Lab Practical 1: Python Fundamentals Exercise and Jupyter Notebook

This week's practical will get you to set-up the python environment for the rest of the Data Science Lab practicals. You will work on an exercise using the Python Fundamentals that we reviewed in lecture. You will also be introduce to the Jupyter Notebook environment to write python code.

Setup

Using the Lab Machines

1. Boot the lab machine into CentOS Linux (this is default)

2. Install Anaconda

We will install a **package manager** called Anaconda, which you will use to manage your own software libraries throughout the rest of this module and other Data Science modules.

3. Download `install-anaconda.sh` script from KEATs.

4. Find the script on your file system.

It may be in the Downloads folder. Move it into your user home directory.

5. Open a Terminal window.

You will be in your home directory. Execute the installation script by typing in on the command line:

```
    sh install-anaconda.sh
```

This script may take 3-10 minutes to install. Use this time to read over the rest of this lab sheet. You can also get started using IDLE in the meantime for Part 1.

Install Jupyter Notebook.

Now that Anaconda is installed, **open a new Terminal window** (close the old Terminal window that you used to install Anaconda). You will use the `conda` command to install the jupyter notebook package into your set-up.

In the **new** Terminal window, run:

```
conda install jupyter
```

Using your own machine. Use conda to install the packages as above on your own machine.

Alternatively, you can use the Python Package Installer (pip) to install these packages. *However, you may have to install some dependencies, so it may take a long time!*

To install Jupyter Notebook:

```
pip install notebook
```

If you have trouble installing Jupyter Notebook, then use a lab machine for this practical.

If you did not attend the Data Science induction mini-course last week, and if this is your first lab, go to the Data Science Mini-course KEATs page and do *Getting Started with Python (lab instructions).* https://keats.kcl.ac.uk/mod/resource/view.php?id=1485793).

Part 1 My Words!

The game of Anagrams (https://en.wikipedia.org/wiki/Anagrams) involves making words using only the letters in a given word. In this exercise, you will create a simple game of Anagrams.

Here's how your game will work. The computer will pick a word from a data file of 7-letter words (i.e., words that are 7 letters long). (I've uploaded a data file on KEATS called `words7.txt` which contains a list of 7-letter words, one word per line in the file.) The computer will display the word to the user. The player then enters words that can be made out of the computer's word. Each word that the player enters is saved in a data file. The game counts how many words the player entered and displays that number at the end.

a. Download `lab1-anagrams.py` from KEATs and open in IDLE. If this is new to you, go to *Getting Started with Python* on the *Data Science Mini-Course* and read about getting started with using IDLE and starting a new program.

You will see that `lab1-anagrams.py` is structured into two functions: `read_words` and `main`. Write your code in the `main` function for the rest of the steps below, which are conveniently commented for you in the template.

b. Open the `words7.txt` data file (download the file from KEATS into the same folder as your Python programme) in a text editor and see the contents of the file.

c. Read the words from the file and store them in a list. To help you get started, the function `read_words` has been provided for you. It takes the name of the file you want to read the words from as an **input argument**, and it **returns** the words from that file as a list.

Read the "`words7.txt`" data file into a list using the `read_words` function.

d. Count how many words were read from the file, and randomly pick one to present to the player. If this is new to you, or if you need a refresher, review *Part 1* and *Part 2* on the *Data Science Mini-Course*. Take a look at the *Data Science Mini-Course Lab 2* for a review to generate `random` numbers. You will see that that the `random` module is already being imported for you at the top of your program file.

e. Show the player the chosen word, and ask them to make little words out of the chosen word. Take a look at the *Data Science Mini-Course Lab 1* to review using the `raw_input` function to get input form the player.

You will see that a *condition-controlled while* loop has been provided for you. Replace the line of code that says "`pass`" with your own code that asks the player for her little words.

f. Give the player the option to quit when they are tired of playing.

Now make your game more sophisticated by adding some *error checking*:

g. No duplicates. Keep track of the words that the player enters and make sure that they don't enter the same word more than once.

h. Valid words. Make sure that the user enters valid words, i.e., that they don't enter words that cannot be made from the letters of the chosen word. For example, if the chosen word is **abiotic**, then the word **bugs** would be invalid because there is no **u**, **g** or **s** in the chosen word.

i. Real Words. Now make your game even more sophisticated by making sure that the words your user enters are actually words.

- A list of valid words is contained in the large file called **words**, which is a standard dictionary file found on most Unix systems. You can access the words dictionary on the Lab's Linux computers by using the file path "`/usr/share/words`" (ask a TA for help on finding the words file).

- Or, you can download the `linuxwords.txt` file from KEATs and use that as you words dictionary.

- Every time the user enters a word that passes the error-checking tests above, check to see if the word is also a "real" word by looking it up in the dictionary file.

j. Add a scoring mechanism. For example, in the game of Boggle, players score points for each word they make based on the length of the word. Words have to be at least 3 letters long. Words that are 3 – 4 letters long are worth 1 point each. Words that are 5 letters long are worth 2 points. Words that are 6 letters long are worth 3 points. Words that are 7 letters long are worth 5 points. Words that are longer than 7 letters in length are worth 11 points.

Or make up your own scoring mechanism!

k. Record game play. It would be interesting to record the player's game play, and output each word the player write to a file. Modify your program to open an output file called "`mywords.txt`". For every word the player inputs, write a new line to `mywords.txt` that includes:

1. the presented game word
2. the player's word
3. whether the word was value or not (hint: use a boolean)
4. the score for that word (an invalid word has a score of 0)

Separate the above values by *commas*. So for example, if the random word is **abiotic**, and the user enters **bugs** followed by the word **bio**, then output file would contain:

```
abiotic,bugs,False,0
abiotic,bio,True,1
```

Be sure to close the output file when the user quit's the program.

Part 2 Introduction to Jupyter Notebook

Now we will introduce you to the Jupter Notebook development environment. A notebook is a different environment for executing python code than from using the script file and the command line with IDLE to run your code. It allows interactive editing and evaluating of your python code, and because of this interactivity it is popular as a data science tool.

a. Launch Jupyter Notebook, create a new Notebook.

From the command prompt, launch Jupyter notebook via this command:

```
    jupyter notebook
```

You will be launched into a *file manager*, and you will see whatever files there are in the current directory that you launched jupyter notebook in.

Create a new notebook by choosing *New -> Python2*. This will launch a notebook that utilises a python2 **kernel**. In jupyter notebook the kernel is what runs the code that you enter in the cells. Jupyter notebook supports programming languages other than python (i.e. the R stats language).

If you had other kernels installed, then you would see options to launch notebooks to utilize those kernels here (i.e. to launch a notebook that allows you to edit R).

To see all of the kernels supported by jupyter notebook:

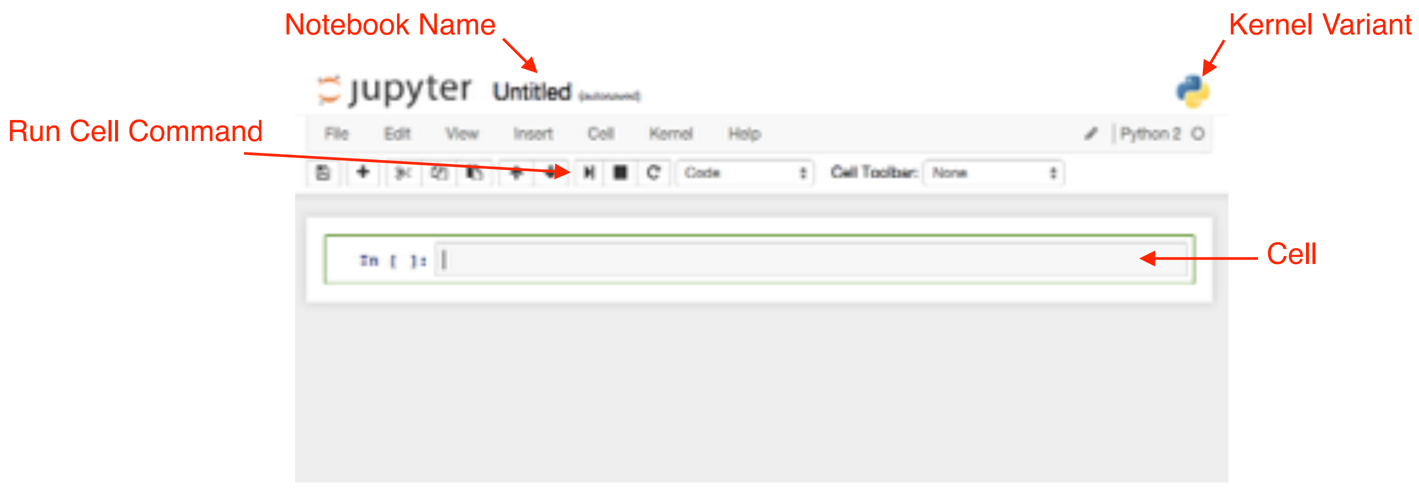
https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages

Each kernel has a different way to install it, for example here is how one would install the R kernel using the conda set-up:

https://www.continuum.io/blog/developer/jupyter-and-conda-r

b. Jupyter Notebook interface.

This is the Jupyter Notebook interface for your newly created notebook.



Click on *Help -> User Interface Tour.*

Step through the user interface elements to see what each thing does.

> *How do you get out of Editing mode when you are in a cell?*
> *How do you know when the Kernel is busy?*
> *What control do you use to change the cell type?*

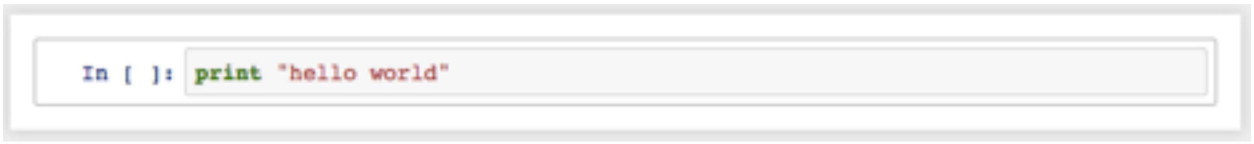
b. Change the Notebook Name

Click on Untitled, which is the current Notebook name. Edit it to “Hello Notebook”. You will notice that in your browser, you still have the File Manager browser window open. Switch to the File Manager, and click on the refresh button (the 🔄 button).

What happened to the *Untitled.ipynb*?

c. Execute some code

In the cell available, add this code:



```
In [ ]: print "hello world"
```

Click on the Run button,

See the results in the 'output area' below.

d. Save the notebook using the File Menus.

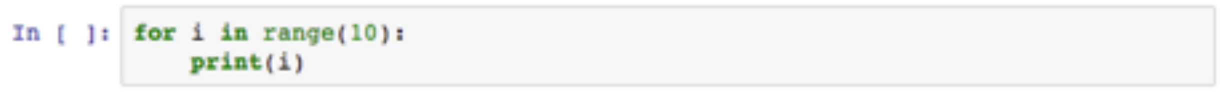
Notice that the notebook is automatically saved from time to time.

e. Create a new cell

Notice that when you ran your cell, a new cell was added for you below that cell. You can also insert a new cell using *Insert -> Insert Cell Below*.

f. Execute some code

In the new cell, add this code:

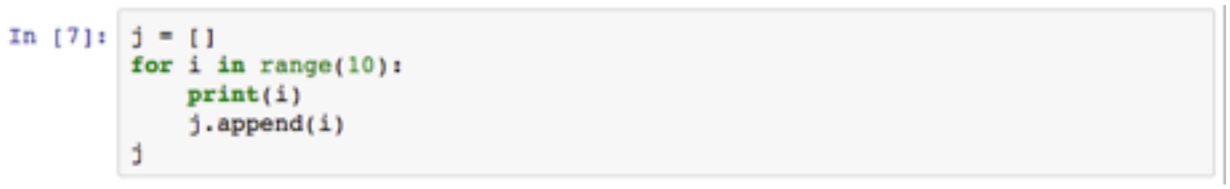


```
In [ ]: for i in range(10):
            print(i)
```

What is printed when you execute the code?

Jupyter will show what is printed below an executed cell.

Modify the cell, by adding this line:

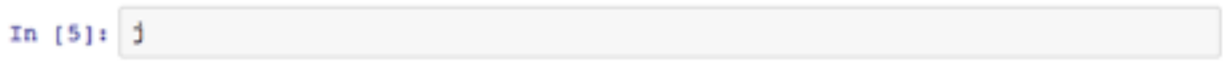


```
In [7]: j = []
        for i in range(10):
            print(i)
            j.append(i)
        j
```

What is printed when you execute the code?

Notice that the last line of code in the cell is just the variable '`j`'. Jupter Notebook will always print the representation of the last line of python code that you execute in a cell. In this case, the contents of the variable `j` are outputted as a string.

Now, add a new cell, and add this code:



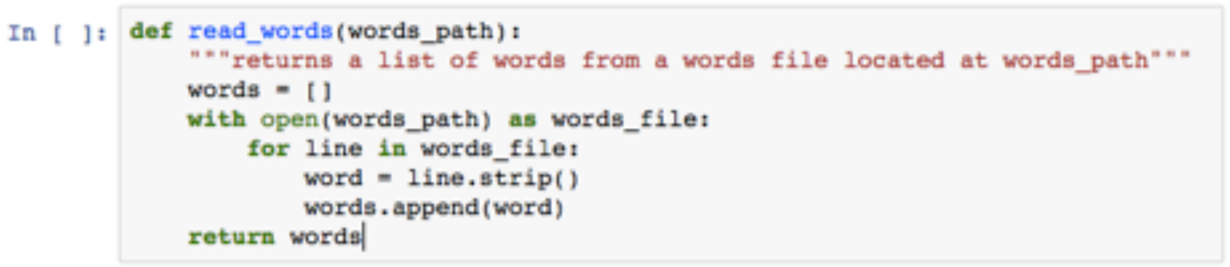
```
In [5]: j
```

What is printed when you execute the code?

You will see that '`j`' still holds the list that was created in the previous cell.

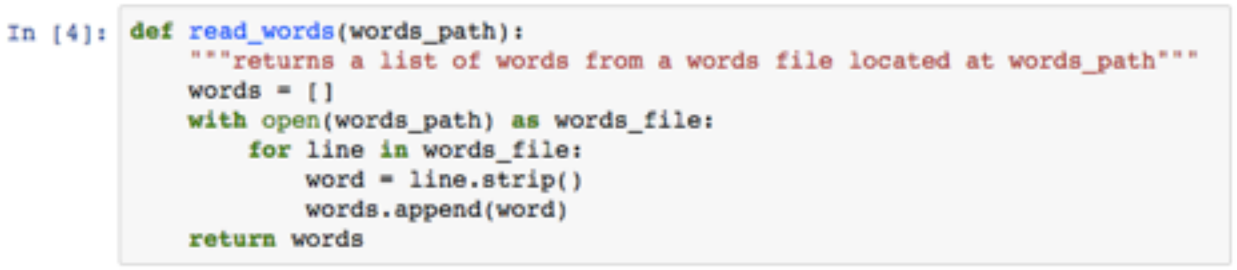
g. Add code to your notebook

From Part 1, copy the `read_words` function from your Anagrams program. You should have this as a cell:



```
In [ ]: def read_words(words_path):
            """returns a list of words from a words file located at words_path"""
            words = []
            with open(words_path) as words_file:
                for line in words_file:
                    word = line.strip()
                    words.append(word)
            return words
```

Run this cell. You'll see that the only thing that changes is a number will appear in the square brackets on the left hand side of the cell (i.e. "*In [4]:* ")



```
In [4]: def read_words(words_path):
            """returns a list of words from a words file located at words_path"""
            words = []
            with open(words_path) as words_file:
                for line in words_file:
                    word = line.strip()
                    words.append(word)
            return words
```

The number in the square brackets is the ***order*** in which the notebook cells have been run. Empty square brackets mean that the cell has not been run yet, and any functions defined in that cell are not yet available to be used in other notebook cells.

h. Read in a game play session (i.e. a “`mywords.txt`” file).

Create a new cell, and write code to read in the “mywords.txt” to a list.

You can use the example “`mywords.txt`” file on KEATs if your Anagram program is not completed to output game play.

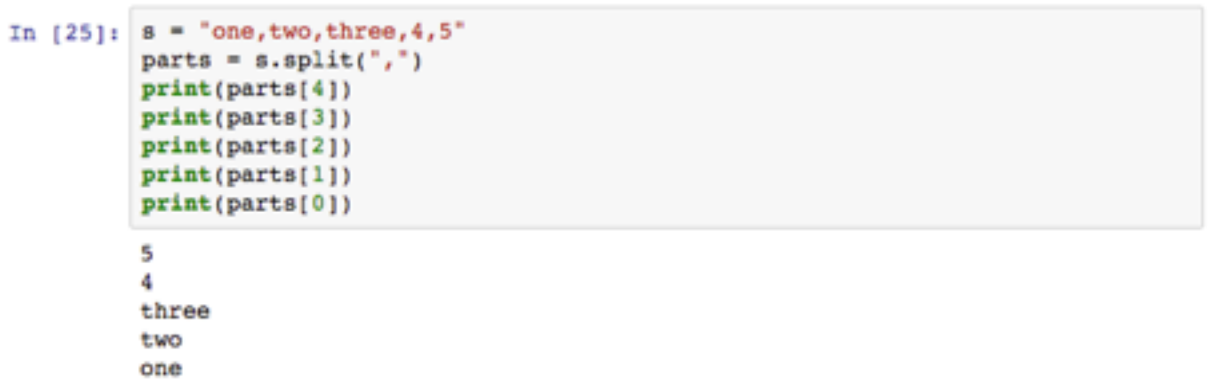
Do you get an error? Check that the data files you are reading are in the same directory as this notebook.

i. Total up the player’s score.

In a new cell, use a for-loop to total the player’s score. And print it to the output.

- Each element in the list that you read in from your session file is a string with *comma-separated values* (We will cover comma-separated values (i.e. CSV) files in the next lecture). You will have to `split` the string into it’s separate parts to access the last value (i.e. the score for that word).

Use the string `split` function to separate a comma-separated values string to a list of it’s separated values. For example,



```
In [25]: s = "one,two,three,4,5"
         parts = s.split(",")
         print(parts[4])
         print(parts[3])
         print(parts[2])
         print(parts[1])
         print(parts[0])
```

```
5
4
three
two
one
```

- Use the `int()` function to convert the score value from a string to an integer.

j. Total the number of valid and invalid word.

Modify your code to total the number of valid and invalid words. Print out the percentage of times the player got a valid word.

k. Export the Notebook as HTML

Save your notebook.

Using the file menu, export the Notebook as .html.

Open the .html in a web-brower. You should see the last output that you generated when you ran each cell. If not, go back to Jupyter Notebook, re-run the cells and output the HTML

l. Export the Notebook as Python

Using the file menu, export the Notebook as Python (.py).

Open the file in the text editor.

How is it different than the script python file from Part 1?

Run the file with the python interpreter. Does it run?

Congratulations! You have successfully created a Jupyter Notebook, run python code in it, and exported both HTML output and a modified Python script.

m. Download an external Notebook

Go to the Gallery of Interesting IPython (Jupyter) Notebooks.

https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks

Find a Notebook that looks interesting to you. If you are having a hard time deciding, here is one:

http://nbviewer.ipython.org/github/rasbt/python_reference/blob/master/tutorials/sorting_csvs.ipynb

Look at the output of the Notebook, note that these notebooks are well-crafted finished pieces. Notebooks in practise can get very messy and cluttered.

What do the authors use in these notebooks that help make them usable ?

Download this notebook (you should look for the .ipynb file).

Move the file into the same directory that you started Jupyter Notebook in.

Go to the File Manager Browser window, click on refresh and load the notebook.

Try running the cells in the Notebook.

Does it run? What issues or problems happen that might prevent the notebook from running?