INTRODUCCIÓN A PYTHON

INTRODUCCIÓN A PYTHON

1. ¿Qué es Python?
2. Elementos básicos del lenguaje
3. Instalación de Python. Librerías Numpy, Scipy.
4. Entornos de programación.
   - Spyder
   - Otros

¿Qué es Python?

- Creado por Guido van Rossum.
- **Python** es un lenguaje de programación interpretado que hace hincapié en la legibilidad y la transparencia de su código.
- Es software libre y está administrado por la Python Software Foundation.
- Una de sus características es que tiene una cantidad enorme de librerías que hacen tareas de todo tipo. Esto es posible por la facilidad para extender el lenguaje usando incluso lenguajes como C/C++.

Interacción con Python

- **Modo de ejecución**. Se necesita un intérprete para ejecutar los programas Python (similar a la máquina virtual de JAVA). Menor velocidad que lenguajes compilados como C/C++.
- **Modo interactivo**. El intérprete tiene un modo interactivo que permite ir ejecutando instrucciones y ver los resultados. Bueno para hacer pruebas.

```
>>> 1+1
2
>>> a = range(10)
>>> print (list(a))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> b=1
>>> a=3
>>> b+a
4
```

INTRODUCCIÓN A PYTHON

Elementos básicos del lenguaje

Módulos

- Las funciones adicionales que necesitemos usar y sean "extensiones" de Python están incluidas en módulos (similar a las librerías de C).
- Para cargarlos se usa la orden `import` al inicio:

```
import cv2 # Carga la librería OpenCV

import numpy as np # Carga la librería numpy y le asigna el alias np
```

Legibilidad

Uso de palabras donde otros lenguajes usan símbolos.

`not, or, and` en lugar de `!, ||, &&`

Favorece la legibilidad y comprensión del código.

Legibilidad

Los bloques de código se delimitan mediante indentación y no con símbolos especiales ({})

Estilo C

```
int factorial(int x)
{
    if (x == 0)
        return 1;
    else
        return x * factorial(x - 1);
}
```

Estilo Python

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Comentarios

Los comentarios se pueden poner de dos formas:

- Usando # Comentario, lo que escribamos desde la almohadilla hasta el final de línea es un comentario.
- Usando ''' Comentario '''. Lo que haya entre los tres apóstrofes de apertura y los tres de final es un comentario y puede tener más de una línea. Se usa para comentarios largos.

Variables

- Las variables se definen de forma dinámica. No se definen explícitamente. Se usa el símbolo = para la asignación.
- Los tipos de las variables pueden cambiar en ejecución.

```
x = 1
x = "texto" # Los tipos son asignados dinámicamente
```

Condicionales (if)

- Especialmente importante no olvidar la indentación.

```
>>> Asig = "PIVC"
>>> if Asig == "Señales": # asignatura no es "PIVC", por lo que este bloque se
obviará y evaluará la siguiente condición
...     print ("Asignatura: Señales")
... elif Asig == "PIVC": # Se pueden añadir tantos bloques "elif" como se quiera
...     print ("Asignatura: PIVC")
... else: # En caso de que ninguna de las anteriores condiciones fuera cierta,
se ejecutaría este bloque
...     print ("Asignatura: ?????")
...
Asignatura: PIVC
```

Bucle for

- Repite operaciones sobre un objeto susceptible de ser iterado. Puede ser una lista de cualquier tipo, no sólo numérico.

```
>>> lista = ["a", "b", "c"]
>>> for i in lista: # Iteramos sobre una lista, que es iterable
...     print (i)

>>> for i in range(1,10,2): # Bucle desde 1 hasta 10 en saltos de 2
...     print (i)
```

NOTA: Evitar en lo posible los bucles *for* si necesitamos acelerar procesos. Sobre todo si es posible realizar operaciones matriciales.

Bucle while

- Realizamos operaciones mientras la condición sea verdadera.

```
>>> numero = 0
>>> while numero < 10:
...     numero += 1
...     print (numero)
```

NOTA: Cuando necesitemos salir de un bucle podemos usar la instrucción *break*.

Funciones

- Las funciones se definen con la palabra clave *def*, seguida del nombre de la función y sus parámetros.
- Los resultados se devuelven con *return*.

```
>>> def suma(x, y = 2):
...     return x + y # Retornar la suma del valor de la variable "x" y el valor de "y"
...
>>> suma(4) # La variable "y" no se modifica, siendo su valor: 2
6
>>> suma(4, 10) # La variable "y" sí se modifica, siendo su nuevo valor: 10
14
```

Listas y Tuplas

- Son similares pero las listas son modificables y las tuplas no.
- Incluyen (de forma ordenada) distintos elementos que pueden ser de diferente tipo.
  - Lista: lista1=[1, 2, 'verano', 'invierno']
  - Tupla: lista2=(1, 2, 'verano', 'invierno')
- Listas y tuplas empiezan en 0. El último elemento es accesible con el índice -1

Listas y Tuplas

- del lista1[1] # Borra un elemento de la lista usando un índice
- del lista2[1] # Da un error
- Se puede acceder a más de un elemento:
  - lista1[0:2] # Mostrar los elementos de la lista del índice "0" al "2" (**sin incluir este último**)

Objetos y clases

- Python soporta programación orientada a objetos. Por tanto se pueden definir clases y objetos de esas clases.
- Las clases son tipos que incluyen no sólo datos, sino que también incluyen las funciones para acceder a ellos y modificarlos.
- Ejemplo. Las listas son clases y para añadir un dato a una lista se llama a una función de la clase que es *append*.

```
lista1.append(2) #Añade el número 2 a la lista creada anteriormente.
```

Ejercicio

- Realizar un programa en Python que permita realizar la multiplicación de dos números y muestre el resultado por pantalla. La multiplicación debe realizarse como sumas sucesivas.

$$3*4=3+3+3+3=12$$

Ejercicio

- Para la edición del programa use el editor de texto que prefiera.
- Guarde el programa como mult.py
- Para ejecutarlo ponga en una ventana de comandos:
  - python mult.py
- Los números a multiplicar serán dos variables definidas dentro del programa.

Parámetros de entrada

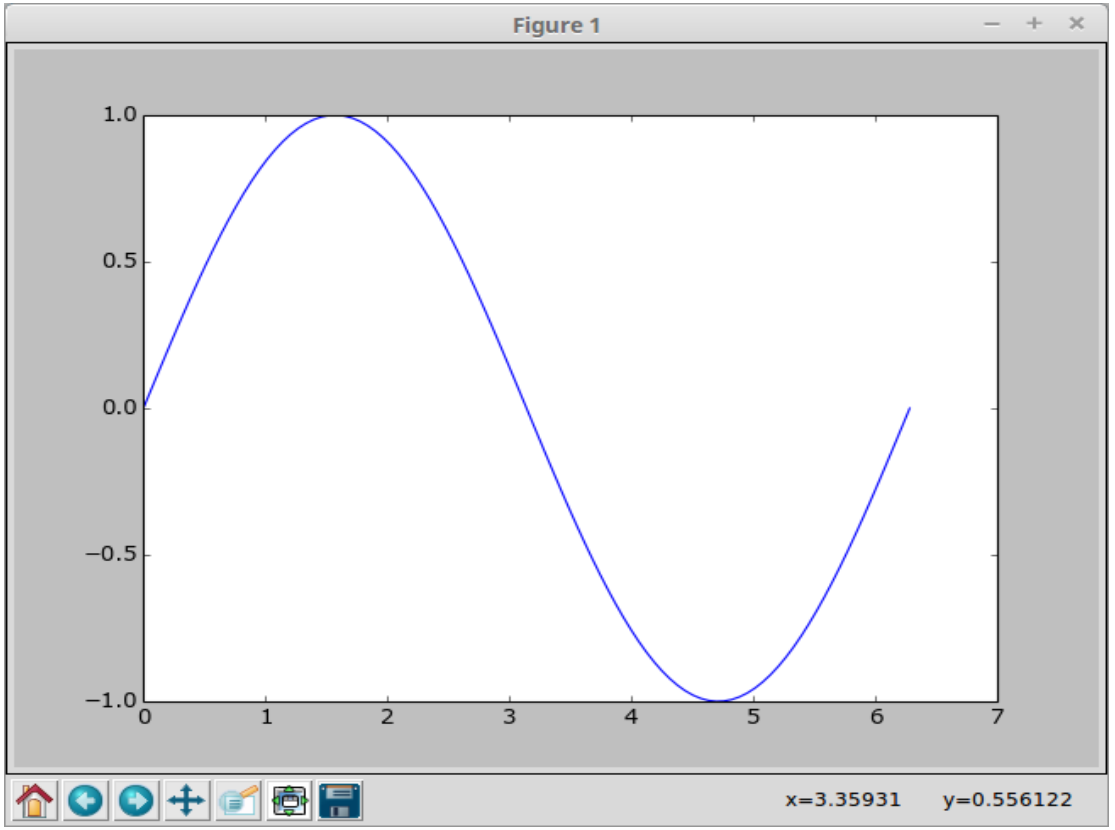
- Se pueden pasar parámetros a un programa con la funcionalidad del módulo sys (*import sys*)
- Funciona de forma similar a C:
  - sys.argv[0] es el nombre del programa
  - sys.argv[1] es el parámetro 1
  - sys.argv es la cadena de lo que hemos tecleado
  - int(argv[1]) nos dará un entero
  - float(argv[1]) nos dará un float
  - len(sys.argv) nos dice el número de argumentos pasados.

Numpy

- **NumPy** es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.
- OpenCV la usa ya que las imágenes serán matrices NumPy.
- La sintaxis y el uso es similar a Matlab.

Ejemplo (Wikipedia)

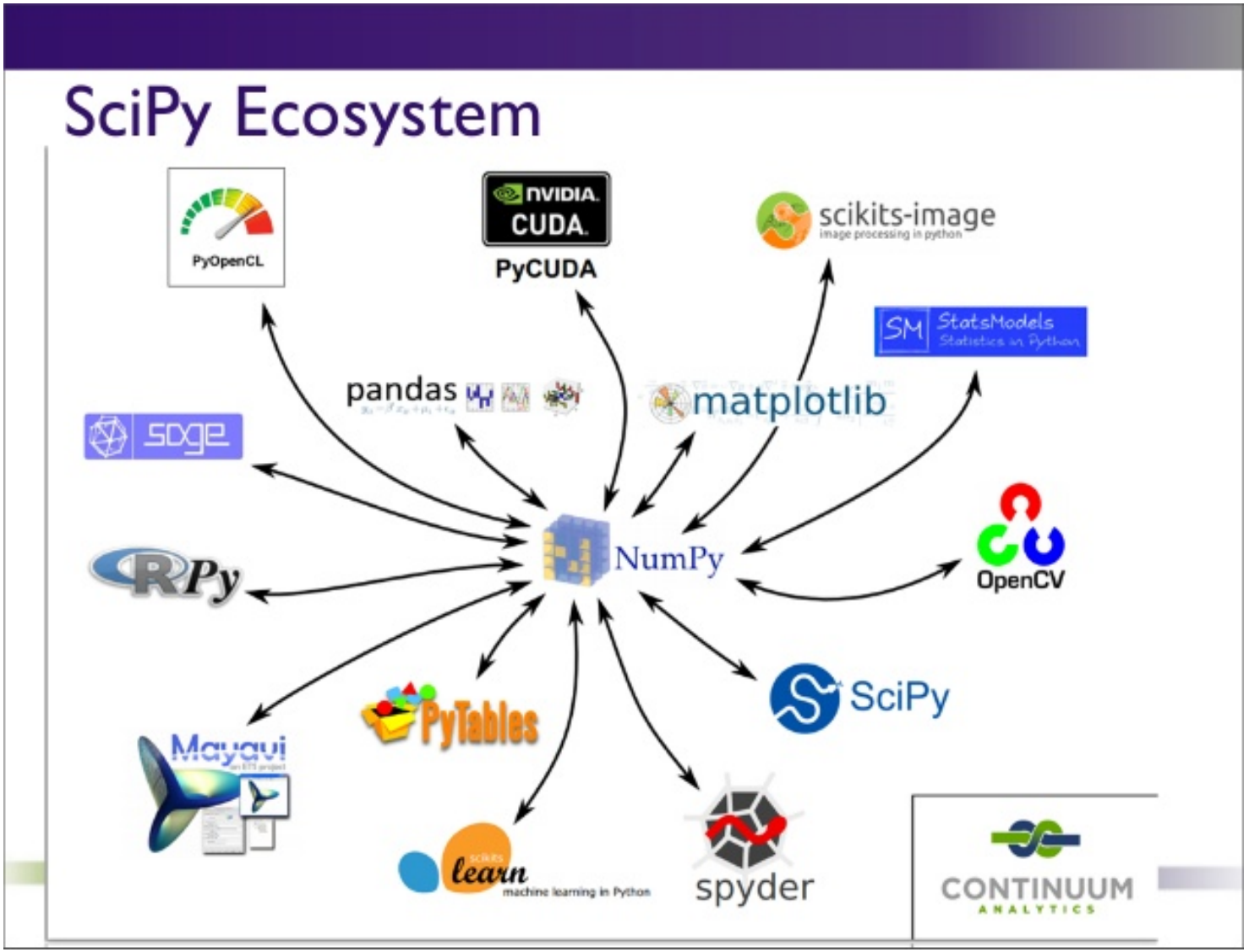
```
import numpy
from matplotlib import pyplot
x = numpy.linspace(0, 2 * numpy.pi, 100)
y = numpy.sin(x)
pyplot.plot(x, y)
pyplot.show()
```



Scipy

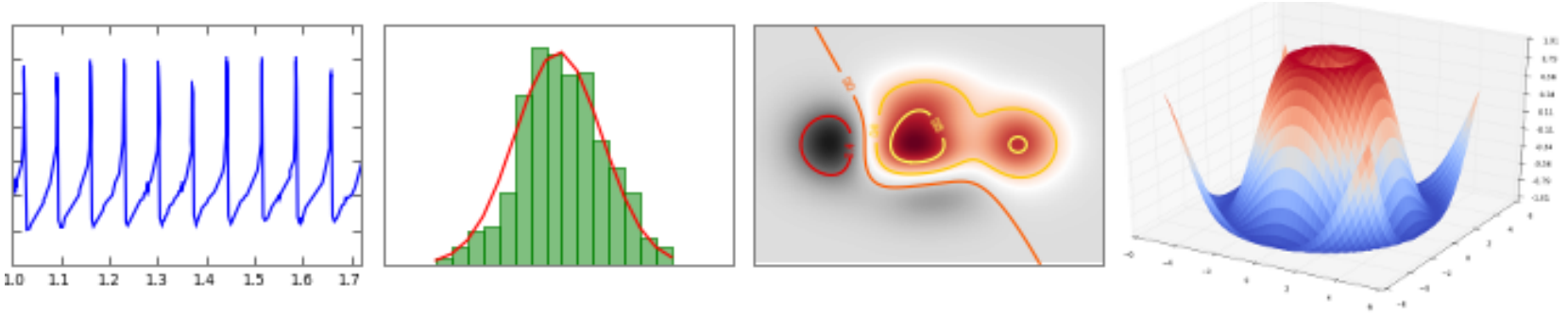
- **SciPy** es una extensión de Python que contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería.
- Nos será útil para el procesado de imagen.

Scipy


SciPy Ecosystem
PyOpenCL
NVIDIA
CUDA
PyCUDA
scikits-image
image processing in python
SM
StatsModels
Statistics in Python
pandas
matplotlib
sage
RPy
NumPy
OpenCV
SciPy
PyTables
Mayavi
scikits
learn
machine learning in Python
spyder
CONTINUUM
ANALYTICS

Matplotlib

- **Matplotlib** es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy



INTRODUCCIÓN A PYTHON

Instalación de Python

Instalación (Linux)

- Viene instalado por defecto en casi todas las distribuciones (Ubuntu, Debian, etc.) y sus derivadas.
- La versión que utilizaremos es la 3.x. Para comprobar la versión que tiene instalada escriba en el terminal:

```
python3 -V
```

- Puede coexistir con la versión antigua 2.x. Para usar dicha versión hay que escribir:

```
python
```

Instalación (Linux)

- Para instalar módulos (Numpy, Scipy) la mejor opción es usar el instalador de paquetes del sistema. Por ejemplo, en un sistema Debian (Ubuntu):

  sudo apt-get install python3-numpy python3-scipy

- Para obtener versiones más recientes se puede usar 'pip'. Se trata del propio gestor de paquetes de Python. Para poder usarlo en Linux hay que instalarlo:

  sudo apt-get install python3-pip

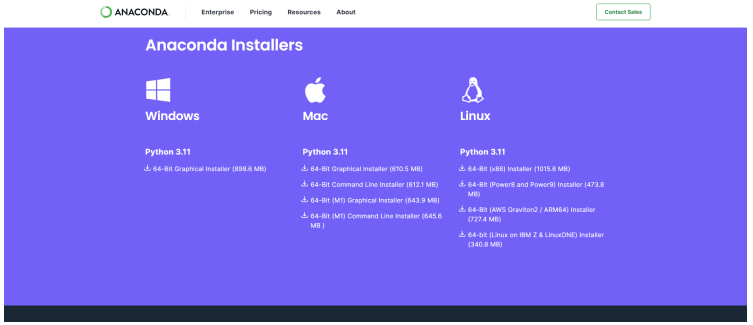
- Una vez instalado, podemos escribir:

  sudo pip3 install --upgrade numpy

Instalación (Windows)

- La mejor opción es instalar una distribución de Python que ya incluye los paquetes necesarios. En la asignatura utilizaremos Anaconda.

- Anaconda. También Mac
  - Haremos click en 'Downloads'
  - Descargaremos la distribución correspondiente a nuestro sistema



  - Una vez instalado, podemos abrir un terminal de Anaconda y comprobar la versión instalada tecleando:

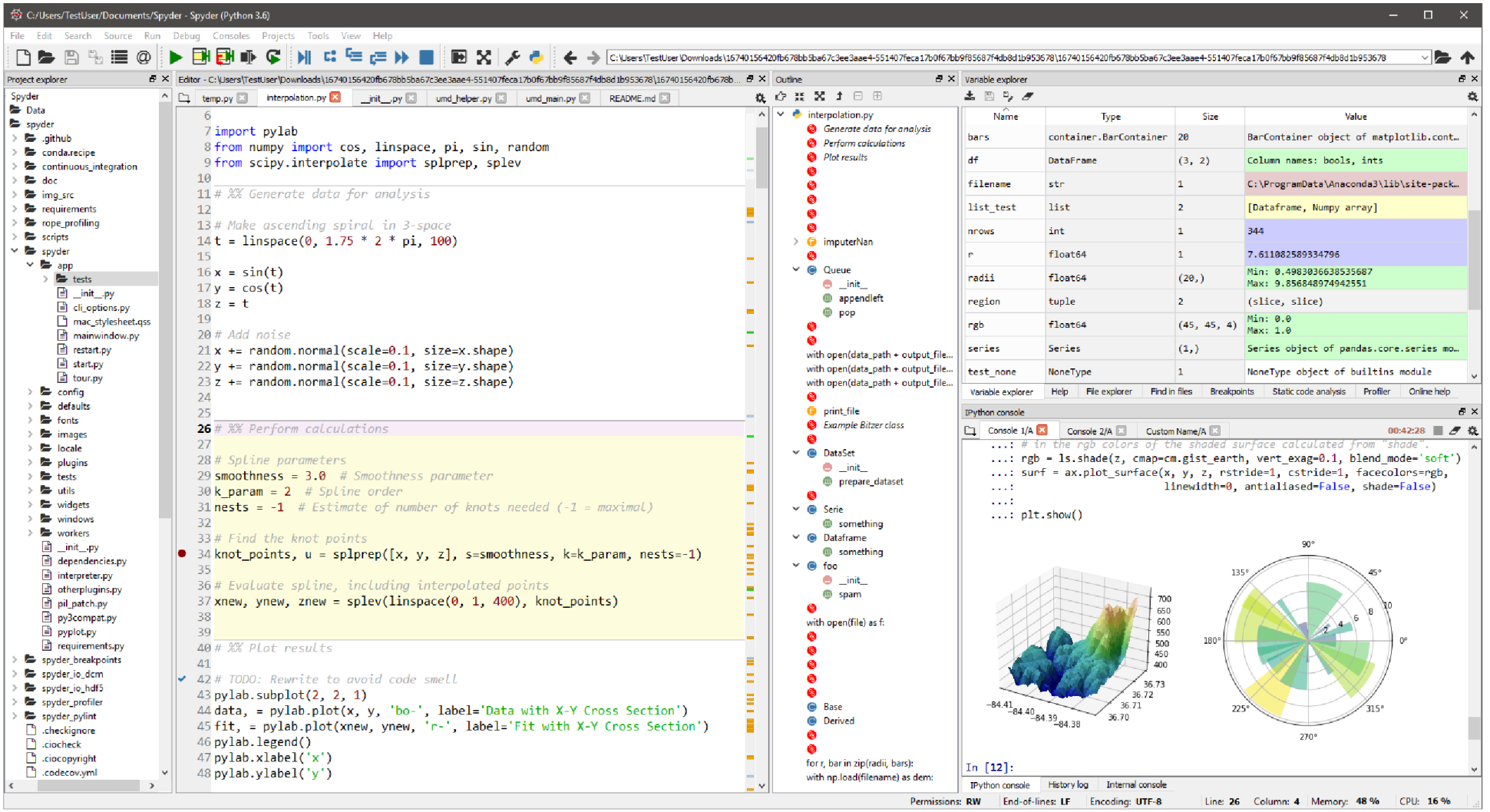
```
python --version
```

Instalación (Mac)

- Instalaremos también Anaconda
- Consultar este manual de instalación.

INTRODUCCIÓN A PYTHON

Entornos de programación


SPYDER
The Scientific Python Development Environment

Spyder

- Spyder es un IDE de código abierto multiplataforma y que está escrito en Python y que sólo permite programar en Python.
- Es un IDE simple, eficiente y bien documentado.
- Viene incluido en la distribución Anaconda.

Otras opciones

- Existen numerosos IDEs con la opción de programar en Python.
- En este enlace hay una lista bastante completa.
- Entre los más usados:
  - **PyCharm**. https://www.jetbrains.com/pycharm/
  - PyDev. https://www.liclipse.com/
  - Idle. https://docs.python.org/3/library/idle.html
  - Visual Studio Code. https://code.visualstudio.com/