prog_datasci_3_apython

September 17, 2022

1 Fundamentos de programación

1.1 Unidad 3: Estructuras de control y funciones en Python

---

2 Instrucciones de uso

Este documento es un *notebook* interactivo que intercala explicaciones más bien teóricas de conceptos de programación con fragmentos de código ejecutables. Para aprovechar las ventajas que aporta este formato, os recomendamos que, en primer lugar, leáis las explicaciones y el código que os proporcionamos. De este modo, tendréis un primer contacto con los conceptos que exponemos. Ahora bien, **la lectura es solo el principio!** Una vez hayáis leído el contenido, no os olvidéis de ejecutar el código proporcionado y modificarlo para crear variantes que os permitan comprobar que habéis entendido la funcionalidad y explorar los detalles de implementación. En último lugar, os recomendamos también consultar la documentación enlazada para explorar con más profundidad las funcionalidades de los módulos presentados.

Para guardar posibles modificaciones que hagáis sobre este notebook, os aconsejamos que montéis la unidad de Drive en Google Colaboratory (colab). Tenéis que ejecutar las instrucciones siguientes:

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
%cd /content/drive/MyDrive/Colab_Notebooks/prog_datasci_3
```

3 Introducción

Esta unidad sigue presentando conceptos básicos de programación en Python. En concreto, presentaremos las operaciones lógicas y veremos cómo se puede alterar el flujo de ejecución de los programas con estructuras iterativas (*for* y *while*), condicionales (*if*) y funciones. Adicionalmente, explicaremos cómo podemos interactuar con ficheros desde Python, y cómo podemos importar otros módulos para incorporar funcionalidades adicionales en nuestros programas.

A continuación, se incluye la tabla de contenidos. Para navegar por el documento usad `Table of contents` (parte superior izquierda) de Google colab.

4 1 £Qué es una estructura de control?

Los programas en Python ejecutan las instrucciones secuencialmente. Cada instrucción se ejecuta de manera ordenada y en el mismo orden que están escritas, como hemos visto en la unidad 2. En el ejemplo siguiente se puede apreciar cómo se ejecutan las instrucciones secuencialmente.

[1]:
```
x = 3
y = x + 7
print("El valor de x es {}".format(x))
print("El valor de y es {}".format(y))
```

```
El valor de x es 3
El valor de y es 10
```

Primero asignamos la variable x igual a 3 y la variable y como la suma de x más el valor numérico 7. A continuación, mostramos por pantalla los valores de x y y. Cada instrucción se ejecuta una detrás de la otra. El flujo de ejecución de un conjunto de instrucciones puede ser modificado mediante las estructuras de control. Las principales estructuras de control son las estructuras de control condicionales (**if-elif-else**) y las estructuras de control iterativas (**for y while**).

5 2 Condicional

La estructura de control condicional es aquella estructura de control que permite implementar acciones según la evaluación de una condición simple, ya sea falsa o verdadera.

5.1 2.1 If

La instrucción if nos permite ejecutar un bloque de código si se cumple una determinada condición.

[2]:
```
a = 10
if a >= 5:
    print('a es mayor o igual a 5')
```

```
a es mayor o igual a 5
```

El objetivo del código anterior es evaluar si la variable *a* es mayor o igual a 5. Si la variable *a* es mayor o igual a 5, se cumple la condición de `if`, de forma que se imprime por pantalla la instrucción **a es mayor o igual a 5**. En cambio, en la celda siguiente *a* es menor que 5. Por lo tanto, la condición de `if` no se cumple, y el bloque de código dentro de `if` no se ejecuta.

[2]:
```
a = 2
if a >= 5:
    print('a es mayor o igual a 5')
```

En los dos ejemplos anteriores podemos apreciar la estructura inherente de `if`. Esta estructura tiene dos partes muy diferenciadas:

- **Condición** que se tiene que cumplir para que el bloque de código se ejecute, en el ejemplo anterior `a >= 5`.
- **Bloque de código** que se tiene que ejecutar si se cumple la condición anterior. Es importante remarcar que el bloque de código siempre debe contener una instrucción, si no es así se genera un error de sintaxis `SyntarError`.

La instrucción `if` debe terminarse con `:` y el bloque de código debe estar tabulado. Toda sentencia insertada después de `if` y correctamente tabulada, forma parte del bloque de código que se ejecutará si la condición se cumple. En el ejemplo siguiente, la instrucción `print('Fuera if')` se está ejecutando siempre porque no forma parte del bloque de código de `if`.

[3]:
```
a = 2
if a >= 5:
    print('a es mayor o igual a 5')
print('Fuera if')
```

```
Fuera if
```

5.2 2.2 if ... else

Si la condición no se cumple, se puede ejecutar un segundo bloque de código especificado dentro de la instrucción `else`.

[4]:
```
a = 9
if a % 2 == 0:
    print('a es par')
else:
    print('a no es par')
```

```
a no es par
```

La cláusula `else` especifica qué hay que hacer si la condición no se cumple. En este ejemplo, la primera condición no se cumple porque a no es par, por lo tanto, se ejecuta la instrucción que hay dentro de `else`.

5.3 2.3 If... elif... else

Si queremos ejecutar condiciones adicionales podemos usar la instrucción `elif`.

[5]:
```
a = 9
if a % 2 == 0:
    print('a es par')
elif a % 3 == 0:
    print('a es multiple de 3')
```

```
a es multiple de 3
```

En el ejemplo anterior, si se cumple la primera condición, se imprime por pantalla **a es par**. Si se cumple la segunda condición, se imprime por pantalla **a es múltiple de 3**. También podemos incluir cláusulas `else` dentro de la estructura`if-elif` si no se cumple ninguna de las condiciones anteriores.

[6]:
```
a = 7
if a % 2 == 0:
    print('a es par')
elif a % 3 == 0:
    print('a es múltiple de 3')
else:
    print('a no es par ni múltiple 3')
```

```
a no es par ni múltiple 3
```

Veamos unos ejemplos más para ver diferentes estructuras condicionales. Os aconsejamos que variéis los valores de las variables de código de los ejemplos siguientes e id ejecutando el código para comprobar cómo se comporta en cada situación.

[7]:
```
# Ejemplo 1
a = 5
b = 5
if a > b:
    print('a es mayor que b')
elif a < b:
    print('a es menor que b')
```

```
else:
    print('a es igual a b')
```

```
a es igual a b
```

[8]:
```
# Ejemplo 2
a = -1
b = -2
if a > b:
    if b >= 0:
        print('a es mayor que b y positivo')
    elif a*-1 > 0:
        print('a es mayor que b y negativo')
    else:
        print('a es mayor que b y positivo')
else:
    print('b es mayor o igual que a')
```

```
a es mayor que b y negativo
```

6 3 Iteración

Las estructuras de control iterativas permiten ejecutar un mismo bloque de código tantas veces como sea necesario. En la mayoría de los lenguajes de programación, hay dos formas de iterar una secuencia: mediante `for` o mediante `while`.

6.1 3.1 For

La instrucción for nos permite crear bucles sobre un número de iteraciones definido inicialmente.

[10]:
```
monsters = ['Kraken', 'Leviathan', 'Uroborus', 'Hydra']

# Primer método iterando mediante for:
for monster in monsters:
    print(monster)
```

```
Kraken
Leviathan
Uroborus
Hydra
```

En el ejemplo anterior, estamos recorriendo cada elemento de la lista `monsters` e imprimiéndolo por pantalla. La estructura de iteración `for` sigue la estructura siguiente:

[28]:
```
# for <variable> in <iterable>:
#<codi>
```

donde la `variable` toma cada uno de los valores del iterable. `iterable` son aquellos objetos que pueden ser iterados o que pueden ser indexados. Algunos ejemplos de iterables son las

`listas`, `tuples`, `cadenas` o `diccionarios`. Es importante remarcar que toda sentencia insertada después del `for` tiene que estar correctamente tabulada para que se ejecute. Como podéis ver en la celda siguiente, el `print` no está tabulado y crea un error `IdentationError`.

[9]:
```
for monster in monsters:
print(monster)
```

```
  File "<ipython-input-9-baf8f65b9206>", line 2
    print(monster)
    ^
IndentationError: expected an indented block
```

En el ejemplo siguiente, veremos diferentes tipos de iterables.

[10]:
```
# Definimos una lista, una tupla, un diccionario y una cadena de caracteres
planetas_lista = ['Mercurio', 'Venus', 'Tierra']
planetas_tupla = ('Mercurio', 'Venus', 'Tierra')
planetas_dict = {"Mercurio": 4880, "Venus": 12105, "Tierra": 12745}
planetas_str = 'Mercurio'

# Recorremos las estructuras con un *for* y mostramos su contenido
print("**Lista**")
for i in planetas_lista:
    print(i, end=" ")
print()

print("**Tupla**")
for i in planetas_tupla:
    print(i, end=" ")
print()

print("**dict**")
for i in planetas_dict:
    print(i, end=" ")
print()

print("**cadena**")
for i in planetas_str:
    print(i, end=" ")
```

```
**Lista**
Mercurio Venus Tierra
**Tupla**
Mercurio Venus Tierra
**dict**
Mercurio Venus Tierra
**cadena**
```