Python Functions Review

This document lists some of the functions in Python that we used in lecture and gives examples of their syntax and uses. The jupyter notebooks from in-class examples (or that correspond to the lecture videos) are great places to start experimenting with the code and playing around with the various options.

- Basics:
  - **Variable assignment:** Python uses a single = to assign a value to a variable (for example my_favorite_number = 91. Variables can contain any data type.
  - **Data types:** We usually won't have much need to distinguish between numeric types. Strings are marked with either single quotes 'a' or double quotes "a". The Boolean values start with a capital letter: True or False.
  - **Conditionals:** Python uses a double == to check for equality and < and > have their normal meanings.
  - **Length:** To find the length of a string or list we use the len command: len("my string")
  - **Data Structures:**
    - **Lists:** In Python, a list is an ordered collection of objects contained in square brackets: my_list_of_numbers = [1,2,5,7,3]. The elements of the list can be accessed also using a square bracket notation, the command my_list_of_numbers[2] will return the number 5, since python is zero-indexed (it starts counting with 0).
    - **Dictionaries:** A dictionary maps input values, called keys, to output values, called values. They are wrapped in curly braces {} and use colons to separate the key from the value of reach pair, while commas separate the pairs. For example, if we set my_first_dictionary = { 'a':1, 'b':2, 13:'unlucky'} then typing my_first_dictionary['b'] will return the value 2.
- Pandas
  - pandas is usually imported with the abbreviation pd: import pandas as pd
  - Columns or vectors of data are represented as 'Series' objects: a = pd.Series([1,2,3,4,5,10,91]) b = pd.Series([1,2,-3,12,45,3])
  - Series objects support a wide variety of arithmetic functions like sum, average, max, etc. that are accessed with the . syntax: a.sum() or b.max()
  - The 'spreadsheet' analogue is known as a DataFrame: df = pd.DataFrame('WSU':a,'UI':b)
  - We usually load in data using the read_csv command rather than typing it directly: df = pd.read_csv("name_of_the_file_goes_here.csv")
  - We can extract a series corresponding to a column of a dataframe using the column name: df['WSU']
  - We can extract a row of a dataframe using loc and the square bracket syntax: df.loc[4]
  - If we want to extract a row or set of rows with a particular property, we can extend the loc syntax. For example, to find all rows corresponding to people that weigh over 200 pounds we could use: df.loc[df['Weight']>200]
  - To inspect the beginning or end of a dataframe we use the head or tail command: df.head()
  - To compute the basic summary statistics we use describe: df.describe()
  - To remove a row or column from a dataframe we use drop: df = df.drop(10) removes the 10th row from the dataframe.

- **Plotting**
  - MatPlotLib pyplot is usually imported as plt: import matplotlib.pyplot as plt
  - The various types of plotting functions we have encountered so far all take one or more lists (or pd.Series) as the main input and then a variety of other ancillary options to change the color or formatting.
  - **Bar charts:** Bar charts require exactly two vectors, the first lists the locations along the x-axis and the second lists the heights of the bars corresponding to those locations. For example, plt.bar([1,2,4],[100,300,200]) makes bars of heights 100, 300, and 200 placed at 1, 2, and 4.
  - **Line and dot plots** The plt.plot function takes either one or two vectors as input. If you supply a single list, it will plot those values along the y-axis against an x-axis that starts at 1. For example, plt.plot([30,40,60,100]) will generate a line plot with points at (1,30), (2,40), (3,60), and (4,100). To replace the line with a dot plot, the additional argument plt.plot([30,40,60,100],'o') or plt.plot([30,40,60,100],'*') will make a dot plot instead.
  - **Scatterplots:** If two vectors are supplied to plt.plot it will plot them against each other, with the first vector in the x–coordinate and the second vector in the y–coordinate. For example, plt.plot([-3,40,12,6],[30,40,60,100],'+') will make a scatterplot of the points (-3,30), (40,40), (12,60), (6,100).
  - **Boxplots:** The boxplot command takes any number of vectors joined together in a list and makes a boxplot for each vector separately - plt.boxplot([vector1,vector2,vector3,vector4]) will make four boxplots, placed vertically at the positions 1,2,3, and 4 along the x-axis.
  - **Histograms:** histograms are plotted individually, so plt.hist takes only a single vector as input and the main parameter to vary is the number or size of bins. plt.hist(vector1, bins=100) makes a histogram for the data in vector1 with exactly 100 bins.
- **Titles and Labels** Inside a figure we can add a title or labels on the x or y axes with plt.title("Title goes here"), plt.xlabel("x-axis label goes here"), or plt.ylabel("y-axis label goes here") respectively.
- **Legends** For plots with multiple elements, we can add labels to a legend to make the plot easier to understand. Individual labels are added to the plot elements: plt.plot([1,2,3],label='Male') plt.plot([4,5,6],label='Female') and then the legend is added to the plot with plt.legend().
- **Ticks** For things like bar charts where we usually want string labels along an axis or data that spans large ranges we may want to control the ticks directly. This can be done with the plt.xticks or plt.yticks commands, which each take two vectors. The first vector should be a list of numbers telling python where to place the ticks and the second is a list of strings describing what labels should be placed at those locations. For example, plt.xticks([1,20,100],['Airplanes','Cars','Trains']) will place three ticks on the x-axis at positions 1, 20, and 100 labelled with 'Airplanes','Cars', and 'Trains' respectively.
- **Multiple Column Analysis**
  - To compute correlation coefficients of the numeric columns of a dataframe we can use the df.corr() function.
  - We can plot a heatmap of the correlation values with
    import seaborn as sns
    sns.heatmap(df.corr())
  - To plot a full matrix of scatterplots for the numeric data in a dataframe we can use
    pd.plotting.scatter_matrix(df)
  - For categorical data we can compute a contingency table for two columns from a dataframe using pd.crosstab. For the abalone data that would look like: pd.crosstab(df['Sex'],df['Rings'])
  - To 'slice' the data in a column from a dataframe (say to create categories for plotting by color) we can use the rectangular bracket syntax and the between function. For example, to separate the income data in the COL dataset into three groups we can select:
    df_low = df[df['Avg Income']<1400]
    df_medium = df[df['Avg Income'].between(1400,2300)]
    df_high = df[df['Avg Income']>2300]
  - To round the numerical values in a dataframe we can use the round command: df.round(3) will represent all of the values with 3 decimal places.