Estructuras de control en python

Las estructuras de control nos permiten alterar el flujo de nuestro programa. Esto significa, que a lo largo del mismo, iremos planteando algunas condiciones, que van a definir qué parte del código va a ejecutarse a continuación, cuál se omitirá, y cuál se va a repetir (y cuántas veces/hasta qué momento)

Las estructuras de control pueden dividirse en 2 grandes grupos: condicionales e iterativos

- Los primeros condicionan la ejecución del bloque de código según el valor de verdad (booleano) de una expresión.
- Los últimos nos marcan la repetición de un proceso un número definido de veces o dependiendo del estado booleano de una expresión.

Pero antes debemos recordar un tema fundamental en Python: **la indentación**

Python utiliza la indentación para delimitar una estructura, permitiendo establecer bloques de código. No existen comandos para finalizar las líneas, ni llaves con las que delimitar el código. Los únicos delimitadores existentes son los dos puntos (:) y la indentación del código.

La indentación va a separar nuestros bloques de código para saber dónde comienza y dónde termina cada proceso.

Estructuras de control condicionales

Son las que definen la ejecución de un bloque de código según el valor de verdad de una expresión

.1 Condicional if

Se utiliza para tomar decisiones. Evalúa si una expresión es True o False y ejecuta el bloque de código si el resultado es True.

Se obtiene False si se encuentra:

- un número igual a cero (0, 0.0, 0 + 0j)
- un contenedor vacío (lista, tupla, conjunto, diccionario)
- False o None

```
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
```

.2 else

Permite agregar un bloque de código que se ejecutará si la condición a evaluar resulta False.

```
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
else:
    print('El número ingresado es mayor o igual a 10')
```

.3 elif

Es una combinación de else + if. Es decir, lo usamos si queremos evaluar otra condición si no se cumple el primer if. Podemos utilizar tantos elif como sean necesarios.

```
nro = int(input('Ingrese un número: '))
if nro < 10:
    print('El número ingresado es menor a 10')
elif nro == 10:
    print('El número ingresado es igual a 10')
else:
    print('El número ingresado es mayor a 10')
```

Estructuras de control iterativas I

Son las que permiten ejecutar un bloque de código múltiples veces (ciclos)

.1 while

Realiza múltiples iteraciones basándose en el resultado de una expresión lógica que puede tener como resultado True o False

.1.a) while controlado por conteo

En este caso, el bucle while está controlado por la variable `nro`, que se va incrementando con cada ejecución.

Cuando esta variable llega a valer 10, la condición del bucle pasa a ser False y termina el ciclo.

```
suma = 0
nro = 0
while nro <=10:
    suma = nro + suma
    nro += 1
print('La suma da ' + str(suma))
```

.1.b) while controlado por evento

El bucle se interrumpe cuando el usuario ingresa -1.

```
contador = 0
total = 0
nota = 99
while nota != -1:
    nota = int(input('Ingrese la nota del alumno (-1 para salir): '))
    if nota != -1:
        total += nota
        contador += 1
promedio = total / contador
print('El promedio de notas es de', promedio)
```

.1.c) while con else

Una manera alternativa más legible para el código anterior se logra combinando while con else.

```python
contador = 0
total = 0
nota = 99
while nota != -1:
    nota = int(input('Ingrese la nota del alumno (-1 para salir): '))
    if nota != -1:
        total += nota
        contador += 1
else:
    promedio = total / contador
    print('El promedio de notas es de', promedio)
print('-- FIN DEL PROGRAMA --')
```

Estructuras de control iterativas II

.2 for

.2.a) for controlado por la cantidad de elementos

En este caso, el código se va a repetir para cada uno de los elementos de la lista numeros.

```
numeros = [2, 4, 6, 23, 19, 36]
for n in numeros:
    if n % 2 == 0:
        print(n, 'es par')
    else:
        print(n, 'es impar')
```

.2.b) for y la clase range

range es un tipo de dato que simula una secuencia numérica. Para construirlo necesitamos de 1 a 3 parámetros:

- range(max) crea un rango de número desde 0 hasta max-1
- range(min, max) crea un rango de número desde min hasta max-1
- range(min, max, step) crea un rango de número desde min hasta max-1, cuyos valores se van incrementando de step en step

Por ejemplo, el siguiente código va a imprimir los números pares desde el 0 hasta el 10.

```
for n in range(0,11,2):
    print(n)
```

Sentencias utilitarias

.1 break

Se utiliza para interrumpir un bucle while o for.

En el siguiente ejemplo, el programa interrumpe el ciclo cuando alcanza un valor ingresado por el usuario:

```
variable = 11
parar = int(input('Dónde me detengo? [1-9]]'))
while variable > 0:
    print('Valor actual', variable)
    variable -= 1
    if variable == parar:
        break
```

También podemos interrumpir un ciclo for. En el siguiente ejemplo, el ciclo se interrumpe al encontrar un número mayor a 10 entre la lista de elementos:

```
lista = [2, 4, 5, 7, 9, 1, 3, 1000, 6, 8]
for e in lista:
    if e > 10:
        print('Nro demasiado grande')
        break
    print(e)
```

.2 continue

La sentencia **continue** dirige el programa hacia un nuevo inicio de bucle while o for, ignorando todo lo que hay debajo, aunque no se haya terminado de ejecutar todo el proceso del ciclo.

Podemos crear un código que ignore los números pares e imprima los impares hasta el 21:

```
variable = 0
while variable < 21:
    variable += 1
    if variable % 2 == 0:
        continue
    print(variable, 'es impar')
```

En el siguiente ejemplo, se imprimen todos los números de la lista, excepto el 9.

```
lista = [2, 4, 5, 7, 9, 1, 3, 1000, 6, 8]
for e in lista:
    if e == 9:
        print('Ese número no me gusta')
        continue
    print(e)
```