Setting up Python

EE599 Deep Learning

Kuan-Wen (James) Huang

Spring 2020

Contributors: Kuan-Wen Huang, Keith Chugg, Sourya Dey

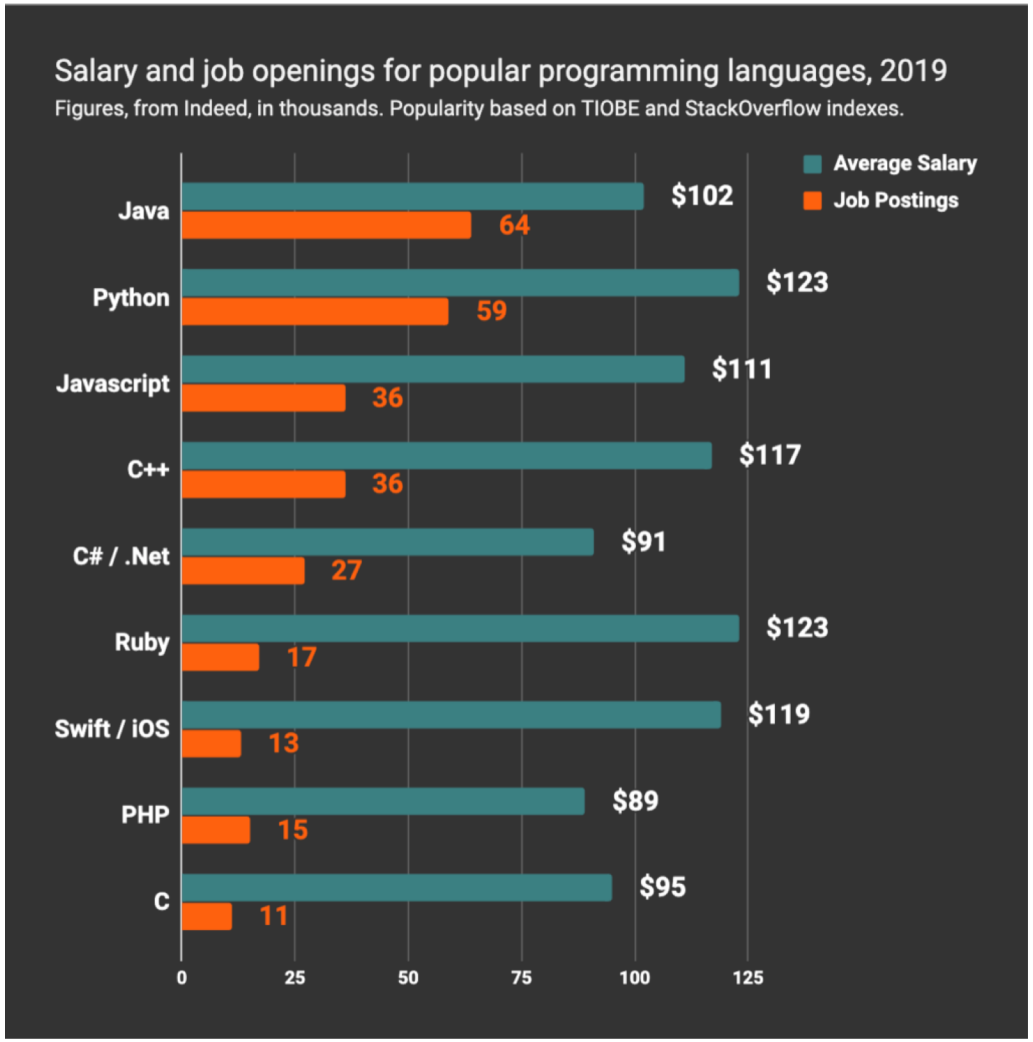
About TAs

- Kuan-Wen (James) Huang
  - Office hours: Thursday 1-3pm at EEB522
  - Email: kuanwenh@usc.edu
- Olaoluwa (Oliver) Adigun
  - Office hours: Tuesday, and Thursday 5-6:30pm at EEB420
  - Email: adigun@usc.edu
- Jiali Duan
  - Office hours: Friday 4-5pm at EEB B10
  - Email: : jialidua@usc.edu
- Arnab Sanyal
  - Office hours: Monday 8-10am at PHE320
  - Email: arnabsan@usc.edu

Why Python but not Matlab?

- Python does nearly everything Matlab does, but it is open source and free.
  - Interpreted language (script) like Matlab
  - Vectorized operations
  - Plotting
  - Data handling, web-accessing
  - Extensive libraries for numerical computation and ML/DL
    - numpy, scipy, scikit-learn, NLTK, Keras, Tensorflow, Pytorch, etc.
  - Full-featured language
    - Classes/Objects, Inheritance, Modules

Why Python but not Matlab?



- Source: https://www.codeplatoon.org/the-best-paying-and-most-in-demand-programming-languages-in-2019/

10 Best Programming Language to Learn in 2020

With time old programming languages become obsolete while new programming languages are launched, but they never gain traction. A common question amongst beginners (and coders alike) is the programming language they should invest learning in, that is in demand, stable outlook, and plenty of jobs.

Here, is a list of top 10 languages that you should learn -

1) Python

Created: Python language developed by Guido van Rossum. It was first released in 1991.

Pros:

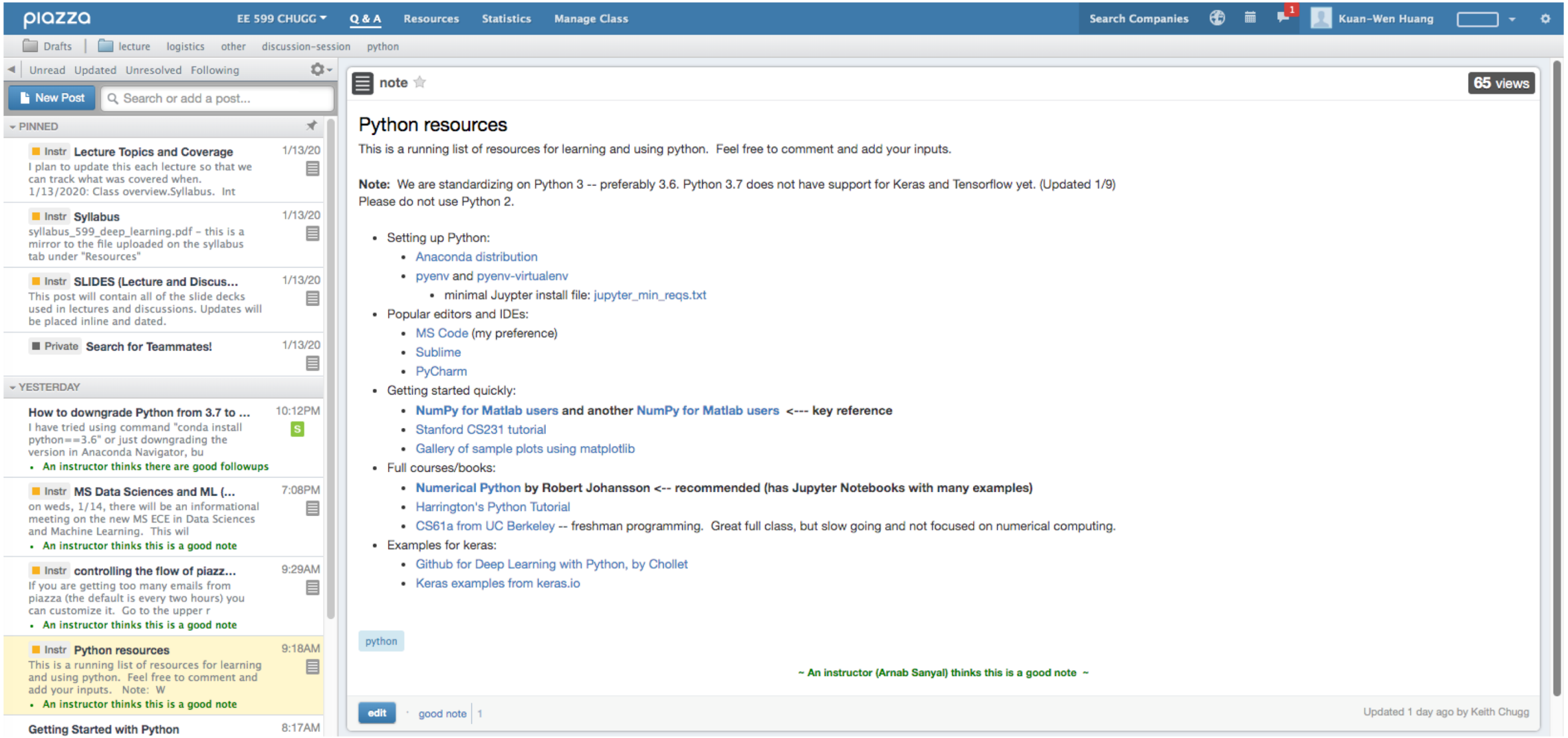
- Supports multiple systems and platforms
- Object-Oriented Programming (OOPs) driven.
- Helps to improve Programmer's Productivity
- Allows you to scale even the most complex applications with ease
- Extensive Support Libraries

Cons:

- Note ideal for Mobile Computing
- Python's database access layer is bit underdeveloped and primitive.

- Source: https://www.guru99.com/best-programming-language.html

Getting Started with Python



Setting up Python Environment

EE599 will use Python 3.6

Anaconda

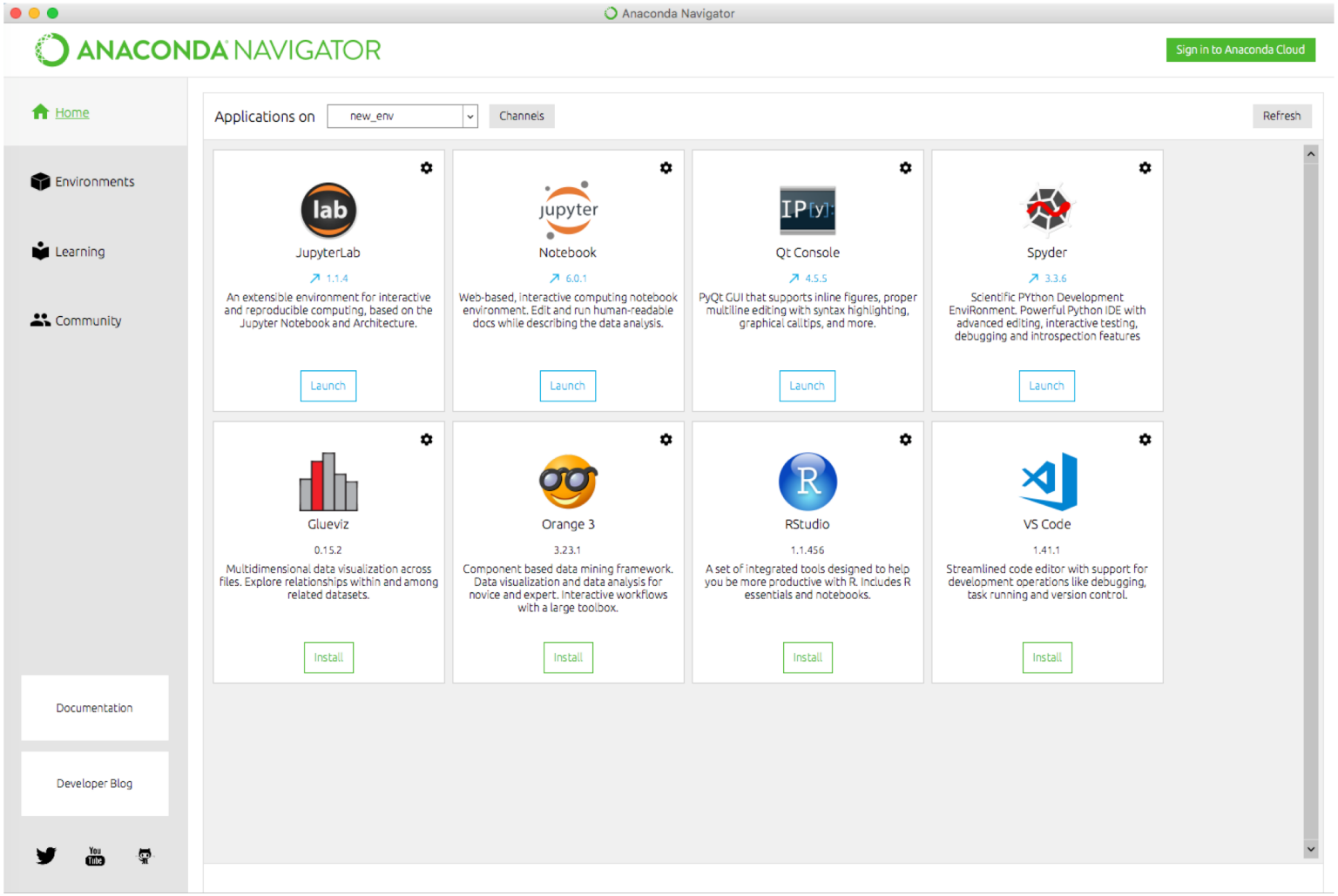
- Install just about everything you ever need (usually more than that..)
- Comes with some IDEs and Jupyter tools:
  - Spyder
  - Jupyter notebook
  - Qt Console

pyenv

- Install the minimum you need for the projects
- Best professional option when working with others
  - Manages library versions w/o ambiguity

you may have a system Python, best practice is not to modify that and instead do a separate install using one of the above

Python Workflows: Anaconda



Positives

Installs almost everything you will ever need

conda package manager and update system

has some useful packages — starts with navigator

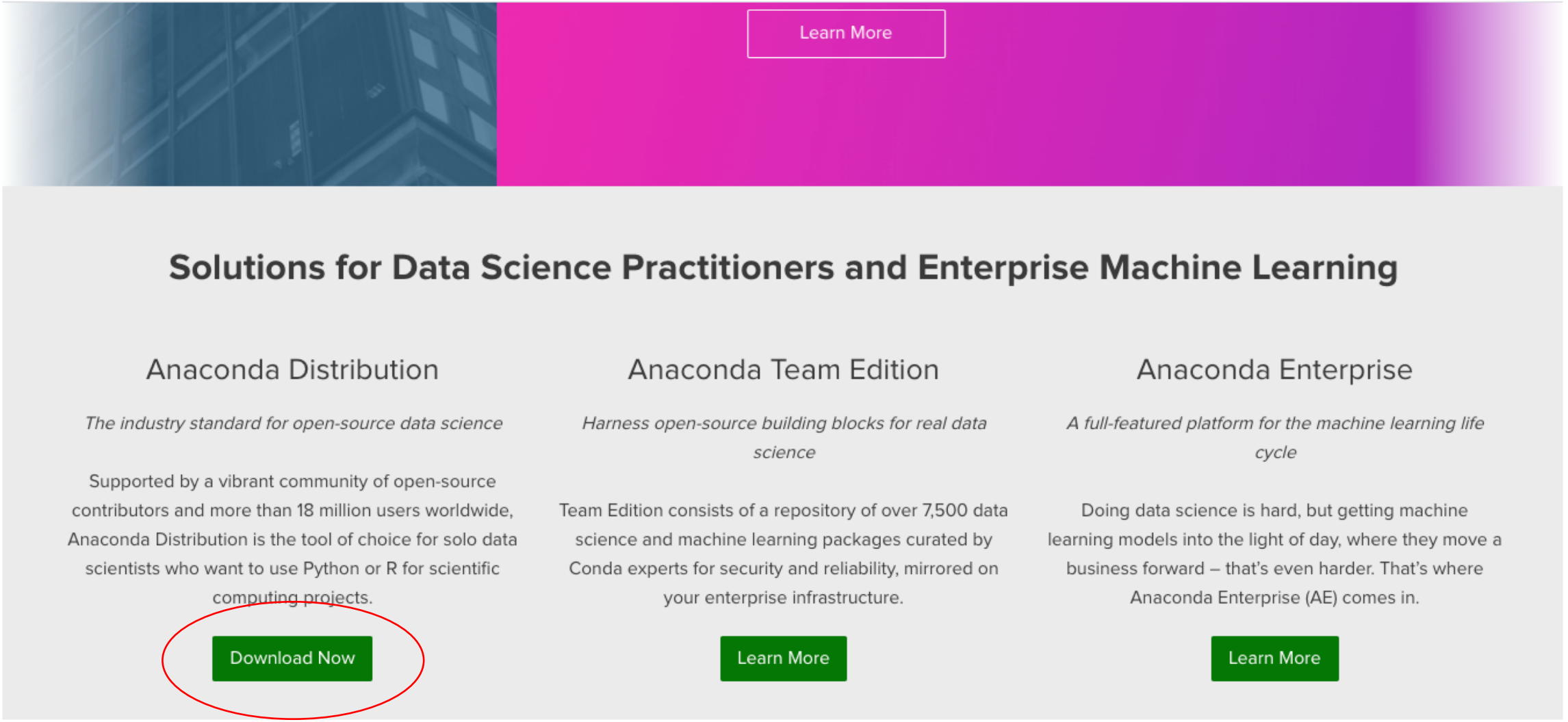
Negatives

takes about 8 GB

Installs a lot stuff you may never use

Not as simple to control package/library versions when working in teams

Python Workflows: Anaconda



Python Workflows: Anaconda

After installing Anacoda, open terminal

```
conda create -n new_env python=3.6 anaconda
```

wait… type y and wait…

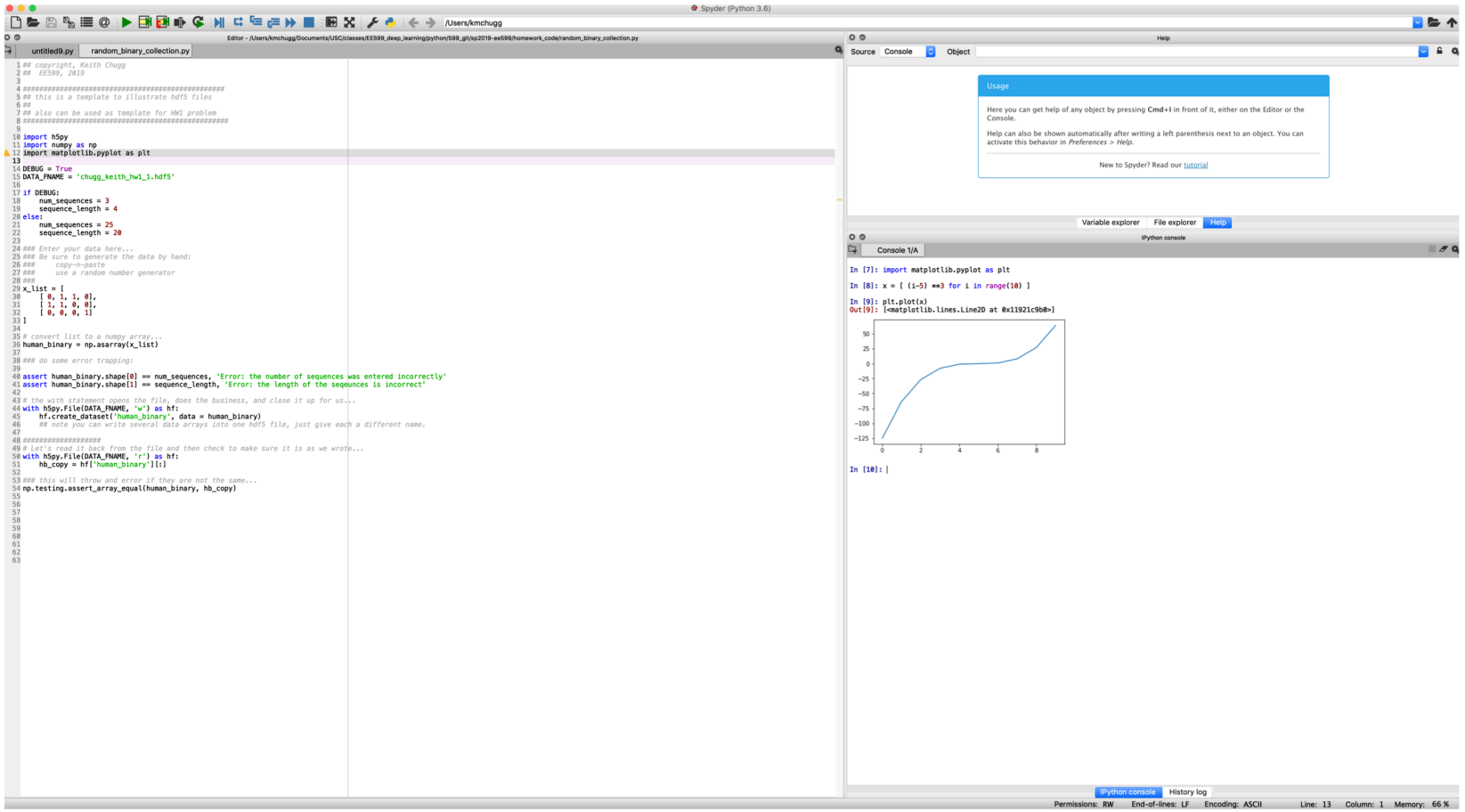
```
conda activate new_env
```

```
python --version
```
should say 3.6.9

You can now work inside virtual environment “new_env”

```
conda deactivate
```
to quit virtual environment

Python Workflows: Anaconda-Spyder



Spyder *IDE included with Anaconda:*

\+ *integrated with Jupiter qtConsole, includes debugger, simple, closest to Matlab*

\- *editor is slow for larger files, similar flow can be replicated with editor of choice*

External Editors/Debuggers

- Language sensitive text editors
  - Sublime text, MS Code, BBEdit, etc.
- Some of these have additional features
  - Debugger for Python – e.g. breakpoints, variable inspection, etc.
  - Git integration, extension system, latex, support for other languages, etc.
- All of these can be used with Anaconda and/or pyenv
  - qtConsole, CLI, jupyter notebook

Other Set-ups

- Jupyter Notebook
  - Web-based interactive development
  - Good for teaching or showing results to others
- Jupyter qtConsole
  - “Smart terminal” where you can plot
  - Good command history and completion
- CLI
  - Useful for “headless” operation – e.g. AWS, google cloud
  - Plots have to be saved and viewed elsewhere

all of these can be used with **Anaconda** install or **pyenv**

Python Workflows: Editor + qtConsole

random_binary_collection.py — UNREGISTERED

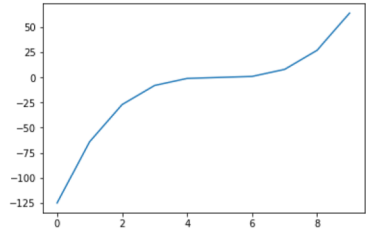
```python
## copyright, Keith Chugg
##  EE599, 2019

################################################
## this is a template to illustrate hdf5 files
##
## also can be used as template for HW1 problem
################################################

import h5py
import numpy as np
import matplotlib.pyplot as plt

DEBUG = True
DATA_FNAME = 'chugg_keith_hw1_1.hdf5'

if DEBUG:
    num_sequences = 3
    sequence_length = 4
else:
    num_sequences = 25
    sequence_length = 20

### Enter your data here...
### Be sure to generate the data by hand:
###     copy-n-paste
###     use a random number generator
###
x_list = [
    [ 0, 1, 1, 0],
    [ 1, 1, 0, 0],
    [ 0, 0, 0, 1]
]

# convert list to a numpy array...
human_binary = np.asarray(x_list)

### do some error trapping:

assert human_binary.shape[0] == num_sequences, 'Error: the number of sequences was entered incorrectly'
assert human_binary.shape[1] == sequence_length, 'Error: the length of the seqeunces is incorrect'

# the with statement opens the file, does the business, and close it up for us...
with h5py.File(DATA_FNAME, 'w') as hf:
    hf.create_dataset('human_binary', data = human_binary)
    ## note you can write several data arrays into one hdf5 file, just give each a different name.

###################
# Let's read it back from the file and then check to make sure it is as we wrote...
with h5py.File(DATA_FNAME, 'r') as hf:
    hb_copy = hf['human_binary'][:]

### this will throw and error if they are not the same...
np.testing.assert_array_equal(human_binary, hb_copy)
```

On master* in sp2019-ee599, Line 1, Column 1 — Spaces: 4 — Python

not tied together, simple text editor + qtConsole

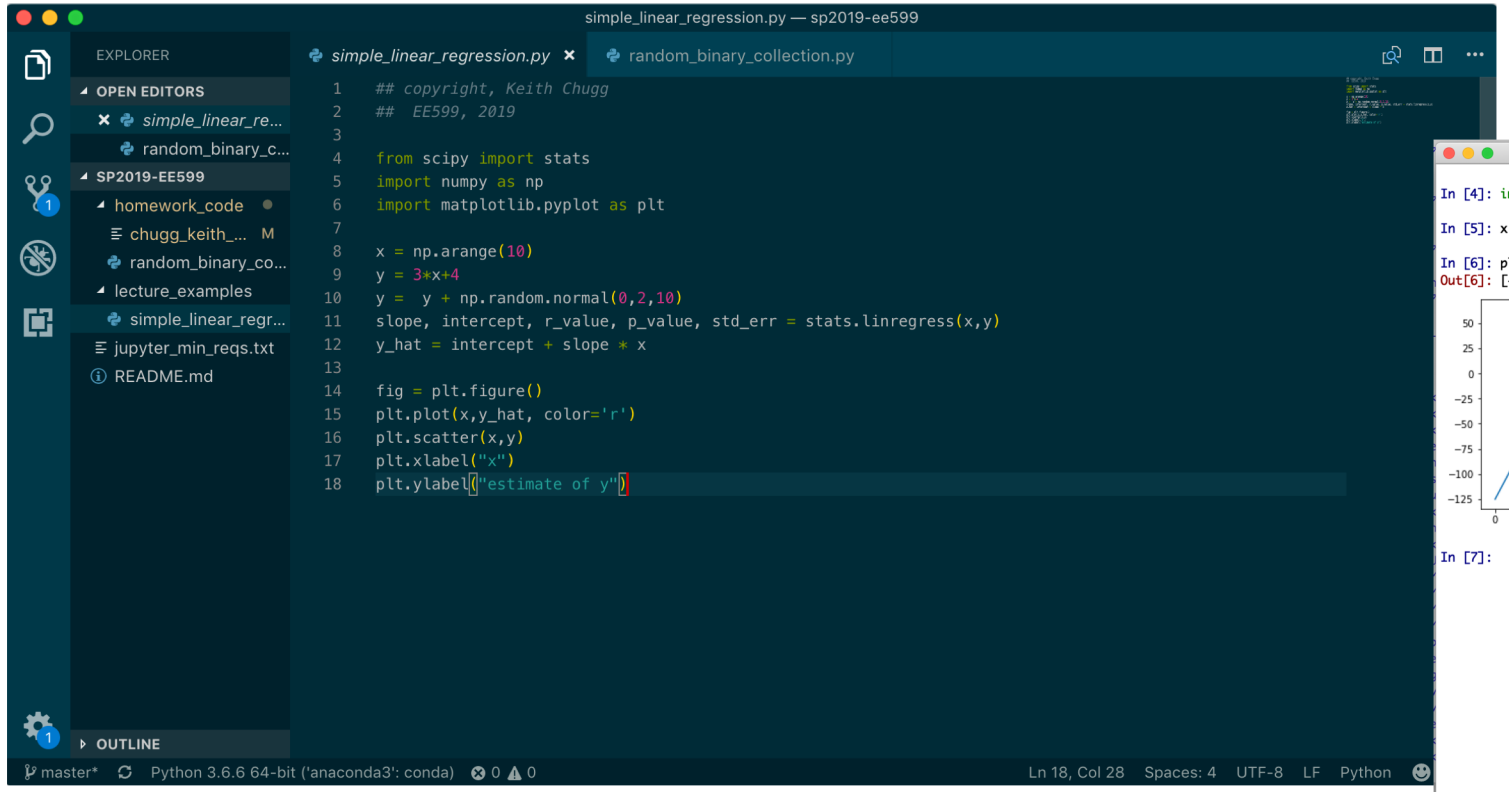
Jupyter QtConsole

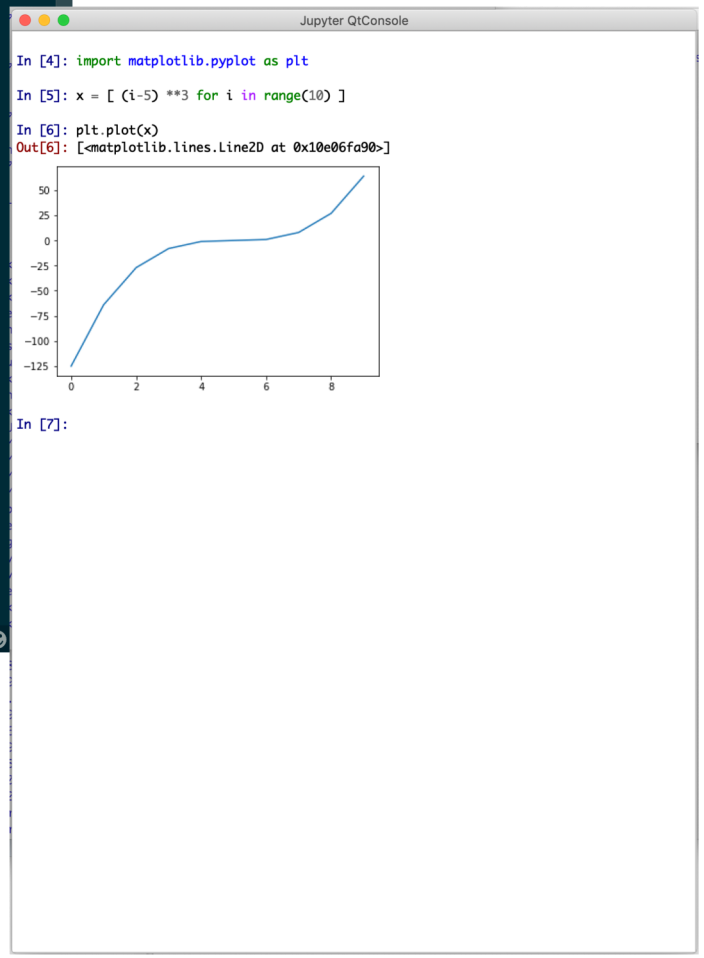
```
In [4]: import matplotlib.pyplot as plt

In [5]: x = [ (i-5) **3 for i in range(10) ]

In [6]: plt.plot(x)
Out[6]: [<matplotlib.lines.Line2D at 0x10e06fa90>]
```




```
In [7]:
```

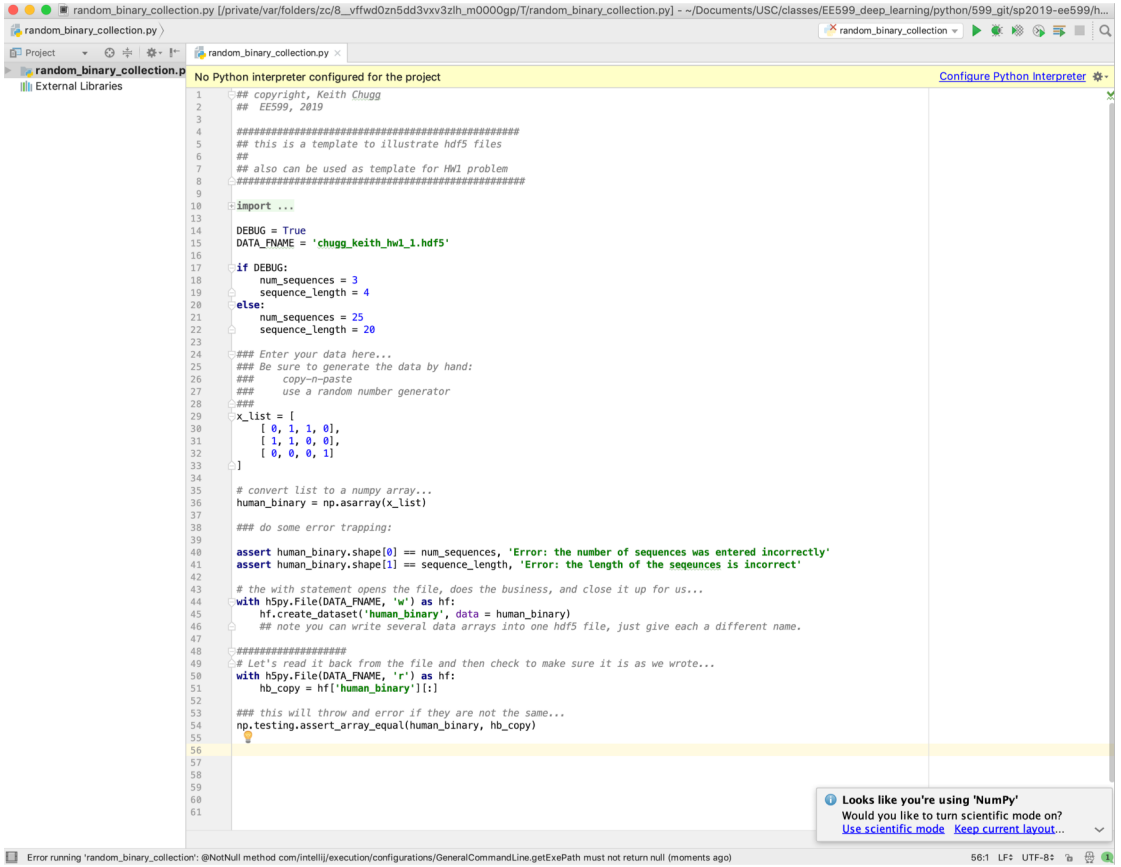
Python Workflows: MSCode + qtConsole





MS Code can be used a text editor in this workflow and/or you can run and debug inside MS Code — i.e., it is an IDE

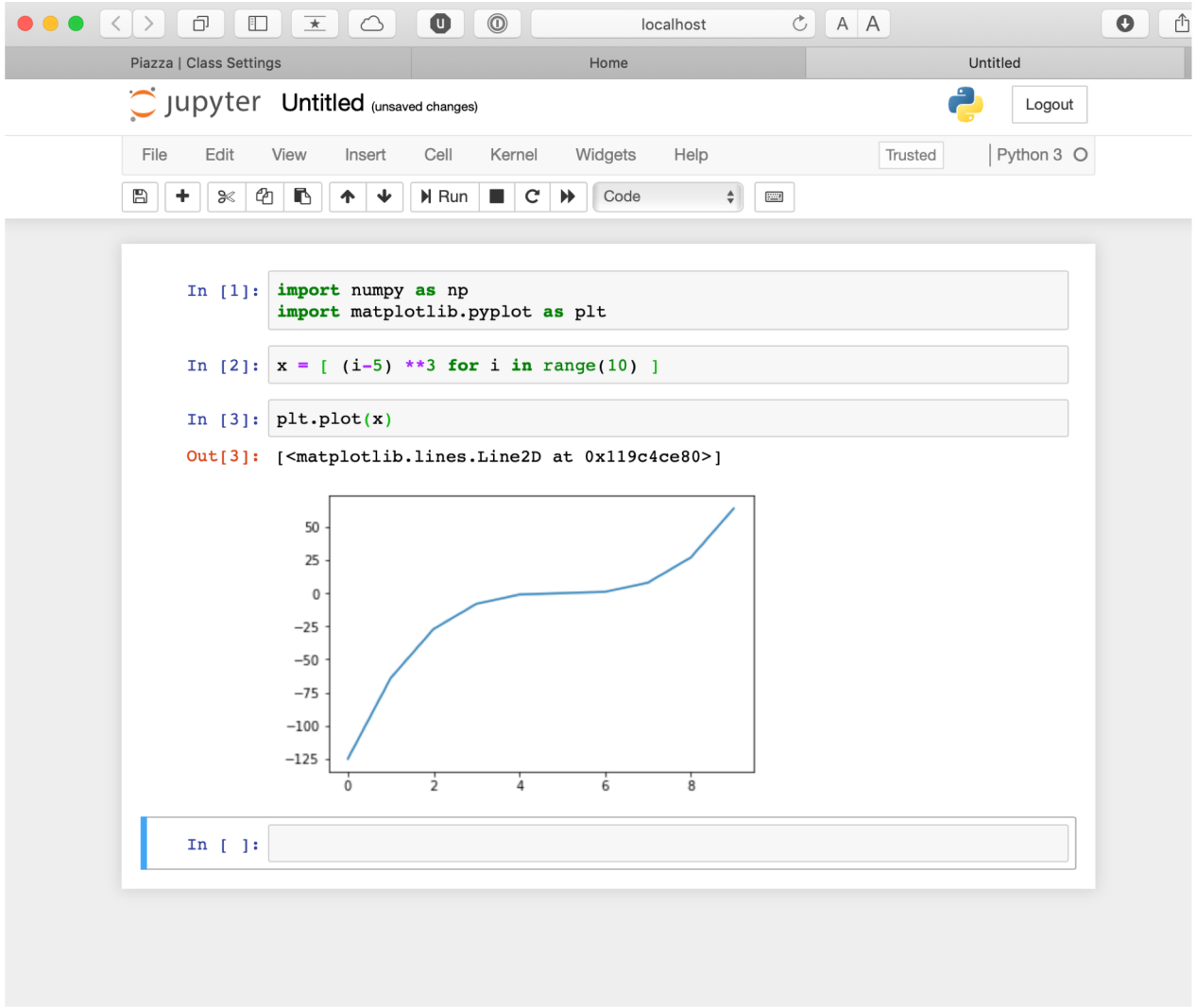
Python Workflows: PyCharm



+ *more polished than Spyder, more widely used. Supports pyenv and Anaconda*

- *Slows down when using Anaconda install because it indexes libraries*

Python Workflows: Jupyter Notebooks



Python Workflows: pyenv Virtual Environment

- To install pyenv, check its websites

1. https://github.com/pyenv/pyenv
2. https://github.com/pyenv/pyenv-virtualenv

The second link has a good description of how to:

- *install various Python versions (e.g., 2.7.x, 3.6.9, 3.7.1)*
- *create, manage and delete virtual-environments on top of these base Python installs*

pyenv Virtual Environment

```
pyenv install -l
pyenv install 3.6.9
pyenv virtualenv 3.6.9 new_env2
pyenv activate new_env2
pip install -r requirements.txt
pip install numpy
pip freeze
pip freeze > new_requirements.txt
```

list available python base python versions

install python 3.6.9 as a base version

create a new virtualenv new_env2

activate new_env2— this is your current python environment

install libraries from a specified list

add other libraries as needed

list the libraries installed in new_env2

save the new set-up to replicate elsewhere

```
pyenv virtualenvs
pyenv uninstall new_env2
```

list all your virtaulenvs

deletes the virtual environment

Issue with many open source solutions is minor incompatibilities by version

This approach allows you to set up and replicate on multiple machines the same python set-up (including all versions)

Final Word

Be proactive and try to figure things out