Artificial Intelligence Lab 01

Objectives

The objective of this session is to get exposure to Python programming , and write some simple code to access data and plot it using some key Python libraries.

Learning Outcomes

1. Write simple Python code to create random numbers and frequency histogram using the NumPy library.
2. Create graphs in Python using the matplotlib library.
3. Understand and work with numeric data in the library pandas.

In [1]:

```
pip install numpy
```

```
Requirement already satisfied: numpy in /home/linux/anaconda3/lib/python3.9/site-packages (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /home/linux/anaconda3/lib/python3.9/site-packages (3.5.2)
Requirement already satisfied: fonttools>=4.22.0 in /home/linux/anaconda3/lib/python3.9/site-packages (from matp
lotlib) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in /home/linux/anaconda3/lib/python3.9/site-packages (from m
atplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /home/linux/anaconda3/lib/python3.9/site-packages (from matplotli
b) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /home/linux/anaconda3/lib/python3.9/site-packages (from matplo
tlib) (21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/linux/anaconda3/lib/python3.9/site-packages (from matp
lotlib) (1.4.2)
Requirement already satisfied: numpy>=1.17 in /home/linux/anaconda3/lib/python3.9/site-packages (from matplotli
b) (1.21.5)
Requirement already satisfied: pillow>=6.2.0 in /home/linux/anaconda3/lib/python3.9/site-packages (from matplotl
ib) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in /home/linux/anaconda3/lib/python3.9/site-packages (from matpl
otlib) (3.0.9)
Requirement already satisfied: six>=1.5 in /home/linux/anaconda3/lib/python3.9/site-packages (from python-dateut
il>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
pip install NetworkX
```

```
Requirement already satisfied: NetworkX in /home/linux/anaconda3/lib/python3.9/site-packages (2.8.4)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```
pip install pandas
```

```
Requirement already satisfied: pandas in /home/linux/anaconda3/lib/python3.9/site-packages (1.4.4)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/linux/anaconda3/lib/python3.9/site-packages (from
pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/linux/anaconda3/lib/python3.9/site-packages (from pandas)
(2022.1)
Requirement already satisfied: numpy>=1.18.5 in /home/linux/anaconda3/lib/python3.9/site-packages (from pandas)
(1.21.5)
Requirement already satisfied: six>=1.5 in /home/linux/anaconda3/lib/python3.9/site-packages (from python-dateut
il>=2.8.1->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Numpy

In [5]:

```
import numpy as np
```

Numpy Arrays

- How to declare an array
- How to access array elements using indexing

In [6]:

```
a = np.array([1,2,3,4,5])
print(a[0])
print(a[4])
type(a)
```

```
1
5
```

Out[6]:

```
numpy.ndarray
```

Properties of Array

- **Size** Returns the size (number of elements) of an array
- **ndim** Returns the dimensions (1D, 2D etc) of an array
- **shape** Returns the shape (rows, cols) of an array

In [7]:

```
print(a)
print(a.size)
print(a.ndim)
print(a.shape)
```

```
[1 2 3 4 5]
5
1
(5,)
```

Two Dim Arrays and it properties

In [8]:

```
b = np.array([ [1,2],
               [3,4],
               [5,6] ])

print(b)
print(b.size)
print(b.ndim)
print(b.shape)
```

```
[[1 2]
 [3 4]
 [5 6]]
6
2
(3, 2)
```

Reshaping the array

- **reshape()** function can be used to reshape/redfine the number of rows and cols

In [9]:

```
c = np.array([ 1,12,3,44,5,6,67,8])

d = c.reshape(4,2)

print("Array c : ", c)
print("Array d : ", d)
print("c.shape : ", c.shape)
print("d.shape : ", d.shape)
```

```
Array c :  [ 1 12  3 44  5  6 67  8]
Array d :  [[ 1 12]
 [ 3 44]
 [ 5  6]
 [67  8]]
c.shape :  (8,)
d.shape :  (4, 2)
```

Array elements can be modified

In [10]:

```
c = np.array([ 1,2,3,4,5,6,7,8])

print("Before change in Array c : ", c)
c[0] = 100
print("After change in Array c : ", c)
```

```
Before change in Array c :  [1 2 3 4 5 6 7 8]
After change in Array c :  [100   2   3   4   5   6   7   8]
```

Slicing the array

In [11]:

```
c = np.array([1,2,3,4,5,6,7,8])

d =  c[1:4]

print(d)
```

```
[2 3 4]
```

In [12]:

```
c[3:5] = 40,50

print(c)

print(d)
```

```
[ 1  2  3 40 50  6  7  8]
[ 2  3 40]
```

Performing operations on arrays

- Addition using Numpy Arrays

In [13]:

```
A = np.array([1,0])
B = np.array([0,1])
C = A + B
C
```

Out[13]:

```
array([1, 1])
```

Subtraction & Scaler Multiplication using Numpy

In [14]:

```
A = np.array([1,0])
B = np.array([0,1])
D = A - B
D
```

Out[14]:

```
array([ 1, -1])
```

In [15]:

```
A = np.array([2,3])
E = 2*A
E
```

Out[15]:

```
array([4, 6])
```

Hadamard Product (Element-wise Multiplication)

In [16]:

```
A = np.array([2,3])
B = np.array([4,5])
F = A* B
F
```

Out[16]:

```
array([ 8, 15])
```

Matrix Multiplication

In [17]:

```
A = np.array([[2,3]])
B = np.array([[4],[5]])
G = np.dot(A,B)
G
```

Out[17]:

```
array([[23]])
```

Broadcasting

In [18]:

```
A = np.array([2,6,8,7,3])
H = A + 4
H
```

Out[18]:

```
array([ 6, 10, 12, 11,  7])
```

Universal Functions

- Various functions can be applied on numpy arrays

In [19]:

```
A = np.array([2,6,8,7,3])
meanA = A.mean()
maxA = A.max()

print(meanA)
print(maxA)
print(np.pi)
```

```
5.2
8
3.141592653589793
```

In [20]:

```
J = np.array([0, np.pi/2, np.pi])
J
```

Out[20]:

```
array([0.        , 1.57079633, 3.14159265])
```

In [21]:

```
K = np.sin(J)
K
```

Out[21]:

```
array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])
```

linspace Function

- **linspace()** Returns number spaces evenly w.r.t interval.
- The following code returns a 5 element numpy array in the range (-2, 2)

In [22]:

```
np.linspace(-2,2,num=5)
```

Out[22]:

```
array([-2., -1.,  0.,  1.,  2.])
```

- The following code returns a 9 element numpy array in the range (-2, 2)

In [23]:

```
np.linspace(-2,2,num=9)
```

Out[23]:

```
array([-2. , -1.5, -1. , -0.5,  0. ,  0.5,  1. ,  1.5,  2. ])
```

- Generate 100 random numbers from a normal distribution with mean = 100 and standard deviation = 15

In [24]:

```
mu, sigma = 100, 15
```

In [25]:

```
x = mu + sigma*np.random.randn(100)
```

In [26]:

```
x
```

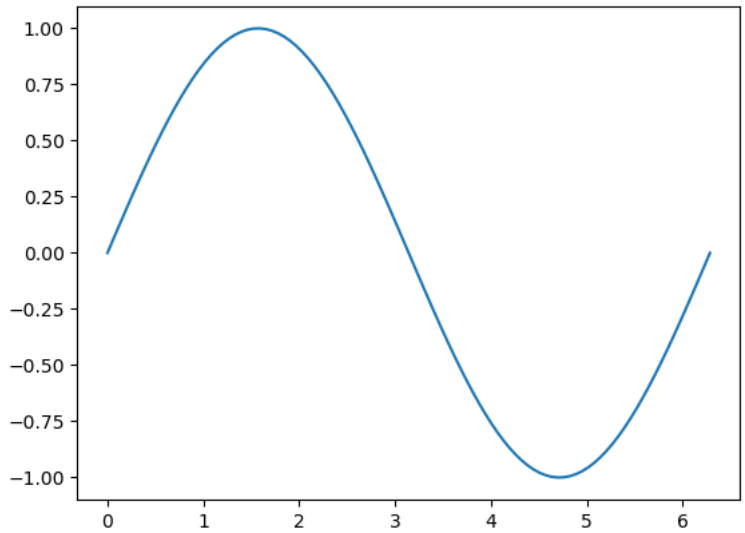
Out[26]:

```
array([ 88.63862109,  84.87048199,  82.09002617, 116.12529148,
        95.98919985, 107.04081412,  95.79851412, 111.23854784,
       109.02412601,  82.97883011, 111.69459097, 119.52456832,
        79.88683152,  58.30246647,  91.22524304, 130.07489411,
       119.45633874,  98.45528031,  95.97864848, 111.14069377,
        82.42433569, 102.82983012,  94.7009925 , 104.40014782,
       108.05082211, 133.75694423,  94.09050677,  85.25479825,
        86.66421967,  89.97058881,  89.27200249, 106.07642502,
        95.91630448,  85.28646623,  83.87922748, 104.18378136,
       114.45559355, 112.82503041,  87.53236168,  91.00852799,
        99.22638869, 115.18065957,  90.35945893, 116.93089828,
        79.56254388, 110.0475761 ,  84.69384714,  81.98502346,
       100.25308385, 108.80663817,  83.90023864,  86.70667968,
       111.37393094,  84.58019201, 106.88267196,  93.35776172,
       116.46456907, 108.60687941,  85.01345673,  65.17455614,
       107.10170424,  76.25187058, 100.79285692, 109.71033613,
        71.14145862, 106.86779172, 105.17263119, 109.42952942,
       118.36017091,  99.97039059, 116.80077926, 124.2390526 ,
        85.98552865,  93.19290909, 115.80999319, 108.95049526,
       115.33728823,  97.9216721 ,  82.79797637,  67.58279167,
        77.66662019,  80.00632717, 113.256103  ,  98.073067  ,
       111.18537093, 100.23779574, 111.10663697,  99.1826228 ,
       122.77024263,  88.82673543, 104.02976414, 100.60416366,
        96.29372574,  79.31351533, 114.12593353, 113.28806489,
        79.98487869,  94.99730929, 111.9345441 ,  94.27041406])
```

Ploting Charts in Python

In [27]:

```
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(0,2*np.pi,100)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```

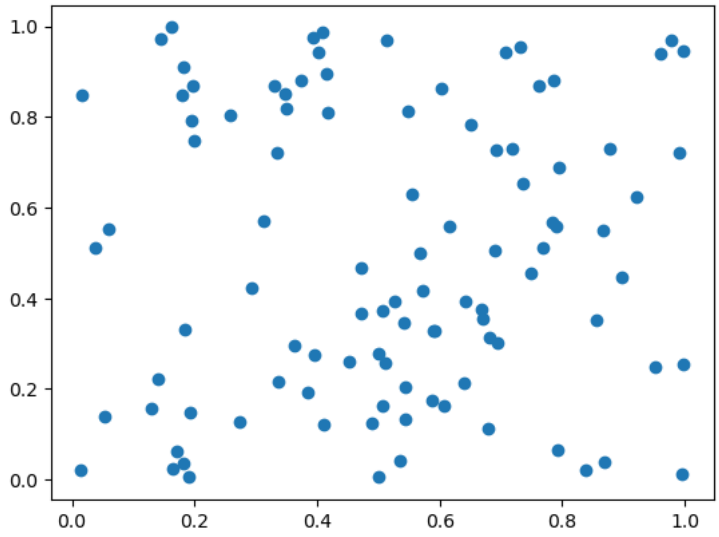


In [28]:

```
a = np.random.random(100)
b = np.random.random(100)
```

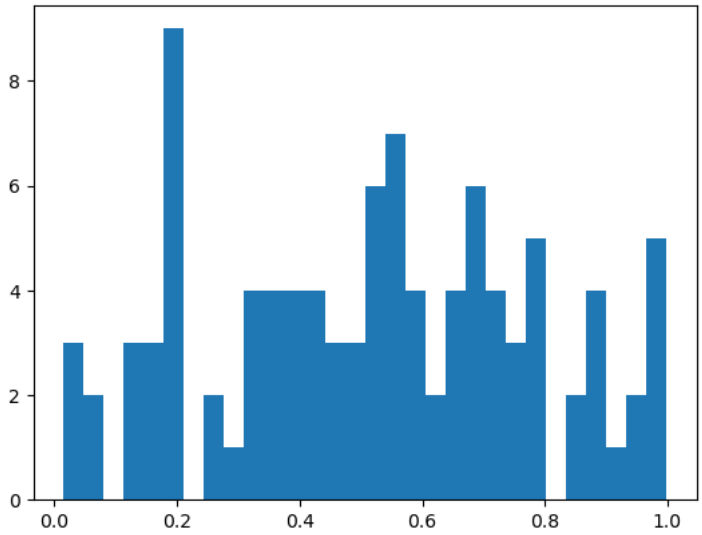
In [29]:

```
plt.scatter(a,b)
plt.show()
```



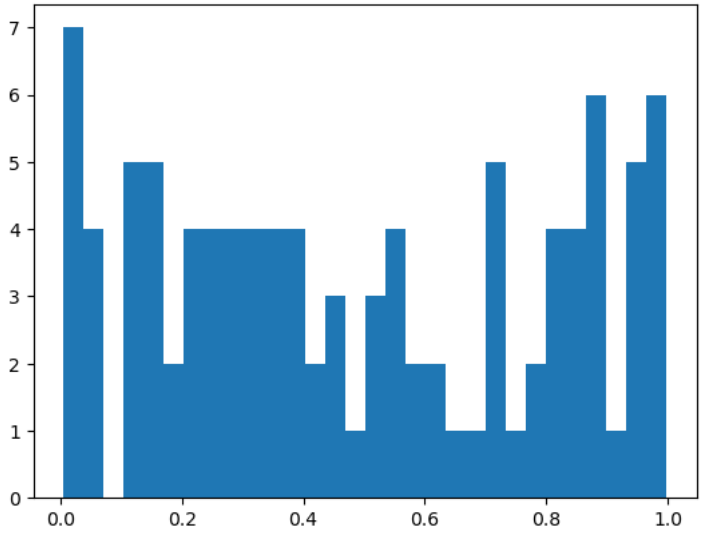
In [30]:

```
plt.hist(a,bins=30)
plt.show()
```



In [31]:

```
plt.hist(b,bins=30)
plt.show()
```

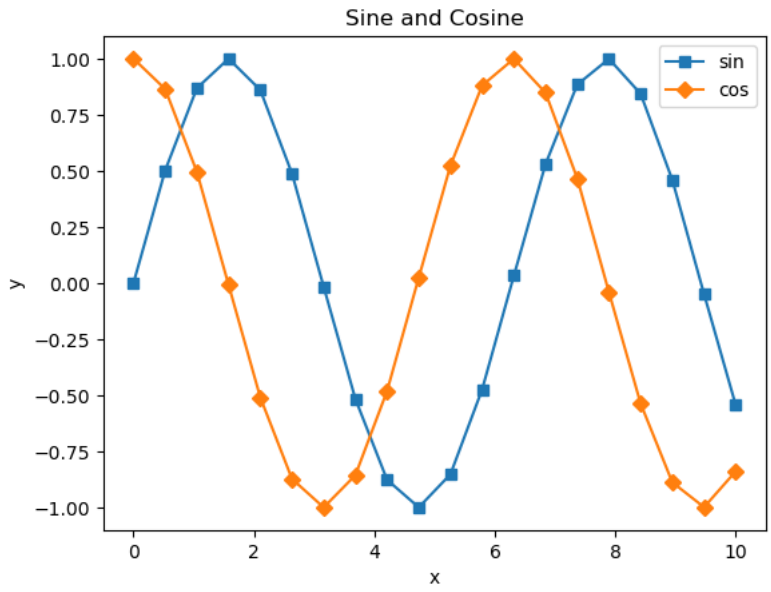


In [ ]:

```
```

In [32]:

```
# Generate data for the first line
x1 = np.linspace(0, 10, 20)
y1 = np.sin(x1)

# Generate data for the second line
x2 = np.linspace(0, 10, 20)
y2 = np.cos(x2)

# Create a figure and axes
fig, ax = plt.subplots()

# Plot the first line with a square marker
ax.plot(x1, y1, '-s', label='sin')

# Plot the second line with a diamond marker
ax.plot(x2, y2, '-D', label='cos')



# Add a legend to the plot
ax.legend()

# Set the title and labels
ax.set_title("Sine and Cosine")
ax.set_xlabel("x")
ax.set_ylabel("y")

# Show the plot
plt.show()
```



Reading & Writing Data in Python

In [33]:

```
import pandas as pd
```

In [34]:

```
path = "Auto85.csv"
data = pd.read_csv(path, header = None) #assumes data has header

data.head(5)
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

5 rows × 26 columns

In [35]:

```
headers = ["symboling","normalized-losses","make","fuel-type",
           "aspiration", "num-of-doors", "body-style", "drive-wheels",
           "engine-location", "wheel-base", "length", "width", "height",
           "curb-weight", "engine-type", "num-of-cylinders", "engine-size",
           "fuel-system", "bore", "stroke", "compression-ratio", "horsepower",
           "peak-rpm", "city-mpg", "highway-mpg", "Price"]
```

In [36]:

```
data.columns = headers
```

In [37]:

```
data.head()
```

Out[37]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsep
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [38]:

```
df = data
```

In [39]:

```
data["Price"]
```

Out[39]:

```
0      13495
1      16500
2      16500
3      13950
4      17450
       ...
200    16845
201    19045
202    21485
203    22470
204    22625
Name: Price, Length: 205, dtype: object
```

In [40]:

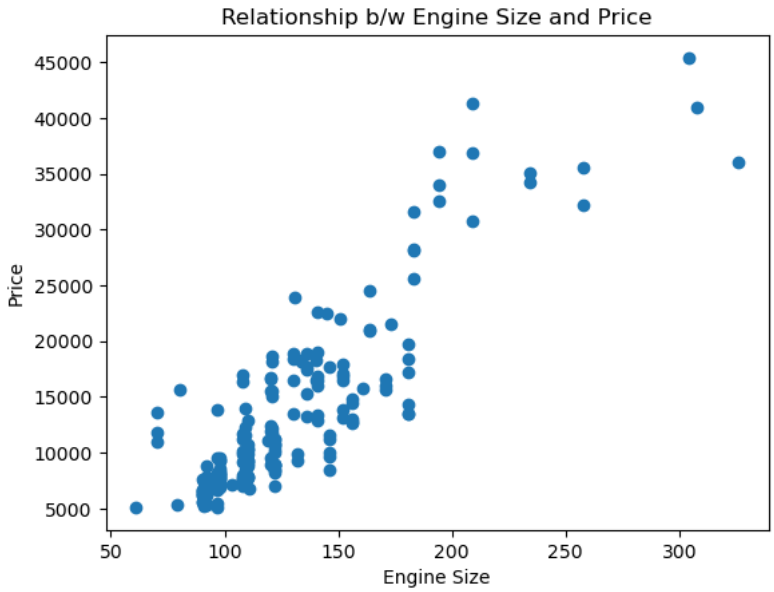
```
data["Price"].replace("?", np.nan, inplace = True)
data["Price"] = pd.to_numeric(data["Price"])
```

In [41]:

```
plt.scatter(data["engine-size"], data["Price"])
plt.title("Relationship b/w Engine Size and Price")
plt.xlabel("Engine Size")
plt.ylabel("Price")
```

Out[41]:

```
Text(0, 0.5, 'Price')
```



NetworkX

In [42]:

```
import networkx as nx
```

In [43]:

```
# Create an empty graph
G = nx.Graph()
```

In [44]:

```
# Adding nodes manually
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
```

In [45]:

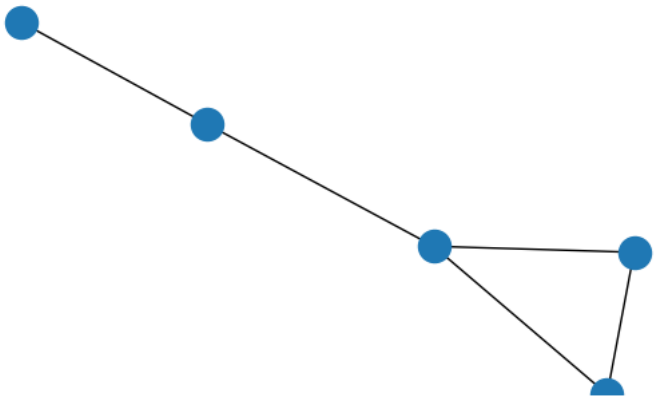
```
G.add_nodes_from([5,6])
```

In [46]:

```
# Adding an edge
G.add_edge(1,2)
G.add_edge(1,3)
G.add_edge(3,1)
G.add_edge(2,4)
G.add_edge(4,1)
G.add_edge(4,5)
G.add_edge(5,6)
```

In [47]:

```
# Drawing graph
nx.draw(G)
```



In [48]:

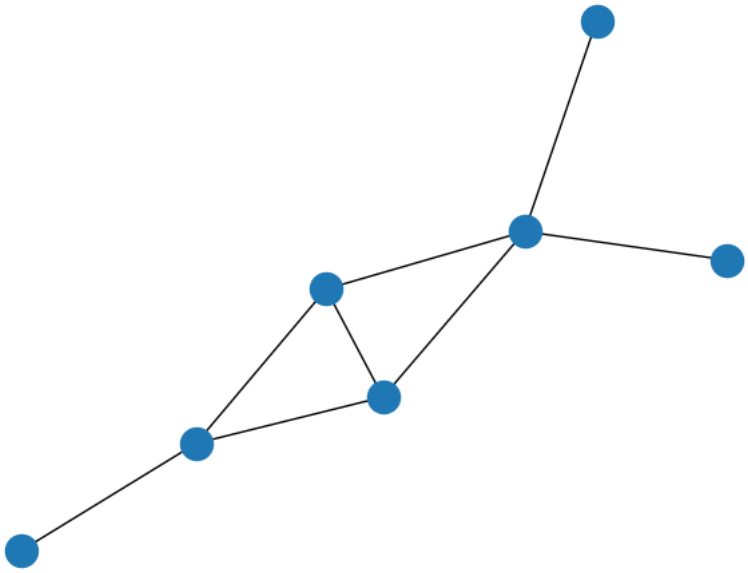
```
# Adding edge between non-existent nodes
G.add_edge(1,9)
```

In [49]:

```
# Adding edges from a list
G.add_edges_from ([(1,4), (2,5)])
```

In [50]:

```
# Drawing graph
nx.draw(G)
```

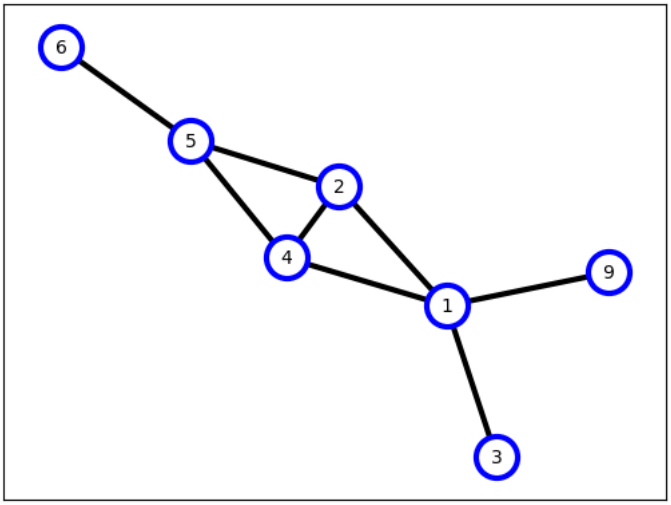


In [51]:

```
# Drawing options
options = {
    "font_size": 10,
    "node_size": 500,
    "node_color": "white",
    "edgecolors": "blue",
    "linewidths":3,
    "width": 3,
}
```

In [52]:

```
# Drawing with options
nx.draw_networkx(G, **options)
```



In [53]:

```
# Writing graph to a text file
nx.write_edgelist(G,"graph.txt")
```

Creating a simple graph from a csv file

In [54]:

```
# Read data from csv file as data fram
df = pd.read_csv('SimpleGraph.csv')
```

In [55]:

```
# Creating an empty graph and then assigning data frame
G = nx.from_pandas_edgelist(df)
```

In [56]:

```
# Print nodes
print(G.nodes)
```

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

In [57]:

```
# Print edges
print(G.edges)
```

```
[(1, 2), (1, 3), (1, 5), (2, 4), (3, 5), (5, 4)]
```

In [58]:

```
# Print number of nodes
print(G.number_of_nodes())
```

```
5
```

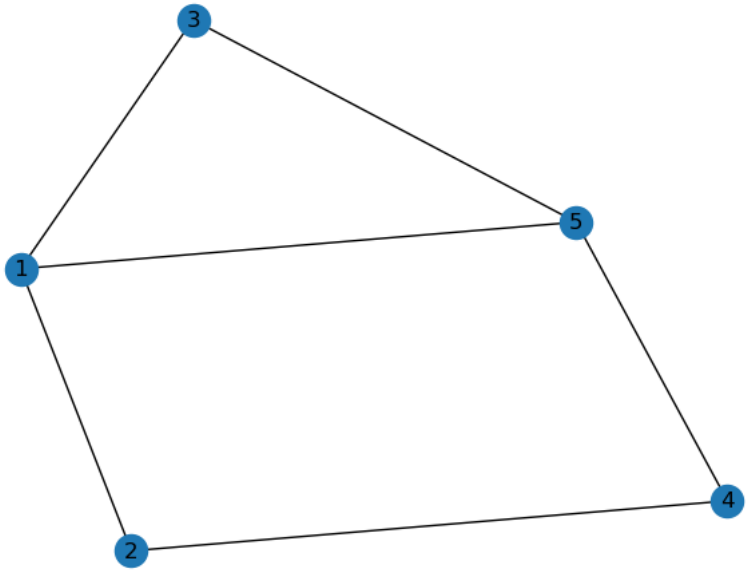
In [59]:

```
# Print number of edges
print(G.number_of_edges())
```

```
6
```

In [60]:

```
# Draw graph with labels
nx.draw(G, with_labels=True)
```



In [61]:

```
# View node 1
G[1]
```

Out[61]:

```
AtlasView({2: {}, 3: {}, 5: {}})
```

In [62]:

```
#Degree of node 1
G.degree[1]
```

Out[62]:

```
3
```

In [63]:

```
# Degrees of node 1 and 3
G.degree([1,3])
```

Out[63]:

```
DegreeView({1: 3, 3: 2})
```

LAB TASKS

Task 1.

Write a NumPy program to create an array of all the even integers from 200 to 240.

In [64]:

```
import numpy as np
even = np.arange(200,240,2)  # program to generate array of even numbers
print(even)
```

```
[200 202 204 206 208 210 212 214 216 218 220 222 224 226 228 230 232 234
 236 238]
```

In [65]:

```
even
```

Out[65]:

```
array([200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224,
       226, 228, 230, 232, 234, 236, 238])
```

TASK_02

Write a NumPy program to generate an array of 10 random numbers from a standard normal distribution.

In [66]:

```
# np.random.normal(mean of the normal distribution, standard deviation, no of samples)
random_num = np.random.normal(0,10,10) # program for random numbers from standard normal distribution
print(random_num)
```

```
[ 5.32554577e+00  9.12374200e-03 -6.18804677e+00 -8.19749238e+00
 -1.03335069e+01 -1.78645628e+01  1.07471863e+01 -1.23781583e+01
  1.51570021e+01 -1.97184091e+01]
```

In [67]:

```
random_num
```

Out[67]:

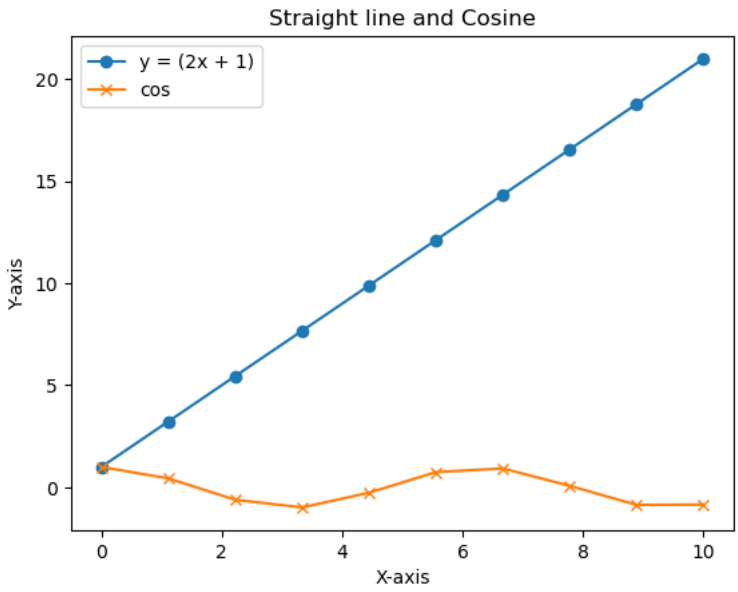
```
array([ 5.32554577e+00,  9.12374200e-03, -6.18804677e+00, -8.19749238e+00,
       -1.03335069e+01, -1.78645628e+01,  1.07471863e+01, -1.23781583e+01,
        1.51570021e+01, -1.97184091e+01])
```

TASK_03

Write a Python program to plot two or more lines and set the line markers. The code snippet gives the output shown in the following screenshot:

In [96]:

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

# (start, stop , number of items to generatev)
# Generate data for the first line
x1 = np.linspace(0,10, 10)
#print(x1)
y1 = (2*x1 + 1) # passing x1 values to function (2x1 + 1)
#print(y1)


# Generate data for the second line
x2 = np.linspace(0,10, 10)
#print(x2)
# passing x2 values to function cos(x2)
y2 = np.cos(x2)
#print(y2)
# Create a figure and axes
fig, ax = plt.subplots()


# Markers just replace them
# Plot the first line with a square marker
ax.plot(x1, y1, '-o', label = 'y = (2x + 1)')

# Plot the second line with a diamond marker
ax.plot(x2, y2, '-x', label = 'cos')

ax.legend() # Add a legend to the plot

# Set the title and labels
ax.set_title("Straight line and Cosine")
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")

# Show the plot
plt.show()
#ax.plot()
```



TASK_04

Write a Python program to change the datatype of a given column or a Series

In [69]:

```
import pandas as pd

series1 = pd.Series(['100', '200', 'python', '300.12', '400'])
print("\nOriginal  Series:\n")
print(series1)

print("\nObject data type to numeric:\n")
series2 = pd.to_numeric(series1, errors = 'coerce') # creating a numeriic series2 from series1 where data is Non_numereic
print(series2)
```

```
Original  Series:

0       100
1       200
2    python
3    300.12
4       400
dtype: object

Object data type to numeric:

0    100.00
1    200.00
2       NaN
3    300.12
4    400.00
dtype: float64
```

TASK_05

Write a pandas program to delete DataFrame row(s) based on given column value.

In [72]:

```
given_data = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}
df1 = pd.DataFrame(data = given_data)
print("\n Given DataFrame \n")
print(df1)
df2 = df1[df1.col2 != 5] # creating dataframe where column value is not 5
print("\n New DataFrame \n")
print(df2)
```

```
 Given DataFrame

   col1  col2  col3
0     1     4     7
1     4     5     8
2     3     6     9
3     4     7     0
4     5     8     1

 New DataFrame

   col1  col2  col3
0     1     4     7
2     3     6     9
3     4     7     0
4     5     8     1
```

In [ ]: