CIND830 - Python Programming for Data Science

Assignment 3 (10% of the final grade)

Due on April 11, 2022 11:59 PM

This is a Jupyter Notebook document that extends a simple formatting syntax for authoring HTML and PDF. Review this (https://jupyter-notebook.readthedocs.io/en/stable/notebook.html) website for more details on using Jupyter Notebooks.

Use the JupyterHub server on the Google Cloud Platform, provided by your designated instructor, for this assignment. Ensure using **Python 3.7** release or higher then complete the assignment by inserting your Python code wherever seeing the string `#INSERT YOUR ANSWER HERE.`

When you click the `File` button, from the top navigation bar, then select `Export Notebook to HTML` , an HTML document will be generated that includes both the assignment content and the output of any embedded Python code chunks.

Use these (https://www.ryerson.ca/courses/students/tutorials/assignments/) guidelines to submit **both** the IPYNB and the exported file (HTML). Failing to submit both files will be subject to mark deduction.

Please be advised that you cannot get more than 100% in this assignment, and the **BONUS** question (if there is any) will only be graded if all other questions have been submitted.

Question 1 [20 pts]:

Download the iris dataset (https://data.mendeley.com/datasets/7xwsksdpy3/1). Ensure that the downloaded CSV file has 151 rows and 5 columns, and the last column presents the iris flower type: Setosa, Viginica or Versicolor.

a) (10 Points) Write a script that reads the iris data and stores it in a dictionary, where the `Keys` are the heads of the columns and the `Values` are the data of the respective column.

The output of running this script would be similar to the following:

```
{'sepal_length': [5.1, 4.9, 4.7, 4.6, ... , 6.3, 6.5, 6.2, 5.9],
 'sepal_width' : [3.5, 3.0, 3.2, 3.1, ... , 2.5, 3.0, 3.4, 3.0],
 'petal_length': [1.4, 1.4, 1.3, 1.5, ... , 5.0, 5.2, 5.4, 5.1],
 'petal_width' : [0.2, 0.2, 0.2, 0.2, ... , 1.9, 2.0, 2.3, 1.8],
 'class'       : ['Iris-setosa', ... , 'Iris-virginica']}
```

In [12]:
```
import csv
csv_reader = csv.DictReader(open('iris-write-from-docker.csv'))

iris_dict = {}

for row in csv_reader:
    for col, value in row.items():
        iris_dict.setdefault(col, []).append(value)

print(iris_dict)
```

{'sepal_length': ['5.1', '4.9', '4.7', '4.6', '5.0', '5.4', '4.6', '5.0', '4.
4', '4.9', '5.4', '4.8', '4.8', '4.3', '5.8', '5.7', '5.4', '5.1', '5.7', '5.
1', '5.4', '5.1', '4.6', '5.1', '4.8', '5.0', '5.0', '5.2', '5.2', '4.7', '4.
8', '5.4', '5.2', '5.5', '4.9', '5.0', '5.5', '4.9', '4.4', '5.1', '5.0', '4.
5', '4.4', '5.0', '5.1', '4.8', '5.1', '4.6', '5.3', '5.0', '7.0', '6.4', '6.
9', '5.5', '6.5', '5.7', '6.3', '4.9', '6.6', '5.2', '5.0', '5.9', '6.0', '6.
1', '5.6', '6.7', '5.6', '5.8', '6.2', '5.6', '5.9', '6.1', '6.3', '6.1', '6.
4', '6.6', '6.8', '6.7', '6.0', '5.7', '5.5', '5.5', '5.8', '6.0', '5.4', '6.
0', '6.7', '6.3', '5.6', '5.5', '5.5', '6.1', '5.8', '5.0', '5.6', '5.7', '5.
7', '6.2', '5.1', '5.7', '6.3', '5.8', '7.1', '6.3', '6.5', '7.6', '4.9', '7.
3', '6.7', '7.2', '6.5', '6.4', '6.8', '5.7', '5.8', '6.4', '6.5', '7.7', '7.
7', '6.0', '6.9', '5.6', '7.7', '6.3', '6.7', '7.2', '6.2', '6.1', '6.4', '7.
2', '7.4', '7.9', '6.4', '6.3', '6.1', '7.7', '6.3', '6.4', '6.0', '6.9', '6.
7', '6.9', '5.8', '6.8', '6.7', '6.7', '6.3', '6.5', '6.2', '5.9'], 'sepal_wi
dth': ['3.5', '3.0', '3.2', '3.1', '3.6', '3.9', '3.4', '3.4', '2.9', '3.1',
'3.7', '3.4', '3.0', '3.0', '4.0', '4.4', '3.9', '3.5', '3.8', '3.8', '3.4',
'3.7', '3.6', '3.3', '3.4', '3.0', '3.4', '3.5', '3.4', '3.2', '3.1', '3.4',
'4.1', '4.2', '3.1', '3.2', '3.5', '3.1', '3.0', '3.4', '3.5', '2.3', '3.2',
'3.5', '3.8', '3.0', '3.8', '3.2', '3.7', '3.3', '3.2', '3.2', '3.1', '2.3',
'2.8', '2.8', '3.3', '2.4', '2.9', '2.7', '2.0', '3.0', '2.2', '2.9', '2.9',
'3.1', '3.0', '2.7', '2.2', '2.5', '3.2', '2.8', '2.5', '2.8', '2.9', '3.0',
'2.8', '3.0', '2.9', '2.6', '2.4', '2.4', '2.7', '2.7', '3.0', '3.4', '3.1',
'2.3', '3.0', '2.5', '2.6', '3.0', '2.6', '2.3', '2.7', '3.0', '2.9', '2.9',
'2.5', '2.8', '3.3', '2.7', '3.0', '2.9', '3.0', '3.0', '2.5', '2.9', '2.5',
'3.6', '3.2', '2.7', '3.0', '2.5', '2.8', '3.2', '3.0', '3.8', '2.6', '2.2',
'3.2', '2.8', '2.8', '2.7', '3.3', '3.2', '2.8', '3.0', '2.8', '3.0', '2.8',
'3.8', '2.8', '2.8', '2.6', '3.0', '3.4', '3.1', '3.0', '3.1', '3.1', '3.1',
'2.7', '3.2', '3.3', '3.0', '2.5', '3.0', '3.4', '3.0'], 'petal_length': ['1.
4', '1.4', '1.3', '1.5', '1.4', '1.7', '1.4', '1.5', '1.4', '1.5', '1.5', '1.
6', '1.4', '1.1', '1.2', '1.5', '1.3', '1.4', '1.7', '1.5', '1.7', '1.5', '1.
0', '1.7', '1.9', '1.6', '1.6', '1.5', '1.4', '1.6', '1.6', '1.5', '1.5', '1.
4', '1.5', '1.2', '1.3', '1.5', '1.3', '1.5', '1.3', '1.3', '1.3', '1.6', '1.
9', '1.4', '1.6', '1.4', '1.5', '1.4', '4.7', '4.5', '4.9', '4.0', '4.6', '4.
5', '4.7', '3.3', '4.6', '3.9', '3.5', '4.2', '4.0', '4.7', '3.6', '4.4', '4.
5', '4.1', '4.5', '3.9', '4.8', '4.0', '4.9', '4.7', '4.3', '4.4', '4.8', '5.
0', '4.5', '3.5', '3.8', '3.7', '3.9', '5.1', '4.5', '4.5', '4.7', '4.4', '4.
1', '4.0', '4.4', '4.6', '4.0', '3.3', '4.2', '4.2', '4.2', '4.3', '3.0', '4.
1', '6.0', '5.1', '5.9', '5.6', '5.8', '6.6', '4.5', '6.3', '5.8', '6.1', '5.
1', '5.3', '5.5', '5.0', '5.1', '5.3', '5.5', '6.7', '6.9', '5.0', '5.7', '4.
9', '6.7', '4.9', '5.7', '6.0', '4.8', '4.9', '5.6', '5.8', '6.1', '6.4', '5.
6', '5.1', '5.6', '6.1', '5.6', '5.5', '4.8', '5.4', '5.6', '5.1', '5.1', '5.
9', '5.7', '5.2', '5.0', '5.2', '5.4', '5.1'], 'petal_width': ['0.2', '0.2',
'0.2', '0.2', '0.2', '0.4', '0.3', '0.2', '0.2', '0.1', '0.2', '0.2', '0.1',
'0.1', '0.2', '0.4', '0.4', '0.3', '0.3', '0.3', '0.2', '0.4', '0.2', '0.5',
'0.2', '0.2', '0.4', '0.2', '0.2', '0.2', '0.2', '0.4', '0.1', '0.2', '0.1',
'0.2', '0.2', '0.1', '0.2', '0.2', '0.3', '0.3', '0.2', '0.6', '0.4', '0.3',
'0.2', '0.2', '0.2', '0.2', '1.4', '1.5', '1.5', '1.3', '1.5', '1.3', '1.6',
'1.0', '1.3', '1.4', '1.0', '1.5', '1.0', '1.4', '1.3', '1.4', '1.5', '1.0',
'1.5', '1.1', '1.8', '1.3', '1.5', '1.2', '1.3', '1.4', '1.4', '1.7', '1.5',
'1.0', '1.1', '1.0', '1.2', '1.6', '1.5', '1.6', '1.5', '1.3', '1.3', '1.3',
'1.2', '1.4', '1.2', '1.0', '1.3', '1.2', '1.3', '1.3', '1.1', '1.3', '2.5',
'1.9', '2.1', '1.8', '2.2', '2.1', '1.7', '1.8', '1.8', '2.5', '2.0', '1.9',
'2.1', '2.0', '2.4', '2.3', '1.8', '2.2', '2.3', '1.5', '2.3', '2.0', '2.0',
'1.8', '2.1', '1.8', '1.8', '1.8', '2.1', '1.6', '1.9', '2.0', '2.2', '1.5',
'1.4', '2.3', '2.4', '1.8', '1.8', '2.1', '2.4', '2.3', '1.9', '2.3', '2.5',
'2.3', '1.9', '2.0', '2.3', '1.8'], 'class': ['Iris-setosa', 'Iris-setosa',
'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'I

ris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iri
s-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-se
tosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-seto
sa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setos
a', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setos
a', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setos
a', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setos
a', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris
-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-
versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-v
ersicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-ve
rsicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-ver
sicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-vers
icolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versi
color', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versic
olor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versico
lor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicol
or', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolo
r', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolo
r', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-virginic
a', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris
-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virg
inica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginic
a', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris
-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virg
inica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginic
a', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris
-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virg
inica', 'Iris-virginica', 'Iris-virginica']}

b) (10 Points) Write a script that reads the iris data into a dictionary where the flower names are the `Keys` and all rows belonging to a flower are the `Values` .

The output of running this script would be similar to the following:

```
Iris-setosa
     [5.1, 3.5, 1.4, 0.2]
     [4.9, 3.0, 1.4, 0.2]
     [4.7, 3.2, 1.3, 0.2]
     [..., ..., ..., ...]
   [4.6, 3.2, 1.4, 0.2]
     [5.3, 3.7, 1.5, 0.2]
     [5.0, 3.3, 1.4, 0.2]
Iris-versicolor
     [7.0, 3.2, 4.7, 1.4]
     [6.4, 3.2, 4.5, 1.5]
     [6.9, 3.1, 4.9, 1.5]
     [..., ..., ..., ...]
   [6.2, 2.9, 4.3, 1.3]
     [5.1, 2.5, 3.0, 1.1]
     [5.7, 2.8, 4.1, 1.3]
Iris-virginica
     [6.3, 3.3, 6.0, 2.5]
     [5.8, 2.7, 5.1, 1.9]
     [7.1, 3.0, 5.9, 2.1]
     [..., ..., ..., ...]
   [6.5, 3.0, 5.2, 2.0]
     [6.2, 3.4, 5.4, 2.3]
     [5.9, 3.0, 5.1, 1.8]
```

In [37]:
```
import csv
csv_reader = csv.DictReader(open('iris-write-from-docker.csv'))

flowers = {}

for row in csv_reader:
    row_data = [row['sepal_length'], row['sepal_width'], row['petal_length'],
row['petal_width']]
    flowers.setdefault(row['class'],[]).append(row_data)

print(flowers)
```

{'Iris-setosa': [['5.1', '3.5', '1.4', '0.2'], ['4.9', '3.0', '1.4', '0.2'],
['4.7', '3.2', '1.3', '0.2'], ['4.6', '3.1', '1.5', '0.2'], ['5.0', '3.6',
'1.4', '0.2'], ['5.4', '3.9', '1.7', '0.4'], ['4.6', '3.4', '1.4', '0.3'],
['5.0', '3.4', '1.5', '0.2'], ['4.4', '2.9', '1.4', '0.2'], ['4.9', '3.1',
'1.5', '0.1'], ['5.4', '3.7', '1.5', '0.2'], ['4.8', '3.4', '1.6', '0.2'],
['4.8', '3.0', '1.4', '0.1'], ['4.3', '3.0', '1.1', '0.1'], ['5.8', '4.0',
'1.2', '0.2'], ['5.7', '4.4', '1.5', '0.4'], ['5.4', '3.9', '1.3', '0.4'],
['5.1', '3.5', '1.4', '0.3'], ['5.7', '3.8', '1.7', '0.3'], ['5.1', '3.8',
'1.5', '0.3'], ['5.4', '3.4', '1.7', '0.2'], ['5.1', '3.7', '1.5', '0.4'],
['4.6', '3.6', '1.0', '0.2'], ['5.1', '3.3', '1.7', '0.5'], ['4.8', '3.4',
'1.9', '0.2'], ['5.0', '3.0', '1.6', '0.2'], ['5.0', '3.4', '1.6', '0.4'],
['5.2', '3.5', '1.5', '0.2'], ['5.2', '3.4', '1.4', '0.2'], ['4.7', '3.2',
'1.6', '0.2'], ['4.8', '3.1', '1.6', '0.2'], ['5.4', '3.4', '1.5', '0.4'],
['5.2', '4.1', '1.5', '0.1'], ['5.5', '4.2', '1.4', '0.2'], ['4.9', '3.1',
'1.5', '0.1'], ['5.0', '3.2', '1.2', '0.2'], ['5.5', '3.5', '1.3', '0.2'],
['4.9', '3.1', '1.5', '0.1'], ['4.4', '3.0', '1.3', '0.2'], ['5.1', '3.4',
'1.5', '0.2'], ['5.0', '3.5', '1.3', '0.3'], ['4.5', '2.3', '1.3', '0.3'],
['4.4', '3.2', '1.3', '0.2'], ['5.0', '3.5', '1.6', '0.6'], ['5.1', '3.8',
'1.9', '0.4'], ['4.8', '3.0', '1.4', '0.3'], ['5.1', '3.8', '1.6', '0.2'],
['4.6', '3.2', '1.4', '0.2'], ['5.3', '3.7', '1.5', '0.2'], ['5.0', '3.3',
'1.4', '0.2']], 'Iris-versicolor': [['7.0', '3.2', '4.7', '1.4'], ['6.4', '3.
2', '4.5', '1.5'], ['6.9', '3.1', '4.9', '1.5'], ['5.5', '2.3', '4.0', '1.
3'], ['6.5', '2.8', '4.6', '1.5'], ['5.7', '2.8', '4.5', '1.3'], ['6.3', '3.
3', '4.7', '1.6'], ['4.9', '2.4', '3.3', '1.0'], ['6.6', '2.9', '4.6', '1.
3'], ['5.2', '2.7', '3.9', '1.4'], ['5.0', '2.0', '3.5', '1.0'], ['5.9', '3.
0', '4.2', '1.5'], ['6.0', '2.2', '4.0', '1.0'], ['6.1', '2.9', '4.7', '1.
4'], ['5.6', '2.9', '3.6', '1.3'], ['6.7', '3.1', '4.4', '1.4'], ['5.6', '3.
0', '4.5', '1.5'], ['5.8', '2.7', '4.1', '1.0'], ['6.2', '2.2', '4.5', '1.
5'], ['5.6', '2.5', '3.9', '1.1'], ['5.9', '3.2', '4.8', '1.8'], ['6.1', '2.
8', '4.0', '1.3'], ['6.3', '2.5', '4.9', '1.5'], ['6.1', '2.8', '4.7', '1.
2'], ['6.4', '2.9', '4.3', '1.3'], ['6.6', '3.0', '4.4', '1.4'], ['6.8', '2.
8', '4.8', '1.4'], ['6.7', '3.0', '5.0', '1.7'], ['6.0', '2.9', '4.5', '1.
5'], ['5.7', '2.6', '3.5', '1.0'], ['5.5', '2.4', '3.8', '1.1'], ['5.5', '2.
4', '3.7', '1.0'], ['5.8', '2.7', '3.9', '1.2'], ['6.0', '2.7', '5.1', '1.
6'], ['5.4', '3.0', '4.5', '1.5'], ['6.0', '3.4', '4.5', '1.6'], ['6.7', '3.
1', '4.7', '1.5'], ['6.3', '2.3', '4.4', '1.3'], ['5.6', '3.0', '4.1', '1.
3'], ['5.5', '2.5', '4.0', '1.3'], ['5.5', '2.6', '4.4', '1.2'], ['6.1', '3.
0', '4.6', '1.4'], ['5.8', '2.6', '4.0', '1.2'], ['5.0', '2.3', '3.3', '1.
0'], ['5.6', '2.7', '4.2', '1.3'], ['5.7', '3.0', '4.2', '1.2'], ['5.7', '2.
9', '4.2', '1.3'], ['6.2', '2.9', '4.3', '1.3'], ['5.1', '2.5', '3.0', '1.
1'], ['5.7', '2.8', '4.1', '1.3']], 'Iris-virginica': [['6.3', '3.3', '6.0',
'2.5'], ['5.8', '2.7', '5.1', '1.9'], ['7.1', '3.0', '5.9', '2.1'], ['6.3',
'2.9', '5.6', '1.8'], ['6.5', '3.0', '5.8', '2.2'], ['7.6', '3.0', '6.6', '2.
1'], ['4.9', '2.5', '4.5', '1.7'], ['7.3', '2.9', '6.3', '1.8'], ['6.7', '2.
5', '5.8', '1.8'], ['7.2', '3.6', '6.1', '2.5'], ['6.5', '3.2', '5.1', '2.
0'], ['6.4', '2.7', '5.3', '1.9'], ['6.8', '3.0', '5.5', '2.1'], ['5.7', '2.
5', '5.0', '2.0'], ['5.8', '2.8', '5.1', '2.4'], ['6.4', '3.2', '5.3', '2.
3'], ['6.5', '3.0', '5.5', '1.8'], ['7.7', '3.8', '6.7', '2.2'], ['7.7', '2.
6', '6.9', '2.3'], ['6.0', '2.2', '5.0', '1.5'], ['6.9', '3.2', '5.7', '2.
3'], ['5.6', '2.8', '4.9', '2.0'], ['7.7', '2.8', '6.7', '2.0'], ['6.3', '2.
7', '4.9', '1.8'], ['6.7', '3.3', '5.7', '2.1'], ['7.2', '3.2', '6.0', '1.
8'], ['6.2', '2.8', '4.8', '1.8'], ['6.1', '3.0', '4.9', '1.8'], ['6.4', '2.
8', '5.6', '2.1'], ['7.2', '3.0', '5.8', '1.6'], ['7.4', '2.8', '6.1', '1.
9'], ['7.9', '3.8', '6.4', '2.0'], ['6.4', '2.8', '5.6', '2.2'], ['6.3', '2.
8', '5.1', '1.5'], ['6.1', '2.6', '5.6', '1.4'], ['7.7', '3.0', '6.1', '2.
3'], ['6.3', '3.4', '5.6', '2.4'], ['6.4', '3.1', '5.5', '1.8'], ['6.0', '3.
0', '4.8', '1.8'], ['6.9', '3.1', '5.4', '2.1'], ['6.7', '3.1', '5.6', '2.

```
4'], ['6.9', '3.1', '5.1', '2.3'], ['5.8', '2.7', '5.1', '1.9'], ['6.8', '3.
2', '5.9', '2.3'], ['6.7', '3.3', '5.7', '2.5'], ['6.7', '3.0', '5.2', '2.
3'], ['6.3', '2.5', '5.0', '1.9'], ['6.5', '3.0', '5.2', '2.0'], ['6.2', '3.
4', '5.4', '2.3'], ['5.9', '3.0', '5.1', '1.8']]}
```

Question 2 [25 pts]:

A Graph is a non-linear data structure consisting of vertices (nodes) and edges (links) connecting the vertices.

In this question, you are expected to create a Graph class with a set of respective functions and methods. Note: While answering this question, you are **not allowed** to use Python's `LinkedDirectedGraph` library.

a) [10 pts] Define and implement a class named 'UndirectedGraph'. This class constructs the necessary attributes that hold the following information about the instantiated graphs:

- The `vertices` attribute, which is a set of nodes that may or may not be connected through edges.
- The `addVertex(vertexName)` method, which accepts one parameter as the vertex name and adds it to the graph.
- The `getVertex(vertexName)` method that returns the name of the vertex if it exists or None if it does not exist in the graph.
- The `removeVertex(vertexName)` method deletes the vertex name if there is one with the same name in the graph
- The `listVertices()` method displays the vertices names included in the graph.

Here is an example of creating an UndirectedGraph() object, adding three vertices: `A`, `B` and `C`, removing vertex `B`, then listing the existing vertices.

```
myGraph = UndirectedGraph()
myGraph.addVertex('A')
myGraph.addVertex('B')
myGraph.addVertex('C')
myGraph.removeVertex('B')
myGraph.listVertices() # returns A, C
```

In [38]:
```
class UndirectedGraph:
    def __init__(self):
        self.vertices = list()
    def addVertex(self, vertexName):
        self.vertices.append(vertexName)
        return(self.vertices)
    def getVertex(self, vertexName):
        if vertexName in self.vertices:
            return(vertexName)
        else:
            print("Error: Vertex not in list.")
    def removeVertex(self, vertexName):
        if vertexName in self.vertices:
            self.vertices.remove(vertexName)
            return self.vertices
    def listVertices(self):
        for i in self.vertices:
            print(i)

myGraph = UndirectedGraph()
myGraph.addVertex('A')
myGraph.addVertex('B')
myGraph.addVertex('C')
myGraph.removeVertex('B')
myGraph.listVertices() # returns A, C
```

```
A
C
```

b) [10 pts] Enhance the `UndirectedGraph` class that you defined in Q2.a by adding the edges information that connects the existing vertices. Here are the methods you need to define and implement.

- The `addEdge(vertex1, vertex2)` method to add an edge connecting vertex1 to vertex2.
- The `checkEdge(vertex1, vertex2)` method returns `True` if an edge exists between `vertex1` and `vertex2` , otherwise it returns `False` . Note that the edges are not directional.
- The `removeEdge(vertex1, vertex2)` method removes the edge between the respective vertices: vertex1 and vertex2.
- The `listEdges(ofVertex)` method displays the edges of the respective vertex.
- Update the `removeVertex(vertexName)` method to ensure removing the associated edges before removing the vertex.

Here is an example that creates a graph object with three vertices: A, B and C, then connects A with B and B with C, then displays the edges of vertex B.

```
myGraph = UndirectedGraph()
myGraph.addVertex('A')
myGraph.addVertex('B')
myGraph.addVertex('C')
myGraph.addEdge('A','B')
myGraph.addEdge('B','C')
myGraph.listEdges('B') # returns 'A-B , B-C'
```

In [39]:
```
class UndirectedGraph:
    def __init__(self):
        self.vertices = list()
        self.edges = list()
    def addVertex(self, vertexName):
        self.vertices.append(vertexName)
        return(self.vertices)
    def getVertex(self, vertexName):
        if vertexName in self.vertices:
            return(vertexName)
        else:
            print("Error: Vertex not in list.")
    def removeVertex(self, vertexName):
        if vertexName in self.vertices:
            for h in self.edges:
                if vertexName in h:
                    self.edges.remove(h)
            self.vertices.remove(vertexName)
            return self.vertices
    def listVertices(self):
        for i in self.vertices:
            return(i)
    def addEdge(self, vertex1, vertex2):
        if vertex1 in self.vertices and vertex2 in self.vertices:
            new_edge = [vertex1, vertex2]
            swap_newedge = [vertex2, vertex1]
            if new_edge in self.edges or swap_newedge in self.edges:
                print("Error: Edge already in list.")
            else:
                self.edges.append(new_edge)
        else:
            print("Error: Vertex not in list.")
    def checkEdge(self, vertex1, vertex2):
        find_edge = [vertex1, vertex2]
        swapfind_edge = [vertex2, vertex1]
        if find_edge in self.edges or swapfind_edge in self.edges:
            return True
        else:
            return False
    def removeEdge(self, vertex1, vertex2):
        find_edge = [vertex1, vertex2]
        swapfind_edge = [vertex2, vertex1]
        if find_edge in self.edges or swapfind_edge in self.edges:
            self.edges.remove(find_edge)
            return self.edges
        elif swapfind_edge in self.edges:
            self.edges.remove(swapfind_edge)
            return self.edges
        else:
            print("Error: Edge not in list.")
    def listEdges(self, vertexName):
        edge = ""
        for j in self.edges:
            if vertexName in j:
                edge += '-'.join(j) + " "
        return edge
```

```
myGraph = UndirectedGraph()
myGraph.addVertex('A')
myGraph.addVertex('B')
myGraph.addVertex('C')
myGraph.addEdge('A','B')
myGraph.addEdge('B','C')
myGraph.listEdges('B') # returns 'A-B , B-C'
```

Out[39]: 'A-B B-C '

c) [5 pts] Use your answer in Q2.b. to create a graph object with the following vertices and edges:

- Vertices = 'A', 'B', 'C', 'D', 'E'
- Edges = 'A-B', 'A-C', 'A-D', 'D-B', 'D-C', 'C-B', 'E-D'

Then, run the following commands, assuming the name of the graph object is `iGraph`):

```
iGraph.getVertex('K')
iGraph.getVertex('A')
print("Vertices of the graph: " + iGraph.listVertices())
print("Edges of Vertex D: \n" + iGraph.listEdges("D"))
graph.removeVertex('D')
print("Vertices of the graph: " +iGraph.listVertices())
print("Edges of Vertex D: \n" + iGraph.listEdges('D'))
```

In [40]:
```
iGraph = UndirectedGraph()
iGraph.addVertex('A')
iGraph.addVertex('B')
iGraph.addVertex('C')
iGraph.addVertex('D')
iGraph.addVertex('E')
iGraph.addEdge('A', 'B')
iGraph.addEdge('A', 'C')
iGraph.addEdge('A', 'D')
iGraph.addEdge('D', 'B')
iGraph.addEdge('D', 'C')
iGraph.addEdge('C', 'B')
iGraph.addEdge('E', 'D')

iGraph.getVertex('K')
iGraph.getVertex('A')
print("Vertices of the graph: " + iGraph.listVertices())
print("Edges of Vertex D: \n" + iGraph.listEdges("D"))
iGraph.removeVertex('D')
print("Vertices of the graph: " +iGraph.listVertices())
print("Edges of Vertex D: \n" + iGraph.listEdges('D'))
```

```
Error: Vertex not in list.
Vertices of the graph: A
Edges of Vertex D:
A-D D-B D-C E-D
Vertices of the graph: A
Edges of Vertex D:
D-B
```

Question 3 [30 pts]:

Define a class named **Stat** that can be instantiated with a list of integers.

Note: Only Python built-in functions (https://docs.python.org/3.7/library/functions.html) are allowed to be used in this question. Do not use a third-party library for any part of the code. Importing any third-party library will result in marks deduction.**

a) [10 pts] Instantiate the Stat class with a constructor that accepts a list of integers. Each instantiated object should have two attributes: length and items. The length returns the number of elements of the object, and the items attribute returns the content of the list.

For example, if objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10]) then the length should be 14 and the items [5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10]

In [43]:
```
class Stat:
    def __init__(self, list):
        self.list = list
    def __len__(self):
        return len(self.list)
    def __iter__(self):
        return iter(self.list)

objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])
print(len(objStat))
for i in objStat:
    print(i, end=' ')
```

```
14
5 3 7 3 6 4 5 8 2 9 3 4 20 -10
```

b) (10 Points) Define and implement a mutator method named **sortStat** as a member of the **Stat** class. The **sortStat** method sorts the `items` attribute elements in ascending order. Use the insertion sort algorithm for this task.

For example, if `objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])` then after invoking the `sortStat` method, the `items` attribute will be `[-10, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 20]`

In [44]:
```
class Stat:
    def __init__(self, list):
        self.list = list
    def __len__(self):
        return len(self.list)
    def __iter__(self):
        return iter(self.list)
    def sortStat(self):
        i = 1
        while i < len(self.list):
            itemToInsert = self.list[i]
            j = i - 1
            while j >= 0:
                if itemToInsert < self.list[j]:
                    self.list[j + 1] = self.list[j]
                    j -= 1
                else:
                    break
            self.list[j + 1] = itemToInsert
            i += 1
        return self.list

objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])
objStat.sortStat()
```

Out[44]:
```
[-10, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 20]
```

c) (10 Points) Define and implement an accessor method named *belongStat* method as a member of the **Stat** class. The *belongStat* method accepts an integer and returns True if the number exists in the `items` list; otherwise, the method should return False. Use the binary search algorithm for this task.

For example, if `objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])` and the user invoked the the `belongStat()` method with `8`, then the method will return `True`.

In [45]:
```
class Stat:
    def __init__(self, list):
        self.list = list
    def __len__(self):
        return len(self.list)
    def __iter__(self):
        return iter(self.list)
    def sortStat(self):
        i = 1
        while i < len(self.list):
            itemToInsert = self.list[i]
            j = i - 1
            while j >= 0:
                if itemToInsert < self.list[j]:
                    self.list[j + 1] = self.list[j]
                    j -= 1
                else:
                    break
            self.list[j + 1] = itemToInsert
            i += 1
        return self.list
    def belongStat(self, target):
        left = 0
        right = len(self.sortStat()) - 1
        while left <= right:
            mid = (left + right) // 2
            if target == self.sortStat()[mid]:
                return True
            elif target < self.sortStat()[mid]:
                right = mid - 1
            else:
                left = mid + 1

objStat = Stat([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])
objStat.belongStat(8)
```

Out[45]: True

Question 4 [25 pts]:

Assume the following class implements the STACK abstract data type (ADT) using the array ADT.

```
class aStack(iArray):
    def __init__(self, capacity = 5):
        self._items = iArray(capacity)
        self._top = -1
        self._size = 0
    def push(self, newItem):
        self._top += 1
        self._size += 1
        self._items[self._top] = newItem
    def pop(self):
        oldItem = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        return oldItem
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result
```

a) [5 pts] Emulate the stack behavior using the Python list data structure rather than the Arrary ADT.

In [46]:
```
class anotherStack():
    def __init__(self, iList):
        self._items = iList
        self._top = -1
    def push(self, newItem):
        self._top += 1
        self._items.insert(self._top, newItem)
    def pop(self):
        to_pop = self._items[self._top]
        self._items.pop()
        return to_pop
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return len(self._items)
    def __str__(self):
        result = ' '
        for i in self._items:
            result += str(i) + ' '
        return result

try_it = anotherStack([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])
try_it.pop()
print(try_it)
try_it.push(25)
print(try_it)
try_it.peek()
print(len(try_it))
print(str(try_it).split())
```

```
 5 3 7 3 6 4 5 8 2 9 3 4 20
 25 5 3 7 3 6 4 5 8 2 9 3 4 20
14
['25', '5', '3', '7', '3', '6', '4', '5', '8', '2', '9', '3', '4', '20']
```

b) [10 pts] Redefine the Stack class methods to push and pop two items rather than one item at a time.

For example, if the stack includes numbers from one to ten: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` , then invoking the `pop()` method twice will remove the last four elements and modify the stack elements to be: `[1, 2, 3, 4, 5, 6]`

In [47]:
```
class anotherStack():
    def __init__(self, iList):
        self._items = iList
        self._top = -1
    def push(self, newItem):
        for i in range(2):
            self._top += 1
            self._items.insert(self._top, newItem)
    def pop(self):
        to_pop = []
        for j in range(2):
            to_pop.append(self._items[self._top])
            self._items.pop()
        return to_pop
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return len(self._items)
    def __str__(self):
        result = ' '
        for i in self._items:
            result += str(i) + ' '
        return result

try_it = anotherStack([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
try_it.pop()
try_it.pop()
print(try_it)
```

```
 1 2 3 4 5 6
```

c) [10 pts] Define and implement a function that reverses the items of a given stack using only the methods defined in the Stack class.

For example, if the stack includes the elements from one to ten: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, then invoking the new function will modify the stack elements to be from ten to one: `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`

In [48]:
```
class anotherStack():
    def __init__(self, iList):
        self._items = iList
        self._top = -1
    def push(self, newItem):
        self._top += 1
        self._items.insert(self._top, newItem)
    def pop(self):
        to_pop = self._items[-1]
        self._items.pop()
        return to_pop
    def peek(self):
        return self._items[-1]
    def __len__(self):
        return len(self._items)
    def __str__(self):
        result = ' '
        for i in self._items:
            result += str(i) + ' '
        return result

try_it = anotherStack([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

def reverseStack(try_it):
    temp_stack = anotherStack([])
    while(len(try_it) > 0):
        temp = try_it.peek()
        try_it.pop()
        while(len(temp_stack) > 0 and int(temp_stack.peek()) < int(temp)):
            try_it.push(temp_stack.peek())
            temp_stack.pop()
        temp_stack.push(temp)
    return(temp_stack)

print(reverseStack(try_it))
```

```
 10 9 8 7 6 5 4 3 2 1
```

This is the end of assignment 3