El ciclo for

Un ciclo for es un ciclo que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del ciclo y cada repetición se suele llamar iteración. La sintaxis de un ciclo for es la siguiente:

```
for variable in elemento_iterable (lista, cadena, range, etc.):
    cuerpo del ciclo
```

Ejemplo de ciclo for 1

En Python no es necesario definir la variable de control antes del ciclo, aunque se puede utilizar como variable de control una variable ya definida en el programa. El cuerpo del ciclo se ejecuta tantas veces como elementos tenga el elemento que se está recorriendo por ejemplo: elementos de una lista o de un range(), caracteres de una cadena, etc.). Por ejemplo:

print("Comienzo") for i in [0, 1, 2]: print("Hola ", end="") print() print("Final")	Comienzo Hola Hola Hola Final

Ejemplo de ciclo for 2

Si la lista está vacía, el ciclo no se ejecuta ninguna vez. Por ejemplo:

print("Comienzo") for i in []: print("Hola ", end="") print() print("Final")	Comienzo Final

Ejemplo de ciclo for 3

Si la variable de control no se va a utilizar en el cuerpo del ciclo, como en los ejemplos anteriores, se puede utilizar el guion (_) en vez de un nombre de variable. Esta notación no tiene ninguna consecuencia con respecto al funcionamiento del programa, pero sirve de ayuda a la persona que esté leyendo el código fuente, que así sabe que los valores no se van a utilizar. Por ejemplo:

print("Comienzo") for _ in [0, 1, 2]: print("Hola ", end="") print() print("Final")	Comienzo Hola Hola Hola Final

El indicador puede incluir cualquier número de guiones bajos (_, __, ___, ____, etc). Los más utilizados son uno o dos guiones (_ o __).

Ejemplo de ciclo for 4

En el ejemplo anterior, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza una variable. Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento que se recorre. Por ejemplo:

print("Comienzo") for i in [3, 4, 5]:	Comienzo Hola. Ahora i vale 3 y su cuadrado 9

print(f"Hola. Ahora i vale {i} y su cuadrado es {i ** 2}") print("Final")	Hola. Ahora i vale 4 y su cuadrado 16 Hola. Ahora i vale 5 y su cuadrado 25 Final

Ejemplo de ciclo for 5

La lista puede contener cualquier tipo de elementos, no sólo números. El ciclo se repetirá siempre tantas veces como elementos tenga la lista y la variable irá tomando los valores de uno en uno. Por ejemplo:

print("Comienzo") for i in ["Pepe", "Pepa", 47]: print(f"Hola. Ahora i vale {i}") print("Final")	Comienzo Hola. Ahora i vale Pepe Hola. Ahora i vale Pepa Hola. Ahora i vale 47 Final

Ejemplo de ciclo for 6

La costumbre más extendida es utilizar la letra *i* como nombre de la variable de control, pero se puede utilizar cualquier otro nombre válido. Por ejemplo:

print("Comienzo") for numero in [0, 1, 2, 3]: print(f"{numero} * {numero} = {numero ** 2}") print("Final")	Comienzo 0 * 0 = 0 1 * 1 = 1 2 * 2 = 4 3 * 3 = 9 Final

Ejemplo de ciclo for 7

La variable de control puede ser una variable empleada antes del ciclo. El valor que tuviese la variable no afecta a la ejecución del ciclo, pero cuando termina el ciclo, la variable de control conserva el último valor asignado:

i = 10 print(f"El ciclo no ha comenzado. Ahora i vale {i}") for i in [0, 1, 2, 3, 4]: print(f"{i} * {i} = {i ** 2}") print(f"El ciclo ha terminado. Ahora i vale {i}")	El ciclo no ha comenzado. Ahora i vale 10 0 * 0 = 0 1 * 1 = 1 2 * 2 = 4 3 * 3 = 9 4 * 4 = 16 El ciclo ha terminado. Ahora i vale 4

Ejemplo de ciclo for 8

Cuando se escriben 2 o más ciclos seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada ciclo establece los valores de la variable sin importar los valores anteriores:

for i in [0, 1, 2]:
print(f"{i} * {i} = {i ** 2}")
print()
for i in [0, 1, 2, 3]:
print(f"{i} * {i} * {i} = {i ** 3}")
	0 * 0 = 0
	1 * 1 = 1
	2 * 2 = 4
	
	0 * 0 * 0 = 0
	1 * 1 * 1 = 1
	2 * 2 * 2 = 8
	3 * 3 * 3 = 27

Ejemplo de ciclo for 9

En vez de una lista se puede escribir una cadena, en cuyo caso la variable de control va tomando como valor cada uno de los caracteres:

for i in "AMIGX": print(f"Dame una {i}") print("¡AMIGX!")	Dame una A Dame una M Dame una I Dame una G Dame una X ¡AMIGX!

Ejemplo de ciclo for 10

En los ejemplos anteriores se ha utilizado una lista para facilitar la comprensión del funcionamiento de los ciclos pero, si es posible hacerlo, se recomienda utilizar tipos range(), entre otros motivos porque durante la ejecución del programa ocupan menos memoria. El siguiente programa es equivalente al programa del ejemplo anterior:

print("Comienzo") for i in range(3): print("Hola ", end="") print() print("Final")	Comienzo Hola Hola Hola Final

Ejemplo de ciclo for 11

Otra de las ventajas de utilizar tipos range() es que el argumento del tipo range() controla el número de veces que se ejecuta el ciclo. En el ejemplo anterior era suficiente cambiar el argumento para que el programa salude muchas más veces.

print("Comienzo") for i in range(10): print("Hola ", end="") print() print("Final")	Comienzo Hola Hola Hola Hola Hola Hola Hola Hola Hola Hola Final

Esto permite que el número de iteraciones dependa del desarrollo del programa.

Ejemplo de ciclo for 12

En el ejemplo siguiente es el usuario quien decide cuántas veces se ejecuta el ciclo:

veces = int(input("¿Cuántas veces quiere que le salude? ")) for i in range(veces): print("Hola ", end="") print() print("Adiós")	¿Cuántas veces quiere que le salude? 6 Hola Hola Hola Hola Hola Hola Adiós

Contadores, variables testigo (o bandera) y acumuladores

En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que indiquen si simplemente ha ocurrido algo (testigo) 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. El ejemplo siguiente es un ejemplo de programa con contador (en este caso, la variable que hace de contador es la variable cuenta):

Ejemplo de contador

print("Comienzo")
cuenta = 0
for i in range(1, 6):
  if i % 2 == 0:
  cuenta = cuenta + 1
print(f"Desde 1 hasta 5 hay {cuenta} múltiplos de 2")
	Comienzo
	Desde 1 hasta 5 hay 2 múltiplos de 2

Importante:

- En cada iteración, el programa comprueba si *i* es múltiplo de 2.
- El contador se modifica sólo si la variable de control *i* es múltiplo de 2.
- El contador va aumentando de uno en uno.
- Antes del ciclo se debe inicializar el contador (en este caso, 0)

Variables testigo o bandera

Se entiende por testigo una variable booleana que indica simplemente si una condición se ha cumplido o no. Es un caso particular de contador, pero se suele hacer con variables lógicas en vez de numéricas (en este caso, la variable que hace de testigo es la variable *encontrado*):

Ejemplo de testigo

print("Comienzo")
encontrado = False
for i in range(1, 6):
  if i % 2 == 0:
  encontrado = True
if encontrado:
  print(f"Entre 1 y 5 hay al menos un múltiplo de 2.")
else:
  print(f"Entre 1 y 5 no hay ningún múltiplo de 2.")
	Comienzo
	Entre 1 y 5 hay al menos un múltiplo de 2.

Importante:

- En cada iteración, el programa comprueba si *i* es múltiplo de 2.
- El testigo se modifica la primera vez que la variable de control *i* es múltiplo de 2.
- El testigo no cambia una vez ha cambiado.
- Antes del ciclo se debe dar un valor inicial al testigo (en este caso, False)

Acumulador

Se entiende por acumulador una variable que acumula el resultado de una operación. El ejemplo siguiente es un ejemplo de programa con acumulador (en este caso, la variable que hace de acumulador es la variable *suma*):

Ejemplo de acumulador

print("Comienzo") suma = 0 for i in [1, 2, 3, 4]:	Comienzo La suma de los números de 1 a 4 es 10

suma = suma + i print(f"La suma de los números de 1 a 4 es {suma}")	

Ciclos anidados

Se habla de ciclos anidados cuando un ciclo se encuentra en el bloque de instrucciones de otro bloque. Al ciclo que se encuentra dentro del otro se le puede denominar ciclo interior o ciclo interno. El otro ciclo sería el ciclo exterior o ciclo externo. Los ciclos pueden tener cualquier nivel de anidamiento (un ciclo dentro de otro ciclo dentro de un tercero, etc.).

Ciclos anidados (variables independientes)

Se dice que las variables de los ciclos son **independientes** cuando los valores que toma la variable de control del ciclo interno **no** dependen del valor de la variable de control del ciclo externo. Por ejemplo:

Ejemplo de ciclo anidado (variables independientes)

for i in [0, 1, 2]:
  for j in [0, 1]:
  print(f"i vale {i} y j vale {j}")
	i vale 0 y j vale 0
	i vale 0 y j vale 1
	i vale 1 y j vale 0
	i vale 1 y j vale 1
	i vale 2 y j vale 0
	i vale 2 y j vale 1

En el ejemplo anterior, el ciclo externo (el controlado por i) se ejecuta 3 veces y el ciclo interno (el controlado por j) se ejecuta 2 veces por cada valor de i. Por ello la instrucción print() se ejecuta en total 6 veces (3 veces que se ejecuta el ciclo externo x 2 veces que se ejecuta cada vez el ciclo interno = 6 veces). El siguiente programa es equivalente al ejemplo anterior:

for i in range(3):
  for j in range(2):
  print(f"i vale {i} y j vale {j}")
	i vale 0 y j vale 0
	i vale 0 y j vale 1
	i vale 1 y j vale 0
	i vale 1 y j vale 1
	i vale 2 y j vale 0
	i vale 2 y j vale 1

Al escribir ciclos anidados, hay que prestar atención a la indentación de las instrucciones, ya que indica a Python si una instrucción forma parte de un bloque u otro. En los tres siguientes programas la única diferencia es el sangrado de la última instrucción:

Ejemplo de indentación en ciclo anidado (1)

for i in [1, 2, 3]:
  for j in [11, 12]:
  print(j, end=" ")
  print(i, end=" ")
	11 1 12 1 11 2 12 2 11 3 12 3

En este caso, la última instrucción forma parte del cuerpo del ciclo interno. Por tanto el valor de i se escribe cada vez que se ejecuta el ciclo interno.

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Ejemplo de indentación en ciclo anidado (2)

for i in [1, 2, 3]:
  for j in [11, 12]:
  print(j, end=" ")
  print(i, end=" ")
	11 12 1 11 12 2 11 12 3

En este caso, la última instrucción forma parte del cuerpo del ciclo externo, pero no del interno. Por tanto el valor de i se escribe cada vez que se ha terminado de ejecutar el ciclo interno.

Ejemplo de indentación en ciclo anidado (3)

for i in [1, 2, 3]:
  for j in [11, 12]:
  print(j, end=" ")
print(i, end=" ")
	11 12 11 12 11 12 3

En este caso, la última instrucción no forma parte de ningún ciclo. Por tanto el valor de i se escribe una sola vez, al terminarse de ejecutar el ciclo externo.

La costumbre más extendida es utilizar la letra "i" como nombre de la variable de control del ciclo externo y la letra "j" como nombre de la variable de control del ciclo interno (o "k" si hay un tercer nivel de anidamiento), pero se puede utilizar cualquier otro nombre válido.

Ejemplo de indentación en ciclo anidado (4)

En Python se puede incluso utilizar la misma variable en los dos ciclos anidados porque Python las trata como si fueran dos variables distintas. Por ejemplo:

for i in range(3):
  print(f"i (externa) vale {i}")
  for i in range(2):
  print(f"i (interna) vale {i}")
	i (externa) vale 0
	i (interna) vale 0
	i (interna) vale 1
	i (externa) vale 1
	i (interna) vale 0
	i (interna) vale 1
	i (externa) vale 2
	i (interna) vale 0
	i (interna) vale 1

De todas formas, este uso no se recomienda porque da lugar a programas difíciles de leer y además no es habitual en otros lenguajes de programación. Se aconseja utilizar siempre nombres de variables distintos.

Ciclos anidados (variables dependientes)

Se dice que las variables de los ciclos son **dependientes** cuando los valores que toma la variable de control del ciclo interno dependen del valor de la variable de control del ciclo externo. Por ejemplo:

Ejemplo de ciclo anidado (variables dependientes) 1

for i in [1, 2, 3]:
  for j in range(i):
  print(f"i vale {i} y j vale {j}")
	i vale 1 y j vale 0
	i vale 2 y j vale 0
	i vale 2 y j vale 1
	i vale 3 y j vale 0
	i vale 3 y j vale 1
	i vale 3 y j vale 2

En el ejemplo anterior, el ciclo externo (el controlado por i) se ejecuta 3 veces y el ciclo interno (el controlado por j) se ejecuta 1, 2 y 3 veces. Por ello la instrucción print() se ejecuta en total 6 veces.

La variable i toma los valores de 1 a 3 y la variable j toma los valores de 0 a i, por lo que cada vez el ciclo interno se ejecuta un número diferente de veces:

- Cuando i vale 1, range(i) devuelve la lista [0] y por tanto el ciclo interno se ejecuta una sola vez y el programa escribe una sola línea en la que i vale 1 (y j vale 0).

- Cuando i vale 2, range(i) devuelve la lista [0, 1] y por tanto el ciclo interno se ejecuta dos veces y el programa escribe dos líneas en la que i vale 2 (y j vale 0 o 1 en cada una de ellas).
- Cuando i vale 3, range(i) devuelve la lista [0, 1, 2] y por tanto el ciclo interno se ejecuta tres veces y el programa escribe tres líneas en la que i vale 3 (y j vale 0, 1 o 2 en cada una de ellas).

El ciclo while

Un ciclo while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True). La sintaxis del ciclo while es la siguiente:

```
while condicion:
   cuerpo del ciclo
```

La ejecución de esta estructura de control while es la siguiente:

- Python evalúa la condición:
    - si el resultado es True se ejecuta el cuerpo del ciclo. Una vez ejecutado el cuerpo del ciclo, se repite el proceso (se evalúa de nuevo la condición y, si es verdadera, se ejecuta de nuevo el cuerpo del ciclo) una y otra vez mientras la condición sea verdadera.
    - si el resultado es False, el cuerpo del ciclo no se ejecuta y continúa la ejecución del resto del programa.

La variable o las variables que aparezcan en la condición se llaman variables de control. Las variables de control deben definirse antes del ciclo while y modificarse en el ciclo while.

Ejemplos de ciclos while

Por ejemplo, el siguiente programa escribe los números del 1 al 3:

Ejemplo de ciclo while 1

i = 1
while i <= 3:
  print(i)
  i += 1
print("Programa terminado")
	1
	2
	3
	Programa terminado

Ejemplo de ciclo while 2

El ejemplo anterior se podría haber programado con un ciclo for. La ventaja de un ciclo while es que la variable de control se puede modificar con mayor flexibilidad, como en el ejemplo siguiente:

i = 1
while i <= 50:
  print(i)
  i = 3 * i + 1
print("Programa terminado")
	1
	4
	13
	40
	Programa terminado

Ejemplo de ciclo while 3

Otra ventaja del ciclo while es que el número de iteraciones no está definida antes de empezar el ciclo, por ejemplo porque los datos los proporciona el usuario. El ejemplo siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:

numero = int(input("Escriba un número positivo: "))
while numero < 0:
  print("¡Ha escrito un número negativo! Inténtelo de nuevo")
  numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")
	Escriba un número positivo: -4
	¡Ha escrito un número negativo! Inténtelo de nuevo
	Escriba un número positivo: -8
	¡Ha escrito un número negativo! Inténtelo de nuevo
	Escriba un número positivo: 9
	Gracias por su colaboración

Ciclos infinitos

Si la condición del ciclo se cumple siempre, el ciclo no terminará nunca de ejecutarse y tendremos lo que se denomina un **ciclo infinito**, normalmente se deben a errores que se deben corregir. Los ciclos infinitos no intencionados deben evitarse pues significan perder el control del programa. Para interrumpir un ciclo infinito, hay que pulsar la combinación de teclas **Ctrl+C**. Al interrumpir un programa se mostrará un mensaje de error similar a éste:

```
Traceback (most recent call last):
  File "ejemplo.py", line 3, in <module>
    print(i)
KeyboardInterrupt
```

Estos algunos ejemplos de ciclos infinitos:

El programador ha olvidado modificar la variable de control dentro del ciclo y el programa imprimirá números 1 indefinidamente:		El programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:		Se aconseja expresar las condiciones como desigualdades en vez de comparar valores. En el ejemplo siguiente, el programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:	

i = 1
while i <= 10:
  print(i, end=" ")
	1 1 1 1 1 1 1 1 ...	i = 1
		while i > 0:
		  print(i, end=" ")
		  i += 1
			1 2 3 4 5 6 7 8 9 10 11 ...	i = 1
				while i != 100:
				  print(i, end=" ")
				  i += 2
					1 3 5 7 9 11 ... 97 99 101 ...