iPython

The SciPy Stack

SciPy is a Python-based ecosystem of libraries and tools for scientific computing and data analytics

- iPython
- Jupyter notebooks
- Numpy
- Pandas
- Matplotlib

iPython is the primary way of interacting with the SciPy stack – whether through the shell or a Jupyter notebook. It's also a fantastic REPL.

iPython

Two modes:

- ▶ Interactive shell
  - ▶ Replacement for `python` REPL
- ▶ Jupyter notebook
  - ▶ Interactive web-based documents mixing text, executable code, graphics

Before we proceed, make sure your computer is ready (OS shell):

```
1 $ pip install ipython
```

iPython Shell History

```
1   In [1]: ['Sage', 'Thyme', 'Oragano', 'Posh']
2   Out[1]: ['Sage', 'Thyme', 'Oragano', 'Posh']
3
4   In [2]: type(In[1])
5   Out[2]: str
6
7   In [3]: type(Out[1])
8   Out[3]: list
9
10  In [4]: spices = Out[1]
11
12  In [5]: spices
13  Out[5]: ['Sage', 'Thyme', 'Oragano', 'Posh']
14
15  In [6]: spices is Out[1]
16  Out[6]: True
```

`In` is a list, `Out` is a dict.

iPython Help

Single `?` gives abbeviated version of python's `help`

```
1  In [7]: def add(a, b):
2     ...:     """Return the result of + operation on a and b"""
3     ...:     return a + b
4     ...:
5  In [8]: add?
6  Signature: add(a, b)
7  Docstring: Return the result of + operation on a and b
8  File:      `/cs2316/<ipython-input-7-af5293282e78>
9  Type:      function
```

Double `??` gives source code, if available.

```
1  In [9]: add??
2  Signature: add(a, b)
3  Source:
4  def add(a, b):
5      """Return the result of + operation on a and b"""
6      return a + b
7  File:      `/cs2316/<ipython-input-7-af5293282e78>
8  Type:      function
```

iPython Magic Commands

Special commands provided by iPython, prepended by %.

▶ Run a Python script from within iPython:

```
1  In [35]: %run people.py
2  [<Stan, 2008-08-13, 150cm, 45kg>,
3   <Kyle, 2008-02-25, 160cm, 50kg>,
4   <Cartman, 2008-05-26, 140cm, 100kg>,
5   <Kenny, 2009-07-30, 130cm, 40kg>]
```

▶ Get help with a magic command with ?

```
1   In [2]: %cd?
2   Docstring:
3   Change the current working directory.
4
5   (content elided)
6
7   Usage:
8
9     cd 'dir': changes to directory 'dir'.
10  (additional output elided)
```

Get a list of all magic commands with %lsmagic

iPython Shell Commands

Run shell commands by prepending with a `!`

```
1  In [27]: !ls *.py
2  fun.py      grades.py  maths.py   people.py  pp.py
3
4  In [28]: pyscripts = !ls *.py
5
6  In [29]: pyscripts
7  Out[29]: ['fun.py', 'grades.py', 'maths.py', 'people.py', 'pp.py']
```

iPython provides magic commands for most common shell commands.

iPython Direcotry Bookmarking

Great timesaving feature: bookmark directories

```
1   In [3]: %pwd
2   Out[3]: '/home/chris/vcs/github.com/cs2316/cs2316.github.io/code'
3
4   In [4]: %cd
5   /home/chris
6
7   In [5]: %bookmark cs2316code
          `chris/vcs/github.com/cs2316/cs2316.github.io/code
8
9   In [6]: cd cs2316code
10  (bookmark:cs2316code) ->
          `chris/vcs/github.com/cs2316/cs2316.github.io/code
11  /home/chris/vcs/github.com/cs2316/cs2316.github.io/code
```

iPython Automagic commands

With `automagic` turned on, some shell commands can be run as if they were built into iPython:

```
1   In [22]: pwd
2   Out[22]: '/Users/chris/cs2316'
3
4   In [23]: ls *.py
5   fun.py     grades.py maths.py  people.py pp.py
```

- ▶ Toggle automagic on and off with `%automagic`.
- ▶ These commands work with automagic:
  - ▶ `%cd`, `%cat`, `%cp`, `%env`, `%ls`, `%man`, `%mkdir`, `%more`, `%mv`, `%pwd`, `%rm`, and `%rmdir`

Timing Code in iPython

```
In [23]: import numpy as np

In [24]: pylist = list(range(1, 100000))

In [25]: nparray = np.arange(1, 1000000)

In [35]: %timeit _ = [x * 2 for x in pylist]
100 loops, best of 3: 7.89 ms per loop

In [37]: %timeit _ = nparray.copy() * 2
100 loops, best of 3: 3.76 ms per loop
```

Notice that I copied the Numpy array before applying the * 2 operation to make the comparison to the Python list comprehension fair. You'll learn why when we discuss Numpy in the next lecture.

Profiling a Script

```
1  In [7]: %run -p -l 10 -s cumulative funcalc.py
2          2673375 function calls (1147466 primitive calls) in 1.691
                  seconds
3
4     Ordered by: cumulative time
5     List reduced from 56 to 10 due to restriction <10>
6
7     ncalls tottime percall cumtime percall filename:lineno(function)
8        2/1    0.000    0.000    1.691    1.691 {built-in method
               builtins.exec}
9          1    0.000    0.000    1.691    1.691 <string>:1(<module>)
10         1    0.000    0.000    1.691    1.691
                  interactiveshell.py:2431(safe_execfile)
11         1    0.000    0.000    1.691    1.691 py3compat.py:182(execfile)
12         1    0.000    0.000    1.690    1.690 funcalc.py:1(<module>)
13         1    0.000    0.000    1.689    1.689 funcalc.py:46(main)
14         1    0.039    0.039    1.689    1.689 funcalc.py:34(profile)
15  510961/10000    0.510    0.000    0.603    0.000 funcalc.py:14(sub)
16  510961/10000    0.514    0.000    0.598    0.000 funcalc.py:6(add)
17  510961/10000    0.340    0.000    0.340    0.000 funcalc.py:22(mult)
```

- ▶ `-p` means profile
- ▶ `-l 10` means show only 10 lines
- ▶ `-s cumulative` means sort by cumulative time

Profiling a Function

`%prun` profiles a function. Uses same options as `% run -p`.

```
In [10]: %prun -l 10 -s cumulative funcalc.profile()
         2673429 function calls (1148052 primitive calls) in 1.726
              seconds

   Ordered by: cumulative time
   List reduced from 15 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    1.726    1.726 {built-in method
              builtins.exec}
        1    0.000    0.000    1.726    1.726 <string>:1(<module>)
        1    0.042    0.042    1.726    1.726 funcalc.py:34(profile)
511231/10000  0.537    0.000    0.620    0.000 funcalc.py:6(add)
511231/10000  0.523    0.000    0.615    0.000 funcalc.py:14(sub)
511231/10000  0.336    0.000    0.336    0.000 funcalc.py:22(mult)
    20000    0.019    0.000    0.097    0.000 random.py:223(randint)
   501231    0.092    0.000    0.092    0.000 funcalc.py:15(dec)
   501231    0.082    0.000    0.082    0.000 funcalc.py:7(inc)
    20000    0.036    0.000    0.078    0.000 random.py:179(randrange)
```