



BOOTCAMP

# Generación T

 streambe

# Clase 28

## Loops



[11 6133-1747](https://wa.me/5491161331747)



GENERACIÓN T

[generaciont@generaciont.org](mailto:generaciont@generaciont.org)

# Qué Son Los Loops En Programación Y Para Qué Nos Sirven

Los Loops (en castellano, Bucles) son muy útiles. Sirven para repetir una tarea, una determinada cantidad de veces, hasta que se cumpla una condición dada de antemano.

# While

el while Loop se parece a la declaración del if...else ya que ambos evalúan una condición y ejecutan un bloque de código.

Sin embargo, el while Loop lo re-evalúa repetidas veces y ejecuta su bloque de código hasta que la condición deja de ser verdadera.

veamoslo en el vscode

# Sintaxis Del while Loop

```
while (unaCondicion) {  
    //Ejecuta este código  
    // Hace algo para que la condición eventualmente se deje de cumplir  
}
```

⚠ Importante: Este código se va a ejecutar siempre que el resultado sea true. Cuando la condición sea false, saldremos del bucle, evitando entrar en lo que se conoce como un Loop Infinito.

# Sintaxis Del while Loop

```
let pasajero = 1
while(pasajero <= 20) {
  console.log("Puede pasar, su asiento es el: " + pasajero)
  pasajero++
}
```


Con este código, podríamos cobrar el boleto a cada una de las 20 personas que estén en la fila, dejando pasar a quienes paguen, desde el primero hasta el último pasajero.

# Probemos en la consola el código y veamos el resultado.

1. Declaramos una Variable: En este caso, la llamamos pasajero y le asignamos el valor 1.
2. Definimos el while y la condición que tiene que evaluar el programa: "Mientras(while) la variable pasajero sea menor o igual a 20 ( $\text{pasajero} \leq 20$ )..."
3. Dentro de las llaves(`{}`), ponemos el código que queremos que se ejecute siempre que la condición sea verdadera. En este caso, que se muestre por consola si el pasajero puede pasar y cuál es su lugar.
4. Antes de cerrar la llave, ponemos un código que modifica la Variable para que, eventualmente, la condición sea false. En este caso, `pasajero++` incrementa en 1 el valor de la Variable cada vez que se ejecuta. Esto hará que, eventualmente, pasajero sea mayor a 20 y que la condición sea falsa (saliendo del bucle).

# ¿Qué Hace ++ Y --?

El ++ incrementa en una unidad el valor de la variable. En cambio, -- la decrece.

 Importante: En caso de querer incrementar de 2 en 2, por ejemplo, podés usar los operadores de esta forma:

```
let count = 5
while(count <= 20) {
  console.log("count es: "+count)
  count+=2
}
```



# Loops Infinitos

Los Loops Infinitos ocurren cuando la condición de un while Loop nunca es falsa. Por lo tanto, el código corre indefinidamente y se cuelga el programa.

Veamos un ejemplo.

```
let x=0
while(x < 10) {
  console.log(x)
}
```

¿Por Qué Es Un Loop Infinito?

En este código, x es igual a 0 y la condicional dice: "Mientras x sea menor que 10 se debe mostrar por consola a x".

# Operadores Aritméticos Avanzados

Los Operadores Aritméticos Avanzados de JavaScript nos permiten escribir un código más preciso y con una mejor funcionalidad.

`++`: Le suma 1 unidad a la Variable.

`--`: Le resta 1 unidad a la Variable.

`+=`: Le suma las unidades que querramos a la Variable.

`-=`: Le resta las unidades que queramos a la Variable.

`*=`: Multiplica a la Variable por la unidad que queramos.

# while Loop Con Cadenas De Caracteres (Strings)

En los while Loop también podemos evaluar condiciones usando cadenas de caracteres (en inglés Strings).

Los while Loop también pueden evaluar condiciones que sean Strings. Por ejemplo, si quisiéramos corroborar cuál es la palabra mágica para entrar a La Caverna De Los 40 Ladrones, podríamos escribir estas líneas de código:

```
let palabraMagica = "¡Ábrete, sésamo!"

let intento = prompt("Dígame, ¿cuál es la Palabra Mágica?")

while(intento != palabraMagica){
  alert("Esa palabra es incorrecta")
  intento = prompt("Inténtelo nuevamente")
}
alert("¡Bienvenido a La Caverna De Los 40 Ladrones!")
```

# Variables Acumuladoras

```
let i = 1
let final = 5
let acumulador = 0
while(i <= final){
  acumulador += i
  i++
}
console.log(acumulador)
```

En este ejemplo vemos un **loop** que tiene 5 iteraciones, ya que arranca desde el 1 y termina en el 5. La variable acumuladora va a juntar el valor de la *i* en cada iteración.

La cuenta **acumulador += i** se puede escribir también como **acumulador = acumulador + i**. Es importante analizar que pasa en cada iteración:

# Variables Acumuladoras

```
let i = 1
let final = 5
let acumulador = 0
while(i <= final){
  acumulador += i
  i++
}
console.log(acumulador)
```

Por ejemplo, en la primera vuelta sería  $\text{acumulador} = 0 + 1$ ;

En la segunda,  $\text{acumulador} = 1 + 2$ ;

En la tercera  $\text{acumulador} = 3 + 3$ ;

En la cuarta  $\text{acumulador} = 6 + 4$  y;

En la quinta  $\text{acumulador} = 10 + 5$  (el valor final de la variable acumulador es 15).

# While Loops Anidados

Para iniciar al entendimiento de los loops anidados te recomendamos que copies este código en tu consola y analices el resultado:

```
let i = 1;
let j = 1;
let final = 5;
while (i <= final) {
  j = 1;
  while (j <= final) {
    console.log(i + " - " + j);
    j++;
  }
  i++;
}
```

# Explicación

```
let i = 1;
let j = 1;
let final = 5;
while (i <= final) {
  j = 1;
  while (j <= final) {
    console.log(i + " - " + j);
    j++;
  }
  i++;
}
```



consola:

```
1 - 1
1 - 2
1 - 3
1 - 4
1 - 5
```

Como podrás observar, el primer **while** se mantiene en la iteración número 1 mientras que el **while** interno se recorre (a la izquierda esa la **i** y a la derecha está la **j**):

# Explicación

```
let i = 1;
let j = 1;
let final = 5;
while (i <= final) {
  j = 1;
  while (j <= final) {
    console.log(i + " - " + j);
    j++;
  }
  i++;
}
```



consola:

```
2 - 1
2 - 2
2 - 3
2 - 4
2 - 5
```

Una vez que termina el **while** interno de hacer su recorrido, pasamos a la segunda vuelta del primer **while**, en donde la *i* va a valer 2, entonces:



# Explicación

```
let i = 1;
let j = 1;
let final = 5;
while (i <= final) {
  j = 1;
  while (j <= final) {
    console.log(i + " - " + j);
    j++;
  }
  i++;
}
```



consola:

```
3 - 1
3 - 2
3 - 3
3 - 4
3 - 5
```

Y así, hasta que termine la última iteración del **while** principal, es decir, en la iteración 5, veamos como sigue hasta el final:

# Explicación

```
let i = 1;
let j = 1;
let final = 5;
while (i <= final) {
  j = 1;
  while (j <= final) {
    console.log(i + " - " + j);
    j++;
  }
  i++;
}
```



consola:

```
5 - 1
5 - 2
5 - 3
5 - 4
5 - 5
```

Y así, hasta que termine la última iteración del **while** principal, es decir, en la iteración 5, veamos como sigue hasta el final:

# Obligar al usuario a ingresar un input

Antes de pasar a los ejercicios, pensemos cómo debería ser el código de un programa que le pida información obligatoria a un usuario.

Podríamos escribir el siguiente código usando un while Loop:

```
let input
while( !(input = prompt('Escriba su nombre, por favor.')) ){
  alert("No recibimos la información.")
}
alert(";Gracias! Su nombre es: " + input + ".")
```

El prompt que le pide el nombre al usuario correrá hasta que el usuario ingrese algo en ese campo.

Una vez recibido un input, tendrá un String con contenido. Por lo tanto, la negación lo convertirá en false y el programa saldrá del Loop.

# Bonus: Objeto Math

¿Qué Es El Objeto **Math**?

El Objeto Math contiene Propiedades y Métodos relacionados a Matemática. Un gran uso del mismo es crear números random.

Probá por consola para saber qué hace cada una de las siguientes expresiones:

```
Math
```

```
Math.PI
```

```
Math.E
```

```
Math.pow(9, 2)
```

```
Math.random()
```

```
Math.floor(7.2)
```

```
Math.ceil(7.2)
```

```
Math.ceil( Math.random() * 10 )
```

```
Math.floor( Math.random() * 10 )
```



GENERACIÓN T