Python: una introducción básica

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística
Universidad Politécnica de Cartagena

@kessler_mathieu

Los principios básicos

1. Los ficheros de programas Python tiene extensión `.py`
2. Para introducir un comentario, usad "#"
3. Al ejecutar un fichero `.py` con `python myfile.py`, todas sus líneas son ejecutadas secuencialmente, desde la primera hasta la última.
4. La indentación es una característica esencial de Python. Indica niveles jerarquizados de código.
5. No es necesario declarar las variables como en C o Java etc...
6. El tipo de datos de una variable es dinámico (es deducido de la primera aparición de la variable en el código)
7. Se puede consultar el tipo de una variable `my_variable` con `type(my_variable)`

Las estructuras de datos básicas que vamos a usar

- Entero (integer) `int`
- Flotante (Float) `float`

Qué es un número flotante?

"Float" es una abreviatura para "floating point..Es un tipo de datos fundamental presente en el compilador que se usa para definir valores numéricos con puntos decimales. Ej: 2.1562

- Booleanos (Boolean) `bool`

Qué es un valor Booleano?

Una variable Booleana puede tomar dos valores: `True` o `False`. Se utiliza para expresar condiciones que permiten desencadenar acciones.

- Cadenas (Strings) `str`

Qué es una cadena?

- Una cadena es una colección de caracteres. En Python, se enmarcan entre comillas simple, doble o triple: `'Hello!'`, `"Hello!"`, `'''Hello!'''`.
- Puedes imprimir una cadena a la consola con `print`: `print('Hello!')`
- Cadenas que abarquen múltiples líneas se pueden construir con comillas triples:

```
print('''Esta cadena abarca
                    varias líneas''')
```

Las estructuras de datos básicas que vamos a usar

- Lista (list) `list`

Qué es una lista en Python?

Una lista es una colección de elementos. Se escribe como una serie de valores separados por comas y enmarcados por corchetes
Los elementos pueden tener tipos diferentes.
Ejemplos:

```
mi_lista = [3.21, 4.87, -9.22]
mi_lista = ['stats', 23, True]
```

En una lista, las posiciones de los elementos empiezan en 0, y para acceder a un elemento, se usan corchetes.

```
>>> mi_lista = ['stats', 12, True]
>>> print(mi_lista[0])
stats
```

- Tupla (Tuple) `tuple`

Qué es un tuple en Python?

Un tuple es una colección de elementos. Se escribe como una serie de valores separados por comas y enmarcados entre paréntesis.
Los elementos pueden tener tipos diferentes.
Examples:

```
mi_tupla = (3.21, 4.87, -9.22)
mi_tupla = ('stats', 23, True)
```

La diferencia con una lista es que un tupla es "imutable", lo que quiere decir que no se puede cambiar un elemento dentro de un tupla.
Se accede a los elementos de un tupla de igual manera que para una lista.

Las estructuras de datos básicas que vamos a usar

- Conjunto (Set) `set`

Qué es un conjunto en Python?

Un conjunto es una colección de elementos que no tiene ningún elemento duplicado. Se escribe como una serie de elementos separados por comas y enmarcados entre llaves.
Los elementos pueden tener tipos diferentes. Se usan los conjuntos para llevar a cabo operaciones como unión, intersección, diferencia, etc. Ejemplos:

```
frutas = {'manzana',  'limón', 'banana'}
verduras = {'tomate', 'zanahoria', 'berenjena'}
```

- Diccionarios (Dictionaries) `dict`

Qué es un diccionario en Python?

Un diccionario es una colección de elementos indexados. Sus elementos son de la forma clave:valor. Se pueden acceder a un elemento de un diccionario usándo su clave. Un diccionario se enmarca entre llaves. Sus elementos pueden ser de distintos tipos: Ejemplos:

```
edades = {'john': 32,  'jane': 28, 'jim': 48}
```

Puede acceder al valor de un elemento indexando el diccionario por la clave del elemento entre corchetes:

```
>>> edades = {'john': 32,  'jane': 28, 'jim': 48}
>>> print(edades['jane'])
28
```

Entrada y salida básica via la consola

- `print` es la instrucción básica para imprimir objetos en la consola.
  Se puede imprimir cadenas:

  ```
  >>> print('Hello world!')
  Hello world!
  ```

  pero también otros objetos directamente

  ```
  >>> edades = {'john': 32,  'jane': 28, 'jim': 48}
  >>> print(edades)
  {'john': 32, 'jane': 28, 'jim': 48}
  ```

- Para obtener valores o datos introducidos por el usuario en la consola, la instrucción es `input`.

  ```
  >>> n = input('Número máximo de casos:')
  ```

  Hay que tener en cuenta que la variable resultado del comando `input` es de tipo cadena (str). Es posible que tenga que transformarlo en entero o en float para su uso posterior:

  ```
  >>> print(int(n) + 6)
  ```

Bucles en Python

El comando for

El comando `for` en Python permite iterar sobre los elementos de un objeto y ejecutar a cada iteración una series de instrucciones.

- Las instrucciones que se deben ejecutar en cada iteración están situadas en un bloque indentado debajo del comando `for`:

```
for ciudad in ['Madrid', 'Murcia', 'Cartagena']:
    print(ciudad)
```

Otro ejemplo:

```
suma_final = 0
for i in (0, 1, 2, 3, 4, 5, 6):
    j = i * 2
    suma_final += j
print('El resultado final is: ' + str(suma_final))
```

Notes

- Para expresar la secuencia 0, 1, 2, 3, 4, 5, 6, podemos usar `range(6)`.
- El operador `+=` indica el incremento propio, i.e `suma_final += j` es exactamente lo mismo que `suma_final = suma_final + j`
- El operador `+` para dos cadenas lleva a cabo la concatenación. Para usarlo, hemos tenido que convertir primero `suma_final` en una cadena.

Ejercicios

1. Escribe un programa en un fichero `ejercicios_introduccion.py` que pide un entero `n` al usuario y imprime en consola la suma de los primeros `n` términos de la secuencia 4, $-4/3$, $4/5$, $-4/7\ldots$, i.e. la secuencia

$$4 \times \frac{(-1)^i}{2 \cdot i + 1}, \; i = 0, 1, 2, \ldots$$

   - *Indicación 1: El operador de potencia en Python es `**`.*
   - *Indicación 2: Que no se os olvide de usar `int(n)` para transformar el input a un entero.*

   Ejecutad vuestro programa con valores de n grandes.

2. Añadid a vuestro programa anterior las instrucciones para devolver la tabla de multiplicación de n, que ha introducido el usuario.

3. Añadid a vuestro programa las instrucciones para imprimir el siguiente patrón:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Las instrucciones `if`, `if ... else`, `if ... else ... elif`

La instrucción `if` en Python permite desencadenar acciones (instrucciones) si se cumple una condición

- La estructura de la instrucción `if` es similar al del bucle `for`, las instrucciones que deben ser ejecutadas si la condición se cumple se recojen en un bloque indentado debajo de la parte `if`

```
import math
n_string = input('Introduzca un número entero ')
n = int(n_string)
if (n > 0):
    print('La raiz cuadrada de  ' + str(n) + ' es ' + str(math.sqrt(n)))
```

Note

La instrucción `import math` importa el módulo `math`, que contiene cientos de funciones matemáticas. Una vez el módulo importado, todas sus funciones están disponibles en nuestro programa. Para invocar una función de un módulo, necesitamos añadirle el prefijo del módulo en el que está definido (en este caso `math`). Se verán más aspectos de los módulos más adelante.

Ejercicios

1. Completad vuestro programa en `ejercicios_introduccion.py` con las instrucciones para comprobar si n introducido por el usuario es un número primo.

Las funciones permiten juntar una serie de instrucciones y llamarlas facilemente con un único comando (el nombre de la función)

- `def` es el comando que se usa para definir una función.
- Como siempre, la serie de instrucciones que deben ser ejecutadas cuando se invoca la función se situan en un bloque intentado debajo de `def`.a
- La función admite argumentos.
- La instrucción `return` se usa para devolver un valor (cualquier objeto de Python)

Example:

```
def calculate_discount(price, reduction):
    return price * (1 - reduction)
```

Una vez definida se puede invocar la función en un programa o en la consola: `calculate_discount(121, 0.2)` devolverá el precio rebajado.

Usad funciones!

El usar funciones es una práctica muy recomendable, mejora la legibilidad de vuestro código y su mantenimiento es más sencillo.

Recordad la función `calculate_discount`:

```
def calculate_discount(price, reduction):
    return price * (1 - reduction)
```

- Cuando invocamos la función con `calculate_discount(121, 0.2)`, Python utiliza la posición de los valores proporcionados para asignarlos a los parámetros de la función the arguments.
  - 121 está asignado al parámetro `price`
  - 0.2 está asignado al parámetro `discount`
- Se pueden usar también los nombres de los parámetros cuando invocamos la función. En este caso su orden es irrelevante.

  ```
  calculate_discount(reduction=0.2, price=121)
  ```
- Debemos proporcionar el número correcto de valores cuando invocamos una función, excepto si se indicaron valores por defecto en la definición de la función:

  ```
  def calculate_discount(price, reduction=0.2):
      return price * (1 - reduction)
  ```

  En este caso, si se invoca la función con un único valor, éste se asignará al parámetro `price` y se aplicará un descuento de 20 %.

Pensad en vosotros mismos y los posibles usuarios: documentad vuestro código!

- Python usa Docstrings para proporcionar información sobre una función. Una docstring es una cadena que está situada como primera instrucción de una función (módulo, clase o método).
- Dcstrings están enmarcadas por triples comillas:
  `"""Aquí viene el texto de información"""`
- Para casos sencillos, se puede usar una docstring de una línea

```
def calculate_discount(price, reduction):
    """Devuelve el precio reducido"""
    return price * (1 - reduction)
```

- Usar docstrings permite al usuario obtener información vía `help`

```
...
>>> help(calculate_discount)
Help on function calculate_discount in module __main__:

calculate_discount(price, reduction)
    Devuelve el precio reducido
```

Ejercicios

1. Añadid a vuestro programa en `ejercicios_introduccion.py` la definición de una función que tenga el entero `n` como argumento y que devuelva `True` si `n` es un número primo y `False` en otro caso.
2. Añadid al programa las instrucciones que permitan imprimir todos los números primos inferiores a `n`, introducido por el usuario.

Qué es un módulo?

Los módulos son ficheros con extensión `.py` que contienen definiciones de funciones y objetos (variables, clases, etc...).

- Si quiero usar las funciones de un módulo, debo importarlo. Por ejemplo, supongamos que he creado un fichero `aux_files.py` que contiene todas mis funciones útiles, lo importaré con

```
import aux_files
```

- Una vez que `aux_file` está importado, si quiero invocar una de sus funciones, por ejemplo `calculate_discount` tendré que añadir como prefijo el nombre del módulo

```
import aux_files
discounted_price = aux_files.calculate_discount(121, 0.2)
```

- Para simplicar el código, puedo definir un alias para un módulo cuando lo importo:

```
import aux_files as af
discounted_price = af.calculate_discount(121, 0.2)
```

Usando módulos

- Es posible importar sólo una o algunas funciones de un módulo determinado:

```
from aux_files import calculate_discount
```

- En este caso, no es necesario añadir el prefijo del módulo a la hora de invocar la función

```
from aux_files import calculate_discount
discounted_price = calculate_discount(121, 0.2)
```

- En algunos sitios podéis ver

```
from aux_files import *
```

Esto permitiría usar cualquier función del módulo `aux_files` sin necesidad de añadir el prefijo del módulo. No es una buena práctica! Es recomendado dejar constancia explícitamente de qué módulo proviene la función que se usa.

Usando módulos

- Python tiene una *"Librería estándar" ("Standard Library")* de módulos útiles que vienen instalados con la distribución de Python. Además, existen miles de módulos compartidos por usuarios, preparados como *"paquetes" ("packages")* preparados para su descarga e instalación.
- Módulos de la *"Librería estándar* pueden ser importados directamente en mi código, usando la instrucción `import`. Por ejemplo, el módulo `math` mencionado anteriormente.

```
import math
x = math.sin(0)
```

- Para usar módulos publicados por otros usuarios, es necesario descargar los paquetes asociados de los repositorios mantenidos por la comunidad.
- Uno de los repositorios muy importantes es `PyPI` https://pypi.org/, the Python Package Index, un enorme repositorio de software desarrollado y compartido por la comunidad Python. Para descargar e instalar paquetes de `PyPI`, se usa la utilidad `pip`.
- Puesto que, en este curso, usamos la distribución Anaconda, no usaremos `pip` y `PyPI`, sino la Anaconda Cloud y la instrucción `conda install`.

Usando módulos externos

- Usaremos de manera intensa algunos de los módulos importantes para el cálculo científico, concretamente, `numpy`, `scipy`, `matplotlib` and `pandas`.
- Si estás utilizando la distribución Anaconda completa, estos módulos ya están disponibles, basta con importarlos.
- Si estás utilizando la distribución miniconda, tendrá que preparar y usar un entorno virtual (ver el documento: "Creating a virtual environment with conda"). En este caso los paquetes se obtendrán y descargarán del repositorio Anaconda Cloud e se instalarán en vuestro entorno virtual.

Importando `numpy`

Como ejemplo, vamos a importar `numpy` usando su alias convencional

```
import numpy as np
x = np.arange(10)
```

Hemos usado la función `arange` del módulo numpy, que crea un vector de números separados con el mismo incremento, en este caso enteros que van de 0 a 9.

Funciones, métodos y atributos

Módulos definen clases, que son como plantillas para determinados tipos de objetos en nuestro programa. Para una clase determinada, se definen, de antemano, atríbutos y métodos. La manipulación objetos de una clase es mucho más sencilla gracias a los métodos y atributos de esta clase.

- Un ejemplo de clase en `numpy` es su vector *N*-dimensional, denominado `ndarray`.
- En el ejemplo anterior,

  ```
  import numpy as np
  x = np.arange(10)
  ```

  `x` es un `ndarray`. Lo podéis comprobar con `type(x)`.
- Existen muchos métodos y atributos predefinidos para los objetos de la clase `ndarray`. Aplicamos métodos y atributos a un objeto de una clase, al añadirlos como sufijo al nombre del objeto después de un punto.
  - `x.shape` es un atributo de `x`.
  - `x.mean()` aplica el método `mean` a `x`
- Un método es como una función pero siempre se aplica a un objeto. Puede aceptar argumentos, que se indican entre paréntesis.

Funciones, métodos y atributos

- El método `mean` sólo se puede aplicar a un vector numpy. El código siguiente da lugar a un error:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9].mean()
```

- Para obtener todos los métodos y atributos disponbles para un objeto, se puede usar `dir`:

```
import numpy as np
x = np.arange(10)
print(dir(x))
```

```
['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__', '__array_function__', '__array_interface__', '__array_pr
epare__', '__array_priority__', '__array_struct__', '__array_ufunc__', '__array_wrap__', '__bool__', '__class__', '__complex__', '
__contains__', '__copy__', '__deepcopy__', '__delattr__', '__delitem__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__'
, '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__iand__', '__iflo
ordiv__', '__ilshift__', '__imatmul__', '__imod__', '__imul__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert
__', '__ior__', '__ipow__', '__irshift__', '__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__', '__len__', '__lshift__',
'__lt__', '__matmul__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand_
_', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmatmul__', '__rmod__', '__rmul__
', '__ror__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem__', '__se
tstate__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__xor__', 'all', 'any', 'argmax', 'argmin', 'ar
gpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumpro
d', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item',
 'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp', 'put', 'rav
el', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', '
std', 'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']
```

Los métodos que empiezan y acaban con un doble "_" se llaman "dunder" (double underscore) methods y son para uso interno del módulo.