

# ESTRUCTURAS DE CONTROL



## Estructuras de control de flujo

---

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa, que por defecto en Python es secuencial.

Si queremos que determinadas instrucciones se ejecuten únicamente si se cumplen ciertos criterios o incluso ejecutar instrucciones un número repetido de veces entonces recurriremos a las estructuras de control. Distinguimos principalmente 2 tipos:

- Condicionales:
  - if, elif, else
- Iterativas:
  - Bucles
    - Determinado: for
    - Indeterminado: while

# ESTRUCTURAS DE CONTROL: CONDICIONALES



## Estructuras condicionales: if

Cuando necesitamos evaluar una condición y que un fragmento de código se ejecute o no según sea el resultado cierto o falso utilizamos las estructuras condicionales:

- **Si** se cumple una o varias condiciones **entonces** se ejecuta un código

```
# Estructura if
a = 33
b = 200
if b > a:
    print("b mayor que a") # Obligatorio hacer la indentación

print("continúa el programa")
```

b mayor que a  
continúa el programa

```
# Estructura if
a = 33
b = 33
if b > a:
    print("b mayor que a") # Obligatorio hacer la indentación

print("continúa el programa")
```

continúa el programa

## Estructuras condicionales: if else

Cuando necesitamos evaluar una condición y que un fragmento de código se ejecute o no según sea el resultado cierto o falso utilizamos las estructuras condicionales:

- **Si** se cumple una o varias condiciones **entonces** se ejecuta un código

```
# Estructura if-else
a = 33
b = 200
if b > a:
    print("b mayor que a")
else:
    print("b menor o igual que a")
```

↳ b mayor que a

```
# Estructura if-else
a = 33
b = 21
if b > a:
    print("b mayor que a")
else:
    print("b menor o igual que a")
```

↳ b menor o igual que a

## Estructuras condicionales: **if elif**

Cuando necesitamos evaluar una condición y que un fragmento de código se ejecute o no según sea el resultado cierto o falso utilizamos las estructuras condicionales:

- **Si** se cumple una o varias condiciones **entonces** se ejecuta un código

```
# Estructura elif
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

☞ a and b are equal

## Estructuras condicionales: **if elif else**

Esta estructura condicional equivale a la sintaxis switch case de otros lenguajes de programación.

Podemos añadir tanto elif como sea necesario para evaluar todas las condiciones que queramos.

```
# Estructura if-elif-else
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

↳ a is greater than b

## Estructuras condicionales: **if-else** en una línea

Los condicionales if else pueden ser expresados en una sola línea, con el fin de evaluar condiciones sencillas o ahorrar líneas de código.

```
# Abreviacion if en una línea
if a > b: print("a is greater than b")
```

a is greater than b

```
# Abreviación if-else en una línea
a = 500
b = 330
if a > b:
    print("A")
else:
    print("=")

# Equivalente en una línea
print("A") if a > b else print("B")
```

A  
A

```
# if-elif-else en una sola línea
a = 330
b = 330
if a > b:
    print("A")
elif a == b:
    print("=")
else:
    print("B")

# Equivalente en una línea
print("A") if a > b else print("=") if a == b else print("B")
```

=  
=

## Estructuras condicionales: **if-else** múltiples condiciones

Es muy común evaluar más de una condición a la vez, para ello utilizamos los operadores lógicos:

### **and, or, not**

Podemos combinarlos con todos los tipos de operadores: **aritméticos**, **comparación**, de **identidad**, **membresía**.

```
# Evaluar más de una condición a la vez,  
# usamos el operador and  
a = 200  
b = 33  
c = 500  
if a > b and c > a:  
    print("Both conditions are True")
```

Both conditions are True

```
# Evaluar más de una condición a la vez  
# usamos el operador or  
a = 200  
b = 33  
c = 500  
if a > b or a > c:  
    print("At least one of the conditions is True")
```

At least one of the conditions is True

## Estructuras condicionales: if else anidadas

Dentro de una estructura condicional **if-else** puede haber cualquier código, por ejemplo otros if-else o incluso bucles.

De esta forma se pueden anidar condicionales en **varios niveles**.

Como buena práctica se aconseja no establecer más de 3 niveles de anidamiento, debido a que aumenta la complejidad del código y dificulta la legibilidad.

En esos casos es mejor crear una función con bloques de código.

```
# if else anidados
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

▶ Above ten,  
and also above 20!

# ESTRUCTURAS DE CONTROL: REPETICIÓN

---

## Estructuras iterativas: **for**

La palabra reservada **for** nos permite crear un bucle determinado, en el que las instrucciones dentro del bucle se ejecutan un número determinado de veces que viene establecido en la definición del **for**.

Se usa comúnmente para recorrer cadenas de textos y estructuras de datos.

```
# Recorrer una cadena de texto
for x in "tesla":
    print(x)
```

```
t
e
s
l
a
```

```
# Recorrer una lista
cars = ["tesla", "rolls", "ferrari"]
for x in cars:
    print(x)
```

```
tesla
rolls
ferrari
```

## Estructuras iterativas: for y break

La palabra reservada **break** nos permite romper la ejecución de un bucle, de manera que cuando el flujo de ejecución llega a la instrucción break entonces sale del bucle y continúa su curso fuera del mismo.

```
# break - Romper la ejecución del bucle cuando se cumple una condición
cars = ["tesla", "rolls", "ferrari"]
for x in cars:
    print(x)
    if x == "rolls":
        break
```

```
tesla
rolls
```

```
# break - Romper la ejecución del bucle cuando se cumple una condición
cars = ["tesla", "rolls", "ferrari"]
for x in cars:
    if x == "rolls":
        break
    print(x)
```

```
tesla
```

```
# break - Romper la ejecución del bucle cuando se cumple una condición
cars = ["tesla", "rolls", "ferrari"]
for x in cars:
    if x == "bentley": # No se cumple nunca
        break
    print(x)
```

```
tesla
rolls
ferrari
```

## Estructuras iterativas: for y continue

La palabra reservada **continue** es diferente a **break**, en lugar de romper la ejecución del bucle lo que hará es saltar a la iteración siguiente.

De esta manera cuando el flujo de ejecución llega a la instrucción **continue** lo que hará es saltar automáticamente a la siguiente iteración en lugar de seguir ejecutando el código que hay después del **continue**.

```
# continue - con la palabra reservada continue forzamos  
# a que salte a la siguiente iteración de bucle  
cars = ["tesla", "rolls", "ferrari"]  
for x in cars:  
    if x == "rolls":  
        continue  
    print(x)
```

```
tesla  
ferrari
```

## Estructuras iterativas: **for** y **range()**

La función **range** nos devuelve una secuencia de números:

- Por defecto empieza en 0
- Por defecto avanza de 1 en 1

Admite hasta 3 argumentos:

*range (start, stop, step)*

```
# Para iterar en una secuencia de números  
# podemos utilizar la función range()  
for x in range(6): # 6 elementos, de 0 a 5  
    print(x)
```

```
0  
1  
2  
3  
4  
5
```

```
# El incremento en la secuencia de números con range()  
# por defecto es de 1 en 1, podemos modificarlo  
for x in range(2, 20, 3):  
    print(x)
```

```
2  
5  
8  
11  
14  
17
```

## Estructuras iterativas: **for** y **range()** y **len()**

La función **range** nos devuelve una secuencia de números:

- Por defecto empieza en 0
- Por defecto avanza de 1 en 1

Admite hasta 3 argumentos:

*range (start, stop, step)*

```
# Uso de range() con len() que nos da la longitud de una lista
a = ['lorem', 'ipsum', 'dolor', 'sit', 'amet']
for i in range(len(a)):
    print(i, a[i])
```

```
0 lorem
1 ipsum
2 dolor
3 sit
4 amet
```

## Estructuras iterativas: for y else

Con la instrucción **else** programamos un código que se ejecuta cuando la condición que se evalúa en el for deja de cumplirse.

De esta forma el **else** se ejecuta tanto si el for no se cumple ninguna vez como si se cumple un número determinado de veces y después deja de cumplirse.

```
# for-else con else especificamos un bloque de código  
# que debe ejecutarse al terminar el bucle  
for x in range(6):  
    print(x)  
else:  
    print("Bucle terminado")
```

```
0  
1  
2  
3  
4  
5  
Bucle terminado
```

```
# for-else sin llegar a ejecutar el for  
for x in range(0):  
    print(x)  
else:  
    print("Bucle terminado")
```

```
Bucle terminado
```

## Estructuras iterativas: **for** y **else** con **break**

En caso de que se ejecute un **break** dentro del bucle entonces el código que hay dentro del else no se ejecutará.

```
# for-else con break no ejecuta el else
for x in range(6):
    print(x)
    if x == 3:
        break
else:
    print("Bucle terminado")
```

0  
1  
2  
3

## Estructuras iterativas: bucles anidados

Es común trabajar con múltiples estructuras de datos al mismo tiempo para ello podemos usar **bucles anidados**.

De esta forma podemos trabajar con varias dimensiones, aplicando 2, 3, 4 bucles anidados al mismo tiempo.

Al igual que con otras estructuras de control que se anidan, no es recomendable establecer demasiados niveles de anidamiento ya que crea complejidad y dificulta la lectura del código.

```
# Bucles anidados|
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

## Estructuras iterativas: for y pass

Al igual que ocurre con las funciones, sin o implementamos el método, es decir, no escribimos el código se producirá un error a menos que utilicemos la palabra reservada **pass**.

```
# pass utilizamos la palabra reservada pass
# cuando nuestro bucle no tiene código
for x in [0, 1, 2]:
    pass

for x in [0, 1, 2]:
```

File "<ipython-input-18-e44a24a6ce58>", line 6  
for x in [0, 1, 2]:  
^  
SyntaxError: unexpected EOF while parsing

[SEARCH STACK OVERFLOW](#)

## Estructuras iterativas: **while**

El bucle **while** es un bucle indeterminado, se ejecuta siempre y cuando se cumpla la condición o condiciones que se evalúan.

```
[1] '''
while condition:
    # Sentencias de código
    Statement
    . . . . .
    Statement
    # Fin Sentencias de código
else:
    # (Opcional)
    # Se ejecuta solamente si la condición del while es False
    # Si se ha ejecutado un break en el while entonces el else no se ejecutará
'''
```

## Estructuras iterativas: **while**

Ejemplos de sintaxis con **while**.

En su forma más básica tiene una condición, y dentro las sentencias de código indentadas que se ejecutarán mientras se cumpla la condición.

Dentro del **while** se ejecutan instrucciones que modificarán variables hasta que finalmente la condición del **while** deje de ser **True**.

```
# sintaxis while  
i = 0  
while i < 6:  
    print(i)  
    i += 1
```

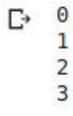
```
0  
1  
2  
3  
4  
5
```

## Estructuras iterativas: **while** con **break**

Ejemplos de sintaxis de cómo crear un bucle con **while**.

La palabra reservada **break** permite romper la ejecución de un bucle.

```
# uso de break
i = 0
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```



## Estructuras iterativas: **while** con **continue**

Ejemplos de sintaxis con **while**.

La palabra reservada **continue** permite pasar a la siguiente iteración, es decir, el programa no ejecuta las instrucciones que hay después del **continue**, si no que salta a la siguiente iteración.

```
# uso de continue
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
```

## Estructuras iterativas: **while** con **else**

Ejemplos de sintaxis con **while**.

El uso de **else** en los bucles es opcional, se ejecuta cuando la condición que está en el **while** se vuelve **False**.

```
# while else
i = 0
while i < 6:
    print(i)
    i += 1
else:
    print('i no es menor que 6 por tanto
no se cumple la condición')
```

```
0
1
2
3
4
5
i no es menor que 6 por tanto
no se cumple la condición
```

## Estructuras iterativas: **while** con **else** y **break**

Ejemplos de sintaxis con **while**.

El uso de **else** en los bucles es opcional, se ejecuta cuando la condición que está en el **while** se vuelve **False**.

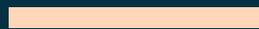
En caso de que **while** tenga un **break**, se rompe la ejecución del bucle y no se ejecuta el código que hay en **else**.

```
# while else con break
i = 0
while i < 6:
    i += 1
    print(i)
    if i == 3:
        break
else:
    print('Este else no se ejecuta porque
    se ha roto el bucle con break')
```

```
1
2
3
```



# SCOPE O ALCANCE



## Alcance de los datos

---

Tanto en las funciones como dentro de las estructuras de control condicionales e iterativas existe el concepto de **alcance**.

El alcance es la visibilidad que tienen las variables o estructuras de datos en función de la región de código en las que son creadas. Distinguimos dos alcances:

- Alcance **local**
- Alcance **global**

## Alcance local

Cuando creamos una variable o una estructura de datos dentro de una función o una estructura de control solamente existe dentro, pero no para el resto del programa.

```
def miFuncion():
    anotherNumber = 10
    print(anotherNumber)
miFuncion()
print(anotherNumber)
```

10

-----

**NameError** Traceback  
<ipython-input-2-c190b4a550c8> in <module>()  
3 print(anotherNumber)  
4 miFuncion()  
----> 5 print(anotherNumber)

**NameError:** name 'anotherNumber' is not defined

## Alcance global

---

Para que una variable que se crea dentro de una función o estructural de control sea visible y se pueda utilizar en el resto del programa utilizaremos la palabra reservada **global**.

```
def miFuncion():  
    global anotherNumber  
    anotherNumber = 10  
    print(anotherNumber)  
miFuncion()  
print(anotherNumber)
```

```
10  
10
```

## Alcance global

La palabra reservada **global** también nos permite modificar una variable global pero desde dentro de una función o estructura de control.

```
▶ anotherNumber = 150
def miFuncion():
    anotherNumber = 10
    print(anotherNumber)
miFuncion()
print(anotherNumber)
```

```
10
150
```

```
▶ anotherNumber = 150
def miFuncion():
    global anotherNumber
    anotherNumber = 10
    print(anotherNumber)
miFuncion()
print(anotherNumber)
```

```
▶ 10
10
```

# RESUMEN



## Resumen

---

Estructuras de control de flujo:

Condicionales:

- if
- if-elif
- if-elif-else
- if-else en una sola línea
- El operador ternario
- Múltiples condiciones: and, or

Repetitivas (bucles):

- Determinadas
  - for
    - for
    - for else
    - break y continue
- Indeterminadas
  - while
    - while
    - while else
    - break y continue