Coding For Paranal

OR

.... How to write a module ....

(.... And how we **try** to make it easy for you ....)

R. Thomas, ESOpy3.0, ESO - Vitacura

Current Status:

More than 60 scripts/codes/programs….

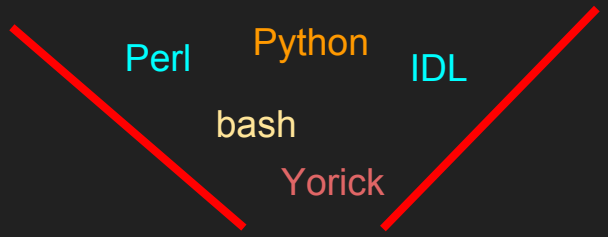
→ Some in Python, some in IDL, some in Perl, some bash, some in Yorick :(

→ Most of them are not commented/documented

→ Hard to maintain, hard to fix, hard to test, well, in other words it is not ideal

Our AimS:

1 - Cleaning Up



Python

→ SciOpsPy 18.10

SciOpsPy 19.10 with py3.7

2 - Merge

→ Some Functions are used for different instrument and should be handled by the same code

Our AimS:

3 - Gather everything into one *Master Software* . It is a Graphical User Interface

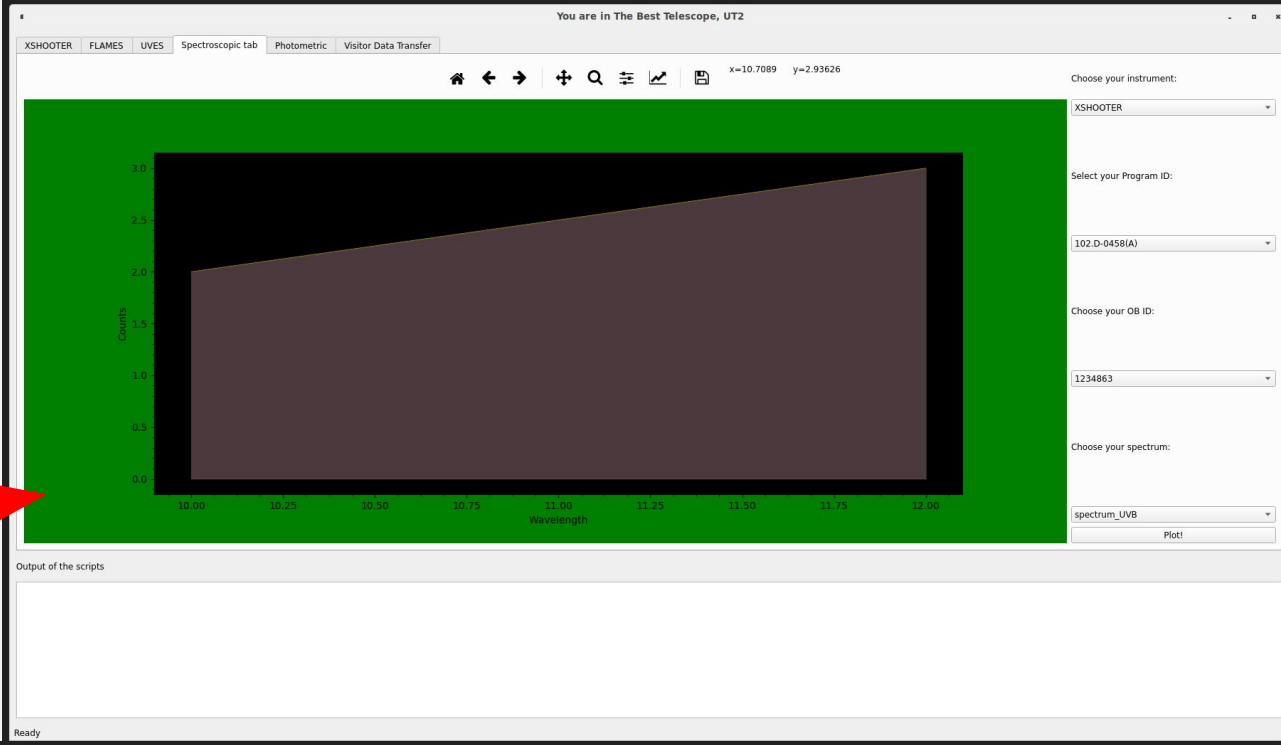
usage: pops [-h] Telescope

POPS: Panel for sciOPs, version 19.2.1, ---
Licence: ESO closed source ---
Authors: R. Thomas

optional arguments:
  -h, --help show this help message and exit

positional argument:
  Telescope Telescope: UT1, UT2,UT12 ,UT3 ,UT4,VLTI,VST,VISTA,CTA,ELT

pops UT2



We are **NOT** asking you to write the panel itself!

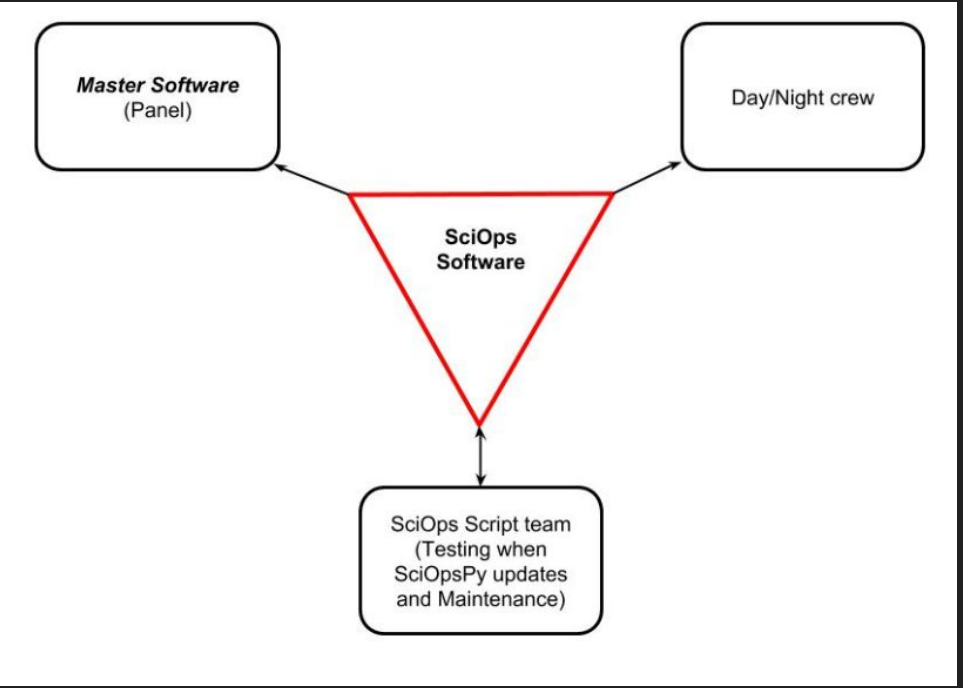
BUT:

This script should:

→ Be integrable in the Master Software Panel

→ Available from the Command

→ Testable



In Practice, How do we do that?

We prepared a Software Template
And we will ask you to use it when
you write softwares

→ This includes:

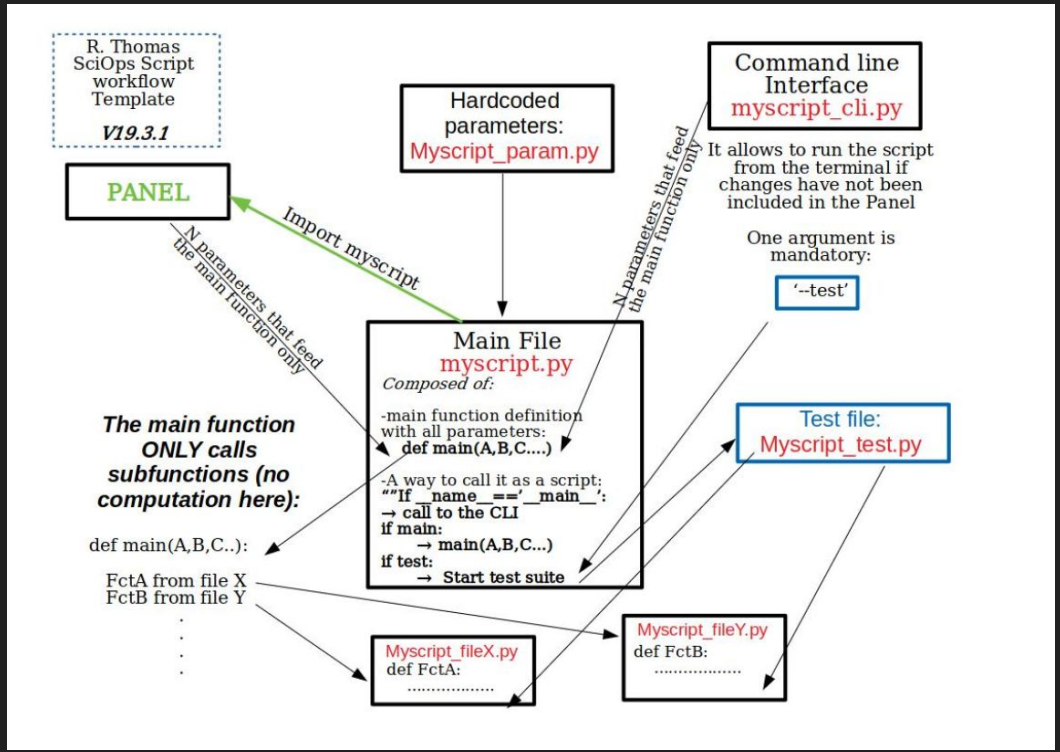
- The Global structure
- A command line interface
- Some mandatory library

In Practice, How do we do that?

We prepared a Software Template
And we will ask you to use it when you write softwares

→ This includes:

- The Global structure
- A command line interface
- Some mandatory library



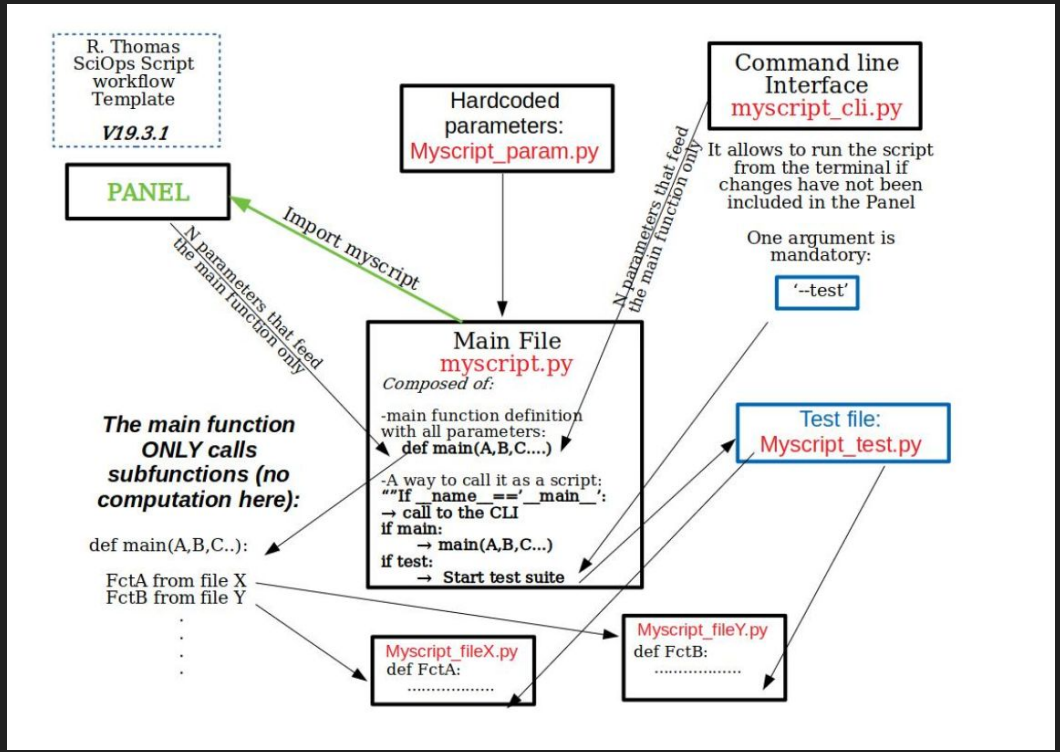
In Practice, How do we do that?

We prepared a Software Template

It is pre-coded and contains:

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py



SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

No computation in the main.py file!

```
1 from . import cli
2 from . import fctA      -Note that you might NOT need
3 from . import fctB      -to import everything!
4 from . import plots
5 from . import tests
```

```
1 def main(parameter1, parameter2, ismaster=False):
2     command1 =...
3     command2 =...
4     command3 =...
5     .....
6     display1 = ....
7     display2 = ....
8     ....
```

```
1 def as_script(): <-- No parameters
2     #call to the command line interface
3     #and extract the parameters
4     #then, if we call the test we start them
5     #otherwise we make a call to the main function
```

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

```
1 import argparse
2 parser = argparse.ArgumentParser(description='whatever')
```

Add all the necessary arguments

```
1 parser.add_argument('--par1', help='describes parameter1')
2 parser.add_argument('--par2', help='describes parameter2')
```

One argument is mandatory: '--test'

```
1 parser.add_argument('--test', help='Start tests', action = store_true)
```

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

The hardcoded parameters (values, directories, filename…..) should all be in the same file: *hardcoded.py*

```
1 ''' Hardcoded parameters file '''
2 par1 = 24
3 par2 = 54
4 file_name = test_file_name.txt
5 directory = /home/rthomas/xshooter
6 ..
7 ..
8 ..
```

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

The function files (name that you should change) are where you actually make what you want.
→ All the computation should go in one of these files

```
1  def myfunction(p1, p2, p3....):
2      '''
3      The documentation of the function defining inputs
4      and outputs
5      '''
6
7      some calculations here,
8      eventually call to other defined functions
9
10
11     return '<--only if it returns something'
```

The *plots.py* is where you make plots (and only plots!!!!!!!!!!!!!!!)

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

Tests → 2 options:

- Test suite: This means that each function of the code must be testable for all the cases. For this option, you can use the pytest module. This can take some time to write but you will learn a lot doing it. The pyptest module allows you to write test and give the expected result for each function for a given set of parameters. If the output is the expected one then the test suite goes on, if it is not, a warning is sent and explains what went wrong.

- Software run on test data: This option is the easy one. This means that once your code is ready, you select test data and attach them to your code. You must also provide the exact expected output of the code when it runs on the data (printouts, plots, files...). You are also in charge to provide enough test data to test the entire software.

SciOps_soft/

- Setup.py
- SciOps/
  - main.py
  - cli.py
  - hardcoded.py
  - fcA.py
  - fcB.py
  - plots.py
  - tests.py

Installation → setup.py

```
1 from setuptools import setup
2 setup(
3     name = 'nodule_template',
4     version = '19.3.1',
5     packages = ['module'],
6     entry_points = {'console_scripts':
7                         ['module_cmd = module.main:as_script',],}
8 )
```

You install the module placing yourself next to the top directory and typing:

pip install -e SciOps_soft

Check first that pip is the one of your environment!

```
SciOps_soft/
    Setup.py
    SciOps/
        main.py
        cli.py
        hardcoded.py
        fcA.py
        fcB.py
        plots.py
        tests.py
```

Installation → setup.py

Then you should be able to call the '*SciOps*' command directly from your terminal

Try: *SciOps --help*

And from the python interpreter with:

from SciOps import main as SciOps

Exercice :) → Let's modify the module

A - Modify the Command line interface with 2 optional arguments:

--e → int, default 3

--b → boolean

B - 3 hardcoded parameters: A=1 B=3 C=5

C - Create a function that return R with R = A*B*C/e

D - call that function from the main

E - if R> 5 plot an horizontal line at y = R

if R<5 plot a diagonal line at x = y = R