Scientific Python

Neelofer Banglawala nbanglaw@epcc.ed.ac.uk
Kevin Stratford kevin@epcc.ed.ac.uk

Original course authors:
Andy Turner
Arno Proeme

Reusing this material

www.archer.ac.uk

support@archer.ac.uk

[Intro] Course overview

Course website:

http://archer.ac.uk/training/courses/2015/11/SciPy_Imp/

You will need:

1. Access to Python (+ packages) & software tools:
   i. your own laptop
   ii. ARCHER
   see ScientificPython_PreparatoryCourseInformation.pdf
1. Course material
   wget URL

Please make sure you have signed in and have a Guset Account sheet

Time	Day One	Day Two
09:30	Registration	-
10:00	L01 - Introduction/Getting Started	L04 - SciPy
11:00	Break (tea/coffee)	Break (tea/coffee)
11:30	L02 - NumPy	L05 - Fortran/C Interface
13:00	Lunch (not provided)	Lunch (not provided)
14:00	L03 - Matplotlib	Exercises
15:30	Break (tea/coffee)	Break (tea/coffee)
16:00	Exercises	Exercises
17:00	Close	Close

[Intro] Scientific computing

Typical workflow

- Generate data
  - Usually from simulation on HPC facilities
  - (also from experiment!)
- Process data
- Generate appropriate results from data
- Visualise results
  - To understand the significance of our work and gain scientific understanding
- Communicate results
  - Through publications, presentations, web, etc.

[Intro] Why Python? I

- Rich set of scientific computing functionality
  - Powerful numerical and scientific libraries
  - Rich plotting functionality
  - Excellent support for interfacing to existing Fortran/C/C++ code, use Python as "glue"
  - Can create unified, multipurpose workflow environment for data analysis and visualisation
- Python is a high level, easy-to-read language
  - Simple to learn and code in
  - High level syntax means more time spent thinking about what code does rather than how to write it
  - Fully-featured, general purpose language that supports different programming styles, e.g. object-oriented

[Intro] Why Python? II

- Interactive interface as well as (non-interactive) scripting
  - Enables rapid prototyping of algorithms
- Free and Open Source
  - Can contribute to Scientific Python ecosystem
- Viable alternative to Matlab and similar
- Using interactive Python, especially iPython, is similar to using other scripting languages e.g. Matlab, Mathematica, Maple, R, etc.

As popularity grows more and more packages become available, Python is increasingly the go-to choice for tying everything together

Useful links

- https://www.python.org/
- https://wiki.python.org/moin/NumericAndScientific

[Intro] Core packages for scientific computing

- IPython
  - Advanced Python shell
- NumPy
  - Tools for manipulating numerical arrays efficiently
- Matplotlib
  - Rich featured plotting in 2D and 3D
- SciPy
  - High-level scientific routines for common algorithms e.g. numerical integration, optimisation, fourier transforms
- f2py
  - Interface Fortran/C external code with Python
  - Comes with NumPy

[Intro] Other useful packages

- mpi4py
  - message passing parallel programming
  - covered in future versions of this course
- pandas
  - data analysis library
- scikit-learn
  - machine learning

... and many more ...

[Intro] Python on ARCHER I

ARCHER is a Cray XC30 MPP supercomputer with 4920 compute nodes (the backend), 8 login nodes and 2 post-processing (pp) nodes (the frontend).

By default, Python is not loaded on ARCHER

Need to choose a Python distribution according to where you will run your code:

- **Anaconda** : for the frontend i.e. login nodes, pp nodes
  - *module load anaconda*
  - contains core scientific computing packages such as NumPy
- **native** : for the backend i.e. compute nodes
  - *module load python-compute*
  - need to separately load packages such as numpy, *module load pc-numpy*
- http://archer.ac.uk
- http://www.archer.ac.uk/documentation/user-guide/python.php

[Intro] Python on ARCHER II

Log into ARCHER and load Python:

- In a command terminal, log into the ARCHER frontend (login nodes):
  - *ssh -X username@login.archer.ac.uk*
- Check your current location with:
  - *pwd*
- Create a directory for this course and navigate to it:
  - *mkdir <directory_name >*
  - *cd <directory_name>*
- Load the Anaconda distribution:
  - *module load anaconda*
- If you wish to use ARCHER for the remainder for the course, get the course material:
  - wget URL

[Intro] How to use Python?

- Python code is not generally compiled into a standalone executable, but executed by the Python interpreter : **python**
- **Non-interactive mode** : supply the Python interpreter with an input script file
  - Python script files end in *.py* extension
  - *~$ python myscript.py*
- **Interactive mode** run Python interpreter without an input script file
  - Interpreter runs as a Python shell (interactive Python runtime environment)
- **Alternative interactive modes**
  - IPython : enchanced Python shell, ideal for data manipulation and visualisation
  - Jupyter (formerly IPython) notebook : web browser-based interactive document

[Intro] IPython shell

IPython extends the standard Python shell with a number of useful things, including

- Tab completion object_name.
- Getting help, ? object_name?
- 'Magic' commands %run (run python script within shell)
- History of commands %hist
- Saving of sessions %save
- System shell access, ! files = !ls -l
- Pasting of code snippets from websites
- Built-in debugging and profiling

quickref command gives a summary of capabilities

http://ipython.readthedocs.org/en/stable/overview.html

[Intro] `Hello ARCHER!`

Create 'helloarcher.py', use editors 'vim', 'emacs', 'nano' or 'gedit':

```
#!/usr/bin/env python


# Hello ARCHER python program


print "Hello ARCHER!";
```

</p>

1. Run '*helloarcher.py*' : **python** *helloarcher.py*

2. Launch an IPython shell and type : **print** "Hello ARCHER!"

3. Run the script from within an IPython shell using 'magic' : **%run** *helloarcher.py*

[Intro] Python basics recap I

- Data types
  - integer (-1), float (3.1412), string ('this' or "this")
  - dynamically typed: don't explicitly specify datatype when declaring variable
- Data structures
  - list ([3,"a",3.14,False]), dictionary ({"key1" : "value1", "key1" : "value1"}), tuples ((1,2,3) or (1.2,) or ( ))
- Whitespace matters : code blocks are indented
  - **for** item **in** list:
    #do some stuff
- Import modules:
  - import module --> module.name
  - from module import name` --> call name
  - avoid universal imports i.e. from module import *

[Intro] Python basics recap II

- Careful - variables are references
  - variables are references to objects: let a = 3, b = a; if b = 5 then a = 5 (try it!)
- Functions
  - functions belonging to one object type accessed with dot operator: **object.method** e.g. list.sort()
  - functions applicable to multiple object types don't use dot operator e.g. len("string"), len(list)
  - can define your own function :

    def *myfunc(arg1, arg2)***:**
    function body is indented
    return *something*
- Misc
  - Many useful **built-in functions** such as *max*, *min*, *reverse*
  - Useful modules in the **Standard Library** such *sys*, *math*
- Documentation
  - https://docs.python.org/2/
  - https://www.codecademy.com/learn/python

[Intro] Warm-up exercise I

Define a function **age** that takes a list of years between 1950 and 2014 and returns the median age, and the two ages closest to it. Assume the input list is randomly ordered and has an odd number of elements N, where $N >= 3$. So for:

years = [1989, 1955, 2011, 1943, 1975], **age** returns [26, 40, 60]

Note: for a sorted list of numbers, the median is the number in the middle of the list.

1. You will need to generate a list of years.

Hint: you may want to use *random.randint(start, stop)*

2. You will need to sort the list.

3. List indexing may help you to get the final result e.g. list[3:5]

4. Make sure to test your function.

[Intro] Warm-up exercise (a solution)

In [ ]:
```
# function to calculate the median age and
# its two neighbours from a random list of
# years
import numpy as np

def medianage(years):
    ages=[];
    for y in years:
        ages.append(2015-y);
    ages.sort()
    n = len(ages);
    mid = n/2;
    return ages[mid-1:mid+2];


years = [1989, 1955, 2011, 1943, 1975];
medianage(years);
```

[Intro] Warm-up exercise II (optional)

Now read the list of years from a text file, 'years.txt', which should have the total number of years N in the first line, followed by a numbered list of years:

total number of years
1 year
...

To generate the input file 'years.txt', you may need:

import sys, yearsfn=open(filename, "w"), input.close(), output.write("{0:2d} {1:2d}\n".format(i, j))

To read the input file 'years.txt', you may need:

input=open(filename, "w"), *line = infile.readline(), line.rstrip(), line.split(), int("9")*,

Optional : could you use list comprehension (if you haven't already)?
E.g. squared = [x*x for x in list]

[Intro] Summary

We have reviewed some core Python basics

We have also been introduced to the IPython shell

We are now ready to look at explore the packages that constitute the backbone of scientific python

Next session : NumPy