How to Feed a Python

R. THOMAS @ ESO

Why would you need to know about that? (Non-exhaustive list!)

[BASICS]:

- Read from files (Hu → Py)
- Ask for parameters on the fly (Py ←→ Hu)

[Intermediate]:

- Multi - parameters/options python codes (Hu → Py)

[Advanced] Wanna make your code public (**YOU SHOULD :)**):

- Assume the user is lazy (often the case!) → Easy way to tune the code
- More attractive if the user does not have to dig into your code to modify things

→ **save time and energy**

[BASICS]

Input:

```
[alien@AlienArchRomain ~]$ ipython
Python 3.6.2 (default, Jul 20 2017, 03:52:27)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: a = input()
what?

In [2]: a
Out[2]: 'what?'

In [3]:

In [3]:

In [3]: b = input('Skip?')
Skip?Yes

In [4]: b
Out[4]: 'Yes'

In [5]:
```

When do you use that?

Particularly suitable for small scripts where only one *on-the-fly* parameter is required.

[BASICS]

Reading files:

```
f = 'filetoread.txt'

##################################

print(1)
H = open(f, 'r')
for i in H:
    print(i[:-1])
H.close()

####################################################
with open(f, 'r') as H:
    lines = H.readlines()
print('2', lines)

####################################################
F = open(f, 'r').read().split('\n')
print('3', F)
####################################################


###using numpy
import numpy

a,b = numpy.loadtxt(f, dtype='str').T
print('numpy1', a,b)

a,b = numpy.genfromtxt(f, dtype='str').T
print('numpy2',a,b)
```

There are always a lot of ways to do one thing….

```
[alien@AlienArchRomain feed]$ python readfile.py
1
werwetwe      12354
qwrrwefe      95164
fwefweff      95432
wefwefwe      85213
fwefwfew      14793
2 ['werwetwe      12354 \n', 'qwrrwefe      95164\n', 'fwefweff      95432 \n', 'wefwefwe      85213\n', 'fwefwfew      14793\n']
3 ['werwetwe      12354 ', 'qwrrwefe      95164', 'fwefweff      95432 ', 'wefwefwe      85213', 'fwefwfew      14793', '']
numpy1 ['werwetwe' 'qwrrwefe' 'fwefweff' 'wefwefwe' 'fwefwfew'] ['12354' '95164' '95432' '85213' '14793']
numpy2 ['werwetwe' 'qwrrwefe' 'fwefweff' 'wefwefwe' 'fwefwfew'] ['12354' '95164' '95432' '85213' '14793']
[alien@AlienArchRomain feed]$
```

[BASICS]

Savings and loading big arrays:

```
import numpy

###create an array of ones with shape 1000x2000
a = numpy.ones((1000,2000))
#### ---> check the shape
print('\n', a.shape)
print('\n', a)

###save it on the disk
numpy.save('numpy_array', a)


### and loag it back
b = numpy.load('numpy_array.npy')

###check if they are the same
print('\n', (a==b).all())
```

```
(1000, 2000)

[[ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 ...,
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]
 [ 1.  1.  1. ...,  1.  1.  1.]]

True
```

[Intermediate] Multi - parameters/options python codes

```
########### define some parameters#############
q = 5
b = 6
d = 10

########### do something with them ###########

Z = q + b * d


########## define other ones ##############

k = 80
s = -2
l = 35

########## do something else #############

P = Z * (k/q) + (s/l)
```

If you find yourself tunning your algorithm modifying the code itself endless of time...

[Intermediate]

Two (main?) methods:

COMMAND LINE INTERFACE (CLI)

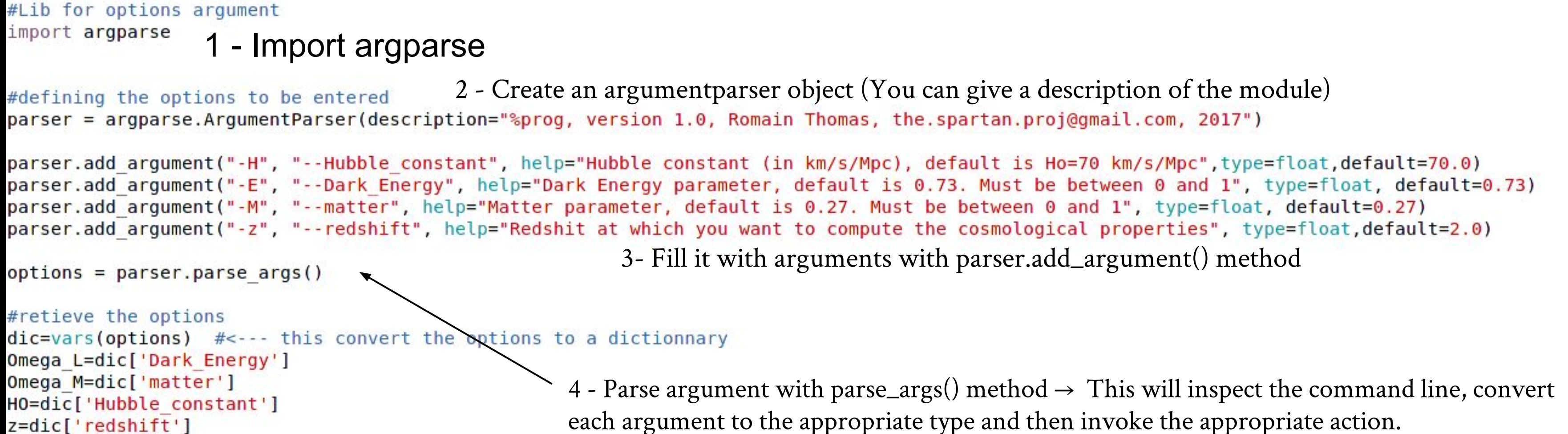
- Command line interface (CLI):

```
[alien@AlienArchRomain feed]$ blabla.py -a 3 -b 5 -g 6 -k 80 -s 2 -l 10
```

→ Allows you to leave your script untouched and still testing different configurations

[Intermediate]

Command line Interface: the *argparse* module (In the standard library)



[Intermediate]

using 'myprogram.py --help' will display the help of your program

```
[alien@AlienArchRomain cosmo]$ cosmo_at_z --h
usage: cosmo_at_z [-h] [-H HUBBLE_CONSTANT] [-E DARK_ENERGY] [-M MATTER]
                  [-z REDSHIFT]

%prog, version 1.0, Romain Thomas, the.spartan.proj@gmail.com, 2017

optional arguments:
  -h, --help            show this help message and exit
  -H HUBBLE_CONSTANT, --Hubble_constant HUBBLE_CONSTANT
                        Hubble constant (in km/s/Mpc), default is Ho=70
                        km/s/Mpc
  -E DARK_ENERGY, --Dark_Energy DARK_ENERGY
                        Dark Energy parameter, default is 0.73. Must be
                        between 0 and 1
  -M MATTER, --matter MATTER
                        Matter parameter, default is 0.27. Must be between 0
                        and 1
  -z REDSHIFT, --redshift REDSHIFT
                        Redshit at which you want to compute the cosmological
                        properties
[alien@AlienArchRomain cosmo]$
```

[Intermediate]

Two (main?) methods:

COMMAND LINE INTERFACE (CLI)

- Command line interface (CLI):

```
[alien@AlienArchRomain feed]$ blabla.py -a 3 -b 5 -g 6 -k 80 -s 2 -l 10
```

→ Allows you to leave your script untouched and still testing different configurations

→→→ That's probably fine for a low number of parameters

[Intermedi

Two (main?) meth   NTERFACE (CLI)

- Command lin

[ali   -s 2 -l 10

→ Allows y   erent configurations

→That's pr

```
1 [General]
2 project_name = Paper_spec_only_VVDS
3 author = Romain 2017 07 31
4 project_directory = /run/media/alien/SSD_240/GITLAB/SPARTAN_tests/
  Files/2017_07_31_spectro_topar
5 nnode = 1
6 ncpu = 3
7 data_cat = final_for_fits_VVDS.txt
8 use_spec = Yes
9 use_phot = No
10 nspec = 1
11 tryz = No
12
13 [Spectroscopy]
14 spectra_directory = /run/media/alien/SSD_240/Data/SPARTAN_2017_07_11_VUDS/
   Spectro/VVDS/ALL_VVDS/ascii
15 use_full_spec = Yes
16 binning =
17 resolution = 230
18 spec_redshift = Yes
19 flux_units = erg/s/cm2/A
20 wave_units = Ang
21 skip_edges = yes
22 size_skipped = 150
23 bad_regions = no
24 bad_regions_list =
25 normalisation_type = mags
26 multi_spec_calibration = no
27
28 [Photo]
29 system = AB
30 photofile = Paper_spec_only_VVDS.mag
31
32 [Cosmo]
33 ho = 70
34 omega_m = 0.27
35 omega_l = 0.73
36 use_cosmo = Yes
37
38 [Library]
39 type = provided
40 basessp = BC03_Delayed_LR_Chab_SPARTAN
41 dustuse = sb_calzetti
42 ebvlist = 0.0;0.05;0.1;0.2;0.3;0.4;0.5;0.6;0.7
43 igmtype = free_meiksin
44 emline = yes
45 age = 0.0500000e+9;0.101518e+9;0.12780400e+9;0.160896e+9;0.18052899e
```

[Intermediate]

Two (main?) methods:

- Command line arguments:

```
[alien@AlienArchRomain feed]$ blabla.py -a 3 -b 5 -g 6 -k 80 -s 2 -l 10
```

→ Allows you to leave your script untouched and still testing different configurations

→That's probably fine for a low number of parameters

- Configuration file (large number of parameters)

```
pi@clusterN1: ~/Documents/py
pi@clusterN1:~/Documents/pycoffee_UP $ blabla.py config.conf
```

[Intermediate]

Configuration file: *configparser* module (in the standard library)

```
#Python general Lib
import numpy,sys

#Personal Lib
import cosmo


#Lib for configfile
import configparser

##Initialize and retrieve the configfile
config_file=sys.argv[1]
config = configparser.ConfigParser()
config.read(config_file)

#retieve the parameters
Omega_L=config.getfloat('Cosmo', 'Omega_l')
Omega_M=1-Omega_L
HO=config.getfloat('Cosmo', 'Ho')
z=config.getfloat('Cosmo', 'z')
```

0 - sys module

1 - Import configparser

2- Retrieve the name of the configuration file

3- create a config object

4- read the configuration with the read method and define its entries

```
[Cosmo]
Omega_l=0.73
Ho=70
z=2.0
```

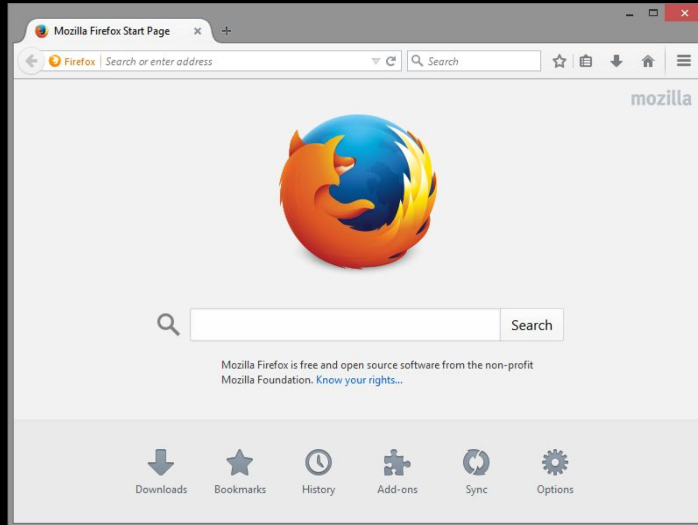
Configuration file

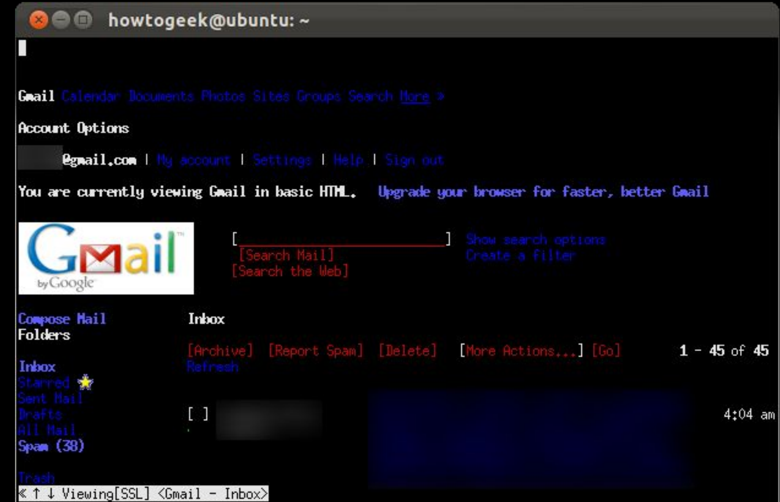
The list of command line arguments passed to a Python script comes from sys.argv. (argv[0] is the script name), argv[1] is the configuration file

[Advanced]

Let's make it more sexy…(in the geekiest way)

- Graphical Interface and Text Based Interface -





[Advanced] TUI and GUI: Some libraries

GUI:

PyQt (PySide) Tkinter

wxpython Kivy Pyforms

PyGi

TUI:

ncurses npyscreen

picotui urwid

And probably others!

[Advanced] TUI and GUI: Some libraries

GUI:

PyQt (PySide) Tkinter

wxpython Kivy Pyforms

PyGi

TUI:

ncurses npyscreen

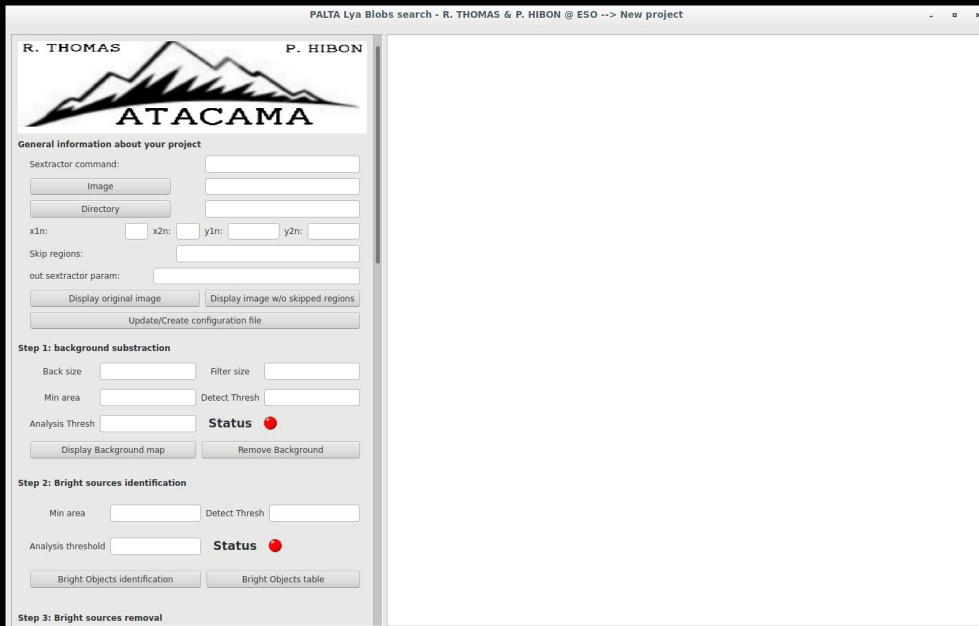
picotui urwid

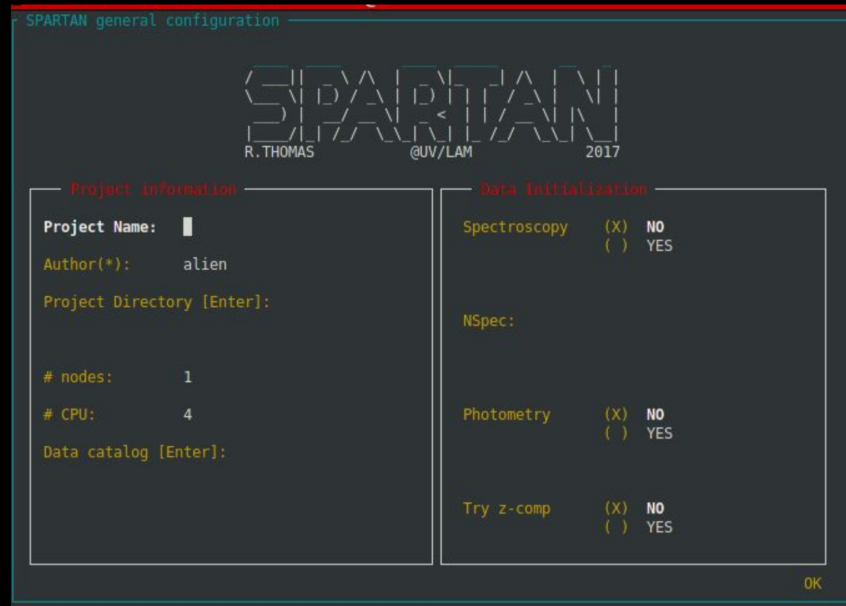
And probably others!

Basically TUI and GUI are the sames:

USER → Interface (widgets) → functions

You create an area that you populate with widgets:

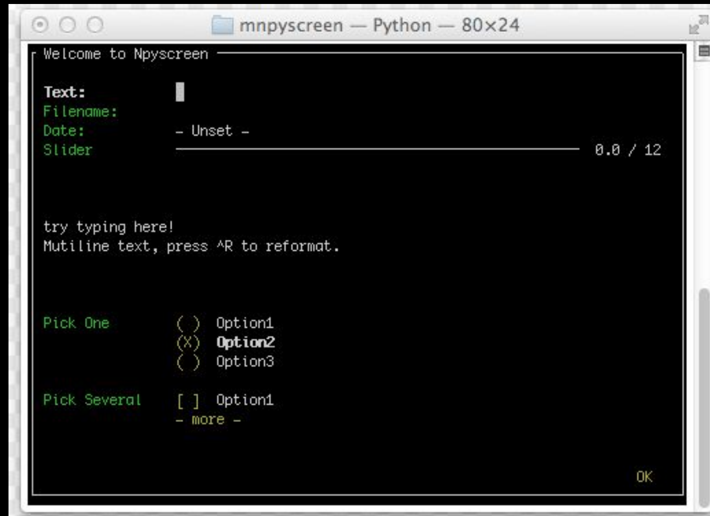




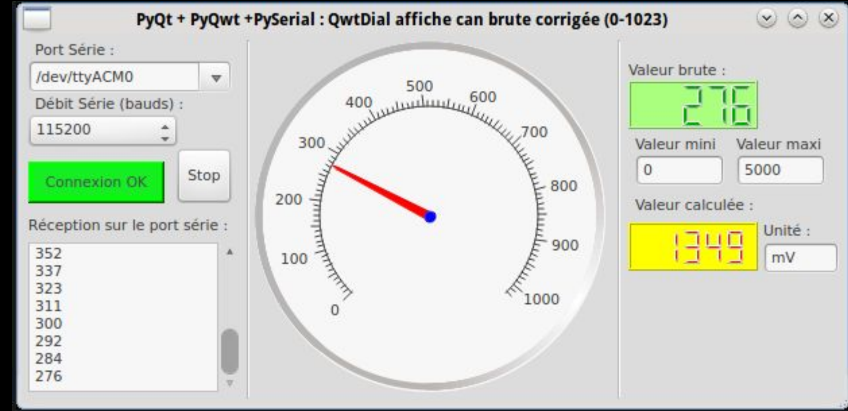
Widgets:

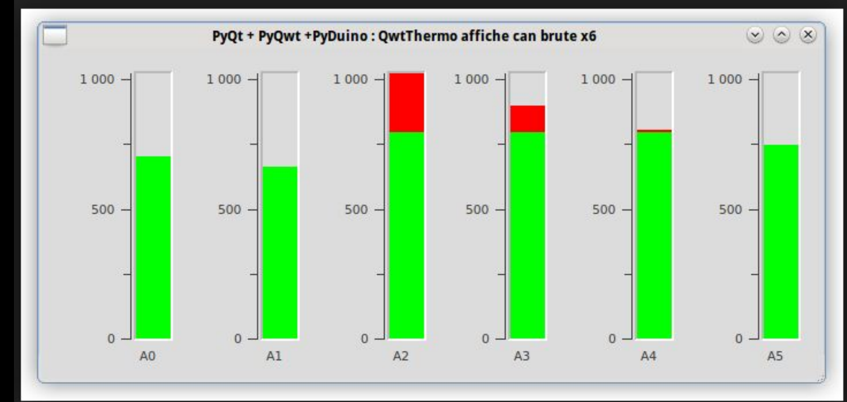
A lot of possibilities

For TUI they are pretty basic



For GUI, you can make fancy options!



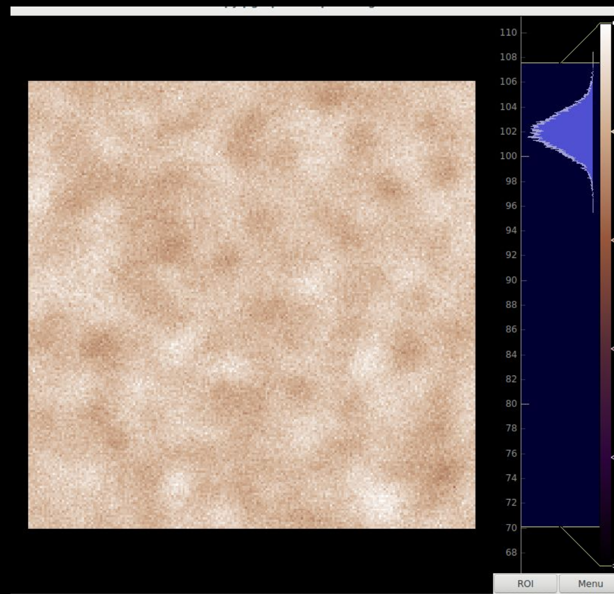


Widgets for science: PYQTGRAPH

Basic plotting examples

Basic array plotting
Multiple curves
Drawing with points
Parametric, grid enabled
Scatter plot, axis labels, log scale
Updating plot
Filled plot, axis disabled
Region Selection
Zoom on selected region

Provides a lot of possibilities for data visualisation



A bit of code… for TUI

```
import npyscreen as np

class myPop(np.NPSApp):

    def setopt(self, title, oList):
        self.title = title
        self.options = oList

    def main(self):
        F = np.Form(name="Choose an option")
        opt = F.add(np.TitleSelectOne, name=self.title, \
                max_height=len(self.options), values=self.options, scroll_exit=True)
        F.edit()
        self.return_this = opt.get_selected_objects()

def ChooseOption(title, oList):

    a = myPop()
    a.setopt(title, oList)
    a.run()
    return a.return_this

print(ChooseOption('fadsfas', ['qwqwqe', 'asdadd', 'zczczcx']))
```

```
Choose an option

fadsfas          ( )  qwqwqe
                 ( )  asdadd
                 ( )  zczczcx

                                                  OK
```

A bit of code… for GUI

```
import sys
from PyQt4 import QtGui


def main():

    app = QtGui.QApplication(sys.argv)

    w = QtGui.QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Simple')
    w.show()

    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

Create the application

Create the window

resize it (pixel)

place it on the screen (pixels)

Give a title

Display on the screen

Ensure a clean exit

And then you populate with widgets……….

One message to take away:

BE LAZY AND DON'T TOUCH A CODE THAT WORKS…
….CREATE INTERFACES

That's it! Thank you!

→ Import this ….