Scientific Python bootcamp
2015 Berkeley Statistics Masters program

Jarrod Millman

342 Evans Hall
UC Berkeley

January 18-19, 2015

Contents

Chapter 1

Python review

Brief review of basic Python. To begin, sign onto your GitHub account and fork the bootcamp repository: ***https://github.com/jarrodmillman/python-bootcamp-2015****.*

Now from your BASH terminal clone your fork(ed) repository. If you are using HTTP to authenticate, you will do something like this:

```
$ cd <to where you keep your source repositories>
$ git clone  https://github.com/<you>/python-bootcamp-2015.git
$ cd python-bootcamp-2015
```

For SSH authentication, do something like this:

```
$ cd <to where you keep your source repositories>
$ git clone git@github.com:<you>/python-bootcamp-2015.git
$ cd python-bootcamp-2015
```

1.1 Introduction

- https://docs.python.org/2/index.html
- https://docs.python.org/2/library/index.html
- https://scipy-lectures.github.io/
- http://software-carpentry.org/v4/python/
- https://docs.python.org/3/howto/pyporting.html

1.1.1 Interpreter

For much of the bootcamp, you should type the example code snippets at an interactive Python terminal. I recommend using either the IPython shell or the IPython notebook. To start an IPython shell, type the following at a BASH prompt:

```
$ ipython
```

To start an IPython notebook, type

```
$ ipython notebook
```

1.1.2 Objects

Everything is an object in Python. Roughly, this means that it can be tagged with a variable and passed as an argument to a function. Often it means that everything has *attributes* and *methods*.

Certain objects in Python are mutable (e.g., lists, dictionaries), while other objects are immutable (e.g., tuples, strings, sets). Many objects can be composite (e.g., a list of dictionaries or a dictionary of lists, tuples, and strings).

1.1.3 Variables

Recall from last semester that variables are not their values in Python (think "I am not my name, I am the person named XXX"). You can think of variables as tags on objects. In particular, variables can be bound to an object of one type and then reassigned to an object of another type without error.

1.1.4 Modules, files, packages, import

While you will often explore things from an interactive Python prompt, you will save your code in files for reuse as well as to document what you've done. You can use Python code saved in a plain text file from a Python prompt or other files by importing it. Typically, this is done at the top of a file (if you are working at a prompt, you just need to import it before you want to use the functionality).

Here are some examples of importing:

```
import math
from math import cos
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

1.1.5 Style

Adopting standard coding conventions is good practice.

- https://www.python.org/dev/peps/pep-0008/
- https://docs.python.org/2/tutorial/controlflow.html#intermezzo-coding-style
- https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
- http://matplotlib.org/devel/coding_guide.html

The first link above is the official "Style Guide for Python Code", usually referred to as PEP8 (PEP is an acronym for Python Enhancement Proposal). There are a couple of potentially helpful tools for helping you conform to the standard. The pep8 package that provides a commandline tool to check your code against some of the PEP8 standard conventions. Similarly, autopep8 provides a tool to automatically format your code so that it conforms to the PEP8 standards. I have used both a little and they seem to work fairly well.

The last two links discuss the NumPy docstring[1] standard. Let's briefly see how you might benefit from NumPy docstrings in practice.

---

[1] Docstrings are an important part of a Python program:

> A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object. All modules should normally have docstrings, and all functions and classes exported by a module should also have docstrings. Public methods (including the `__init__` constructor) should also have docstrings.
>
> — https://www.python.org/dev/peps/pep-0257/

Docstrings also allow for the use of doctests.

> The doctest module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.
>
> — http://docs.python.org/2/library/doctest.html

```
In [1]: import numpy as np

In [2]: np.ndim?
Type:        function
String form: <function ndim at 0x7fcabd864938>
File:        /usr/lib64/python2.7/site-packages/numpy/core/fromnumeric.py
Definition:  np.ndim(a)
Docstring:
Return the number of dimensions of an array.

Parameters
----------
a : array_like
    Input array.  If it is not already an ndarray, a conversion is
    attempted.

Returns
-------
number_of_dimensions : int
    The number of dimensions in `a`.  Scalars are zero-dimensional.

See Also
--------
ndarray.ndim : equivalent method
shape : dimensions of array
ndarray.shape : dimensions of array

Examples
--------
>>> np.ndim([[1,2,3],[4,5,6]])
2
>>> np.ndim(np.array([[1,2,3],[4,5,6]]))
2
>>> np.ndim(1)
0
```

Exercises

- What happens if you type `np.ndim??` (i.e., use two question marks)?
- Type `np.tril?` at an IPython prompt. What does `np.tril` do?
- Type `np.ndarray?` at an IPython prompt. Briefly skim the docstring. `ndarray` is the basic datastructure provided by NumPy. We will examine it in much more detail in the next chapter.
- Type `np.` followed by the `<Tab>` key at an IPython prompt. Choose two or three of the completions and use `?` to view their docstrings. In particular, pay attention to the examples provided near the end of the docstring and see whether you can figure out how you might use this functionality. Use on them as well.

1.1.6 Python 2 vs. 3

Python 3 is a new version of Python, which is incompatible with Python 2. We will use Python 2 in this bootcamp, but Python 3 is the future. Due to the large installed codebase of Python 2, the transition will take years.

If you are writing new Python code at this point, require Python 2.7 as the minimum support version. You should also import the following functionality from the `__future__` module.

```
from __future__ import (absolute_import,
                        division,
                        print_function,
                        unicode_literals)
```

While we will be using Python 2 for this bootcamp, in the near future you may consider using the `future` package.[2] The idea is that by using this package and adding a few imports to the top of your Python modules you can write "predominantly standard, idiomatic Python 3 code that then runs similarly on Python 2.6/2.7 and Python 3.3+."

1.2 Data Structures

- https://docs.python.org/2/library/stdtypes.html
- https://docs.python.org/2/tutorial/datastructures.html
- https://docs.python.org/2/reference/datamodel.html

1.2.1 Numbers

Python has integers, floats, and complex numbers with the usual operations (beware: division).

```
In [1]: 2/3
Out[1]: 0

In [2]: from __future__ import division

In [3]: 2/3
Out[3]: 0.6666666666666666

In [4]: x = 1.1

In [5]: x.
x.as_integer_ratio  x.hex               x.real
x.conjugate         x.imag
x.fromhex           x.is_integer

In [5]: x * 2
Out[5]: 2.2

In [6]: x**2
Out[6]: 1.2100000000000002

In [7]: 100000**10
Out[7]: 100000000000000000000000000000000000000000000000000L

In [8]: 100000**100
Out[8]: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

[2] https://pypi.python.org/pypi/future

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000L

In [9]: cos(0)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-edaadd132e03> in <module>()
----> 1 cos(1)

NameError: name 'cos' is not defined

In [10]: import math

In [11]: math.cos(0)
Out[11]: 1.0

In [12]: math.cos(math.pi)
Out[12]: -1.0

In [13]: (type(1), type(1.1), type(1+2j))
Out[13]: (int, float, complex)
```

The above line is an example of a composite object called a tuple, which we will discuss more in § 1.2.3 below. At an IPython prompt, use `type?` to see what `type` does.

The `math` package in the standard library includes many additional numerical operations.

```
In [14]: math.
math.acos       math.degrees    math.fsum       math.pi
math.acosh      math.e          math.gamma      math.pow
math.asin       math.erf        math.hypot      math.radians
math.asinh      math.erfc       math.isinf      math.sin
math.atan       math.exp        math.isnan      math.sinh
math.atan2      math.expm1      math.ldexp      math.sqrt
math.atanh      math.fabs       math.lgamma     math.tan
math.ceil       math.factorial  math.log        math.tanh
math.copysign   math.floor      math.log10      math.trunc
math.cos        math.fmod       math.log1p
math.cosh       math.frexp      math.modf
```

Exercises

Using the section on "Built-in Types" from the official "The Python Standard Library" reference (follow the first link at the top of § 1.2), figure out how to compute:

1. $3 \leq 4$,
2. $3 \mod 4$,
3. $|-4|$,
4. $\left(\lceil \frac{3}{4} \rceil \times 4\right)^3 \mod 2$, and
5. $\sqrt{-1}$.

Questions

1. How do you get the list of completions for `x.`?

2. What is the difference in the old and new behavior of division?

3. Read the "Truth Value Testing" and "Boolean Operations" subsections at the top of the "Built-in Types" section of the Library reference. How does this compare to how R handles things?

1.2.2 Strings

Strings are immutable sequences of (zero or more) characters.

Sequences

Unlike numbers, Python strings are container objects. Specifically, it is a sequence. Python has several sequence types including strings, tuples, and lists. Sequence types share some common functionality, which we can demonstrate with strings.

- **Indexing** To see how indexing works in Python let's use the string containing the digits 0 through 9.

  ```
  In [1]: import string

  In [2]: string.digits
  Out[2]: '0123456789'

  In [3]: string.digits[1]
  Out[3]: '1'

  In [4]: string.digits[-1]
  Out[4]: '9'
  ```

  Note that indexing starts at 0 (unlike R and Fortran, but like C). Also negative integers index starting from the end of the sequence. You can find the length of a sequence using the `len()` function.

- **Slicing** Slicing allows you to select a subset of a string (or any sequence) by specifying start and stop indices as well as a step, which you specify using the `start:stop:step` notation inside of square braces.

  ```
  In [5]: string.digits[1::2]
  Out[5]: '13579'

  In [6]: string.digits[9::-1]
  Out[6]: '9876543210'
  ```

- **Subsequence testing**

  ```
  In [7]: '23' in string.digits
  Out[7]: True

  In [16]: '25' not in string.digits
  Out[16]: True
  ```

String methods

```
In [1]: string1 = "my string"

In [2]: string1.
string1.capitalize  string1.islower     string1.rpartition
string1.center      string1.isspace     string1.rsplit
string1.count       string1.istitle     string1.rstrip
string1.decode      string1.isupper     string1.split
string1.encode      string1.join        string1.splitlines
string1.endswith    string1.ljust       string1.startswith
string1.expandtabs  string1.lower       string1.strip
string1.find        string1.lstrip      string1.swapcase
string1.format      string1.partition   string1.title
string1.index       string1.replace     string1.translate
string1.isalnum     string1.rfind       string1.upper
string1.isalpha     string1.rindex      string1.zfill
string1.isdigit     string1.rjust

In [2]: string1.upper()
Out[2]: 'MY STRING'

In [3]: string1.upper?
Type:        builtin_function_or_method
String form: <built-in method upper of str object at 0x7fa136f8ced0>
Docstring:
S.upper() -> string

Return a copy of the string S converted to uppercase.

In [4]: string1 + " is your string."
Out[4]: 'my string is your string.'

In [5]: "*"*10
Out[5]: '**********'

In [6]: string1[3:]
Out[6]: 'string'

In [7]: string1[3:4]
Out[7]: 's'

In [8]: string1[4::2]
Out[8]: 'tig'

In [9]: string1[3:5] = 'ts'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-d7a58dc91703> in <module>()
----> 1 string1[3:5] = 'ts'

TypeError: 'str' object does not support item assignment

In [10]: string1.__
```

```
string1.__add__           string1.__len__
string1.__class__         string1.__lt__
string1.__contains__      string1.__mod__
string1.__delattr__       string1.__mul__
string1.__doc__           string1.__ne__
string1.__eq__            string1.__new__
string1.__format__        string1.__reduce__
string1.__ge__            string1.__reduce_ex__
string1.__getattribute__  string1.__repr__
string1.__getitem__       string1.__rmod__
string1.__getnewargs__    string1.__rmul__
string1.__getslice__      string1.__setattr__
string1.__gt__            string1.__sizeof__
string1.__hash__          string1.__str__
string1.__init__          string1.__subclasshook__
```

Exercises

At an interactive Python prompt, type `x = "The ant wants what all ants want."`. Using string indexing, slicing, subsequence testing, and methods, solve the following:

1. Convert the string to all lower case letters (don't change x).
2. Count the number of occurrences of the substring `"ant"`.
3. Create a list of the words occurring in `x`. Make sure to remove punctuation and convert all words to lowercase.
4. Using only string methods on `x`, create the following string: `"The chicken wants what all chickens want."`
5. Using indexing and the `+` operator, create the following string: `"The tna wants what all ants want."`
6. Do the same thing except using a string method instead.

Questions

1. How do the string method's `split` and `rsplit` differ? [Hint: use `?` to view the method's docstrings.]
2. What happens when you multiple a string by a number? How does this relate to the string method `__mul__`? [Hint: look at the docstring.]
3. How does the `len()` function know how to find the length of a sequence?
4. How do the `in` and `not in` operators work?

1.2.3 Tuples

Tuples are immutable sequences of (zero or more) objects. Functions in Python often return tuples.

```
In [1]: x = 1; y = 2

In [2]: xy = (x, y)

In [3]: xy
Out[3]: (1, 2)
```

```
In [4]: xy[1]
Out[4]: 2

In [5]: xy[1] = 3
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-b22951f8a33e> in <module>()
----> 1 xy[1] = 3

TypeError: 'tuple' object does not support item assignment

In [6]: (x, y)
Out[6]: (1, 2)

In [7]: x, y
Out[7]: (1, 2)
```

Exercises

1. Note that `x, y` and `(x, y)` both print the same string. To see why that is assign them to variables and check their type.

2. Create the following `x=5` and `y=6`. Now swap their values. (How would you do this in R?)

1.2.4 List

Lists are mutable sequences of (zero or more) objects.

```
In [1]: dice = [1, 2, 3, 4, 5, 6]

In [2]: dice[1::2]
Out[2]: [2, 4, 6]

In [3]: dice[1::2] = dice[::2]

In [4]: dice
Out[4]: [1, 1, 3, 3, 5, 5]

In [5]: dice*2
Out[5]: [1, 1, 3, 3, 5, 5, 1, 1, 3, 3, 5, 5]

In [6]: dice+dice[::-1]
Out[6]: [1, 1, 3, 3, 5, 5, 5, 5, 3, 3, 1, 1]

In [7]: 1 in dice
Out[7]: True
```

Exercises

1. Create a list of numbers. Reverse the order of the items in the list using slicing. Now reverse the order of the items using a list method. How does using the method differ from slicing? Do you think you think tuples have a method to reverse the order of its items? Why or why not? Check to see if you are correct or not.

2. Using a list method sort your numbers. Create a list of strings and sort it. Put your list of numbers and strings together in one list and sort it. What happened?

1.2.5 Dictionaries

Dictionaries are mutable, unordered collections of key-value pairs.

```
In [99]: students = {"Jarrod Millman": [10, 11, 9],
   ....:             "Thomas Kluyver":  [11, 9, 10],
   ....:             "Stefan van der Walt": [12, 9, 9]}

In [100]: students
Out[100]:
{'Jarrod Millman': [10, 11, 9],
 'Stefan van der Walt': [12, 9, 9],
 'Thomas Kluyver': [11, 9, 10]}

In [102]: students.keys()
Out[102]: ['Thomas Kluyver', 'Stefan van der Walt', 'Jarrod Millman']

In [103]: students["Jarrod Millman"]
Out[103]: [10, 11, 9]

In [104]: students["Jarrod Millman"][1]
Out[104]: 11
```

1.2.6 Sets

Sets are immutable, unordered collections of unique elements.

```
In [1]: x =  {1, 2, 4, 1, 4}

In [2]: x
Out[2]: {1, 2, 4}

In [3]: x.
x.add                          x.issubset
x.clear                        x.issuperset
x.copy                         x.pop
x.difference                   x.remove
x.difference_update            x.symmetric_difference
x.discard                      x.symmetric_difference_update
x.intersection                 x.union
x.intersection_update          x.update
x.isdisjoint
```

1.2.7 And more

```
In [1]: import collections

In [2]: collections.
collections.Callable         collections.MutableSequence
collections.Container        collections.MutableSet
collections.Counter          collections.OrderedDict
collections.Hashable         collections.Sequence
```

```
collections.ItemsView        collections.Set
collections.Iterable         collections.Sized
collections.Iterator         collections.ValuesView
collections.KeysView         collections.defaultdict
collections.Mapping          collections.deque
collections.MappingView      collections.namedtuple
collections.MutableMapping
```

1.3 Built-in functions

- https://docs.python.org/2/library/functions.html

Python has several built-in functions (you can find a full list using the link above). We've already used a few (e.g., `len()`, `type()`, `print()`). Here are a few more that we you will find useful.

1.3.1 zip

```
In [108]: zip([1, 2], ["a", "b"])
Out[108]: [(1, 'a'), (2, 'b')]
```

1.3.2 enumerate

```
In [109]: enumerate(["a", "b"])
Out[109]: <enumerate at 0x7f5e3e018640>

In [110]: list(enumerate(["a", "b"]))
Out[110]: [(0, 'a'), (1, 'b')]
```

Question

- What do the built-in functions `abs()`, `all()`, `any()`, `dict()`, `dir()`, `id()`, `list()`, and `set()` do? Make sure to use `?` from the IPython prompt as well as looking at the documentation in the official Python Standard Library reference (use the above link).

1.4 Control flow

- https://docs.python.org/2/tutorial/controlflow.html

1.4.1 If-then-else

- https://docs.python.org/2/tutorial/controlflow.html#if-statements

```
In [44]: x = 2

In [45]: if x < 2:
   ....:     print("Yes")
   ....: else:
   ....:     print("No")
   ....:
No
```

1.4.2 For-loops (and list comprehension)

- https://docs.python.org/2/tutorial/controlflow.html#for-statements
- https://docs.python.org/2/whatsnew/2.0.html#list-comprehensions

```
In [49]: for x in [1,2,3,4]:
   ....:     print(x)
   ....:
1
2
3
4


In [50]: for x in [1,2,3,4]:
   ....:      print(x, end="")
   ....:
1234
```

Building up a list piece-by-piece is a common task, which can easily be done in a for-loop. List comprehension provide a compact syntax to handle this task.

```
In [64]: x = [1, 2, 3, 4]

In [65]: zip(x, x[::-1])
Out[65]: [(1, 4), (2, 3), (3, 2), (4, 1)]

In [66]: [y for y in zip(x, x[::-1]) if y[0] > y[1]]
Out[66]: [(3, 2), (4, 1)]
```

Exercises

- Write a for-loop that produces `[(3, 2), (4, 1)]` from `x`. How does it compare to the list comprehension above?
- Use `print?` to see what the `end` argument to the print function does. Are there any additional arguments to `print()`? If so, try using the additional arguments.
- Find the section on the `range()` function in Python tutorial. Rewrite the two for-loops above using it rather than explicitly constructing the list of numbers.
- See what `[1, 2, 3] + 3` returns. Try to explain what happened and why. In R, when you add a scalar to a vector the result is the element-wise addition.

  ```
  > 3 + c(1,2,3)
  [1] 4 5 6
  ```

  Use list comprehension to perform element-wise addition of a scalar to a list of scalars.

1.4.3 Functions

- https://docs.python.org/2/tutorial/controlflow.html#defining-functions

```
In [105]: def add(x, y):
   .....:     return x+y
   .....:

In [106]: add(2, 3)
```

```
Out[106]: 5

In [105]: def add(x, y=1):
   .....:     return x+y
   .....:

In [106]: add(3)
Out[106]: 4
```

1.5 Exercise: DNA

For this exercise, please see `ex1/dna.py`.

1.6 Classes

- https://docs.python.org/2/tutorial/classes.html

```
In [224]: class Rectangle(object):
   .....:     def __init__(self, height, width):
   .....:         self.height = height
   .....:         self.weight = width
   .....:     def __repr__(self):
   .....:         return "{0} by {1}".format(self.height, self.width)
   .....:     def area(self):
   .....:         return self.height*self.width
   .....:

In [225]: x = Rectangle(10,5)

In [228]: x
Out[228]: 10 by 5

In [229]: x.area()
Out[229]: 50
```

1.7 Exercise: Cipher

For this exercise, please see `ex2/cipher.py`.

1.8 Data formats

1.8.1 CSV

- https://docs.python.org/2/library/csv.html

The Python standard library provides a package for reading and writing CSV files. This is a somewhat low-level library, so in practice you will often use NumPy, SciPy, or Pandas CSV functionality.

1.8.2 JSON

- https://docs.python.org/2/library/json.html

However the JSON package in the standard library is much more useful.

```
In [182]: import json

In [183]: x = {"name": "Jarrod", "department": "Biostatistics"}

In [186]: with open("tmp.json", "w") as outfile:
   .....:     json.dump(x, outfile)
   .....:

In [187]: cat tmp.json
{"department": "Biostatistics", "name": "Jarrod"}

In [192]: with open("tmp.json") as infile:
   .....:     y = json.load(infile)
   .....:

In [193]: y
Out[193]: {u'department': u'Biostatistics', u'name': u'Jarrod'}
```

Note that `cat` is not a Python statement. IPython is clever enough to quess that you want it to call out to the underlying operating system.

Exercise

- One of the nice things above the JSON format is that it so well structured that it easy for a machine to parse, but simple enough that it easy for humans to read. By default `json.dump` writes everything out to disk without line breaks. For readability purposes, use `json.dump?` to figure out how to pretty-print the text as well as sort it alphabetically by key.

1.8.3 HTML

We will use Thomas Kluyver's web scraping example notebook for this section. You can view a rendered version of it here. To get an interactive version of it, you can do the following from your BASH prompt:

```
$ git clone https://github.com/dlab-berkeley/python-fundamentals.git
$ cd python-fundamentals/cheat-sheets/
$ ipython notebook Web-Scraping.ipynb
```

1.9 Standard library

- https://docs.python.org/2/tutorial/stdlib.html

1.9.1 Batteries included

Python provides a wealth of functionality in its huge standard library. We've already seen several (e.g., math, csv, json). If you need some functionality the standard library is one of the first places to look.

Here are a couple packages that you may find useful.

1.9.2 os

- https://docs.python.org/2/tutorial/stdlib.html#operating-system-interface

```
In [147]: import os

In [148]: os.getcwd()
Out[148]: '/home/jarrod'

In [149]: pwd
Out[149]: u'/home/jarrod'
```

Exercise

- Use `os?` and `dir(os)` to explore the os package.

1.9.3 re

- https://docs.python.org/2/howto/regex.html

The `re` package provides support for regular expressions.

1.10 Exercise: State of the Union addresses

For this exercise, you will revisit the State of the Union Addresses, which you worked with in Statistics 243 (problem set 3).

I've provided a Python script (`ex3/munge.py`) that scraps the web, processes the speeches, and saves the processed information to a JSON file (`ex3/speeches.json`).

You should write a Python script[3] to:

1. Load the data from the JSON file.
2. Count the number of speeches (you may need to do some data cleaning).
3. Create a list with just the names of the presidents.
4. Create a list with just the dates of the speeches.
5. Write a function that takes a speech object a returns the tuple `(<president>, <number of words>, <date>)`
6. Create a list of tuples by calling your function on all the speech objects.
7. Create a list of the speech dates sorted based on the length of speech.

For the last task you may want to take a look at the Python Sorting How To:

- https://wiki.python.org/moin/HowTo/Sorting

[3]You will probably need to explore the data interactively from and IPython prompt and in tandem write your script

Chapter 2

NumPy, SciPy, and matplolib

Introduce Python's core numerical, scientific, and plotting packages.

- *Fernando Pérez, Brian E. Granger, and John D. Hunter. "Python: an ecosystem for scientific computing."* Computing in Science & Engineering *13, no. 2 (2011): 13-21.*
- *Stéfan van der Walt, S. Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation."* Computing in Science & Engineering *13, no. 2 (2011): 22-30.*
- *John D. Hunter. "Matplotlib: A 2D graphics environment."* Computing in Science & Engineering *9, no. 3 (2007): 0090-95.*

2.1 Introduction

- `http://docs.scipy.org/doc/`
- `http://matplotlib.org/gallery.html`
- `https://scipy-lectures.github.io/`
- `https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks`

2.2 NumPy and matplotlib

2.2.1 Exercise: lock 'n load

For this exercise please work through Stéfan van der Walt's NumPy lock 'n load.

- `https://github.com/stefanv/teaching/tree/master/2008_numpy_load_n_load`

2.2.2 ndarray

- `http://scipy-lectures.github.io/intro/numpy/array_object.html`
- `http://scipy-lectures.github.io/intro/numpy/operations.html`

2.2.3 2D plotting

- `http://scipy-lectures.github.io/intro/matplotlib/matplotlib.html`

2.3 Example: random walk redux

Recall the random walk simulation from 243.

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2)

code  = {"up": (0,1),
         "down": (0,-1),
         "left": (-1,0),
         "right": (1,0)}

def random_2d_walk(nsteps=100):
    steps = np.random.choice(code.keys(), nsteps)
    walk = np.array([code[step] for step in steps])
    xy = walk.cumsum(axis=0)
    return xy

xy = random_2d_walk()
plt.plot(xy[:,0], xy[:,1])
plt.savefig("test.png")
```

Here is a version implemented as a class.

```
import numpy as np
import matplotlib.pyplot as plt

class Random2DWalk(object):

    def __init__(self, start=(0,0)):
        self.steps = ["start"]
        self.walk = np.array([start])
        self._code = {"up": (0,1),
                      "down": (0,-1),
                      "left": (-1,0),
                      "right": (1,0)}

    def __str__(self):
        current_position = self.position()[-1]
        total_steps = len(self.steps) - 1
        message = "After {0} steps you are at position: {1}"
        return message.format(total_steps, current_position)

    def step(self, nsteps=100):
        steps = np.random.choice(self._code.keys(), nsteps)
        walk = np.array([self._code[step] for step in steps])
        self.steps += steps
        self.walk = np.vstack([self.walk, walk])
        return None

    def position(self):
        return self.walk.cumsum(axis=0)
```

```
    def plot(self):
        xy = self.position()
        plt.plot(xy[:,0], xy[:,1])
        plt.savefig("test.png")
        return None

np.random.seed(2)
rw = Random2DWalk()
print rw
rw.step()
print rw
print rw.steps[:5]
print rw.walk[:5]
rw.plot()

## After 0 steps you are at position: [0 0]
## After 100 steps you are at position: [-9  7]
## ['start', 'down', 'left', 'right', 'down']
## [[ 0  0]
##  [ 0 -1]
##  [-1  0]
##  [ 1  0]
##  [ 0 -1]]
```

2.3.1 Exercise: Sierpinski triangle

Write a Python script to construct Sierpinski triangle using the following algorithm:

1. Choose 3 points in the plane (forming a triangle).
2. Choose another "starting" pointing (current position).
3. Randomly choose one of the corners of the triangle.
4. Move halfway from your current position to the selected corner.
5. Plot the new current position.
6. Repeat from step 3 (for 100 times).

2.4 SciPy

- http://scipy-lectures.github.io/intro/scipy.html

2.5 Exercise: State of the Union addresses

For this exercise, you will revisit the State of the Union Addresses. Load the data, use whatever tools and analysis you want, and use matplotlib to make plots.

Chapter 3

Scikit Image

Introduction to image processing with `scikit-image` *by Stéfan van der Walt. Stéfan just the Berkeley Institute for Data Science. Previously, he was a senior lecturer in applied mathematics at Stellenbosch University, South Africa, and an associate project scientist in the astronomy department, UC Berkeley. He has been involved in the development of scientific open source software since 2003 and enjoys teaching Python at workshops and conferences. Stéfan is the founder of scikit-image (an image-processing library written in the Python language) and a contributor to NumPy, SciPy and Dipy.*

3.1 Links

- http://scikit-image.org/