

Introducción a Python

Temas Avanzados de Fluidos

Federico Cerisola

2C 2017

Porqué usar Python?

- ▶ Es **fácil** de aprender

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible
 - ▶ Mucha **información** disponible online fácilmente

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible
 - ▶ Mucha **información** disponible online fácilmente
- ▶ Una muy buena **comunidad online** tanto para uso general como científico

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible
 - ▶ Mucha **información** disponible online fácilmente
- ▶ Una muy buena **comunidad online** tanto para uso general como científico
- ▶ Permite escribir programas bastante **rápidamente** y relativamente **claro**

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible
 - ▶ Mucha **información** disponible online fácilmente
- ▶ Una muy buena **comunidad online** tanto para uso general como científico
- ▶ Permite escribir programas bastante **rápidamente** y relativamente **claro**
- ▶ Buen soporte **multiplataforma**

Porqué usar Python?

- ▶ Es **fácil** de aprender
- ▶ Es un lenguaje muy **popular** para uso genérico
- ▶ Muy buen soporte **científico**
 - ▶ Gran cantidad de **librerías** disponible
 - ▶ Mucha **información** disponible online fácilmente
- ▶ Una muy buena **comunidad online** tanto para uso general como científico
- ▶ Permite escribir programas bastante **rápidamente** y relativamente **claro**
- ▶ Buen soporte **multiplataforma**
- ▶ Licencia **libre**

El intérprete de Python

Modo interactivo de ejecutar código.

Útil para probar rápidamente pequeñas porciones de código

El intérprete de Python

Modo interactivo de ejecutar código.

Útil para probar rápidamente pequeñas porciones de código

```
$ python3
```

```
>>>
```

```
# Intérprete de Python
```

El intérprete de Python

Modo interactivo de ejecutar código.

Útil para probar rápidamente pequeñas porciones de código

```
$ python3
```

```
>>> # Intérprete de Python
```

```
>>> 1 + 1
```

```
2
```

El intérprete de Python

Modo interactivo de ejecutar código.

Útil para probar rápidamente pequeñas porciones de código

```
$ python3
```

```
>>> # Intérprete de Python
```

```
>>> 1 + 1
```

```
2
```

```
>>> 2 * 3
```

```
6
```

Variables

Queremos crear alguna variable que contenga un numero, una cadena de text, etc.

Variables

Queremos crear alguna variable que contenga un numero, una cadena de text, etc.

Creamos variable con valor entero:

Variables

Queremos crear alguna variable que contenga un numero, una cadena de text, etc.

Creamos variable con valor entero:

```
>>> a = 2  
>>>
```

Variables

Queremos crear alguna variable que contenga un numero, una cadena de text, etc.

Creamos variable con valor entero:

```
>>> a = 2
>>>
```

Cómo imprimir en pantalla el valor?

Variables

Queremos crear alguna variable que contenga un numero, una cadena de text, etc.

Creamos variable con valor entero:

```
>>> a = 2
>>>
```

Cómo imprimir en pantalla el valor?

```
>>> print(a)
2
```

Tipos numérico y operaciones básicas

Una variable puede tener distintos **tipos**, por ejemplo: enteros, puntos flotantes (i.e. valores *reales*), texto, otros ...

Tipos numérico y operaciones básicas

Una variable puede tener distintos **tipos**, por ejemplo: enteros, puntos flotantes (i.e. valores *reales*), texto, otros ...

```
>>> a = 2
```

```
>>> b = 1.3
```

Tipos numérico y operaciones básicas

Una variable puede tener distintos **tipos**, por ejemplo: enteros, puntos flotantes (i.e. valores *reales*), texto, otros ...

```
>>> a = 2
>>> b = 1.3
```

Operaciones básicas sobre tipos numéricos

Tipos numérico y operaciones básicas

Una variable puede tener distintos **tipos**, por ejemplo: enteros, puntos flotantes (i.e. valores *reales*), texto, otros ...

```
>>> a = 2
>>> b = 1.3
```

Operaciones básicas sobre tipos numéricos

```
>>> a + 1
3
>>> a - 1
1
>>> a * 2
6
>>> b / 2
0.65
>>> a**3
8
>>> a**3 % 5
3
```

Tipos numérico y operaciones básicas

Atención: en Python 2 la división es entera

```
>>> 2 / 4
0
>>> 2.0 / 4.0
0.5
```

En Python 3 la división convierte a punto flotante

```
>>> 2 / 4
0.5
```


Otros tipos útiles

Texto:

```
>>> s = 'Hola'
```

```
>>> print(s)
```

```
Hola
```

```
>>> t = '!'
```

```
>>> s + t
```

```
Hola!
```

Otros tipos útiles

Listas:

```
>>> l = [ 1, 2, 'a' ]
>>> print(l)
[1, 2, 'a']
>>> l[0]
1
>>> l[2]
a
>>> l[-1]
a
>>> len(l)
3
>>> l + [ 'b', 2.0 ]
[1, 2, 'a', 'b', 2.0]
```

Otros tipos útiles

Tuplas, Diccionarios, ...

Otros tipos útiles

Tuplas, Diccionarios, ...

Observación: a diferencia con otros lenguajes, al inicializar las variables nunca tenemos que declarar de qué tipo son (i.e. si va contener un entero o un float o una cadena de texto, etc.).

Cómo crear un programa

Hasta ahora estuvimos ejecutando las cosas en el intérprete. Cómo armamos un programa?

Cómo crear un programa

Hasta ahora estuvimos ejecutando las cosas en el intérprete. Cómo armamos un programa?

Simplemente escribimos la secuencia de expresiones en un archivo **.py**

```
# file: prueba1.py  
a = 3  
b = 7  
c = a * b + b**2  
  
print(c)
```

Cómo crear un programa

Hasta ahora estuvimos ejecutando las cosas en el intérprete. Cómo armamos un programa?

Simplemente escribimos la secuencia de expresiones en un archivo **.py**

```
# file: prueba1.py
a = 3
b = 7
c = a * b + b**2

print(c)
```

Luego ejecutamos en una terminal

```
$ python3 /home/username/Documents/prueba.py
70
```

Control de flujo: **if**

Permite ejecutar código condicionado a que una dada condición sea verdadera.

```
$ python3 prueba_if.py  
a es menor a 10
```

```
# file: prueba_if.py  
a = 7  
if a < 10:  
    print('a es menor a 10')
```


Control de flujo: **if**

Permite ejecutar código condicionado a que una dada condición sea verdadera.

```
$ python3 prueba_if.py  
a es menor a 10
```

Un poco más avanzado ...

```
$ python3 prueba_ifelse.py  
a es mayor a 10
```

```
# file: prueba_if.py  
a = 7  
if a < 10:  
    print('a es menor a 10')
```

```
# file: prueba_ifelse.py  
a = 11  
if a < 10:  
    print('a es menor a 10')  
elif a > 10:  
    print('a es mayor a 10')  
else:  
    print('a es igual a 10')
```

Control de flujo: **if**

Permite ejecutar código condicionado a que una dada condición sea verdadera.

```
$ python3 prueba_if.py  
a es menor a 10
```

Un poco más avanzado ...

```
$ python3 prueba_ifelse.py  
a es mayor a 10
```

```
# file: prueba_if.py  
a = 7  
if a < 10:  
    print('a es menor a 10')
```

```
# file: prueba_ifelse.py  
a = 11  
if a < 10:  
    print('a es menor a 10')  
elif a > 10:  
    print('a es mayor a 10')  
else:  
    print('a es igual a 10')
```

Atención: el indentado **importa!** El espacio en blanco delimita el código dentro de cada bloque condicional.

Control de flujo: **for**

Permite iterar sobre los elementos de una lista

```
$ python3 prueba_for.py  
1  
3  
a
```

```
# file: prueba_for.py  
l = [1, 3, 'a']  
for e in l:  
    print(e)
```

Control de flujo: **for**

Permite iterar sobre los elementos de una lista

```
$ python3 prueba_for.py  
1  
3  
a
```

```
# file: prueba_for.py  
l = [1, 3, 'a']  
for e in l:  
    print(e)
```

Un tipo de lista muy útil

```
$ python3 prueba_range.py  
0  
1  
2  
3
```

```
# file: prueba_range.py  
N = 4  
for n in range(N):  
    print(n)
```

Control de flujo: **break**, **continue**

break: para la iteración

```
$ python3 prueba_break.py  
0  
1
```

```
# file: prueba_break.py  
for n in range(4):  
    if n == 2:  
        break  
    print(n)
```

Control de flujo: **break**, **continue**

break: para la iteración

```
$ python3 prueba_break.py  
0  
1
```

```
# file: prueba_break.py  
for n in range(4):  
    if n == 2:  
        break  
    print(n)
```

continue: saltea un paso de la iteración

```
$ python3 prueba_continue.py  
0  
1  
3
```

```
# file: prueba_continue.py  
for n in range(4):  
    if n == 2:  
        continue  
    print(n)
```

Control de flujo: **while**

Otra forma de crear loops

```
$ python3 prueba_while.py  
0  
1  
2  
3
```

```
# file: prueba_while.py  
a = 0  
while a < 4:  
    print(a)  
    a = a + 1
```

Definición de funciones

Definimos una función para sumar

```
$ python3 prueba_func1.py  
7
```

```
# file: prueba_func1.py  
def suma(a, b):  
    s = a + b  
    return s  
  
print(suma(3, 4))
```


Definición de funciones

Definimos una función para sumar

```
$ python3 prueba_func1.py  
7
```

```
# file: prueba_func1.py  
def suma(a, b):  
    s = a + b  
    return s  
  
print(suma(3, 4))
```

Los parámetros de la función pueden tener valores default

```
$ python3 prueba_func2.py  
4  
7
```

```
# file: prueba_func2.py  
def suma(a, b=1):  
    s = a + b  
    return s  
  
print(suma(3))  
print(suma(3,4))
```

Librerías Científicas

... o porqué usamos Python

Cálculo numérico:

- ▶ **NumPy**: <http://www.numpy.org/>
- ▶ **SciPy**: <https://www.scipy.org/>

Gráficos:

- ▶ **Matplotlib**: <http://matplotlib.org/>

Otros:

- ▶ **Sympy**: <http://www.sympy.org/> (Cálculo simbólico)
- ▶ **Pandas**: <http://pandas.pydata.org/> (Manipulación y análisis de datos)

NumPy

- ▶ Librería central para la programación científica con Python.
- ▶ Introduce un nuevo tipo fundamental: el **NumPy array** (para representar vectores, matrices, tensores de mayores rangos).
- ▶ Librería de rutinas de álgebra lineal, integración numérica, transformadas Fourier, generación números aleatorios, estadística, ...
- ▶ Integración con código en C y FORTRAN

NumPy arrays

```
>>> import numpy as np
>>> a = np.array([1, 3, 5, 8, 9])
>>> a
array([1, 3, 5, 8, 9])
>>> a[0]
1
>>> a[-1]
9
>>> a[1:4]
array([3, 5, 8])
>>> b = np.array([5, 4, 1, 7, 0])
>>> a + b
array([6, 7, 6, 15, 9])
>>> a * b
array([5, 12, 5, 56, 0])
>>> a**2
array([1, 9, 25, 64, 81])
```

NumPy arrays

```
>>> M = np.zeros((2, 5))
>>> M.shape
(2, 5)
>>> M[0, :] = a
>>> M[1, :] = b
>>> M
array([[1, 3, 5, 8, 9],
       [5, 4, 1, 7, 0]])
>>> L = M.T
>>> L.shape
(5, 2)
>>> np.dot(M, L)
array([[180, 78],
       [ 78, 91]])
```

NumPy arrays

```
>>> a = np.linspace(0, 1, 5)
>>> a
array([0, 0.25, 0.5, 0.75, 1])
>>> b = np.arange(0, 1, step=0.3)
>>> b
array([0, 0.3, 0.6, 0.9])
```

Matplotlib

Librería para producir gráficos de alta calidad

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
z = np.cos(x)

plt.figure()
plt.plot(x, y, '-o', color='blue', label='sin(x)')
plt.plot(x, z, '-s', color='red', label='cos(x)')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Matplotlib

