Advanced Programming

Prepared by
Prof. Dr. Gamal M. Behery
Prof. of AI and image processing
Damietta University-Faculty of Computers and AI

1

1

جامعة دمياط
Damietta University

Book Part II

2

Matrix formation

- To form the matrix in a determinant form we use the bracket of the category with matrix command:

```
a = matrix ('{} {}; {} {}'.format(1,2,3,4))
b = matrix ('{} {} {}; {} {} {}'.format(1,2,3,4,5,6))
```

```
[[1 2]
 [3 4]]
[[1 2 3]
 [4 5 6]]
```


Trace command

- To sum the values of the main diameter elements of an array:

```
a = arange (9)
b = a.reshape(3, 3)
c = trace (b)
print(a)
print(b)
print(c)
```

```
[0 1 2 3 4 5 6 7 8]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
12
```

linalg.det & linalg.eig commands

To compute the **determine** and **eigen** values of matrix, use the **linalg.det & linalg.eig commands:**

```
c =linalg.det(b)
d = linalg.eig(b)
```


linalg.det & linalg.eig commands

```
from numpy import *
a = arange (9)
b = a.reshape(3, 3)
c =linalg.det(b)
d = linalg.eig(b)
print(a)
print(b)
print(c)
print(d)
```

```
[0 1 2 3 4 5 6 7 8]
[[0 1 2]
 [3 4 5]
 [6 7 8]]
0.0
(array([ 1.33484692e+01, -1.34846923e+00, -2.48477279e-16]), array([[ 0.16476382
,  0.79969966,  0.40824829],
       [ 0.50577448,  0.10420579, -0.81649658],
       [ 0.84678513, -0.59128809,  0.40824829]]))
```


To cut a part of a matrix

```
from numpy import *
a = arange (10)
b = a[3]
c =a[3:9]
d =a[3:9:2]
e = a[-1]
f = a[-3]
```

```
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
```

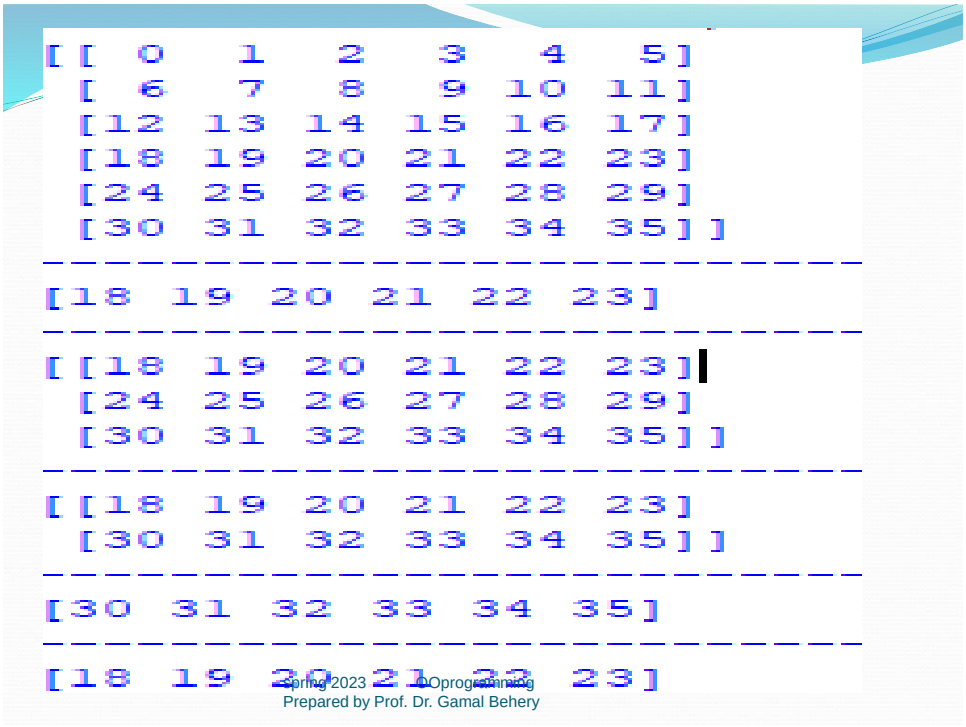
```
[0 1 2 3 4 5 6 7 8 9]
3
[3 4 5 6 7 8]
[3 5 7]
9
7
```


9

To cut a part of a matrix

```
from numpy import *
a = arange(36).reshape(6,6)
b = a[3]
c =a[3:9]
d =a[3:9:2]
e = a[-1]
f = a[-3]
print(a)
print('-------------------------')
print(b)
print('-------------------------')
print(c)
print('-------------------------')
print(d)
print('-------------------------')
print(e)
print('-------------------------')
print(f)
```

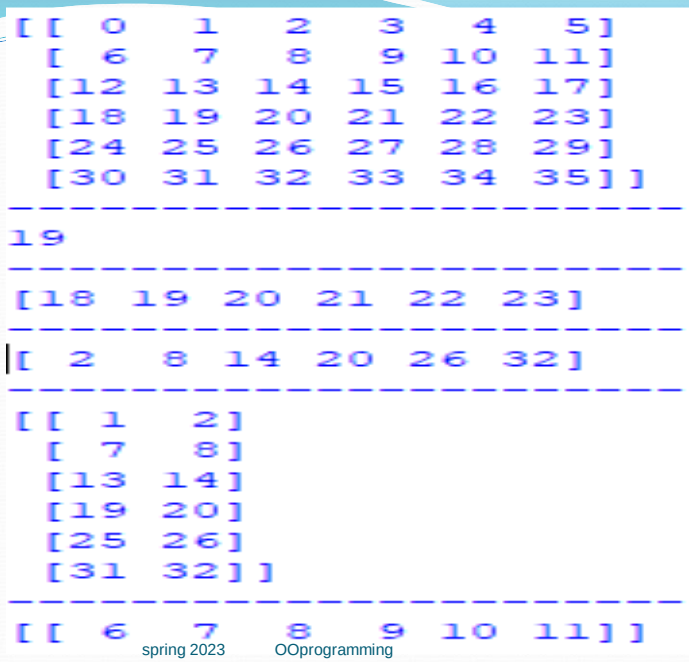

10



To cut a part of a matrix

```
from numpy import *
a = arange (36).reshape(6, 6)
b = a[3, 1]
c =a[3, :]
d =a[: ,2]
e = a[:, 1:3]
f = a[1:2, :]
```

```
print(b)
print('-------------------------')
print(c)
print('-------------------------')
print(d)
print('-------------------------')
print(e)
print('-------------------------')
print(f)
```



To cut a part of a matrix

```
from numpy import *
a = arange (36).reshape(6,6)
b = a[3:4,1:5]
c =a[2:,3:]
d =a[:2,:3]
e = a[:,-1]
f = a[-1,:]
```

```
print(b)
print('-------------------------')
print(c)
print('-------------------------')
print(d)
print('-------------------------')
print(e)
print('-------------------------')
print(f)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
------------------------
[[19 20 21 22]]
------------------------
[[15 16 17]
 [21 22 23]
 [27 28 29]
 [33 34 35]]
------------------------
[[0 1 2]
 [6 7 8]]
------------------------
[ 5 11 17 23 29 35]
------------------------
[30 31 32 33 34 35]
```


To cut a part of a matrix

```
from numpy import *
a = arange (36).reshape(6,6)
b = a[::2, ::3]
c =a[::-1, ::-1]
d =a[:2:-1, :3:-1]
e = a[2::2, 3::3]
f = a[-1::, -1::]
print(b)
print('------------------------')
print(c)
print('------------------------')
print(d)
print('------------------------')
print(e)
print('------------------------')
print(f)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
--------------------------
[[ 0  3]
 [12 15]
 [24 27]]
--------------------------
[[35 34 33 32 31 30]
 [29 28 27 26 25 24]
 [23 22 21 20 19 18]
 [17 16 15 14 13 12]
 [11 10  9  8  7  6]
 [ 5  4  3  2  1  0]]
--------------------------
[[35 34]
 [29 28]
 [23 22]]
--------------------------
[[15]
 [27]]
--------------------------
[[35]]
```



```
from numpy import *
a = arange (16).reshape(4,4)
print(a)
print('--------------------')
a[2, 3] = 0
print(a)
print('--------------------')
a[:, 3] = 0
print(a)
```

Modify values in the array

```
from numpy import *
a = arange (16).reshape(4,4)
print(a)
print('--------------------')
a[2, 3] = 0
print(a)
print('--------------------')
a[:, 3] = 0
print(a)
print('--------------------')
a[2, :] = 0
print(a)
print('--------------------')
a[:, :] = 0
print(a)
```


19

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
--------------------
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10  0]
 [12 13 14 15]]
--------------------
[[ 0  1  2  0]
 [ 4  5  6  0]
 [ 8  9 10  0]
 [12 13 14  0]]
--------------------
[[ 0  1  2  0]
 [ 4  5  6  0]
 [ 0  0  0  0]
 [12 13 14  0]]
--------------------
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```


20

Modify values in the array

```
from numpy import *
a = arange (16).reshape(4, 4)
print(a)
print('--------------------')
b = a[:, 1: 3]
print(b)
print('--------------------')
a[:, :] = 5
print('--------------------')
print(a)
print('--------------------')
print(b)
```



```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
--------------------
[[ 1  2]
 [ 5  6]
 [ 9 10]
 [13 14]]
--------------------
--------------------
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]]
--------------------
[[5 5]
 [5 5]
 [5 5]
 [5 5]]
```

Modify values in the array

```
from numpy import *
a = arange (18).reshape(3, 3, 2)
c = a[1]  # the second layer
d = a[1, 2] #the second layer, third raw
e = a[2, 2, 1] #the third layer, third raw, second column
print(a)
print('--------------------')
print(c)
print('--------------------')
print(d)
print('--------------------')
print(e)
print('--------------------')
```



```
[[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]

 [[12 13]
  [14 15]
  [16 17]]]
--------------------
[[ 6  7]
 [ 8  9]
 [10 11]]
--------------------
[10 11]
--------------------
17
--------------------
```

Matrix (segmentation) splitting, split command

```
from numpy import *
x = [11, 22, 33, 44, 55, 66, 77, 88]
x1, x2, x3 = split(x, (3, 6))
print (x1, x2, x3)
print('--------------------')
x1, x2, x3 = split(x, (1, 5))
print(x1, x2, x3)
print('--------------------')
```

```
x1, x2, x3 = split(x, (6, 3))
print (x1, x2, x3)
print('--------------------')
x1, x2, x3 = split(x, (0, 3))
print (x1, x2, x3)
print('--------------------')
x1, x2, x3 = split(x, (4, 0))
print (x1, x2, x3)
```



```
[11 22 33] [44 55 66] [77 88]
--------------------
[11] [22 33 44 55] [66 77 88]
--------------------
[11 22 33 44 55 66] [] [44 55 66 77 88]
--------------------
[] [11 22 33] [44 55 66 77 88]
--------------------
[11 22 33 44] [] [11 22 33 44 55 66 77 88]
```

Merge two matrices (Vertical)

vstack Command

```
from numpy import *
a = arange (4).reshape(2,2)
b = 2*arange (4).reshape(2,2)
c = vstack((a, b))
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```



```
[[0  1]
 [2  3]]
--------------------------
[[0  2]
 [4  6]]
--------------------------
[[0  1]
 [2  3]
 [0  2]
 [4  6]]
--------------------------
```

Merge two matrices (Horizontal)

hstack Command

```
from numpy import *
a = arange (4).reshape(2,2)
b = 2*arange (4).reshape(2,2)
c = hstack((a, b))
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```


29

```
[[0 1]
 [2 3]]
--------------------------
[[0 2]
 [4 6]]
--------------------------
[[0 1 0 2]
 [2 3 4 6]]
--------------------------
```


30

Merge two matrices (Axis = 0)

concatenate Command

```
from numpy import *
a = arange (4).reshape(2,2)
b = arange (4).reshape(2,2)
#should the no. of row in array = no. of row in array b
#axis = 0 means merge vertically
c = concatenate((a, b), axis = 0)
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```


31

```
[[0  1]
 [2  3]]
--------------------------
[[0  1]
 [2  3]]
--------------------------
[[0  1]
 [2  3]
 [0  1]
 [2  3]]
--------------------------
```


32

Merge two matrices (Axis = 1)

concatenate Command

```
from numpy import *
a = arange (4).reshape(2,2)
b = arange (4).reshape(2,2)
#should the no. of row in array = no. of row in array b
#axis = 0 means merge horizental
c = concatenate((a, b), axis = 1)
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```



```
[[0 1]
 [2 3]]
--------------------------
[[0 1]
 [2 3]]
--------------------------
[[0 1 0 1]
 [2 3 2 3]]
--------------------------
```

max, min, argmax, argmin commands

```
import numpy as np
a = np.random.randint(5,20, size = 9).reshape(3,3)
b=np.max(a)
c=np.min(a)
d=np.argmax(a)
e=np.argmin(a)
print(a)
print('--------------------------')
print(b)
```


max, min, argmax, argmin commands

```
print('--------------------------')
print(c)
print('--------------------------')
print(d)
print('--------------------------')
print(e)
print('--------------------------')
```

```
[[ 5  13   6]
 [15  16  19]
 [13  12  15]]
----------
19
----------
5
----------
5
----------
0
----------
```


Variance and covariance

var & cov commands

```
import numpy as np
a = np.random.randint(5,20, size = 9).reshape(3,3)
b=np.var(a)
c=np.cov(a)
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```

```
[[ 5 15 13]
 [10 13 11]
 [12 10 16]]
----------------------------
9.333333333333334
----------------------------
[[28.          7.          0.        ]
 [ 7.          2.33333333 -2.33333333]
 [ 0.         -2.33333333  9.33333333]]
----------------------------
```


The Mathematical operations on Arrays

addition and subtraction

```
import numpy as np
a = np.random.randint(5,20, size = 9).reshape(3,3)
b = np.random.randint(5,20, size = 9).reshape(3,3)
c = a + b
d = a - b
print(a)
print('----------------------------')
print(b)
print('----------------------------')
print(c)
print('----------------------------')
print(d)
print('----------------------------')
```

```
[[19   8  19]
 [18  13  11]
 [10   6  15]]
----------------------
[[10  19   7]
 [10  11  11]
 [ 5   5   9]]
----------------------
[[29  27  26]
 [28  24  22]
 [15  11  24]]
----------------------
[[  9 -11  12]
 [  8   2   0]
 [  5   1   6]]
----------------------
```


Scaler product and divison on arrays

```
import numpy as np
a = np.random.randint(1,10, size = 9).reshape(3,3)
b = np.random.randint(1,10, size = 9).reshape(3,3)
c = a * b
d = a / b
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
print(d)
print('--------------------------')
```

```
[[1 9 8]
 [1 7 4]
 [1 3 3]]
----------------------------
[[7 7 9]
 [1 2 3]
 [5 4 9]]
----------------------------
[[ 7 63 72]
 [ 1 14 12]
 [ 5 12 27]]
----------------------------
[[0.14285714 1.28571429 0.88888889]
 [1.         3.5        1.33333333]
 [0.2        0.75       0.33333333]]
----------------------------
```


Cartesian product of matrices

```
import numpy as np
a = np.random.randint(1,10, size = 9).reshape(3,3)
b = np.random.randint(1,10, size = 9).reshape(3,3)
c = np.dot(a,b)
print(a)
print('----------------------------')
print(b)
print('----------------------------')
print(c)
print('----------------------------')
print(d)
print('----------------------------')
```

```
[[1  2  3]
 [9  5  4]
 [9  4  5]]
----------------------------
[[9  8  2]
 [5  1  1]
 [2  2  1]]
----------------------------
[[  25   16    7]
 [114   85   27]
 [111   86   27]]
----------------------------
```


Cartesian product of matrices

```
import numpy as np
a = np.random.randint(1,10, size = 8).reshape(2, 4)
b = np.random.randint(1,10, size = 12).reshape(4, 3)
c = np.dot(a,b)
print(a)
print('----------------------------')
print(b)
print('----------------------------')
print(c)
print('----------------------------')
```

```
[[5  7  4  9]
 [9  9  9  7]]
----------------------
[[9  1  6]
 [9  4  1]
 [6  7  1]
 [7  9  5]]
----------------------
[[195  142   86]
 [265  171  107]]
----------------------
```


sum command

```
from numpy import *
a = random.randint(5,20, size = 9).reshape(3,3)
b = sum(a) # sum all elements of array a
c = a.sum()  # sum all elements of array a
d = a.sum(axis = 1) # sum elements array raw by row
e = a.sum(axis = 0) # sum elements array column by column
print(a)
print('----------------------------')
print(b)
print('----------------------------')
print(c)
print('----------------------------')
print(d)
print('----------------------------')
print(e)
print('----------------------------')
```

```
[[16  18   9]
 [ 9  13  11]
 [19  18   7]]
--------------------
120
--------------------
120
--------------------
[43  33  44]
--------------------
[44  49  27]
--------------------
```


49

statistical value of array
mean(), std(), var() command

```
from numpy import *
a = random.randint(5,20, size = 9).reshape(3,3)
b = a.mean()
c = a.std()
d = a.var()
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
print(d)
print('--------------------------')
```


50

```
[[ 8  15   6]
 [17  11  14]
 [18  18   7]]
--------------------------
12.666666666666666
--------------------------
4.521553322083512
--------------------------
20.444444444444443
--------------------------
```


51

statistical value of array (correlation coefficient) corrceof() command

```
from numpy import *
a = random.randint(5,20, size = 9).reshape(3,3)
b = corrcoef(a)
print(a)
print('--------------------------')
print(b)
print('--------------------------')
```


52

```
[[ 7  9 19]
 [ 9 16 13]
 [ 7  8  6]]
----------------------------
[[ 1.          0.23621544 -0.77771377]
 [ 0.23621544  1.          0.4271211 ]
 [-0.77771377  0.4271211   1.        ]]
----------------------------
```


Sorting of array elements

sort() command

```
from numpy import *
a = random.randint(5,20, size = 9).reshape(3,3)
b = sort(a, axis = 0)
b = sort(a, axis = 1)

print(a)
print('----------------------------')
print(b)
print('----------------------------')
```

```
[[  8  17   9]
 [17  13  10]
 [15   5  12]]
--------------------
[[  8   5   9]
 [15  13  10]
 [17  17  12]]
--------------------
[[  8   9  17]
 [10  13  17]
 [ 5  12  15]]
--------------------
```

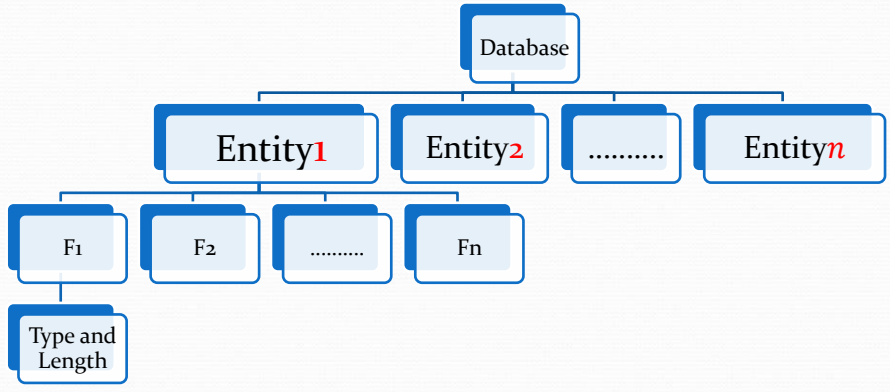

Inverse of array
linalg.inv() command

```
from numpy import *
a = random.randint(5,20, size = 9).reshape(3,3)
b = linalg.inv(a)
c = dot(a, b)
print(a)
print('--------------------------')
print(b)
print('--------------------------')
print(c)
print('--------------------------')
```

```
[[16 11 12]
 [ 6  9 16]
 [12 13  7]]
----------------------------
[[ 0.1407767  -0.07669903 -0.06601942]
 [-0.14563107  0.03106796  0.17864078]
 [ 0.02912621  0.07378641 -0.07572816]]
----------------------------
[[ 1.00000000e+00  5.55111512e-17  1.66533454e-16]
 [-2.22044605e-16  1.00000000e+00  0.00000000e+00]
 [-3.19189120e-16  1.38777878e-17  1.00000000e+00]]
----------------------------
```


57

Database Components



58

Python MySQL

- **SQL** stands for Structured Query Language and is a widely used programming language for managing relational databases.
- You may have heard of the different flavour's of SQL-based DBMSs.
- The most popular ones include MySQL, PostgreSQL, SQLite, and SQL Server. All of these databases are compliant with the SQL standards but with varying degrees of compliance.
- **MySQL is also a part of the Oracle ecosystem**. While its core functionality is completely free, there are some paid add-ons as well.

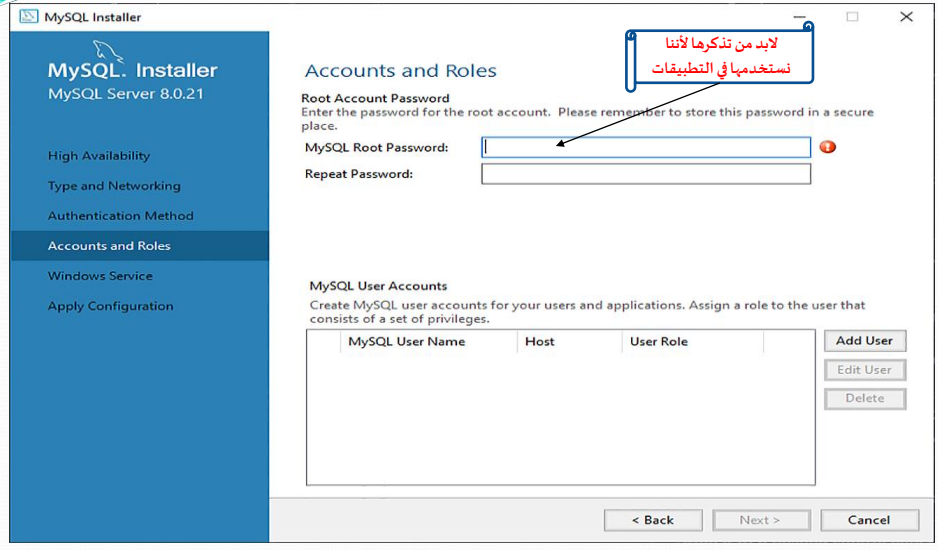
Installing MySQL Server and MySQL Connector/Python

- Now, to start working through this tutorial, you need to set up two things: a **MySQL server** and a **MySQL connector**.
- MySQL server will provide all the services required for handling your database.
- **Once the server is up and running, you can connect your Python application with it using MySQL Connector/Python.**

Installing MySQL Server

- For Windows, the best way is to download **MySQL Installer** and let it take care of the entire process.
- The installation manager also helps you configure the security settings of the MySQL server.
- On the Accounts and Roles page, you need to enter a password for the **root (admin)** account and also optionally add other users with varying privileges.

61



62

Note: Remember the hostname, username, and password as these will be required to establish a connection with the MySQL server later on.

63

Installing MySQL Connector/Python

- A **database driver** is a piece of software that allows an application to connect and interact with a database system.
- Programming languages like Python need a special driver before they can speak to a database from a specific vendor.

64

Installing MySQL Connector/Python

- **Note:** MySQL's official documentation uses the term **connector** instead of **driver**.
- Technically, connectors are associated only with connecting to a database, not interacting with it. However, the term is often used for the entire database access module comprising the connector *and* the driver.
- To maintain consistency with the documentation, you'll see the term **connector** whenever MySQL is mentioned.

Installing MySQL Connector/Python

- Similarly, in Python you need to install a Python MySQL connector to interact with a MySQL database.
- Many packages follow the DB-API standards, but the most popular among them is MySQL Connector/Python.
- You can get it with pip:

shell

pip install mysql-connector-python

Installing MySQL Connector/Python

- To test if the installation was successful, type the following command on your Python terminal:

python

>>> import mysql.connector

- ***If the above code executes with no errors, then mysql.connector is installed and ready to use.***
- ***If you encounter any errors, then make sure you're in the correct virtual environment and you're using the right Python interpreter.***

Establishing a Connection With MySQL Server

- MySQL is a **server-based** database management system.
- One server might contain multiple databases.
- To interact with a database, you must first establish a connection with the server.
- **The general workflow of a Python program that interacts with a MySQL-based database is as follows:**

Establishing a Connection With MySQL Server

1. Connect to the MySQL server.
2. Create a new database.
3. Connect to the newly created or an existing database.
4. Execute a SQL query and fetch results.
5. Inform the database if any changes are made to a table.
6. Close the connection to the MySQL server.

Establishing a Connection

- The first step in interacting with a MySQL server is to establish a connection.
- To do this, you need connect() from the mysql.connector module.
- This function takes in parameters like host, user, and password and returns a MySQLConnection object.
- You can receive these credentials as input from the user and pass them to connect():

Connecting to MySQL databases

- The first thing you need to do to connect to the database is to include the **mysql.connector** module in your code.
- Now you can use **connect(), cursor(), execute()** and build in functions in this module to deal with databases.

Connect() function:

- This function is used to **create** an object of type MySQLConnection that allows us to **connect** to the database. You can built it as follows:

```
mysql.connector.connect(host='', user='', passwd='', database='')
```


Connecting to MySQL databases

- The **host** parameter**:** We specify the database server, and if it is a local, we choose **"localhost"**.
- **The user** Parameter: We specify the name of the user through which we will deal with the database, default is "**root**".
- The **passwd** Parameter: We specify the password of the user through which we will deal with the database.
- The **database** parameter: We specify the name of the database we intend to work with.

cursor() function:

- This function is called from the object returned by **connect(),** which returns an object of type MySQLCursor.
- The MySQLCursor object contains build-in functions that can be used for the following reasons:
  - To send queries to the database server.
  - Gets the results returned by the database server after the query is executed.

function execute(operation)

- This function is called from the object returned by cursor(), in order to send queries to the database server.
- In place of the parameter operation we pass a text representing the query we want to send to the database server.

Creating a Database

- To create a database in MySQL, use the "***CREATE DATABASE***" statement.
- Example_01: Create a database named "mydatabase":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$")
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```


Check if Database Exists

- You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:
- Example_02 :

  Return a list of your system's databases?

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$")
mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
```


The output:

```
('company',)
('company1',)
('company2',)
('gamalbehery',)
('gamalbehery1',)
('information_schema',)
('mydatabase',)
('mydatabase1',)
('mysql',)
('performance_schema',)
('sakila',)
('sys',)
('world',)
```

Connecting to existing database

- Example_03 : Connect to the database named "mydatabase":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
```


Create Table

- To create a table in MySQL, use the "CREATE TABLE" statement.
- Make sure you define the name of the database when you create the connection.

```
"CREATE TABLE Name_of_table (name_of_field1 type and length, name_of_field2 type and length, …)"
```

- Example_04:

  Create a table named "**customers**"

Create Table

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers
(name VARCHAR(255), address
VARCHAR(255))")
```


Check if Table Exists

- You can check if a table exist by listing all tables in your database with the "**SHOW TABLES**" statement:

Example_05:

Return a list of your table's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW TABLES")

for x in mycursor:
    print(x)
```


Primary Key

- When creating a **table**, you should also create a column with a unique key for each record.
- This can be done by defining a **PRIMARY KEY**.
- We use the statement "**INT AUTO_INCREMENT PRIMARY KEY**" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Primary Key

- **Example_06:** Create primary key when creating the table:

```
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Gam@#$2911961@#$",
    database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers1 (id INT
AUTO_INCREMENT PRIMARY KEY,name VARCHAR(255),
address VARCHAR(255))")
```

Primary Key

Tables

If the table already exists, use the ALTER TABLE keyword

Example_07:

Create primary key on an existing table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase"
)

mycursor = mydb.cursor()
mycursor.execute("ALTER TABLE customers
ADD COLUMN id INT AUTO_INCREMENT PRIMARY
KEY")
```


Insert Into Table

- To fill a table in MySQL, use the "INSERT INTO" statement.

Example_08:

- Insert a record in the "customers" table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
```

```
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address)
VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows

- To insert multiple rows into a table, use the **executemany()** method.
- The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

- **Example_09:**

  Fill the "customers" table with data:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
```


91

```
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]
```


92

```
mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")
```


93

Get Inserted ID

- You can get the id of the row you just inserted by asking the cursor object.
- **Note:** If you insert more than one row, the id of the last inserted row is returned.
- **Example_10:**

  Insert one row, and return the ID:

94

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Michelle", "Blue Village")

mycursor.execute(sql, val)
mydb.commit()

print("1 record inserted, ID:", mycursor.lastrowid)
```


Python MySQL Select From

Select From a Table

To select from a table in MySQL, use the "**SELECT**" statement:

Example_11:

Select all records from the "customers" table, and display the result:

- **Note: We use the fetchall() method, which fetches all rows from the last executed statement.**

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Selecting Columns

- To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

Example_12:

Select only the name and address columns:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
mycursor.execute("SELECT name, address FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Using the fetchone() Method

- If you are only interested in one row, you can use the fetchone() method.
- The fetchone() method will return the first row of the result:

Example_13:

Fetch only one row:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchone()
print(myresult)
```


Python MySQL Where

Select With a Filter

- When selecting records from a table, you can filter the selection by using the "**WHERE**" statement:

Example_14:

Select record(s) where the address is "Park Lane 38": result:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "SELECT * FROM customers WHERE address
='Park Lane 38'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Wildcard Characters

- You can also select the records that starts, includes, or ends with a given letter or phrase.
- Use the % to represent wildcard characters:

Example_15:

Select records where the address contains the word "way":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
sql = "SELECT * FROM customers WHERE address LIKE '%way%'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Prevent SQL Injection

- When query values are provided by the user, you should escape the values.
- This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.
- The mysql.connector module has methods to escape query values:

Example_16:

Escape query values by using the placeholder %s method:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "SELECT * FROM customers WHERE address =
%s"
adr = ("Yellow Garden 2", )
mycursor.execute(sql, adr)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Python MySQL Order By

Sort the Result

- Use the **ORDER BY** statement to sort the result in ascending or descending order.
- The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

Example_17:

Sort the result alphabetically by name:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


ORDER BY DESC

- Use the DESC keyword to sort the result in a descending order.

Example_18:

Sort the result reverse alphabetically by name:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name
DESC"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Python MySQL Delete From By

Delete Record

- You can delete records from an existing table by using the "**DELETE FROM**" statement:

Example_19:

Delete any record where the address is "Mountain 21":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
sql = "DELETE FROM customers WHERE address =
'Mountain 21'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
```


Important!:
Notice the statement: **mydb.commit().** It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the DELETE syntax:
The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

Prevent SQL Injection

- It is considered a good practice to escape the values of any query, also in delete statements.
- This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.
- The mysql.connector module uses the placeholder %s to escape values in the delete statement:

- **Example_20:**
- Escape values by using the placeholder %s method:


```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "DELETE FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )
mycursor.execute(sql, adr)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
```

Python MySQL Drop Table

Delete a Table

- You can delete an existing table by using the "**DROP TABLE**" statement:

Example_21:

Delete the table "customers":


```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()

sql = "DROP TABLE customers1"

mycursor.execute(sql)
```

Drop Only if Exist

- If the table you want to delete is already deleted, or for any other reason does not exist, you can use the **IF EXISTS** keyword to avoid getting an error.

Example_22:

Delete the table "customers_2" if it exists:


```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()

sql = "DROP TABLE IF EXISTS customers_2"

mycursor.execute(sql)
```

Python MySQL Update Table

Update Table

- You can update existing records in a table by using the "**UPDATE**" statement:

Example_23:

Overwrite the address column from "Valley 345" to "Canyon 123":


```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "UPDATE customers SET address = 'Canyon 123'
WHERE address = 'Valley 345'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

Important!: Notice the statement: **mydb.commit().** It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Prevent SQL Injection

- It is considered a good practice to escape the values of any query, also in update statements.
- This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.
- The mysql.connector module uses the placeholder %s to escape values in the delete statement:

Example_23:

Escape values by using the placeholder %s method:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "UPDATE customers SET address = %s WHERE
address = %s"
val = ("Valley 345", "Canyon 123")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```


Python MySQL Limit

Limit the Result

- You can limit the number of records returned from the query, by using the "**LIMIT**" statement:

- **Example_25:**

Select the 5 first records in the "customers" table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers LIMIT 5")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Start From Another Position

- If you want to return five records, starting from the third record, you can use the "**OFFSET**" keyword:

Example_26:

Start from position 3, and return 5 records:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


Python MySQL Join

Join Two or More Tables

- You can combine rows from two or more tables, based on a related column between them, by using a **JOIN** statement. Consider you have a "users" table and a "products" table:

users

```
{ id: 1, name: 'John', fav: 154},
{ id: 2, name: 'Peter', fav: 154},
{ id: 3, name: 'Amy', fav: 155},
{ id: 4, name: 'Hannah', fav:},
{ id: 5, name: 'Michael', fav:}
```

products

```
{ id: 154, name: 'Chocolate Heaven' },
{ id: 155, name: 'Tasty Lemons' },
{ id: 156, name: 'Vanilla Dreams' }
```

- These two tables can be combined by using users' **fav field** and products' **id field**.

Example_27:

Join users and products to see the name of the users favourite product:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")
mycursor = mydb.cursor()
```



```
#mycursor.execute("CREATE TABLE users (id int,
name VARCHAR(255),fav VARCHAR(255))")
#mycursor.execute("CREATE TABLE products (id int,
name VARCHAR(255))")
sql = "INSERT INTO users (id,name, fav) VALUES (%s,
%s, %s)"
val = [(1,'John',154),(2,'Peter',154),(3, 'Amy',155),(4,
'Hannah',153),(5, 'Michael', 156)]

mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount,"was inserted.")
```

```
sql = "INSERT INTO products (id,name) VALUES (%s,
%s)"
val= [(154,'Chocolate Heaven' ),(155, 'Tasty
Lemons'),(156, 'Vanilla Dreams')]
mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount,"was inserted.")
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  INNER JOIN products ON users.fav = products.id"
```



```
mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

LEFT JOIN

- In the example above, Hannah, and Michael were excluded from the result, that is because INNER JOIN only shows the records where there is a match.
- If you want to show all users, even if they do not have a favourite product, use the **LEFT JOIN** statement:

Example_28:

Select all users and their favourite product:


```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Gam@#$2911961@#$",
  database="mydatabase")

mycursor = mydb.cursor()
```

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  LEFT JOIN products ON users.fav =
products.id"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```


RIGHT JOIN

- If you want to return all products, and the users who have them as their favourite, even if no user have them as their favourite, use the **RIGHT JOIN** statement:

Example_29:

Select all products, and the user(s) who have them as their favourite:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase")

mycursor = mydb.cursor()
```


139

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  RIGHT JOIN products ON users.fav =
products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```


140

How to read a numerical data or file in Python with numpy?

- Numerical data can be present in different formats of file :
- The data can be saved in a txt file where each line has a new data point.
- The data can be stored in a CSV(comma separated values) file.
- The data can be also stored in TSV(tab separated values) file.
- There are multiple ways of storing data in files and the above ones are some of the most used formats for storing numerical data. To achieve our required functionality numpy's loadtxt() function will be used.

How to read a numerical data or file in Python with numpy?

Syntax:

numpy.loadtxt(fname, dtype='float', comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)

parameters:

fname : *File, filename, or generator to read. If the filename extension is .gz or .bz2, the file is first decompressed. Note that generators should return byte strings for Python 3k.*

How to read a numerical data or file in Python with numpy?

dtype : *Data-type of the resulting array; default: float. If this is a structured data-type, the resulting array will be 1-dimensional, and each row will be interpreted as an element of the array.*

delimiter : *The string used to separate values. By default, this is any whitespace.*

converters : *A dictionary mapping column number to a function that will convert that column to a float. E.g., if column 0 is a date string: converters = {0: datestr2num}. Default: None.*

skiprows : *Skip the first skiprows lines; default: 0.*

Examp_1

```
# Python program explaining
# loadtxt() function
import numpy as np
# StringIO behaves like a file object
from io import StringIO
c = StringIO("0 1 2 \n3 4 5")
d = np.loadtxt(c)
print(d)
```

```
[[0. 1. 2.]
 [3. 4. 5.]]
```

Examp_2

```
# Python program explaining
# loadtxt() function
import numpy as geek
# StringIO behaves like a file object
from io import StringIO
d = StringIO("M 21 72\nF 35 58")
e = geek.loadtxt(d, dtype ={'names': ('gender', 'age',
'weight'),
                            'formats': ('S1', 'i4', 'f4')})
print(e)
```

```
[(b'M', 21, 72.) (b'F', 35, 58.)]
```


Examp_3

```
# Importing libraries that will be used
import numpy as np
# Setting name of the file that the data is to be extracted from
in python
# This is a comma separated values file
filename = 'gfg_example4.csv'
# Loading file data into numpy array and storing it in
variable.
# We use a delimiter that basically tells the code that at every
',' we encounter,
```

Examp_3

```
# we need to treat it as a new data point.
# The data type of the variables is set to be int using dtype parameter.
data_collected = np.loadtxt(filename, delimiter=',', dtype=int)
# Printing data stored
print(data_collected)
# Type of data
print(f'Stored in : {type(data_collected)} and data type is : {data_collected.dtype}')
```


Examp_4

```
# Importing libraries that will be used
import numpy as np
# Setting name of the file that the data is to be extracted from in python
# This
filename = 'gfg_example3.tsv'
# Loading file data into numpy array and storing it in variable called data_collected
# We use a delimiter that basically tells the code that at every ',' we encounter,
# we need to treat it as a new data point.
data_collected = np.loadtxt(filename, delimiter='\t')
# Printing data stored
print(data_collected)
# Type of data
print(f'Stored in : {type(data_collected)} and data type is : {data_collected.dtype}')
```

Matplotlib Library

matplotlib

149

What is Matplotlib?

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

```
import matplotlib
print(matplotlib.__version__)
```


150

Matplotlib Pyplot

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

Matplotlib has 13 branches as follows:

151

- Style
- Plot
- Scatter
- Pie
- Bar
- Histogram
- Contour
- 3D
- Subplot
- Annotate
- Imshow
- Legend

152

1- style branch

153

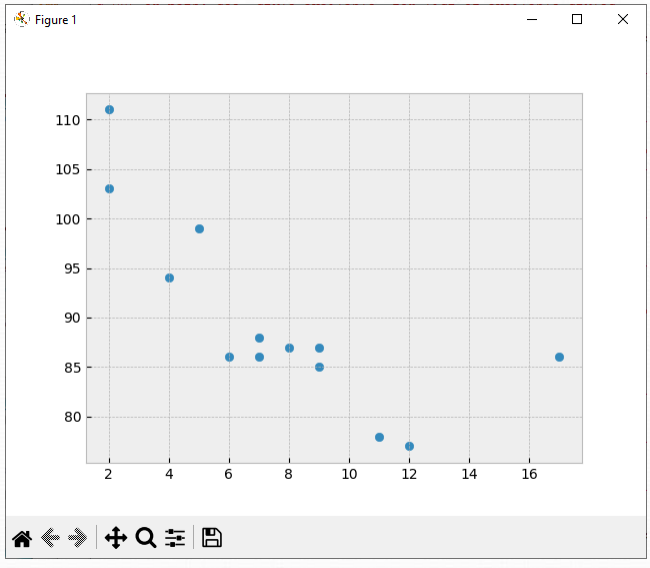
Plt.style.use()

- The style is a colour package related to the drawing and is determined to determine the shape of the drawing for the product and this is done with the following command:
- **Plt.style.use(‘ ’)**

154

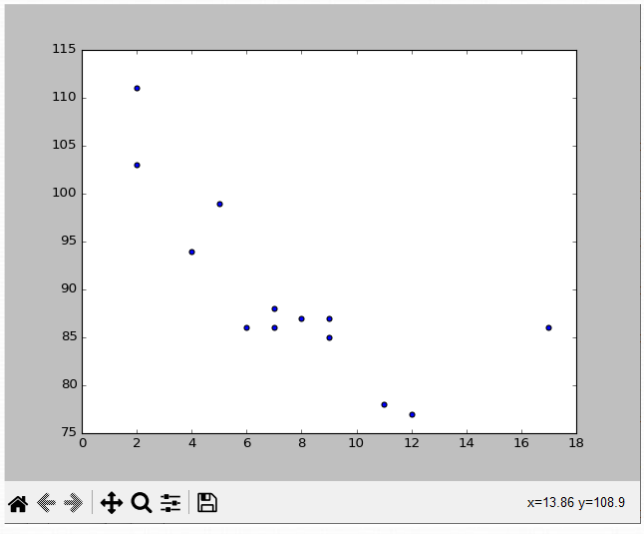
a) bmh Bouquet

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use(bmh')
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

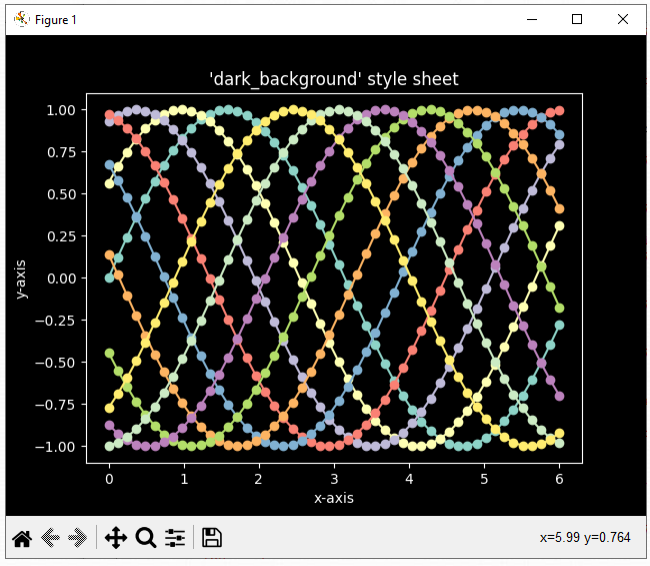
b) classic Bouquet

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

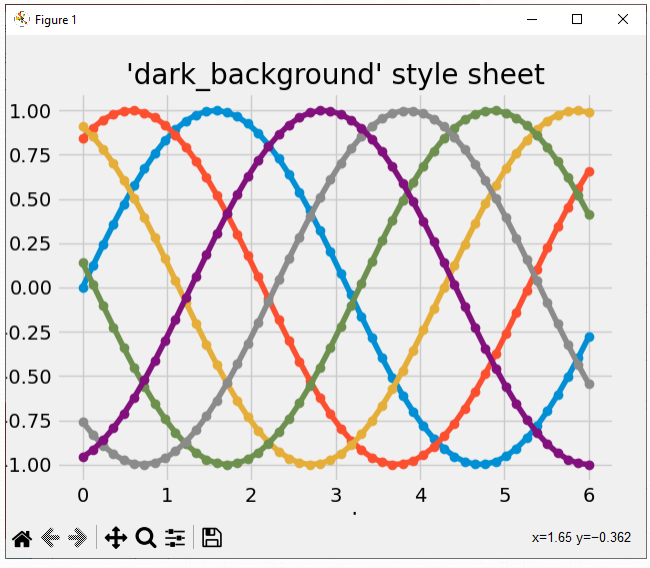



c) dark_background Bouquet

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('dark_background')
fig, ax = plt.subplots()
L = 6
x = np.linspace(0, L)
ncolors = len(plt.rcParams['axes.prop_cycle'])
shift = np.linspace(0, L, ncolors, endpoint=False)
for s in shift:
    ax.plot(x, np.sin(x + s), 'o-')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_title("'dark_background' style sheet")
plt.show()
```

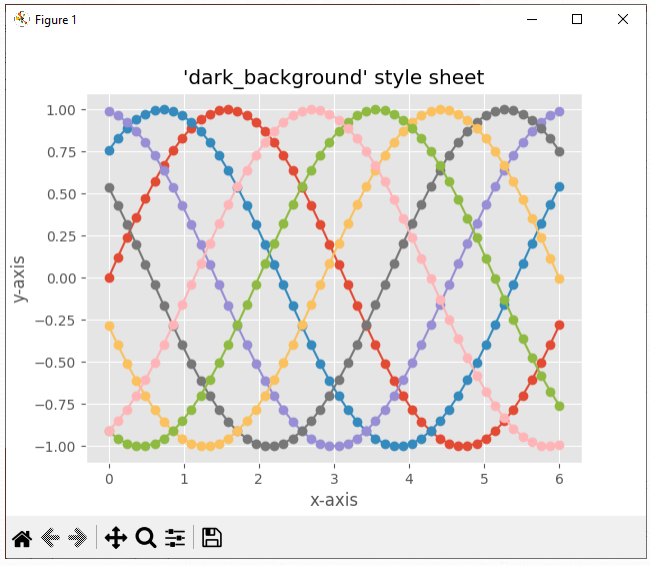



d) fivethirtyeight Bouquet



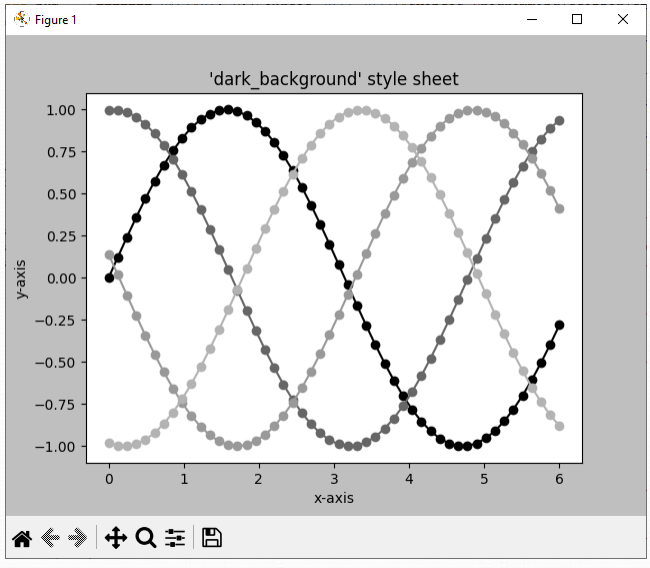
e) ggplot Bouquet

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('dark_background')
fig, ax = plt.subplots()
L = 6
x = np.linspace(0, L)
ncolors = len(plt.rcParams['axes.prop_cycle'])
shift = np.linspace(0, L, ncolors, endpoint=False)
for s in shift:
    ax.plot(x, np.sin(x + s), 'o-')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_title("'dark_background' style sheet")
plt.show()
```

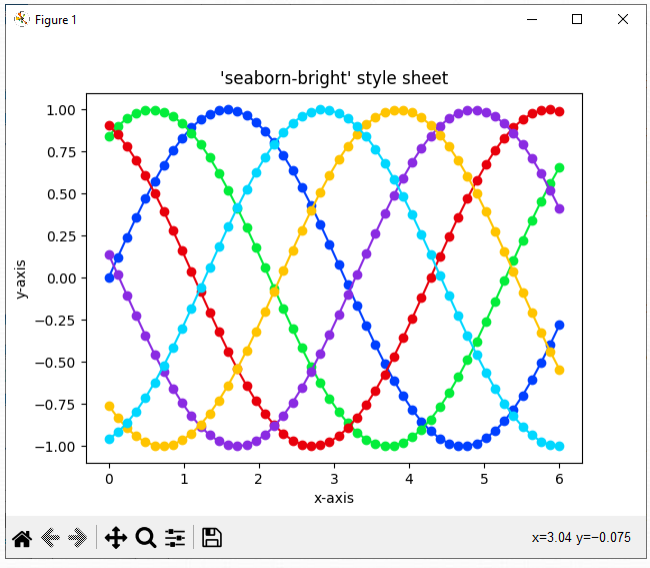



f) grayscale Bouquet

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('grayscale')
fig, ax = plt.subplots()
L = 6
x = np.linspace(0, L)
ncolors = len(plt.rcParams['axes.prop_cycle'])
shift = np.linspace(0, L, ncolors, endpoint=False)
for s in shift:
    ax.plot(x, np.sin(x + s), 'o-')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_title("'dark_background' style sheet")
plt.show()
```

g) seaborn-bright Bouquet



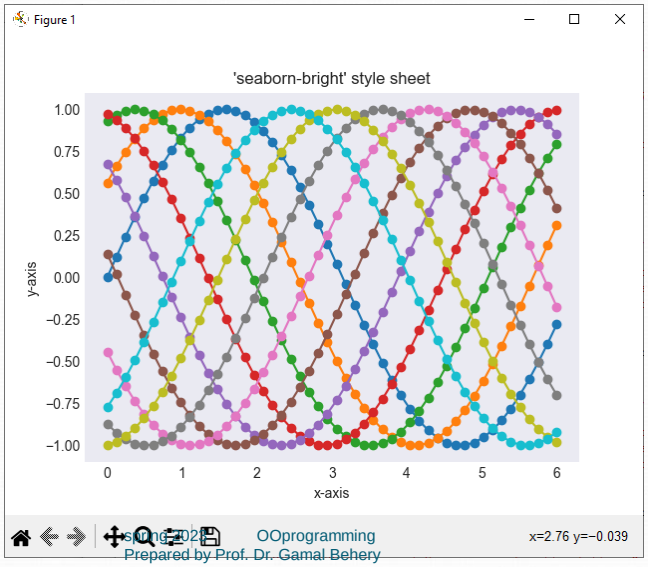
g) seaborn-bright Bouquet

169

h) seaborn-colorblind Bouquet

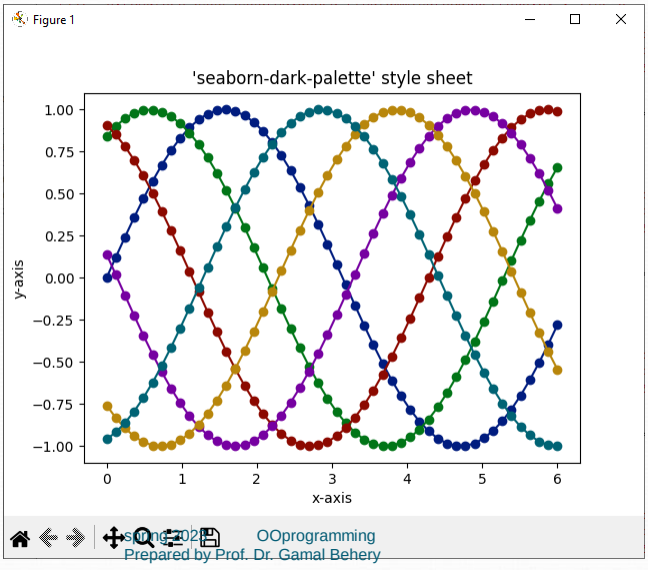
170

i) seaborn-dark Bouquet

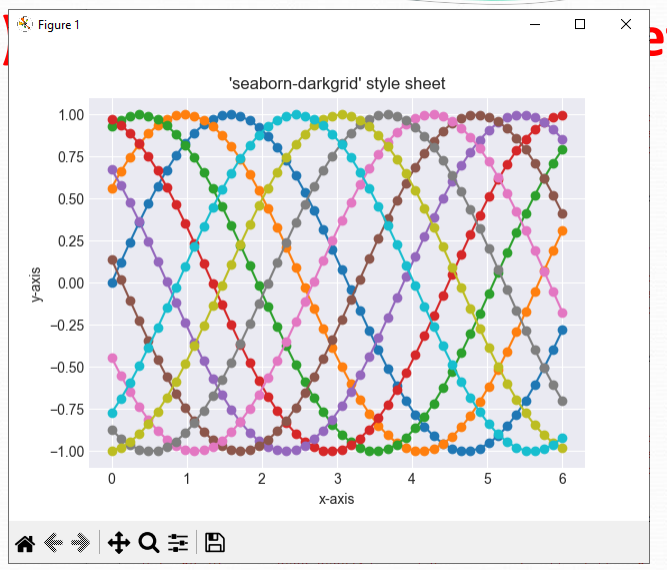


171

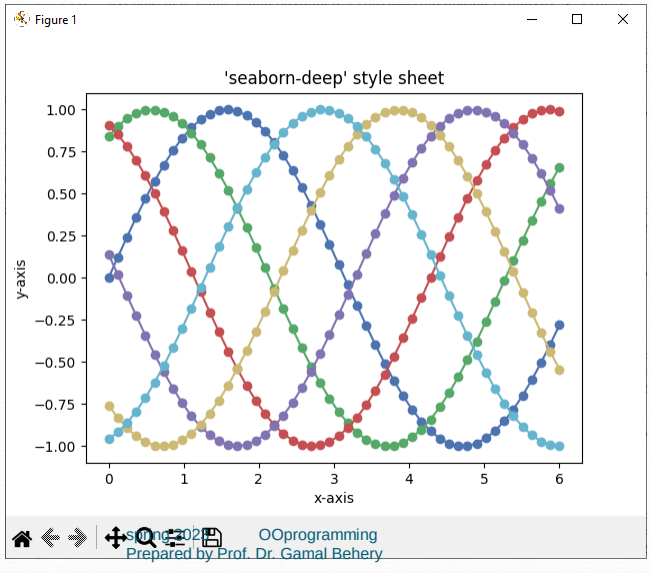
j) seaborn-dark-palette Bouquet



172



I) seaborn-deep Bouquet



m) seaborn-muted Bouquet

n) seaborn-notebook Bouquet

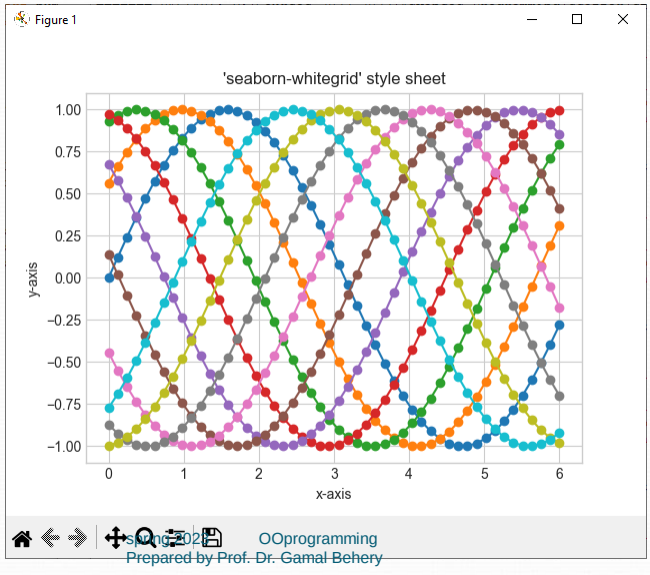
o) seaborn-paper Bouquet

q) seaborn-poster Bouquet

r) seaborn-talk Bouquet

179

u) seaborn-whitegrid Bouquet



180

2- plot branch

plt.plot()

181

Matplotlib Plotting

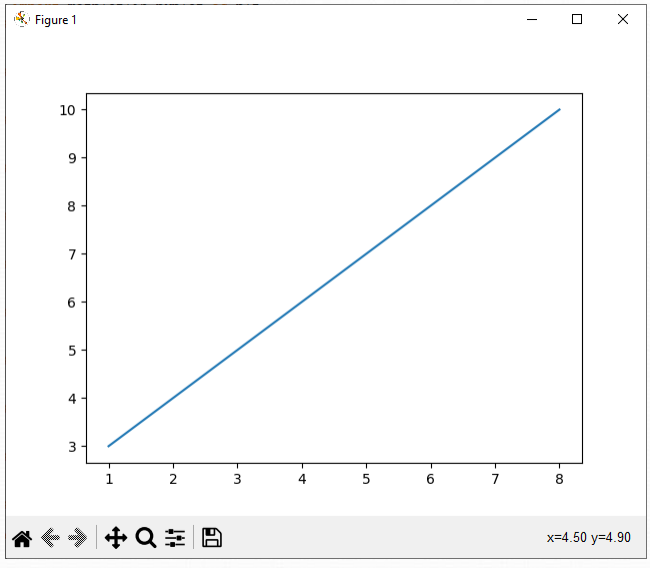
- **Plotting x and y points**
- The **plot()** function is used to draw points (markers) in a diagram.
- The function takes parameters for specifying points in the diagram.
- plot command is used to draw lines and curves provided that the data is in two dimensions
- Parameter 1 is an array containing the points on the x-axis.
- Parameter 2 is an array containing the points on the y-axis.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

182

Example-2: matplotlib .plot()

- **Draw a line in a diagram from position (1, 3) to position (8, 10):**

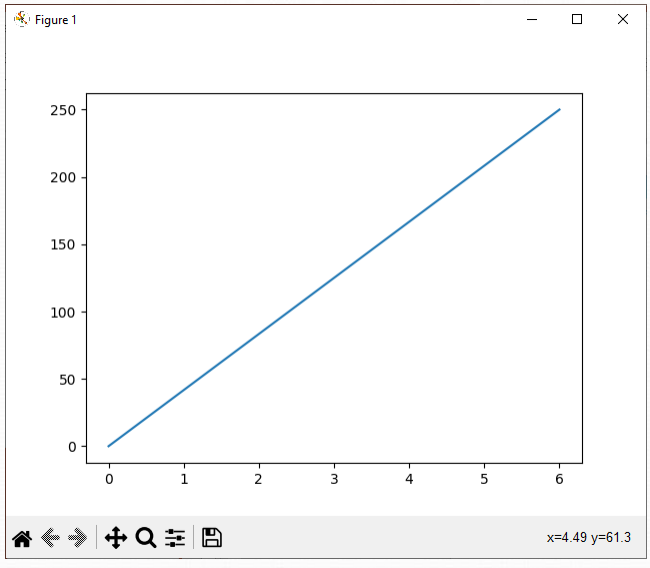
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 3])
ypoints = np.array([8, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Example-1: matplotlib.pyplot

- **Draw a line in a diagram from position (0,0) to position (6,250):**

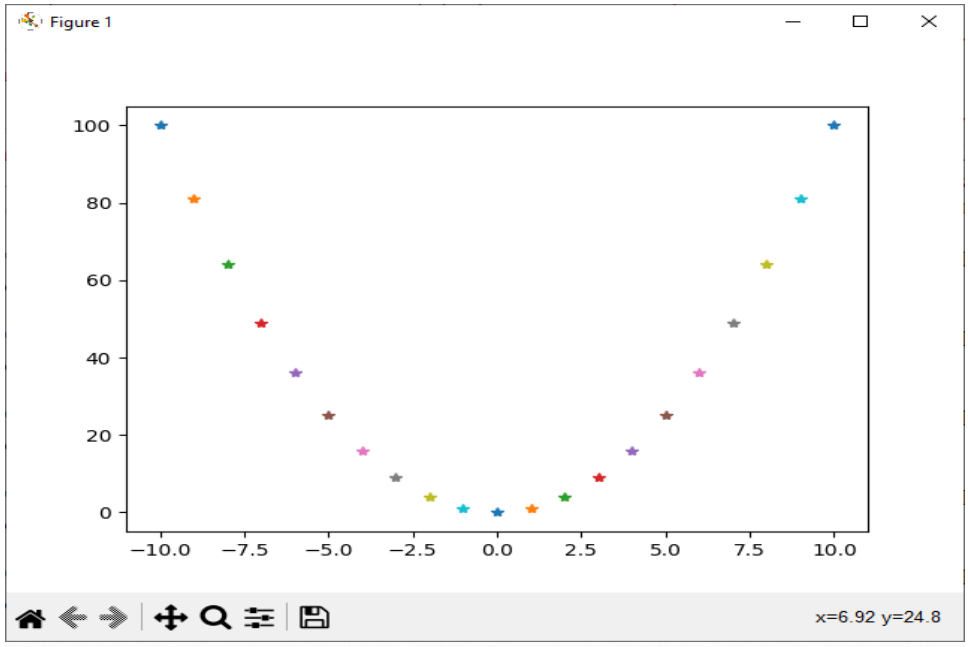
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

Example-1: matplotlib.pyplot

- **Draw a a curve in a diagram from x = -n, n-1, …,-2,-1,0,1,2, …,n and $y=x^2$.**

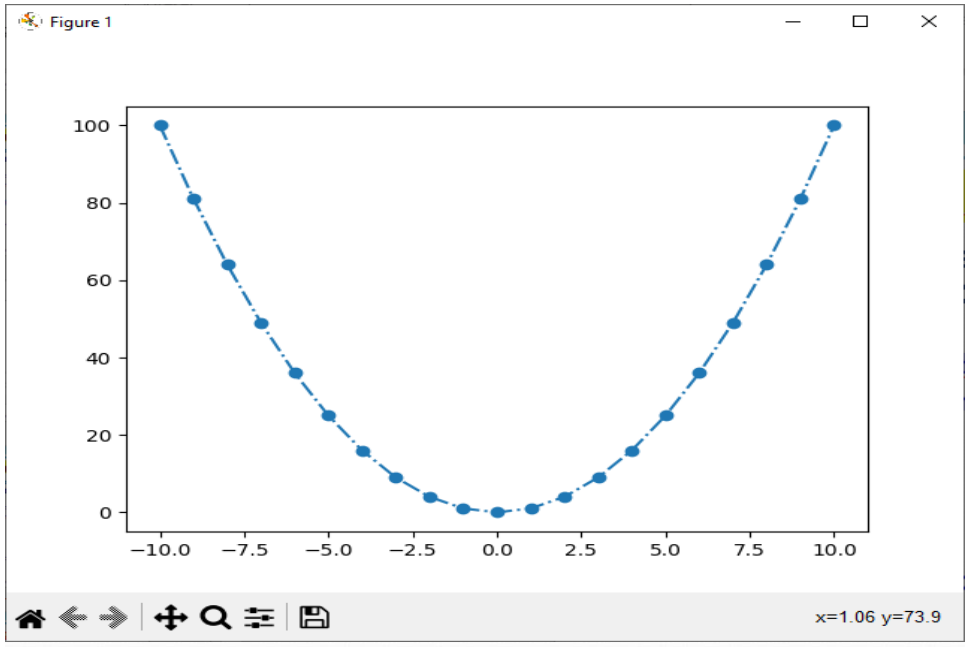
```
import matplotlib.pyplot as plt
import numpy as np
z=range(-10,11)
x = np.array([z])
y = x**2
y=np.array(y)
plt.plot(x,y,"-*")
plt.show()
```

Example-1: matplotlib.pyplot

- **Draw a a curve in a diagram from x = -n, n-1, …,-2,-1,0,1,2, …,n and $y=x^2$.**

```
import matplotlib.pyplot as plt
import numpy as np
z=range(-10,11)
x = np.array(z)
y = x**2
y=np.array(y)
plt.plot(x,y,"-.o")
plt.show()
```

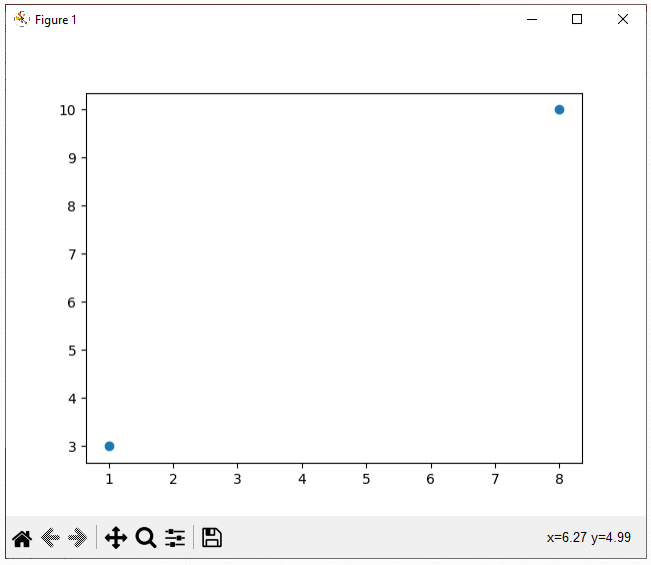
Plotting Without Line

- To plot only the markers, you can use *shortcut string notation* parameter 'o', which means **'rings'**.

Example-3: matplotlib .plot()

- **Draw two points in the diagram, one at position (1, 3) and one in position (8,10):**

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, "*o")
plt.show()
```



Multiple Points

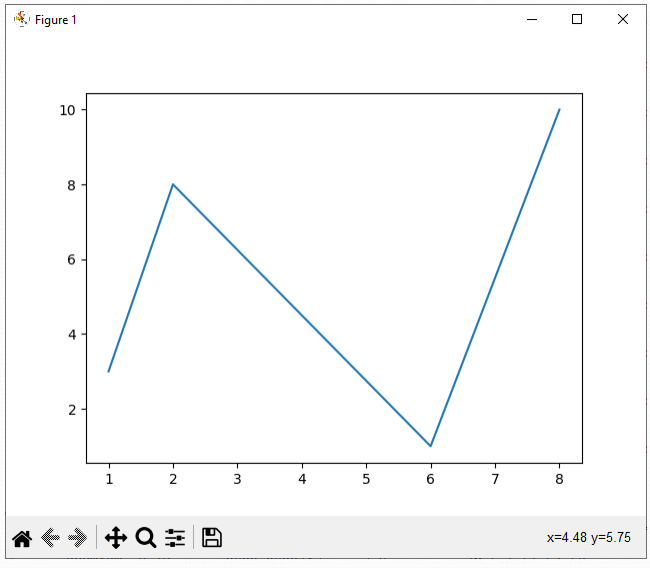
- You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example-4: matplotlib .plot()

- **Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):ات**

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()
```


195



196

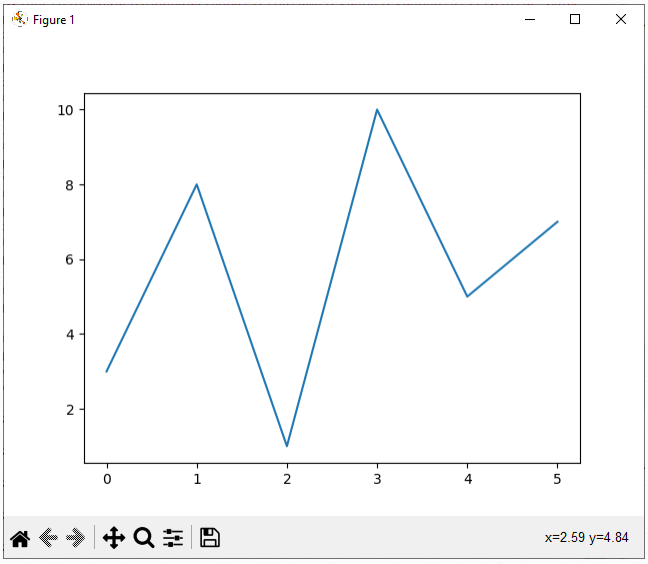
Default X-Points

- If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.
- So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example-5: matplotlib .plot()

- **Plotting without x-points:**

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()
```



199

Matplotlib Markers

200

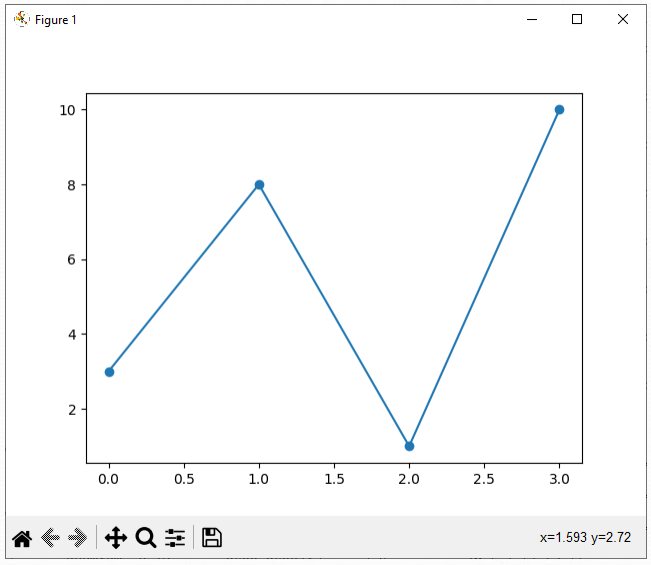
Matplotlib Markers

- You can use the keyword argument ***marker*** to emphasize each point with a specified marker:

Example-5: Matplotlib Markers

- **Mark each point with a circle:**

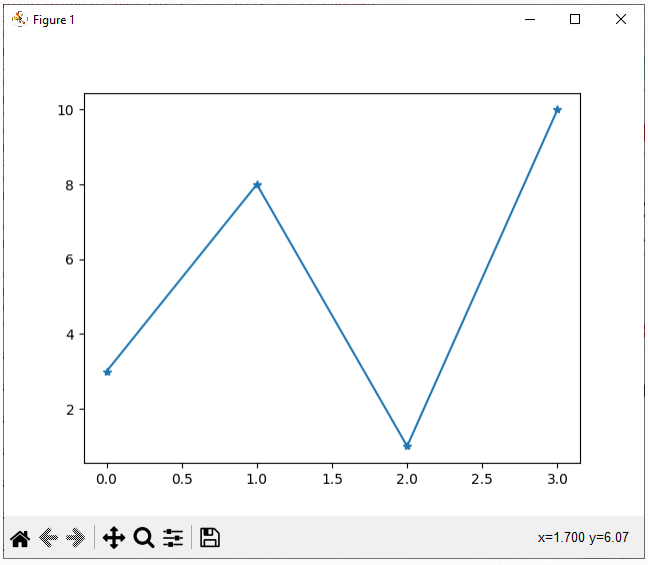
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```



Example-5: Matplotlib Markers

- **Mark each point with a star:**

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = '*')
plt.show()
```



205

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)

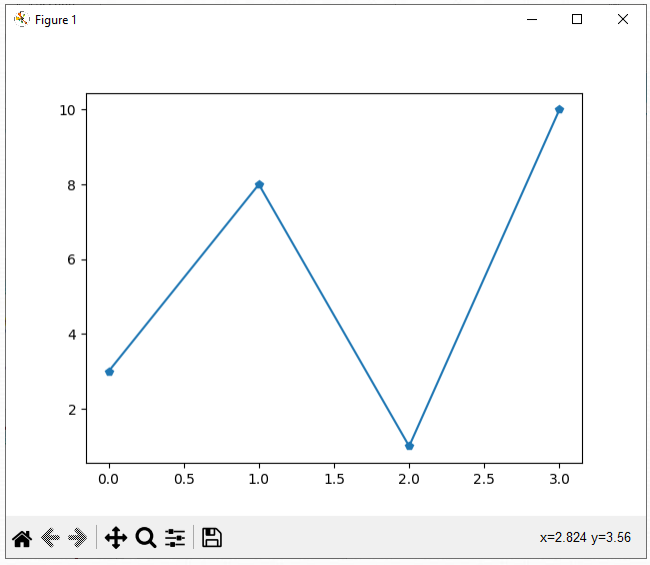
206

'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
'|'	Vline
'_'	Hline

Example-5: Matplotlib Markers

- **Mark each point with a pentagon:**

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3,8,1,10])
plt.plot(ypoints, marker = 'p')
plt.show()
```



Format Strings fmt

- We can use the shortcut string notation parameter to specify the marker.
- This parameter is also **called fmt**, and is written with this syntax:

marker|line|color

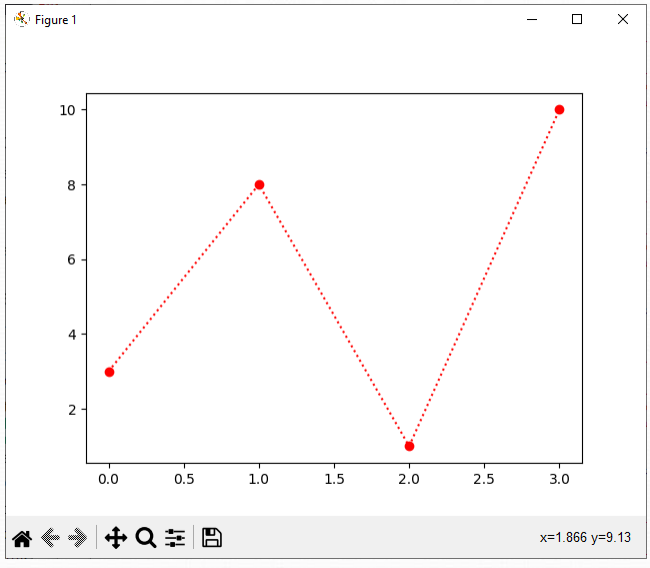
Format Strings fmt

- **Examp:**
- **Mark each point with a circle:**

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```

- **Note: r: means Red colour, o: means circle**

211



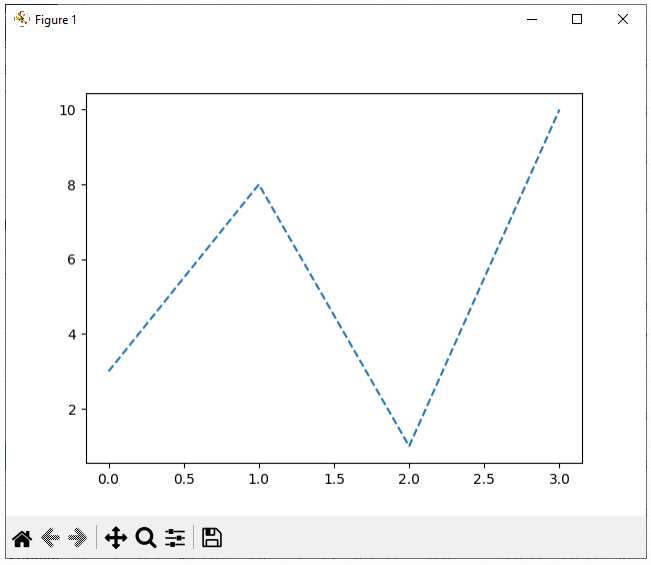
212

Line Reference

- The marker value can be anything from the Marker Reference above.
- The line value can be one of the following:

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line


```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, '--')
plt.show()
```



215

Color Reference

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

216

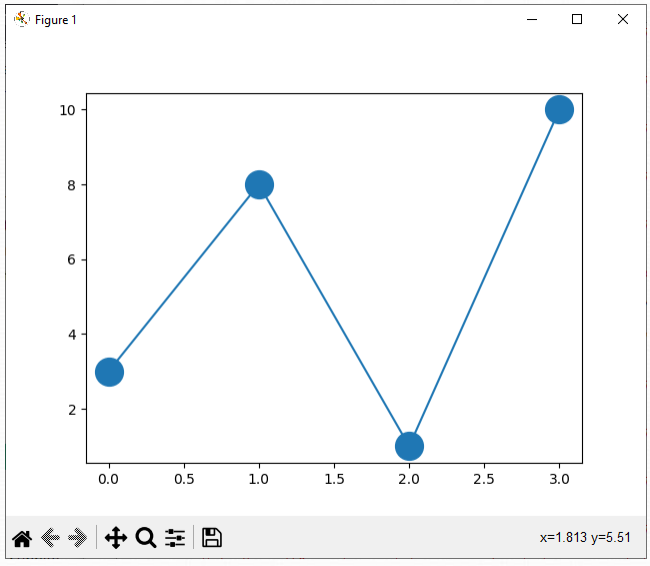
Marker Size

- You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

Example

Set the size of the markers to 20:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```

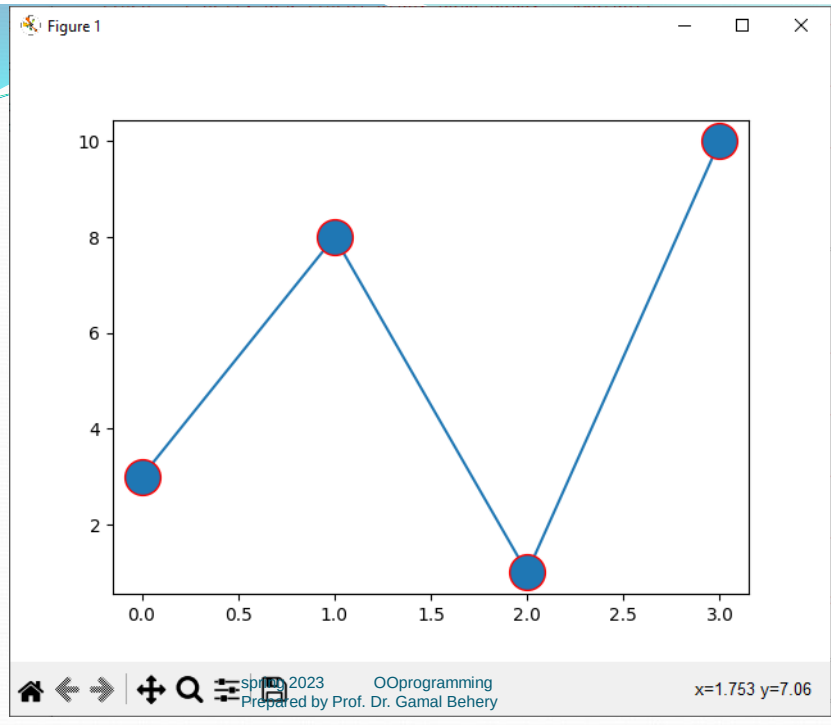
Marker Color

- You can use the keyword argument markeredgecolor or the shorter **mec** to set the color of the edge of the markers:

Example

Set the EDGE color to red:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms=20, mec='r')
plt.show()
```

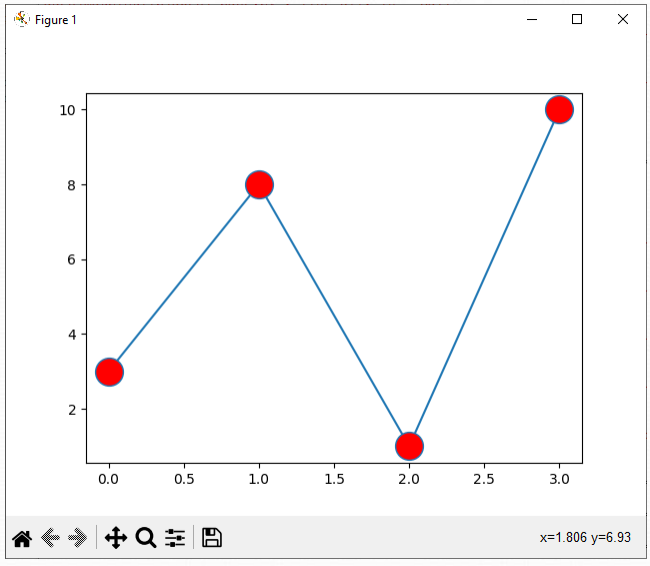



- You can use the keyword argument **markerfacecolor** or the shorter **mfc** to set the color inside the edge of the markers:

Example

Set the FACE color to red:

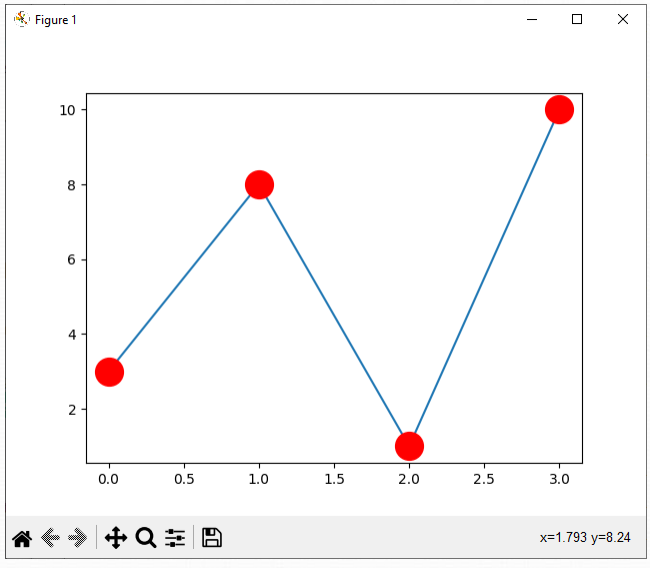
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mfc='r')
plt.show()
```

Use both the **mec** and **mfc** arguments to color the entire marker:

Example
Set the color of both the *edge* and the *face* to red:

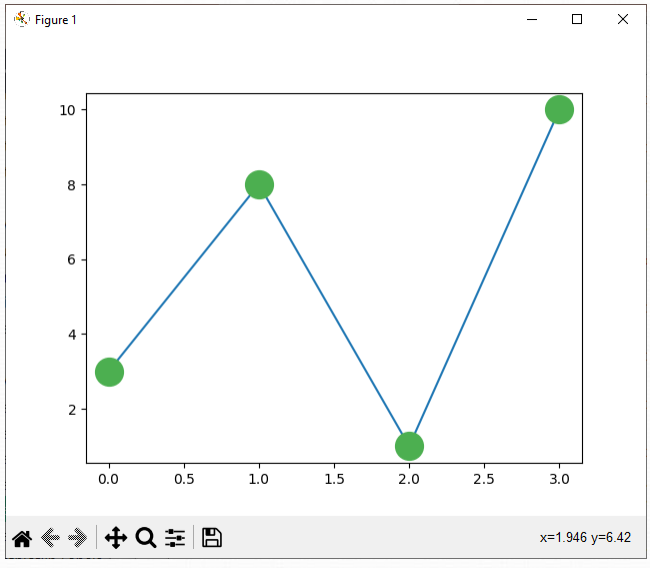
```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
plt.show()
```

Use both the **mec** and **mfc** arguments to color the entire marker:

Example
Set the color of both the *edge* and the *face* to red:

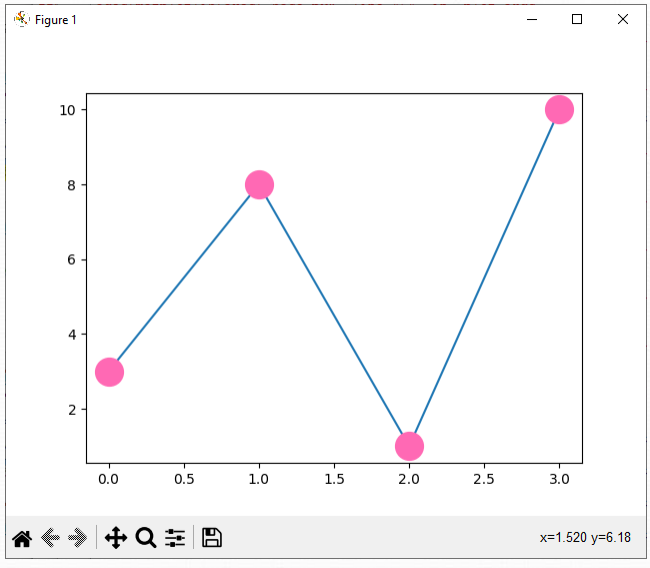
```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
plt.show()
```

Use both the **mec** and **mfc** arguments to color the entire marker:

Example
Set the color of both the *edge* and the *face* to red:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec
= 'hotpink', mfc = 'hotpink')
plt.show()
```

Matplotlib Line

229

Linestyle

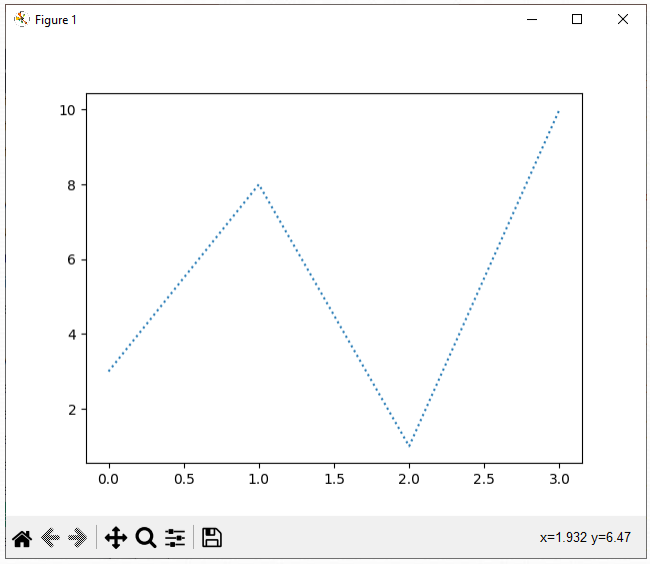
- You can use the keyword argument linestyle, or **shorter ls**, to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```


230



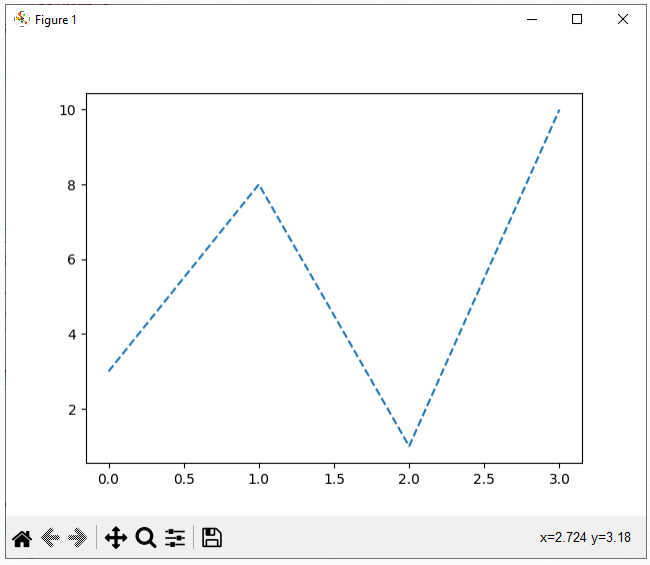
Linestyle

- You can use the keyword argument linestyle, or **shorter ls**, to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dashed')
plt.show()
```



Shorter Syntax

The linestyle can be written in a shorter syntax:
`linestyle` can be written as `ls`.
`dotted` can be written as `:`.
`dashed` can be written as `--`.

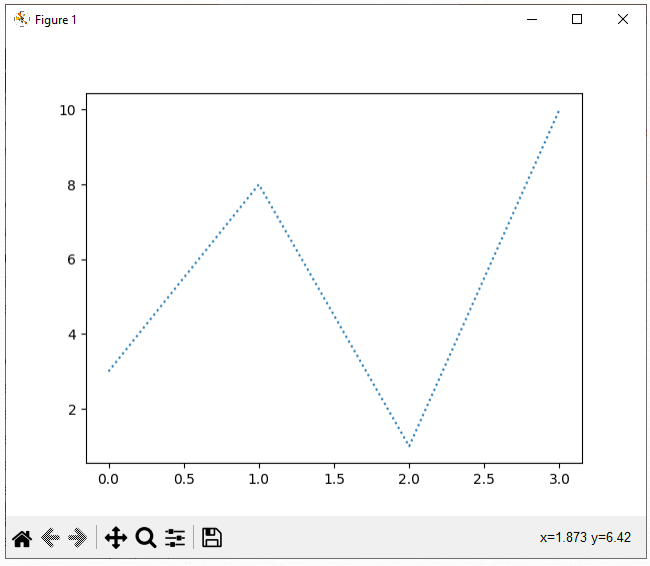
Linestyle

- You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, ls = ':')
plt.show()
```

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

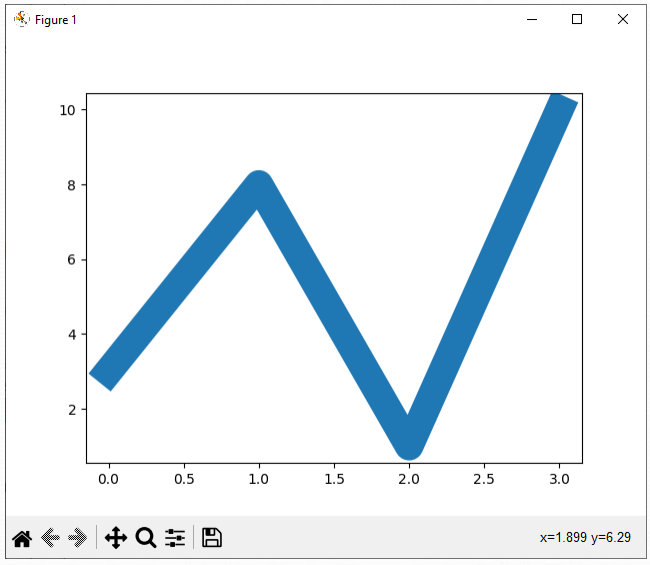
Line Width

You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line. The value is a floating number, in points:

Example

Plot with a 20.5pt wide line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linewidth = '20.5')
plt.show()
```



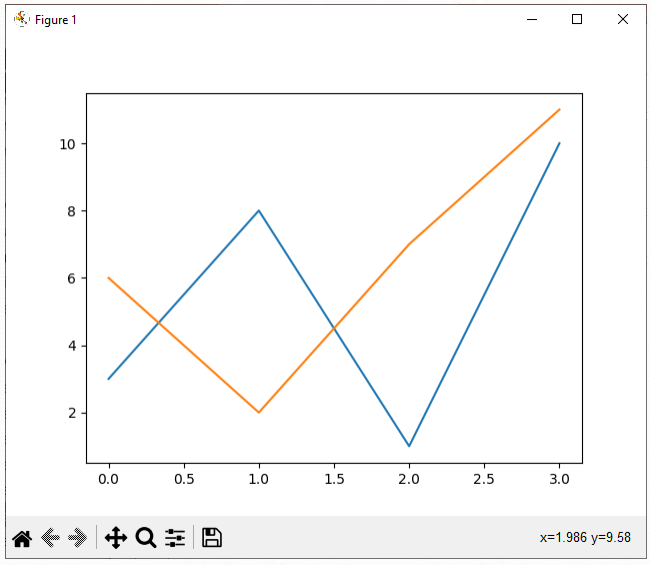
Multiple Lines

You can plot as many lines as you like by simply adding more `plt.plot()` functions:

Example

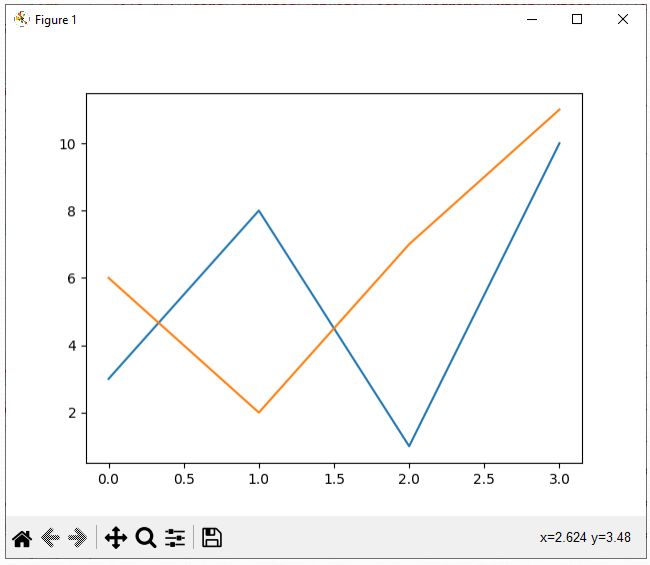
Draw two lines by specifying a `plt.plot()` function for each line:

```
import matplotlib.pyplot as plt
import numpy as np
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])
plt.plot(y1)
plt.plot(y2)
plt.show()
```



Example
Draw two lines by specifiyng the x- and y-point values for both lines:

```
import matplotlib.pyplot as plt
import numpy as np
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])
plt.plot(x1, y1, x2, y2)
plt.show()
```



Matplotlib
Labels and Title

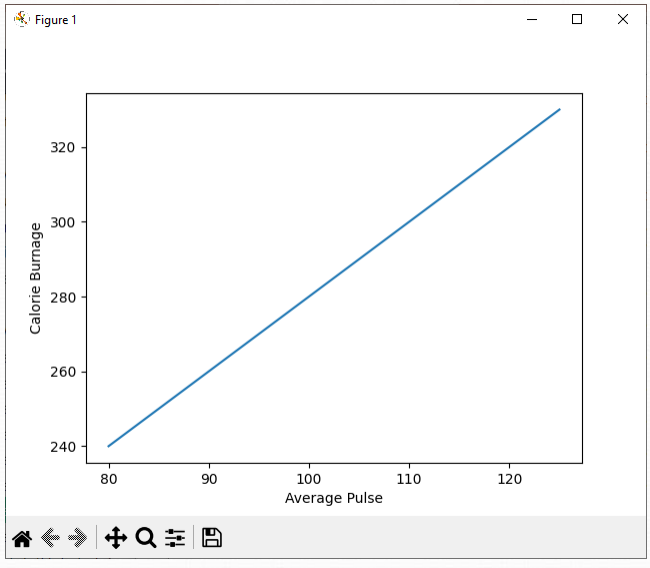
Create Labels for a Plot

- With Pyplot, you can use the ***xlabel()*** and ***ylabel()*** functions to set a label for the x- and y-axis.

Example

Add labels to the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115,
              120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300,
              310, 320, 330])
plt.plot(x, y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

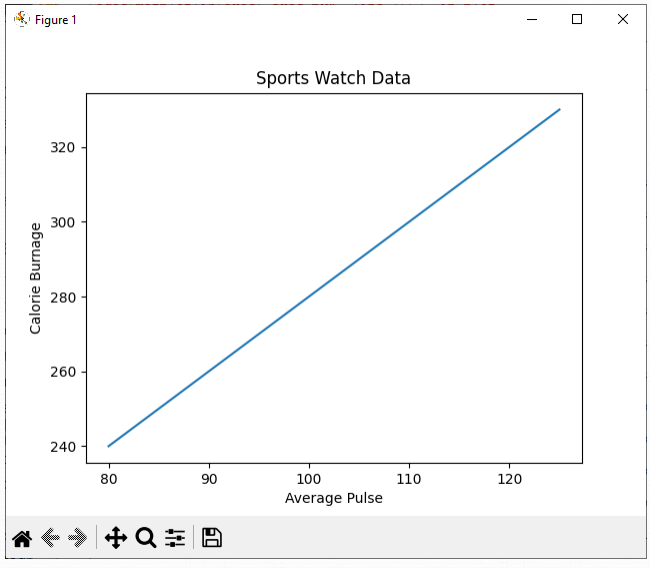
Create a Title for a Plot

With Pyplot, you can use the `title()` function to set a title for the p

Example

Add a plot title and labels for the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

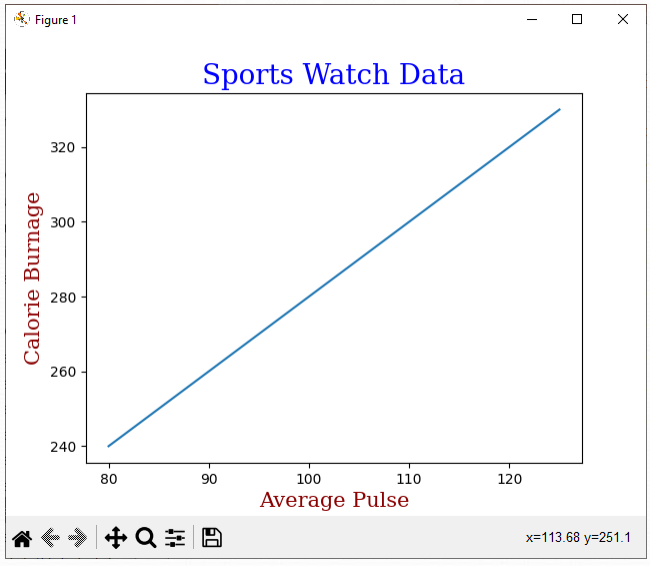
Set Font Properties for Title and Labels

You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels.

Example

Set font properties for the title and labels:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)
plt.plot(x, y)
plt.show()
```

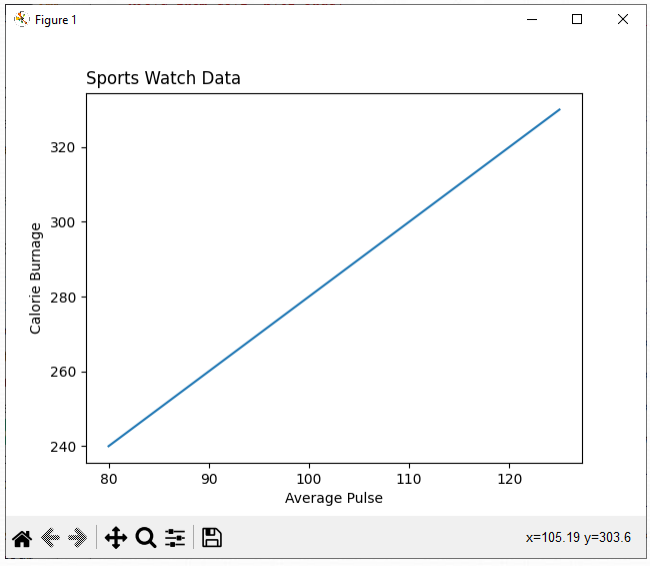
Position the Title

You can use the `loc` parameter in `title()` to position the title. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

Example

Position the title to the left:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.show()
```

Matplotlib
Adding Grid Lines

253

Add Grid Lines to a Plot

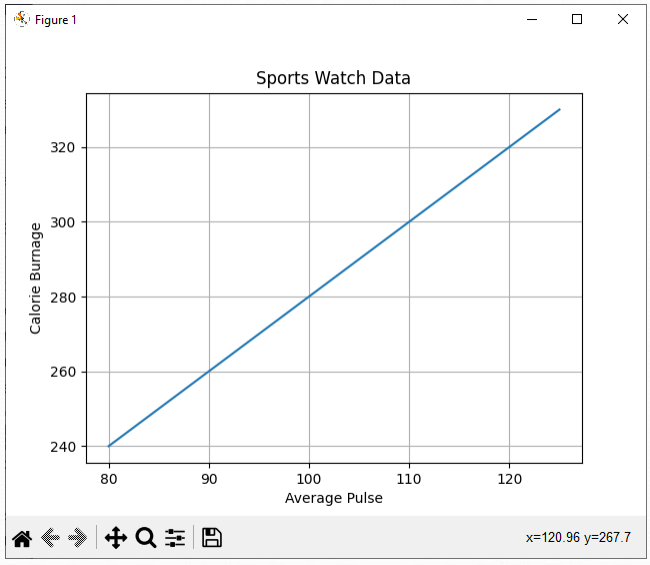
- With Pyplot, you can use the ***grid()*** function to add grid lines to the plot.

Example

Add grid lines to the plot:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid()
plt.show()
```


254



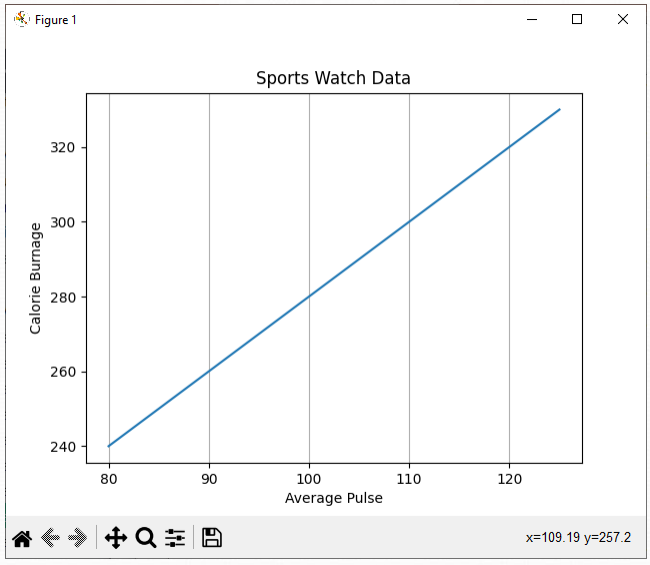
Specify Which Grid Lines to Display

You can use the `axis` parameter in the `grid()` function to specify which grid lines to display. Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

Example

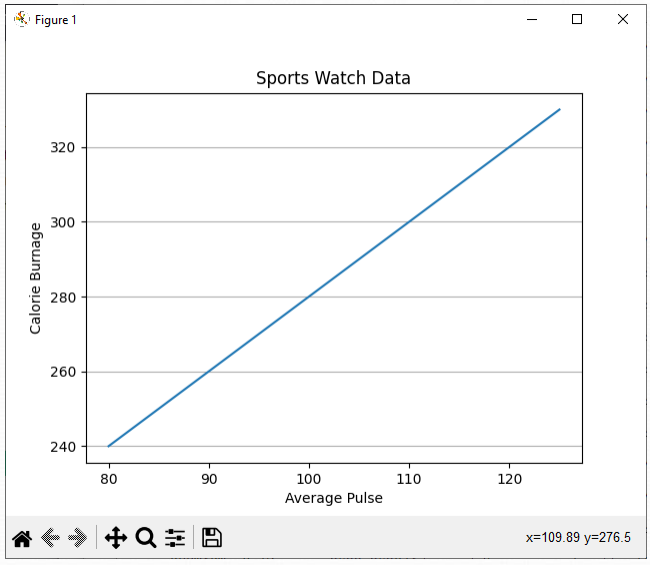
Display only grid lines for the x-axis:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid(axis = 'x')
plt.show()
```



Example
Display only grid lines for the y-axis:

```
import numpy as np
import matplotlib.pyplot as plt
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120,
 125])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310,
320, 330])
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid(axis = 'y')
plt.show()
```



259

Matplotlib Subplot

260

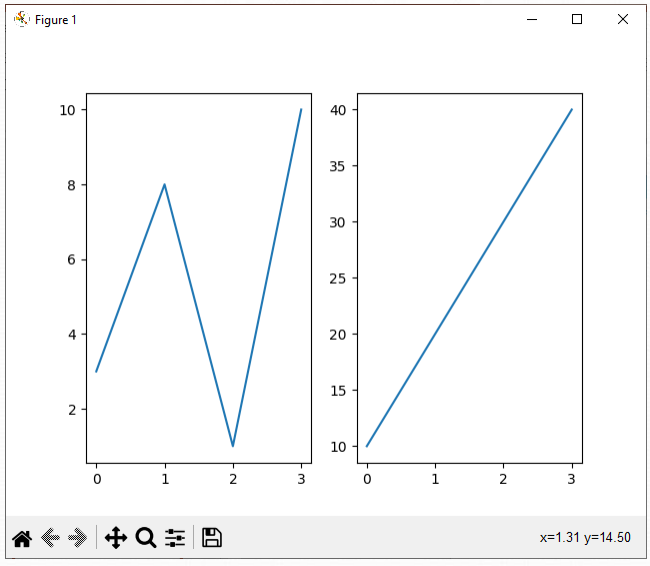
Display Multiple Plots

With the `subplot()` function you can draw multiple plots in one figure:

Example

Draw 2 plots:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```

The subplot() Function

- The ***subplot()*** function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the first and second argument.
- The third argument represents the index of the current plot.

The subplot() Function

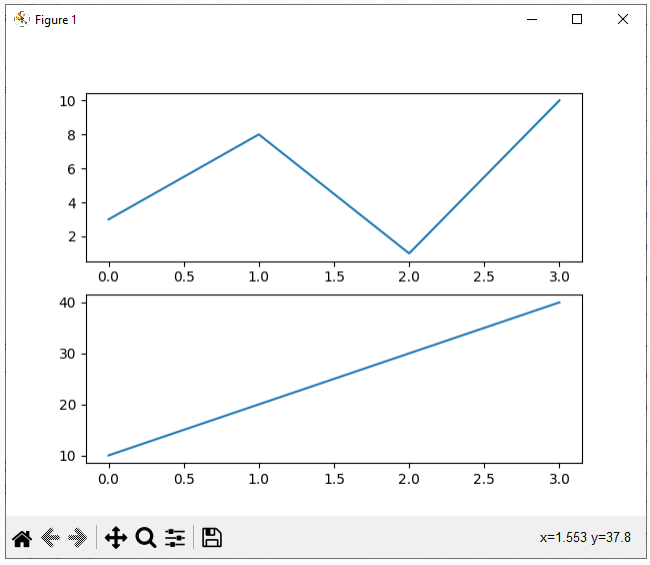
```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this
plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this
plot is the second plot.
```

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:
Example.

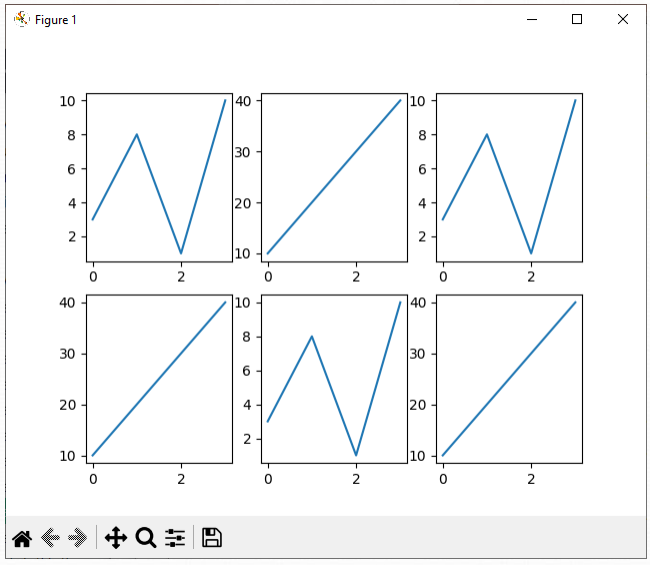
Example
Draw 2 plots on top of each other:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 1, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 1, 2)
plt.plot(x,y)
plt.show()
```



Example
Draw 6 plots:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 1)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 2)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 3)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 4)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 5)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 6)
plt.plot(x,y)
plt.show()
```

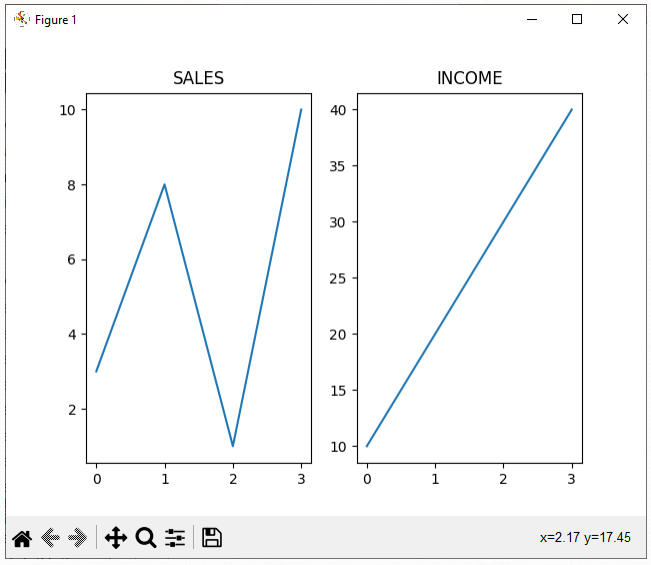


269

Example
2 plots, with titles:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
plt.show()
```


270



Super Title

- You can add a title to the entire figure with the ***suptitle()*** function:

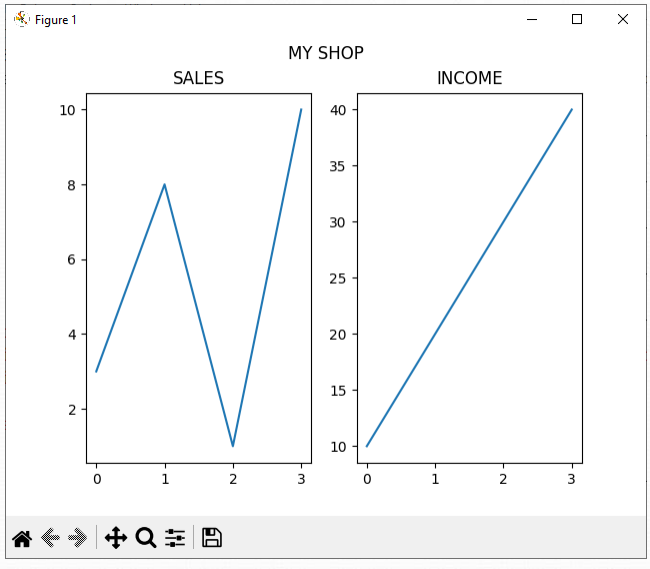
Example

Add a title for the entire figure:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
```

Super Title

```
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
plt.suptitle("MY SHOP")
plt.show()
```

Matplotlib Scatter

275

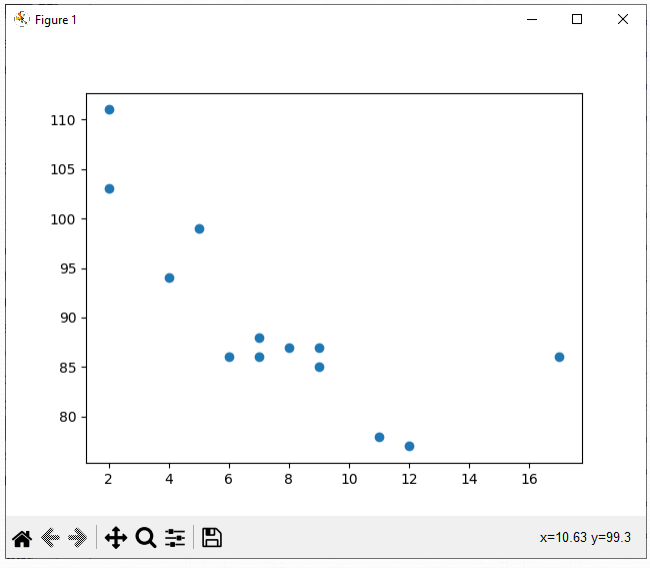
Creating Scatter Plots

- With Pyplot, you can use the ***scatter()*** function to draw a scatter plot.
- The ***scatter()*** function plots one dot for each observation.
- It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

276

Example
A simple scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x =
np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78
,77,85,86])

plt.scatter(x, y)
plt.show()
```

Compare Plots

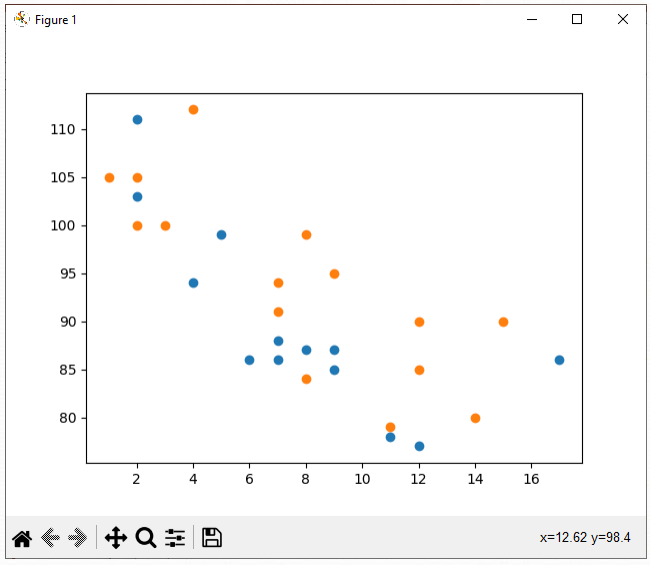
Example

Draw two plots on the same figure:

```
import matplotlib.pyplot as plt
import numpy as np
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77
,85,86])
plt.scatter(x, y)
#day two, the age and speed of 15 cars:
```


Compare Plots

```
x =
np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y =
np.array([100,105,84,105,90,99,90,95,94,100,79
,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



281

Colors

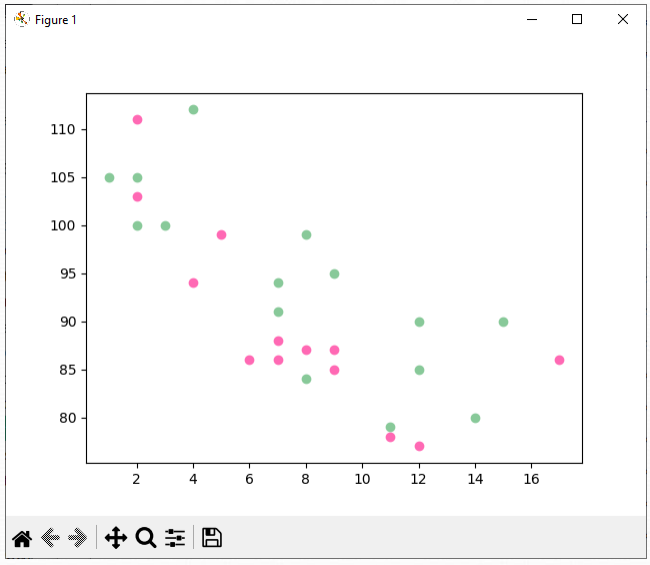
You can set your own color for each scatter plot with the `color` or the `c` argument:

Example

Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y =
np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')
plt.show()
```


282

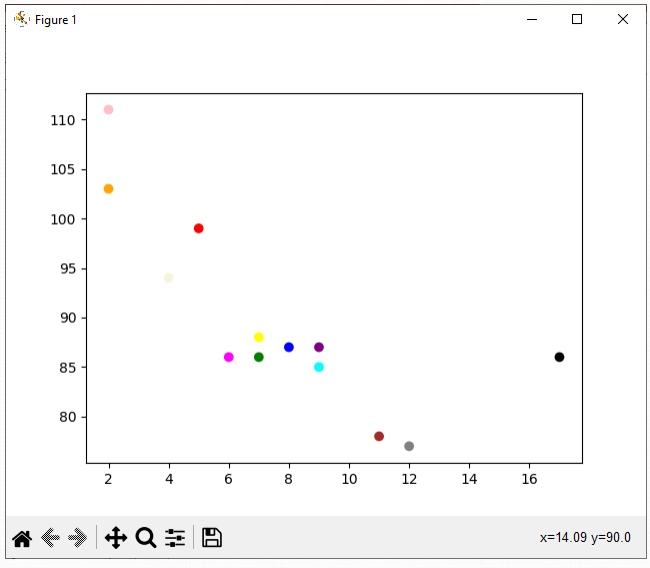


Color Each Dot

- You can even set a specific color for each dot by using an array of colors as value for the c argument:
- Note: You cannot use the **color** argument for this, only the **c** argument.
- **Example**
- **Set your own color of the markers:**

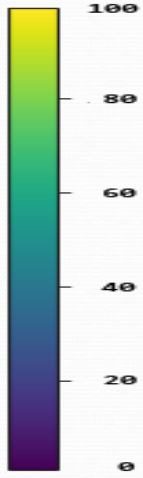
Color Each Dot

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,
85,86])
colors =
np.array(["red","green","blue","yellow","pink
","black","orange","purple","beige","brown","
gray","cyan","magenta"])
plt.scatter(x, y, c=colors)
plt.show()
```

ColorMap

- The Matplotlib module has a number of available colormaps.
- A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
- This colormap is called '*viridis*' and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.
- Here is an example of a colormap:



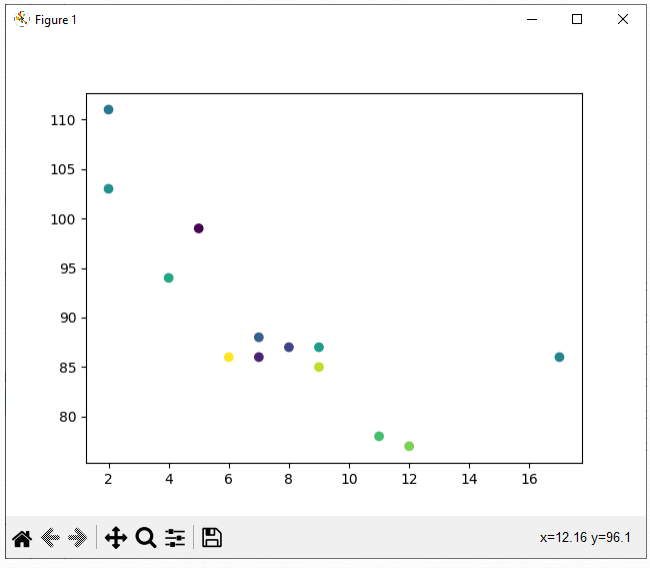
How to Use the ColorMap

- You can specify the colormap with the keyword argument *cmap* with the value of the colormap, in this case '*viridis*' which is one of the built-in colormaps available in Matplotlib.
- In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

Example

Create a color array, and specify a colormap in the scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,86
])
colors =
np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 8
0, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis')
plt.show()
```

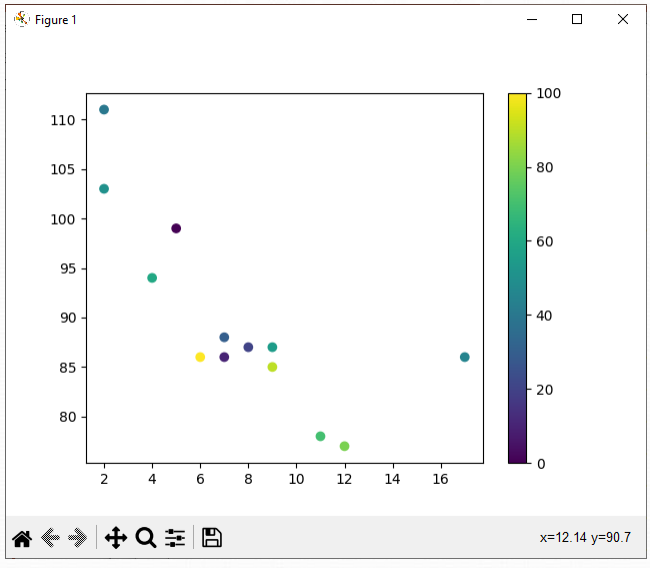



You can include the colormap in the drawing by including the *plt.colorbar()* statement:

Example

Include the actual colormap:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 9
0, 100])
plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
```

Available ColorMaps

- You can choose any of the built-in colormaps:

Name	Reverse
Accent	Accent_r
Blues	Blues_r
BrBG	BrBG_r
BuGn	BuGn_r
BuPu	BuPu_r
CMRmap	CMRmap_r

Dark2	Dark2_r
GnBu	GnBu_r
Greens	Greens_r
Greys	Greys_r
OrRd	OrRd_r
Oranges	Oranges_r
PRGn	PRGn_r

Paired	Paired_r
Pastel1	Pastel1_r
Pastel2	Pastel2_r
PiYG	PiYG_r
PuBu	PuBu_r
PuBuGn	PuBuGn_r
PuOr	PuOr_r

PuRd	PuRd_r
Purples	Purples_r
RdBu	RdBu_r
RdGy	RdGy_r
RdPu	RdPu_r
RdYlBu	RdYlBu_r
RdYlGn	RdYlGn_r

Reds	Reds_r
Set1	Set1_r
Set2	Set2_r
Set3	Set3_r
Spectral	Spectral_r
Wistia	Wistia_r

YlGn	YlGn_r
YlGnBu	YlGnBu_r
YlOrBr	YlOrBr_r
YlOrRd	YlOrRd_r
afmhot	afmhot_r
autumn	autumn_r

binary	binary_r
bone	bone_r
brg	brg_r
bwr	bwr_r
cividis	cividis_r
cool	cool_r
coolwarm	coolwarm_r
copper	copper_r
cubehelix	cubehelix_r

flag	flag_r
gist_earth	gist_earth_r
gist_gray	gist_gray_r
gist_heat	gist_heat_r
gist_ncar	gist_ncar_r
gist_rainbow	gist_rainbow_r
gist_stern	gist_stern_r
gist_yarg	gist_yarg_r
gnuplot	gnuplot_r

gnuplot2	gnuplot2_r
gray	gray_r
hot	hot_r
hsv	hsv_r
inferno	inferno_r
jet	jet_r
magma	magma_r
nipy_spectral	nipy_spectral_r
ocean	ocean_r
pink	pink_r
plasma	plasma_r

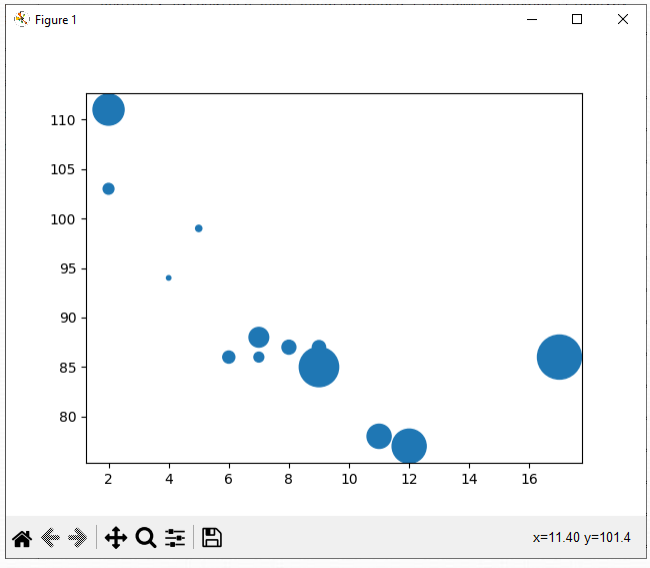
rism	prism_r
rainbow	rainbow_r
seismic	seismic_r
spring	spring_r
summer	summer_r
tab10	tab10_r
tab20	tab20_r
tab20b	tab20b_r
tab20c	tab20c_r
terrain	terrain_r
twilight	twilight_r
twilight_shifted	twilight_shifted_r

viridis	viridis_r
winter	winter_r

Example
Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,8
6])
sizes
= np.array([20,50,100,200,500,1000,60,90,10,300,6
00,800,75])

plt.scatter(x, y, s=sizes)

plt.show()
```


299



300

Matplotlib Bars

301

Creating Bars

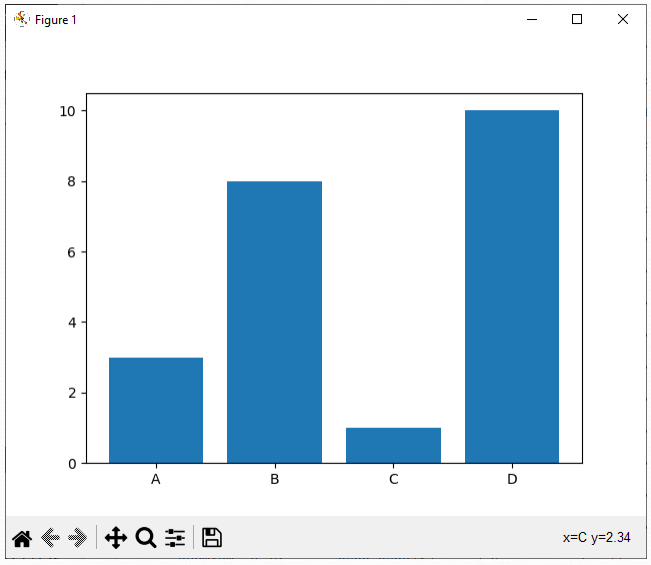
With **Pyplot**, you can use the ***bar()*** function to draw bar graphs:

Example

Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```


302



303

Horizontal Bars

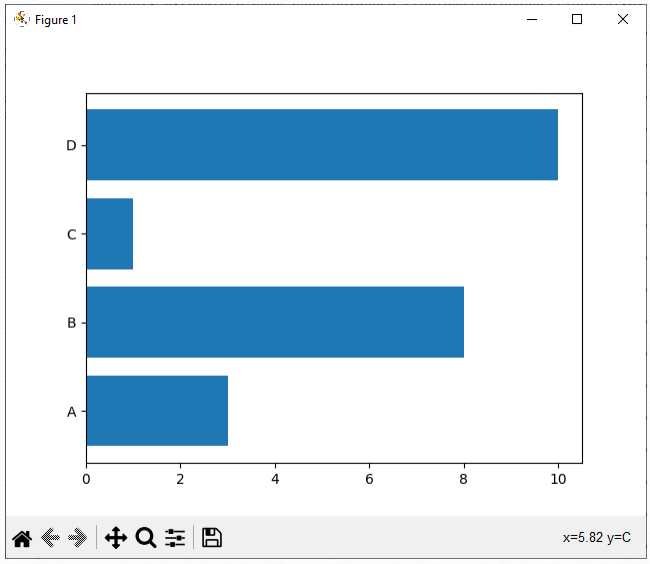
If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function:

Example

Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```


304



305

Bar Color

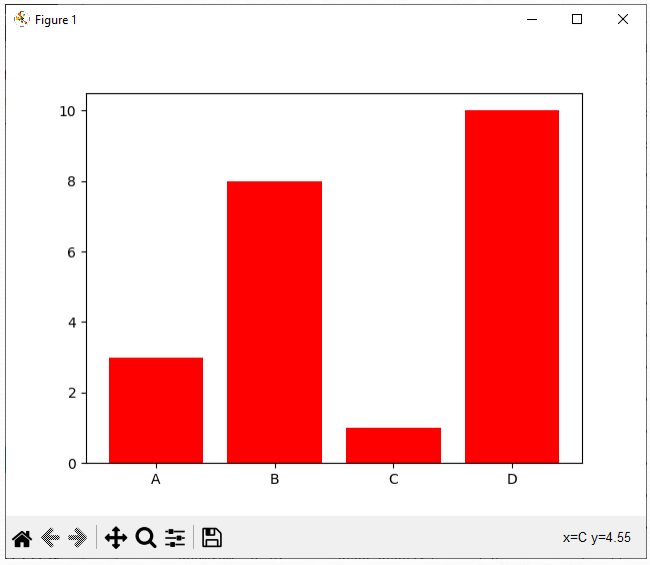
The `bar()` and `barh()` take the keyword argument `color` to set the color of the bars:

Example

Draw 4 red bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = "red")
plt.show()
```


306



307

Bar Color

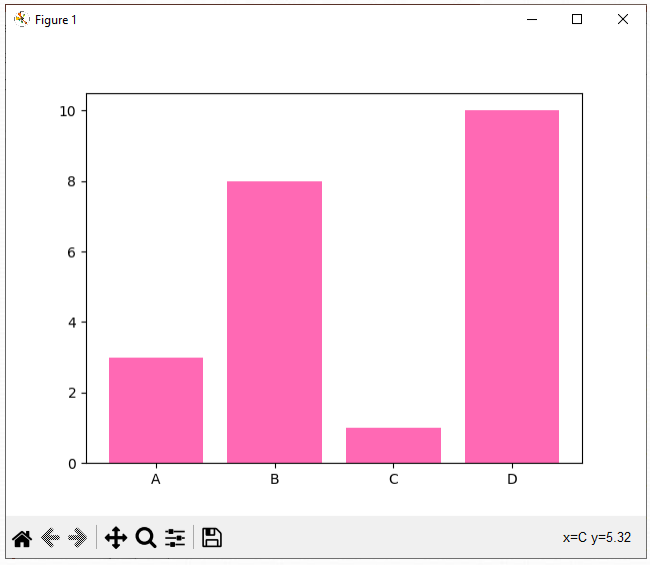
The `bar()` and `barh()` take the keyword argument `color` to set the color of the bars:

Example

Draw 4 red bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = "hotpink")
plt.show()
```


308



309

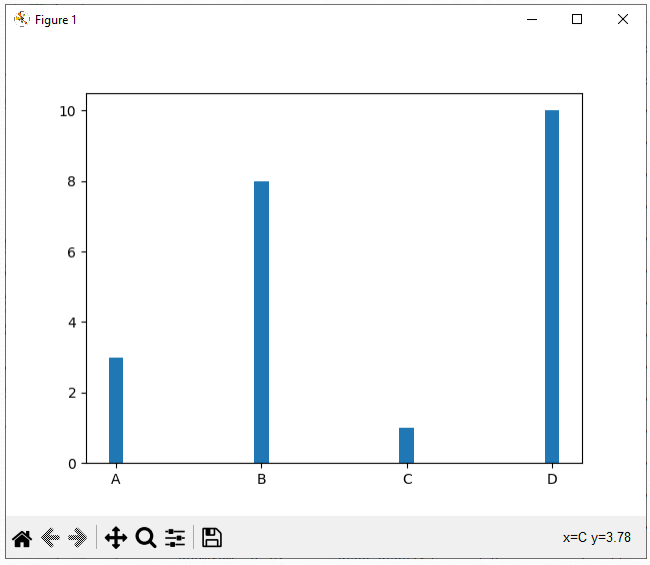
Bar Width

The `bar()` takes the keyword argument ***width*** to set the width of the bars:

Example
Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```


310



311

Bar Height

The `barh()` takes the keyword argument `height` to set the height of the bars:

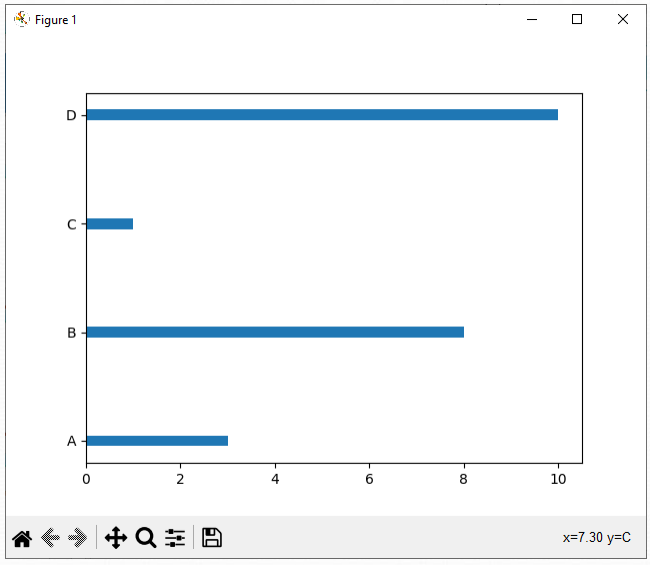
Example

Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```


312

The default height value is 0.8



313

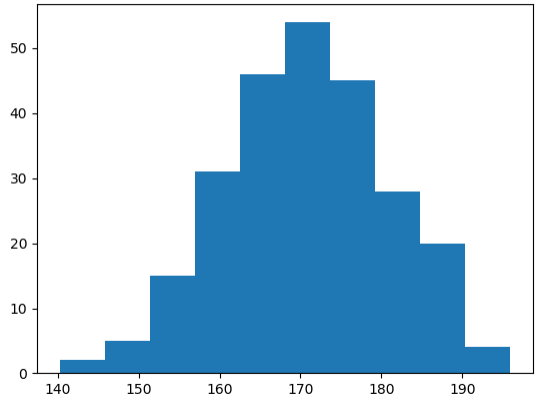
Matplotlib Histograms

314

Histogram

- A histogram is a showing graph ***frequency*** distributions
- It is a graph showing the number of observations within each given interval.
- Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

315



316

Create Histogram

- In Matplotlib, we use the ***hist()*** function to create histograms.
- The ***hist()*** function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- For simplicity we use **NumPy** to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

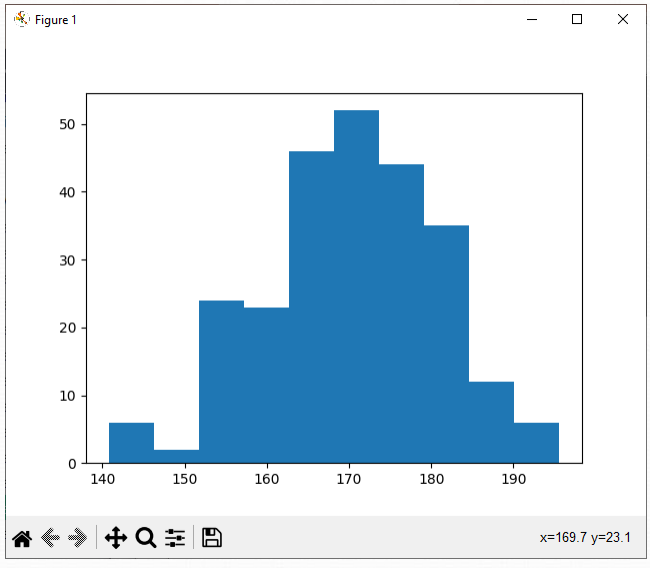
Example
A Normal Data Distribution by NumPy:

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```

The hist() function will read the array and produce a histogram:
Example
A simple histogram:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```


319



320

Matplotlib Pie Charts

321

Creating Pie Charts

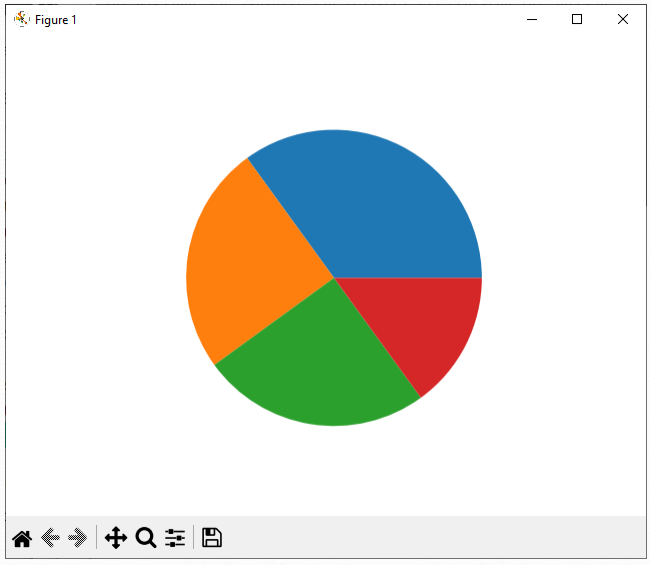
With Pyplot, you can use the ***pie()*** function to draw pie charts:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```


322



323

Labels

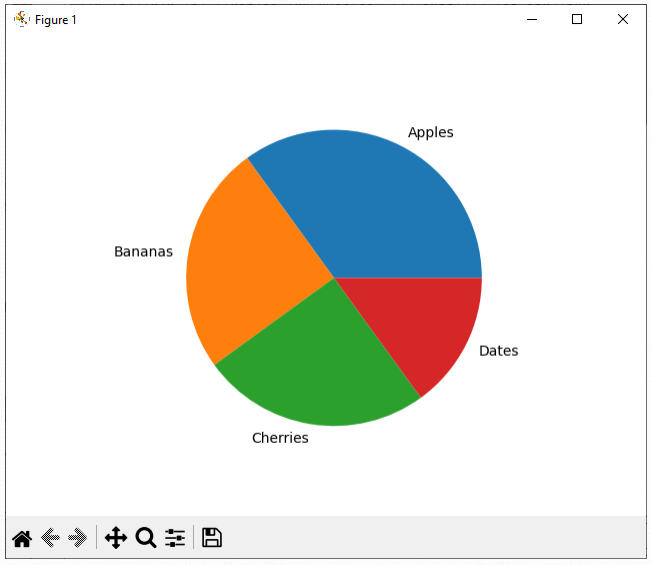
Add labels to the pie chart with the `label` parameter.
The `label` parameter must be an array with one label for each wedge:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.show()
```

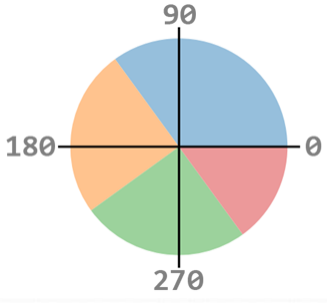

324



325

Start Angle

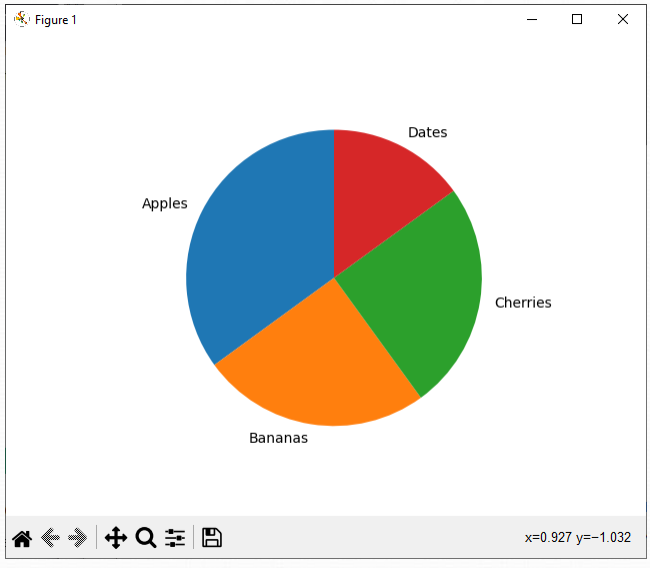
As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a `startangle` parameter.
The `startangle` parameter is defined with an angle in degrees, default angle is 0:



326

Example
Start the first wedge at 90 degrees:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels, startangle
= 90)
plt.show()
```

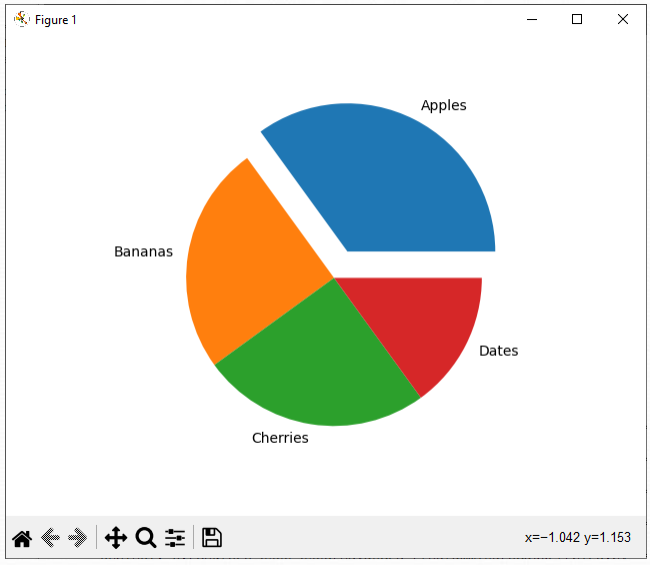
Explode

- Maybe you want one of the wedges to stand out?
- The `explode` parameter allows you to do that.
- The `explode` parameter, if specified, and not `None`, must be an array with one value for each wedge.
- Each value represents how far from the center each wedge is displayed:

Example

Pull the "Apples" wedge 0.2 from the center of the pie:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode =
myexplode)
plt.show()
```



331

Shadow

Add a shadow to the pie chart by setting the **shadows** parameter to **True**:

Example

Add a shadow:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode,
shadow = True)
plt.show()
```


332

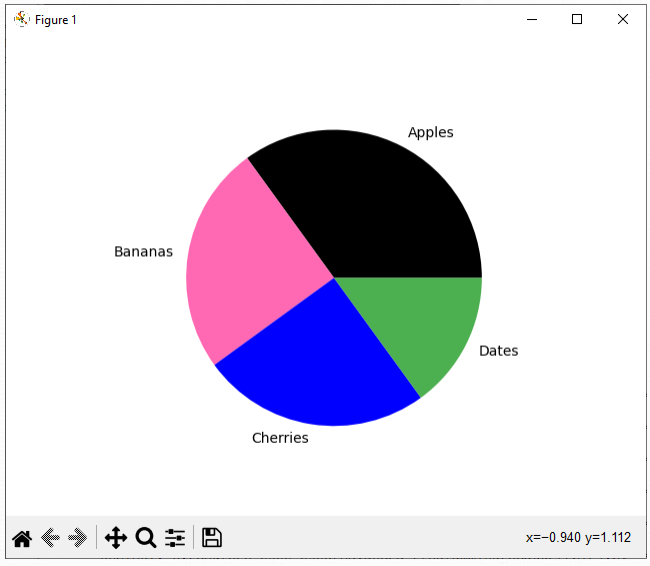
Colors

- You can set the color of each wedge with the colors parameter. The **colors** parameter, if specified, must be an array with one value for each wedge:

Example

Specify a new color for each wedge:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:

'r' - Red
'g' - Green
'b' - Blue
'c' - Cyan
'm' - Magenta
'y' - Yellow
'k' - Black
'w' - White

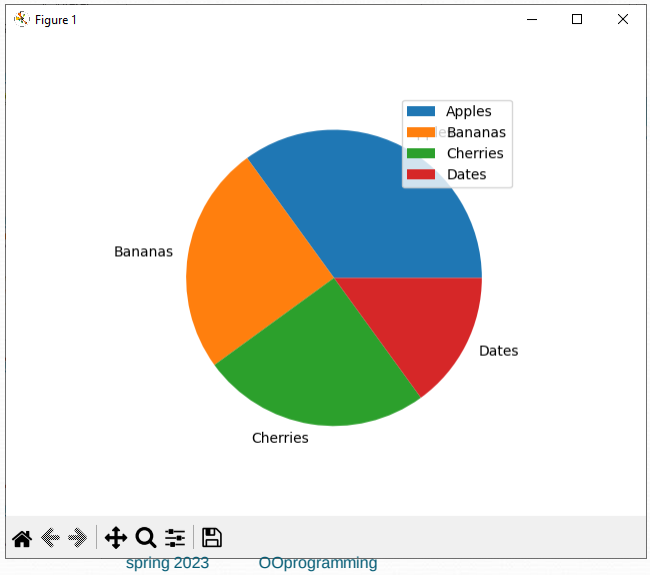
Legend

To add a list of explanation for each wedge, use the ***legend()*** function:

Example

Add a legend:

```
import matplotlib.pyplot as plt
port numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```



337

Legend With Header

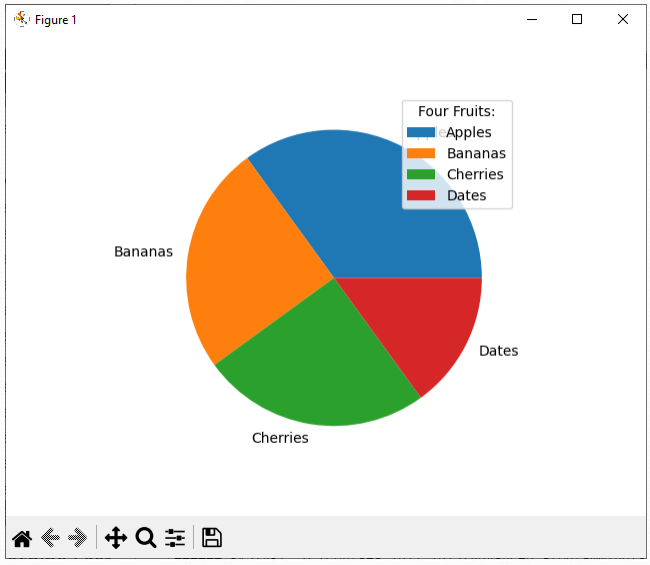
To add a header to the legend, add the title parameter to the legend function.

Example

Add a legend with a header:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels =
["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```


338



339

- https://www.w3schools.com/python/matplotlib_markers.asp
- رابط لمعالجة الصور
- https://akademik.adu.edu.tr/fakulte/muhendislik/personel/sbasarici/courses

340

Faculty of Computers and AI
Level -2 (first Semester)
Faculty of Science (computer science)
Course Name: Advanced programming (CS 241)

341

Prepared By
Prof. Dr. Gamal M. Behery

مكان المحاضرة مدرج (B105)

342

Matplotlib Image

343

Importing image data into Numpy arrays

- Matplotlib used to load image data.
- Here's the image we're going to play with:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('1.png')
print(img)
```


344

```
====== RESTART: D:/Courses_2022_2023/Advanced Programming/scatter/img1.py ======
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  ...
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  ...
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  ...
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 ...
```

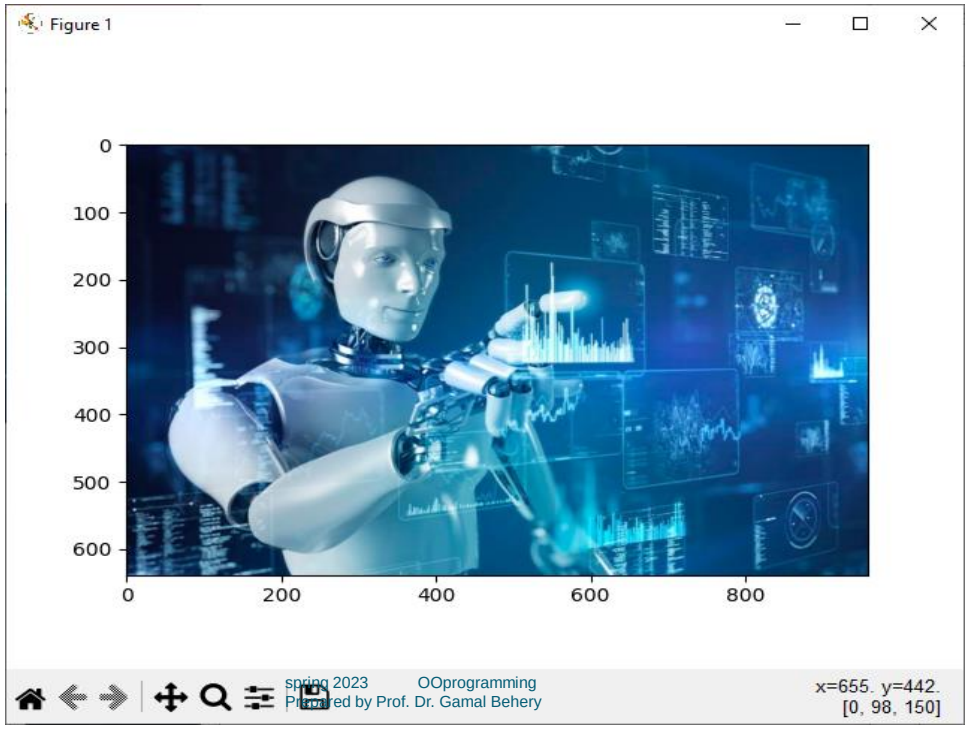

Plotting numpy arrays as images

- you have your data in a numpy array (either by importing it, or by generating it).
- Let's render it. In Matplotlib, this is performed using the ***imshow()*** function.
- Here we'll grab the plot object. This object gives you an easy way to manipulate the plot from the prompt.

imgplot = plt.imshow(img)

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('2.jpg')
imgplot = plt.imshow(img)
plt.show()
```


347



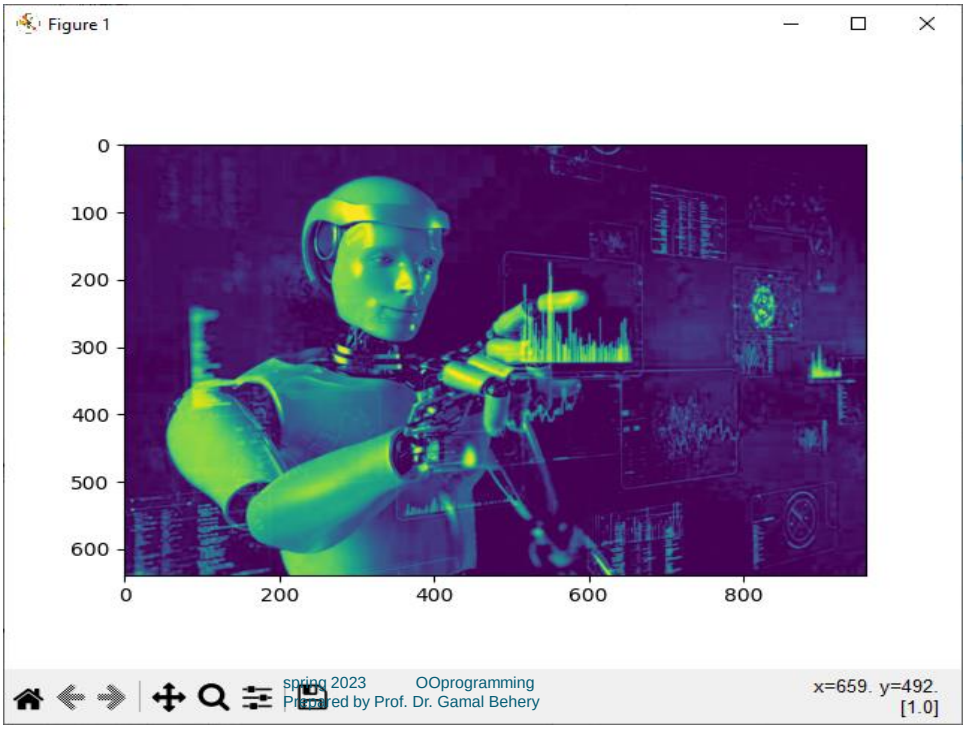
348

Applying pseudocolor schemes to image plots

- Pseudocolor can be a useful tool for enhancing contrast and visualizing your data more easily. This is especially useful when making presentations of your data using projectors - their contrast is typically quite poor.
- Pseudocolor is only relevant to single-channel, grayscale, luminosity images. We currently have an RGB image. Since R, G, and B are all similar (see for yourself above or in your data), we can just pick one channel of our data:

lum_img = img[:,:,0]

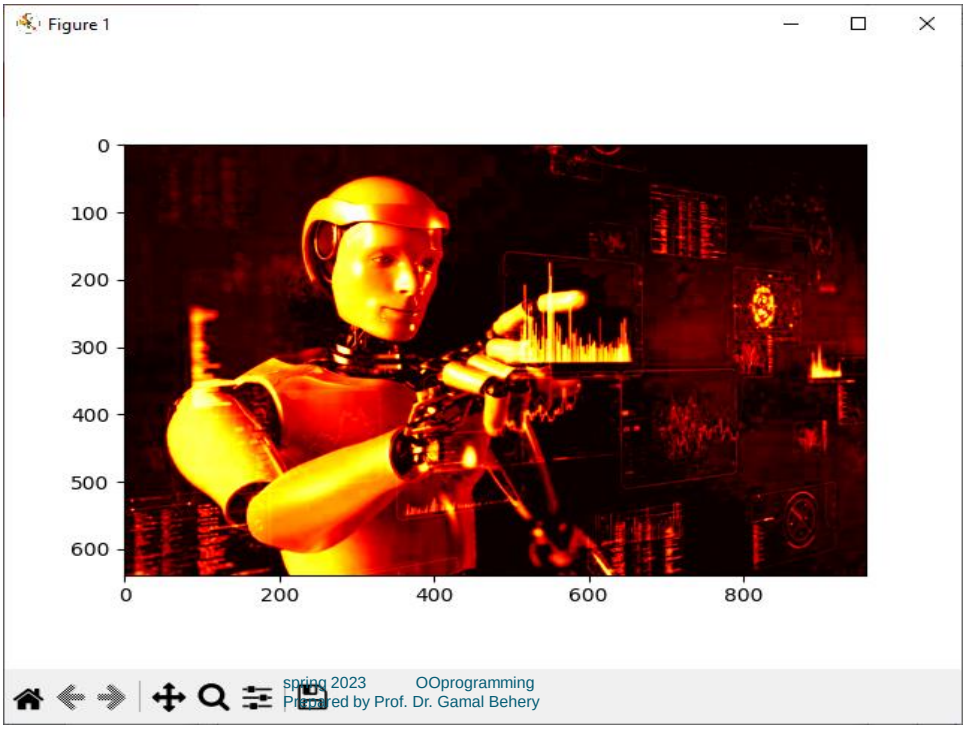

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('2.jpg')
lum_img = img[:,:,0]
imgplot = plt.imshow(lum_img)
plt.show()
```



Applying pseudocolor schemes to image plots

- Now, with a luminosity (2D, no color) image, the default ***colormap*** (aka lookup table, LUT), is applied. The default is called viridis. There are plenty of others to choose from.

plt.imshow(lum_img, cmap="hot")



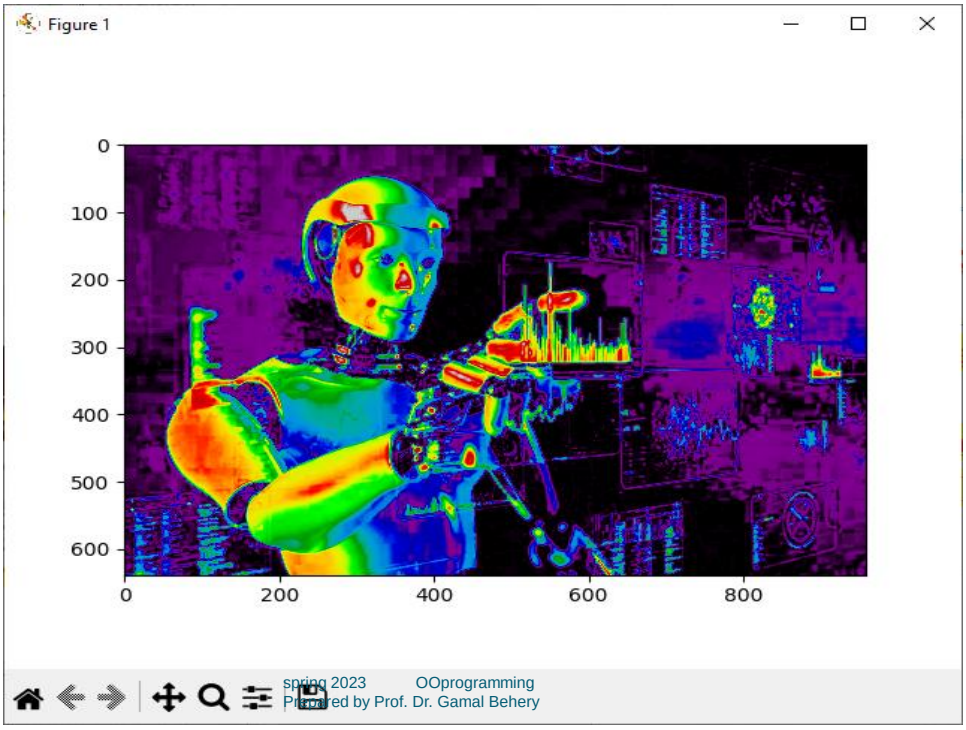
353

Applying pseudocolor schemes to image plots

- Note that you can also change colormaps on existing plot objects using the ***set_cmap()*** method:

```
imgplot = plt.imshow(lum_img)
imgplot.set_cmap('nipy_spectral')
```


354



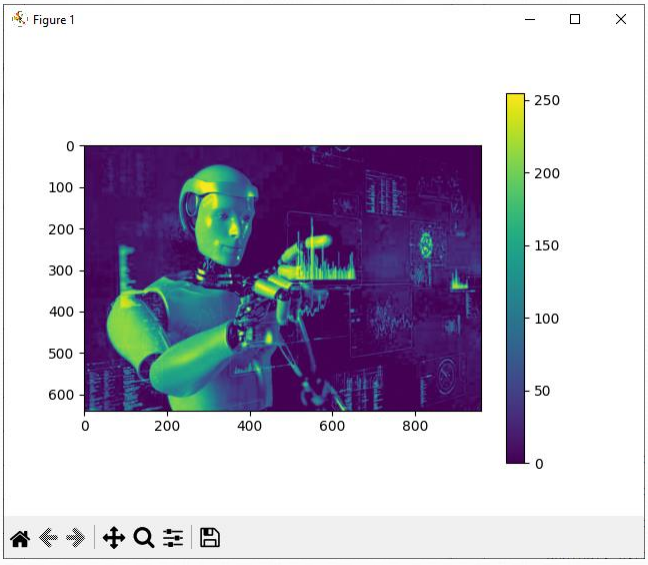
355

Color scale reference

- It's helpful to have an idea of what value a color represents. We can do that by adding color bars.

```
imgplot = plt.imshow(lum_img)
plt.colorbar()
```


356



357

pandas Library

Panel Data Analysis

358

What Is Pandas In Python?

- Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks.
- It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.
- As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution.

What Can You Do With DataFrames Using Pandas?

- pandas makes it simple to do many of the time consuming, repetitive tasks associated with working with data, including:
- Data cleansing
- Data fill
- Data normalization
- Merges and joins
- Data visualization
- Statistical analysis
- Data inspection
- Loading and saving data

Importing pandas lib

import pandas as pd

from pandas import *

from pandas import (lib1, lib2, …)

361

```
import pandas as pd
dat=pd.Series([3,6,9,8,5,4,2,6,3,5,8])
print(dat.describe())
pd.S
```

Period
PeriodDtype
PeriodIndex
RangeIndex
Series
SparseDtype
StringDtype
Timedelta
TimedeltaIndex
Timestamp

362

pandas

The basic operation

363

Create 1-D array (Series command)

- Put the vector [3,6,9,8,5,4,2,6,3,5,8] as a **table** by using pandas **Series command**:

```
import pandas as pd
dat=pd.Series([3,6,9,8,5,4,2,6,3,5,8])
print(dat)
```

Note: S is capital letter

```
0     3
1     6
2     9
3     8
4     5
5     4
6     2
7     6
8     3
9     5
10    8
dtype: int64
```


364

Create 1-D array (Series command)

- Put the vector [3,6,9,8,5,4,2,6,3,5,8] as a **table** by using pandas **Series command**:

```
import pandas as pd
y = [3,6,9,8,5,4,2,6,3,5,8]
dat=pd.Series(y)
print(dat)
```

Note: S is capital letter

```
0     3
1     6
2     9
3     8
4     5
5     4
6     2
7     6
8     3
9     5
10    8
dtype: int64
```


365

Create 1-D array (Series command)

- Put the vector [3,6,9,8,5,4,2,6,3,5,8] as a **table, values, index and keys** by using pandas **Series command**:

```
import pandas as pd
y = [3,6,9,8,5,4,2,6,3,5,8]
dat=pd.Series(y)
print(dat.values)
print(dat.index)
print(dat.keys)
```

Note: S is capital letter

```
[3 6 9 8 5 4 2 6 3 5 8]
RangeIndex(start=0, stop=11, step=1)
<bound method Series.keys of 0     3
1     6
2     9
3     8
4     5
5     4
6     2
7     6
8     3
9     5
10    8
dtype: int64>
```


366

Create 1-D array (Series command)

- Compute the statistics data for a list using pandas **describe() command**:

```
import pandas as pd
dat=pd.Series([3,6,9,8,5,4,2,6,3,5,8])
print(dat.describe())
```

Note: S is capital letter

```
dtype: int64
count    11.000000
mean      5.363636
std       2.292280
min       2.000000
25%       3.500000
50%       5.000000
75%       7.000000
max       9.000000
dtype: float64
```


Create 1-D array (Series command)

- You can Compute the previous statistics data for a list using pandas aggregate **agg() command**:

```
import pandas as pd
dat=pd.Series([3,6,9,8,5,4,2,6,3,5,8])
stat1 = dat.agg(['max','min','sum', 'mean', 'std'])
print(stat1)
```

```
max       9.000000
min       2.000000
sum      59.000000
mean      5.363636
std       2.292280
dtype: float64
```

Create 1-D array (Series command)

- You can Compute the ranges in array:

```
import pandas as pd
dat=pd.Series([3,6,9,8,5,4,2,6,3,5,8])
print(dat[1])
print(dat[1: 3])
print(dat[:3:2])
```

```
6
1    6
2    9
dtype: int64
0    3
2    9
dtype: int64
```


Create 1-D array (Series command)

- You can create an string index (keys) for list by using pandas **Series command**:

```
import pandas as pd
dat=pd.Series([3,6,9,8,5], index = ['a', 'b', 'c', 'd','e' ])
print(dat)
```

```
a    3
b    6
c    9
d    8
e    5
dtype: int64
```

Create 1-D array (Series command)

- You can create an string index (keys) for dictionary by using pandas **Series command**:

```
import pandas as pd
dat=pd.Series({'a':3,'b':6, 'c':9, 'd':8, 'e':5})
print(dat)
```

```
a    3
b    6
c    9
d    8
e    5
dtype: int64
```


Create 1-D array (Series command)

- You can print an element from dictionary by using pandas **Series command**:

```
import pandas as pd
dat=pd.Series({'a':3,'b':6, 'c':9, 'd':8, 'e':5})
print(dat['b'])
print(dat['d'])
```

```
6
8
```

Create 1-D array (Index command)

- Compute the data for a list using pandas **Index() command**:

```
import pandas as pd
dat=pd.Index([3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8])
print(dat)
```

Note: I is capital letter

```
Int64Index([3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8], dtype='int64')
```


Create 1-D array (Index command)

- Compute the logical operation using **Index() command**:

```
import pandas as pd
dat1=pd.Index([1, 3, 5, 7, 9])
dat2=pd.Index([2, 3, 5, 7, 11])
print(dat1)
print(dat2)
print(dat1 & dat2)
print(dat1 | dat2)
print(dat1 ^ dat2)
```

```
Int64Index([1, 3, 5, 7, 9], dtype='int64')
Int64Index([2, 3, 5, 7, 11], dtype='int64')
Int64Index([3, 5, 7], dtype='int64')
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
Int64Index([1, 2, 9, 11], dtype='int64')
```

Pandas

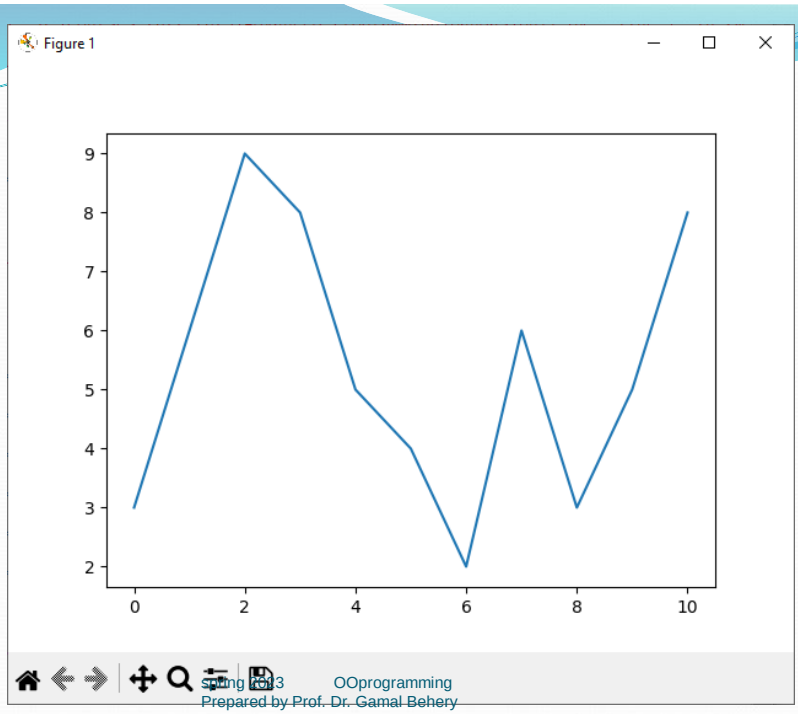
Graphics

375

Plot command

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot()
plt.show()
```


376

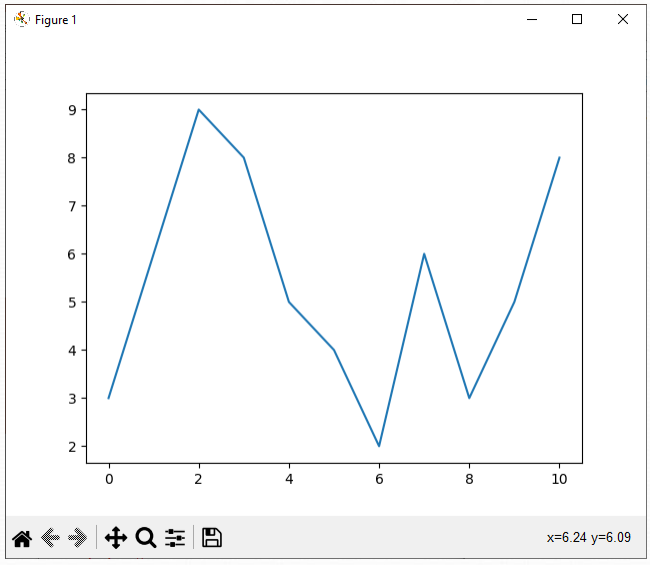


377

Plot command (line)

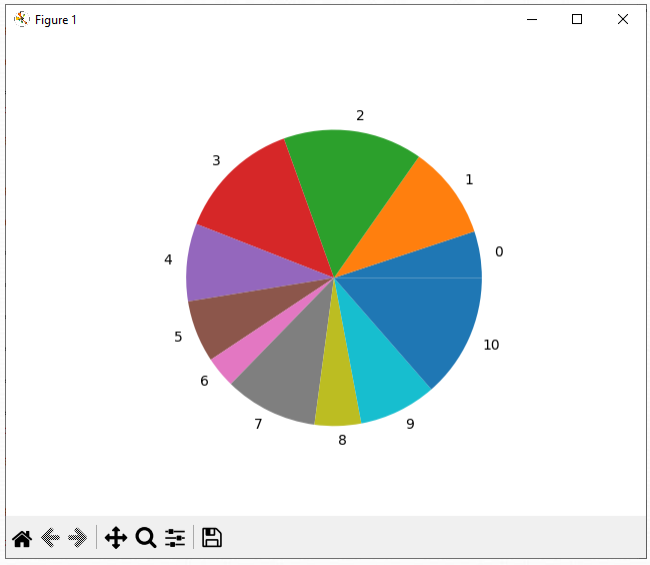
```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'line')
plt.show()
```


378



Plot command (pie)

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'pie')
plt.show()
```

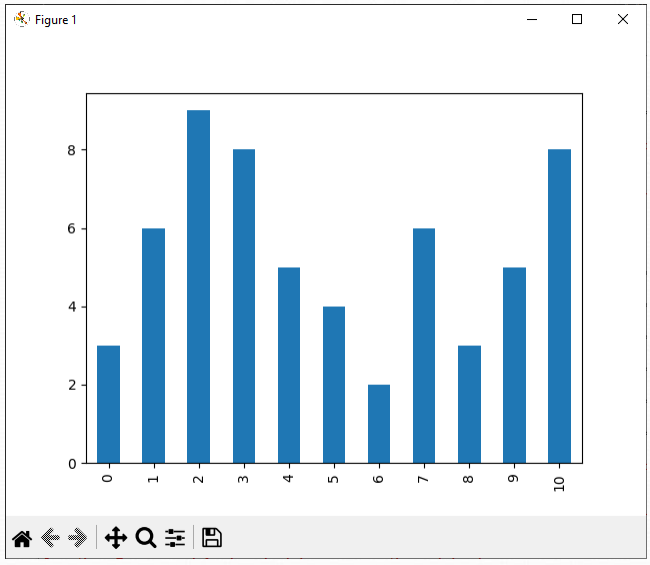


381

Plot command (bar)

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'bar')
plt.show()
```


382

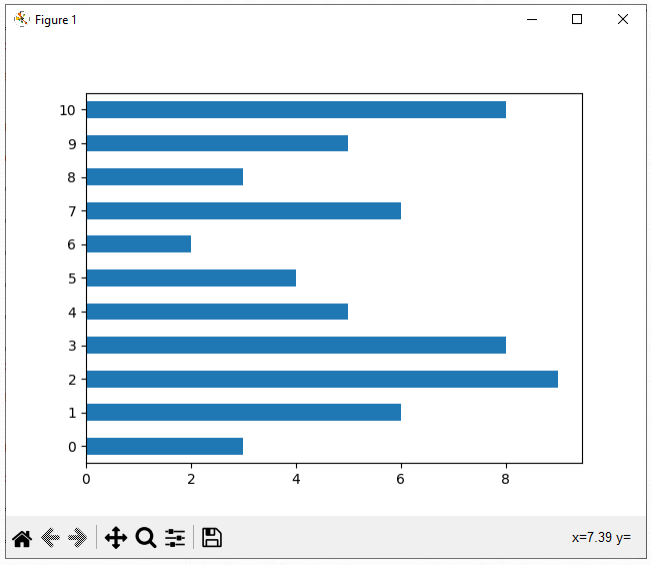


383

Plot command (bar horizontal)

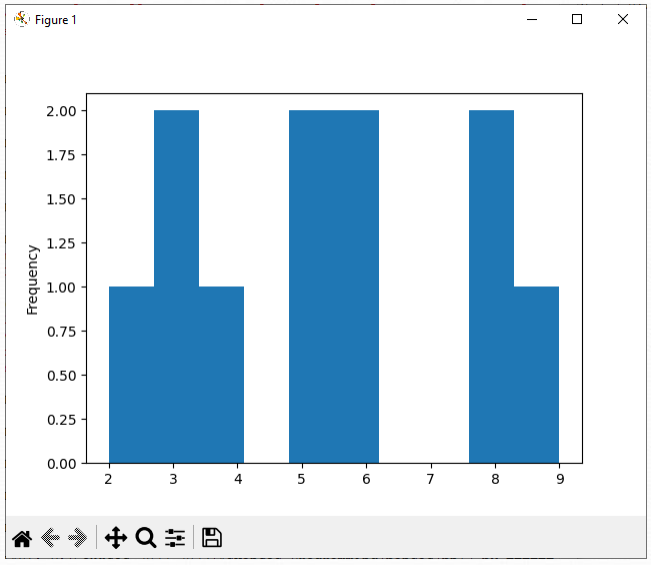
```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'barh')
plt.show()
```


384



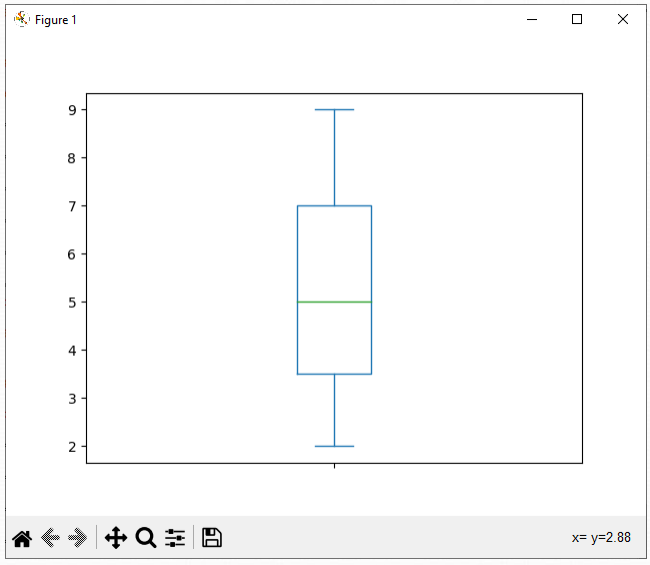
Plot command (histogram)

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'hist')
plt.show()
```



Plot command (box)

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
df.plot(kind = 'box')
plt.show()
```

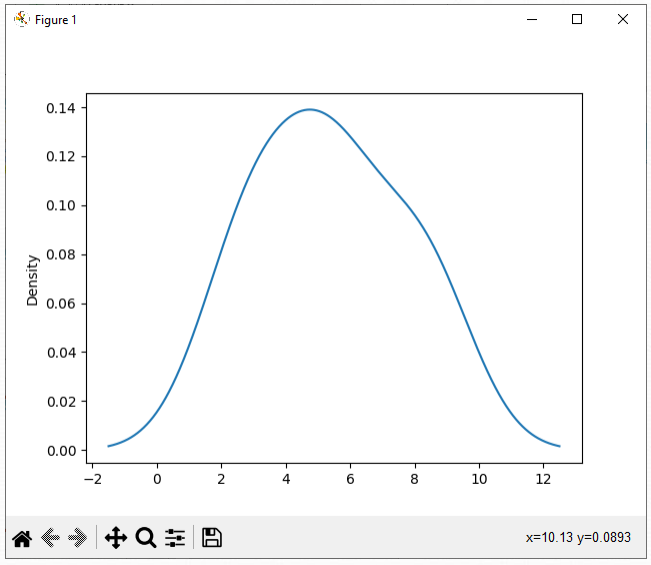


389

Plot command (kde)

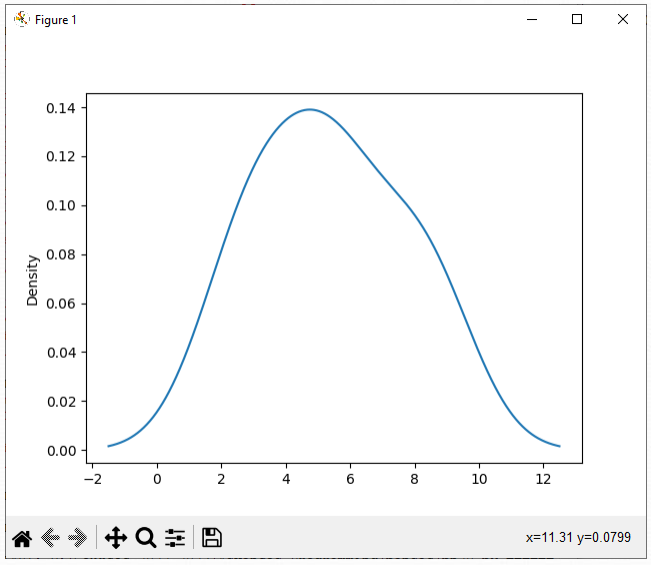
```
import pandas as pd
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
data=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
data.plot(kind = 'kde')
plt.show()
```


390



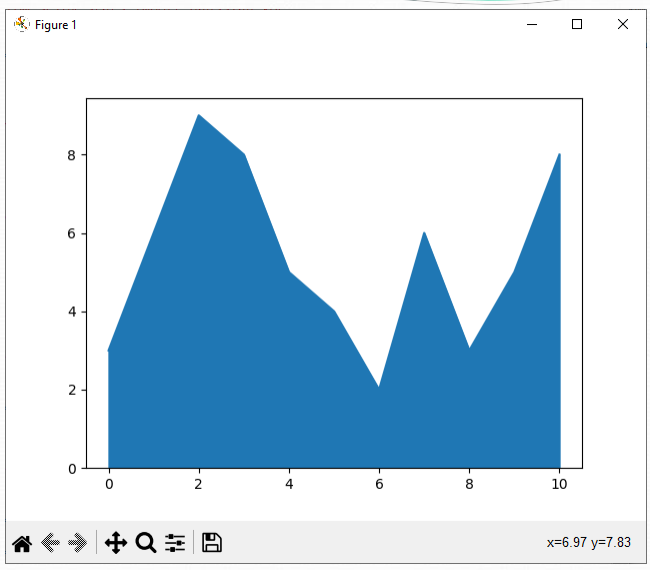
Plot command (density)

```
import pandas as pd
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
data=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
data.plot(kind = 'density')
plt.show()
```



Plot command (density)

```
import pandas as pd
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
data=pd.Series((3,6,9,8,5,4,2,6,3,5,8))
data.plot(kind = 'area')
plt.show()
```



395

pandas

Datat frame tools

396

DataFrame() command

- The only pandas library that can handle data frame tools.
- **Note: D & F is capital letters**

DataFrame() command

```
import pandas as pd
import numpy as np
myarray = np.array([[6,9,8,5,4,2],[0,2,5,6,3,9],
                    [8,5,4,1,2,3],[6,9,8,5,4,2],
                    [0,5,3,6,9,8],[8,7,4,5,2,3]])
rownames = ['a','b','c','d','e','f']
colnames = ['one', 'two', 'three','four','five', 'six']
grades = pd.DataFrame(myarray, index=rownames, columns=colnames)
print(myarray)
print(grades)
```

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

	one	two	three	four	five	six
a	6	9	8	5	4	2
b	0	2	5	6	3	9
c	8	5	4	1	2	3
d	6	9	8	5	4	2
e	0	5	3	6	9	8
f	8	7	4	5	2	3

DataFrame() command

```
import pandas as pd
w = pd.Series({'a':1,'b':2,'c':3,'d':4,'e':5})
x = pd.Series({'a':6,'b':7,'c':8,'d':9,'e':10})
y = pd.Series({'a':11,'b':12,'c':13,'d':14,'e':15})
z = pd.Series({'a':16,'b':17,'c':18,'d':19,'e':20})
colnames = ['one', 'two', 'three','four','five', 'six']
grades = pd.DataFrame({'math':w,'physics':x,
'french':y, 'chemstry':z})
print(grades)
```

	math	physics	french	chemstry
a	1	6	11	16
b	2	7	12	17
c	3	8	13	18
d	4	9	14	19
e	5	10	15	20

401

DataFrame() command

```
import pandas as pd
w = pd.Series({'a':1,'b':2,'c':3,'d':4,'e':5})
x = pd.Series({'a':6,'b':7,'c':8,'d':9,'e':10})
y = pd.Series({'a':11,'b':12,'c':13,'d':14,'e':15})
z = pd.Series({'a':16,'b':17,'c':18,'d':19,'e':20})
colnames = ['one', 'two', 'three','four','five', 'six']
grades = pd.DataFrame({'math':w,'physics':x,
'french':y, 'chemistry':z})
print(grades)
Print('--------------------------------------------------')
print (grades['chemistry'])
```


402

```
   math  physics  french  chemistry
a     1        6      11         16
b     2        7      12         17
c     3        8      13         18
d     4        9      14         19
e     5       10      15         20
-----------------------------------
a    16
b    17
c    18
d    19
e    20
Name: chemistry, dtype: int64
```


403

DataFrame() command

```
import pandas as pd
w = pd.Series({'a':1,'b':2,'c':3,'d':4,'e':5})
x = pd.Series({'a':6,'b':7,'c':8,'d':9,'e':10})
y = pd.Series({'a':11,'b':12,'c':13,'d':14,'e':15})
z = pd.Series({'a':16,'b':17,'c':18,'d':19,'e':20})
colnames = ['one', 'two', 'three','four','five', 'six']
grades = pd.DataFrame({'math':w,'physics':x,
'french':y, 'chemistry':z})
print(grades.T)   # Transpose
Print('-----------------------------------------------')
print (grades['chemistry'])
```


404

	a	b	c	d	e
math	1	2	3	4	5
physics	6	7	8	9	10
french	11	12	13	14	15
chemistry	16	17	18	19	20

Project:

- Write a project to compute the grade of 10 students in the courses: Math, Prog1, Stat, Intel_sys and cybersecurity. Put the general grade in a table using DataFrame() command.

St_name	Math	Prog1	Stat	Intel_sys	cybersecurity
Ahmed	A	A+	B	C+	D+
Sayed	B	C	D	A	D

DataFrame() command

```
import pandas as pd
w = pd.Series({'a':1,'b':2,'c':3,'d':4,'e':5})
x = pd.Series({'a':6,'b':7,'c':8,'d':9,'e':10})
y = pd.Series({'a':11,'b':12,'c':13,'d':14,'e':15})
z = pd.Series({'a':16,'b':17,'c':18,'d':19,'e':20})
colnames = ['one', 'two', 'three','four','five', 'six']
grades = pd.DataFrame({'math':w,'physics':x,
'french':y, 'chemistry':z})
print(grades.keys())
print('-----------------------------------------------')
print(grades.values)
```



```
Index(['math', 'physics', 'french', 'chemistry'], dtype='object')
-----------------------------------------------
[[ 1  6 11 16]
 [ 2  7 12 17]
 [ 3  8 13 18]
 [ 4  9 14 19]
 [ 5 10 15 20]]
```

DataFrame() command

```
import pandas as pd
w = pd.Series({'a':1,'b':2,'c':3,'d':4,'e':5})
x = pd.Series({'a':6,'b':7,'c':8,'d':9,'e':10})
y = pd.Series({'a':11,'b':12,'c':13,'d':14,'e':15})
z = pd.Series({'a':16,'b':17,'c':18,'d':19,'e':20})
grades = pd.DataFrame({'math':w,'physics':x, 'french':y,
'chemistry':z})
print('math' in grades.keys())
print('Math' in grades.keys())
print('----------------------------------------')
print(12 in grades.values)
print(55 in grades.values)
```


409

```
True
False
----------
True
False
```


410

OpenCV In Python

411

OpenCV in Python

- OpenCV is a Python library that allows you to perform image processing and computer vision tasks.
- It provides a wide range of features, including:
  - **Object detection,**
  - **Face recognition, and**
  - **Tracking**.

412

What is OpenCV?

- OpenCV is an open-source software library for computer vision and machine learning.

What is Computer Vision?

- The term Computer Vision (CV) is used and heard very often in artificial intelligence (AI) and deep learning (DL) applications.
- The term essentially means giving a computer the ability to see the world as we humans do.

How does a computer read an image?

415

OpenCV installation

- **Install using Anaconda:**

```
conda install -c conda-forge opencv
```

- **For Windows:**

```
pip install opencv-python
```


416

Read & Save Images

- **Imread function in OpenCV**

```
cv2.imread(path, flag)
```

- Usually the method ***imread()*** returns an image that is loaded from the specified file.

Example:

```
#importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color image
img = cv2.imread('dog.jpg',1)
#This is using for display the image
cv2.imshow('image', img)
cv2.waitKey() # This is necessary to be required so that
the image doesn't close immediately.
#It will run continuously until the key press.
cv2.destroyAllWindows()
```

Imwrite function in OpenCV

- We can use OpenCV's **imwrite()** function to save an image in a storage device and the file extension defines the image format as shown in the example below.
- The syntax is the following:

  cv2.imwrite(filename, image)

Example:

```
import cv2

# read image

img = cv2.imread(r'C:\Users\Mirza\dog.jpeg', 1)

# save image

status = cv2.imwrite(r'C:\Users\Mirza\dog.jpeg',img)

print("Image written sucess? : ", status)
```

Basic Operation On images

- In this section, we are going to discuss some of the basic operations that we can do on the images once we have successfully read them.
- The operations we are going to do here ae:
  - **Access pixel values and modify them**
  - **Access image properties**
  - **Set a Region of Interest (ROI)**
  - **Split and merge image channels**

421

Access pixel values and modify them

- how we can access a particular pixel value of an image.

```
import numpy as np
import cv2 as cv
img = cv.imread(r'C:\Users\Mirza\dog.jpeg')
px = img[100,100]
print( px )
```

Output:

[157 166 200]

422

Access pixel values and modify them

- We can also access only one of the channels as shown below:

```
# accessing only blue pixel
blue = img[100,100,0]
print( blue )
```

Output:

157

Access pixel values and modify them

- We can also access only one of the channels as shown below:

```
# accessing only blue pixel
blue = img[100,100,0]
print( blue )
```

Output:

157

spring 2023 OOprogramming
Prepared by Prof. Dr. Gamal Behery

Access pixel values and modify them

- To modify the values, we just need to access the pixel and then overwrite it with a value as shown below:

```
img[100,100] = [255,255,255]
print( img[100,100] )
```

Output:

```
[255 255 255]
```


Access Image properties

- number of rows, columns, and channels.
- We can access the later three by using the ***shape()*** method as shown below:

```
print( img.shape )
(342, 548, 3)
print( img.size )
562248
```

Image ROI(Region of interest)

- Often you may come across some images where you are only interested in a specific region.
- Say you want to detect eyes in an image, will you search the entire image, possibly not as that may not fetch accurate results.
- But we know that eyes are a part of face, so it is better to detect a face first ,thus here the face is our **ROI**.

Splitting and Merging Image Channels

- We can split the channels from an image and then work on each channel separately.
- sometimes you may need to merge them back together, here is how we do it:

```
b,g,r = cv.split(img)
img = cv.merge((b,g,r))
b = img[:,:,0]
g = img[:,:,1]
r = img[:,:,2]
```

OpenCV Resize Image

Mostly you will do such operation in Machine learning and deep learning as it reduces the time of training of a neural network.

As the number of pixels in an image increases, the more is the number of input nodes that in turn increases the complexity of the model.

We use an inbuilt ***resize()*** method to resize an image.

OpenCV Resize Image

Syntax:

cv2.resize(s, size, fx, fy, interpolation)

Parameters:

s – input image (required).

size – desired size for the output image after resizing (required)

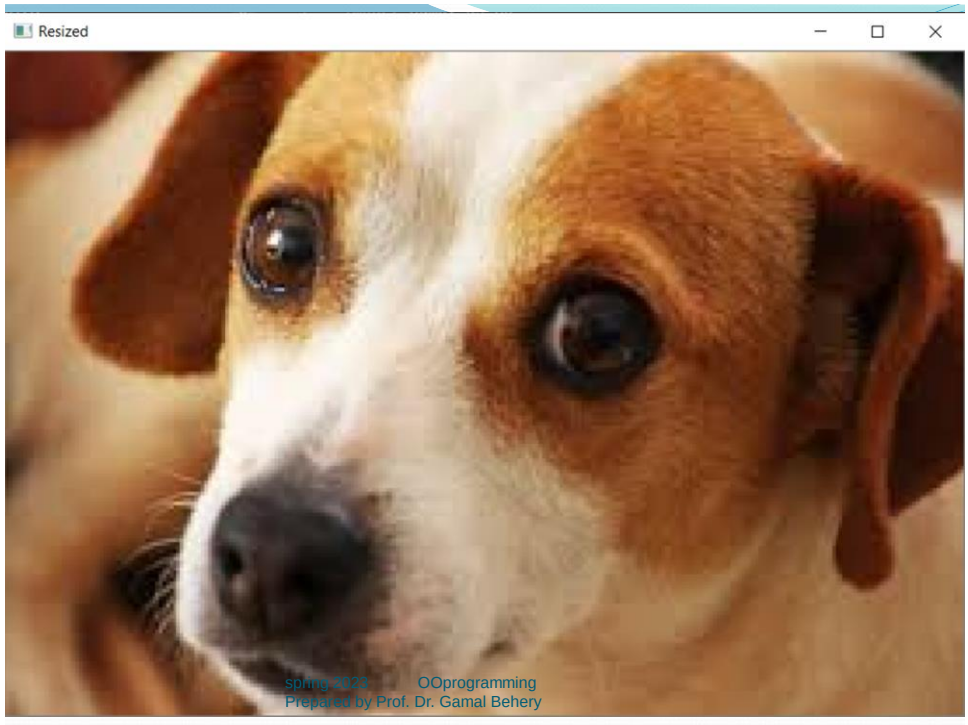
fx – Scale factor along the horizontal axis.(optional)

fy – Scale factor along the vertical axis.

Here is an example of how we can use this method:

```
import cv2
import numpy as np
#importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color image
img = cv2.imread('dog.jpg',1)
print(img.shape)
img_resized=cv2.resize(img, (780, 540),
            interpolation = cv2.INTER_NEAREST)
cv2.imshow("Resized",img_resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



433

OpenCV Image Rotation

- We may need to rotate an image in some of the cases and we can do it easily by using OpenCV .We use cv2.rotate() method to rotate a 2D array in multiples of 90 degrees. Here is the syntax:

cv2.rotate(src, rotateCode[, dst])

Parameters:

src: It is the image to be rotated.

rotateCode: It is an enum to specify how to rotate the array. Here are some of the possible values :

cv2.cv2.ROTATE_90_CLOCKWISE

cv2.ROTATE_180

cv2.ROTATE_90_COUNTERCLOCKWISE

434

Here is an example using this function.

```
import cv2
import numpy as np
#importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color
image
img = cv2.imread('dog.jpg',1)
print(img.shape)
image = cv2.rotate(img,
cv2.ROTATE_90_COUNTERCLOCKWISE)
cv2.imshow("Rotated",image)
cv2.waitKey()
cv2.destroyAllWindows()
```

Output:

Rotated image

- Now what if we want to rotate the image by a certain angle.
- We can use another method for that.
- First calculate the affine matrix that does the affine transformation (linear mapping of pixels) by using the ***getRotationMatrix2D*** method,
- next we warp the input image with the affine matrix using ***warpAffine*** method.

437

Rotated image

- Here is the syntax of these functions:

```
cv2.getRotationMatrix2D(center, angle, scale)
cv2.warpAffine(Img, M, (W, H))
```

center: center of the image (the point about which rotation has to happen)
angle: angle by which image has to be rotated in the anti-clockwise direction.
scale: scales the image by the value provided,1.0 means the shape is preserved.
H: height of image **W**: width of the image.
M: affine matrix returned by cv2.getRotationMatrix2D
Img: image to be rotated.

438

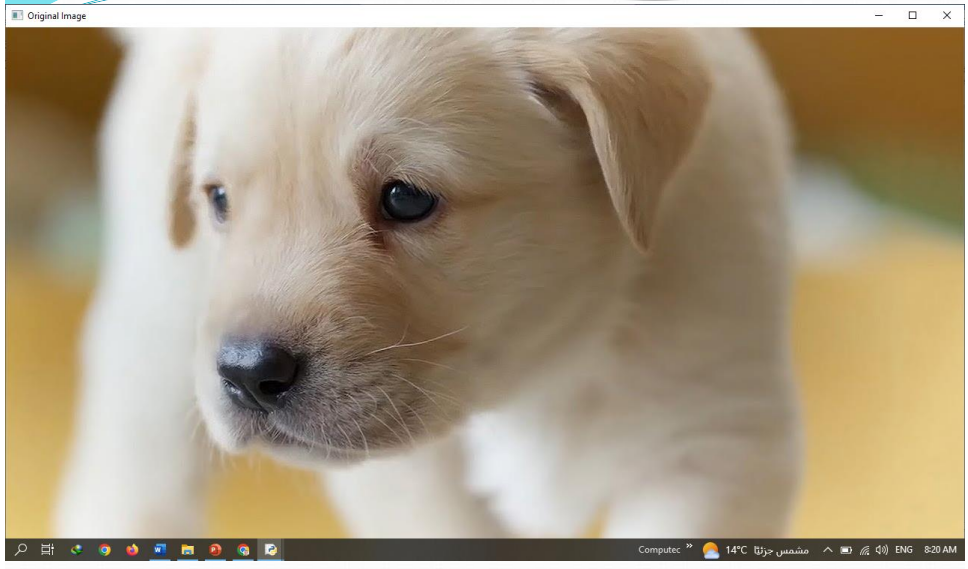
Here is an example in which we rotate an image by various angles.

```
import cv2
import numpy as np
#importing the opencv module
import cv2
# using imread('path') and 1 denotes read as  color image
img = cv2.imread('dog.jpg',1)
# get image height, width
(h, w) = img.shape[:2]
# calculate the center of the image
center = (w / 2, h / 2)
scale = 1.0
```


Perform the counter clockwise rotation holding at the center

```
# 45 degrees
M = cv2.getRotationMatrix2D(center, 45, scale)
print(M)
rotated45 = cv2.warpAffine(img, M, (h, w))
# 110 degrees
M = cv2.getRotationMatrix2D(center,110, scale)
rotated110 = cv2.warpAffine(img, M, (w, h))
# 150 degrees
M = cv2.getRotationMatrix2D(center, 150, scale)
rotated150 = cv2.warpAffine(img, M, (h, w))
cv2.imshow('Original Image',img)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
```

```
cv2.imshow('Image rotated by 45 degrees',rotated45)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
cv2.imshow('Image rotated by 110 degrees',rotated110)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
cv2.imshow('Image rotated by 150 degrees',rotated150)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
```

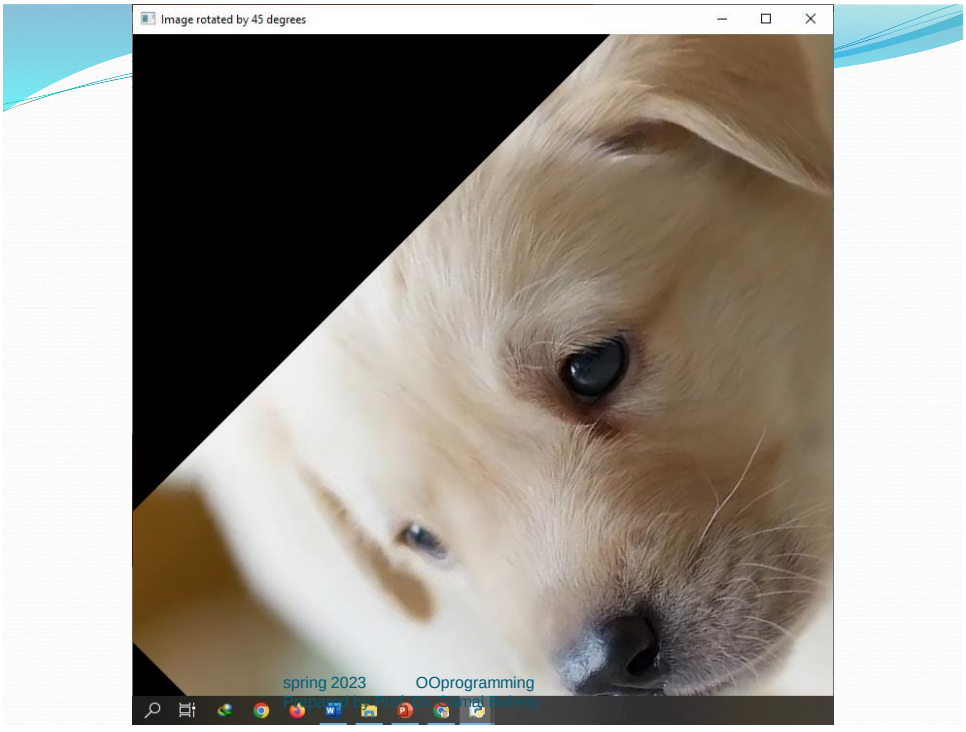


Image rotated by 45 degrees

443

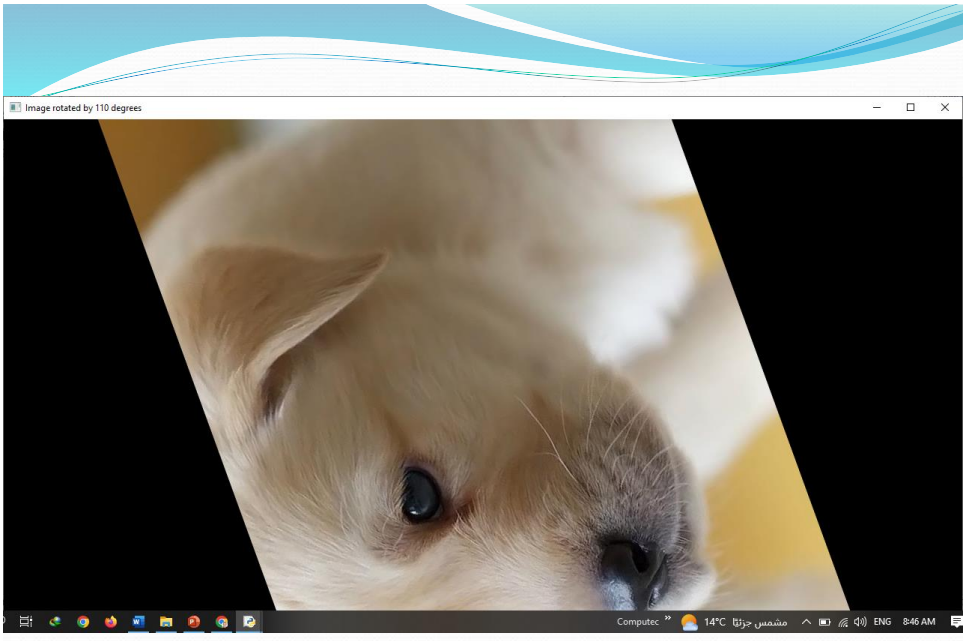
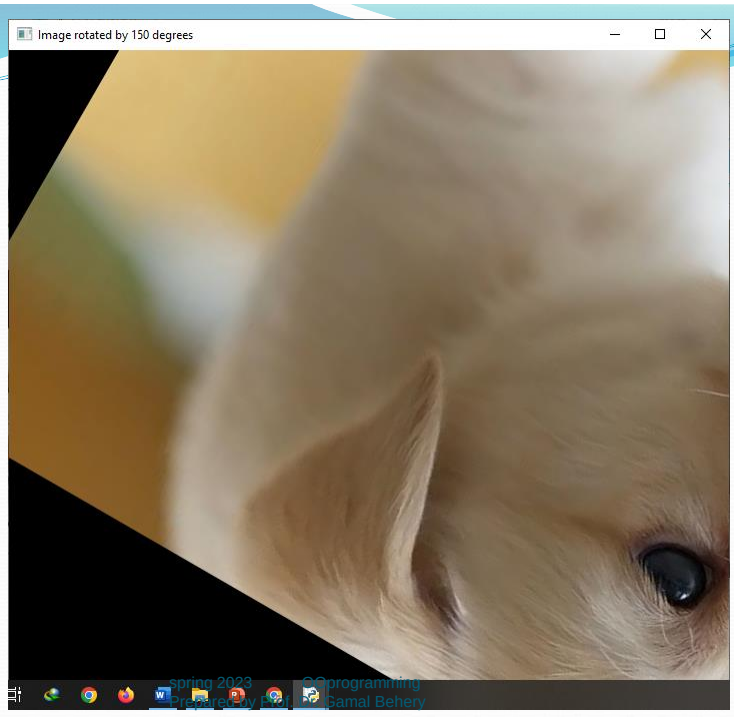


Image rotated by 110 degrees
Computec
14°C
ENG
8:46 AM

444



445

OpenCV Drawing Functions

- We may require to draw certain shapes on an image such as **circle, rectangle, ellipse, polylines, convex**, etc. and we can easily do this using OpenCV.
- It is often used when we want to highlight any object in the input image for example in case of face detection, we might want to highlight the face with a rectangle.
- Here we will learn about the drawing functions such as circle, rectangle, lines, polylines and also see how to write text on an image.

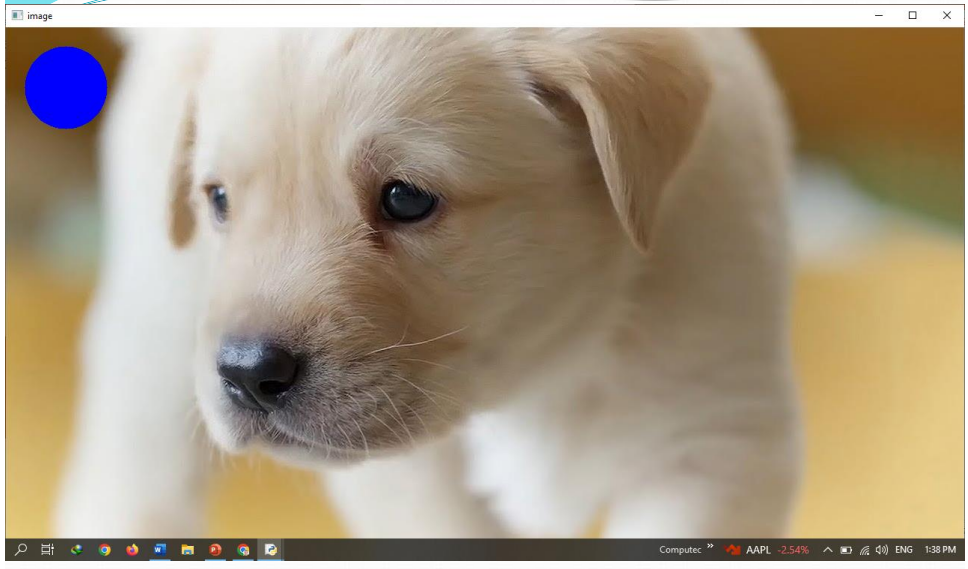
446

Drawing circle:

- The syntax and parameters:

```
cv2.circle(image, center_coordinates, radius, color, thickness)
```

```
import numpy as np
import cv2
img = cv2.imread(r'dog.jpg', 1)
cv2.circle(img,(80,80), 55, (255,0,0), -1)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

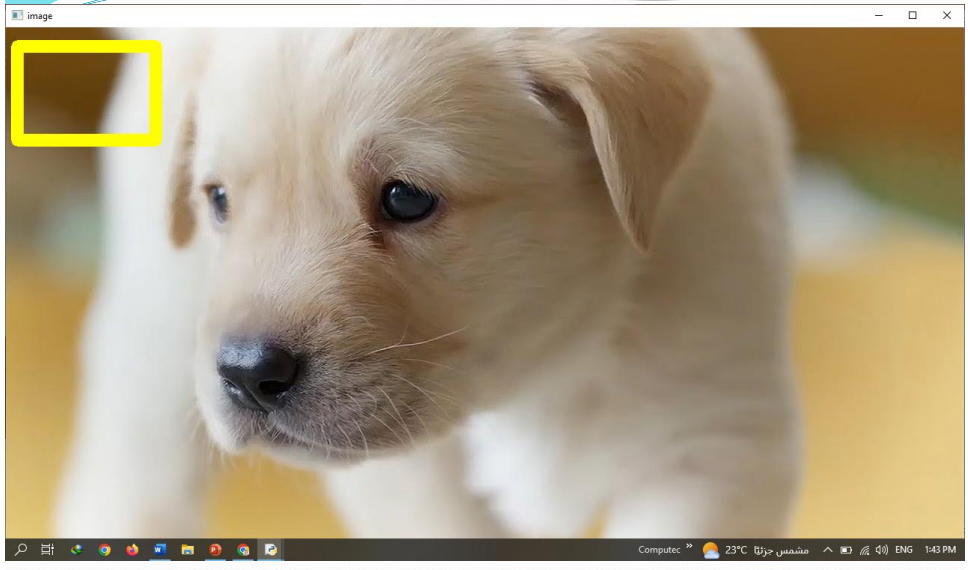



Drawing Rectangle

- The syntax for this function:

```
cv2.rectangle(image, start_point, end_point, color, thickness)
```

```
import numpy as np
import cv2
img = cv2.imread(r'C:\Users\Mirza\dog.jpeg', 1)
cv2.rectangle(img,(15,25),(200,150),(0,255,255),15)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

451

452