STA141C: Big Data & High Performance Statistical Computing

Lecture 1: Python programming (1)

Cho-Jui Hsieh
UC Davis

April 4, 2017

Python

- Python is a **scripting language**:
- Non-scripting language (C++. java): Need to compile the code before running it:
  - Very fast
  - Can easily control memory allocation
  - No interactive mode for development
  - Difficult for non-programmers
- Scripting language (python, matlab, R, julia): No need for compilation
  - Convenient for developers
  - Usually slower than C/C++
- Python:
  - Very rich computing libraries
  - Many powerful environments to work in

Tools and environments

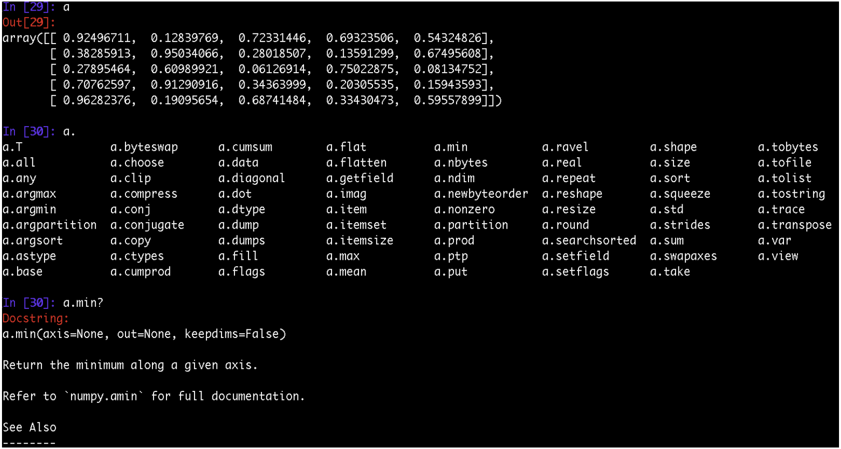
ipython

- An interactive environments to run python
- Can type any python command and check the result immedinately
- Also support
    - Tab completion
    - magic commands
    - System shell commands
- Jupyter notebook: can browse and run the code on your web browser.

    very useful for visualization
- We recommend to install python with anaconda (it includes all the modules we need)

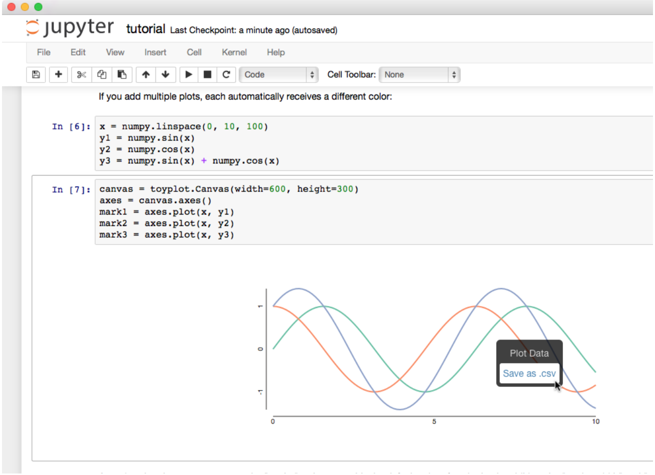
    `https://www.continuum.io/downloads`

ipython



```
In [29]: a
Out[29]:
array([[ 0.92496711,  0.12839769,  0.72331446,  0.69323506,  0.54324826],
       [ 0.38285913,  0.95034066,  0.28018507,  0.13591299,  0.67495608],
       [ 0.27895464,  0.60989921,  0.06126914,  0.75022875,  0.08134752],
       [ 0.70762597,  0.91290916,  0.34363999,  0.20305535,  0.15943593],
       [ 0.96282376,  0.19095654,  0.68741484,  0.33430473,  0.59557899]])

In [30]: a.
a.T             a.byteswap      a.cumsum        a.flat          a.min           a.ravel         a.shape         a.tobytes
a.all           a.choose        a.data          a.flatten       a.nbytes        a.real          a.size          a.tofile
a.any           a.clip          a.diagonal      a.getfield      a.ndim          a.repeat        a.sort          a.tolist
a.argmax        a.compress      a.dot           a.imag          a.newbyteorder  a.reshape       a.squeeze       a.tostring
a.argmin        a.conj          a.dtype         a.item          a.nonzero       a.resize        a.std           a.trace
a.argpartition  a.conjugate     a.dump          a.itemset       a.partition     a.round         a.strides       a.transpose
a.argsort       a.copy          a.dumps         a.itemsize      a.prod          a.searchsorted  a.sum           a.var
a.astype        a.ctypes        a.fill          a.max           a.ptp           a.setfield      a.swapaxes      a.view
a.base          a.cumprod       a.flags         a.mean          a.put           a.setflags      a.take

In [30]: a.min?
Docstring:
a.min(axis=None, out=None, keepdims=False)

Return the minimum along a given axis.

Refer to `numpy.amin` for full documentation.

See Also
--------
```

Jupyter Notebook



tutorial Last Checkpoint: a minute ago (autosaved)

Cell Toolbar: None

If you add multiple plots, each automatically receives a different color:

In [6]:
```
x = numpy.linspace(0, 10, 100)
y1 = numpy.sin(x)
y2 = numpy.cos(x)
y3 = numpy.sin(x) + numpy.cos(x)
```

In [7]:
```
canvas = toyplot.Canvas(width=600, height=300)
axes = canvas.axes()
mark1 = axes.plot(x, y1)
mark2 = axes.plot(x, y2)
mark3 = axes.plot(x, y3)
```

Write and run your code on servers (recommended, but not required)

For example:

- Connect to the server:

```
ssh chohsieh@hilbert.ucdavis.edu
```

- Run “screen”, which allows you to
  - Use multiple shell windows from a single ssh session
  - Keep a shell active, and able to disconnect/reconnect from multiple locations
  - Run a long running process without keep an active ssh session

```
screen (open a new screen)
screen -r (reopen the previous session)
screen -r 1111 (reopen the session number 1111)
```

- Check `http://www.tecmint.com/screen-command-examples-to-manage-linux-terminals/`

Useful shell commands

- Check this page (very useful!): http://practical-data-science.blogspot.com/2012/09/basic-unix-shell-commands-for-data.html
- Useful ones:
  - grep: return all the input lines that match the specified pattern

    ```
    grep 'tutorial' filename
    ```
  - Redirect the results to a file or another program

    ```
    grep 'tutorial' filename >> output_file
    ```
  - cut: select certain fields of the input file

    ```
    cut -d'' '' -f2,4 sample.txt
    (cut the 2nd and 4th column of the file)
    ```
  - cat: concatenate two files into a single file
  - ls: list the files and directories
  - cd: change the directory
  - head/tail: output the first (or last) x lines of the file:

    ```
    head -n 5 sample.txt
    ```
  - ...

Python programming (basic grammar)

Variable types in python

- No need to declare the type
- Type of a variable can change
- The following tutorial is based on

  http://www.scipy-lectures.org/intro/language/python_language.html

Numerical Types

- Integer

```
>>> a = 4
>>> type(a)
<type ’int’>
>>> a/3
1
```

- Float

```
>>> b = 4.5
>>> type(b)
<type ’float’>
>>> a/3
1.125
```

Numerical Types

- Complex

```
>>> a = 1+2j
```

- Booleans

```
>>> a = 3>4
>>> type(a)
<type 'bool'>
```

- Check memory usage (in bytes)

```
>>> import sys
>>> a = 3
>>> sys.getsizeof(a)
24
```

Strings

```
>>> s = 'Hello,'
>>> s = s + ' World'
>>> s
'Hello World'
>>> s[0]
'H'
>>> s[-1]
'd'
>>> s[1:-4]
'ello, W'
>>> len(s)
12
```

- String is **immutable**: cannot do

  `s[1]='a'`
- Use string.* to modify/update the string

More on Strings

- str.replace: return a copy of a string str with all occurrences of substring old replaced by new.

```
>>> s
'Hello World'
>>> s = s.replace(' ','')
>>> s
'HelloWorld'
```

- str.split: return a list of the words in the string S, using sep as the delimiter string.

```
>>> s
'Hello World !!'
>>> s.split(' ')
['Hello', 'World', '!!']
```

String formatting operations

- Use the formatting operator “%”

```
>>> 'aaa %d bbb'%1234
aaa 1234 bbb
>>> 'aaa %s %d bbb'%('test', 1234)
aaa test 1234 bbb
```

- Common types:
  - d: integer
  - f: floating point
  - s: string

Lists

- A list is an ordered collection of objects
- Objects in a list can have different types

```
>>> colors = [’red’, ’blue’, ’green’, ’black’, ’white’]
>>> type(colors)
<type ’list’>
>>> colors[2]
’green’
>>> colors.append(’pink’)
>>> colors
[’red’, ’blue’, ’green’, ’black’, ’white’, ’pink’]
>>> colors[0] = ’yellow’
>>> colors
[’yellow’, ’blue’, ’green’, ’black’, ’white’, ’pink’]
```

Tuple

- A tuple is a sequence of (immutable) objects. Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists.

```
>>> colors = (’red’, ’blue’, ’green’, ’black’, ’white’)
>>> type(colors)
<type ’tuple’>
>>> colors[0]
’red’
>>> colors[0] = yellow
Traceback (most recent call last):
  File ‘‘<stdin>’’, line 1, in <module>
TypeError: ’tuple’ object does not support item assignme
```

Tuple

- A tuple is a sequence of (immutable) objects. Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists.

```
>>> colors = (’red’, ’blue’, ’green’, ’black’, ’white’)
>>> type(colors)
<type ’tuple’>
>>> colors[0]
’red’
>>> colors[0] = yellow
Traceback (most recent call last):
  File ‘‘<stdin>’’, line 1, in <module>
TypeError: ’tuple’ object does not support item assignme
```

```
>>> colors = (’yellow’) + colors[1:]
>>> colors
(’yellow’, ’blue’, ’green’, ’black’, ’white’)
```

Set

- Set is a collection of unordered, unique items.

```
>>> s = {1, 5, 10, 5}
>>> s
{1, 5, 10}
>>> s.difference({1, 5})
{10}
>>> s.union({1, 15})
{1,5,10,15}
```

Dictionaries

- Dictionary is an efficient table that maps keys to values (implemented by hash table, will introduce later)

  ```
  dict = { key_1:value_1, key_2:value_2, ..., key_n:value_n}
  ```

- Very useful!

  ```
  >>> wordcount = {’is’:10, ’test’:2, ’statistics’:3}
  >>> wordcount[’is’]
  10
  >>> wordcount[’statistics’] = 5
  >>> wordcount
  {’is’: 10, ’statistics’: 5, ’test’: 2}
  >>> ’computer’ in wordcount
  False
  >>> wordcount = {} # empty dictionary
  ```

Dictionaries

- Loop over all the keys in a dictionary *d*

```
>>> for key in d:
>>>     print key
test
is
statistics
```

- Loop over all the key:value pairs in dictionary:

```
>>> for key,val in d:
>>>     print key, val
test 2
is 10
statistics 3
```

Other build-in functions

- sorted: return a sorted list:

```
>>> mylist = [3, 1, 4]
>>> sorted(mylist)
[1, 3, 4]
>>> sorted(mylist, reverse=True)
[4, 3, 1]
```

- Check `http://pythoncentral.io/how-to-sort-python-dictionaries-by-key-or-value/` for sorting a dictionary.
- Print: output to the screen

```
>>> print ’test print’
test print
>>> print ’test %d print’%10
test 10 print
```

Mutable vs immutable

Python objects can be either mutable or immutable:

- Mutable objects can be altered (in place):

  list, dict, set, user-defined classes

- Immutable objects cannot be changed but rather return new objects when attempting to update.

  int, float, complex, bool, string, tuple

```
In [22]: a = 10
In [23]: id(a)
Out[23]: 14807168
In [24]: a+=10
In [25]: id(a)
Out[25]: 14806928
```

Control Flow: if/else

- Blocks are delimited by indentation.
- When will “if (expression):” be False?
  - (expression) is an object: it is False if the object is 0, empty container, False, or None (NoneType). Otherwise it is True.
  - (expression) is “a == b” (or >=, <=, etc)
  - (expression) is “a in b”: if a is in the container b, then True.

```
>>> a = 10
>>> if a == 1:
...     print(1)
... elif a == 2:
...     print(2)
... else:
...     print(’other’)
other
```

Control Flow: for loop

- Range: can generate a list containing arithmetic progressions.

```
>>> range(4)
[0, 1, 2, 3]
>>> range(0,5,2)
[0, 2, 4]
```

- For loop: iterate over all the objects in a container

```
>>> for i in range(3):
...     print(i)
0
1
2
>>> for word in [’aa’, ’bb’]
...     print(’Hi %s’%word)
Hi aa
Hi bb
```

Control Flow: while loop

- Similar to other languages (C, java)

```
>>> z = 1
>>> while z < 100:
        z = z**2 + 1
>>> z
677
```

- We can use break and continue in both for loop and while loop.

Looping over a dictionary

```
>>> d = {’a’: 1, ’b’:1.2, ’c’:5}
>>> for key in d:
...     print(’Key %s has value %s’%(key, d[key]))
Key a has value 10
Key c has value 5
Key b has value 1.2
>>> for key, val in d.items():
Key a has value 10
Key c has value 5
Key b has value 1.2
```

Defining Functions

- Defining a function:

```
>>> def double_it(x):
...     return x*2
...
>>> double_it(3)
... 6
```

- Passing by value or reference?
  - Immutable objects:
    (anything done in the function will not change the original copy)
  - Mutable objects:
    (anything done in the function will also change the original copy)

Examples

```
>>> def try_to_modify(x, y):
...     x = 23
...     y.append(42)
>>> a = 77
>>> b = [99]
>>> try_to_modify(a,b)
23
[99,42]
>>> print(a)
77
>>> print(b)
[99,42]
```

Global variables

- Local variables: variables defined in the function, cannot be accessed outside the function
- Global variables: variables declared outside the function, can be referenced in the function
- In python, the global variables cannot be modified within the function, unless declared “global” in the function

Global variables (examples)

```
>>> x = 5
>>> def setx(y):
>>>     x = y
>>>     print(x)
>>>
>>> setx(10)
10
>>> x
5
>>> def setx(y):
>>>     global x
>>>     x=y
>>>
>>> setx(10)
>>> x
10
```

Python programming—File I/O

Read File

- Using “read”: Careful! It will be extremely slow if the file cannot fit in memory.

```
>>> f = open(’sample.txt’, ’r’)
>>> s = f.read() # s will be a string of the whole file
>>> f.close()
>>> f = open(’sample.txt’, ’r’)
>>> s = f.read(10) # s will be a string of the first 10 ch
>>> f.close()
```

- Using “readline”:

```
>>> f = open(’sample.txt’, ’r’)
>>> s = f.readline() # s will be the first line
>>> s = f.readline() # s will be the second line
```

Read File

- Loop over the lines of a file:

```
>>> f = open(’sample.txt’,’r’)
>>> for line in f:
>>>    print line
first line

second line

third line
```

Write to a file

```
>>> f = open(’workfile’,’w’)
>>> f.write(’This is a test\nand another test’)
>>> f.close()
>>> f = open(’workfile’,’r’)
>>> s = f.read()
>>> print s
This is a test
and another test
>>> f.close()
```

Check `https://docs.python.org/3/tutorial/inputoutput.html` for more functions for file IO.

Read from CSV file

- CSV: Common-Separated Values is a file stores tabular data
- It may or may not have a “header” (name of columns)
- Simple example:

```
Year,Make,Model
1997,Ford,E350
2000,Mercury,Cougar
```

- Complicated examples: (examples from our homework)

```
"216125","Which one is better, the Tata Safari Storme
or the XUV500?",
```

- Even more complicated:

```
"2228072","What are the full forms of the ""vi"", ""vim"",
and ""gcc"" commands in Linux?"
```

CSV File Reading/Writing

- Use csv.reader (need to import csv module):

```
>>> import csv
>>> f = open(’samplefile.csv’, ’r’)
>>> reader = csv.reader(f)
>>> for row in reader:
...      Do something
...
```

- Write to a csv file:

```
>>> import csv
>>> f = open(’output.csv’, ’w’)
>>> mywriter = csv.writer(f)
>>> mywriter.writerow([’a’, ’b’, ’c’])
```

Python programming—Standard Library

import os

- “os.listdir()”: list the files in a directory:

```
>>> import os
>>> s = os.listdir(’./’) # list the files in the cur
>>> s
[’aa’, ’main.html’, ’main_tmp’]
```

- Running an external command:

```
>>> import os
>>> os.system(’ls -ltr’)
total 72
-rw-r--r--@ 1 hsieh  staff  20322 Mar 30 16:52 main_tm
-rw-r--r--@ 1 hsieh  staff   9609 Mar 31 13:34 main.ht
-rw-r--r--@ 1 hsieh  staff     34 Apr  2 00:07 aa
```

import sys

- sys.argv is important for writing a python script.
- Let test_argv.py be the file with two lines

```
import sys
print sys.argv
```

- Now we can see sys.argv contains information about input parameters

```
In [1]: run test_argv.py
['test_argv.py']

In [2]: run test_argv.py 1 2 3
['test_argv.py', '1', '2', '3']
```

Pickle

- Very important tool! Pickle is used for serializing and de-serializing a python object.
- In other word, pickling is a way to convert a python object into a character stream, which can then be saved to the disk.
- Pickling with default mode

```
In [9]: import cPickle
In [10]: data = [10000, 20000, 30000, 40000]
In [11]: f = open(’data_standard.pl’, ’w’)
In [12]: cPickle.dump(data, f)
In [13]: f.close()
In [14]: f = open(’data_standard.pl’, ’r’)
In [15]: data_loaded = cPickle.load(f)
In [16]: f.close()
In [17]: data_loaded
Out[17]: [10000, 20000, 30000, 40000]
```

Pickle

- Pickling with binary format: smaller file size, but not human-readable

```
In [9]: import cPickle
In [10]: data = [10000, 20000, 30000, 40000]*1000
In [11]: f = open(’data_standard.pl’, ’wb’)
In [12]: cPickle.dump(data, f, -1)  ## -1 means binary format
In [13]: f.close()
In [14]: f = open(’data_binary.pl’, ’rb’)
In [15]: cPickle.dump(data, f, -1)  ## -1 means binary format
In [16]: f.close()
In [17]: !ls -l *.pl
-rw-r--r--@ 1 hsieh  staff  32006 Apr  2 01:29 data_standard.pl
-rw-r--r--@ 1 hsieh  staff  12014 Apr  2 01:30 data_binary.pl
```

random

- The module “random” is often used for generating random numbers.
- Check https://docs.python.org/2/library/random.html for more details.
- Here are some examples

```
>>> import random
>>> random.randint(0,10) ## generate a random number 0<=x<=10
7
>>> a = range(10)
>>> random.shuffle(a)
>>> a
[7, 9, 8, 0, 2, 5, 4, 6, 3, 1]
>>> random.random() ## a random number in [0.0, 1.0)
0.229950285456076
>>> random.gauss(5,0.1)  ## a random gaussian varaible
5.22891420314669
```

Coming up

- More on python programming (numpy)

Questions?