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

```
In [1]: ['Sage', 'Thyme', 'Oragano', 'Posh']
Out[1]: ['Sage', 'Thyme', 'Oragano', 'Posh']

In [2]: type(In[1])
Out[2]: str

In [3]: type(Out[1])
Out[3]: list

In [4]: spices = Out[1]

In [5]: spices
Out[5]: ['Sage', 'Thyme', 'Oragano', 'Posh']

In [6]: spices is Out[1]
Out[6]: True
```

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

iPython Help

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

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

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

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

iPython Magic Commands

Special commands provided by iPython, prepended by `%`.

- Run a Python script from within iPython:

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

- Get help with a magic command with `?`

```
In [2]: %cd?
Docstring:
Change the current working directory.

(content elided)

Usage:

  cd 'dir': changes to directory 'dir'.
(additional output elided)
```

Get a list of all magic commands with `%lsmagic`

iPython Shell Commands

Run shell commands by prepending with a `!`

```
In [27]: !ls *.py
fun.py       grades.py   maths.py    people.py   pp.py

In [28]: pyscripts = !ls *.py

In [29]: pyscripts
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

```
In [3]: %pwd
Out[3]:
    '/home/chris/vcs/github.com/cs2316/cs2316.github.io/code'

In [4]: %cd
/home/chris

In [5]: %bookmark cs2316code
    `chris/vcs/github.com/cs2316/cs2316.github.io/code

In [6]: cd cs2316code
(bookmark:cs2316code) ->
    `chris/vcs/github.com/cs2316/cs2316.github.io/code
/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:

```
In [22]: pwd
Out[22]: '/Users/chris/cs2316'

In [23]: ls *.py
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

```
In [7]: %run -p -l 10 -s cumulative funcalc.py
         2673375 function calls (1147466 primitive calls) in 1.691 seconds

   Ordered by: cumulative time
   List reduced from 56 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      2/1    0.000    0.000    1.691    1.691 {built-in method builtins.exec}
        1    0.000    0.000    1.691    1.691 <string>:1(<module>)
        1    0.000    0.000    1.691    1.691 interactiveshell.py:2431(safe_execfile)
        1    0.000    0.000    1.691    1.691 py3compat.py:182(execfile)
        1    0.000    0.000    1.690    1.690 funcalc.py:1(<module>)
        1    0.000    0.000    1.689    1.689 funcalc.py:46(main)
        1    0.039    0.039    1.689    1.689 funcalc.py:34(profile)
510961/10000    0.510    0.000    0.603    0.000 funcalc.py:14(sub)
```

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)
```