02 Introducción a Python

Instrucciones de uso

En el módulo anterior hemos introducido conceptos básicos sobre variables y su uso en Python. En este módulo estudiaremos conceptos más avanzados como las instrucciones de flujo de ejecución (*for*, *while*, *if*), cómo definir y utilizar funciones, cómo leer y escribir archivos y cómo organizar el código.

Iteración y operaciones lógicas

En la mayoría de casos tendremos que manipular nuestros datos, y para hacerlo utilizaremos los conceptos de iteración y operaciones lógicas. Las operaciones lógicas nos permiten comparar valores entre variables (mayor, menor, igualdad), y la iteración, visitar uno a uno los elementos de una lista, tupla, diccionario o cualquier estructura de datos que sea susceptible de ser secuenciada.

```
# Las operaciones lógicas tendrán como resultado un valor verdadero
(True) o falso (False):

a = 5
b = 1

# ¿El valor de a es mayor que b?
print(a > b)
```

```
True
```

```
# El valor de a es menor que b?
print(a < b)
```

```
False
```

```
b = 5
# El valor de b es igual que el de a?
print(b == a)
```

```
True
```

```
# Otros operadores lógicos disponibles son menor o igual '<=', mayor o
igual '>=',
# o la negación 'not'
print(a <= b)
print(a >= b)

a = False
print(not a)
```

```
True
True
True
```

También podemos alterar el flujo de ejecución de nuestro programa utilizando las estructuras *if... else* o *if... elif... else*. Veamos algunos ejemplos:

```
a = 5
b = 6

if a > b:
    print('a es mayor que b')
else:
    print('a es menor o igual que b')
```

```
a es menor o igual que b
```

```
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
```

En Python hay solo dos maneras de iterar una secuencia: mediante ***for*** o mediante ***while***. La primera de las opciones, *for*, iterará uno por uno los elementos contenidos en una lista. En el caso de *while*, iteraremos mientras la condición de permanencia en el bucle se cumpla. Veamos algunos ejemplos:

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

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

print()

# Segundo método. La función especial 'enumerate' nos retorna una
tupla en la que el primer elemento es un
# índice que comienza en 0 y aumenta de 1 en 1, y el segundo elemento
es el valor de la posición en la lista:
for i, monster in enumerate(monsters):
    print(i, monster)
```

```
Kraken
Leviathan
Uroborus
Hydra

0 Kraken
1 Leviathan
2 Uroborus
3 Hydra
```

```
# También podríamos iterar la lista usando un bucle while, pero es una forma mucho menos idiomática en Python
# y preferiremos siempre la opción de for:
i = 0
# Mientras que el índice 'i' sea menor que la longitud de la lista 'monsters':
while i < len(monsters):
    # Imprime el valor de la lista en la posición 'i'.
    print(i, monsters[i])
    # No olvidemos actualizar el valor de 'i' sumándole 1 o tendremos un bucle infinito.
    i += 1
```

```
0 Kraken
1 Leviathan
2 Uroborus
3 Hydra
```

En este momento seríamos capaces de calcular la serie de Fibonacci hasta un determinado valor:

```
# Calculamos el valor de la serie hasta un valor n = 100.
n = 100

a, b = 0, 1
while a < n:
    print(a, end=" ")
    a, b = b, a + b
```

```
0 1 1 2 3 5 8 13 21 34 55 89
```

En Python disponemos de una función muy útil para generar una secuencia de números, que podemos utilizar de diferentes maneras:

```
# La función 'range' nos retorna un objeto iterable que genera una secuencia de números.
# Para convertirlo en una lista, usamos la función 'list':
print(list(range(10)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Observad que `range` no devuelve directamente una lista, sino que devuelve un tipo propio del mismo nombre. Por este motivo, para obtener una lista es necesario hacer una conversión de tipo utilizando `list()`. Esto es una novedad en Python 3, ya que en Python 2 la función `range` devolvía una lista.

```
# Visualizamos el tipo de retorno de 'range'
print(type(range(10)))  # Muestra el tipo de objeto 'range'

# Convertimos 'range(10)' en una lista y mostramos el tipo de objeto resultante
print(type(list(range(10))))  # Muestra el tipo de objeto 'list'
```

```
<class 'range'>
<class 'list'>
```

Veamos algunos de los usos más habituales de `range`:

```
# Podemos utilizar 'range' para iterar:
for i in range(10):
    print(i, end=" ")

print()
```

```
0 1 2 3 4 5 6 7 8 9
```

```
# Podemos definir el rango de acción.

# Por ejemplo, especificando solo el final como hicimos antes:
for i in range(10):
    print(i, end=" ")

print()

# Especificando inicio y fin:
for i in range(5, 10):
    print(i, end=" ")

print()

# O especificando también el salto entre cada valor:
for i in range(5, 10, 3):
    print(i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
5 6 7 8 9
5 8
```

Siempre que no necesitemos explícitamente una lista, usaremos directamente el tipo `range` (sin hacer la conversión a lista), ya que esto es generalmente más eficiente (ahorra memoria). Solo cuando necesitemos una lista (por ejemplo, para visualizar el resultado como hicimos en el primer ejemplo) convertiremos el resultado de `range` a lista.

```
# También es posible iterar sobre un diccionario:
country_codes = {34: 'Spain', 376: 'Andorra', 41: 'Switzerland', 424:
None}

# Por clave:
for country_code in country_codes.keys():
    print(country_code)
print()

# Por valor:
for country in country_codes.values():
    print(country)
print()

# Por ambos al mismo tiempo:
for country_code, country in country_codes.items():
    print(country_code, country)
```

```
34
376
41
424

Spain
Andorra
Switzerland
None

34 Spain
376 Andorra
41 Switzerland
424 None
```

Funciones

Otra manera muy importante de organizar el flujo de ejecución es encapsulando una cierta porción de código en una función reutilizable. Una función en Python utiliza el mismo concepto que una función matemática. Por ejemplo, imaginemos la función matemática:

suma(x, y) = x + y

En Python podemos definir la misma función de la siguiente manera:

```
# La función 'suma' se define mediante la palabra especial 'def' y
tiene dos argumentos: 'x' y 'y':
def suma(x, y):
    # Devolvemos el valor de la suma
    return x + y

# En este momento, podemos llamarla con cualquier valor:
```

```
print(suma(2, 4))    # Imprime la suma de 2 y 4, que es 6
print(suma(5, -5))   # Imprime la suma de 5 y -5, que es 0
print(suma(3.5, 2.5)) # Imprime la suma de 3.5 y 2.5, que es 6.0
```

```
6
0
6.0
```

```
# Podemos definir una función que no haga nada utilizando la palabra especial 'pass':

def dummy():
    pass

dummy()

# Podríamos redefinir el fragmento de código de la secuencia de Fibonacci como una función:

def fibonacci(n=100):
    a, b = 0, 1
    while a < n:
        print(a, end=" ")
        a, b = b, a + b

# Llamamos a la función con el valor 10 para imprimir la secuencia de Fibonacci hasta 10
fibonacci(10)
```

```
0 1 1 2 3 5 8
```

```
# En el ejemplo anterior, hemos definido que el argumento 'n' tenga un valor por defecto. Esto es muy útil
# en los casos en los que usamos la función siempre con un mismo valor, queremos dejar constancia de un
# caso de ejemplo o por defecto. En este caso, podemos ejecutar la función sin pasarle ningún valor:
fibonacci()
```

```
0 1 1 2 3 5 8 13 21 34 55 89
```

```
# Podemos definir algunos argumentos con valores por defecto y otros sin.
# Los argumentos sin valor por defecto siempre deben estar a la izquierda en la definición de la función:

def potencia(a, b=2):
    # Por defecto, elevaremos al cuadrado.
    return a**b
```

```
print(potencia(3))   # Imprime 3 elevado a 2, que es 9
print(potencia(2, 3)) # Imprime 2 elevado a 3, que es 8
```

```
9
8
```

Recomendamos la lectura de la documentación oficial para consolidar conocimientos.

Leer y escribir desde archivos

Una tarea habitual es leer líneas de un archivo o escribir líneas en un archivo. En Python, leer y escribir archivos se hace con la biblioteca *os*. Una biblioteca es un conjunto de código que tiene sentido agrupar para ser utilizado por otras personas. En nuestro caso, *os* es una biblioteca que agrupa funciones relacionadas con el sistema operativo (***o**perating **s**ystem*). Para cargar una biblioteca, utilizaremos la palabra reservada ***import***. Más adelante explicaremos algunas particularidades de utilizar e importar bibliotecas. A continuación, os explicamos cómo escribir y leer un archivo:

```
import os

# Abrimos un archivo llamado 'a_file.txt' para escritura (de ahí la
'w', 'writing').
# Lo asignamos a un objeto llamado 'out' para gestionar el archivo.
out = open('a_file.txt', 'w')

# Escribiremos 10 líneas, cada una con un número del 0 al 9.
for i in range(10):
    # La línea siguiente escribe en el archivo todo lo que pongamos
dentro de 'out.write()'.
    # En nuestro caso, es una cadena del tipo 'Línea 0\n', 'Línea 1\
n', etc. Esto lo conseguimos
    # utilizando los placeholders %d y %s.
    # %s representa una cadena de caracteres y %d un número entero.
    # Concatenamos en este caso un número con una cadena que es el
salto de línea,
    # os.linesep, que es equivalente a '\n' en sistemas Linux.
    out.write("Línea %d%s" % (i, os.linesep))

# Cerramos el archivo para asegurarnos de que todos los datos se
escriben correctamente.
out.close()

# Ahora leeremos el archivo que acabamos de escribir de tres maneras
diferentes:

# Primer método
f = open('a_file.txt')
for line in f:
    print(line, end="")  # Imprime cada línea sin añadir un salto de
```

```
línea extra
f.close()
print()

# Segundo método
f = open('a_file.txt')
lines = f.readlines()  # Lee todas las líneas del archivo en una lista
for line in lines:
    print(line, end="")  # Imprime cada línea sin añadir un salto de
línea extra
f.close()
print()

# Tercer método
with open('a_file.txt') as f:  # Usa 'with' para manejar
automáticamente el cierre del archivo
    for line in f:
        print(line, end="")  # Imprime cada línea sin añadir un salto
de línea extra
```

```
Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
Línea 8
Línea 9

Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
Línea 8
Línea 9

Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
```

```
Línea 8
Línea 9
```

Organización del código

Un módulo de Python es cualquier archivo con extensión *.py* que esté bajo la ruta del *path* de Python. El path de Python se puede consultar importando la biblioteca `sys`:

```
import sys
print(sys.path)
```

```
['/usr/lib/python310.zip', '/usr/lib/python3.10',
'/usr/lib/python3.10/lib-dynload', '',
'/home/ubuntu/.local/lib/python3.10/site-packages',
'/usr/local/lib/python3.10/dist-packages', '/usr/lib/python3/dist-
packages']
```

Por defecto, Python también busca las bibliotecas que se hayan definido en la variable de entorno `$PYTHONPATH` (esto puede variar ligeramente en un entorno Windows).

Un paquete en Python es cualquier directorio que contenga un archivo especial llamado *__init__.py* (este archivo estará vacío la mayoría de las veces).

Un módulo puede contener diferentes funciones, variables u objetos. Por ejemplo, definamos un módulo llamado *prog_datasci.py* que contenga:

```
# prog_datasci.py

PI = 3.14159265

def suma(x, y):
    return x + y

def resta(x, y):
    return x - y
```

Para utilizar desde otro módulo o *script* estas funciones, deberíamos escribir lo siguiente:

```
from prog_datasci import PI, suma, resta

# Luego podemos utilizar estas funciones y constantes normalmente.
rset = suma(2, 5)
print("Resultado de la suma:", rset)

# También podemos usar la constante PI y la función resta:
resultado_resta = resta(7, 3)
print("Resultado de la resta:", resultado_resta)
print("Valor de PI:", PI)
```

```
Resultado de la suma: 7
Resultado de la resta: 4
Valor de PI: 3.14159265
```

En Python también podemos usar la directiva `from prog_datasci import *`, pero su uso está **totalmente desaconsejado**. La razón es que estaríamos importando una gran cantidad de código que no utilizaremos (con el consiguiente aumento del uso de memoria), y además podríamos tener colisiones de nombres (funciones que se llamen igual en diferentes módulos) sin nuestro conocimiento. A menos que sea imprescindible, no utilizaremos esta directiva e importaremos una a una las bibliotecas y funciones que necesitaremos.

Pueden aprender más sobre cómo importar bibliotecas y definir sus propios módulos aquí.