AGH

Organization of work with Python and the ipython interactive shell

Wydział:	—
Instytut:	podaj nazwę instytutu
Autor:	Piotr Kramarczyk
Recenzent:	Beata Hejmanowska

Kraków - October 2019

Spis treści

1. Introduction

Working with Python will include:

- executing individual code lines in the interactive ipython shell
- creating simple functions in the interactive ipython shell
- saving functions in module files with the py extension using a text editor
- importing and running functions in the interactive ipython shell
- creating scripts using a text editor
- running scripts in the interactive ipython shell
- running scripts from the system terminal

These tasks require mastering the basic operation of the Python interpreter and the interactive ipython shell.

2. ipython keyboard shortcuts

Start *ipython*: in the system terminal (Linux) or in the *Anaconda Prompt* (Windows) write *ipython.*

Keyboard shortcuts:

TAB - completes and suggests syntax

name_object + ? e.g. myFunction? display description (doc string) if available

%hist / %hist n - displays history of entered commands / with rows numbering

ctrl + o - put a new line behind the cursor

ctrl + a - moves the cursor to the beginning of the line

ctrl + e - moves the cursor to the end of the line

ctrl + l - cleans screen

ctrl + c - interrupts the current action

ctrl + u - deletes the entire text line

Help: https://ipython.readthedocs.io/en/stable/interactive/tutorial.html

3. Using *Python* modules

Generally speaking, python code is divided into parts called modules. The modules are text files with the extension *py*. By default, only base modules are loaded. Using other modules, e.g. to perform scientific calculations, charts, etc. requires: - installing them on the system - each time loading modules into the computer's memory called importing

Because the modules are files saved on your computer, *Python* must „know" where to look for these files. By default, *Python* searches the disk in the order:

- in current folder
- in folder specified in the system variable *PYTHONPATH*
- in folders given during installation *Pythona*
- in folders secified in variable *sys.path*, which is oridinaly list. This path can be edited while the interpreter is running, the path configuration is lost after ending interpreter.

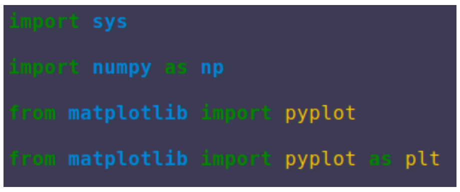
Because I'm going the course is to use the simplest solutions are:

- created modules will be placed in the working directory
- a temporary search path will be added to *sys.path*

3.1. Modules import

Basic ways to import modules or their parts (functions, classes):

- *import module_name* e.g.: import sys - imports a module e.g: sys module
- *import module_name as alias* e.g.: import numpy as np - imports a module by assigning an alias to it
- *from module_name import name_submodule_class_function* e.g: from matplotlib import pyplot



```
import sys

import numpy as np

from matplotlib import pyplot

from matplotlib import pyplot as plt
```

If changes are made to the module to make them visible, it must be reloaded:

reload module_name: reload module - *Python 2.7.*

from importlib import reload: for *Python 3.*, reloading the module requires importing the *reload* module first and next:

reload module_name - reload module

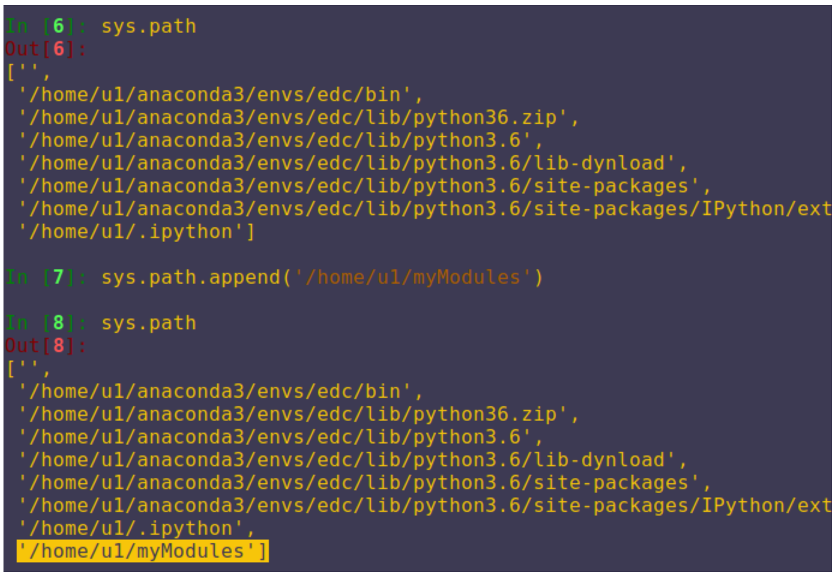
3.2. Temporarily adding a directory to the search path

To add a new temporary search path, you must import the *sys* module:

„This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available."

From: https://docs.python.org/2/library/sys.html

1. Import module: *import sys*
2. Displaying list of the paths - *sys.path*
3. Add a new temporary path - *sys.path.append('path to folder with module files')*



```
In [6]: sys.path
Out[6]:
['',
 '/home/u1/anaconda3/envs/edc/bin',
 '/home/u1/anaconda3/envs/edc/lib/python36.zip',
 '/home/u1/anaconda3/envs/edc/lib/python3.6',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/lib-dynload',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/site-packages',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/site-packages/IPython/ext
 '/home/u1/.ipython']

In [7]: sys.path.append('/home/u1/myModules')

In [8]: sys.path
Out[8]:
['',
 '/home/u1/anaconda3/envs/edc/bin',
 '/home/u1/anaconda3/envs/edc/lib/python36.zip',
 '/home/u1/anaconda3/envs/edc/lib/python3.6',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/lib-dynload',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/site-packages',
 '/home/u1/anaconda3/envs/edc/lib/python3.6/site-packages/IPython/ext
 '/home/u1/.ipython',
 '/home/u1/myModules']
```

4. Executing *Python* code

Python commands can be can be:

- write directely in *ipython* consola
- save to a text file with an extension *.py*

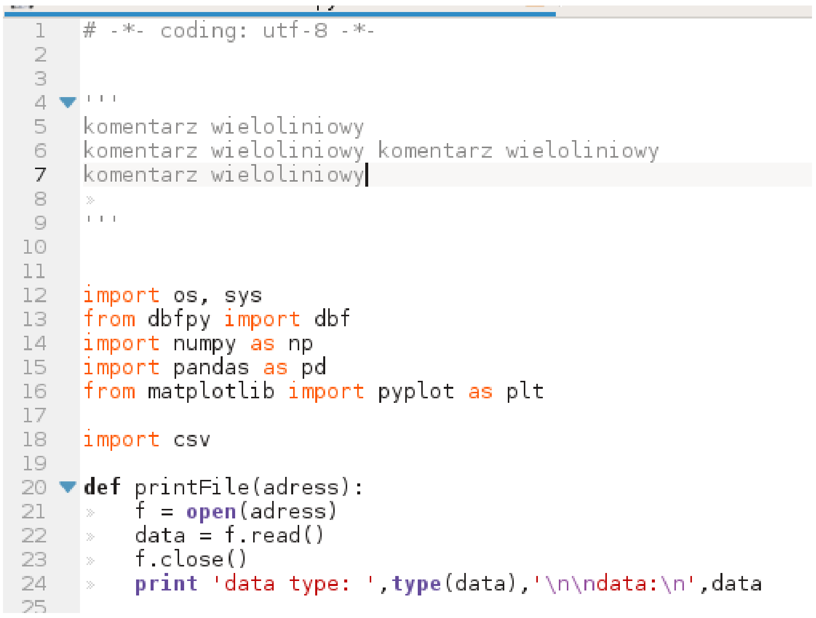
4.1. Interactive console *ipython*

In *ipython* console:

- write single-line instructions e.g. 2 + 2, *print('some text')*, etc.
- write a multi-line command, e.g. *for i in range:...*
- define function, classes etc.

4.2. Files *.py*

Bigger (many-lines) programs is better to save in text files with extension *py*. This files can be recognized by *Python* interpreter as modules/programms.



```
1  # -*- coding: utf-8 -*-
2
3
4  '''
5  komentarz wieloliniowy
6  komentarz wieloliniowy komentarz wieloliniowy
7  komentarz wieloliniowy
8
9  '''
10
11
12 import os, sys
13 from dbfpy import dbf
14 import numpy as np
15 import pandas as pd
16 from matplotlib import pyplot as plt
17
18 import csv
19
20 def printFile(adress):
21     f = open(adress)
22     data = f.read()
23     f.close()
24     print 'data type: ',type(data),'\n\ndata:\n',data
25
```

In the figure below an example of the file part *py* is presented, where:

line 1 : # says that line is a comment. In this case it is a special comment concering the way of character encoding.

line 4-9 : ' ' ' indicates many lines comment

line 12-16 : modules importing

line 20-24 : key word *def* and function definicion

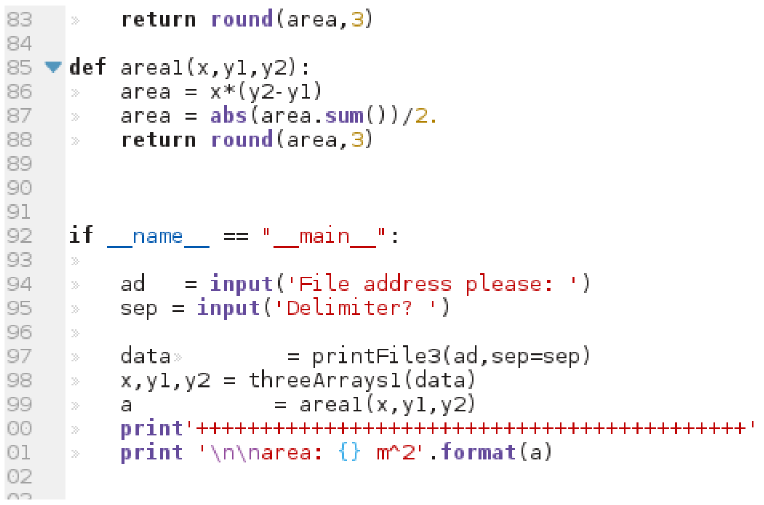
In the figure below the end part of the same file is shown, where:

line 92 line says that code above it can not be executed, and code below it should be executed

line 92 " characters that limit text type data(string)

line 94 ' the second type of character - signs limiting data type text(string) simllar like in line 94

line 94-101 successive program instructions executed in the order indicated from top to bottom



```
    return round(area,3)

def area1(x,y1,y2):
    area = x*(y2-y1)
    area = abs(area.sum())/2.
    return round(area,3)



if __name__ == "__main__":

    ad   = input('File address please: ')
    sep = input('Delimiter? ')

    data         = printFile3(ad,sep=sep)
    x,y1,y2 = threeArrays1(data)
    a           = area1(x,y1,y2)
    print'+++++++++++++++++++++++++++++++++++++++++++'
    print '\n\narea: {} m^2'.format(a)

```

4.3. Executing *Python* code saved in *.py* files

Python code saved in the *.py* file (functions, classes, scripts) can be run in several ways:

1. From interactive *ipython* shell:
   - in the form of imported functions and classes
   - as a script run using the magic function *%run*: *%run 'full path to the file .py'*
2. From the system terminal or *Anaconda Prompt* - as scripts :
   - system terminal: *python full path to the file .py*
   - *Anaconda Prompt full path to the file .py*