UC Berkeley Extension

Python for Data Analysis and Scientific Computing

X433.3 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Course Content Outline

- **Introduction to Python®**
- Python - pros and cons
- Installing the environment with core packages
- Python modules, packages and scientific blocks
- Working with the shell, IPyton and the editor HW1 (5pts)

- **Basic language specifics 1/2**
- Basic arithmetic operations, assignment operators, data types, containers
- Control flow (if/elif/else)
- Conditional expressions
- Iterative programming (for/continue/while/break)
- Functions: definition, return values, local vs. global variables

- **Basic language specifics 2/2**
- Classes / Functions (cont.): objects, methods, passing by value and reference
- Scripts, modules, packages
- I/O interaction with files
- Standard library
- Exceptions HW2 (5pts)

- **NumPy 1/3**
- Why NumPy?
- Data type objects
- NumPy arrays
- Indexing and slicing of arrays

- **Matplotlib**
- What is Matplotlib?
- Basic plotting
- Tools: title, labels, legend, axis, points, subplots, etc.
- Advanced plotting: scatter, pie, bar, 3D plots, etc. HW3 (5pts)

Course Content Outline

- **NumPy 2/3**
- Array operations
- Reductions
- Broadcasting
- Array: shaping, reshaping, flattening, resizing, dimension changing
- Data sorting — Midterm / Project proposal due (30pts)

- **NumPy 3/3**
- Type casting
- Masking data
- Organizing arrays
- Loading data files
- Dealing with polynomials
- Good coding practices

- **Scipy 1/2**
- What is Scipy?
- Working with files
- Algebraic operations
- The Fast Fourier Transform
- Signal Processing — HW4 (5pts)

- **Scipy 2/2**
- Interpolation
- Statistics
- Optimization

- **Project**
- Project Presentation — Final Project (40pts)

Course Content Outline

- **Methods of Instruction**

  Lectures and in-class discussions will be the main tools of instruction. Homework problems will help students absorb the material and get to practice in their free time. Since this is a more specialized course that naturally delves into specific topics a midterm exam will be held to assess the performance of the students midway through the course.

- **Credit Requirements**

  Students must complete all homework assignments and pass all exams. They also must receive a passing grade on the final exam to receive a passing grade in the course.

- **Course Grade Weighting**

  1. In-class participation: 10%
  2. Homework: 20% - Wednesdays - use the weekend to prepare
  3. Midterm 30% - Monday - use the weekend to prepare
  4. Final Project 40% - Wednesday

- **Optional Reading for the Course**

  Title: Python for Data Analysis

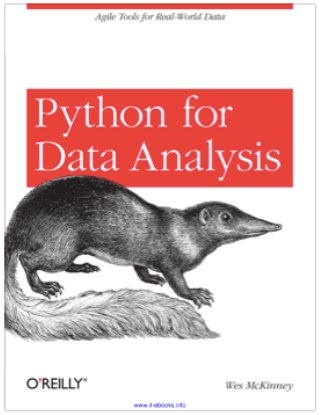
  ISBN-10: 1449319793 and ISBN-13: 978-1449319793

  Author(s): Wes McKinney

  Publisher: O'Reilly Media

  Edition Number: 1 edition

  Publication Date: November 1, 2012



Course Content Outline

- **Introduction to Python®**
- Python - pros and cons
- Installing the environment with core packages
- Python modules, packages and scientific blocks
- Working with the shell, IPyton and the editor HW1

- **Basic language specifics 1/2**
- Basic arithmetic operations, assignment operators, data types, containers
- Control flow (if/elif/else)
- Conditional expressions
- Iterative programming (for/continue/while/break)
- Functions: definition, return values, local vs. global variables

- **Basic language specifics 2/2**
- Functions (cont.): objects, methods, passing by value and reference
- Scripts, modules, packages
- I/O interaction with files
- Standard library
- Exceptions

- **NumPy 1/3**
- Why NumPy?
- Data type objects
- NumPy arrays
- Indexing and slicing of arrays

- **Matplotlib**
- What is Matplotlib?
- Basic plotting
- Tools: title, labels, legend, axis, points, subplots, etc.
- Advanced plotting: scatter, pie, bar, 3D plots, etc.

Python for Data Analysis and Scientific Computing

- Python implementation started in 1989 by Guido Van Rossum
- Python intended to be the successor of the ABC language
- Python 2.0 released in late 2000
- Python 3.0 released in late 2008 and was mostly incompatible with its predecessor
- EOL for Python 2.7 was initially set at 2015, but postponed to 2020
- Google announced to work on a transcompiler for Python 2.7 – Jan.2017

Python for Data Analysis and Scientific Computing

- Python comparison to Java:
  - Dynamic vs static typing
  - Braces vs indentation
  - Speed and portability
  - Interpreted vs compiled

Python for Data Analysis and Scientific Computing

- Python with NumPy and SciPy – why using it?
  - Because you can develop proof-of-concept solutions fairly easily and free
  - Open-source software that is widely spread
  - Fast development time
  - Many scientific libraries
  - Mobile computing for smartphone development by using Kivy for Android and iOS
  - Web-site incorporation by using Django

Python for Data Analysis and Scientific Computing

- Python – pros and cons:

  Pros

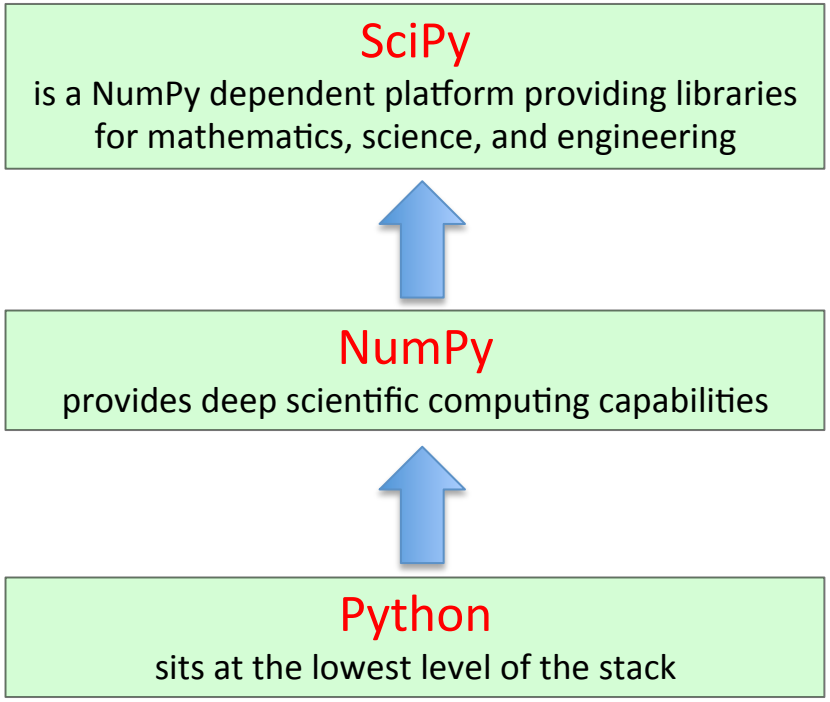
  - High-level object-oriented language that is easy to learn
  - Good readability of the code
  - Quick development time
  - Rich scientific libraries and many others like web server management, serial port access, etc.
  - Easy to share and communicate results
  - Easy to read, manipulate and process data
  - A single environment for many tasks. No need to learn another language
  - Free and open-source software that is widely spread

  Cons

  - Speed of execution since it is higher level language
  - Mobile computing for smartphone development (unless you use Kivy: Linux, Windows, OS X, Android and iOS)
  - It is not included in Web browsers, because it is hard to secure
  - Design flows because it is dynamically typed language and some errors show at runtime such as syntactic errors (mistyping variable names)
  - Python can be both interpreted (.py) and compiled (.pyc)
  - Documentation isn't at the level of PHP or Java
  - Not all algorithms found in other specialized platforms or toolboxes (Matlab) are available

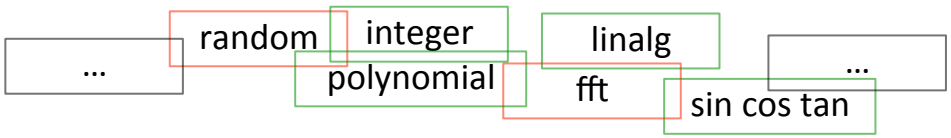
Introduction to Python

- Basic stack using NumPy and SciPy for Python:



Introduction to Python

- Python + NumPy
  - … is like Matlab
  - NumPy provides the code for any fundamental data structure:



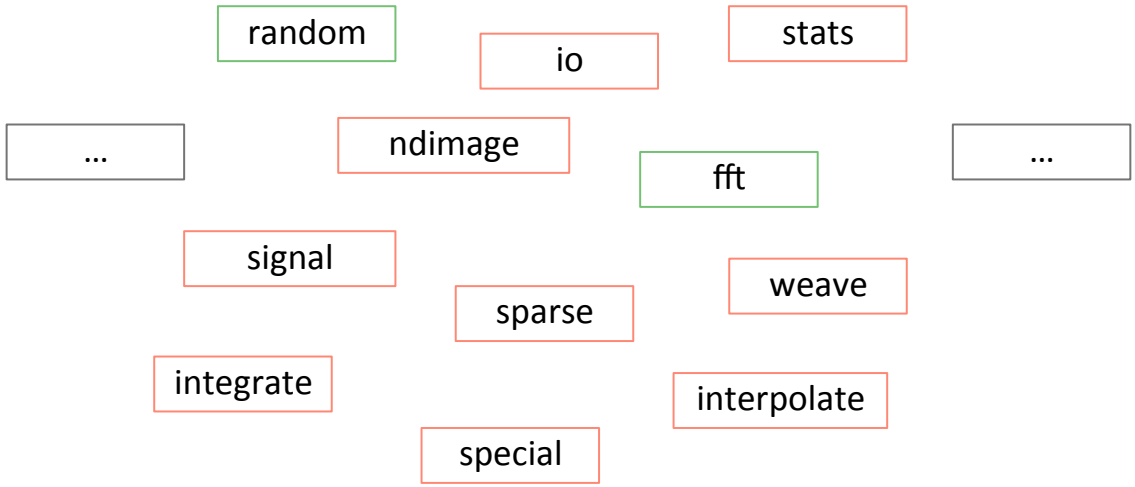
  - There two basic data structures in NumPy:

NDArray
- is an N-dimensional array type
- describes a collection of "items" of the same type
- the items are indexed using N integers

UFunc
- provides fast operations on arrays
- operates element-by-element on arrays
- example are: *sin, cos, tan, +, -, etc.*

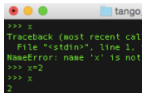
Introduction to Python

- Python + NumPy + SciPy
  - … is like Matlab + toolboxes
  - SciPy provides many specialized packages, which makes it very competitive and powerful platform



Introduction to Python

- Choosing your environment:
  - Using the shell (primarily for Linux and OSX users):



    - Obtaining Python with core packages:
      https://www.python.org/downloads/
    - Obtaining the NumPy & SciPy libraries:
      http://www.scipy.org/scipylib/download.html
  - GUI environments using an interactive editor and debuggers for Python:

IP[y]:

    - IPython – a GUI environment for interactive Python use with shells (terminal- and Qt-based):
      http://www.ipython.org/

    - Python(x,y) – free scientific and engineering environment for numerical computations, data analysis and data visualization:

    - Canopy – Python based environment providing core scientific analytic and scientific Python packages, for scientific development and visualization:
      https://store.enthought.com/

    - IEP – Interactive Editor for Python, is an interactive cross-platform Python IDE that is simple and efficient. IEP consists of an **editor**, a **shell**, and a set of **tools** to help the developer
      http://www.iep-project.org/

    - Pyzo – a free open-source environment based on Python:
      http://www.pyzo.org/downloads.html
    - … other

Introduction to Python

- Using the shell (linux and OSX users):
  - Using 'pip': a tool for installing Python packages from the Python Package Index
    - In your terminal type:

      $ sudo easy_install pip

      $ sudo pip install --upgrade pip
    - Then you can install python and some packages:

      $ pip install python

      $ pip install numpy

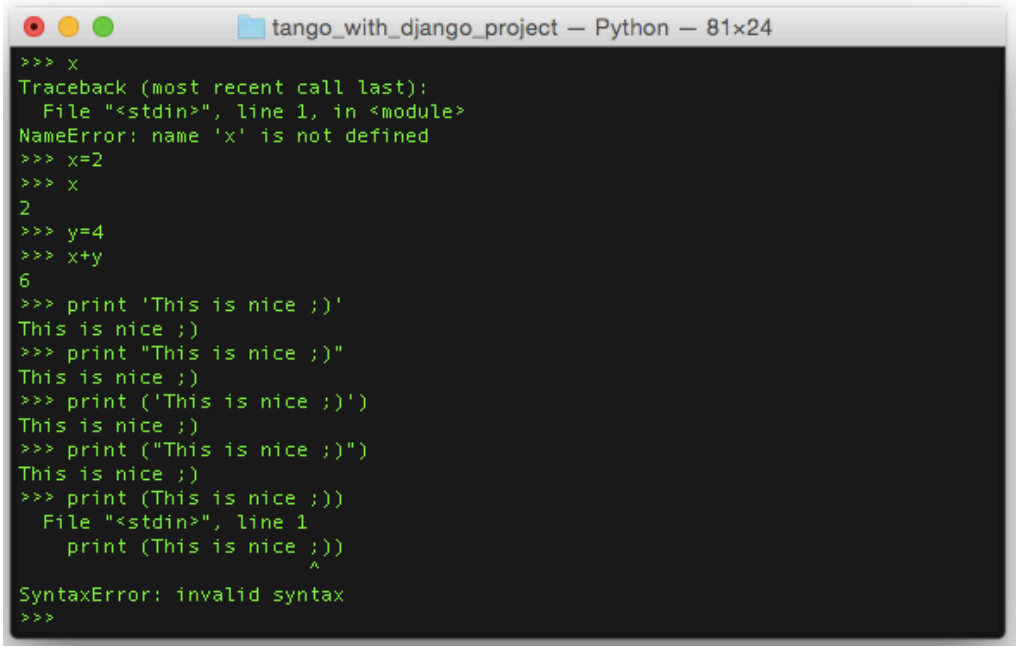
      $ pip install scipy

      $ pip install matplotlib

      $ pip install django

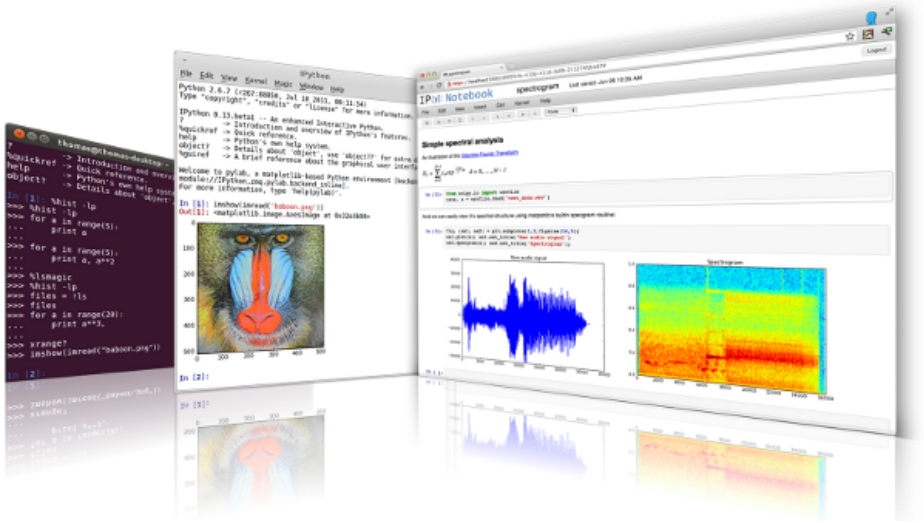
Introduction to Python

- Working with the shell
  - The shell:
    - present on OSX, Unix and Linux systems
    - provides a quick and easy to use plain text environment for swift execution of simple commands
    - it is flexible and lightweight
    - it has a Unix/Linux look and feel and is not used only for Python and it's packages and modules



Introduction to Python

- Working with IPyton
  - IPython
    - Is a GUI environment for interactive Python use with shells (terminal- and Qt-based) and these key features:
      - Interpreters that can be embedded into users' custom projects
      - It has a build-in web-based interactive notebook environment with complete set of shell features and support for embedded figures, animations and multimedia
      - Use of GUI toolkits and support for interactive data visualization
      - A high-performance library for parallel computing portable to multicore systems, clusters and cloud computing



Introduction to Python

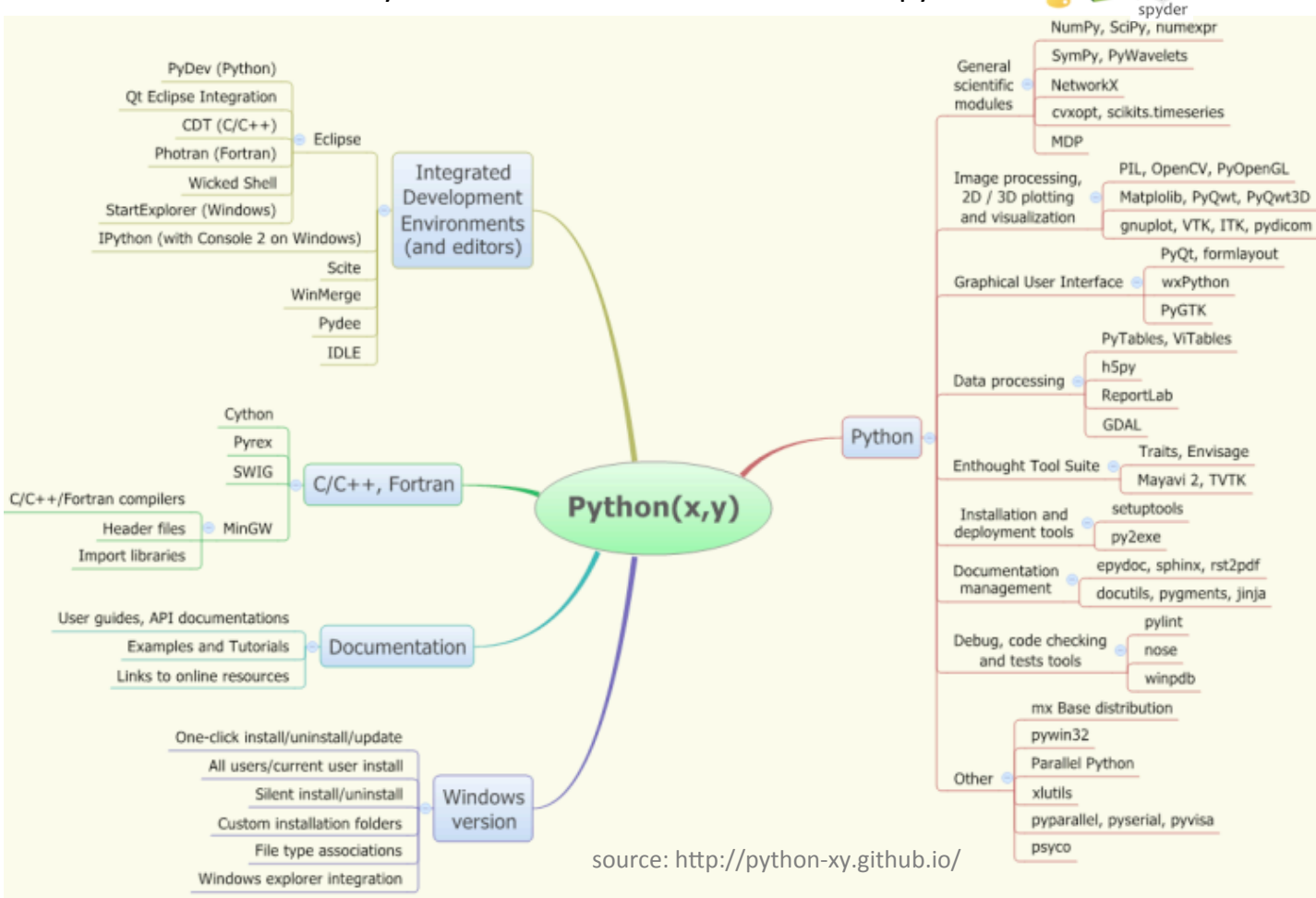
- Working with IPyton and the editor
  - IPython and the editor
    - History, persistent across sessions
    - Caching of output results during a session
    - Extensible tab completion, with support by default for completion of python variables and keywords, filenames and function keywords
    - Comprehensive system of 'magic' commands for controlling the environment and performing many tasks related either to IPython or the operating system
    - A rich configuration system with easy switching between different setup
    - System shell access with user-extensible alias setup
    - Session logging and reloading
    - Code is easy to embed and use in other Python programs and GUIs
    - Embedded debugger

Introduction to Python

- Working with alternative environments:
  - Python(x,y) – a free scientific-oriented Python Distribution based on Qt and Spyder –

It offers:

- object-oriented prog.
- MATLAB-like syntax
- parallel computing
- 2D & 3D plotting
- any level coding



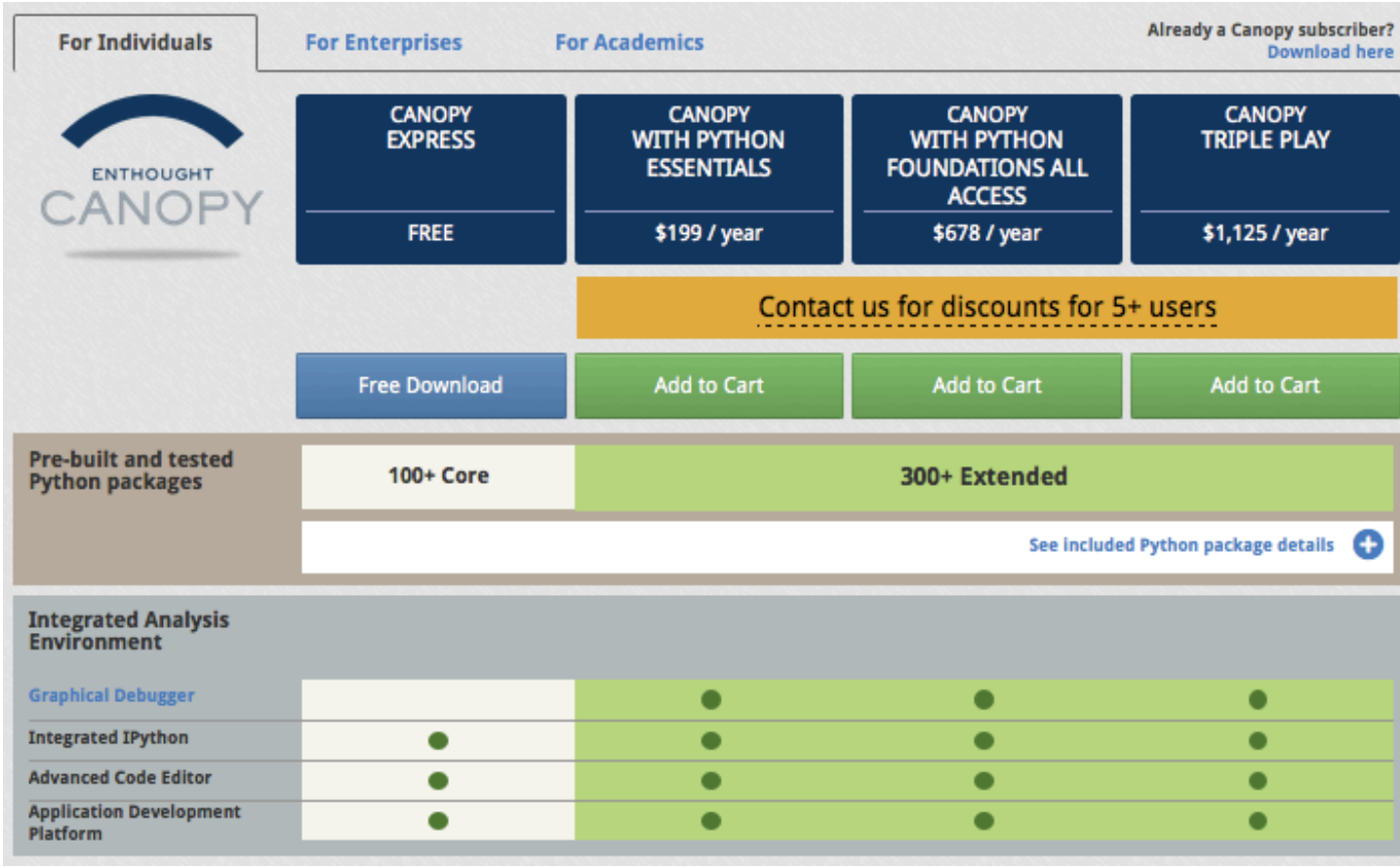
source: http://python-xy.github.io/

Introduction to Python

- Working with alternative environments:
  - Canopy – scientific and analytic Python package distribution. has a free version, but limited

It offers:

- object-oriented prog.
- MATLAB-like syntax
- many packages
- graphical debugger is NOT free



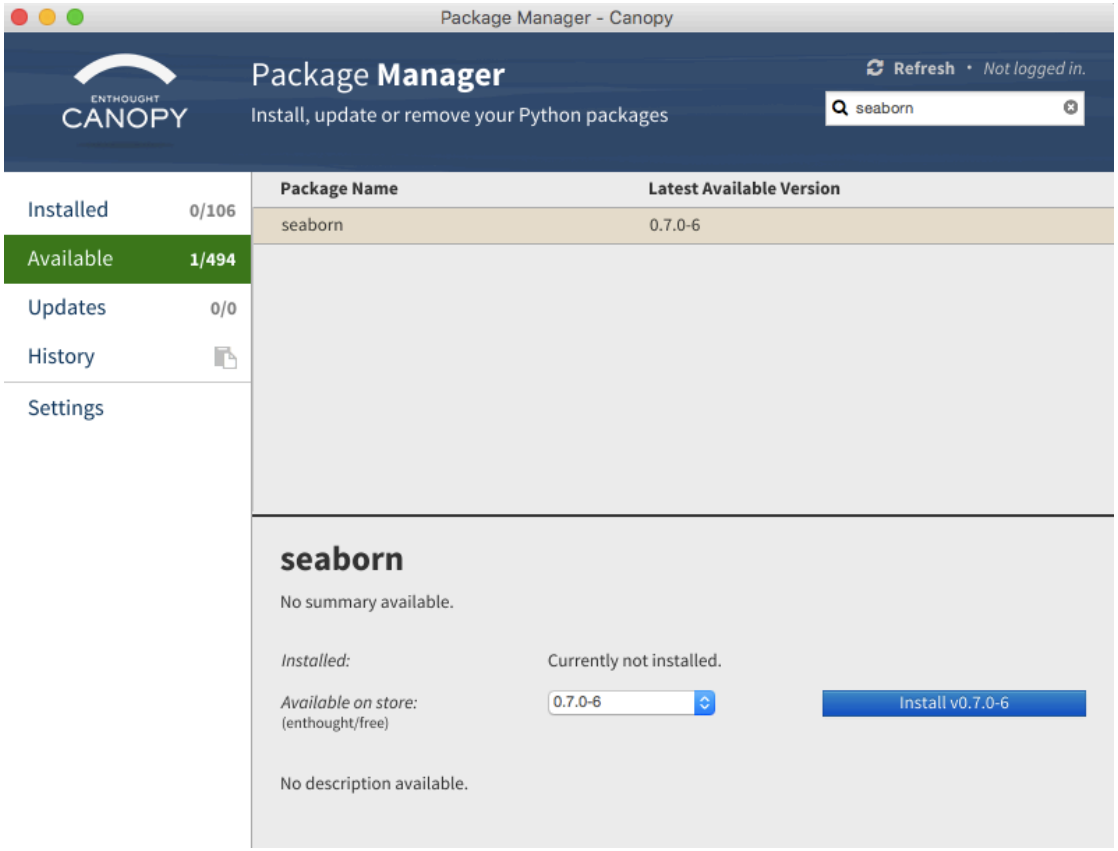
source: https://store.enthought.com/

Python for Data Analysis and Scientific Computing

- Working with alternative environments:
  - Canopy – scientific and analytic Python package distribution. has a free version, but limited

Package Manager:

- Packages are installed through Package Manager



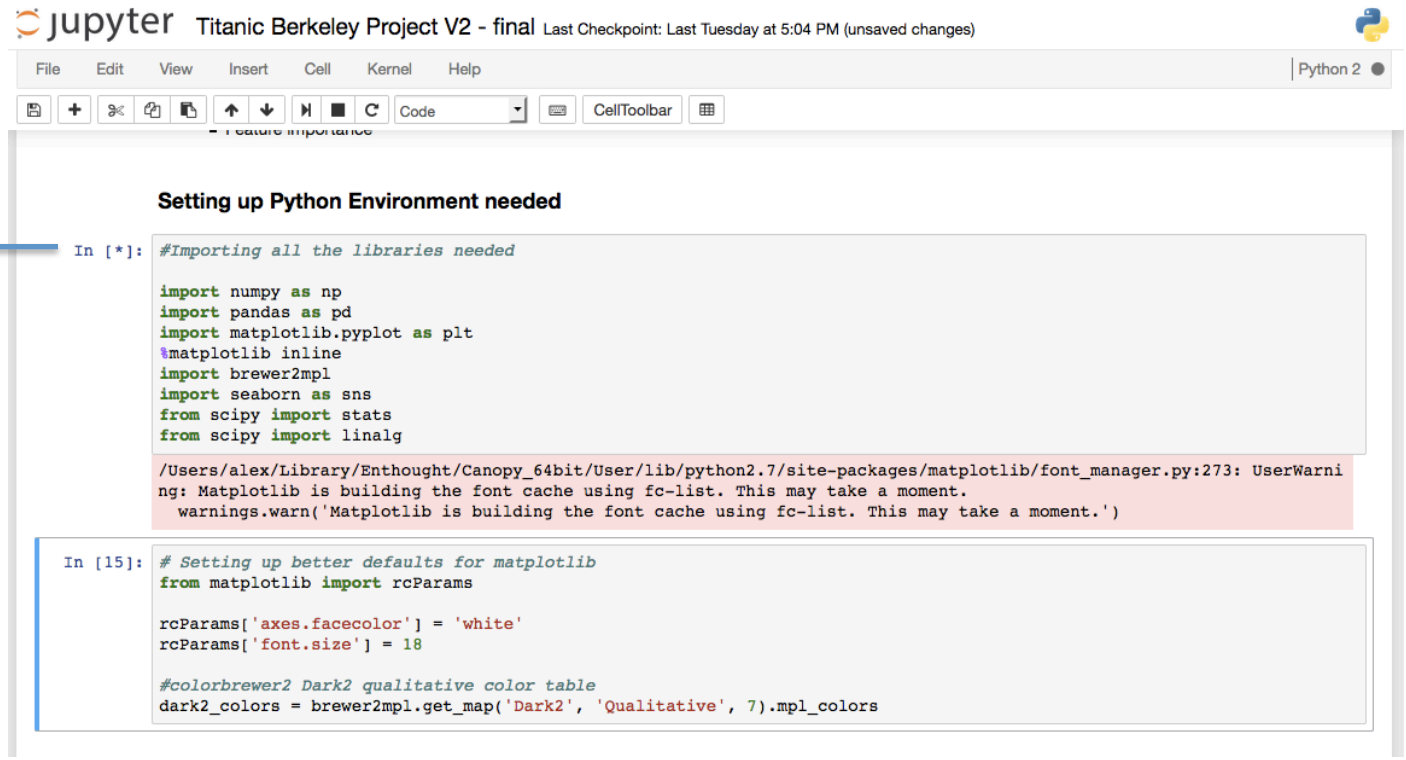
source: https://store.enthought.com/

Python for Data Analysis and Scientific Computing

- Working with alternative environments:
  - Canopy – scientific and analytic Python package distribution. has a free version, but limited

Using the browser

- [*] the line is in a process of executing
- Execute your code sequentially using Shift+Enter



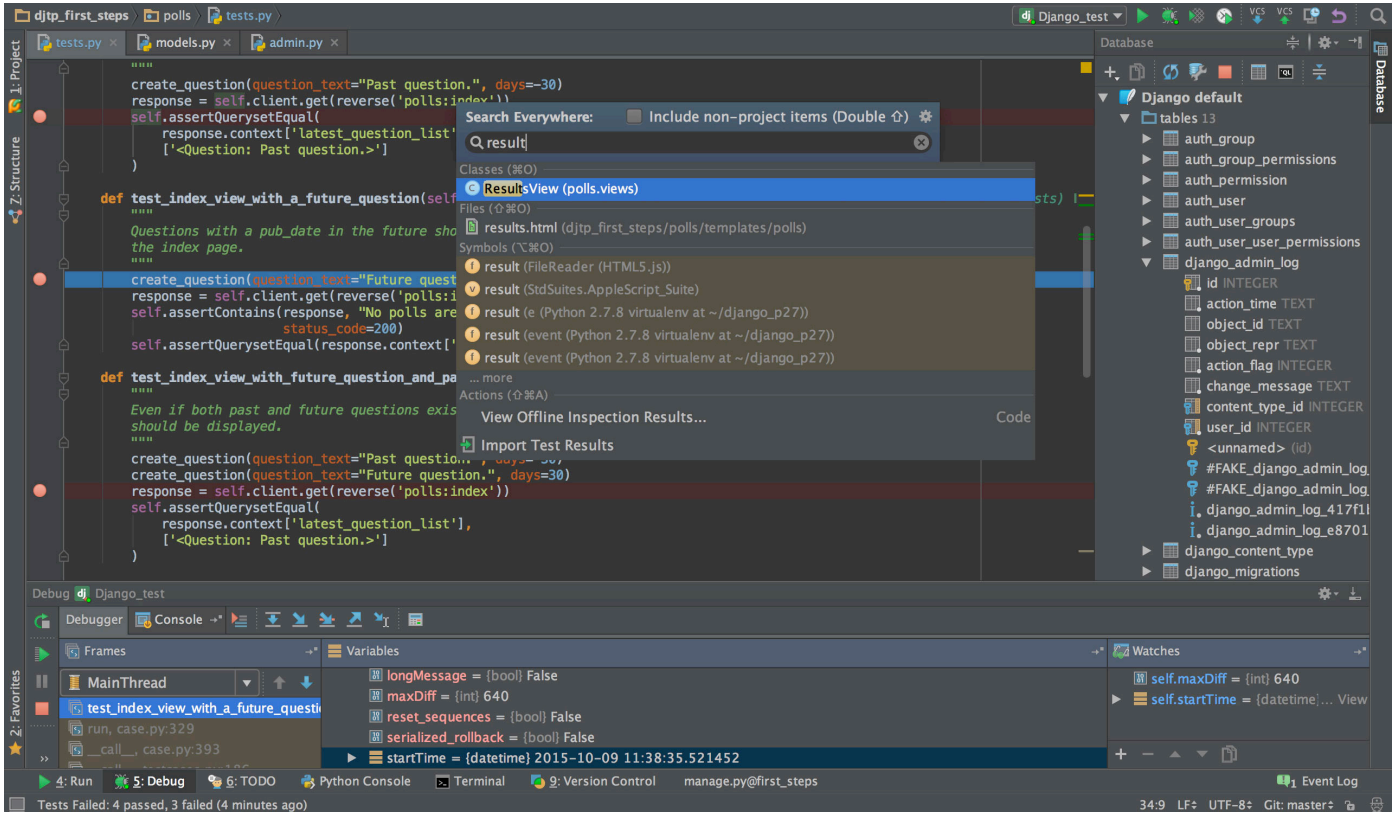
source: https://store.enthought.com/

Introduction to Python

- Working with alternative environments:
  - Pycharm – scientific and analytic Python package distribution. has a free version, but limited

It offers:

- object-oriented prog.
- save time by auto completing the routines
- on-the-fly error checking
- focus on bigger picture
- many features
- many packages



source: https://www.jetbrains.com/pycharm/

Python for Data Analysis and Scientific Computing

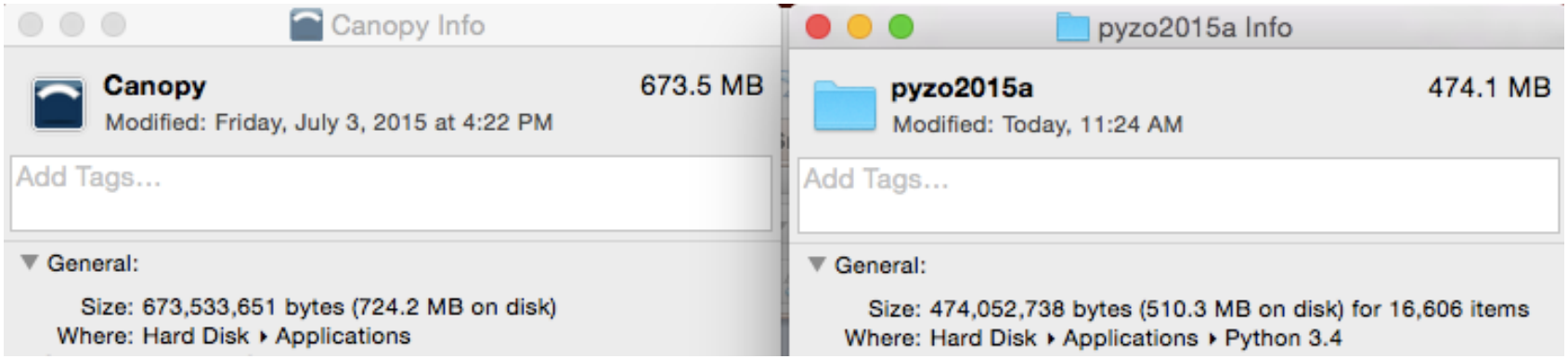
- Working with alternative environments:
  - Pyzo … what?

PYZO: PYTHON TO THE PEOPLE

    - ***Pyzo*** is a free Open-source BSD-licensed
    - It is an easy to use GUI computing environment based on Python 3
    - It is built to make Python's potential for scientific and engineering computing to be easily accessible
    - Pizo is combined with IEP to make a powerful easy to use comprehensive environment for scientific computing and visualization
    - Pyzo is a comprehensive development environment pre-built with:
      - many scientific packages including NumPy, SciPy and Matplotlib
      - uses IEP as the front-end IDE
      - IPython and a notebook
    - Complete packages for Pyzo are given here: http://www.pyzo.org/packages.html

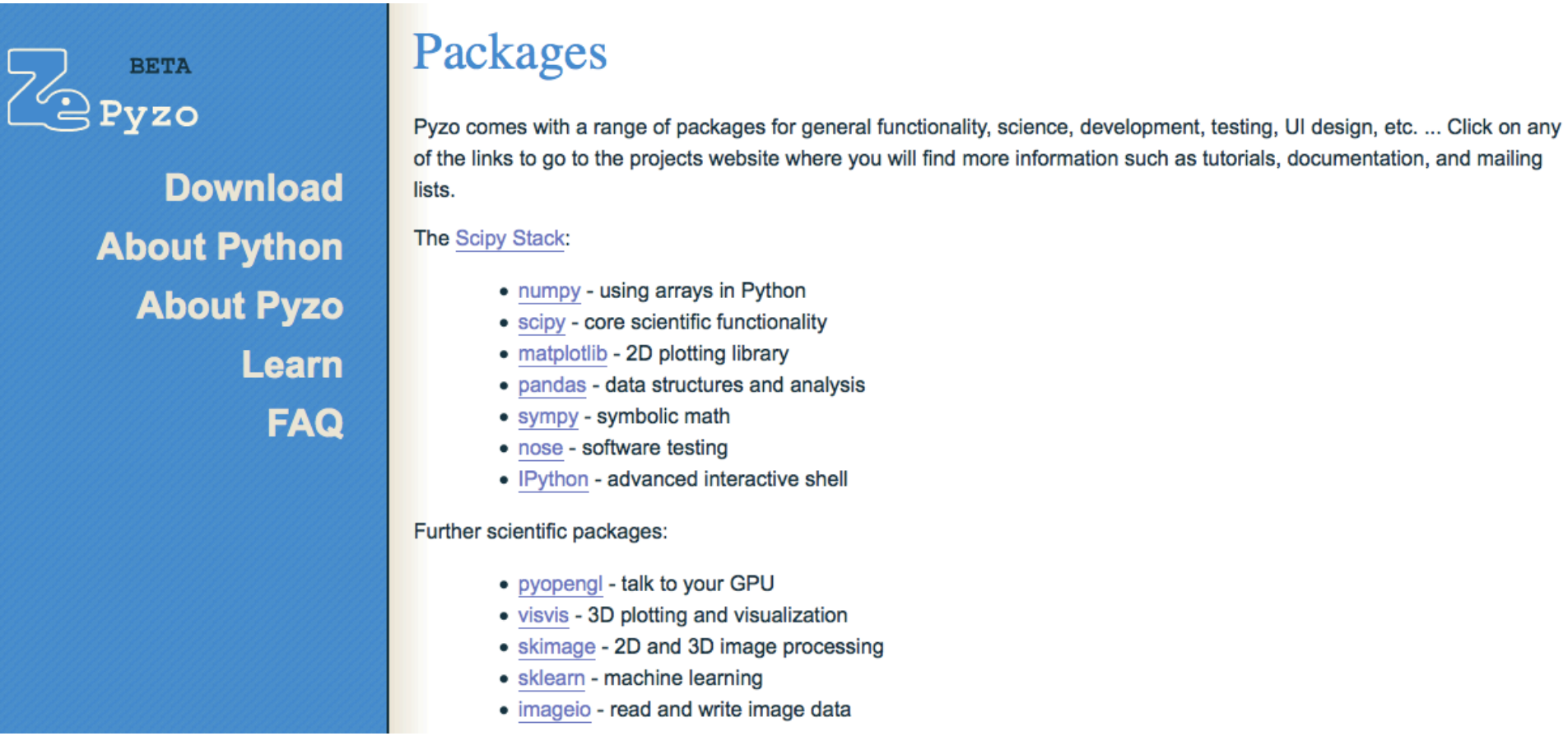
Introduction to Python

- Working with alternative environments:
  - Pyzo takes less space than Canopy (without Canopy debugger installed)



Introduction to Python

- Working with Pyzo



source: http://www.pyzo.org/packages.html#packages

Introduction to Python

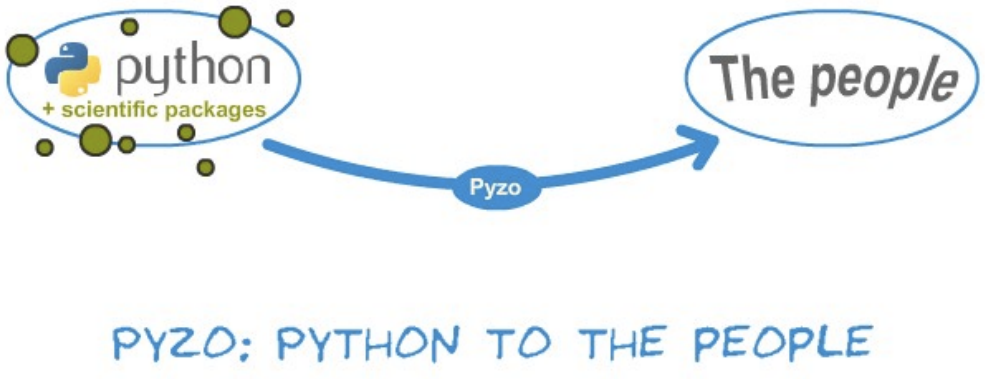
- Working with Pyzo
  - Useful magic commands:

Linux-like:

- pwd
- ls
- mkdir
- rmdir
- whos
- dir()
- cd
- history

Python IEP/Pyzo specific:

- edit *file*
- del(parameter)
- reset
- reset -f
- run *file*
- time
- colors (nocolor,linux,lightbg)
- dhist



Tip: Using a command followed by '?' will give you help for it. Example: 'cd?'

magic comprehensive list: http://ipython.org/ipython-doc/3/interactive/magics.html

Introduction to Python

- How to check your packages? .. (In your interactive editor - I use Pyzo)

```
In [3]: import pip
In [4]: pip
Out[4]: <module 'pip' from '/Applications/Python 3.4/pyzo2015a/lib/python3.4/site-packages/pip/__init__.py'>
In [5]: pip list
Out[5]:
You are using pip version 6.0.8, however version 7.0.1 is available.
…. snip snip snip …
Django 1.8.1
…. snip snip snip …
matplotlib 1.4.2
…. snip snip snip …
numpy 1.9.1
…. snip snip snip …
scipy 0.15.1          ->          one can use: "scipy.__version__" to check the package version (scipy in this case)
```

or

```
In [5]: pip.get_installed_distributions()
Out[5]:
…. snip snip snip …
Django 1.8.1 (/Applications/Python 3.4/pyzo2015a/lib/python3.4/site-packages),
…. snip snip snip …
matplotlib 1.4.2 (/Applications/Python 3.4/pyzo2015a/lib/python3.4/site-packages),
…. snip snip snip …
numpy 1.9.1 (/Applications/Python 3.4/pyzo2015a/lib/python3.4/site-packages),
…. snip snip snip …
scipy 0.15.1 (/Applications/Python 3.4/pyzo2015a/lib/python3.4/site-packages),
```

- Using:

```
In [6]: help()
```

Introduction to Python

- Python modules, packages and scientific blocks
  - A module is a Python file with .py extension containing definitions and statements
  - Modules implement a set of functions and can be *imported* into other modules using the *import* command:

Example:

Lets create our first Python program called – *lecture1.py*:

```
# Let's make a simple addition until certain criteria is not met
def add(a):       # the criteria to be met is a
        x, y = 2, 4
                while x < a:
                print(x,y),
                x = x + y
```

Possible usage:

```
# importing a module called 'lecture1'
import lecture1          -> must be in the same working folder
lecture1.add(24)
```

Tips:

1. *to check what's loaded in memory simply look at the workspace*
2. *if you enter an infinite loop hit 'ctrl-c' to break the loop and stop the execution*
3. *try debugging your program using breakpoints*

Introduction to Python

- Python modules, packages and scientific blocks
  - **'sys'**: includes information about the modules available to us. It includes built-in and imported modules
  - **Imported modules:** usually Python programs are a combination of several modules imported with a main application

    They can be seen in the following way:

```
import sys, textwrap
names = sorted(sys.modules.keys())
name_text = ' || '.join(names)
print(textwrap.fill(name_text))
```

  - **Built-in modules:** the interpreter can be built and compiled with some C modules, that do not appear in the imported modules list since they were included in the package

    They can be seen in the following way:

```
import sys
for module_name in sys.builtin_module_names:
    print(module_name)
```

  - **The search path:** the search path for modules is managed as a list saved in **sys.path.** The default path list consist of the script directory used to start the application and the current working directory

```
sys.path
```

Introduction to Python

- Python modules, packages and scientific packages
  - A *module* in Python is a source file that can expose classes, functions and global variables
  - A *package* in Python is a directory of one or several Python modules (source .py files)

Example:

```
# 1. this is how we import a package:
import math
        Usage:
        math.acos(0.3454453)

# 2. this is how we import a module from a package:
import acos from math … ( or: from math import acos)
        Usage:
        acos(0.3454453)
```

Tips:

1. *Observe what's loaded in workspace in the two cases above*
2. *you can delete a particular item from workspace by using **'del(item)'***
3. *to clear the workspace use **'reset'***
4. *The built-in function **'dir()'** is used to find what's currently defined and shown in workspace*

Introduction to Python

Magic commands:

Magic list of important commands:

magic [-brief,]	automagic
ls	lsmagic
pwd	ed, edit
cd	load
mv	run
dir	time
mkdir	writefile
rm, rmdir	env
who, whos	system
alias, unalias	config
hist, history	save
dhist	logoff logon logstart logstate logstop

Introduction to Python

- Saving individual lines:

save [option] [log_name [line-region]] – saving lines of your code that you want to reload next time around

Python

```
In [1]: whos
Variable   Type    Data/Info
---------------------------
name       str     __loader__

In [2]: a=[3,5,8]

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

In [4]: len(a)
Out[4]: 3

In [5]: whos
Variable   Type    Data/Info
---------------------------
a          list    n=3
name       str     __loader__

In [6]: b=20

In [7]: history -n
   1: whos
   2: a=[3,5,8]
   3: type(a)
   4: len(a)
   5: whos
   6: b=20
   7: history -n

In [8]: save session 2 6
The following commands were written to file `session.py`:
a=[3,5,8]
b=20
```

lecture1.py | session.py

```
1  # coding: utf-8
2  a=[3,5,8]
3  b=20
4
```

Python

```
In [9]: ed session.py
Editing... done. Executing edited code...

In [10]: c=345

In [11]: save -a session 10
The following commands were written to file `session.py`:
c=345
```

lecture1.py | session.py

```
1  # coding: utf-8
2  a=[3,5,8]
3  b=20
4  c=345
5
```


hint: type 'magic' for details

Introduction to Python

- Saving your session:

logstart [-o|-r|-t] [log_name [log_mode]] - Start logging anywhere in a session for the first time with a specific name. defaults is 'ipython_log.py'. It saves to file 'name' in 'backup' mode. It saves the history up to that point and then continues logging. logstart takes a second optional parameter 'logging mode'. This can be one:

of (note that the modes are given unquoted):

append - Keep logging at the end of any existing file

backup - Rename any existing file to name~ and start name

global - Append to a single logfile in your home directory

over - Overwrite any existing log

rotate - Create rotating logs: name.1~, name.2~, etc.

logoff - Temporarily stop logging. You must have previously started logging

logon - Restart logging after temporarily stopped with %logoff

logstate - Print the status of the logging system

logstop - Fully stop logging and close log file

Introduction to Python

- Python modules, packages and scientific packages

  - **Numpy** : provides powerful numerical arrays objects, and routines to manipulate them
    http://www.numpy.org/

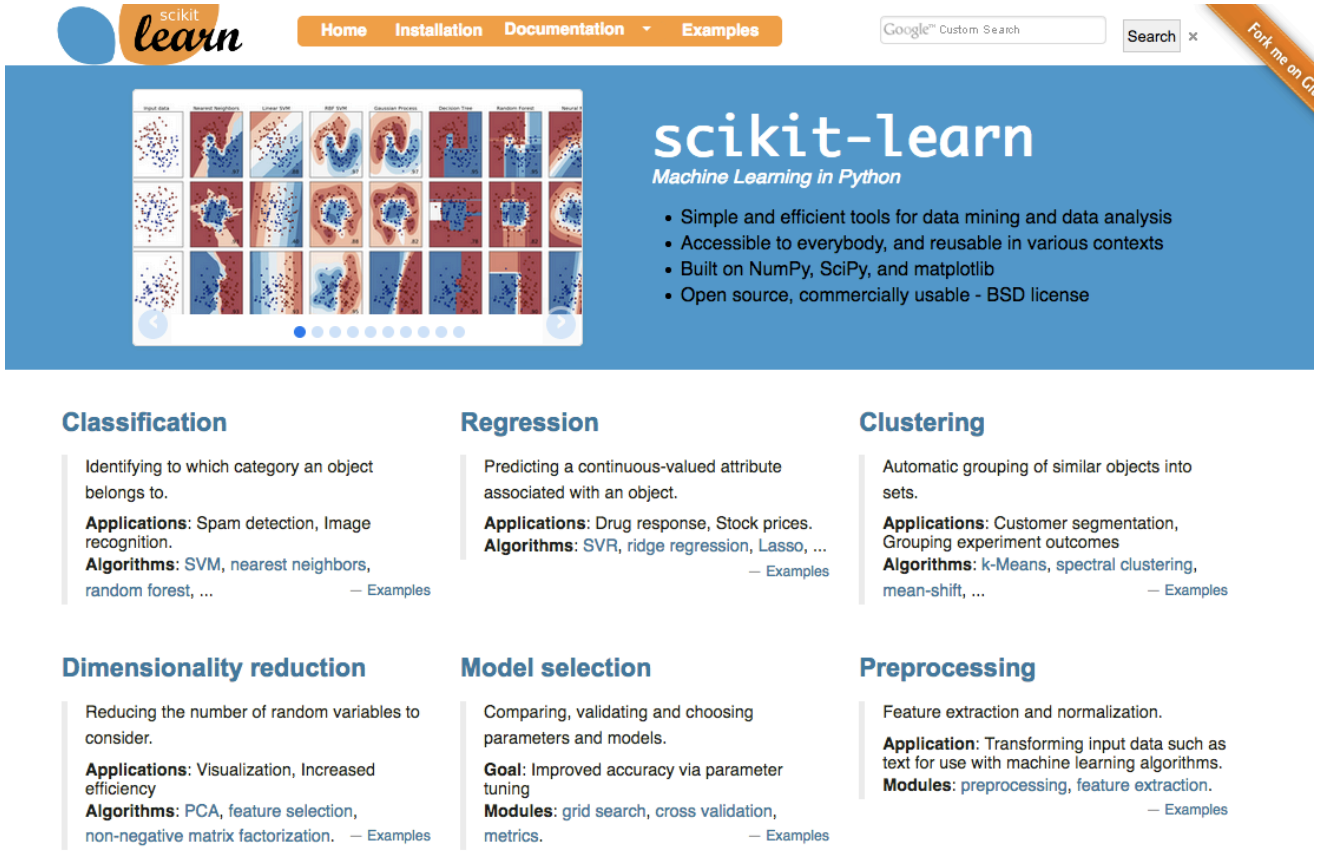
  - **Scipy** : high-level data processing routines. Optimization, regression, interpolation, etc
    http://www.scipy.org/

  - **Matplotlib** : 2-D visualization, "publication-ready" plots
    http://matplotlib.sourceforge.net/

  - **Mayavi** : 3-D visualization
    http://code.enthought.com/projects/mayavi/

Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

NumPy :

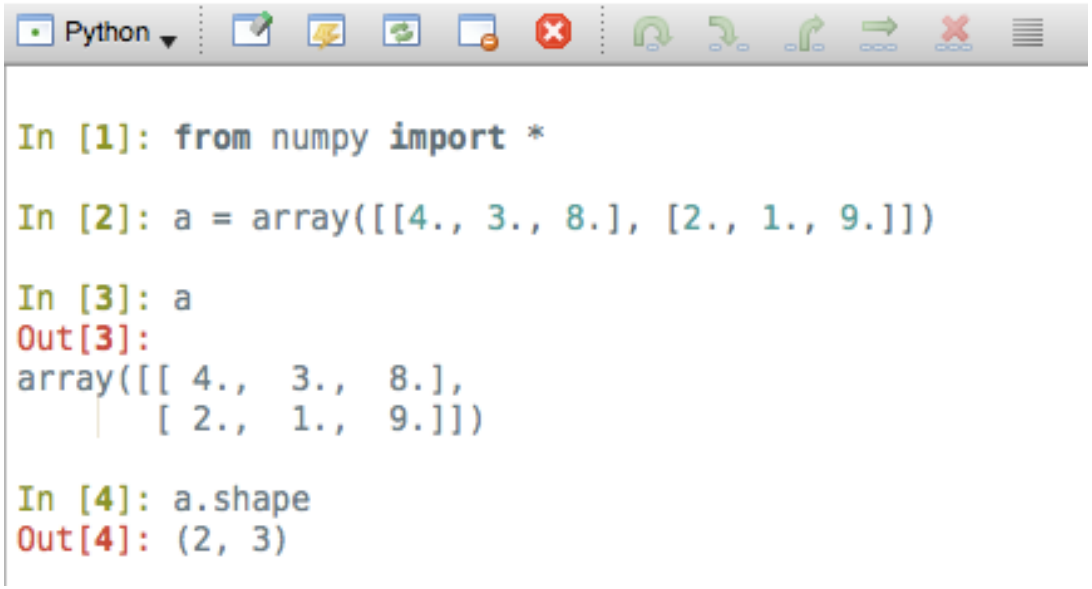
- This is the fundamental package for scientific computing with Python
- Some of its most attractive key points are:

  - easy integration with C/C++ and Fortran code
  - Fourier transform, linear algebra, and random number capabilities
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - NumPy can be used as an efficient multi-dimensional container of generic data, where arbitrary data-types can be defined. This allows NumPy to easily integrate with a wide variety of databases.

Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

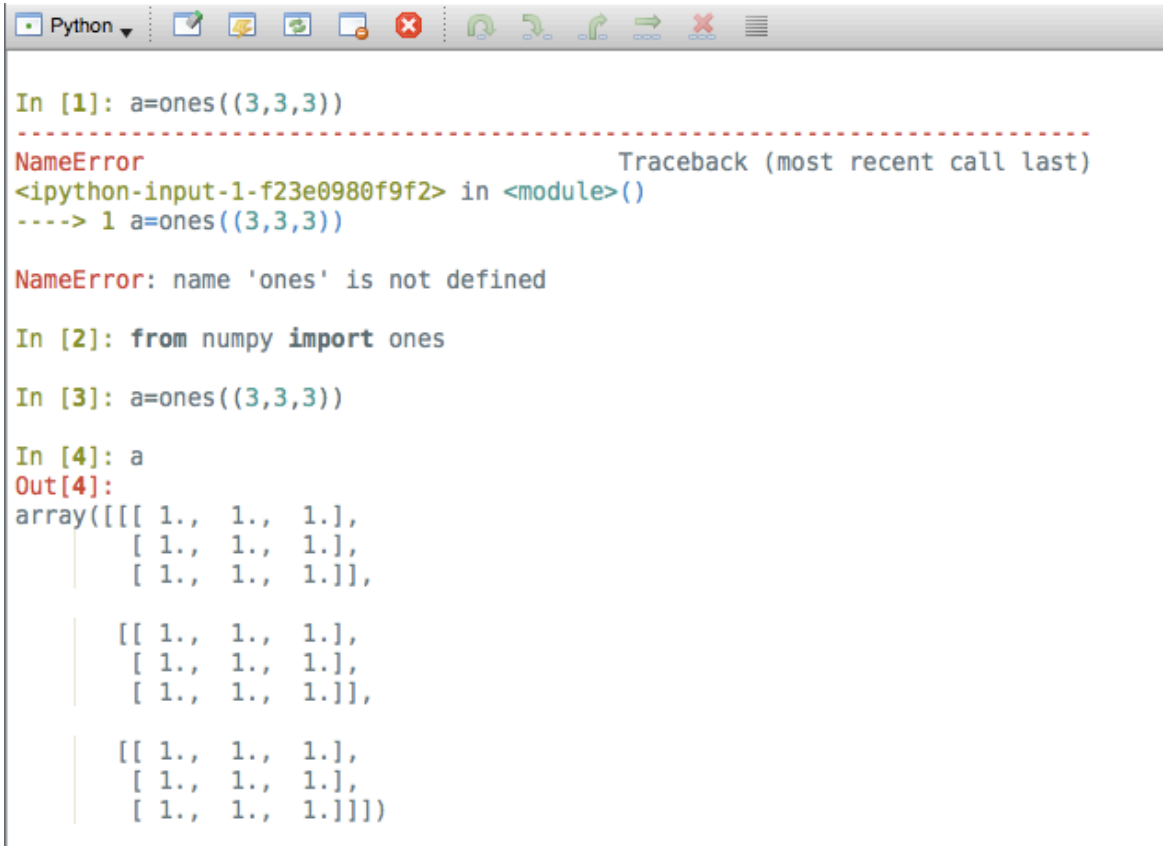
NumPy : *simple example 1 – using the array function*



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

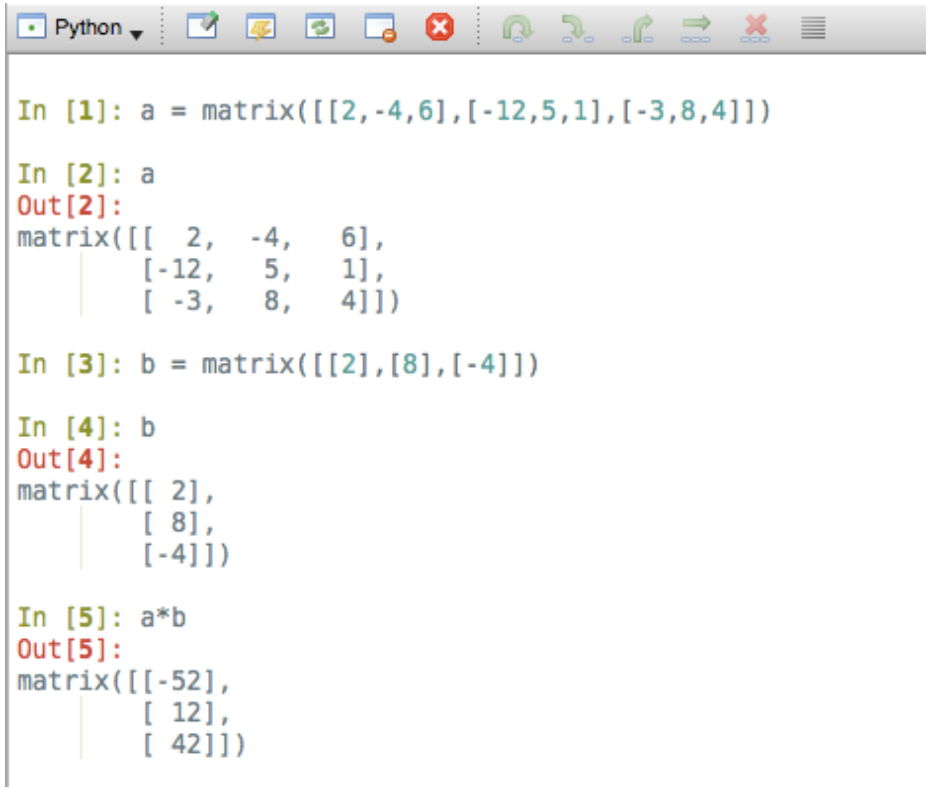
NumPy : *simple example 2 – using the zeros / ones functions*



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

NumPy : *simple example 3 – two-dimensional array – a matrix*



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

SciPy :

- This package is open-source for mathematics, science, and engineering
- Some of its most attractive key points are:
  - SciPy depends on the NumPy library and that provides convenient and fast N-dimensional array manipulation
  - SciPy is built to work with NumPy arrays, and provides many numerical routines like routines for numerical integration and optimization

SciPy and NumPy :

- run on all popular operating systems
- are quick to install and are free of charge
- are easy to use and so powerful that some of the world's leading scientists and engineers depend on it
- easy to use when manipulating numbers, display and publish the results

Python for Data Analysis and Scientific Computing

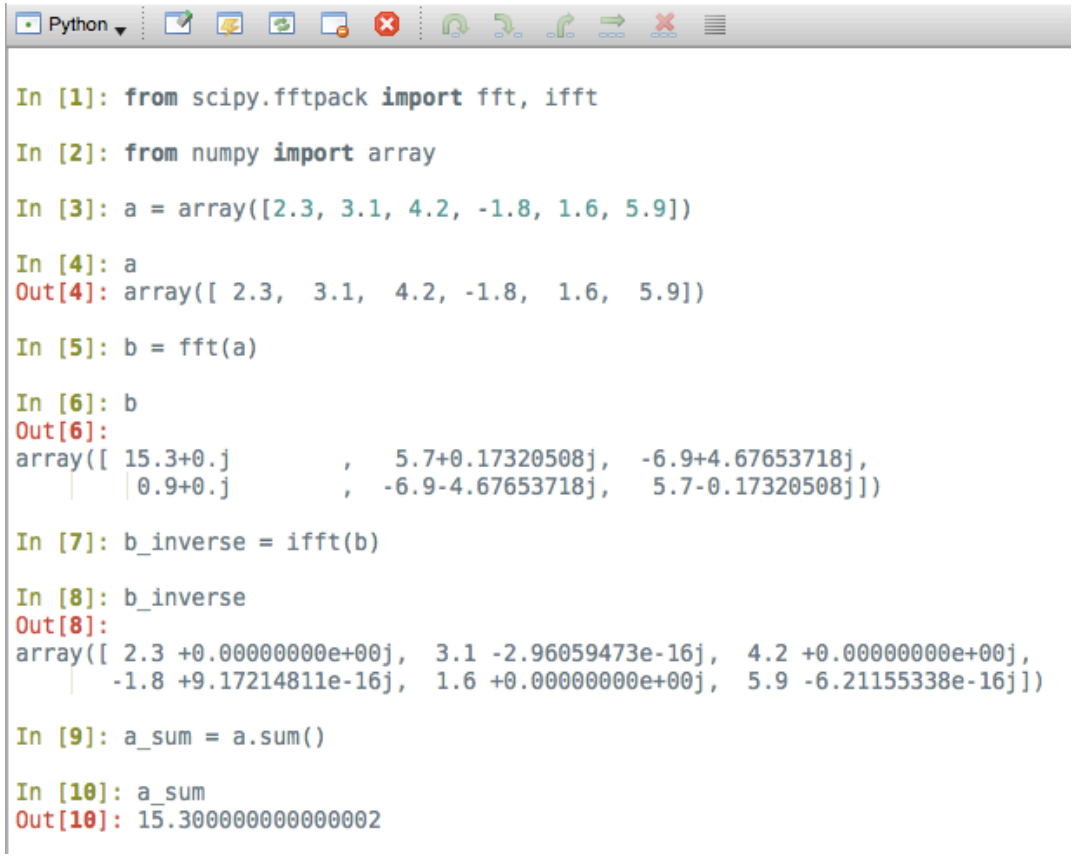
- Python modules, packages and scientific packages

SciPy : *simple example 1 – working with polynomials*

Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

SciPy : *simple example 2 – using the FFT and IFFT*



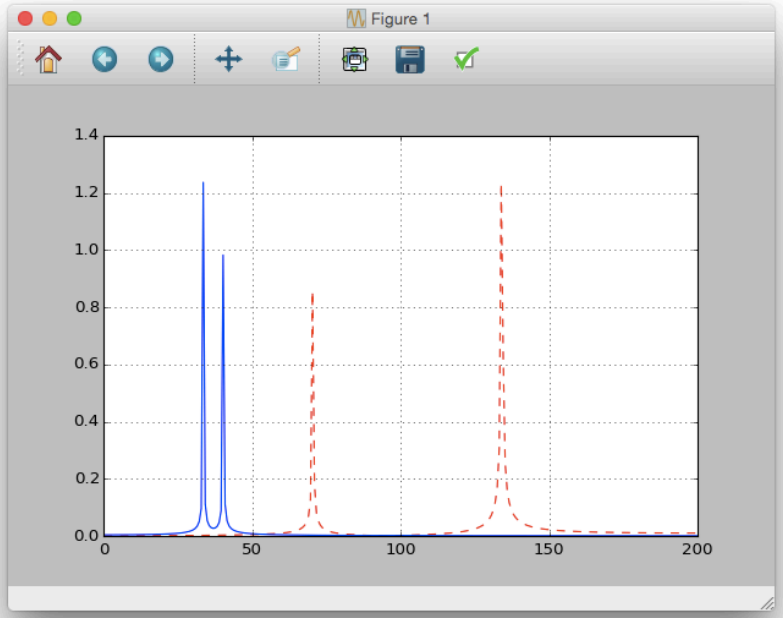
Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

SciPy : *simple example 3 – plot the FFT of the sum of two series*

code:

```
from scipy.fftpack import fft
import numpy as np
import matplotlib.pyplot as plt
F = 600   # Sample frequency:
T = 1.0 / F   # Period T:
a = np.linspace(0.0, F*T, F)
b = np.sin(100.0 * 4.0*np.pi*a) + 0.6*np.sin(70.0 * 3.0*np.pi*a)
c = np.cos(50.0 * 2.0*np.pi*a) + 0.8*np.sin(60.0 * 2.0*np.pi*a)
bf = fft(b); cf = fft(c)
af = np.linspace(0.0, 1.0/(3.0*T), F/2)
plt.plot(af, 3.0/F * np.abs(bf[0:F/2]),'r--')
plt.plot(af, 2.5/F * np.abs(cf[0:F/2]),'b-')
plt.grid(); plt.pause(1) # plt.show()
```



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

Matplotlib :

- The development of Matplotlib was initiated by John Hunter (1968-2012)
- It works great with NumPy and SciPy providing the platform for visualizing results
- This package is built for Python and provides a powerful set of 2D plotting functions
- It can be used in:
  - python scripts
  - the python and iPython shell
  - web application servers,
  - additional graphical user interface toolkits
- The user can easily generate plots, histograms, bar charts, scatterplots, and much more
- Users have the ability to manipulate the axis dimensions and labeling, title, legend, grid, fitting, zooming, etc.
- Working with Matplotlib resembles working with Matlab plotting
- Matplotlib is an open source package

Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

Matplotlib :

- There are several toolkits for Matplotlib:
  - General Toolkits
    - mplot3d
    - AxesGrid
    - MplDataCursor
    - GTK Tools
    - Excel Tools
    - Natgrid
  - Mapping Toolkits
    - Basemap
    - Cartopy
  - High-Level Plotting
    - seaborn
    - ggplot
    - prettyplotlib

(source: http://matplotlib.org/mpl_toolkits/index.html)

Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

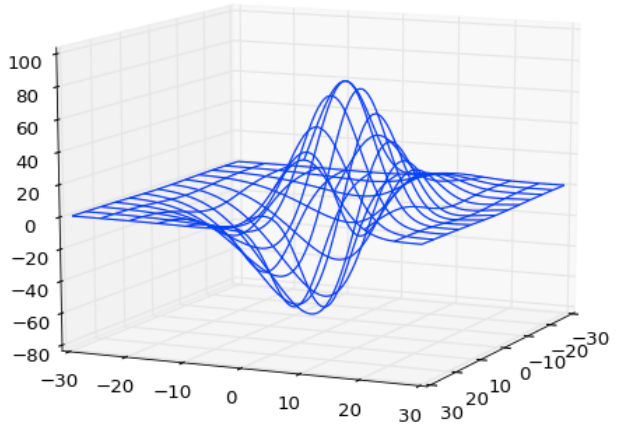
Matplotlib : *simple example 1 – using mplot3d and a wireframe*

– There are several toolkits for Matplotlib:
  - General Toolkits
    – mplot3d example:

code:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.pause(1) # plt.show()
```



(source: http://matplotlib.org/mpl_toolkits/index.html)

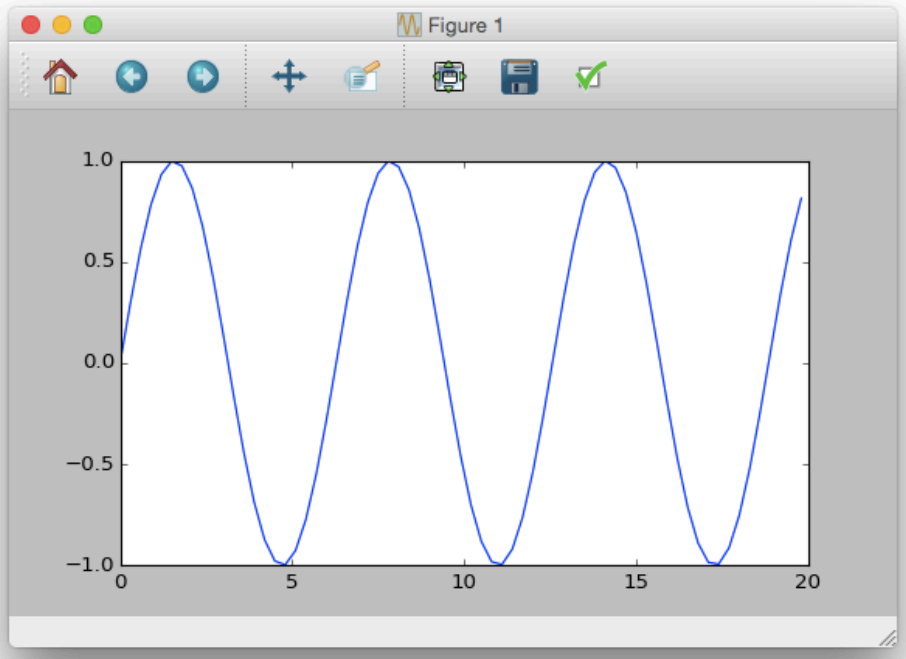
Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

Matplotlib : *simple example 2 – plotting a simple sinusoid*

code:

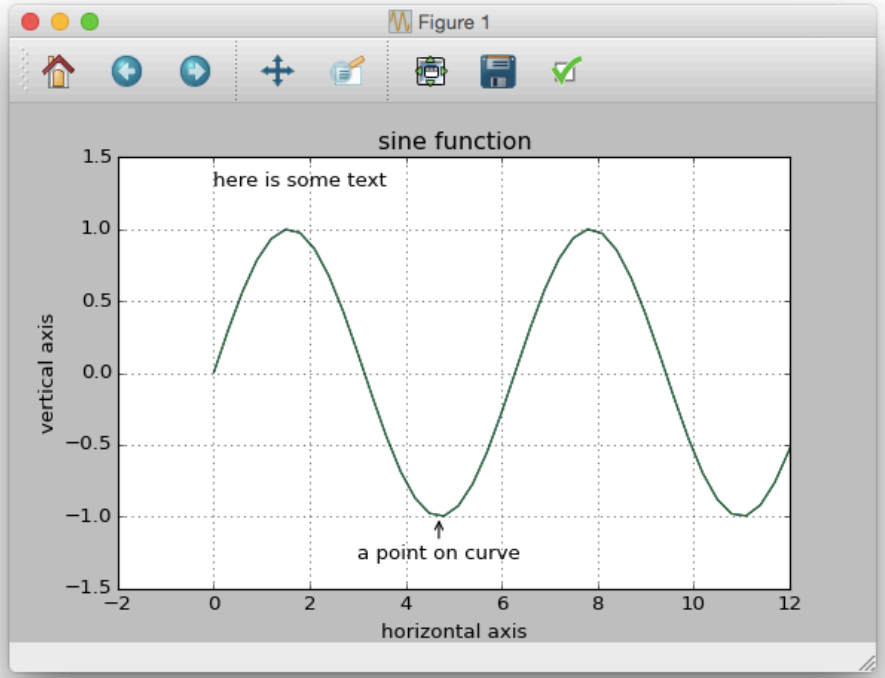
```
from numpy import *
import matplotlib.pyplot as plt
x = arange(0.,20.,0.3)
y = sin(x)
ll = plt.plot(x,y)
plt.pause(1) # plt.show()
```



Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

Matplotlib : *simple example 3 – plotting a simple sinusoid with annotation*



code:

```
ll = plt.plot(x,y)
xl = plt.xlabel('horizontal axis')
yl = plt.ylabel('vertical axis')
ttl = plt.title('sine function')
ax = plt.axis([-2, 12, -1.5, 1.5])
grd = plt.grid(True)
txt = plt.text(0,1.3,'here is some text')
ann = plt.annotate('a point on curve',xy=(4.7,-1),xytext=(3,-1.3), arrowprops=dict(arrowstyle='->'))
plt.pause(1) # plt.show()
```

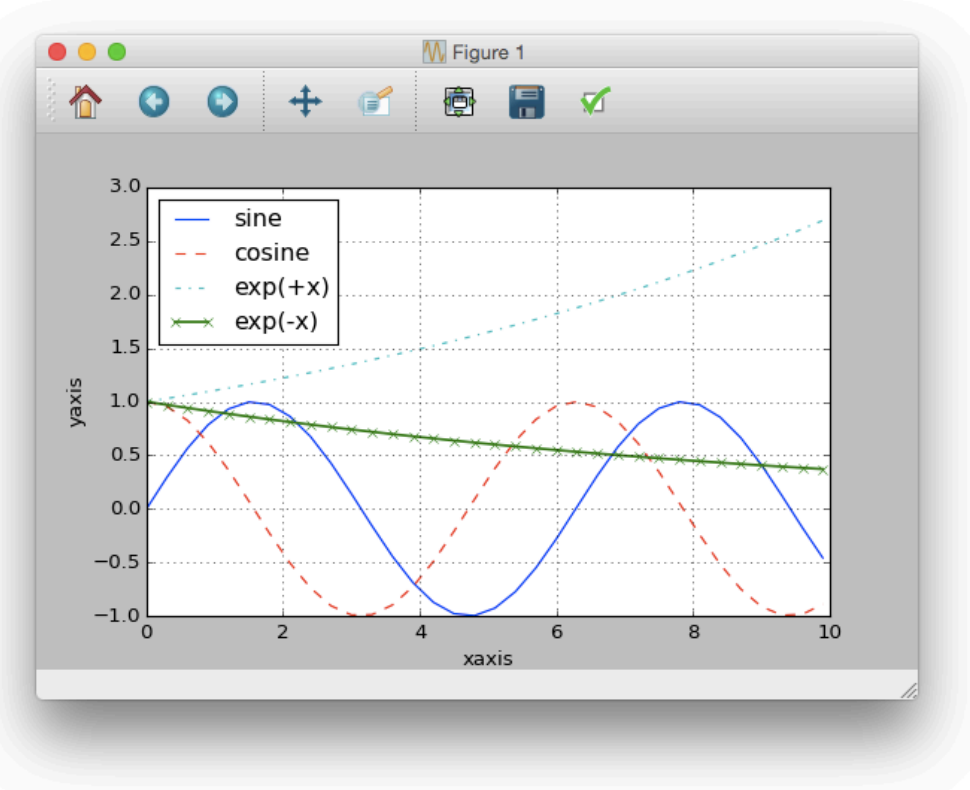
Python for Data Analysis and Scientific Computing

- Python modules, packages and scientific packages

Matplotlib : *simple example 4 – plotting the sin and cos functions with annotation*

code:

```
x = arange(0.,10,0.3)
a = sin(x); b = cos(x);
c = exp(x/10); d = exp(-x/10)
plt.plot(x,a,'b-',label='sine')
plt.plot(x,b,'r--',label='cosine')
plt.plot(x,c,'c-.',label='exp(+x)')
plt.plot(x,d,'gx-', linewidth = 1.5,label='exp(-x)')
plt.legend(loc='upper left')
plt.grid()
plt.xlabel('xaxis')
plt.ylabel('yaxis')
plt.pause(1) # plt.show()
```



Introduction to Python

HW assignment 1:

Make sure you have chosen, installed and configured your Python-NumPy-SciPy environment