Estructuras de control

Python soporta las sentencias de control de flujo que podemos encontrar en otros lenguajes de programación.

Instrucciones if, elif, else

La instrucción condicional **if** se utiliza para comprobar una condición.

- Si la condición es verdadera entonces procesaremos un bloque de instrucciones (llamado bloque if).
- Si la condicicón no se cumple, se ejecuta el bloque de instrucciones alternativas (llamado bloque else).
- Cada uno de los bloques están sangrados, sin necesidad de escribir llaves o sentencias del tipo `begin ... end`.
- La cláusula else es opcional.

```
if <condition>:
    <instrucciones del bloque if>
else:
    <instrucciones del bloque else>
```

Recordar que en Python los bloques se delimitan por sangrado.

- Cuando ponemos los dos puntos al final de la primera línea del condicional, todo lo que vaya a continuación con **un** nivel de sangrado superior se considera dentro del condicional.
- En cuanto escribimos la primera línea con un nivel de sangrado inferior, hemos cerrado el condicional.

```
x, y = 2 , 0

if x > y:
    print("x es mayor que y")
    print("x sigue siendo mayor que y")
else:
    pass       # no se hace nada
```

```
x es mayor que y
x sigue siendo mayor que y
```

La instrucción **pass** no hace nada. Se puede usar cuando la sintaxis obliga a escribir al menos una instrucción pero el programa no necesita ninguna acción.

```
if 1 < 0:                             # esta condición siempre es falsa
    print("1 es menor que 0")
print("Esto se ejecuta siempre")  # esto no está dentro del bloque y no se ejecuta
                                                    # dentro del if
```

```
Esto se ejecuta siempre
```

```
if 1 < 0:
    print("1 es menor que 0")
     print("1 sigue siendo menor que 0")    # error de sangrado
```

```
  File "<ipython-input-3-f2e366661c08>", line 3
    print("1 sigue siendo menor que 0")    # error de sangrado
    ^
IndentationError: unexpected indent
```

Si queremos añadir ramas adicionales al condicional, podemos emplear la instrucción **elif** (abreviatura de *else if*). Para la parte final, que debe ejecutarse si ninguna de las condiciones anteriores se ha cumplido, usamos la instrucción **else**. Veamos algunos ejemplos:

```
if x < 0:
    x = 0
    print('Negativo cambiado a cero')
elif x == 0:
    print('Cero')
elif x == 1:
    print('Uno')
else:
    print('Más de uno')
```

```
Más de uno
```

```
x, y = 2 , 0
if x > y:
    print("x es mayor que y")
else:
    print("x es menor que y")
```

```
x es mayor que y
```

```
x, y = 2 , 0
if x < y:
    print("x es menor que y")
elif x == y:
    print("x es igual a y")
else:
    print("x no es ni menor ni igual que y")
```

```
x no es ni menor ni igual que y
```

Expresiones ternarias

Las expresiones ternarias en Python tienen la siguiente forma:

```
e = valorSiTrue if <condicion> else valorSiFalse
```

Permite definir la instrucción **if-else** en una sola línea. La expresión anterior es equivalente a:

```
if <condicion>:
    e = valorSiTrue
else:
    e = valorSiFalse
```

Bucles for y while

El bucle `for`

La instrucción **for** de Python itera sobre los elementos de cualquier secuencia (una tupla, una lista o una cadena de texto), en el orden que aparecen en la secuencia.

Se utiliza para recorrer una colección completa de elementos, es decir pasa a través de cada elemento.

```
for <element> in <iterable_object>:
    <hacer algo...>
```

- Aquí el objeto `<iterable_object>` puede ser una lista, tupla, array, etc.
- El bucle se repite un número fijo de veces, que es la longitud de la colección de elementos.

```
for elemento in (1,2,3,4,5):   # itera soble los elementos de la tupla
    print(elemento)
```

```
1
2
3
4
5
```

```
suma = 0
for elemento in (1,2,3,4,5):   # Suma todos los elementos de la tupla
    print(suma)
    suma = suma + elemento
print("El bucle for ha terminado")
```

```
0
1
3
6
10
El bucle for ha terminado
```

El bucle anterior ejecuta el cuerpo del bucle 5 veces. En este caso decimos que se han realizado 5 **iteraciones**.

```
dias = ["Lunes", "Martes", "Miércoles", "Jueves" ]
for nombre in dias:
    print(nombre)
```

```
Lunes
Martes
Miércoles
Jueves
```

En el siguiente ejemplo mostramos cómo un objeto iterable como `range(3)` es susceptible de ser recorrido mediante una instrucción **for**. En este sentido podemos decir que la instrucción **for** es un iterador.

```
for i in range(3):    # range(3) crea la lista de enteros en el intervalo [0,3)
    print(i)
```

```
0
1
2
```

La instrucción **continue** permite saltar de una iteración a otra.

```
 #range(10): crea la lista de enteros en el intervalo [0,10)
for j in range(10):
    if j % 2 == 0:
        continue        # si el valor de j es par, saltamos a la siguiente iteración
    print(j)            # no se ejecuta si j es un número par
```

```
1
3
5
7
9
```

También es posible recorrernos una tupla de elementos:

```
puntos = ((0,1), (1,2), (1,3))
for (x,y) in puntos: # desempaquetado
    print( x + y )

# x se refiere a la primera componente de cada una de las tuplas
# y se refiere a la segunda componente de cada una de las tuplas
```

```
1
3
4
```

Un diccionario también es un objeto iterable en Python:

```
dic =  {1:'Lunes', 2:'Martes', 3:'Miércoles' }

for i in dic.items():
    print(i)
```

```
(1, 'Lunes')
(2, 'Martes')
(3, 'Miércoles')
```

En el caso de los diccionarios podemos iterar también sobre las claves y sobre los valores.

```
# Python 3.5
for i in dic.keys():
    print(i)
```

```
1
2
3
```

```
# Python 3.6
for i in dic.values():
    print(i)
```

```
Lunes
Martes
Miércoles
```

El Bucle `while`

La instrucción **while** permite ejecutar repetidamente un bloque de código mientras la condición asociada al while sea verdadera.

```
while <condition>:
    <cuerpo del bucle>
```

- El cuerpo del bucle está sangrado y no se necesitan instrucciones de tipo `end`.
- El número de iteraciones es variable, depende de la condición.

En el siguiente ejemplo, el cuerpo del bucle está compuesto por dos instrucciones. El cuerpo del bucle se ejecutará mientras la condición asociada al while, `i < 5` , sea cierta.

```
i = -2
while i < 5:
    print(i)          # cuerpo
    i = i + 1         # cuerpo
print("Estoy fuera del while")
```

```
-2
-1
0
1
2
3
4
Estoy fuera del while
```

Como regla y para evitar bucles infinitos, debemos asegurarnos de que en cada iteración, la condición del while está más cerca de hacerse cierta. En el ejemplo anterior, en el cuerpo del bucle se incrementa la variable `i`, por lo que cada vez el valor de `i` se aproxima más a 5.

En Python tmbién existe la instrucción que nos permite interrumpir el bucle antes de que la condición se haga falsa. Se trata de la instrucción **break**:

```
i = 0
while i < 5:
    print(i)
    i = i + 1
    if i == 3:
        break       # salimos del bucle cuando el valor de i es 3
```

```
0
1
2
```

La función `enumerate`

Cuando trabajamos con secuencias de elementos puede resultar útil conocer el índice de cada elemento. La función **enumerate** devuelve una secuencia de tuplas de la forma **(i, valor)**.

Mediante un bucle es posible recorrerse dicha secuencia:

```
ciudades = ["Madrid", "Sevilla", "Segovia", "Valencia" ]

for (i, valor) in enumerate(ciudades):
    print('%d: %s' % (i+1, valor))
```

```
1: Madrid
2: Sevilla
3: Segovia
4: Valencia
```

```
# uso de la función reversed
# la función reversed devuelve un iterador inverso de una lista

for (i, valor) in enumerate(reversed(ciudades)):
    print('%d: %s' % (len(ciudades)-i, valor))
```

```
4: Valencia
3: Segovia
2: Sevilla
1: Madrid
```