Introducción a Python

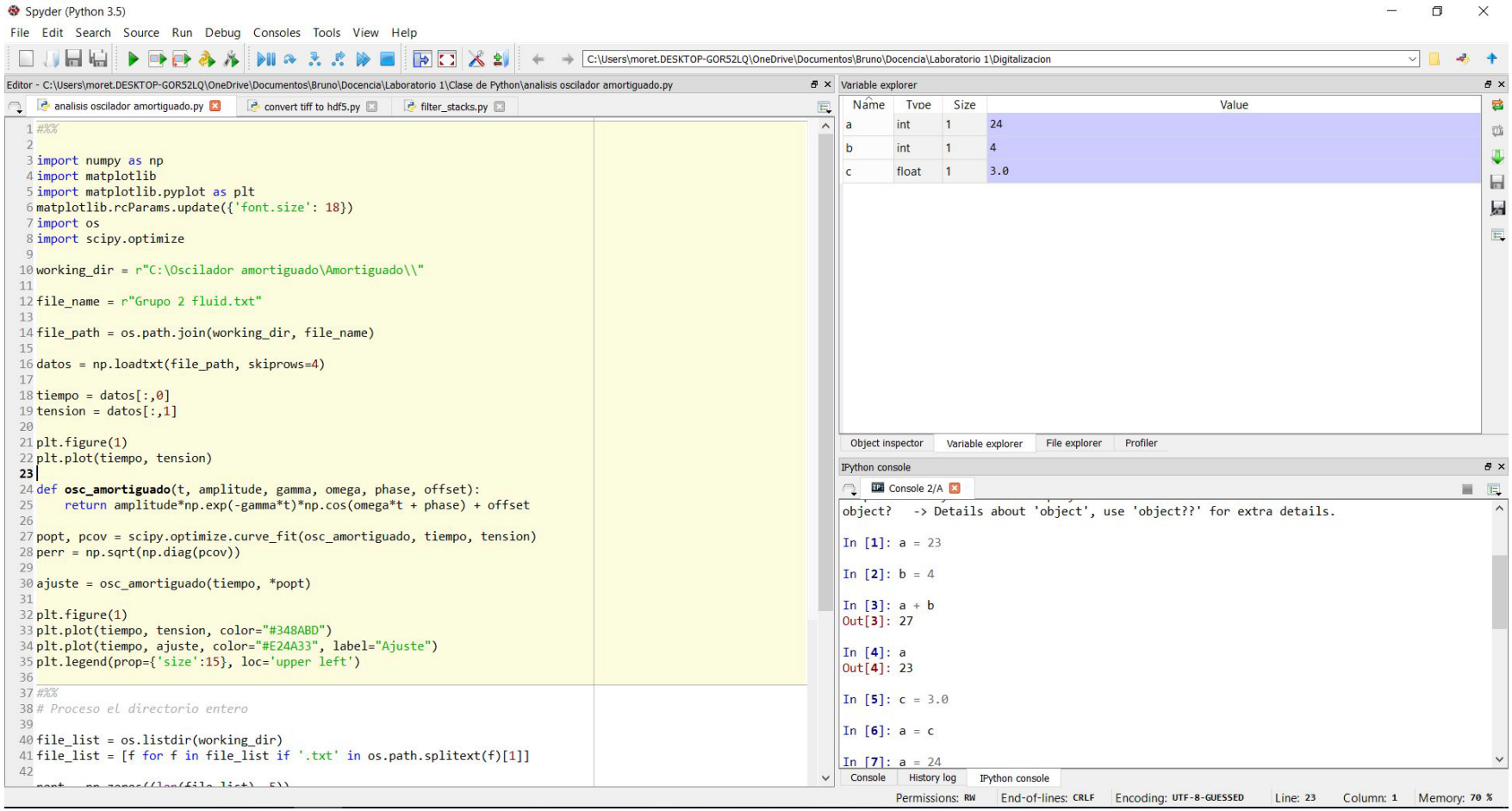
Python es un lenguaje de programación:

- Interpretado (no compilado)
- Multipropósito
- De alto nivel
- Enfocado en la **legibilidad**
- Muy usado en la comunidad científica

En el laboratorio nos resulta útil para hacer:

- Análisis de datos
- Simulaciones
- Control de instrumentos
- Automatización de experimentos

Spyder: entorno interactivo de Python



Hay distintos *tipos de datos*

- Números enteros (int)
- Números de punto flotante (float)
- Cadenas de caracteres de texto (string)
- Valores lógicos (bool)
- Listas de cualquier tipo de los anteriores (list)

Números

- Números enteros (int):

```
In [11]: a = 3

In [12]: b = 4

In [13]: a + b
Out[13]: 7

In [14]: a*b
Out[14]: 12
```

- Números de punto flotante (float):

```
In [19]: d = 3.0

In [20]: e = 1.43

In [21]: d*e
Out[21]: 4.29
```

Ante la duda, el comando 'type' nos dice de que tipo es una variable:

```
In [23]: type(a)
Out[23]: int

In [24]: type(d)
Out[24]: float

In [25]: type(a/b)
Out[25]: float
```

Texto (strings)

```
In [37]: palabra_1 = "Hola "

In [38]: palabra_2 = "que "

In [39]: palabra_3 = "tal"

In [40]: frase = palabra_1 + palabra_2 + palabra_3

In [41]: print(frase)
Hola que tal

In [42]: type(palabra_1)
Out[42]: str
```

Texto (strings)

```
In [37]: palabra_1 = "Hola "

In [38]: palabra_2 = "que "

In [39]: palabra_3 = "tal"

In [40]: frase = palabra_1 + palabra_2 + palabra_3

In [41]: print(frase)
Hola que tal

In [42]: type(palabra_1)
Out[42]: str
```

Como regla general, solo podemos hacer operaciones entre variables del mismo tipo:

```
In [45]: palabra_1 + 2
Traceback (most recent call last):

  File "<ipython-input-45-c354e20d870e>", line 1, in <module>
    palabra_1 + 2

TypeError: Can't convert 'int' object to str implicitly


In [46]: palabra_1 + str(2)
Out[46]: 'Hola 2'
```

Variables lógicas (bool)

```
In [54]: cond = 20 > 10

In [55]: cond
Out[55]: True

In [56]: type(cond)
Out[56]: bool
```

```
In [47]: 2 == 2
Out[47]: True

In [48]: 2 == 3
Out[48]: False

In [49]: 9 > 40
Out[49]: False

In [50]: 3 != 3
Out[50]: False
```

Variables lógicas (bool)

```
In [54]: cond = 20 > 10

In [55]: cond
Out[55]: True

In [56]: type(cond)
Out[56]: bool
```

```
In [47]: 2 == 2
Out[47]: True

In [48]: 2 == 3
Out[48]: False

In [49]: 9 > 40
Out[49]: False

In [50]: 3 != 3
Out[50]: False
```

Se pueden aplicar varias condiciones
a la vez con las palabras claves ‘and’ y ‘or’:

```
In [51]: 20 > 10 and 20 > 18
Out[51]: True

In [52]: 20 > 10 or 20 < 10
Out[52]: True

In [53]: 20 > 30 or 20 < 10
Out[53]: False
```

Listas

```
In [68]: frutas = ["manzana", "naranja", "banana"]

In [69]: frutas.append("frutilla")

In [70]: print(frutas)
['manzana', 'naranja', 'banana', 'frutilla']

In [71]: frutas[0]
Out[71]: 'manzana'

In [72]: frutas[1]
Out[72]: 'naranja'

In [73]: frutas[-1] # ultimo elemento de la lista
Out[73]: 'frutilla'
```

“Slicing” de listas

```
In [74]: numeros = [1, 2, 3, 4, 5, 6, 7]

In [75]: numeros[0]
Out[75]: 1

In [76]: numeros[-1]
Out[76]: 7

In [77]: numeros[2:4]
Out[77]: [3, 4]

In [78]: numeros[2:]
Out[78]: [3, 4, 5, 6, 7]
```

```
In [85]: len(numeros) # Cantidad de elementos de la lista
Out[85]: 7

In [86]: len(frutas)
Out[86]: 4
```

Estructuras de control

- Loops (*for*)
- Condicionales (*if*)
- Loops condicionales (*while*)

Loops (*for*)

```
In [87]: for i in range(10): # 'range(10)' da todos los indices entre 0 y 9
    ...:     print(i)
    ...:
0
1
2
3
4
5
6
7
8
9
```

Loops (*for*)

```
In [88]: for i in range(10):
    ...:     print("Hola")
    ...:
Hola
Hola
Hola
Hola
Hola
Hola
Hola
Hola
Hola
Hola
```

También podemos iterar listas

```
In [90]: for fruta in frutas:
    ...:     print(fruta)
manzana
naranja
banana
frutilla
```

El comando ‘enumerate’ nos devuelve también los índices:

```
In [91]: for n, fruta in enumerate(frutas):
    ...:     print(n)
    ...:     print(fruta)
    ...:
0
manzana
1
naranja
2
banana
3
frutilla
```

Condicionales (*if*)

```
In [101]: nota = 7

In [102]: if nota > 4:
     ...:     print("Aprobado")
     ...:     if nota > 7:
     ...:         print("Promociona")
     ...: else:
     ...:     print("Desaprobado")
     ...:
Aprobado
```

Condicionales (*if*)

```
In [101]: nota = 7

In [102]: if nota > 4:
     ...:     print("Aprobado")
     ...:     if nota > 7:
     ...:         print("Promociona")
     ...: else:
     ...:     print("Desaprobado")
     ...:
Aprobado
```

```
In [97]: nota = 8

In [98]: if nota > 4:
    ...:     print("Aprobado")
    ...:     if nota > 7:
    ...:         print("Promociona")
    ...: else:
    ...:     print("Desaprobado")
Aprobado
Promociona
```

```
In [99]: nota = 2

In [100]: if nota > 4:
     ...:     print("Aprobado")
     ...:     if nota > 7:
     ...:         print("Promociona")
     ...: else:
     ...:     print("Desaprobado")
Desaprobado
```

Loop condicional (*while*)

```
In [104]: while i < 10:
     ...:     print(i)
     ...:     i = i + 1
0
1
2
3
4
5
6
7
8
9
```

Importando librerías: *numpy*

Sin agregarle nada extra, Python no puede hacer muchas operaciones matemáticas:

```
In [107]: sin(9)
Traceback (most recent call last):

  File "<ipython-input-107-75d6e8105620>", line 1, in <module>
    sin(9)

NameError: name 'sin' is not defined


In [108]: pi
Traceback (most recent call last):

  File "<ipython-input-108-68f7b1e53523>", line 1, in <module>
    pi

NameError: name 'pi' is not defined


In [109]: exp(4)
Traceback (most recent call last):

  File "<ipython-input-109-c5a79b883a1e>", line 1, in <module>
    exp(4)

NameError: name 'exp' is not defined
```

Importando librerías: *numpy*

Vamos a importar y usar la librería *numpy* para hacer estas cosas (y muchas más):

```
In [117]: import numpy as np

In [118]: np.pi
Out[118]: 3.141592653589793

In [119]: np.cos(24)
Out[119]: 0.42417900733699698

In [120]: np.cos(np.pi)
Out[120]: -1.0

In [121]: np.exp(4)
Out[121]: 54.598150033144236

In [122]: np.log(0)
C:\Anaconda3\lib\site-packages\spyderlib\widgets\externalshell\start_ipython_kernel.py:1: RuntimeWarning: divide by zero
encountered in log
  # -*- coding: utf-8 -*-
Out[122]: -inf

In [123]: np.log(1)
Out[123]: 0.0
```

Definiendo funciones

Si vamos a reutilizar pedazos de código que ya implementamos, es recomendable definir funciones. Por ejemplo, esta que suma dos números:

```
In [124]: def sumar(x, y):
     ...:     return x + y
     ...:

In [125]: sumar(5, 6)
Out[125]: 11
```

x e *y* son las *variables de entrada* de la función, que devuelve lo que está siguiendo a 'return'

Definiendo funciones

Si vamos a reutilizar pedazos de código que ya implementamos, es recomendable definir funciones. Por ejemplo, esta que suma dos números:

```
In [124]: def sumar(x, y):
     ...:     return x + y
     ...:

In [125]: sumar(5, 6)
Out[125]: 11
```

x e *y* son las *variables de entrada* de la función, que devuelve lo que está siguiendo a 'return'

O esta que eleva un número a la enésima potencia:

```
In [126]: def elevar_a_la_n(x, n):
     ...:     return x**n
     ...:

In [127]: elevar_a_la_n(3, 2)
Out[127]: 9
```

Importando librerías: *matplotlib*

La librería que vamos a usar para graficar se llama *matplotlib:*

```
In [2]: x = np.linspace(0, 30, 1000) # Defino un vector de mil puntos entre 0 y 30

In [3]: y = np.cos(x)

In [4]: import matplotlib.pyplot as plt

In [5]: plt.plot(x, y)
Out[5]: [<matplotlib.lines.Line2D at 0x2c256bda908>]
```

Importando librerías: *matplotlib*

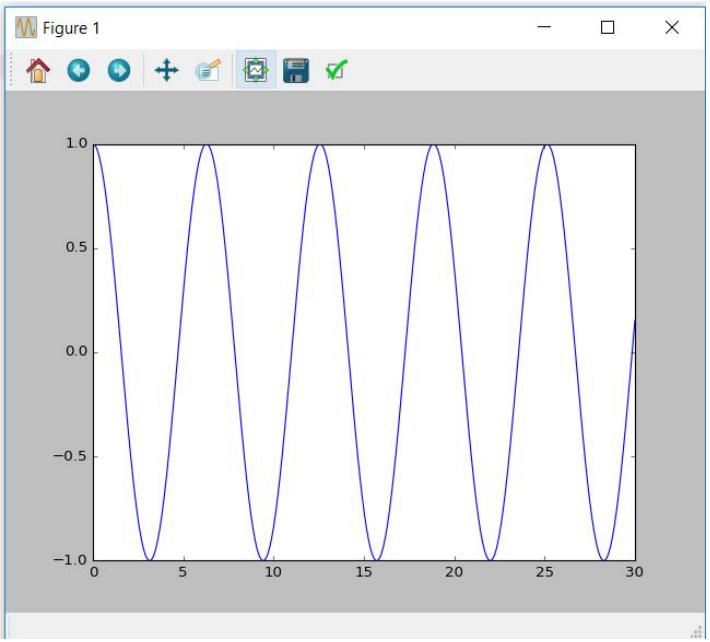
La librería que vamos a usar para graficar se llama *matplotlib:*

```
In [2]: x = np.linspace(0, 30, 1000) # Defino un vector de mil puntos entre 0 y 30

In [3]: y = np.cos(x)

In [4]: import matplotlib.pyplot as plt

In [5]: plt.plot(x, y)
Out[5]: [<matplotlib.lines.Line2D at 0x2c256bda908>]
```



Ahora si, carguemos algunos datos experimentales

```
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.optimize

ruta_archivo = r"C:\Oscilador amortiguado\Amortiguado\Grupo 2 fluid.txt\\"

# Usamos la funcion np.loadtxt para cargar archivos de texto.
# skiprows = 4 hace que saltee el encabezado que genera el MotionDAQ
datos = np.loadtxt(ruta_archivo, skiprows=4)

# 'datos' es una matriz con dos columnas, una con los tiempos y otra con las
# tensiones. Genero dos variables, una con cada columna:
tiempo = datos[:,0]
tension = datos[:,1]

plt.figure(1)
plt.plot(tiempo, tension)
```

Y hagamos un ajuste no lineal

```
# Defino la función por la que voy a ajustar
def osc_amortiguado(t, amplitud, gamma, omega, fase, offset):
    return amplitud*np.exp(-gamma*t)*np.cos(omega*t + fase) + offset

# Para ajustar usamos la funcion scipy.optimize.curve_fit:
parametros_ajuste, pcov = scipy.optimize.curve_fit(osc_amortiguado, tiempo, tension)
errores_ajuste = np.sqrt(np.diag(pcov))
```

```
In [20]: print(parametros_ajuste)
[ -0.25472283   0.13203138  11.37390231  -1.70307493  -0.66584592]

In [21]: print(errores_ajuste)
[  5.63387699e-04   4.15213411e-04   4.13519543e-04   2.19411844e-03
   9.92726701e-05]
```

Y hagamos un ajuste no lineal

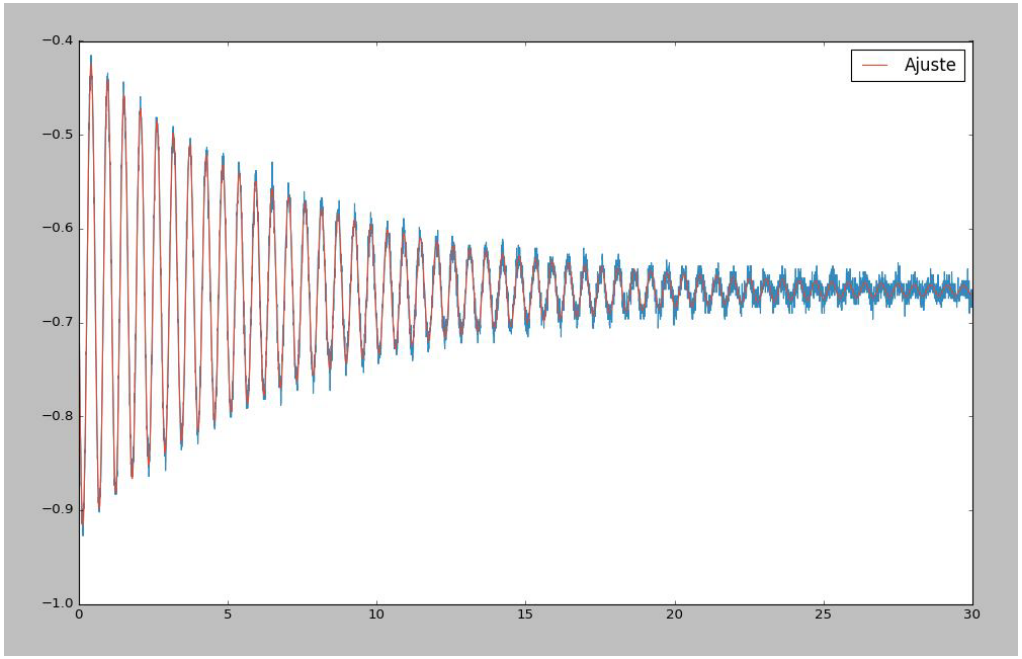
```
# Para graficar el ajuste, le aplico la funcion osc_amortiguado a los parametros
# que encontre usando curve_fit:
ajuste = osc_amortiguado(tiempo, *parametros_ajuste)

plt.figure(1)
plt.plot(tiempo, tension, color="#348ABD")
plt.plot(tiempo, ajuste, color="#E24A33", label="Ajuste")
plt.legend(prop={'size':15}, loc='upper right')
```

Y hagamos un ajuste no lineal

```
# Para graficar el ajuste, le aplico la funcion osc_amortiguado a los parametros
# que encontre usando curve_fit:
ajuste = osc_amortiguado(tiempo, *parametros_ajuste)

plt.figure(1)
plt.plot(tiempo, tension, color="#348ABD")
plt.plot(tiempo, ajuste, color="#E24A33", label="Ajuste")
plt.legend(prop={'size':15}, loc='upper right')
```



Si ordenamos los archivos **de forma razonable**, podemos procesar un directorio entero:

```
# Proceso el directorio entero

working_dir = r"C:\Oscilador amortiguado\Amortiguado\\"

# Leo todos los archivos del directorio y me quedo con los que terminan en .txt:
file_list = os.listdir(working_dir)
file_list = [f for f in file_list if '.txt' in os.path.splitext(f)[1]]

# Ahora voy a hacer una matriz de parametros y una de errores
parametros_ajustes = np.zeros((len(file_list), 5))
errores_ajustes = np.zeros_like(parametros_ajustes)
for (n, file) in enumerate(file_list):
    file_path = os.path.join(working_dir, file)

    datos = np.loadtxt(file_path, skiprows=4)
    tiempo = datos[:,0]
    tension = datos[:,1]

    parametros_ajustes[n], pcov = scipy.optimize.curve_fit(osc_amortiguado, tiempo, tension)
    errores_ajustes[n] = np.sqrt(np.diag(pcov))
```

```
In [23]: print(parametros_ajustes)
[[ -1.09749608   0.1792061   11.90232221   0.5635373   -0.8488655 ]
 [ -0.25472283   0.13203138  11.37390231  -1.70307493  -0.66584592]
 [ -1.08396827   0.14420449  10.70295152  -2.24555336  -0.39100076]
 [ -1.16051188   0.14068943  10.30841036   0.03362741  -0.20586446]
 [  1.1730082    0.12892326  -9.80059022   4.44814514   0.07088519]
 [ -0.91844318   0.10788191   9.0943735   -0.53141611   0.53042319]]
```