COGS 118B- Assignment 0: Setting Up by Qiyuan Wu and Owen Zhang

This assignment is out of 15 points, worth 5% of your grade.

The following couple cells are checks that you are running the right tools for these assignments.

In [97]:
```
# These assignments require Python 3 (not Python 2).
# Run this cell to check your version of Python.
import sys
assert sys.version_info[0] == 3, "You must use Python 3 (preferably 3.6 or 3.7) for these assignments."
```

In [98]:
```
# Run this cell to check your version of IPython.
# If you get an error (the message prints out), stop and upgrade Jupyter.
import IPython
assert IPython.version_info[0] >= 3, "Your version of IPython is too old, please update it."
```

This part of the assignment is focused on some practice with Python, and with practicing working with the format of the assignments.

This class assumes some prior knowledge of Python. In the following questions, you will need to work with basic (standard library) data types (floats, lists, dictionaries, etc.) and control flow (conditionals, loops, functions, etc). If the questions in this section are totally unfamiliar to you, you may need revisit some practice materials to catch up with some of the programming.

Through these questions, we will also prompt you to use a couple slightly more advanced standard library functions (for example, 'enumerate' and 'zip'), that may be new to you.

Each question should be answerable with a relatively small number of lines of code, up to about 5-7 lines at most.

In [99]:
```
# PRINTING VARIABLES
# A reminder that you can (and should) print and check variables as you go.
#  This allows you to check what values they hold, and debug if anything unexpected happens.

# Define a variable
math_result = 2 * 4

# Print out the value(s) of a variable.
print(math_result)
```

```
8
```

Question 1: Defining variables (0.5 points)

Define a tuple called `var_a`, that contains the numbers 1-10 (inclusively). Define a list called `var_b`, that contains individual letters a-j (inclusively).

In [100…
```
# YOUR CODE HERE
var_a = tuple(range(1,11))
var_b = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

In [101…
```
# Tests for Q1

# These tests check the variables are defined
assert var_a
assert var_b

# These tests check that the variables are the right data types
assert isinstance(var_a, tuple)
assert isinstance(var_b, list)
```

Question 2: Defining Variables, Part II (0.5 points)

Create a Python dict called `dictionary` where the keys are the elements in `var_b` and the values are the corresponding elements in `var_a`.

The `zip` function may be useful.

You might also make use of a Python dictionary comprehension

In [102…
```
# YOUR CODE HERE
dictionary= dict(zip(var_a, var_b))


# These tests check that dictioinary has the right data types
assert isinstance(dictionary,dict)

print(dictionary) # Do not delete
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j'}
```

Question 3: Control Flow (1 point)

Loop through the provided list `lst`. For each element, check if it is an even number. If the element is an even number, append the INDEX of that element to the list `inds`.

Note that you are adding the index to `inds`, **not the element itself**.

Hint: to check if a number is even, you can use the modulo `%` operator.

Hint: to loop through an iterable, keeping track of the index, you can use the `enumerate` function.

In [103…
```
# These variables are provided to you.
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
inds = []
```

```
# YOUR CODE HERE
for i in range (len(lst)):
    if lst[i] % 2 == 0:
        inds.append(i)


print(inds) # Do not delete
```

```
[1, 3, 5, 7, 9, 11, 13]
```

Question 4: Functions (1 point)

Write a function `squared_diff` that takes two number inputs and returns the squared difference of the two numbers i.e. $(a-b)^2$. For example:

```
>>> squared_diff(2, 4)
4
>>> squared_diff(10, 1)
81
```

In [104…

```
...  # YOUR CODE HERE
def squared_diff (a,b):
    return (a-b)**2

assert squared_diff(2,4) == 4     # Do not delete
assert squared_diff(10,1) == 81   # Do not delete
assert squared_diff(-10,2) == 144 # Do not delete
assert squared_diff(15,16) == 1   # Do not delete
assert squared_diff(16,16) == 0   # Do not delete
```

Question 5: Putting it all together (1 point)

Create a dictionary named `second_dictionary`. It should have the key-value pairs as `dictionary` (the variable we created in question 2), unless the original value is odd. If so, the value in `second_dictionary` should be the squared difference of the original value and 10.

Note: to loop through key-value pairs in a dictionary, check out the `.items` method.

In [116…

```
second_dictionary = dict()
# YOUR CODE HERE
for key, value in dictionary.items():
    if key % 2 == 0:
        second_dictionary[key] = squared_diff(key, 10)
    else:
        second_dictionary[key] = value


print(second_dictionary) # Do not delete
```

```
assert second_dictionary['a'] == 81   # Do not delete
assert second_dictionary['c'] == 49   # Do not delete
```

```
{1: 'a', 2: 64, 3: 'c', 4: 36, 5: 'e', 6: 16, 7: 'g', 8: 4, 9: 'i', 10: 0}
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/var/folders/0y/nzqg4x0d6ms4hrqngsbxd6840000gn/T/ipykernel_4385/1618477100.py in <module>
     10 print(second_dictionary) # Do not delete
     11
---> 12 assert second_dictionary['a'] == 81   # Do not delete
     13 assert second_dictionary['c'] == 49   # Do not delete

KeyError: 'a'
```

Question 6: Data Science Modules (1 point)

This question is just about importing the more core data science modules. We won't start using them yet.

This is just to test that you have them available and working (this is the only time we'll ask you to write out your own import modules in assignments).

If you have not loaded those modules into your envoirnment yet, you can run the `!pip install < package name >` commmand.

If you are unable to import them, ask for help in office hours.

Import the numpy, scipy, pandas, matplotlib.pyplot, sklearn modules as np, sp, pd, plt, skl, respectively.

In [106…
```
# YOUR CODE HERE
import numpy as np, scipy as sp, pandas as pd, matplotlib.pyplot as plt, sklearn as skl
```

In [107…
```
# Tests for Q6
assert(np.array)        # Do not delete
assert(sp.integrate)      # Do not delete
assert(pd.DataFrame)    # Do not delete
assert(plt.plot)        # Do not delete
assert(skl.set_config)  # Do not delete
```

Question 7: Basic Matrix Operations (4 point)

Define,

$$A = \begin{bmatrix} 3 & 5 & 2 \\ 6 & 1 & 7 \\ 2 & 5 & 2 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}$$

$$Ax + b = Y$$

Find x.

Hint: use numpy.linalg.inv and the @ operator for matrix multiplication.

In [108…
```
# YOUR CODE HERE
A = np.array([[3, 5, 2], [6, 1, 7], [2, 5, 2]])
b = np.array([[1], [1], [1]])
Y = np.array([[4], [3], [2]])

A_inv = np.linalg.inv(A)
x = A_inv @ (Y - b)

print(x)                          # Do not delete
assert isinstance(x, np.ndarray) # Do not delete
```

```
[[ 2.        ]
 [-0.03030303]
 [-1.42424242]]
```

Question 8: Data Manipulation (5 points)

In this question, we will be using the Iris dataset. First, you need to obtain the dataset by running `from sklearn import datasets` and `iris = datasets.load_iris()`.

1. Print the shape of the dataset. You will see that the dataset has 150 sample and each sample has 4 features.(Note: the `load_iris` method returns a dictionary-like object and the dataset is store in one of the object attributes) (0.5 point)

In [110…
```
# YOUR CODE HERE
from sklearn import datasets
iris = datasets.load_iris()
print(iris.data.shape)
```

```
(150, 4)
```

2. Print the last three samples of the dataset. (0.5 point)

In [117…
```
# YOUR CODE HERE
print(iris.data[-3:])
```

```
[[6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

3. Perform a linear projection on the samples using vector $w = [1, 2, 3, 4]$. We can do this by calculating the dot product between the data points and the vector w. The shape of the output vector should be (150,) or (150,1). Find and print the mean of the projected vector. Hint: You can use numpy.dot for calculating the dot product. This problem can be done using a single line of code. (2 points)

In [118…
```
# YOUR CODE HERE
proj = np.dot(iris.data, [1,2,3,4])
print(proj.mean())
```

```
28.029333333333337
```

4. Randomly select 3 samples (rows) from the dataset. Print the indices and the selected data points. Hint: use numpy.random.randint to sample the indices. (1 point)

In [119…
```
# YOUR CODE HERE
indices = np.random.randint(0,len(iris.data), size  = 3)
random_samples = iris.data[indices]
print (random_samples)
```

```
[[6.2 2.2 4.5 1.5]
 [5.2 3.4 1.4 0.2]
 [6.6 3.  4.4 1.4]]
```

5. Add one more feature (one more column) to the right side of the dataset. The value of the new feature for each sample is 1. Print the first 5 samples of the new dataset. (1 point) Hint: Use nupmy.ones and numpy.hstack.

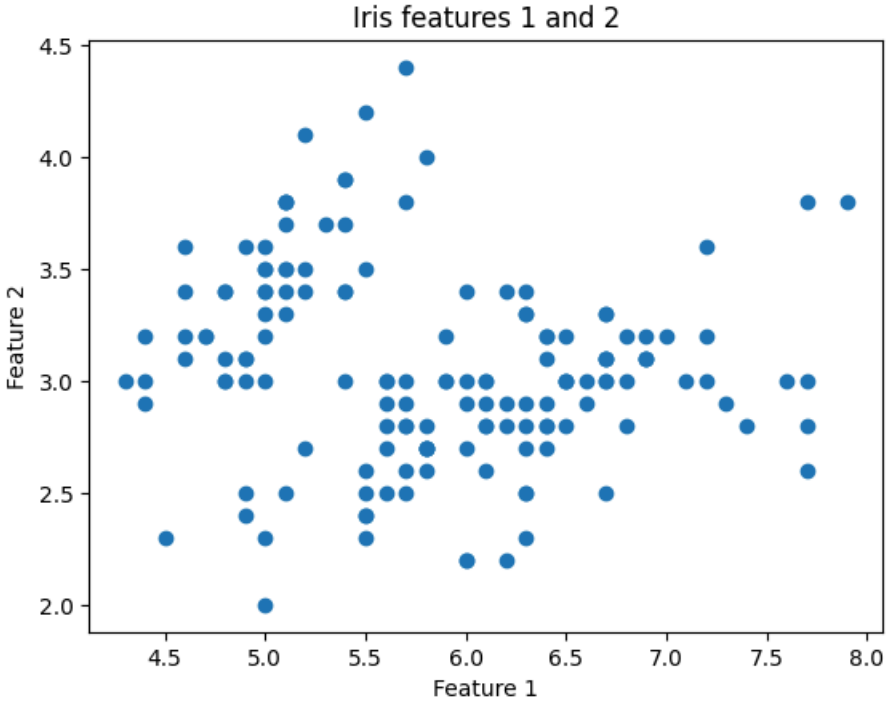
In [113…
```
# YOUR CODE HERE
new_feature = np.ones((len(iris.data),1))
new_data = np.hstack((iris.data, new_feature))
print(new_data[:5])
```

```
[[5.1 3.5 1.4 0.2 1. ]
 [4.9 3.  1.4 0.2 1. ]
 [4.7 3.2 1.3 0.2 1. ]
 [4.6 3.1 1.5 0.2 1. ]
 [5.  3.6 1.4 0.2 1. ]]
```

Question 9: Data Presentation (1 point)

Use Matplotlib to plot the samples on a 2D plot. Use the values of the first and second features as the x and y coordinates. Make sure it is a scatter plot.

In [114…
```
# YOUR CODE HERE
plt.scatter(iris.data[:,0], iris.data[:,1])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Iris features 1 and 2')
plt.show()
```



STOP ! Have you run all the code blocks ?

If not, please do so. All the requested code output should be visible.

Submission:

Please submit your notebook (.ipynb file) and an exported PDF copy (you can upload seperately without zipping the files) on GradeScope under the correct assignment with all of the cell outputs visible. It is normal to see Grapescope auto-grader outputs a score of 0.

If you are running on Datahub or the Anaconda Navigator and you are having trouble with exporting to PDF, you can use browser print page. If nothing works, you can submit without the PDF. Note, the .ipynb notebook file is required.