Python y GeoGebra

Computación 2021

FaMAF

26 mar. 2021

Contenidos

Python como programa

Línea de comandos

- El *shell* de python
  - calculadora
  - desarrollo
  - correr scripts
  - ver y correr scripts
- python desde el *shell* de linux
  - correr scripts

En general, la documentación distingue entre el shell de linux ($) y el shell de python (>>>).

Se puede acceder al shell de linux desde el shell de ipython anteponieno un signo de admiración (!). Muchos comandos (como `ls` o `pwd`) no requieren el signo (!). ▷e.g.:

```
>>> !which python
```

```
>>> ls
```

El intérprete de python

Además de un lenguaje, **python es también un programa** que se utiliza para correr código escrito en python, que se conoce como "intérprete". Este programa puede estar escrito en python (pypy), en C (cpython), Java (jython) o .NET (ironpython).

Se accede escribiendo "python" en una terminal. También se puede usar "ipython" que provee algunas funcionalidad extra (pero el lenguaje es el mismo!) De esta forma se puede correr código de manera interactiva.

Para salir, se puede hacer alternativamente:

- `<Ctrl> d`
- `exit()`

El intérprete de python

También se puede cargar código desde el intérprete con:

- `run` (solo en ipython!)
- `load` (solo en ipython!)
- `exec(open('hello.py').read)`
- `import hello`
- `execfile()` NO funciona (era de python2)

Python desde el shell de linux

Se puede ejecutar código escrito en python, **sin entrar el intérprete**, de muchas formas. Principalmente:

- desde el shell de linux pasando un archivo
- desde el shell de linux pasando un string
- haciendo un ejecutable
- usando un IDE
- desde el manejador de archivos (depende de la configuración del sistema)

Ejecutar un script de python

Lo más usado es pasarle al intérprete un archivo con código:

```
$ python hello.py
```

Se puede incluso pasar por línea de comandos el texto que se quiere ejecutar (notar la opción -c):

```
$ python -c "\nprint('hola mundo!\n')"
```

También está la opción -m para cargar un módulo.

```
$ python -m hello
```

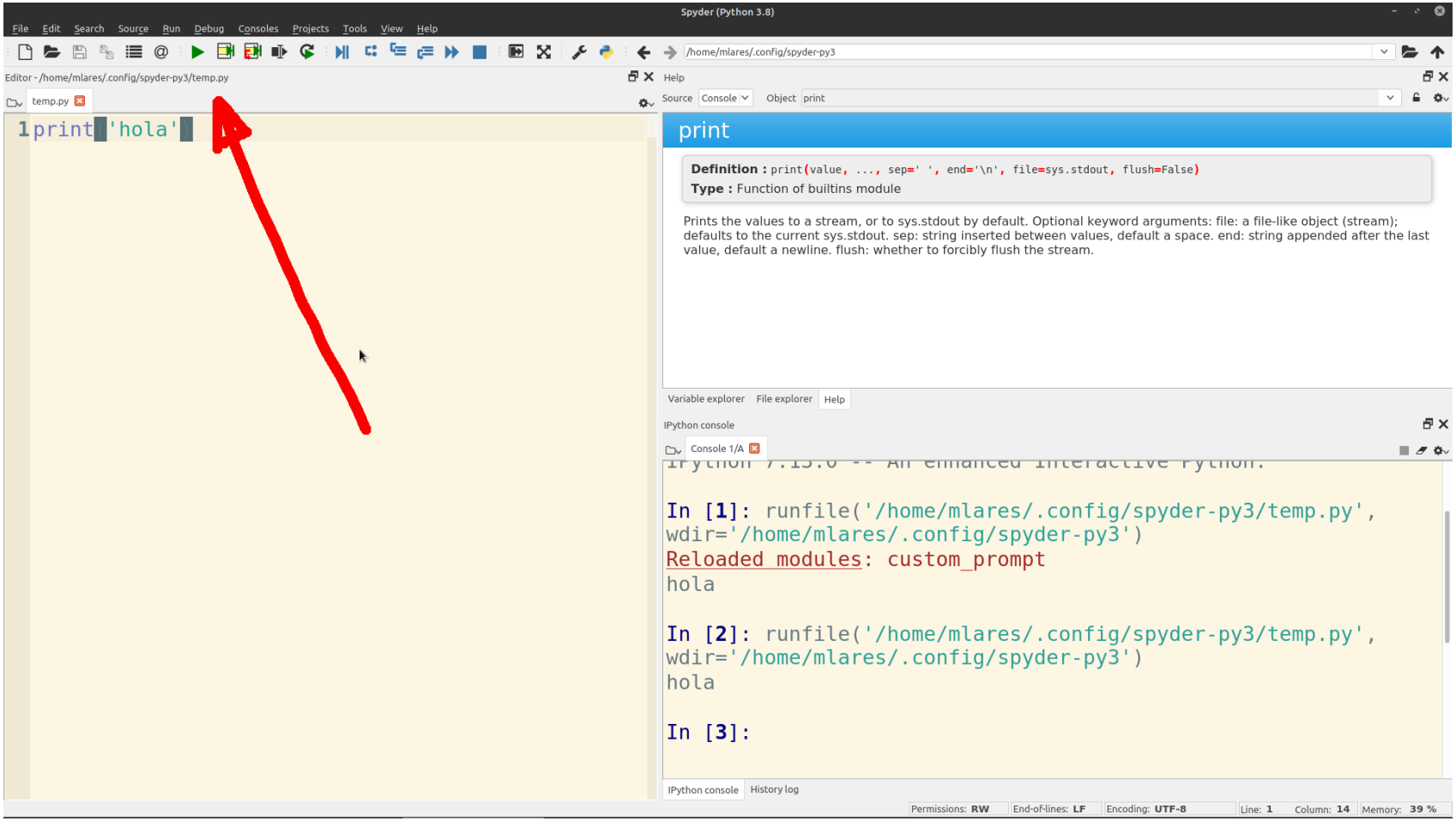
Ejecutables de python

- Encontrar el archivo ejecutable de python (*). En linux se puede usar el comando `which`
- Crear un archivo de texto (e.g. hello_exe.py)
- Escribir en la primera línea "#!" y agregar la ubicación (*)
- Escribir el código y guardar el archivo
- Darle permisos de ejecución: chmod +x hello_exe.py
- Poner en nombre del archivo anteponiendo "./"

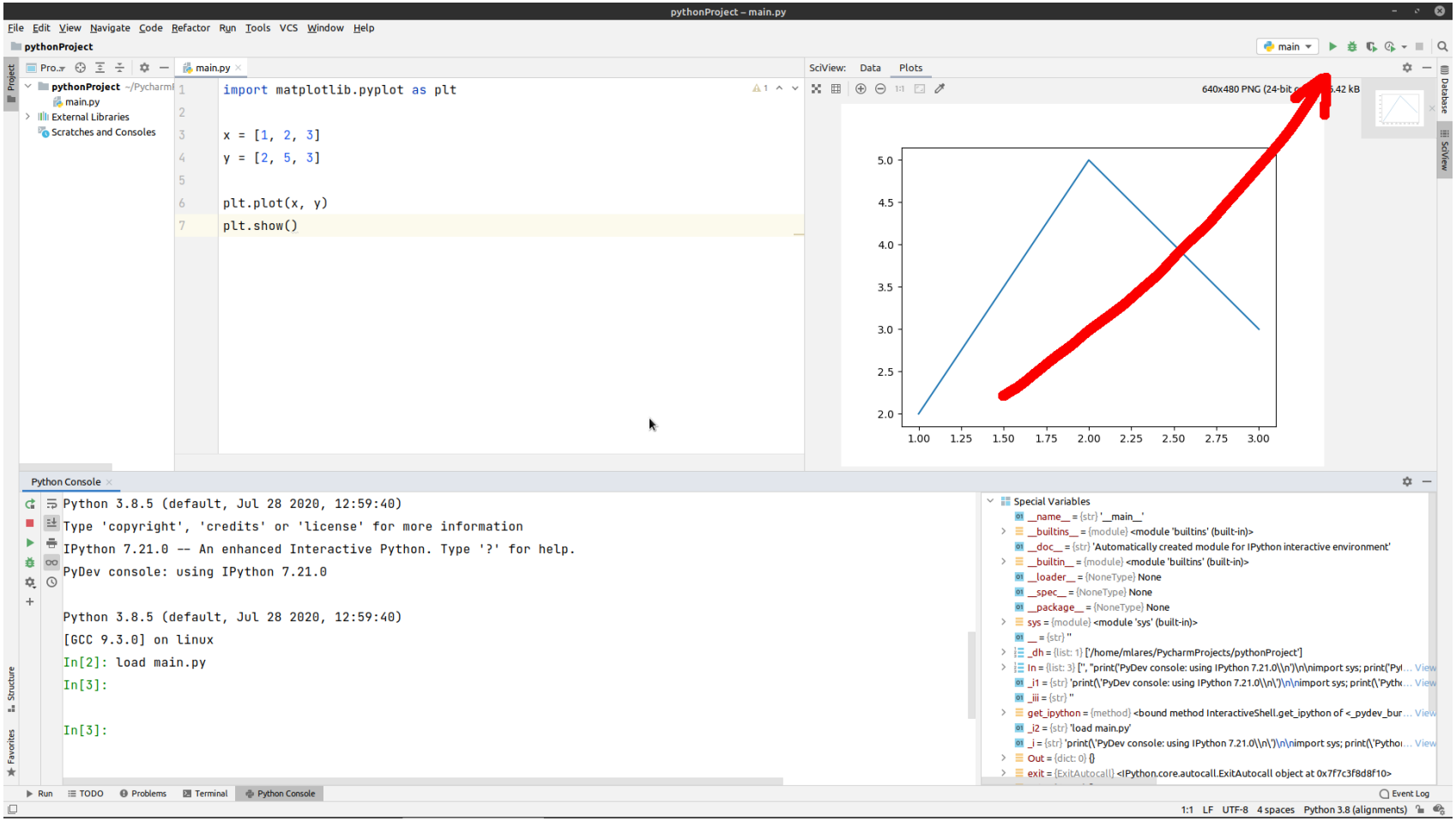
Entornos de desarrollo más usados

- jupyter
- spyder
- pycharm
- vscode
- muchos más!

Ejecutar un script de python: spyder



Ejecutar un script de python: pycharm



Entornos virtuales

Dado que python es un lenguaje que evoluciona muy rápidamente, a veces pasa que determinados paquetes cambian y hay cosas que dejan de funcionar. Para solucionar esto los paquetes mantienen un registro de la versión (con un número) y se puede trabajar en entornos virtuales.

Más info: `https://docs.python.org/3/tutorial/venv.html`

Además, los entornos virtuales permiten ensayar la instalación de paquetes, borrarlos, modificarlos, etc., sin peligro de degradar el sistema.

Virtualenvwrapper y conda

En general se terminan usando muchos entornos (uno por cada proyecto). Para facilitar el manejo de entornos se cuenta con virtualenwrapper o anaconda. ▷e.g.:Para un environment llamado "myenv":

acción	virtualenvwrapper	anaconda
crear(*)	`mkvirtualenv myenv`	`conda create myenv`
entrar	`workon myenv`	`conda activate myenv`
salir	`deactivate`	`conda deactivate`
borrar	`rmvirtualenv myenv`	`conda env remove -n myenv`
listar	`lsvirtualenv`	`conda info -envs`
directorio	`setvirtualenvproject`	

Para elegir el python, `--python=$(which python3)`

`conda` es al mismo tiempo un manjeador de paquetes (como pip) y un manejador de entornos virtuales.

Python como lenguaje

scripts vs. módulos

Una de las ventajas de los lenguajes de programación es la modularización, que permite separar porciones de código en achivos separados y utilizarlos desde un script.

Un script es un archivo de texto plano que contiene código a ser ejecutado por el usuario.

Un módulo es un archivo de texto plano que contiene código a ser utilizado por por otro archivo. En python hay muchos módulos que son muy usados, por ejemplo: `math, numpy, matplotlib, astropy, seaborn, datetime, random, itertools, pandas, ...`
Estos paquetes o módulos NO vienen incluidos en la librería estándar, sí en anaconda.

scripts vs. módulos

Un paquete es un conjunto organizado de módulos que están en un árbol de directorios. Por ejemplo, `numpy` o `matplotlib`.

▷e.g.: numpy

Para cargar módulos de un paquete hay que saber sus nombres (ver documentación)

Para instalar un paquete o módulo que falta, usar `pip` desde la terminal, y para cargar un módulo usar el comando de python `import`.

scripts vs. módulos

Instalar paquetes o módulos desde el prompt de linux:

```
$ pip install numpy
```

O bien con anaconda:

```
$ conda install numpy
```

Cargar paquetes o módulos desde el intérprete de python:

```
>>> import math
>>> from numpy.polynomial import legendre
>>> from matplotlib import pyplot as plt
>>> from random import random
```

Definición del lenguaje

El lenguaje queda definido básicamente por:

- sintaxis (▷e.g.:import, function, while, help, etc.)
- estructura (▷e.g.:cómo hacer blucles, condicionales, funciones, etc.)
- tipos de datos (▷e.g.:enteros, cadenas de caracteres, etc.) y cómo se comportan.

Variables

Cuando se crea un objeto en python, se reserva un porción de memoria para almacenarlo. Existe una ubicación en la memoria de esta porción, que se denomina "dirección de memoria". Un usuario de python no necesita conocer la dirección de memoria de un objeto (pero se puede, con `id`)

Para trabajar haciendo cómputos o aplicando algoritmos es conveniente darle un nombre conveniente y que sea fácil de identificar. Un identificador de un objeto es un nombre que sirve para acceder a ese objeto, identificarlo y usarlo. Los identificadores se conocen comunmente como "variables".

El contenido de una variable, para los tipos más básicos, es el "valor" de esa variable.

Objetos

- enteros (`int`), reales (`float`), complejos (`complex`)
- cadenas de caracteres (`str`)
- listas (`list`)
- tuplas (`tuple`)
- diccionarios (`dict`)
- conjuntos (`set`)

Paradigmas de programación

Los estilos de programación estructurada se dividen básicamente en dos tipos:

- procedimental (o "funcional")
- orientada a objetos

El estilo procedimental consiste en encapsular un conjunto de procedimientos para reutilizarlos escribiendo poco código.
Las funciones en python siguen el estilo procedimental, y es el estilo que vamos a usar en el práctico. Una vez definidas, las funciones no modifican su estado ni almacenan datos, pero responden distintas cosas según cómo se las llame.

Muchas herramientas de python usan el estilo orientado a objetos, por lo que para entender cómo se usan veremos brevemente de qué se trata.

Programación funcional

Una función es un conjunto de sentencias agrupadas bajo un nombre, de tal forma que pueden ser ejecutadas más de una vez desde un programa. Las ventajas de las funciones son que:

- pueden ser reusadas sin necesidad de repetir muchas sentencias parecidas en diferentes partes de un programa
- permiten separar tareas complejas en unidades más simples

La sentencia para crear funciones es def.

```
1  def func(x, y):
2      """ una funcioncita simple que devuelve
3          el mayor de dos números. """
4      z = x if x>y else y
5      return z
```

Programación orientada a objetos

Un objeto representa un concepto de algún tipo, almacena datos sobre sí mismo (sus "`atributos`") y es capaz de llevar a cabo ciertas acciones mediante funciones (los "`métodos`"). Al ejecutar un método, un objeto puede cambiar sus atributos, que representan su estado, sus propiedades o su descripción. Un objeto es creado a partir de una especie de "molde", que se denomina `clase`

▷e.g.:Puedo crear un objeto tipo lámpara, que tienen atributos como "cantidad de tiempo que estuvo prendida", "marca", "potencia", etc., y tienen métodos tales como "prender", "apagar", o "quemar".

Programación orientada a objetos

```
1  toby = perro()
2  toby.nombre = 'Toby'
3  toby.ladrar()
4  toby.comer()
```

Programación orientada a objetos

En python todo es un objeto. Tomemos por ejemplo un string.

```
1  s = str("mensaje")
```

estamos creando un objeto de la clase "string". Este proceso se llama "instanciar". Al crearlo le asignamos datos, que es el string "mensaje". Además, por se un objeto de ese tipo, ya tiene los métodos de todo string. Por ejemplo, podemos pedirle que se pase a mayúsculas.

```
1  s.capitalize()
```

Esto es porque hay un método (una función) almacenado en la definición de la clase. Notar que hay un punto, que separa el identificador (el nombre de la variable) del método. Cada vez que veamos un punto, estamos ejecutando un método de un objeto.

Visualización orientada a objetos

```
1  from matplotlib import pyplot as plt
2
3  fig = plt.figure()
4  ax = fig.add_subplot(111)
5  ax.grid()
6  fig.savefig('figura.png')
```

GeoGebra

No todo es python...