

Estructuras de Control de Repetición

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Bucle

Un **bucle** es una estructura de control que **repite instrucciones**.

Un bucle entonces nos permite repetir un bloque de instrucciones determinado hasta que se cumpla cierta condición.

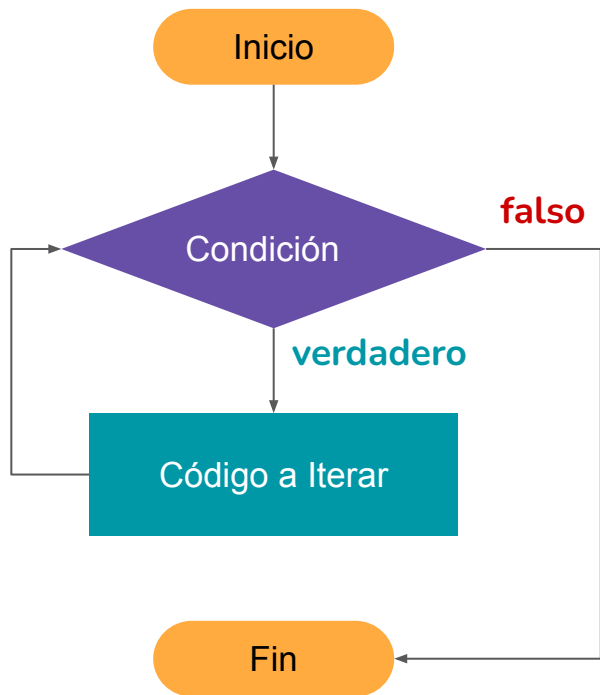
Cada repetición se suele llamar iteración.



Ciclo While

DEV.F
DESARROLLAMOS(PERSONAS);

dev



Ciclo While

La idea principal del ciclo while es: **MIENTRAS** se cumpla la condición **REALIZAR** estas acciones. Cuando la condición deje de cumplirse salimos del bucle y continúa el flujo del programa.

Muy importante: En el ciclo while la condición es lo primero que se evalúa, antes de ejecutar el código a iterar.

Sintaxis: Ciclo while

Usamos la palabra reservada **while**, seguida de la condición entre paréntesis **()** y finalmente colocamos el código que se repetirá entre llaves **{}**

```
while (condicion) {  
    // codigo a ejecutar  
}
```

Importante: Necesitamos en el código a iterar insertar una variable de control que nos permita salir eventualmente el ciclo while. En caso contrario nuestro programa se quedará ciclado “infinitamente”.

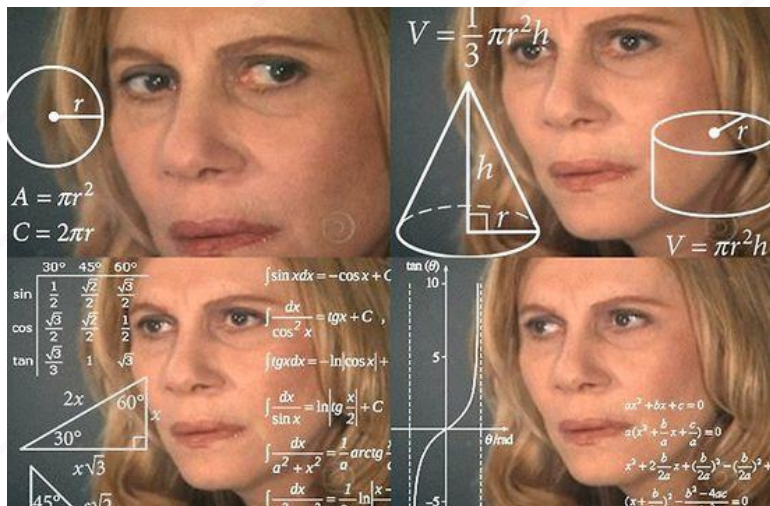
Demostración: Ciclo while

Me gustaría escribir un programa que pida al usuario un dato. Que éste imprima en consola “El usuario ingresó DATO” y vuelva a pedir otro.

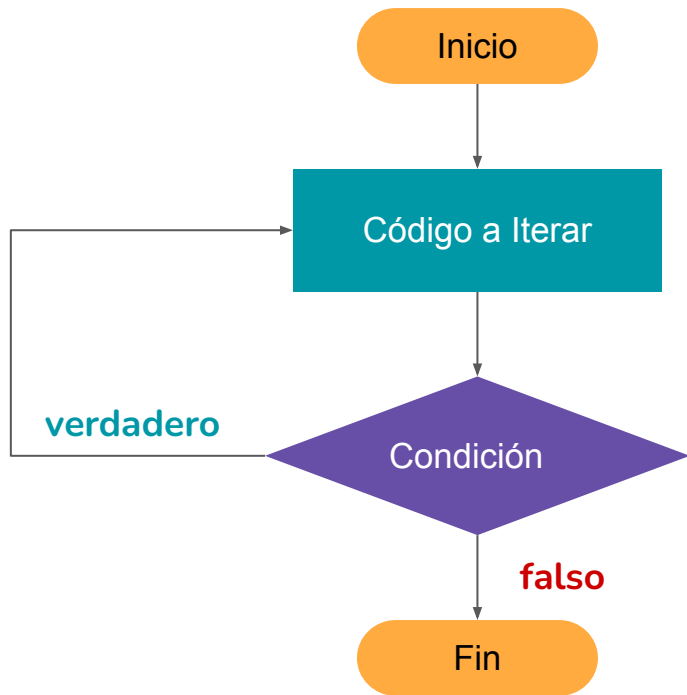
Pero si el usuario ingresa ‘esc’ el programa ya no pida más datos.

Ejercicio en clase: La tabla de 12

Escribamos un programa que imprima en consola la tabla del 12
(del 1 al 10)



05:00



Ciclo Do While

Variante del ciclo While puro, con la diferencia que la primera vez siempre se ejecuta el código y posteriormente evalúa la condición para ver si se vuelve a ejecutar.

Sintaxis: Ciclo do while

Usamos la palabra reservada **do**, seguido del código que se repetirá entre llaves **{}**, seguido de la palabra reservada **while** y finalmente la condición a evaluar en cada iteración entre paréntesis **()**.

```
do {  
    // código a ejecutar  
}  
while (condicion);
```

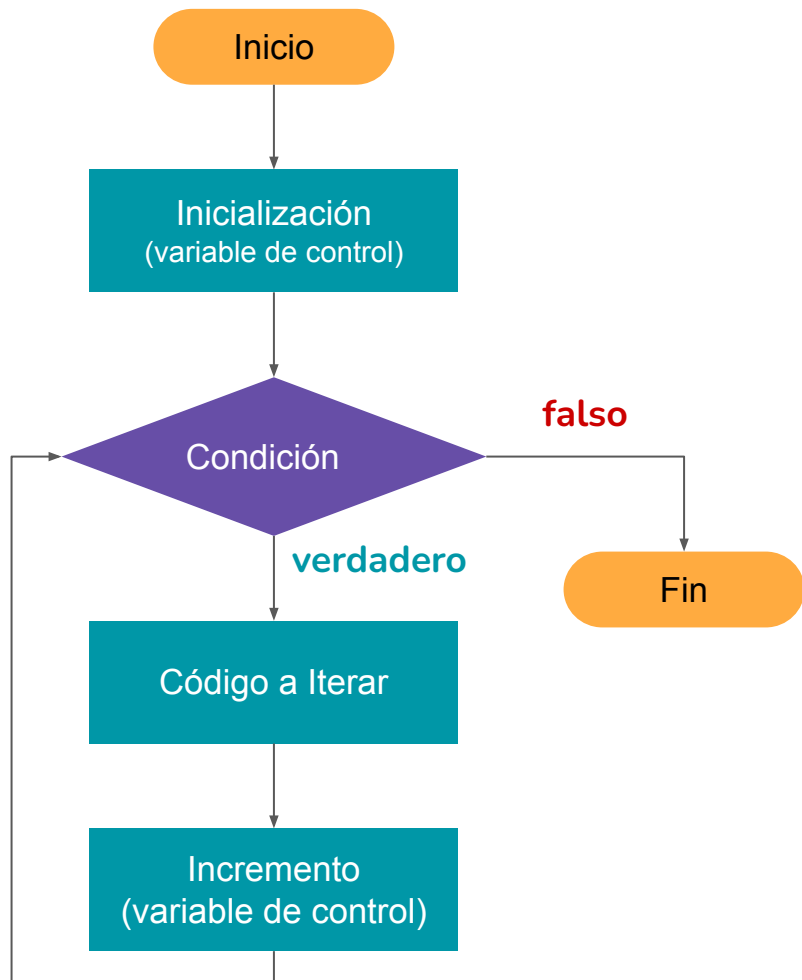
Demostración: Ciclo do while

Pidamos al usuario que ingrese un número entre 1 y 10. Utilizando un ciclo do while, le seguiremos pidiendo que ingrese el un número hasta que ingrese un número dentro de ese rango.

Ciclo For

DEV.F
DESARROLLAMOS(PERSONAS);

dev

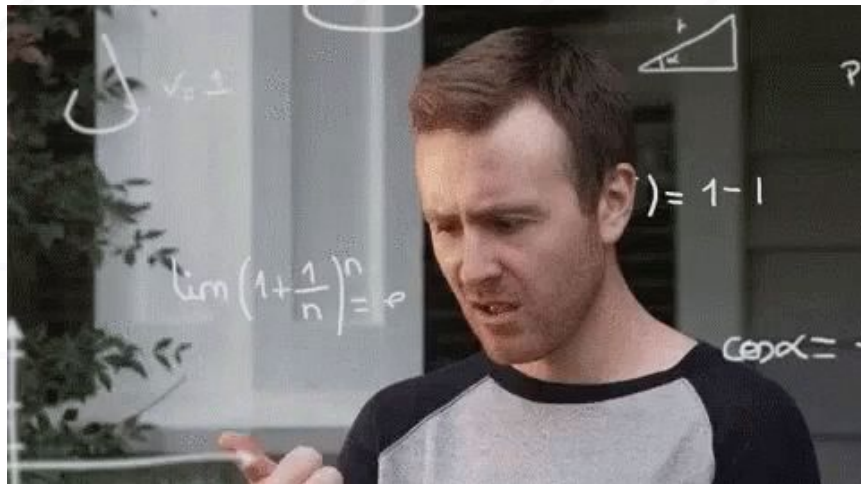


Ciclo for

Un **bucle for** es un bucle que **repite** el bloque de instrucciones **un número predeterminado de veces**.

Contadores y acumuladores

En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que acumulen valores (acumuladores).



Contador

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición.

```
> // Del 1 al 10 ¿Cuántos números son múltiplos de 2?  
var contador = 0;  
for (var index = 1; index <= 10; index++) {  
  if (index % 2 == 0) {  
    contador = contador + 1;  
    console.log(`${index} es múltiplo de 2`);  
  }  
}  
console.log(`De 0 a 10 existen ${contador} múltiplos de 2`);
```

2 es múltiplo de 2

4 es múltiplo de 2

6 es múltiplo de 2

8 es múltiplo de 2

10 es múltiplo de 2

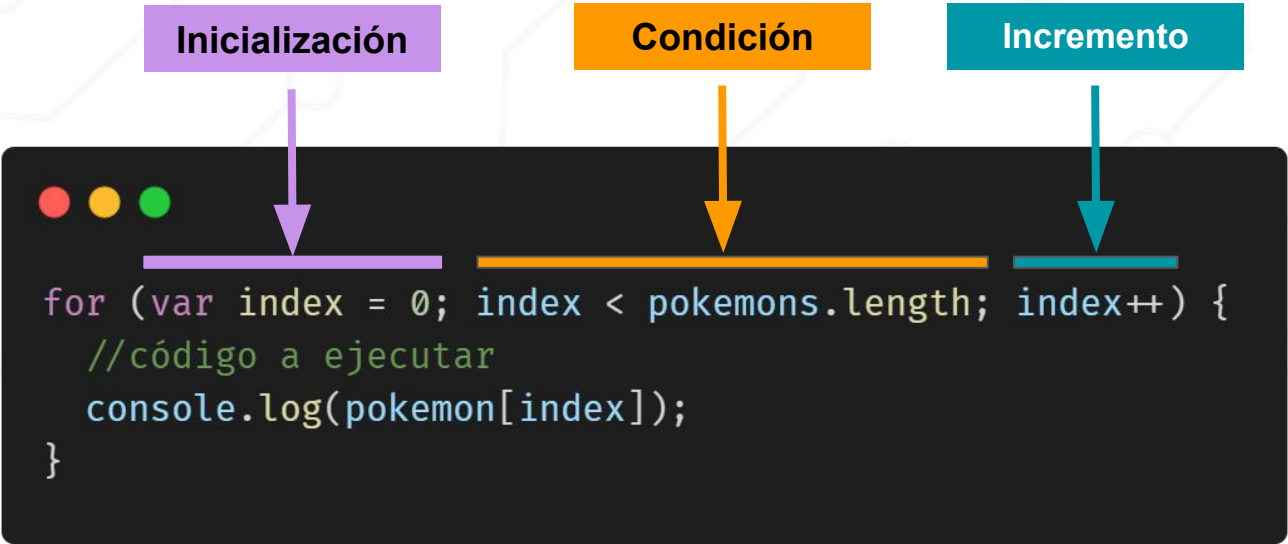
De 0 a 10 existen 5 múltiplos de 2

Sintaxis Ciclo for

Inicialización

Condición

Incremento



```
for (var index = 0; index < pokemons.length; index++) {  
  //código a ejecutar  
  console.log(pokemon[index]);  
}
```

The diagram illustrates the syntax of a for loop. Three colored boxes at the top are labeled 'Inicialización' (purple), 'Condición' (orange), and 'Incremento' (teal). Arrows from these boxes point to the corresponding parts of a code snippet in a dark-themed editor window. The code snippet is: `for (var index = 0; index < pokemons.length; index++) { //código a ejecutar console.log(pokemon[index]); }`. The initial part `var index = 0;` is underlined in purple, `index < pokemons.length;` is underlined in orange, and `index++` is underlined in teal.

Inicialización: De la variable que llevará el conteo de cuantas veces se iterará.

Condición: Mientras la condición se cumpla, se ejecutará el código dentro de las llaves {}.

Incremento: Se ejecuta después de cada iteración, normalmente se coloca un **contador** que incremente en 1 la variable de inicialización.

Acumulador

Se entiende por acumulador:

Una **variable que acumula el resultado de una operación.**

```
> var acumulador = 0;  
  for (var index = 0; index <= 4; index++) {  
    acumulador = acumulador + index;  
    console.log(acumulador);  
  }
```

0

1

3

6

10

Recorriendo Arreglos



```
1  for (let index = 0; index < array.length; index++) {  
2      console.log(array[index]);  
3  }
```

**continue
y break**

DEV.F
DESARROLLAMOS(PERSONAS);

dev

continue

Continue para la ejecución de la iteración actual pero continua con el resto de iteraciones

```
const array1 = [1, 2, 3, 4, 5]

for (let i = 0; i < array1.length; i++) {
  if( array1[i] === 3) {
    continue;
  }
  console.log(array1[i]);
}

// -> 1, 2, 4, 5
```

break

break para la ejecución del ciclo por completo



```
1  const array1 = [1, 2, 3, 4, 5]
2
3  for (let i = 0; i < array1.length; i++) {
4      if( array1[i] === 3) {
5          break;
6      }
7      console.log(array1[i]);
8  }
9  // -> 1, 2
```

For

VS

while

¿Cuándo usar *While* y cuándo *For*?

No existen reglas fijas, pero una buena recomendación para escoger entre ambas es el caso de si conozco o no el número de iteraciones que voy a realizar:

- Usamos el **ciclo for** para iterar un **arreglo**.
- Usamos el **ciclo for** cuando sabemos que el código a iterar debería ejecutarse **n veces**.
- Usamos el **ciclo while** para la variable que nos permite leer un archivo.
- Usamos el **ciclo while** para preguntar por entradas del usuario (user input).
- Usamos el **ciclo while** cuando el incremento de valor en iteración es algún valor no estándar.

También es importante mencionar que conforme adquiramos más habilidades podríamos usar estructuras de iteración más avanzadas diferentes a for y while.

Actividad

Ejercicio Recorriendo Arreglos

Challenge:

Vamos a seguir puliendo nuestra calculadora de propinas.

Recuerda que si la cuenta está entre los \$100 y \$800 la propina es del 15% y si está fuera de este rango la propina es del 20%.

1. Crea un arreglo **bills** que contenga los siguientes 10 datos: 22, 295, 176, 440, 37, 105, 10, 1100, 86 y 52
2. Crea un arreglo vacío para guardar los totales a pagar **totals**
3. Usa una función **calculateTip** para calcular el valor total a pagar (cuenta + propina) de cada uno de los valores en **bills** y agrégalo al arreglo **totals**.
4. Imprime en consola el arreglo **totals**

Actividad

Ejercicio Recorriendo Arreglos

Bonus:

Crea una función que tome como argumento un arreglo y calcule el promedio de los valores del arreglo