Joint Hall A&C Data Analysis Workshop

Introduction to Python

Eric Pooser

Jefferson Lab

06/25/2018

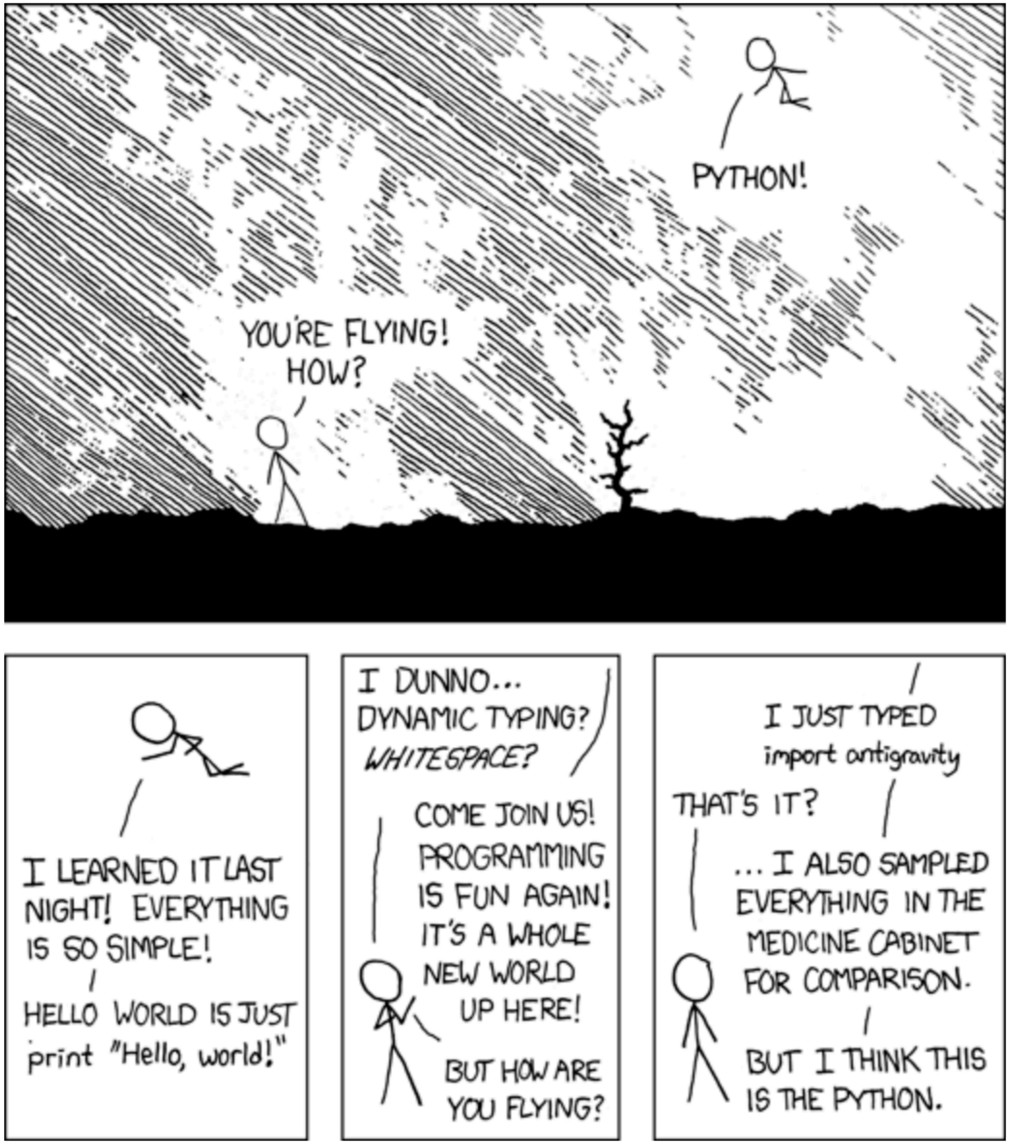
Jefferson Lab

U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

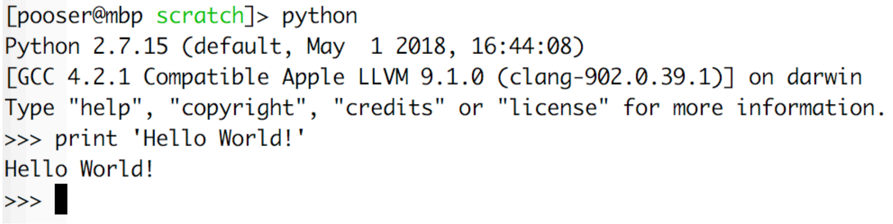
What is Python?

- Dynamic, high level programming language
- Interpreted & object oriented
- Powerful standard library
- Suited for a multitude of tasks
- Simple, easy to learn syntax
- Supports modules and packages

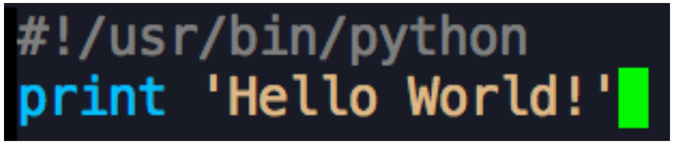

YOU'RE FLYING!
HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Why Python?

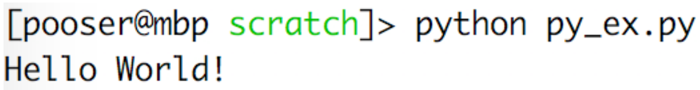
- Python’s object oriented structure supports such concepts as polymorphism, operation overloading, and multiple inheritance
- Indentation yields tidy, easily readable code
- Automatic memory management
- Vast array of third party utilities (NumPy, SciPy, Numeric, etc.)
- Built in types and tools
- Runs on virtually every major platform used today
- Python programs run in exact same manner irrespective of platform



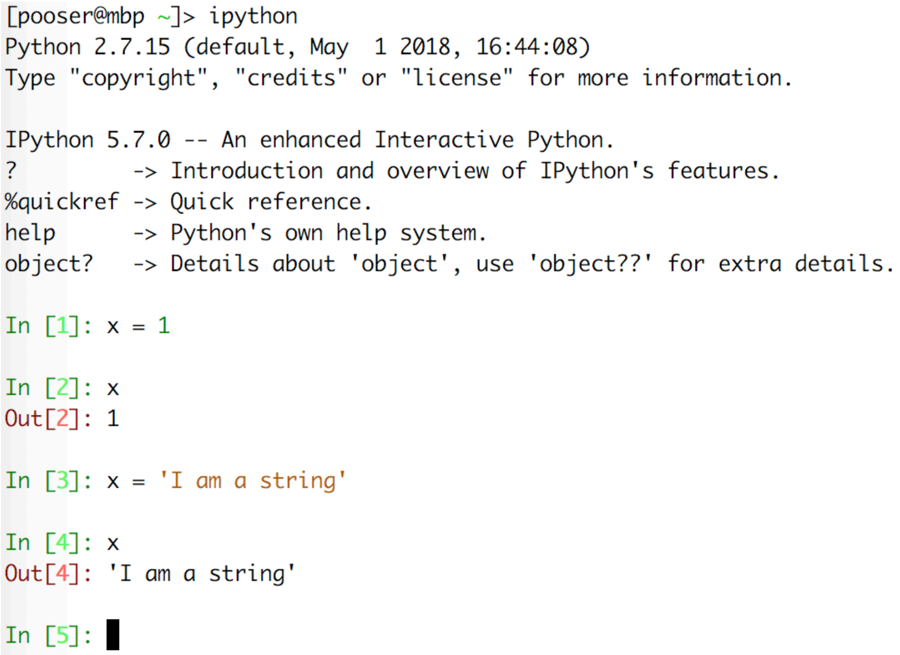
```
[pooser@mbp scratch]> python
Python 2.7.15 (default, May  1 2018, 16:44:08)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello World!'
Hello World!
>>> 
```



```
#!/usr/bin/python
print 'Hello World!'
```



```
[pooser@mbp scratch]> python py_ex.py
Hello World!
```

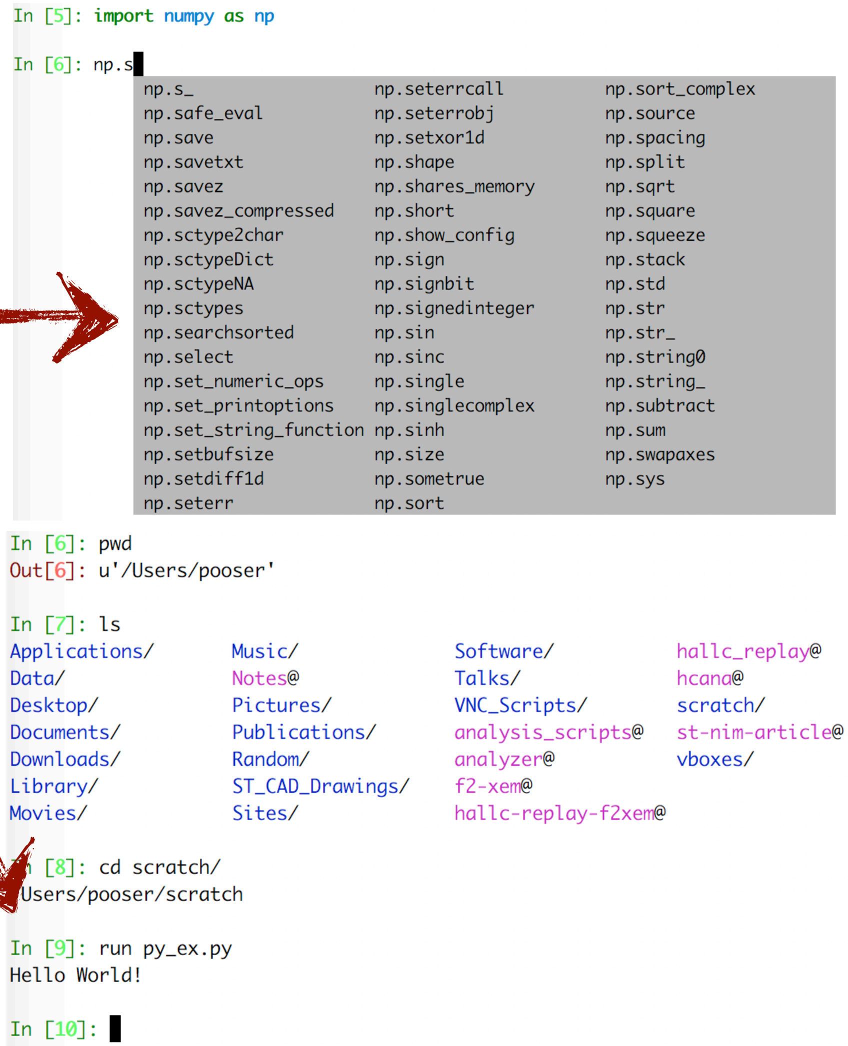


```
[pooser@mbp ~]> ipython
Python 2.7.15 (default, May  1 2018, 16:44:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.7.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: x = 1

In [2]: x
Out[2]: 1

In [3]: x = 'I am a string'

In [4]: x
Out[4]: 'I am a string'

In [5]: 
```

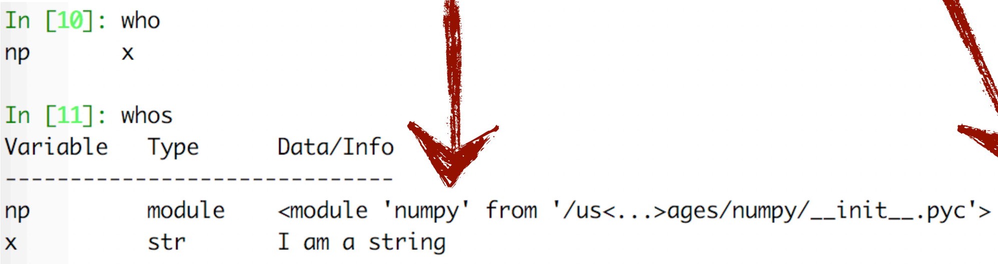
iPython

- A powerful interactive shell
- A kernel for Jupyter
- Support for interactive data visualizations and use of GUI toolkits
- Provides auto-complete feature with modules and utilities
- Supports most Linux commands
- Run python scripts interactively
- Tracks variable definitions

IP[y]: IPython Interactive Computing



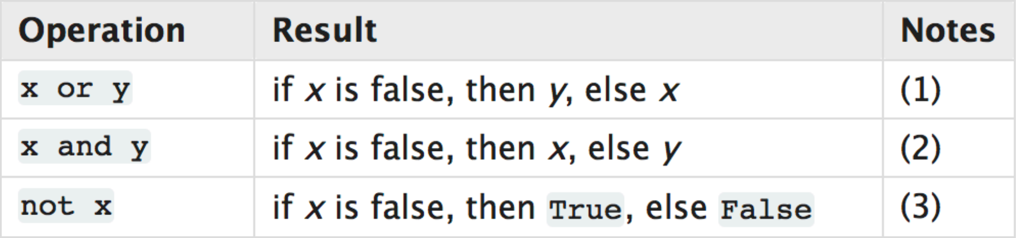
```
In [5]: import numpy as np

In [6]: np.s
        np.s_                   np.seterrcall           np.sort_complex
        np.safe_eval            np.seterrobj            np.source
        np.save                 np.setxor1d             np.spacing
        np.savetxt              np.shape                np.split
        np.savez                np.shares_memory        np.sqrt
        np.savez_compressed     np.short                np.square
        np.sctype2char          np.show_config          np.squeeze
        np.sctypeDict           np.sign                 np.stack
        np.sctypeNA             np.signbit              np.std
        np.sctypes              np.signedinteger        np.str
        np.searchsorted         np.sin                  np.str_
        np.select               np.sinc                 np.string0
        np.set_numeric_ops      np.single               np.string_
        np.set_printoptions     np.singlecomplex        np.subtract
        np.set_string_function  np.sinh                 np.sum
        np.setbufsize           np.size                 np.swapaxes
        np.setdiff1d            np.sometrue             np.sys
        np.seterr               np.sort

In [6]: pwd
Out[6]: u'/Users/pooser'

In [7]: ls
Applications/      Music/             Software/          hallc_replay@
Data/              Notes@             Talks/             hcana@
Desktop/           Pictures/          VNC_Scripts/       scratch/
Documents/         Publications/      analysis_scripts@  st-nim-article@
Downloads/         Random/            analyzer@          vboxes/
Library/           ST_CAD_Drawings/   f2-xem@
Movies/            Sites/             hallc-replay-f2xem@

In [8]: cd scratch/
/Users/pooser/scratch

In [9]: run py_ex.py
Hello World!

In [10]: 
```



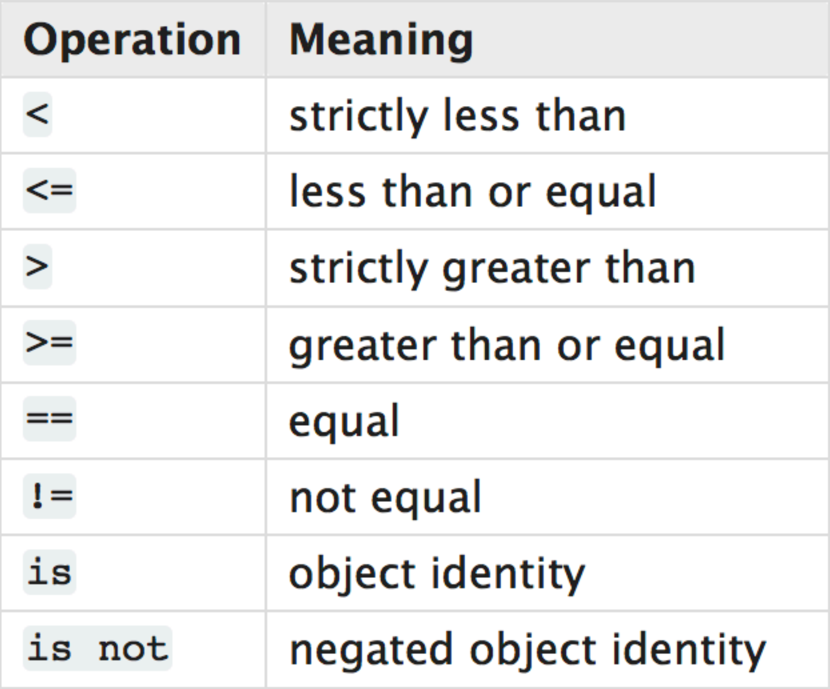
```
In [10]: who
np       x

In [11]: whos
Variable   Type      Data/Info
------------------------------
np         module    <module 'numpy' from '/us<...>ages/numpy/__init__.pyc'>
x          str       I am a string
```

Python Built-in Types

- The principal built-in types are numerics, sequences, mappings, classes, instances and exceptions
- Any object can be tested for truth value, for use in an **`if`** or **`while`** condition or as operand of the Boolean operations
- There are eight comparison operations in Python
- They all have the same priority (which is higher than that of the Boolean operations)
- Comparisons can be chained arbitrarily



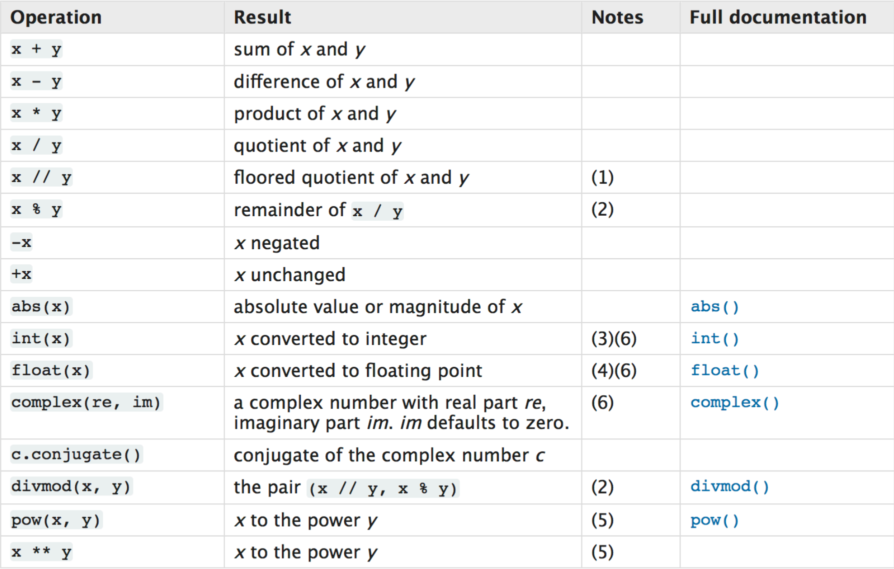
Operation	Result	Notes
`x or y`	if *x* is false, then *y*, else *x*	(1)
`x and y`	if *x* is false, then *x*, else *y*	(2)
`not x`	if *x* is false, then `True`, else `False`	(3)



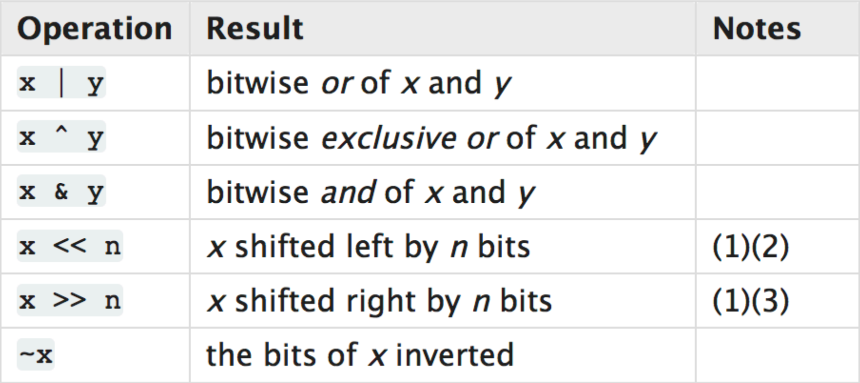
Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Python Built-in Types

- There are three distinct numeric types: integers, floating point numbers, and complex numbers
- All numeric types (except complex) support these operations, sorted by ascending priority
- Booleans are a subtype of integers
- Integers have unlimited precision
- Floating point numbers are usually implemented using **`double`** in C
- Complex numbers have a real and imaginary part, which are each a floating point number
- The priorities of the binary bitwise operations are all lower than the numeric operations and higher than the comparisons



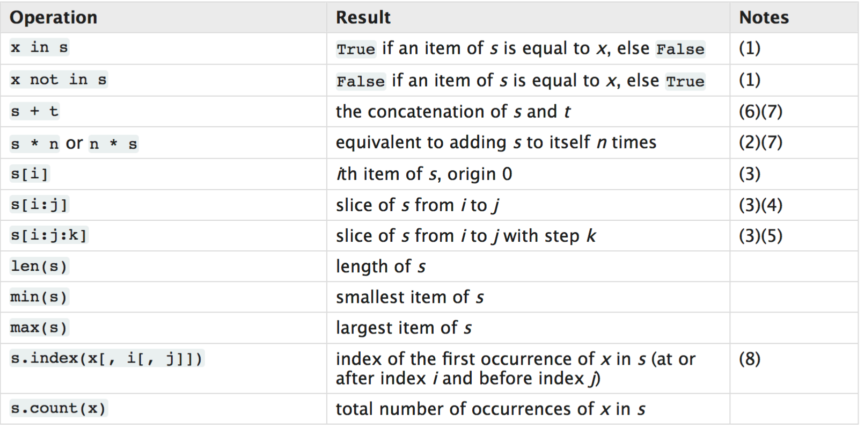
Operation	Result	Notes	Full documentation
`x + y`	sum of *x* and *y*		
`x - y`	difference of *x* and *y*		
`x * y`	product of *x* and *y*		
`x / y`	quotient of *x* and *y*		
`x // y`	floored quotient of *x* and *y*	(1)	
`x % y`	remainder of `x / y`	(2)	
`-x`	*x* negated		
`+x`	*x* unchanged		
`abs(x)`	absolute value or magnitude of *x*		`abs()`
`int(x)`	*x* converted to integer	(3)(6)	`int()`
`float(x)`	*x* converted to floating point	(4)(6)	`float()`
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	(6)	`complex()`
`c.conjugate()`	conjugate of the complex number *c*		
`divmod(x, y)`	the pair `(x // y, x % y)`	(2)	`divmod()`
`pow(x, y)`	*x* to the power *y*	(5)	`pow()`
`x ** y`	*x* to the power *y*	(5)	



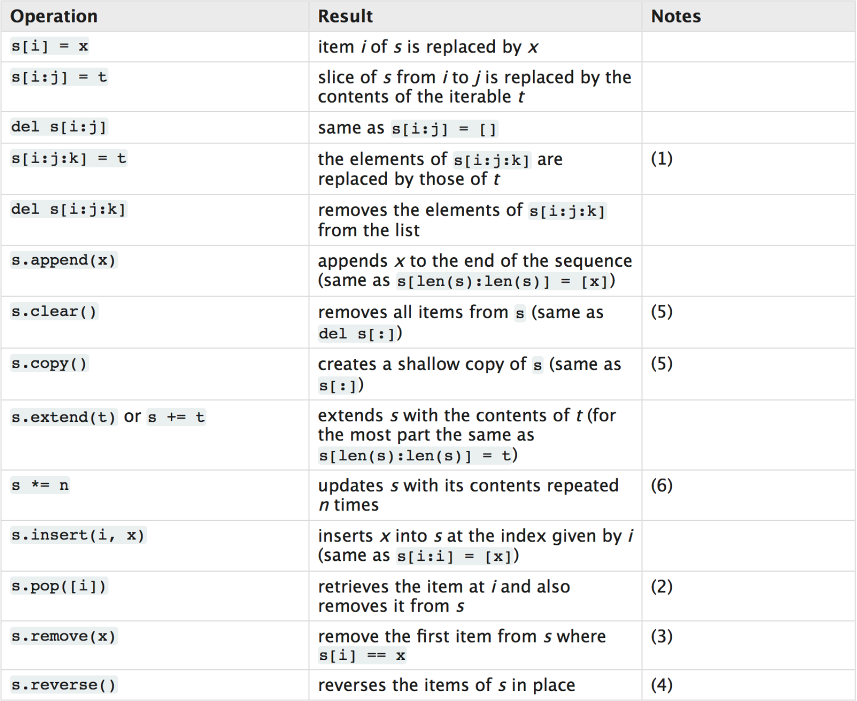
Operation	Result	Notes
`x | y`	bitwise *or* of *x* and *y*	
`x ^ y`	bitwise *exclusive or* of *x* and *y*	
`x & y`	bitwise *and* of *x* and *y*	
`x << n`	*x* shifted left by *n* bits	(1)(2)
`x >> n`	*x* shifted right by *n* bits	(1)(3)
`~x`	the bits of *x* inverted	

Python Built-in Types

- There are three basic sequence types: lists, tuples, and range objects
- Sequences of the same type also support comparisons
- Tuples and lists are compared lexicographically by comparing corresponding elements
- To compare equal, every element must compare equal and the two sequences must be of the same type and have the same length
- Concatenating immutable sequences always results in a new object



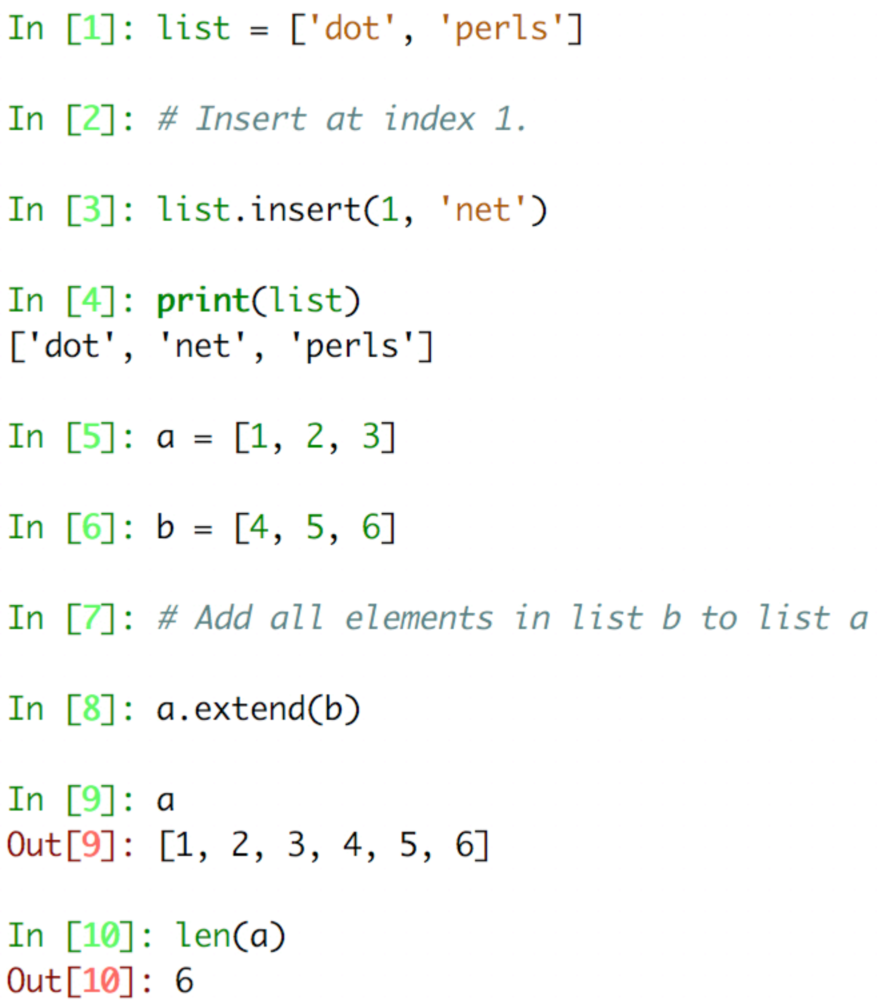
Operation	Result	Notes
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	(1)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	(1)
`s + t`	the concatenation of *s* and *t*	(6)(7)
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times	(2)(7)
`s[i]`	*i*th item of *s*, origin 0	(3)
`s[i:j]`	slice of *s* from *i* to *j*	(3)(4)
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*	(3)(5)
`len(s)`	length of *s*	
`min(s)`	smallest item of *s*	
`max(s)`	largest item of *s*	
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)	(8)
`s.count(x)`	total number of occurrences of *x* in *s*	



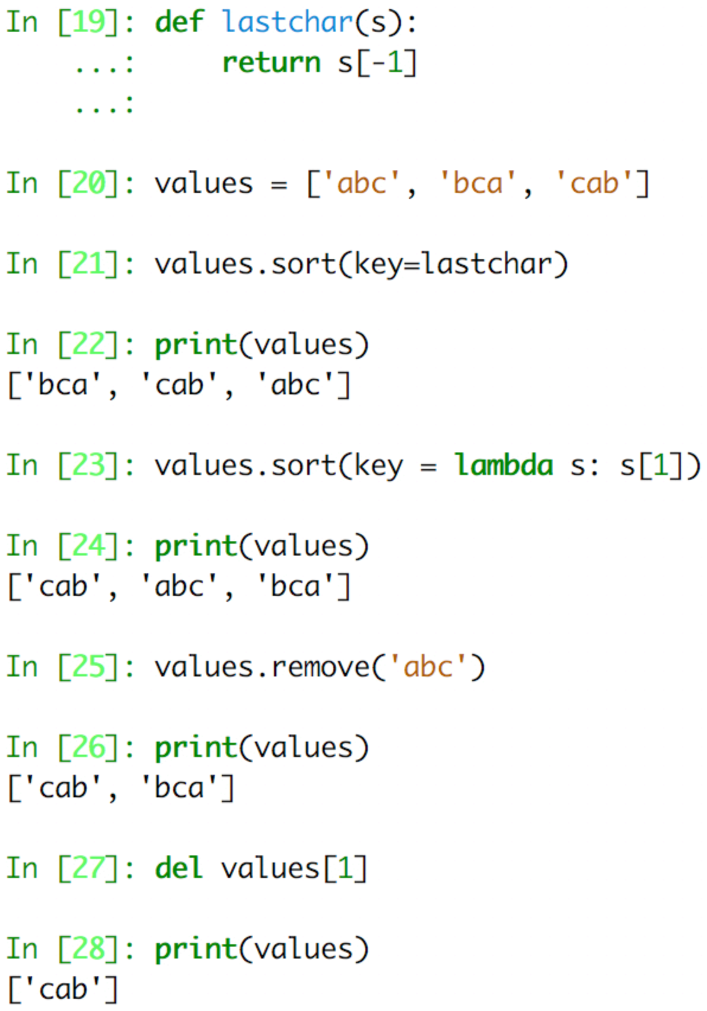
Operation	Result	Notes
`s[i] = x`	item *i* of *s* is replaced by *x*	
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*	
`del s[i:j]`	same as `s[i:j] = []`	
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*	(1)
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list	
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)	
`s.clear()`	removes all items from `s` (same as `del s[:]`)	(5)
`s.copy()`	creates a shallow copy of `s` (same as `s[:]`)	(5)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)	
`s *= n`	updates *s* with its contents repeated *n* times	(6)
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)	
`s.pop([i])`	retrieves the item at *i* and also removes it from *s*	(2)
`s.remove(x)`	remove the first item from *s* where `s[i] == x`	(3)
`s.reverse()`	reverses the items of *s* in place	(4)

Python Lists

- Lists are mutable sequences, typically used to store collections of homogeneous items



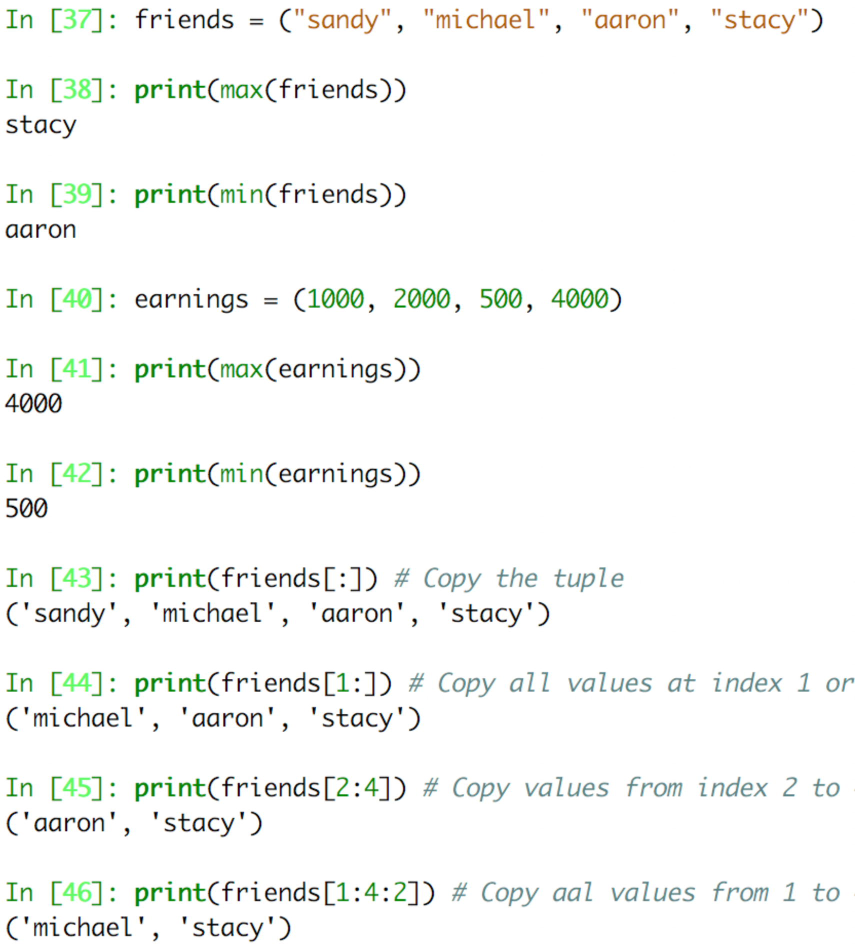
```
In [1]: list = ['dot', 'perls']

In [2]: # Insert at index 1.

In [3]: list.insert(1, 'net')

In [4]: print(list)
['dot', 'net', 'perls']

In [5]: a = [1, 2, 3]

In [6]: b = [4, 5, 6]

In [7]: # Add all elements in list b to list a

In [8]: a.extend(b)

In [9]: a
Out[9]: [1, 2, 3, 4, 5, 6]

In [10]: len(a)
Out[10]: 6
```



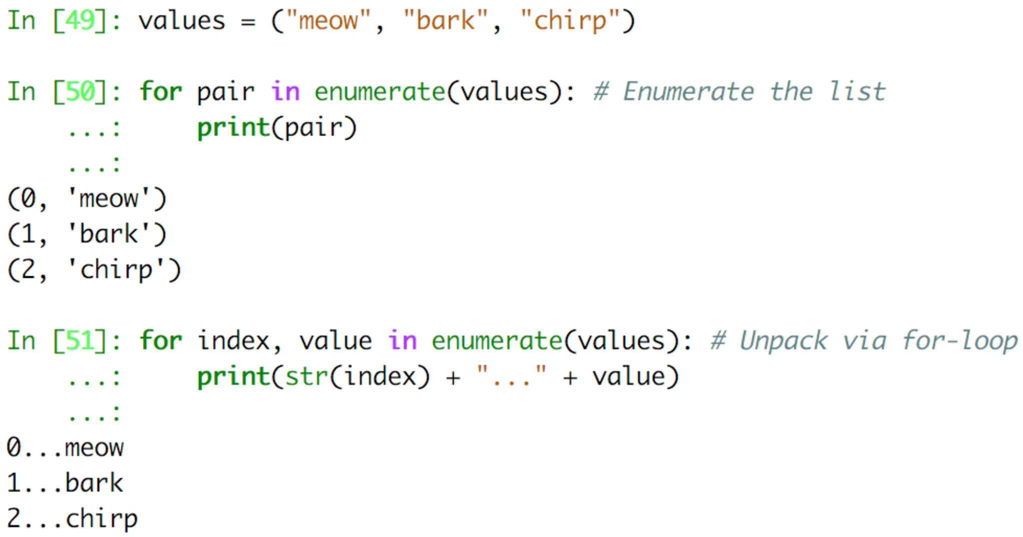
```
In [19]: def lastchar(s):
    ...:     return s[-1]
    ...:

In [20]: values = ['abc', 'bca', 'cab']

In [21]: values.sort(key=lastchar)

In [22]: print(values)
['bca', 'cab', 'abc']

In [23]: values.sort(key = lambda s: s[1])

In [24]: print(values)
['cab', 'abc', 'bca']

In [25]: values.remove('abc')

In [26]: print(values)
['cab', 'bca']

In [27]: del values[1]

In [28]: print(values)
['cab']
```

Python Tuples

- Tuples are immutable sequences, typically used to store collections of heterogeneous data



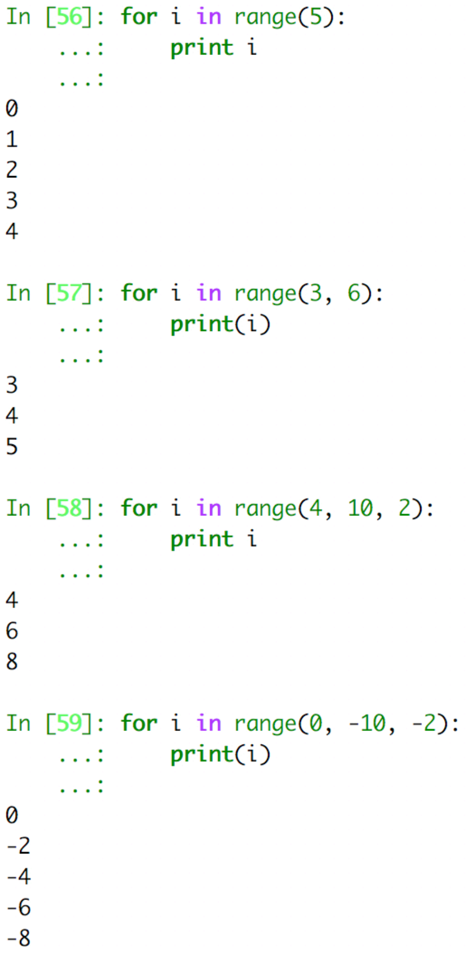
```
In [37]: friends = ("sandy", "michael", "aaron", "stacy")

In [38]: print(max(friends))
stacy

In [39]: print(min(friends))
aaron

In [40]: earnings = (1000, 2000, 500, 4000)

In [41]: print(max(earnings))
4000

In [42]: print(min(earnings))
500

In [43]: print(friends[:]) # Copy the tuple
('sandy', 'michael', 'aaron', 'stacy')

In [44]: print(friends[1:]) # Copy all values at index 1 or more
('michael', 'aaron', 'stacy')

In [45]: print(friends[2:4]) # Copy values from index 2 to 4
('aaron', 'stacy')

In [46]: print(friends[1:4:2]) # Copy aal values from 1 to 4 skipping elements
('michael', 'stacy')
```



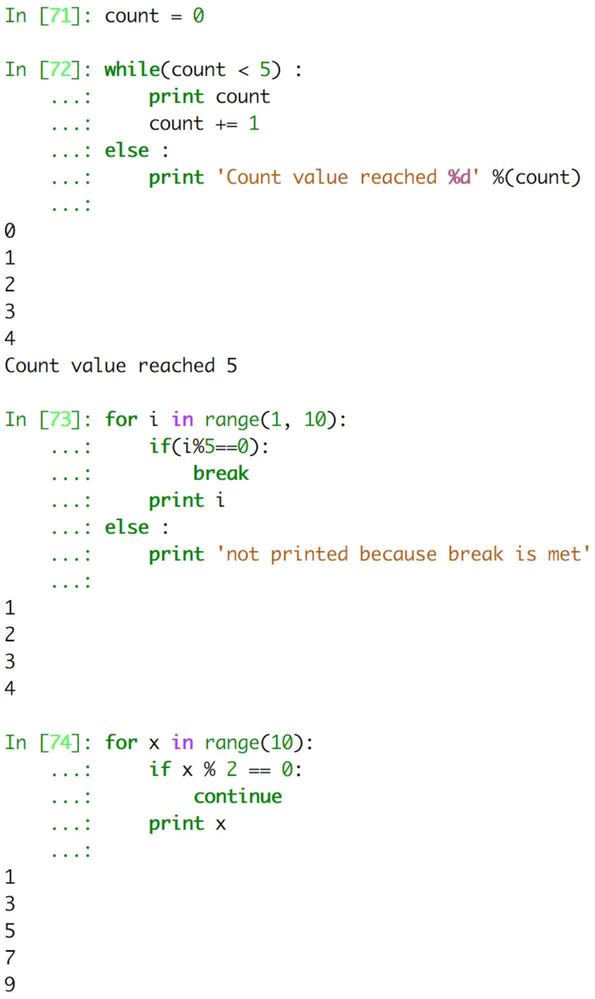
```
In [49]: values = ("meow", "bark", "chirp")

In [50]: for pair in enumerate(values): # Enumerate the list
    ...:     print(pair)
    ...:
(0, 'meow')
(1, 'bark')
(2, 'chirp')

In [51]: for index, value in enumerate(values): # Unpack via for-loop
    ...:     print(str(index) + "..." + value)
    ...:
0...meow
1...bark
2...chirp
```

Python Ranges

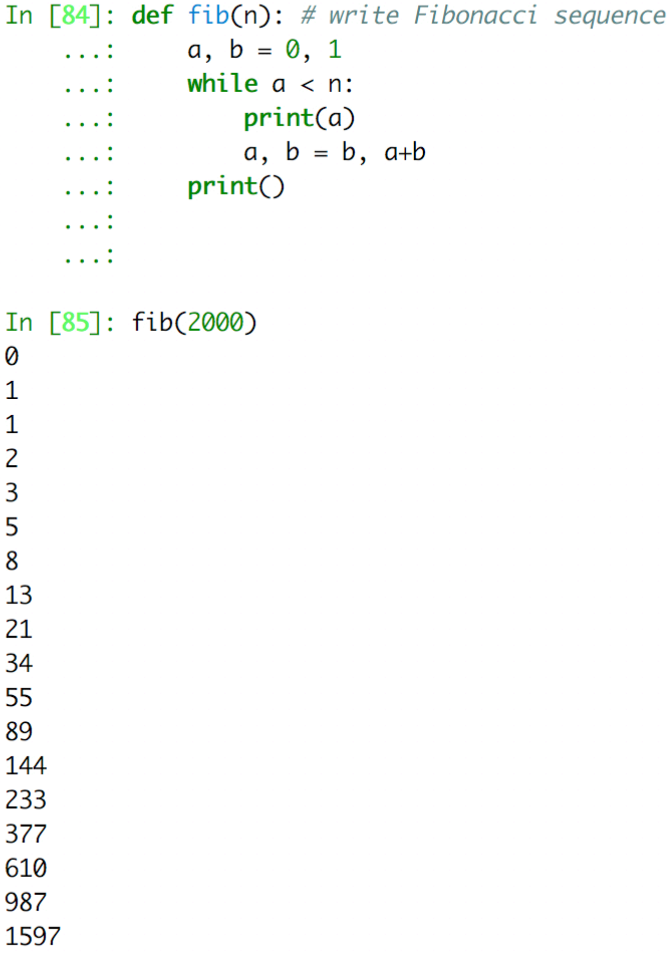
- The **`range`** type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in **`for`** loops



```
In [56]: for i in range(5):
    ...:     print i
    ...:
0
1
2
3
4

In [57]: for i in range(3, 6):
    ...:     print(i)
    ...:
3
4
5

In [58]: for i in range(4, 10, 2):
    ...:     print i
    ...:
4
6
8

In [59]: for i in range(0, -10, -2):
    ...:     print(i)
    ...:
0
-2
-4
-6
-8
```



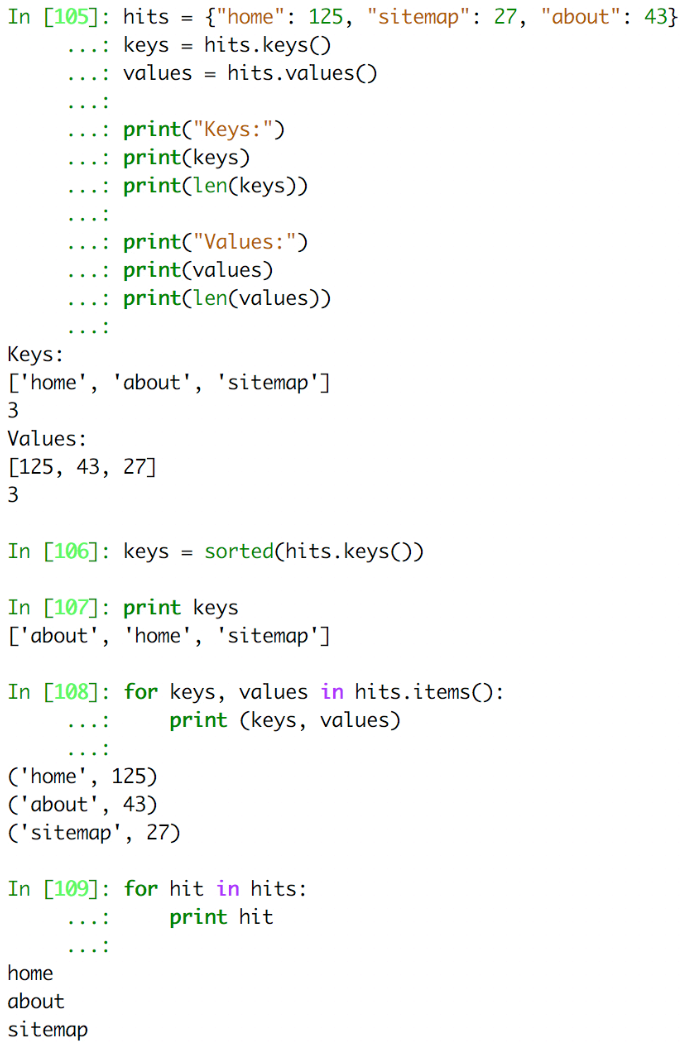
```
In [71]: count = 0

In [72]: while(count < 5) :
    ...:     print count
    ...:     count += 1
    ...: else :
    ...:     print 'Count value reached %d' %(count)
    ...:
0
1
2
3
4
Count value reached 5

In [73]: for i in range(1, 10):
    ...:     if(i%5==0):
    ...:         break
    ...:     print i
    ...: else :
    ...:     print 'not printed because break is met'
    ...:
1
2
3
4

In [74]: for x in range(10):
    ...:     if x % 2 == 0:
    ...:         continue
    ...:     print x
    ...:
1
3
5
7
9
```



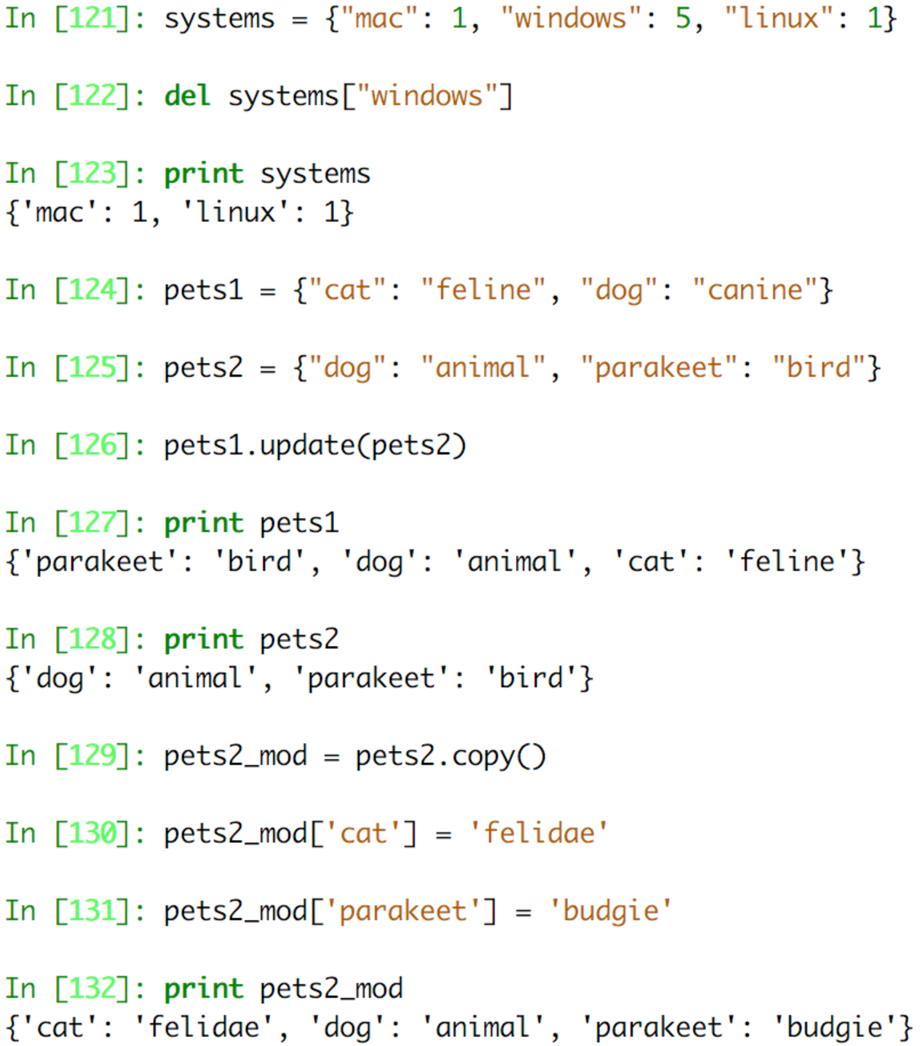
```
In [84]: def fib(n): # write Fibonacci sequence
    ...:     a, b = 0, 1
    ...:     while a < n:
    ...:         print(a)
    ...:         a, b = b, a+b
    ...:     print()
    ...:
    ...:

In [85]: fib(2000)
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
```

Python Dictionaries

- A Python dictionary is a mutable, unordered, mapping of unique keys to values of any value

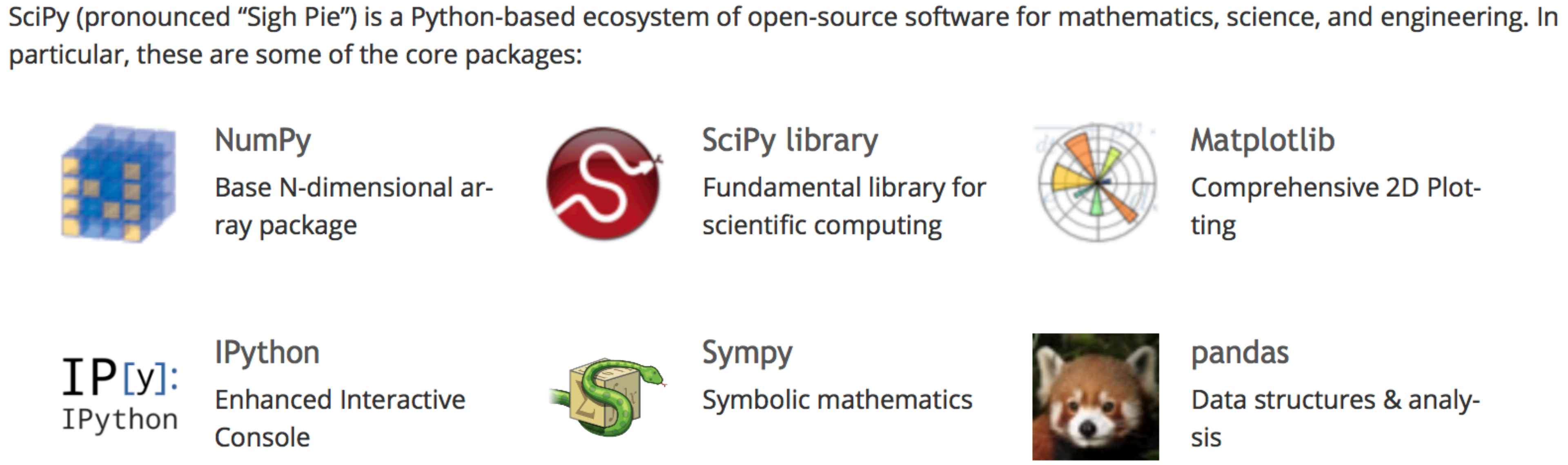


```
In [105]: hits = {"home": 125, "sitemap": 27, "about": 43}
     ...: keys = hits.keys()
     ...: values = hits.values()
     ...:
     ...: print("Keys:")
     ...: print(keys)
     ...: print(len(keys))
     ...:
     ...: print("Values:")
     ...: print(values)
     ...: print(len(values))
     ...:
Keys:
['home', 'about', 'sitemap']
3
Values:
[125, 43, 27]
3

In [106]: keys = sorted(hits.keys())

In [107]: print keys
['about', 'home', 'sitemap']

In [108]: for keys, values in hits.items():
     ...:     print (keys, values)
     ...:
('home', 125)
('about', 43)
('sitemap', 27)

In [109]: for hit in hits:
     ...:     print hit
     ...:
home
about
sitemap
```



```
In [121]: systems = {"mac": 1, "windows": 5, "linux": 1}

In [122]: del systems["windows"]

In [123]: print systems
{'mac': 1, 'linux': 1}

In [124]: pets1 = {"cat": "feline", "dog": "canine"}

In [125]: pets2 = {"dog": "animal", "parakeet": "bird"}

In [126]: pets1.update(pets2)

In [127]: print pets1
{'parakeet': 'bird', 'dog': 'animal', 'cat': 'feline'}

In [128]: print pets2
{'dog': 'animal', 'parakeet': 'bird'}

In [129]: pets2_mod = pets2.copy()

In [130]: pets2_mod['cat'] = 'felidae'

In [131]: pets2_mod['parakeet'] = 'budgie'

In [132]: print pets2_mod
{'cat': 'felidae', 'dog': 'animal', 'parakeet': 'budgie'}
```

Python Third Party Utilities



SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IPython
Enhanced Interactive Console

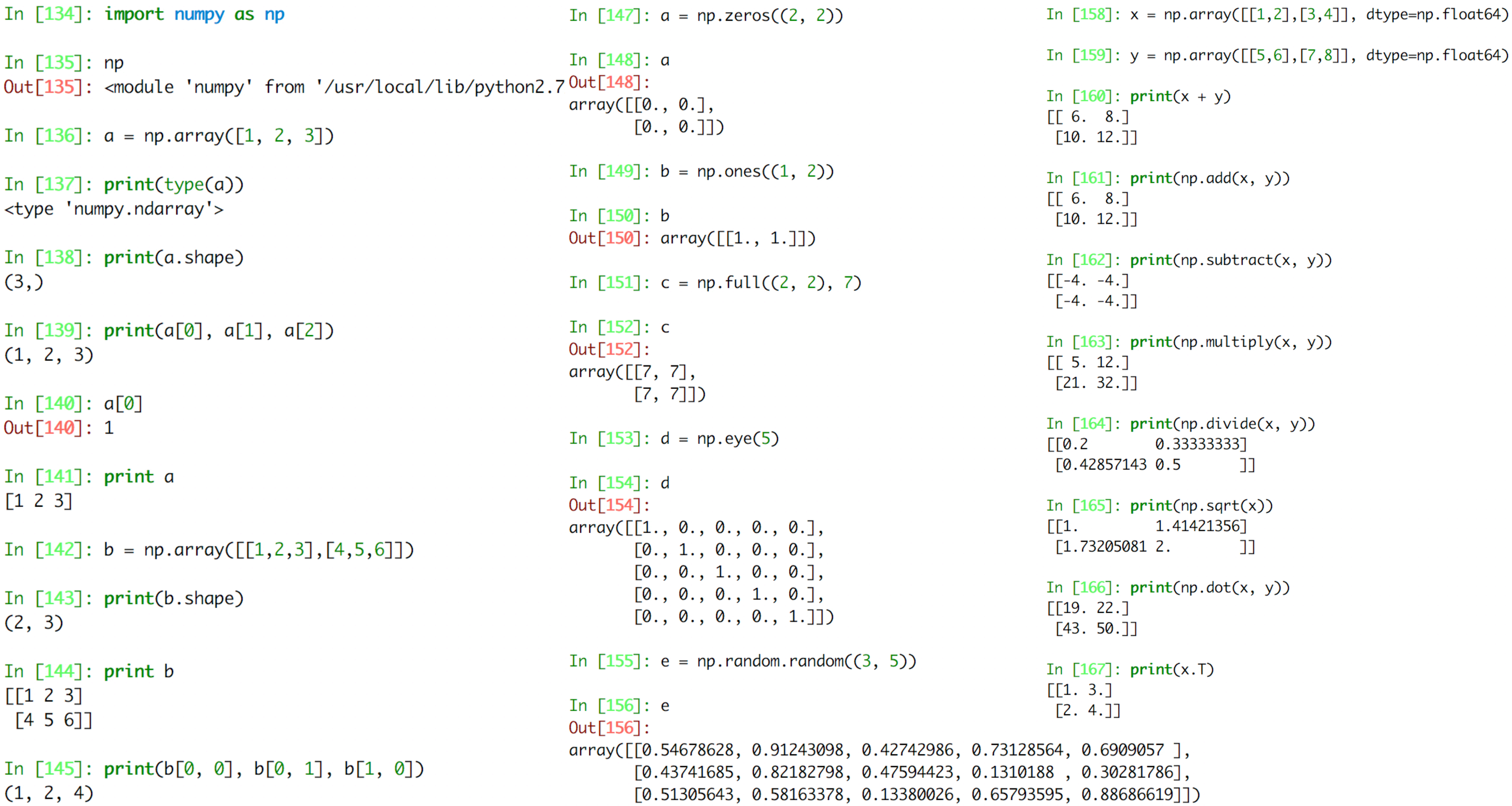
Sympy
Symbolic mathematics

pandas
Data structures & analysis

- Numpy: is the foundational library for scientific computing in Python. NumPy introduces objects for multidimensional arrays and matrices, as well as routines that allow developers to perform advanced mathematical and statistical functions on those arrays
- SciPy: builds on NumPy by adding a collection of algorithms and high-level commands for manipulating and visualizing data. Includes functions for computing integrals numerically, solving differential equations, optimization, and much more…
- Matplotlib: is the standard Python library for creating plots and graphs
- Pandas: adds data structures and tools that are designed for practical data analysis in finance, statistics, social sciences, and engineering

Numpy

- Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays



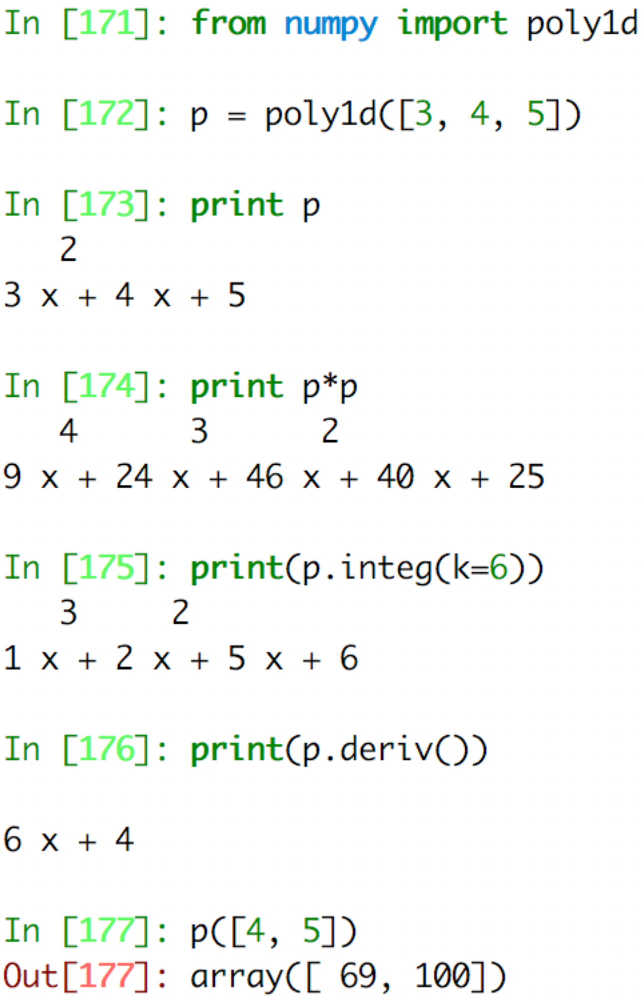
```
In [134]: import numpy as np

In [135]: np
Out[135]: <module 'numpy' from '/usr/local/lib/python2.7

In [136]: a = np.array([1, 2, 3])

In [137]: print(type(a))
<type 'numpy.ndarray'>

In [138]: print(a.shape)
(3,)

In [139]: print(a[0], a[1], a[2])
(1, 2, 3)

In [140]: a[0]
Out[140]: 1

In [141]: print a
[1 2 3]

In [142]: b = np.array([[1,2,3],[4,5,6]])

In [143]: print(b.shape)
(2, 3)

In [144]: print b
[[1 2 3]
 [4 5 6]]

In [145]: print(b[0, 0], b[0, 1], b[1, 0])
(1, 2, 4)
```

```
In [147]: a = np.zeros((2, 2))

In [148]: a
Out[148]:
array([[0., 0.],
       [0., 0.]])

In [149]: b = np.ones((1, 2))

In [150]: b
Out[150]: array([[1., 1.]])

In [151]: c = np.full((2, 2), 7)

In [152]: c
Out[152]:
array([[7, 7],
       [7, 7]])

In [153]: d = np.eye(5)

In [154]: d
Out[154]:
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])

In [155]: e = np.random.random((3, 5))

In [156]: e
Out[156]:
array([[0.54678628, 0.91243098, 0.42742986, 0.73128564, 0.6909057 ],
       [0.43741685, 0.82182798, 0.47594423, 0.1310188 , 0.30281786],
       [0.51305643, 0.58163378, 0.13380026, 0.65793595, 0.88686619]])
```

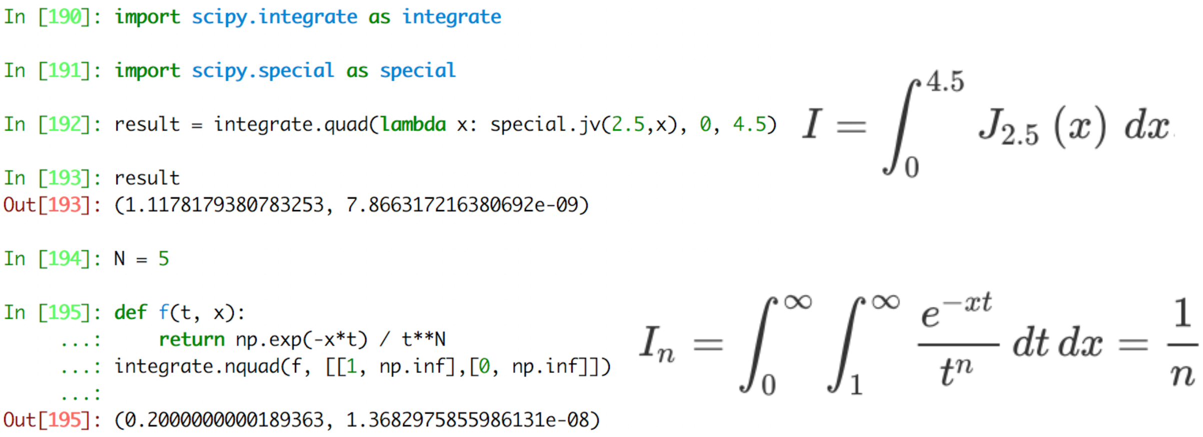
```
In [158]: x = np.array([[1,2],[3,4]], dtype=np.float64)

In [159]: y = np.array([[5,6],[7,8]], dtype=np.float64)

In [160]: print(x + y)
[[ 6.  8.]
 [10. 12.]]

In [161]: print(np.add(x, y))
[[ 6.  8.]
 [10. 12.]]

In [162]: print(np.subtract(x, y))
[[-4. -4.]
 [-4. -4.]]

In [163]: print(np.multiply(x, y))
[[ 5. 12.]
 [21. 32.]]

In [164]: print(np.divide(x, y))
[[0.2        0.33333333]
 [0.42857143 0.5       ]]

In [165]: print(np.sqrt(x))
[[1.         1.41421356]
 [1.73205081 2.        ]]

In [166]: print(np.dot(x, y))
[[19. 22.]
 [43. 50.]]

In [167]: print(x.T)
[[1. 3.]
 [2. 4.]]
```

SciPy

- SciPy builds onNumPy, and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications



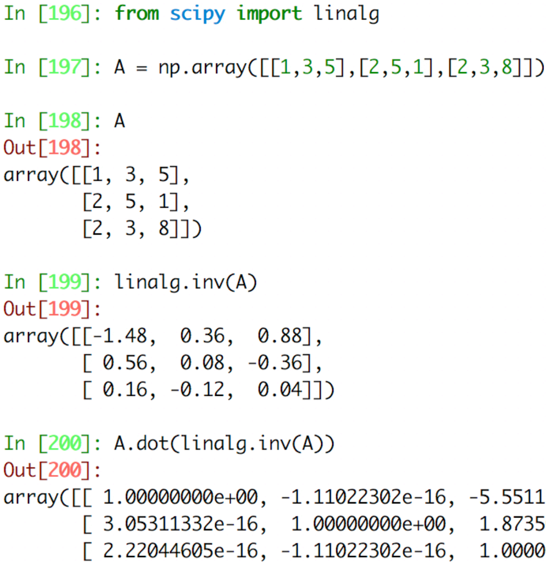
```
In [171]: from numpy import poly1d

In [172]: p = poly1d([3, 4, 5])

In [173]: print p
   2
3 x + 4 x + 5

In [174]: print p*p
   4      3      2
9 x + 24 x + 46 x + 40 x + 25

In [175]: print(p.integ(k=6))
   3     2
1 x + 2 x + 5 x + 6

In [176]: print(p.deriv())

6 x + 4

In [177]: p([4, 5])
Out[177]: array([ 69, 100])
```



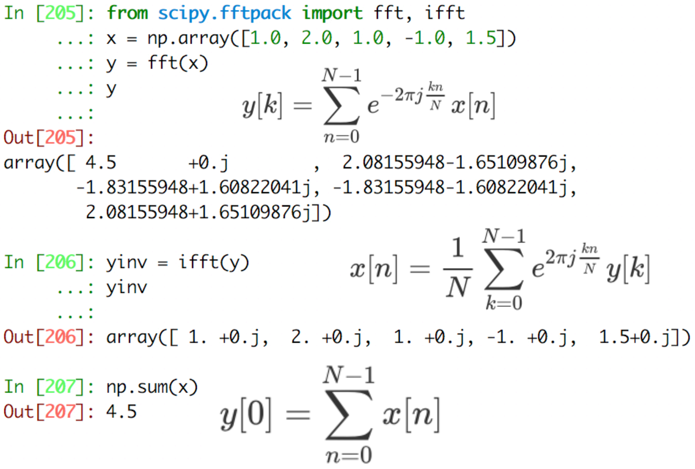
```
In [190]: import scipy.integrate as integrate

In [191]: import scipy.special as special

In [192]: result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)

In [193]: result
Out[193]: (1.1178179380783253, 7.866317216380692e-09)

In [194]: N = 5

In [195]: def f(t, x):
     ...:     return np.exp(-x*t) / t**N
     ...: integrate.nquad(f, [[1, np.inf],[0, np.inf]])
     ...:
Out[195]: (0.2000000000189363, 1.3682975855986131e-08)
```

$$I = \int_0^{4.5} J_{2.5}\left(x\right)\,dx$$

$$I_n = \int_0^{\infty}\int_1^{\infty}\frac{e^{-xt}}{t^n}\,dt\,dx = \frac{1}{n}$$



```
In [196]: from scipy import linalg

In [197]: A = np.array([[1,3,5],[2,5,1],[2,3,8]])

In [198]: A
Out[198]:
array([[1, 3, 5],
       [2, 5, 1],
       [2, 3, 8]])

In [199]: linalg.inv(A)
Out[199]:
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])

In [200]: A.dot(linalg.inv(A))
Out[200]:
array([[ 1.00000000e+00, -1.11022302e-16, -5.55111512e-17],
       [ 3.05311332e-16,  1.00000000e+00,  1.87350135e-16],
       [ 2.22044605e-16, -1.11022302e-16,  1.00000000e+00]])
```



```
In [205]: from scipy.fftpack import fft, ifft
     ...: x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
     ...: y = fft(x)
     ...: y
     ...:
Out[205]:
array([ 4.5       +0.j        ,  2.08155948-1.65109876j,
       -1.83155948+1.60822041j, -1.83155948-1.60822041j,
        2.08155948+1.65109876j])

In [206]: yinv = ifft(y)
     ...: yinv
     ...:
Out[206]: array([ 1. +0.j,  2. +0.j,  1. +0.j, -1. +0.j,  1.5+0.j])

In [207]: np.sum(x)
Out[207]: 4.5
```

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j\frac{kn}{N}}\,x[n]$$

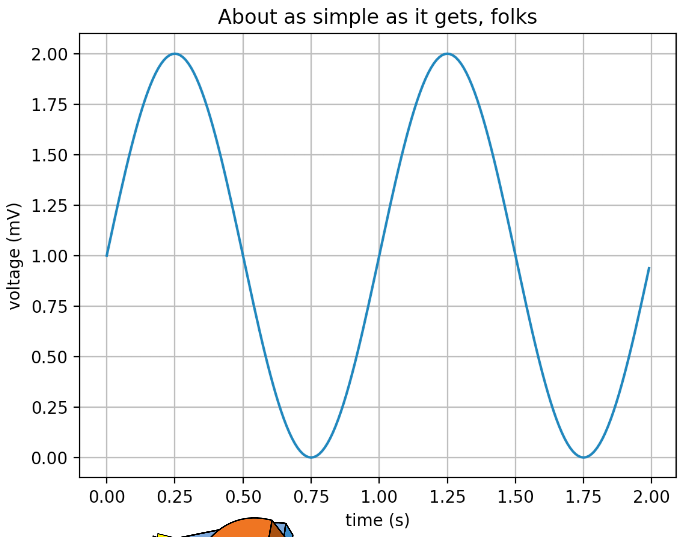
$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} e^{2\pi j\frac{kn}{N}}\,y[k]$$

$$y[0] = \sum_{n=0}^{N-1} x[n]$$

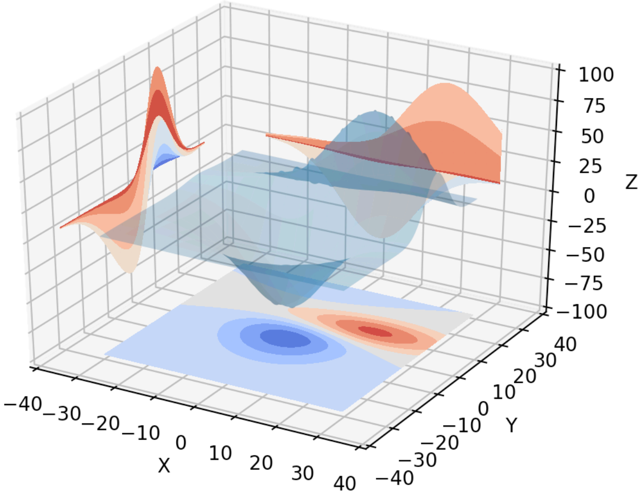
Matplotlib

- Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

```
In [209]: import matplotlib.pyplot as plt
     ...: import numpy as np
     ...:
     ...: t = np.arange(0.0, 2.0, 0.01)
     ...: s = 1 + np.sin(2*np.pi*t)
     ...: plt.plot(t, s)
     ...:
     ...: plt.xlabel('time (s)')
     ...: plt.ylabel('voltage (mV)')
     ...: plt.title('About as simple as it gets, folks')
     ...: plt.grid(True)
     ...: plt.savefig("test.png")
     ...: plt.show()
```



```
In [212]: from mpl_toolkits.mplot3d import axes3d
     ...: import matplotlib.pyplot as plt
     ...: from matplotlib import cm
     ...:
     ...: fig = plt.figure()
     ...: ax = fig.gca(projection='3d')
     ...: X, Y, Z = axes3d.get_test_data(0.05)
     ...: ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
     ...: cset = ax.contourf(X, Y, Z, zdir='z', offset=-100, cmap=cm.coolwarm)
     ...: cset = ax.contourf(X, Y, Z, zdir='x', offset=-40, cmap=cm.coolwarm)
     ...: cset = ax.contourf(X, Y, Z, zdir='y', offset=40, cmap=cm.coolwarm)
     ...:
     ...: ax.set_xlabel('X')
     ...: ax.set_xlim(-40, 40)
     ...: ax.set_ylabel('Y')
     ...: ax.set_ylim(-40, 40)
     ...: ax.set_zlabel('Z')
     ...: ax.set_zlim(-100, 100
     ...:
     ...: plt.show()
```

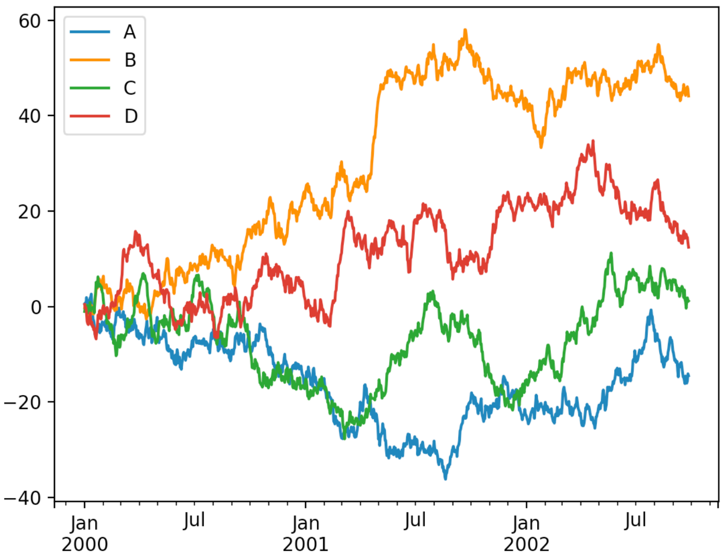


Pandas

- Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python

```
In [230]: import pandas as pd

In [231]: s = pd.Series([1,3,5,np.nan,6,8])

In [232]: s
Out[232]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64

In [233]: dates = pd.date_range('20130101', periods=6)

In [234]: dates
Out[234]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [235]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=['A', 'B', 'C', 'D'])

In [236]: df
Out[236]:
                   A         B         C         D
2013-01-01 -0.225703  0.613700  0.915022 -0.051510
2013-01-02 -0.225572  0.587910  2.566853  0.330648
2013-01-03  0.989051 -1.358961 -0.440455  0.264423
2013-01-04 -1.377799 -0.870809  2.401718  0.006320
2013-01-05 -0.012309 -1.007861 -0.270010 -0.631313
2013-01-06  0.112272 -1.559541 -0.336788  1.161615
```

```
In [264]: import pandas as pd

In [265]: import matplotlib.pyplot as plt

In [266]: ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1

In [267]: ts = ts.cumsum()

In [268]: df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
     ...:                   columns=['A', 'B', 'C', 'D'])

In [269]: df = df.cumsum()

In [270]:  plt.figure(); df.plot(); plt.legend(loc='best')
Out[270]: <matplotlib.legend.Legend at 0x11b015a50>

In [271]: plt.show()
```

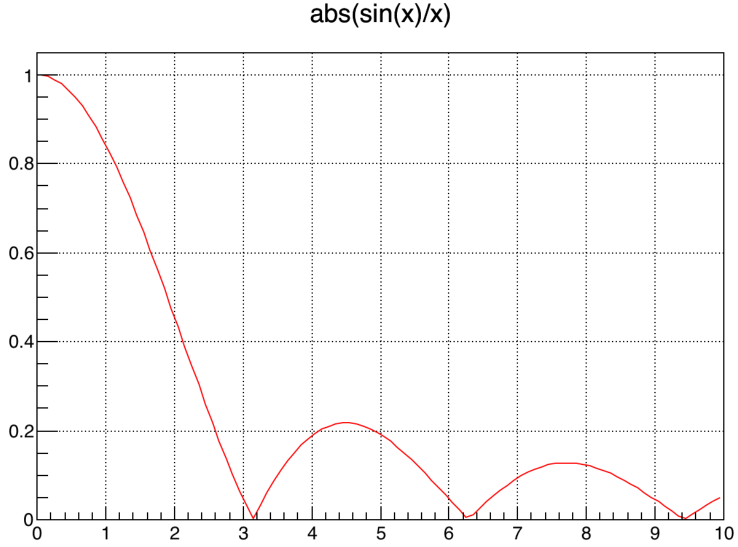


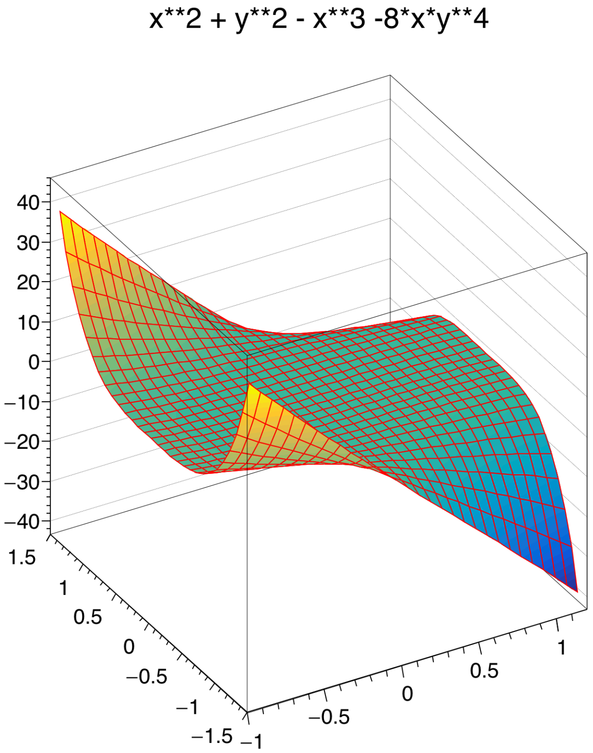
PyROOT

- PyROOT is a Python extension module that allows the user to interact with any ROOT class from the Python interpreter. PyROOT offers the possibility to execute and evaluate any Python command or start a Python shell from the ROOT/CLING prompt

```
In [29]: import ROOT

In [30]: c1 = ROOT.TCanvas( 'c1', 'Example with Formula', 200, 10, 700, 500 )

In [31]: fun1 = ROOT.TF1( 'fun1', 'abs(sin(x)/x)', 0, 10 )

In [32]: c1.SetGridx()

In [33]: c1.SetGridy()

In [34]: fun1.Draw()

In [35]: c1.Update()
```

```
In [2]: import ROOT as R

In [3]: c1 = R.TCanvas( 'c1', 'Surfaces Drawing Options', 200, 10, 700, 900 )

In [4]: f2 = R.TF2( 'f2', 'x**2 + y**2 - x**3 -8*x*y**4', -1, 1.2, -1.5, 1.5 )

In [5]: R.f2.SetContour(48)

In [6]: R.f2.SetFillColor(45)

In [7]: f2.Draw('surf1')

In [8]: c1.Update()
```





Joint Hall A&C Data Analysis Workshop

Questions?

Backup Slides

Slide Title