos de la estructura.

Este ciclo recorre de principio a fin el *array* y valida de forma interna el total de elementos u objetos que tenga el *array*.

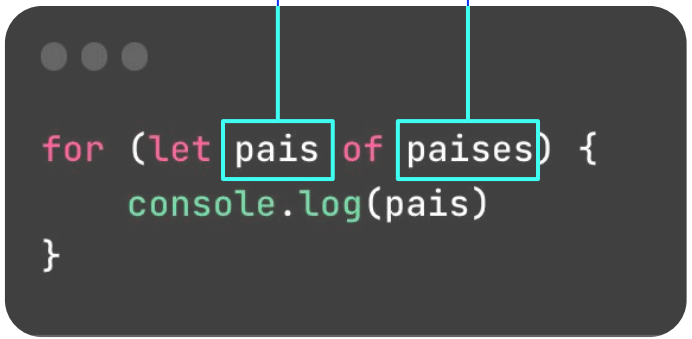
También, **ayuda a eliminar el uso de índice para acceder a cada elemento u objeto del *array***.

Estas **funcionalidades simplificadas** lo convierten en una alternativa al ciclo `for` tradicional, mucho más atractiva, dado que **simplifica la sintaxis y mejora la legibilidad del código**.

Veamos un ejemplo en la próxima diapositiva.

Variable donde asignamos cada elemento del *array*

Nombre del *array* a iterar



Otro ejemplo

Veamos otro ejemplo del ciclo **for...of aplicado a un *array* de objetos**. En este caso, el valor que toma la variable producto, declarada dentro del ciclo, contendrá la estructura de cada objeto literal que conforma este *array*.

```
const productos = [
  { id: 1, nombre: 'TV 55', precio: 1000, stock: 30, categoria: 'Video'},
  { id: 2, nombre: 'Laptop', precio: 1500, stock: 100, categoria: 'Computación'},
  { id: 3, nombre: 'iPhone 8', precio: 800, stock: 42, categoria: 'Telefonía'},
  { id: 4, nombre: 'Tablet', precio: 500, stock: 71, categoria: 'Computación'},
  { id: 5, nombre: 'Pods', precio: 100, stock: 28, categoria: 'Audio'},
  { id: 6, nombre: 'MP3 player', precio: 200, stock: 11, categoria: 'Audio'},
  { id: 7, nombre: 'Videocámara', precio: 300, stock: 22, categoria: 'Video'},
  { id: 8, nombre: 'Smartwatch', precio: 250, stock: 88, categoria: 'Computación'},
  { id: 9, nombre: 'Impresora', precio: 150, stock: 14, categoria: 'Accesorios'},
  { id: 10, nombre: 'Altavoces', precio: 120, stock: 18, categoria: 'Audio'}
];

for (let producto of productos) {
    console.table(producto)  //mostrará cada objeto contenido en el array
}
```

educación IT
CODE YOUR FUTURE

Ciclo de iteración por repetición (*while* y *do-while*)

Ciclo de iteración por repetición (*while*)

El ciclo de iteración **`while` repite un bloque de código mientras una condición sea verdadera**. Antes de cada iteración, se evalúa la condición. Si es verdadera, ejecuta el bloque de código; si no, detiene el ciclo.

Siempre debemos tener presente que, este ciclo de iteración, **debe controlar dentro del bloque de código el cambio de la condición** que lo hace funcionar para, en algún momento, poder interrumpir su ejecución. **Si no lo hace, desencadenará un bucle infinito**.

```
let contador = 0

while (contador < 5) {
  console.log("Contador: " + contador)
  contador++
}
```

Ejemplo

Veamos un ejemplo de código funcional, que integra el ciclo **`while`**, y se ejecutará repetidamente, hasta tanto el país que se ingrese en el cuadro de diálogo **`prompt()`** se encuentre listado dentro del *array* **`países`**.

Sino, seguirá pidiéndonos que ingresemos otro país para buscar en el *array*.

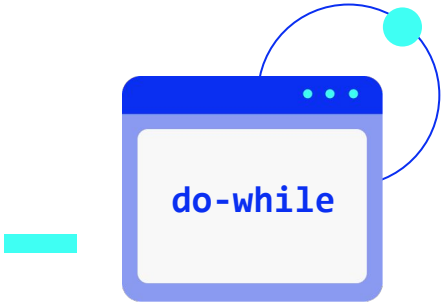
Veámoslo en la pantalla a continuación.

```
const paises = ["Argentina", "Bolivia", "Brasil", "Chile", "Colombia", "Ecuador",
"Guyana", "Paraguay", "Perú", "Surinam", "Uruguay", "Venezuela"]

let continuar = true

while (continuar === true) {
    let paisAbuscar = prompt("Ingresa el país a buscar:")
    let indice = paises.indexOf(paisAbuscar)
    if (indice > -1) {
        console.log("El país ingresado se encuentra en el índice:", indice)
        continuar = false
    } else {
        alert("No se encontró el país " + paisAbuscar)
    }
}
```

Ciclo de iteración por repetición (*do-while*)

El ciclo de iteración **`do-while` repite un bloque de código mientras una condición sea verdadera**.

Aquí, **la condición se analiza al final de cada iteración**, y se debe contemplar la misma lógica que con el ciclo **`while`**, para no generar un *loop* infinito.



Diferencia entre *while* y *do-while*

La diferencia de funcionamiento entre los ciclos `while` y `do-while` es que, **`while`**, **puede ejecutarse cero veces**, si la condición que evalúa da como resultado `false`.

```
let contador = 0

while (contador < 5) {
  console.log("Contador: " + contador)
  contador++
}
```

En cambio, **`do-while` se ejecutará al menos una vez**, dado que la condición que evalúa, lo hace al final de la iteración.

```
let contador = 0;

do {
  console.log("Contador: " + contador);
  contador++;
} while (contador < 5);
```

Interrupción del ciclo

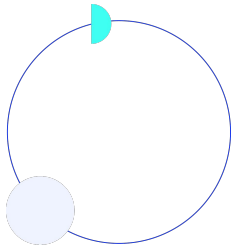
Ambos ciclos de iteración por repetición tienen la posibilidad de integrar, en el bloque de código de ejecución, las cláusulas **`break`** y **`continue`** (como se vió en el ciclo `for`, que permite interrumpir o saltear una iteración, en base a una condición válida).

Atención

Si bien los ciclos **`while`** y **`do-while`** pueden pensarse también para recorrer *arrays* y operar con sus elementos u objetos, **lo más efectivo para el manejo de arreglos, siempre, es utilizar alguna de las alternativas del ciclo `for`**:

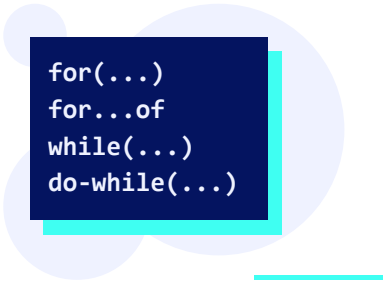
- `for` convencional.
- `for...of`.
- `forEach()`.

De **`forEach`** no hemos hablado todavía, pero **es la más apropiada para iterar elementos de un *array***, dado que está 100% optimizada para este propósito, lo cual hace que, en *arrays* con cientos o miles de elementos, sea la opción más efectiva de iteración, por sobre las otras.



Revisión

- Repasar el concepto de **bucle**.
- Mostrar datos de un arreglo con **`for`**.
- Mostrar datos de un arreglo con **`while`**.
- Mostrar datos de un **arreglo con objetos que contengan propiedades**.
- Trabajar en el ***Proyecto integrador***.
- Realizar las preguntas necesarias a la/el docente antes de continuar.



¡Sigamos trabajando!