Writing CSV files in Python

CSV (Comma Separated Values) is a simple file format used to store tabular data, such as a spreadsheet or database. CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

Python provides an in-built module called `csv` to work with CSV files. There are various classes provided by this module for writing to CSV:

- Using csv.writer class
- Using csv.DictWriter class

Using csv.writer class

`csv.writer` class is used to insert data to the CSV file. This class returns a writer object which is responsible for converting the user's data into a delimited string. A csvfile object should be opened with `newline=''` otherwise newline characters inside the quoted fields will not be interpreted correctly.

> ***Syntax:*** *csv.writer(csvfile, dialect='excel', **fmtparams)*
>
> ***Parameters:***
> ***csvfile:*** *A file object with write() method.*
> ***dialect (optional):*** *Name of the dialect to be used.*
> ***fmtparams (optional):*** *Formatting parameters that will overwrite those specified in the dialect.*

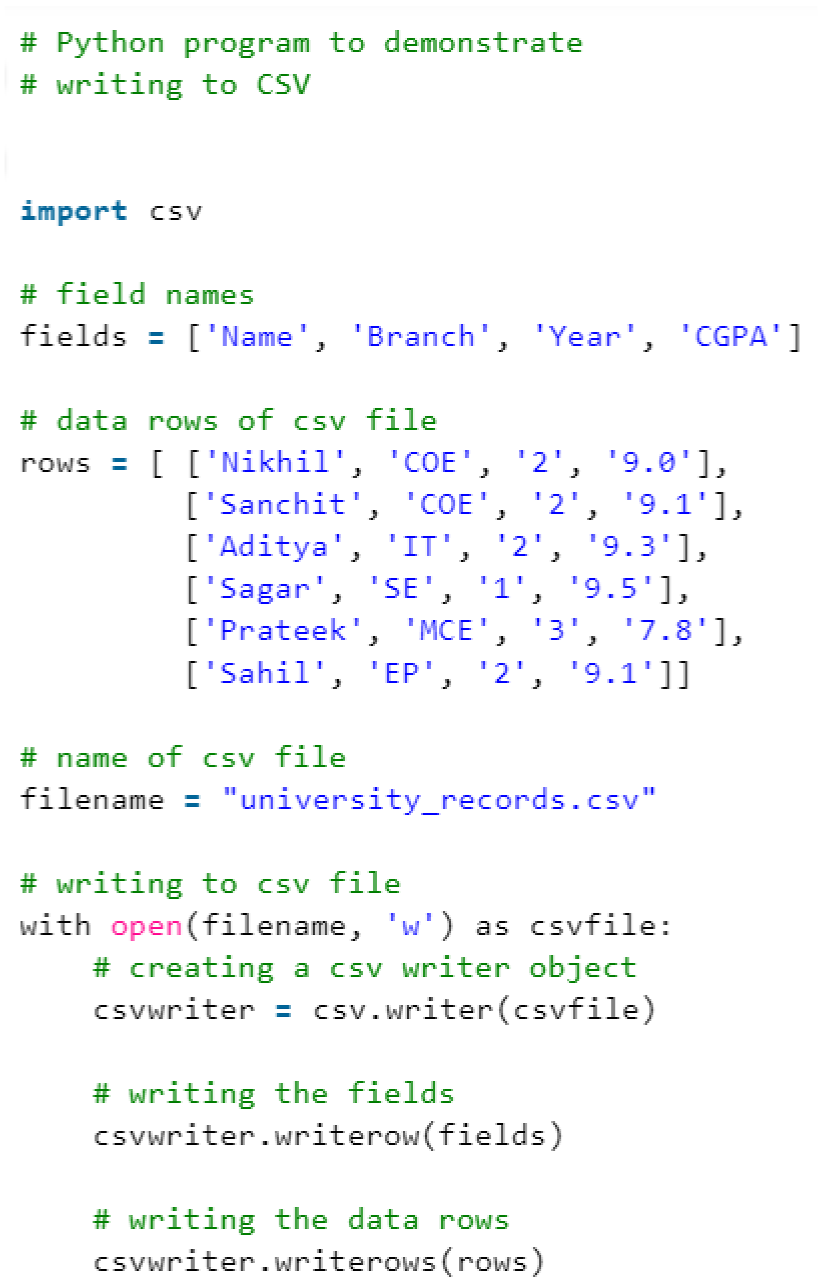
`csv.writer` class provides two methods for writing to CSV. They are `writerow()` and `writerows()`.

- **writerow():** This method writes a single row at a time. Field row can be written using this method.
  Syntax:

  ```
  writerow(fields)
  ```

- **writerows():** This method is used to write multiple rows at a time. This can be used to write rows list.
  Syntax:

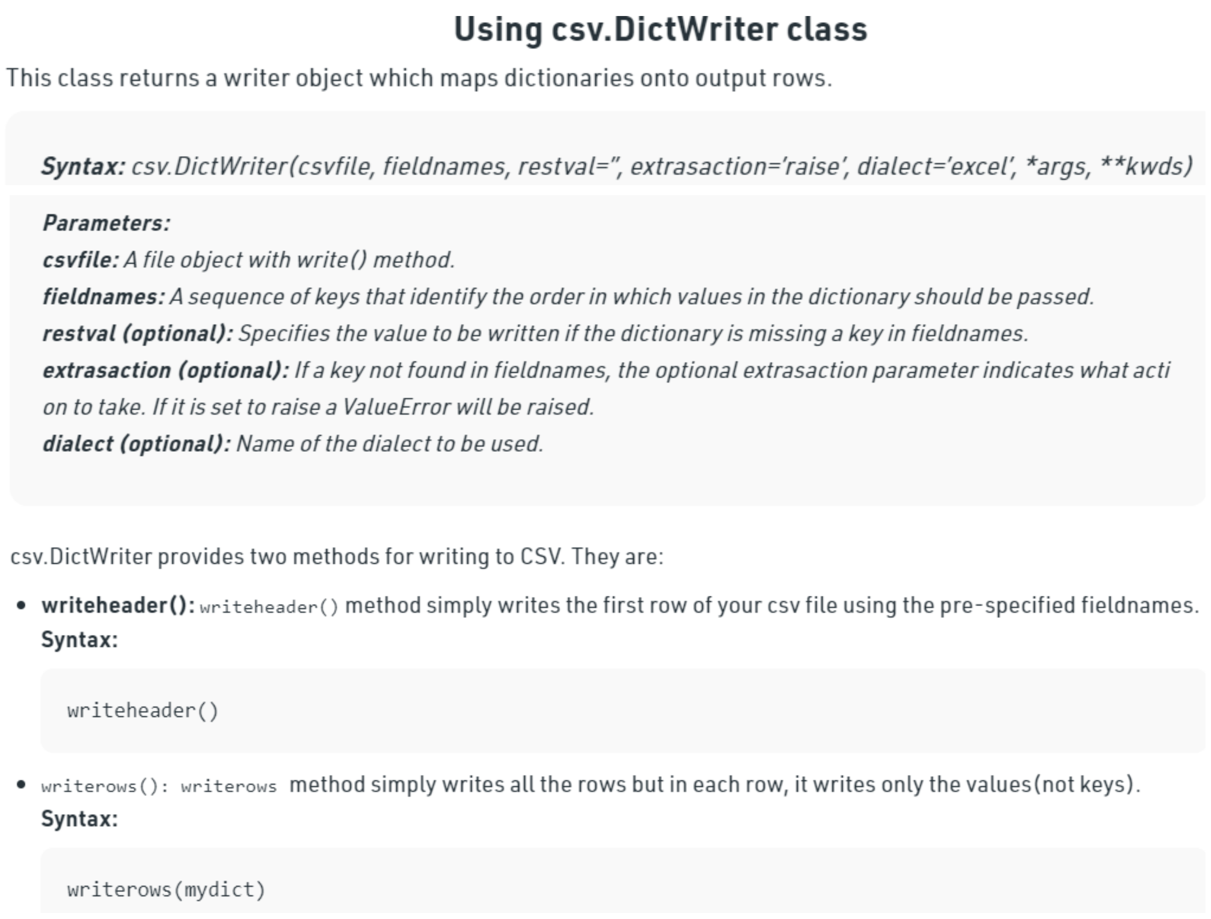
  ```
  Writing CSV files in Python
  writerows(rows)
  ```



```python
# Python program to demonstrate
# writing to CSV


import csv

# field names
fields = ['Name', 'Branch', 'Year', 'CGPA']

# data rows of csv file
rows = [ ['Nikhil', 'COE', '2', '9.0'],
         ['Sanchit', 'COE', '2', '9.1'],
         ['Aditya', 'IT', '2', '9.3'],
         ['Sagar', 'SE', '1', '9.5'],
         ['Prateek', 'MCE', '3', '7.8'],
         ['Sahil', 'EP', '2', '9.1']]

# name of csv file
filename = "university_records.csv"

# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the fields
    csvwriter.writerow(fields)

    # writing the data rows
    csvwriter.writerows(rows)
```

```python
import csv

# field names
fields = ['Name', 'Branch', 'Year', 'CGPA']

# data rows of csv file
rows = [ ['Nikhil', 'COE', '2', '9.0'],
         ['Sanchit', 'COE', '2', '9.1'],
         ['Aditya', 'IT', '2', '9.3'],
         ['Sagar', 'SE', '1', '9.5'],
         ['Prateek', 'MCE', '3', '7.8'],
         ['Sahil', 'EP', '2', '9.1']]

# name of csv file
filename = "university_records.csv"

# writing to csv file
```

```
with open(filename, 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the fields
    csvwriter.writerow(fields)

    # writing the data rows
    csvwriter.writerows(rows)
```



Using csv.DictWriter class

This class returns a writer object which maps dictionaries onto output rows.

Syntax: *csv.DictWriter(csvfile, fieldnames, restval=”, extrasaction='raise', dialect='excel', *args, **kwds)*

Parameters:
csvfile: *A file object with write() method.*
fieldnames: *A sequence of keys that identify the order in which values in the dictionary should be passed.*
restval (optional): *Specifies the value to be written if the dictionary is missing a key in fieldnames.*
extrasaction (optional): *If a key not found in fieldnames, the optional extrasaction parameter indicates what acti on to take. If it is set to raise a ValueError will be raised.*
dialect (optional): *Name of the dialect to be used.*

csv.DictWriter provides two methods for writing to CSV. They are:

- **writeheader():** `writeheader()` method simply writes the first row of your csv file using the pre-specified fieldnames.
  Syntax:

  ```
  writeheader()
  ```

- `writerows(): writerows` method simply writes all the rows but in each row, it writes only the values(not keys).
  Syntax:

  ```
  writerows(mydict)
  ```

```
# importing the csv module
import csv

# my data rows as dictionary objects
mydict =[{'branch': 'COE', 'cgpa': '9.0', 'name': 'Nikhil', 'year': '2'},
         {'branch': 'COE', 'cgpa': '9.1', 'name': 'Sanchit', 'year': '2'},
         {'branch': 'IT', 'cgpa': '9.3', 'name': 'Aditya', 'year': '2'},
         {'branch': 'SE', 'cgpa': '9.5', 'name': 'Sagar', 'year': '1'},
         {'branch': 'MCE', 'cgpa': '7.8', 'name': 'Prateek', 'year': '3'},
         {'branch': 'EP', 'cgpa': '9.1', 'name': 'Sahil', 'year': '2'}]

# field names
fields = ['name', 'branch', 'year', 'cgpa']

# name of csv file
filename = "university_records.csv"

# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv dict writer object
    writer = csv.DictWriter(csvfile, fieldnames = fields)

    # writing headers (field names)
    writer.writeheader()

    # writing data rows
    writer.writerows(mydict)
```

https://www.geeksforgeeks.org/how-to-create-multiple-csv-files-from-existing-csv-file-using-pandas/?ref=rp

Python: Read a CSV file line by line with or without header

Suppose we have a csv file ***students.csv*** and its contents are:

<u>Id,Name,Course,City,Session</u>

21,Mark,Python,London,Morning

22,John,Python,Tokyo,Evening

23,Sam,Python,Paris,Morning

32,Shaun,Java,Tokyo,Morning

We want to read all the rows of this csv file line by line and process each line at a time.

Also note that, here we don't want to read all lines into a list of lists and then iterate over it, because that will not be an efficient solution for large csv file i.e. file with size in GBs. We are looking for solutions where we read & process only one line at a time while iterating through all rows of csv, so that minimum memory is utilized.

Let's see how to do this,

Python has a csv module, which provides two different classes to read the contents of a csv file i.e. *csv.reader* and *csv.DictReader*. Let's discuss & use them one by one to read a csv file line by line,

Read a CSV file line by line using csv.reader

With csv module's reader class object we can iterate over the lines of a csv file as a list of values, where each value in the list is a cell value. Let's understand with an example,

```python
from csv import reader

# open file in read mode
with open('students.csv', 'r') as read_obj:
    # pass the file object to reader() to get the reader object
    csv_reader = reader(read_obj)
    # Iterate over each row in the csv using reader object
    for row in csv_reader:
        # row variable is a list that represents a row in csv
        print(row)
```

Output:

```
['Id', 'Name', 'Course', 'City', 'Session']
['21', 'Mark', 'Python', 'London', 'Morning']
['22', 'John', 'Python', 'Tokyo', 'Evening']
['23', 'Sam', 'Python', 'Paris', 'Morning']
['32', 'Shaun', 'Java', 'Tokyo', 'Morning']
```

It iterates over all the rows of *students.csv* file. For each row it fetched the contents of that row as a list and printed that list.

How did it work ?

It performed the following steps,

1. Open the file 'students.csv' in read mode and create a file object.
2. Create a reader object (iterator) by passing file object in csv.reader() function.
3. Now once we have this reader object, which is an iterator, then use this iterator with for loop to read individual rows of the csv as list of values. Where each value in the list represents an individual cell.

This way only one line will be in memory at a time while iterating through csv file, which makes it a memory efficient solution.

Read csv file without header

In the previous example we iterated through all the rows of csv file including header. But suppose we want to skip the header and iterate over the remaining rows of csv file.
Let's see how to do that,

```
from csv import reader
# skip first line i.e. read header first and then iterate over each row od csv
#as a list
with open('students.csv', 'r') as read_obj:
csv_reader = reader(read_obj)
header = next(csv_reader)
# Check file as empty
if header != None:
# Iterate over each row after the header in the csv
for row in csv_reader:
# row variable is a list that represents a row in csv
print(row)
```

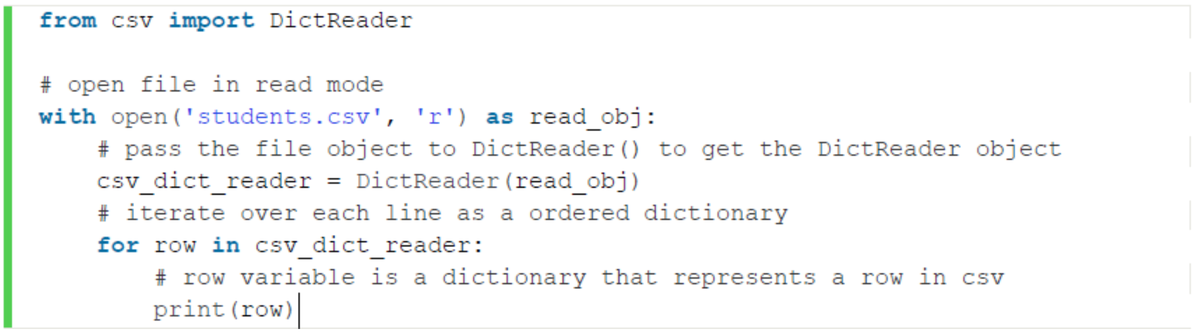
It skipped the header row of csv file and iterate over all the remaining rows of students.csv file. For each row it fetched the contents of that row as a list and printed that list. In initially saved the header row in a separate variable and printed that in end.

How did it work ?

As reader() function returns an iterator object, which we can use with Python for loop to iterate over the rows. But in the above example we called the next() function on this iterator object initially, which returned the first row of csv. After that we used the iterator object with for loop to iterate over remaining rows of the csv file.

Read csv file line by line using csv module DictReader object

With csv module's DictReader class object we can iterate over the lines of a csv file as a dictionary i.e.
for each row a dictionary is returned, which contains the pair of column names and cell values for that row.
Let's understand with an example,



```python
from csv import DictReader

# open file in read mode
with open('students.csv', 'r') as read_obj:
    # pass the file object to DictReader() to get the DictReader object
    csv_dict_reader = DictReader(read_obj)
    # iterate over each line as a ordered dictionary
    for row in csv_dict_reader:
        # row variable is a dictionary that represents a row in csv
        print(row)
```

How did it work ?

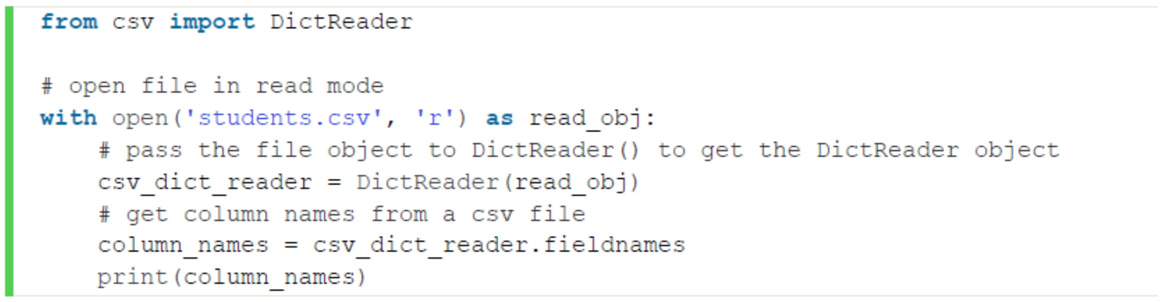
It performed the following steps,

1. Open the file 'students.csv' in read mode and create a file object.
2. Create a DictReader object (iterator) by passing file object in csv.DictReader().
3. Now once we have this DictReader object, which is an iterator. Use this iterator object with for loop to read individual rows of the csv as a dictionary. Where each pair in this dictionary represents contains the column name & column value for that row.

It is a memory efficient solution, because at a time only one line is in memory.

Get column names from header in csv file

DictReader class has a member function that returns the column names of the csv file as list.
let's see how to use it,



```
from csv import DictReader

# open file in read mode
with open('students.csv', 'r') as read_obj:
    # pass the file object to DictReader() to get the DictReader object
    csv_dict_reader = DictReader(read_obj)
    # get column names from a csv file
    column_names = csv_dict_reader.fieldnames
    print(column_names)
```

```
from csv import DictReader
# open file in read mode

# pass the file object to DictReader() to get the DictReader object
with open('c:\Iris.csv', 'r') as read_obj:
    csv_dict_reader = DictReader(read_obj)
    column_names = csv_dict_reader.fieldnames    # get column names from a csv file
    print(column_names)
```

Read specific columns from a csv file while iterating line by line

Read specific columns (by column name) in a csv file while iterating row by row

Iterate over all the rows of students.csv file line by line, but print only two columns of for each row,

```
from csv import DictReader
# open file in read mode
# pass the file object to DictReader() to get the DictReader object
with open('c:\Iris.csv', 'r') as read_obj:
    csv_dict_reader = DictReader(read_obj)
    for row in csv_dict_reader:
        print(row['SepalLengthCm'], '\t', row['Species'])
```

DictReader returns a dictionary for each line during iteration. As in this dictionary keys are column names and values are cell values for that column. So, for selecting specific columns in every row, we used column name with the dictionary object.

Read specific columns (by column Number) in a csv file while iterating row by row

Iterate over all rows students.csv and for each row print contents of 2ns and 3rd column,

#Program to read and display the contents of a .csv file

```
1) import csv
   with open('D:\crime_data.csv') as csvfile:
                readCSV = csv.reader(csvfile, delimiter=',')
   for row in readCSV:
                           print(row)

2) import csv
f=open('C:/Users/USER/Desktop/MLLAB-07SEP2021/crime_data.csv')

#Create an empyt list to store the country column values
country = []
csv_f = csv.reader(f)

#loop over all the rows and append the first column values into the country[] list
for row in csv_f: #print(row[0])
              country.append(row[0])

print(country)
len1 = len(country)
print('list length = ', len1)
country1 =set(country)
country2 =set(country)
diff = country1.difference(country2)
print('difference = ',diff)
f.close()
```

set() method: **Used to convert any of the iterable to sequence of iterable elements with distinct elements**, commonly called Set.

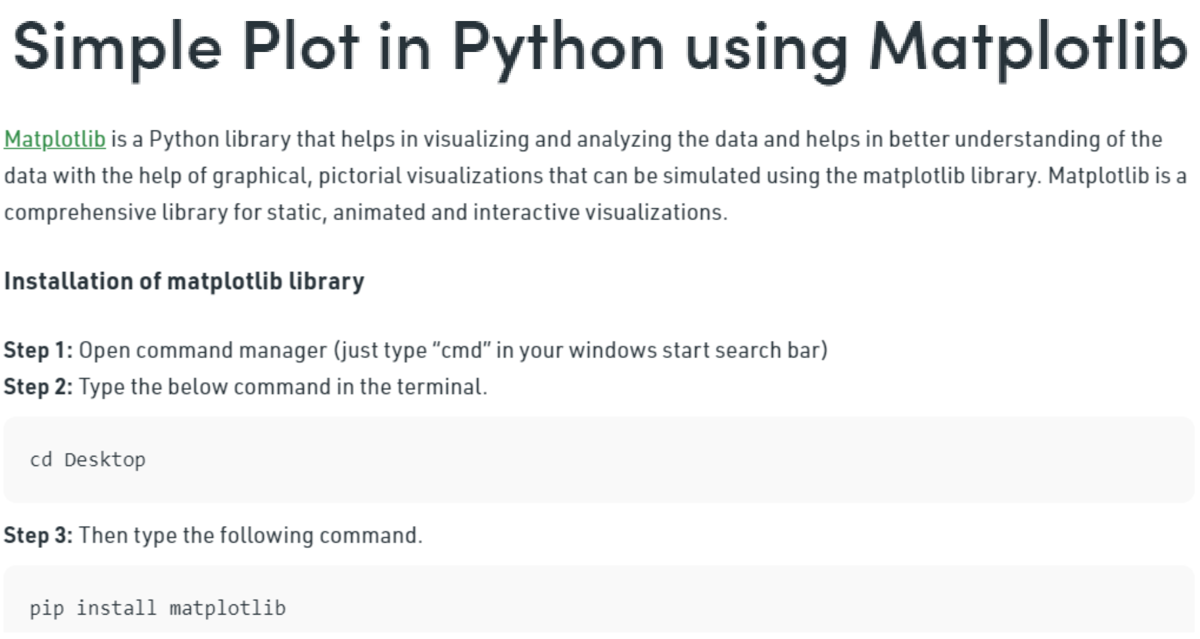
Syntax: **set(iterable)**

Parameters: Any iterable sequence like list, tuple or dictionary. Returns : An empty set if no element is passed.

MATPLOTLIB

The code seems self-explanatory. Following steps were followed:

- Define the x-axis and corresponding y-axis values as lists.
- Plot them on canvas using .plot() function.
- Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions.
- Give a title to your plot using .title() function.
- Finally, to view your plot, we use .show() function.



Simple Plot in Python using Matplotlib

Matplotlib is a Python library that helps in visualizing and analyzing the data and helps in better understanding of the data with the help of graphical, pictorial visualizations that can be simulated using the matplotlib library. Matplotlib is a comprehensive library for static, animated and interactive visualizations.

Installation of matplotlib library

Step 1: Open command manager (just type "cmd" in your windows start search bar)
Step 2: Type the below command in the terminal.

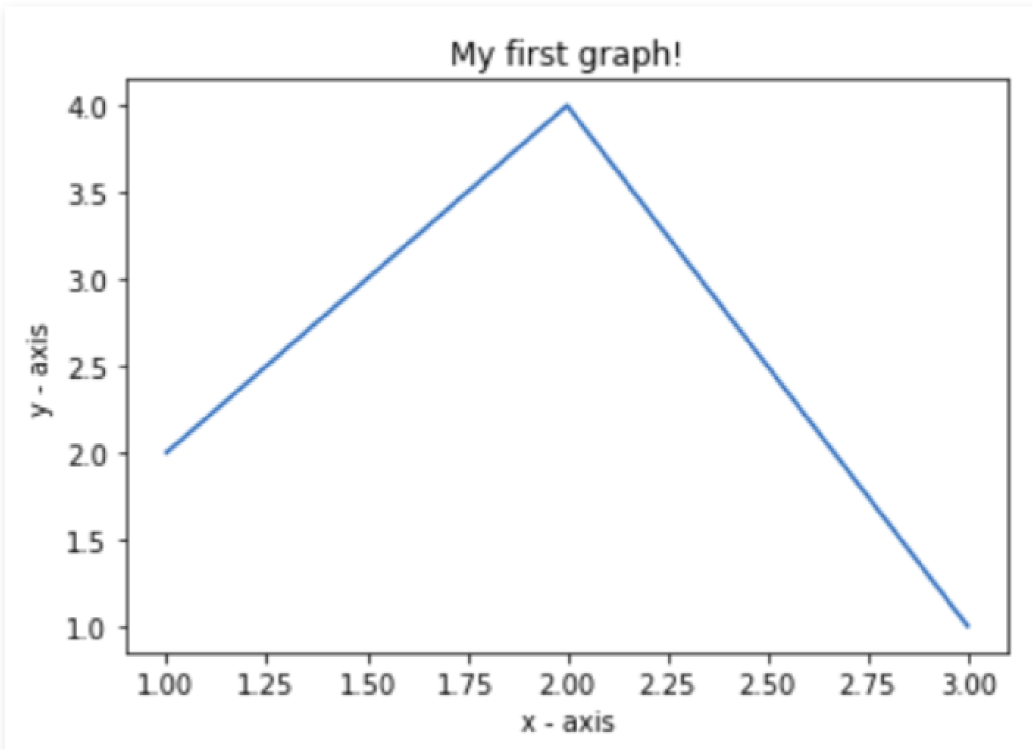
```
cd Desktop
```

Step 3: Then type the following command.

```
pip install matplotlib
```

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

Output:



Let's have a look at some of the basic functions that are often used in matplotlib.

Method	Description
plot()	it creates the plot at the background of computer, it doesn't displays it. We can also add a label as it's argument that by what name we will call this plot – utilized in legend()
show()	it displays the created plots
xlabel()	it labels the x-axis
ylabel()	it labels the y-axis
title()	it gives the title to the graph
gca()	it helps to get access over the all the four axes of the graph
gca().spines['right/left/top/bottom'].set_visible(True/False)	it access the individual spines or the individual boundaries and helps to change theoir visibility

xticks()	it decides how the markings are to be made on the x-axis
yticks()	it decides how the markings are to be made on the y-axis
gca().legend()	pass a list as it's arguments of all the plots made, if labels are not explicitly specified then add the values in the list in the same order as the plots are made
annotate()	it is use to write comments on the graph at the specified position
figure(figsize = (x, y))	whenever we want the result to be displayed in a separate window we use this command, and figsize argument decides what will be the initial size of the window that will be displayed after the run
subplot(r, c, i)	it is used to create multiple plots in the same figure with r signifies the no of rows in the figure, c signifies no of columns in a figure and i specifies the positioning of the particular plot
set_xticks	it is used to set the range and the step size of the markings on x – axis in a subplot
set_yticks	it is used to set the range and the step size of the markings on y – axis in a subplot

Python range() function

`range()` is a built-in function of Python. It is used when a user needs to perform an action for a specific number of times. `range()` in Python(3.x) is just a renamed version of a function called xrange in Python(2.x). The **range()** function is used to generate a sequence of numbers.

`range()` is commonly used in for looping hence, knowledge of same is key aspect when dealing with any kind of Python code. Most common use of `range()` function in Python is to iterate sequence type (List, string etc..) with for and while loop.

Python `range()` Basics :
In simple terms, `range()` allows user to generate a series of numbers within a given range. Depending on how many arguments user is passing to the function, user can decide where that series of numbers will begin and end as well as how big the difference will be between one number and the next.`range()` takes mainly three arguments.

- **start**: integer starting from which the sequence of integers is to be returned
- **stop:** integer before which the sequence of integers is to be returned.
  The range of integers end at stop – 1.
- **step:** integer value which determines the increment between each integer in the sequence

```python
# Python Program to
# show range() basics

# printing a number
for i in range(10):
    print(i, end =" ")
print()

# using range for iteration
l = [10, 20, 30, 40]
for i in range(len(l)):
    print(l[i], end =" ")
print()

# performing sum of natural
# number
sum = 0
for i in range(1, 11):
    sum = sum + i
print("Sum of first 10 natural number :", sum)
```

Output:

```
0 1 2 3 4 5 6 7 8 9

10 20 30 40

Sum of first 10 natural number : 55
```

There are three ways you can call `range()` :

- `range(stop)` takes one argument.
- `range(start, stop)` takes two arguments.
- `range(start, stop, step)` takes three arguments.

range(stop)

When user call `range()` with one argument, user will get a series of numbers that starts at 0 and includes every whole number up to, but not including, the number that user have provided as the stop. For Example –

Points to remember about Python `range()` function :

- `range()` function only works with the integers i.e. whole numbers.
- All argument must be integers. User can not pass a string or float number or any other type in a **start**, **stop** and **step** argument of a range().
- All three arguments can be positive or negative.
- The **step** value must not be zero. If a step is zero python raises a ValueError exception.
- `range()` is a type in Python

numpy.linspace

numpy.linspace(*start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0*)

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*].

The endpoint of the interval can optionally be excluded.

Parameters: **start : *array_like***

The starting value of the sequence.

stop : *array_like*

The end value of the sequence, unless *endpoint* is set to False. In that case, the sequence consists of all but the last of `num + 1` evenly spaced samples, so that *stop* is excluded. Note that the step size changes when *endpoint* is False.

num : *int, optional*

Number of samples to generate. Default is 50. Must be non-negative.

endpoint : *bool, optional*

If True, *stop* is the last sample. Otherwise, it is not included. Default is True.

retstep : *bool, optional*

If True, return (*samples*, *step*), where *step* is the spacing between samples.

dtype : *dtype, optional*

The type of the output array. If `dtype` is not given, the data type is inferred from *start* and *stop*. The inferred dtype will never be an integer; `float` is chosen even if the arguments would produce an array of integers.
New in version 1.9.0.

axis : *int, optional*

The axis in the result to store the samples. Relevant only if start or stop are array-like. By default (0), the samples will be along a new axis inserted at the beginning. Use -1 to get an axis at the end.
New in version 1.16.0.

Returns: **samples : *ndarray***

There are *num* equally spaced samples in the closed interval `[start, stop]` or the half-open interval `[start, stop)` (depending on whether *endpoint* is True or False).

step : *float, optional*

Only returned if *retstep* is True
Size of spacing between samples.

matplotlib.pyplot.ylim() Function

The **ylim() function** in pyplot module of matplotlib library is used to get or set the y-limits of the current axes.

Syntax: *matplotlib.pyplot.ylim(*args, **kwargs)*

Parameters: *This method accept the following parameters that are described below:*

- ***bottom:*** *This parameter is used to set the ylim to bottom.*
- ***top:*** *This parameter is used to set the ylim to top.*
- *****kwargs:*** *This parameter is Text properties that is used to control the appearance of the lab els.*

Returns: *This returns the following:*

- ***bottom, top :****This returns the tuple of the new y-axis limits.*

To display plot outputs inside the notebook itself (and not in the separate viewer), enter the following magic statement −

```
%matplotlib inline
```