**Python Cheatsheet**

**(Using Anaconda and Jupyter)**

**Jupyter Notebook**

Launching: Start Menu -> cmd -> cd C:\Users\BrendanAlbertHaas\Documents\PythonScripts -> jupyter notebook

Run Code: Shift + Enter

Close kernels: Ctrl + c (in terminal window)

Can also simulate jupyter notebook in terminal with ipython notebook. Simply type ipython into terminal to open it. The exit command will close it.

**Install New Packages (using anaconda)**

Update current framework: Start Menu -> cmd -> conda update anaconda

Update numpy: pip install --upgrade numpy

Update pip: python –m pip install --upgrade pip

**Text Editors**

Currently using Sublime Text (last updated 10/19/17)

Version Control

Github website, and git application

**Python Packages (libraries)**

NumPy (import numpy as np)

Basic array data type, simple processing operations

np.random, np.linalg commonly used

SciPy (from scipy.stats import norm, from scipy.integrate import quad)

Built on top of NumPy. Includes standard routines used in linear algebra, integration, interpolation, optimization, distributions/random number generation, signal processing

Matplotlib (http://matplotlib.org/index.html)

Most popular/comprehensive python library for creating figures/graphs. Plots histograms, contour images, bar charts, 3D. Outputs in many forms (PDF/PNG/EPS, etc.). Has LaTeX integration.

Plotly/Bokeh/VPython

Other graphics libraries (less used). VPython good for 3D graphics/animations.

SymPy (from sympy import Symbol)

Allows manipulation of sympolic expressions (like Mathematica)

Ex: x, y = Symbol(‘x’), Symbol(‘y’)

x+x+x+y now gives the output 3\*x + y

Ex 2: from sympy import solve

solve(x\*\*2 + x + 2) now gives the output sqrt(7)\*I/2, -1/2 + sqrt(y)\*I/2]

Pandas (import pandas as pd)

Popular library for working with data. Fast, flexible, well designed.

NetworkX

Standard graph algorithms for analyzing network structure, plotting routines, etc.

QuantEcon (QuantEcon.py)

Used for econ lectures, install with pip install quantecon

Notes: import module vs from module import foo (trying to use item foo from a library module)

import module

Pro: Less maintenance of import statements (don’t need more import statements to use another item other than foo from the library)

Con: Have to type module.foo repeatedly (longer, somewhat redundant. Can be minimized with import module as mo, so you can type mo.foo instead.

From module import foo

Pro: Less typing to use foo, more control over which module items can be used

Con: Lose context about foo: math.ceil() is more clear than ceil().

Notes: Contents of a package

1. Files with Python code (‘modules’)

2. Possibly some compiled code that can be accessed by python (e.g. functions compiled from C or FORTRAN code)

3. a file called \_\_init\_\_.py that specifies what will be executed when we type import package\_name

Notes: General package locations with anaconda are C:\Anaconda3\pkgs

**General Commands**

pwd

Shows the present working directory

dir

Lists files/folders in the present working directory (‘ls’ only works for unix)

run (ex: run test.py)

Looks in present directory, attempts to run test.py

cat (ex: cat test.py)

Looks in present directory, prints contents of test.py

%load (ex: %load test.py)

Loads test.py into current window for easy access

**Python**

Bools = True, False (not TRUE, FALSE), True = 1, False = 0

Lists: a = [2,4,6,8], a[1:] returns [4,6,8] (inclusive)

a[1:3] returns [4,6] (not inclusive)

a[-2:] returns [6,8] (last 2 elements of list)

Open a file: f = open(‘newfile.txt, ‘w’) (w signifies opening for writing)

f.write(‘Testing\n’)

f.write(‘Testing again’)

f.close()

Loops without indices

For a list x\_values = [1,2,3]

for x in x\_values: print(x\*)

NOT: for I in range(len(x\_values)):

Lambda functions: Define functions in one line

def f(x):

return x\*\*3

OR

f = lambda x: x\*\*3

Function keyword arguments

def f(x, coefficients=(1,1)) #function defaults to a, b = 1,1

a,b = coefficients

return a + b \* x

Arrays through numpy:

Careful modifying ‘copies’ of the array – also modifies the original; a = array, b = a, b[0]=0 also changes the first element of a to 0

Avoid this with np.copyto(b,a) [actually copies the array]

%whos:

Lists all initialized variables thus far present in the namespace

%precision 4:

Sets printed precision for floats to 4 decimal places

%debug:

Opens the debugger

**Pandas:**

Series: A column of data (ex: s=pd.Series(np.random.randn(4), name = ‘returns’)

Built on top of NumPy arrays, have similar operations

Further options: s.describe(), s.index = [‘a’, ‘b’, ‘c’, d’]

pd.read\_csv: Reads a comma separated variable document, creating a dataframe

**GitHub:** Acc: BrendanAHaas

git init: Initialize your git repository for the folder you’re in

git add . : Add all files in current folder to the git repository staging area

git rm --cached index.html: Remove index.html file from staging area

git status: Return current git status

git commit: Opens vim editor for commiting

type i to start typing in vim; remove a # to create a comment (necessary)

then hit esc (gets us out of insert mode), then type :wq then hit enter

Code should now be ready for committing to online repository

git remote add origin https://github.com/BrendanAHaas/QuantEconProblems.git

git push –u origin master

ren gitignore.txt .gitignore (renames gitignore to remove .txt so that it works on windows)

**Definitions**

Namespace: A symbol table that maps names to objects in memory. Python uses multiple namespaces; each imported module has its own namespace (import math gives the variable math.pi = 3.141….)

Dataframe: An object for storing related columns of data

Series: A ‘column’ of data (ex: observations of variation in a single variable)

**Current Progress**

**Part 1** [**https://lectures.quantecon.org/py/numpy.html #3**](https://lectures.quantecon.org/py/numpy.html%20%20%20%20%20%20%20%20%20%20%20#3)

**Part 2 https://lectures.quantecon.org/py/python\_advanced\_features.html**