Python Tips & Tricks

Things I learnt during my PhD

Mike Laverick
Centre for eResearch
25/06/2020

Resources from the 25/06/20 session + chat

Presentation links

- https://uoa-eresearch.github.io/HackyHour/
- Stop using numpy.loadtxt: http://akuederle.com/stop-using-numpy-loadtxt
- Generators: https://realpython.com/introduction-to-python-generators/

Chat recommended links

- https://www.youtube.com/watch?v=8DvywoWv6fI&list=PLY9xW0dssvfYuTAVS7eNoqhLdcsu291LI
- https://colab.research.google.com/
- https://runestone.academy/runestone/books/published/thinkcspy/index.html
- https://numpy.org/doc/stable/user/numpy-for-matlab-users.html
- http://swcarpentry.github.io/python-novice-gapminder/
- https://exercism.io/tracks/python
- https://online-learning.harvard.edu/course/using-python-research

What’s this session about?

- Providing some tips & tricks I picked up during my PhD
- Not a formal seminar or course / not a Python 101 or advanced session
- A quick ~25 minute presentation, followed by a 20-30 minute chat + Q&A
- Goal(s) : - Bring people together to discuss their work + coding difficulties
  - Provide a (slack) space for people to share their coding knowledge
  - To help save you some Python coding time and effort!

Tip #1

Use a package/ environment manager

Use a package/environment manager

- Have you ever had to install/upgrade/downgrade Python modules to run a piece of code?
- Do you need to share code with other collaborators?
- Have you ever come back to some old code and can’t get it to run properly again?
- Do you need to keep switching between Python 2 and Python 3 ???

Use a package/environment manager

Package/environment managers allow you to easily control which Python modules and which version of that module are installed and available

Environment managers allow you to create groups of modules that you can activate/enable for a given session.

They also allow you to import/export these “environments” for others to use or to come back to later

Use a package/environment manager

There are a few to choose from:

Pip	-	Module installation
Virtualenv	-	Module installation, Environment manager
Anaconda	-	Module installation, Environment manager
Pyenv	-	“Environment manager” Manager

My personal preference is Anaconda, but they all do the job!

Tip #2

If you can’t do it (quickly) in Python, wrap Python around it

If you can’t beat them, join them!

Python is a “powerful” language in terms of it’s flexibility, support, modules

Toe-to-toe: cannot compete with Fortran/C in terms of pure computational speed

Fortunately, Python is a great “wrapper” language!

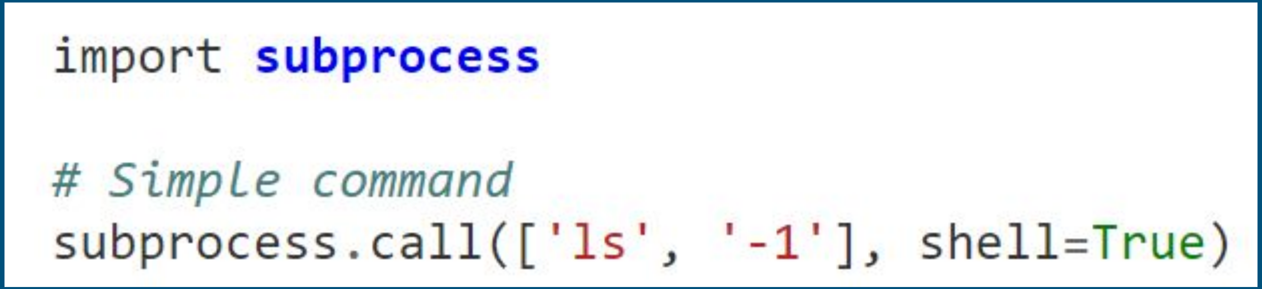
If you c

Python is a “p odules

Toe-to-toe: ca tional speed

WITH GREAT POWER COMES GREAT ~~RESPONSIBILITY~~
Flexibility
Speed up
BUT PROBABLY NOT A ~~PAYRISE~~
memegenerator.net

Use subprocess module to run command line



```python
import subprocess

# Simple command
subprocess.call(['ls', '-1'], shell=True)
```

1) You can execute other scripts/codes/languages/programs
2) Provide inputs + receive outputs
3) Spawn multiple subprocesses to work in the background

And much much more! Make sure to check it out!

Tip #3

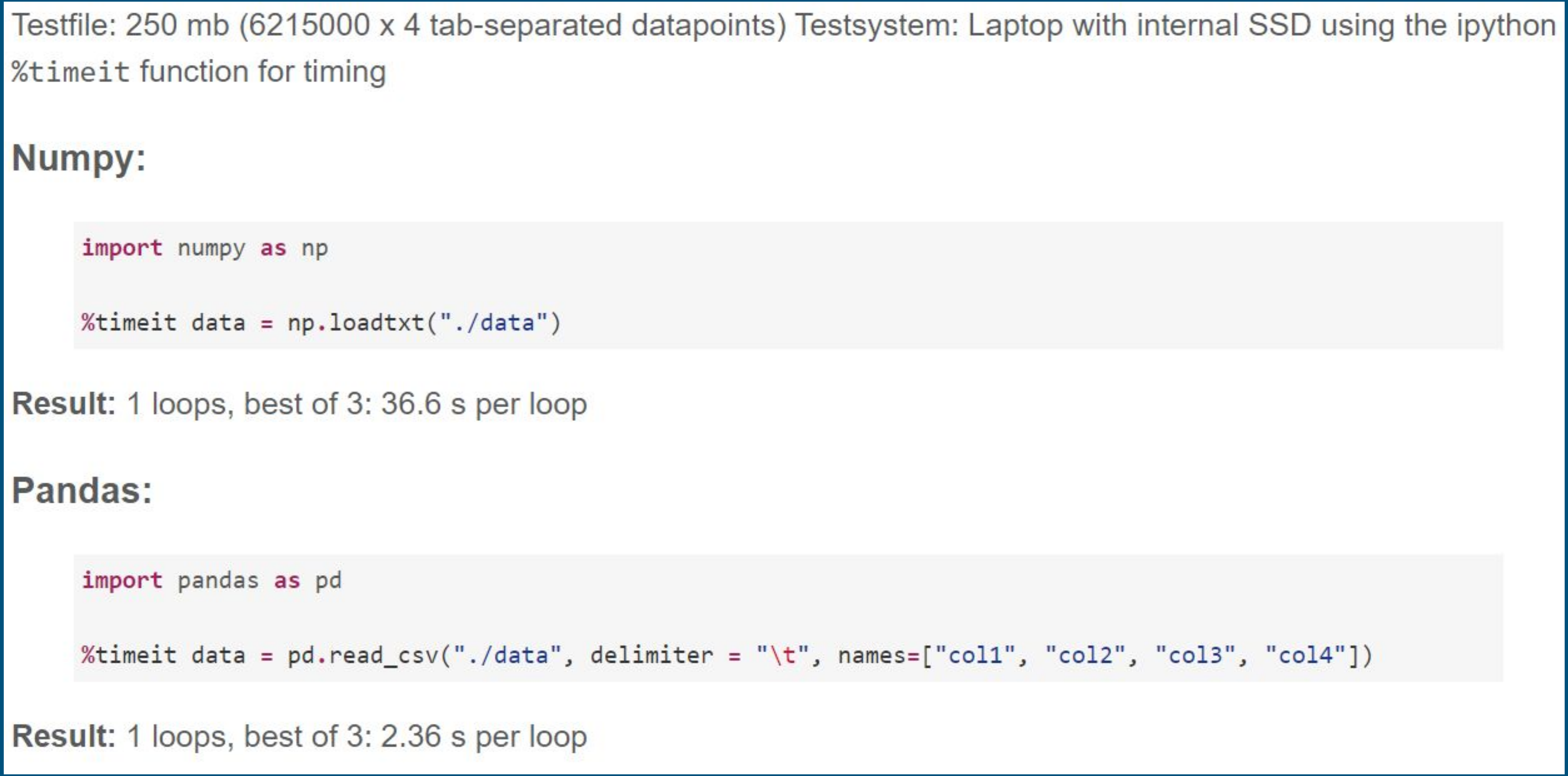
I/O speedups

I/O speedups

There are many great and useful ways and modules to help load in your data:

Pandas, json, csv, to name but a few (there are many bespoke research file types)

numpy.loadtxt is not one of them!



Testfile: 250 mb (6215000 x 4 tab-separated datapoints) Testsystem: Laptop with internal SSD using the ipython `%timeit` function for timing

Numpy:

```
import numpy as np

%timeit data = np.loadtxt("./data")
```

Result: 1 loops, best of 3: 36.6 s per loop

Pandas:

```
import pandas as pd

%timeit data = pd.read_csv("./data", delimiter = "\t", names=["col1", "col2", "col3", "col4"])
```

Result: 1 loops, best of 3: 2.36 s per loop

Example taken from http://akuederle.com/stop-using-numpy-loadtxt

I/O speedups: other solutions

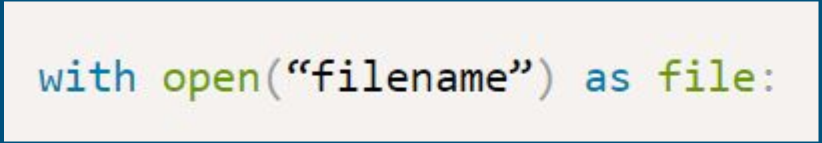
Save data in binary formats for quick re-loading in the future:

numpy.load() on a compiled .npy array is far quicker than numpy.loadtxt()

Or even use the amazing Pickle module to save arbitrary data structures/objects for future reloading.   (Warning: does not work well with matplotlib, sorry!)

Final note on safe file reading/writing

Use “with open(“file”) as f:” instead of “open()”



```
with open(“filename”) as file:
```

You don’t need to explicitly close the file this way

Far safer if program crashes

Tip #4

Python is object-oriented, use objects!

Python is object-oriented, use objects

This point is very generic, and far too long to really cover properly here

But speaking from experience, a little time investment to switch to Object-oriented thinking goes a long way in fast + easy Python

Python is object-oriented, use objects

You can put anything in a list: even functions, classes, and modules

Use dictionaries: being able to call your data using keywords rather than index numbers is far easier for a human to remember, and can save needless iterations

Use classes (even!): You might not need to use these in your research, but they are powerful for defining and creating sets of variables and methods, with both common and instance-specific values/attributes

(Think of the common and unique properties that humans have... Now as a set of Python variables)

Tip #5

Efficiency ~~Efficiency, Efficiency~~

Efficiency: a few examples w.r.t loops

Loops are unavoidably important for coding, yet often avoidably slow

Nested loops are particularly bad; time scales dramatically per extra loop

Fortunately, you can often speed up loops through sensible design

Efficiency: a few examples w.r.t loops

Think of how you structure your code:

- do I need to declare this variable in every iteration?
- Are there Built-in/Numpy/Scipy functions to do this more efficiently?

Don’t forget:

- for/while loops: you can use pass, continue, and break

  My nested loop from 2 weeks to 30s thanks to these functions!

Efficiency: a few examples w.r.t loops

Do you need to loop over Gigabytes of input data?

Use a generator!

Normal loops need to load the entire list/array/object in memory before looping

Generators load on the fly, reducing RAM and speeding up iteration over large data

Check out this link: https://realpython.com/introduction-to-python-generators/

That’s all folks!

Now to hear from you…

Bonus Tip

How to write your code neatly

Get a nice text editor + python linter + PEP8 style

Rule of thumb: if PEP8 makes it less readable - ignore PEP8