Lesson 1: Introduction to computing using python and Jupyter Notebooks

Key Ideas:

1. What Jupyter notebooks are and how to use them
    - Code cells vs. markdown (text) cells
2. Computing and arithmetic with python
    - Using code cells for computation
    - Using comments
3. Naming quantities and objects
4. Understanding different types of data
5. Using functions
6. Python libraries
7. Lists and data frames

1. Jupyter Notebooks

A Jupyter notebook is a type of document that can be used for both writing text (like a Word document) and writing codes.

A Jupyter Notebook consists of cells (places to enter our texts or codes). There are two main types of cells in a Jupyter notebook:

- Markdown cells": for writing texts
- Code cells: for writing codes and carrying out computations and to enter codes/commands.

This paragraph, for example, is written in a markdown cell. This page (https://www.markdownguide.org/cheat-sheet/) is a useful guide for how texts in markdown cells can be formatted.

Exercise

Double click this text to edit this markdown cell; replace this sentence with your name.

Exercise

Add a new cell below this one by first clicking this markdown cell to select it and then clicking the "+" sign on the menu bar near the top left corner. By default, the new cell will be a code cell. To change a code cell into a different type of cell, select the cell and use the drop-down menu on the top.

2. Computing and arithmetic in python

Using Code Cells

One of the simplest tasks we can dowith python is basic computation and arithmetic operations such as adding, subtracting, multiplying and dividing, and taking exponents/powers.

Example

In the code cells below, we compute $15+3$, 15×3, 15^3, and $(15-3)*2^4$. To "run" the commands (to carry out computation), click on the code cell and press Shift + Enter or click on the "Run" button on the top menu.

```
In [ ]: 15 + 3
In [ ]: 15 * 3 # * is used for multiplication
```

```
In [ ]: 15 ** 3 # ** is used for exponentiation
```

```
In [ ]: (15 - 3) * 2 ** 4  # order of operations is observed
```

Comments

In the code cells above, everything written to the right of the `#` symbol are called "comments" and are ignored when codes are run. The purpose of comments is to explain the code to future (human) readers. It is good practice to leave short comments to explain codes and computation.

Exercise

To convert from Fahrenheit to Celcius, we subtract 32 and multiply by 9/5.

In the code cell below, please convert 73.5F to Celcius.

```
In [ ]: # add your work for the above exercise here
```

3. Naming quantities and objects

Sometimes, we would like to give names for the values that we are working with so that we can easily refer to them.

This is similar to how math variables might stand for a value (e.g. saying $x = 2$ means that $x + 4$ stands for the value 6). That is, names are "labels" that point to a particular number or other objects.

In python, we assign a value to a name, we use “=”

```
NAME = VALUE
```

For example:

```
x = 2
my_favorite_number = 3.456
```

means that `x` now “stands for” the number 2 and `my_favorite_number` now stands for the number 3.456.

Rules for choosing python names/variables

In general, it is useful to choose names that describes the quantity that it represents. Names must satisfy the following constraints:

- names can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- must start with a letter or the underscore character
- cannot start with a number
- cannot contain spaces
- case-sensitive

Example

Please run the code cells below by clicking the code cell and pressing Shift + Enter.

- In the first code cell, we will see that 2 is being displayed, because it is the value set for `x`.
- In the second code cell, we will see that only 3.456 is being displayed, but 2 isn't. This is because only the latest value will be displayed.
- In the third code cell, we see that both 2 and 3.456 are displayed. This is because we use the `print(NAME)` function to explicitly tell python ot display the value represented by each name. We also see that this code cell displays the result of computing `x * 2 + 3`, which is 7 because `x` points to the value 2.

```
In [ ]: x = 2  # here we set the value of the name x to 2
        x      # here we display the value represented by x
```

```
In [ ]: x = 2
        my_favorite_number = 3.456
        x
        my_favorite_number
```

```
In [ ]: x = 2
        my_favorite_number = 3.456
        print(x)
        print(my_favorite_number)
        print( x * 2 + 3)
```

Exercise

In the code cell below, create a variable name `temp_Fahrenheit` that points to the value 73.5.

Then, use create a variable name `temp_Celcius` whose value is the result of converting 73.5F to Celcius. (Recall that to convert from Fahrenheit to Celcius, we subtract 32 and multiply by 5/9.)

```
In [ ]: # add your work for the above exercise here
```

4. Data Types

There are different types of data that python can work with. Below are the main data types that are most important for us:

- int: Integer/whole numbers
- float: Numbers that could involve fractional parts
- str: String/text data. In python, a text value is surrounded by a pair of either single or double quotes and could contain any characters.

We can give names not only to numbers but also to text values (and other types of objects).

Example

In the following code cell, we store the text 'The Color Purple' in name/variable `my_favorite_book` , and display this value.

```
In [ ]: my_favorite_book = 'The Color Purple'
```

```
In [ ]:
```

5. Using Functions

python allows us to do a lot of things using "functions". Functions in python is somewhat analogous to mathematical functions.

For example, $f(x) = x^2$ means that we have a function called f which takes an input x and whose output is x multiplied by itself. The "formula" for the function, $f(x) = x^2$, specifies the action that the function f does to the input x.

We can also think of functions in python as "verbs" which we can use to tell python to carry out a particular action or task. Just as some verbs in English must be followed by a noun ("transitive verbs") and some don't, some functions in python must take a particular object or input (often called an "argument") and some don't.

A function

- takes **input(s)** (or argument(s)) and
- **does something** based on / to the inputs.
- A function might also returns **output(s)**.

A few simple python functions

`print(NAME)`

- Input: a name
- Action: to print the value represented/referred by a name
- Output: the value referred by that name

`type(NAME)`

- Input: a name
- Action: to output the type of data referred to by a name
- Output: the type of data referred to by that name

We will see more python functions later.

Example

The code cell below confirms that the value stored in the name `x` is an integer/whole number.

```
In [ ]: x = 2
        type(x)
```

Exercise

Please find the type of values stored in the names `my_favorite_number` and `my_favorite_book` below, using the `type()` function.

```
In [ ]: my_favorite_number = 3.456
```

```
In [ ]: my_favorite_book = 'The Color Purple'
```

Exercise

Use the empty code cell below to experiment. Create new variables, assign values (numerical or text) to them, use the `print()` function to display the values, and use the `type()` function to display the type.

```
In [ ]:
```

6. Python libraries

Python is a general purpose programming language. In order to ask python to carry out certain types of specialized tasks (e.g., data analysis, etc.), we need to ask python to "fetch" or to "import" specialized "toolboxes". These "toolboxes" are called libraries or packages.

Some of the libraries that we will use:

- numpy: for mathematical computation
- pandas: for working with data tables
- seaborn: for making plots
- sklearn: for working with mathematical models

Importing a library

To ask python to load/import a particular library, we use the `import` command:

```
import numpy
```

We can also give libraries a nickname of our choice:

```
import numpy as np
```

Here, `np` could have been any other nickname that we choose. However, `np` is a commonly used nickname for the `numpy` package, so we will use that here.

Looking ahead, to import the `pandas` and `seaborn` libraries, we use:

```
import pandas as pd
import seaborn as sns
```

Exercise

Please type

```
import numpy as np
```

in the code cell below, and run it.

```
In [ ]: # import ......
```

The contents of a library

A library contains "tools" that we can use to accomplish various tasks. One of the main types of tools in a library are **functions**.

For example, numpy contains mathematical functions (taking a square root, absolute value, etc.). The function for taking the square root of a number is `sqrt( NUMBER )`. But since this function can only be found inside of the `numpy` library, we need to write `np.sqrt( NUMBER )` to use the function.

Example

In the code cell below, we compute the square root of 4 using the `np.sqrt( NUMBER )` function. Before running the code cell below, please make sure you have imported numpy as np in the code cell above!

```
In [ ]: np.sqrt(4)
```

We will use other tools inside the numpy and pandas toolboxes later in this lesson.

7. Lists and Data Frames

Lists

Sometimes, we need to work not just with one number but a collection of numbers; in python, these collections of numbers are called lists.

Lists are created by placing the list elements between a pair of square brackets, separated by commas. As with numbers and texts, we can give names to lists.

For example, to specify a list named `Mylist` which contains the numbers 1, 3, 5, and 7, we use the command

```
Mylist = [ 1 , 3 , 5 , 7 ]
```

A list could contain texts, numbers, or even other lists. For example, the list `Mylist2` below has three elements: the text 'The Color Purple', the number 1, and the list [1, 3, 5, 7].

```
Mylist2 = [ 'The Color Purple', 1, [1, 3, 5, 7] ]
```

Extracting elements of a list

Python indexes elements starting from 0, 1, 2, ... So, the “first” element has index 0, the “second” element has index 1, etc. To get the element at index `n` from a list `LISTNAME` , we use the square brackets:

```
LISTNAME[n]
```

Example

In the code cell below, we specify a list named `Mylist` which contains the numbers 1, 3, 5, and 7.

- Please run to code cell to verify that this is done correctly.
- Then, extract the number 5 from this list (which is located at index 2)

```
In [ ]: Mylist = [ 1 , 3 , 5 , 7 ]
        Mylist
```

```
In [ ]: # extract the number 5 from Mylist above (the number 5 is located at index 2)
```

Example

In the code cell below, please specify a list named `Mylist2` which contains the text 'The Color Purple', the number 1, and the list [1, 3, 5, 7].

What is the element at index 2?

```
In [ ]: # please complete the code cell below to create a list called Mylist 2
        #  which contains the text 'The Color Purple', the number 1, and the list [1, 3, 5, 7].

        Mylist2 =
```

```
In [ ]: # After completing the above code cell, please run this code cell to extract the element at index 2
        #  Does the result make sense?

        Mylist2[2]
```

Useful functions for working with lists and arrays

The following are a few other useful functions that we can use to examine lists:

- `len( LISTNAME )` : to find the "length" of a list (i.e. how many values are stored in a list
- `max( LISTNAME )` : to find the largest value in a list
- `min( LISTNAME )` : to find the smallest value in a list
- `np.sum( LISTNAME )` : to find the sum of numerical values in a list or a numpy array
- `np.mean( LISTNAME )` : to find the mean of numerical values in a list or a numpy array

Exercise

Please try the above functions on the lists `Mylist3` and `favorite_books` below

```
In [ ]: Mylist3 = [ 10, -13.3, 200, 1, 0.0034 ]
```

```
In [ ]: favorite_books = ['The Color Purple', 'The Sun Also Rises', 'The Hobbit']
```

```
In [ ]:
```

Arithmetic Operations on Lists

Suppose that we would like to carry out arithmetic operations to numbers in a list. For instance, suppose that the list `temp_forecast` below stores average daily tempreatures, in Fahrenheit, for the next 7 days, and we would like to convert each temperature to Celcius.

```
temp_forecast = [ 73.5, 82, 69.5, 68 , 70.1 ]
```

We could do the following one-by-one to each of the five numbers: subtract 32 from each number and multiply by 5/9. However, this is tedious. It would be nice if we could simply carry out this arithmetic operation directly using the list name, as below.

```
(temp_forecast - 32) * 5/9
```

However, we will see below that this does not work.

Example

Please run the code cells below. We expect to see an error message in the second code cell.

```
In [ ]: temp_forecast = [ 73.5, 82, 69.5, 68 , 70.1 ]
        temp_forecast
```

```
In [ ]: (temp_forecast - 32) * 9.5
```

Numpy Arrays

The good news: We can carry out arithmetic operations to a numpy array.

A numpy array is a python object that is essentially a list of numbers, but one that we could apply arithmetic operations to.

To create a numpy array, we use the function `np.array( LISTNAME )` from the numpy library.

- Input: a list of numbers
- Action: convert the list to a 'numpy array', a special type of list-like object that we can apply arithmetic operations to
- Output: a numpy array

Example

In the example below, we create a numpy array called `temp_forecast_array` and apply arithmetic operations to convert the temperatures stored in this array to Celcius.

Please run the code cells below, in order, and confirm that the list `temp_forecast_array_Celcius` contains the result of applying arithmetic operations to `temp_forecast_array` .

```
In [ ]: temp_forecast = [ 73.5, 82, 69.5, 68 , 70.1 ]  # a list of numbers

        temp_forecast
```

```
In [ ]: import numpy as np                             # import the numpy library, give it a nickname np
        temp_forecast_array = np.array(temp_forecast ) # create an array

        temp_forecast_array
```

```
In [ ]: temp_forecast_array_Celcius = (temp_forecast_array - 32) * 5 / 9
        temp_forecast_array_Celcius
```

Exercise

Please create a numpy array called `heights_inches` which contains the height of five people (in inches): 73, 64, 69, 58, 75.

Please multiply this array by 2.54 to create a numpy array called `heights_cm` containing the conversion of these numbers to centimeters.

```
In [ ]:
```

Arithmetic operations between two numpy arrays

If we have two numpy arrays that have the same number of elements, then applying arithmetic operations between two arrays results in another numpy arrays where the operations are done element-wise.

Example

Consider the two arrays below, `array1` and `array2` ; each array contains 4 numbers. Adding the two arrays together, `array1 + array2` results in an array of length 4 whose elements are the sum of corresponding elements of `array1` and `array2` . Similarly with other operations (multiplication, division, exponentiation, etc.)

```
In [ ]: array1 = np.array( [1, -3, 2.5 ] )
        array2 = np.array( [-10, 7, 2 ] )
        print(array1)
        print(array2)
```

```
In [ ]: array1 + array2
```

```
In [ ]: array1 - array2
```

In [ ]:
```
array1 * array2
```

In [ ]:
```
array1 / array2
```

In [ ]:
```
array1 ** array2  # exponentiation
```

Data Frames

Recall that pandas is a python library for working with data. To work with data frames, we first import the pandas library.

Exercise

Please run the code cell below to import pandas, giving it a "nickname" pd.

In [ ]:
```
# run this code cell

import pandas as pd
```

Data frames simply as the term that pandas uses to refer to data tables (similar to how excel spreadsheets organize data, in rows and columns).

More precisely, though, data frames are a particular type of objects in python/pandas. We have seen other objects as well earlier in this homework: lists are objects, numpy arrays are objects, integers are objects, etc.

There are two main ways we can get data frames in python.

1. Importing a file from a directory in you computer or from the internet
2. Entering data manually into a data frame

1. Importing a file from a directory in you computer or from the internet

A common format to store data is a “comma separated values” file (.csv). We use the function

```
pd.read_csv( 'FILENAME' )
```

when we want pandas to "read" a csv file and store it as a data frame:

- Input: the name of the csv file located in the same directory as the jupyter notebook
- Output: a pandas data frame

Example

In this lesson01 folder, we have a csv file called `berkeley73.csv` . In the code cell below, we load the content of this file as a pandas data frame and name it `berkeleydata` .

In [ ]:
```
berkeleydata = pd.read_csv('berkeley73.csv')
berkeleydata
```

We can also use the `pd.read_csv()` function to read a csv file from a URL:

```
pd.read_csv(‘URL’)
```

Example

The following URL is a link to global temperature time series dataset from DataHub (https://datahub.io/core/global-temp):

https://pkgstore.datahub.io/core/global-temp/annual_csv/data/a26b154688b061cdd04f1df36e4408be/annual_csv.csv (https://pkgstore.datahub.io/core/global-temp/annual_csv/data/a26b154688b061cdd04f1df36e4408be/annual_csv.csv)

If we simply click on this link, we will see the csv file that contains the temperature data. To load this dataset as a pandas data frame, we use the `pd.read_csv()` function. Please run the following code cell to accomplish this.

In [ ]:
```
global_temp = pd.read_csv('https://pkgstore.datahub.io/core/global-temp/annual_csv/data/a26b154688b061cdd04f1df36e4408be/annual_csv.csv')

global_temp
```

2. Entering data manually into a data frame

Occasionally, we might want to type in our data manually into a data frame. To do this, we use the `pd.DataFrame()` function

```
pd.DataFrame( { ‘COLNAME1’ : LIST1, ‘COLNAME2’: LIST2, … } )
```

Example

Suppose that an animal shelter has the following data on animals in their care that are ready for adoption.

Name	Species	Age_yr	Weight_lb
Artoo	Dog	5	121.7
Bunbun	Rabbit	2	5.1
Catherine	Cat	12	15.5
Doug	Dog	3	17.2
Ernie	Guinea Pig	1	2.7

We will create a dataframe called `animals` containing the above information (four variables, five observations).

```
In [ ]:
```

To check that the `animals` dataframe has been created correctly, run the code cell below.

```
In [ ]:
```

This is the end of Lesson 1. You have now learned the basics of working with datasets using python and Jupyter notebooks. You are encouraged to try the exercises below to practice your new skills.

Lesson 1 Exercises

Exercise 1 *(Using python for computing and arithmetic; using descriptive variable names and comments)*

Ali and Sam went to a restaurant and together purchased a $11.48 beef enchilada and a \$9.62 quesadilla. Sales tax for restaurants in this city is 8.875\%. Please use python to answer the two questions below.

1. How much is their total bill, including this tax (but not including additional tip)?
2. Suppose that they added a $3.15 tip at the end. How many percent of the pre-tax total is this tip?

Note: Instead of simply entering one line of numbers and arithmetic operations, you must do the following:

- Give descriptive names to the various numbers that are involved.
  (For example, `enchilada = 11.48` , `taxrate = 0.08875` , etc.)
- As needed, give names to some of the intermediate quantities.
  (For example, `subtotal = ...` , etc. before computing the total bill.)
- Add a couple of short comments to explain the computation. (Imagine that you have to explain the arithmetic to a foreign friend who isn't familiar with how tax and tips work.)

Please briefly explain your work as a comment or in the provided markdown cell.

```
In [ ]:
```

...

***Exercise 2** (Working with lists and arrays)*

Charlie commutes to school using public transportation and regularly takes notes of their daily morning commute length. Below are their commute lengths (in minutes) in one particular week.

30, 62, 40, 42, 39

1. Please create a **list** called `commute` that stores Charlie's commute times in minutes.
2. On average, how long is Charlie's morning commute? Use the `sum()` and `len()` functions to compute this number.
3. Create a **numpy array** called `commute_array` that stores Charlie's commute times. (Use the `np.array()` function we learned above.) Then, multiply `commute_array` by 60 and name the result `commute_array_seconds` . Note that `commute_array_seconds` is a numpy array that stores Charlie's commute times, in seconds.
4. Try multiplying the **list** `commute` by 60. Does it give you the same result as when you multiply the **numpy array** `commute_array` by 60 ? What do you observe?
5. In Question 2 above, we were able to compute Charlie's average morning commute using the `sum()` and `len()` function. We will compute Charlie's average morning commute using another method.
   There is a function called `mean()` **from the numpy library**, which will compute the average of an array of numbers. Please use this function to compute the average length of Charlie's morning commute (in minutes). Hint: since the mean function is from the numpy library, use `np.mean()` .
6. How much longer than average is Charlie's longest commute time last week? How much shorter than average is Charlie's shortest commute time last week?

```
In [ ]:
```

...

***Exercise 3** (Working with data frames)*

Charlie took note not just of their commute times but also of other weather data everyday in one particular week.

Day	Commmute_length	Temperature	Humidity
Day 1	30	5	121.7
Day 2	60	2	5.1
Day 3	40	12	15.5
Day 4	42	3	17.2
Day 5	39	1	2.7

Please create a data frame called `commute_data` that contains the above information (four variables and five observations).

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```