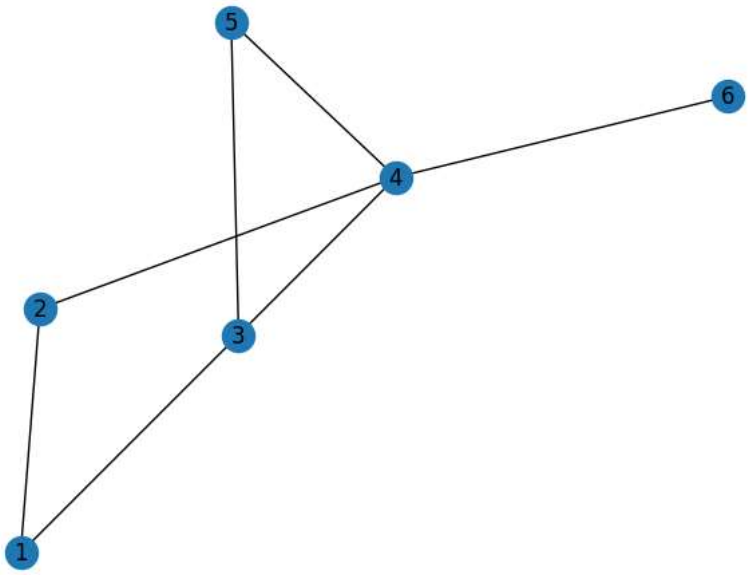
```
import networkx as nx
import numpy as np
```

Que1

```
G1 = nx.Graph()

G1.add_node(1)
G1.add_node(2)
G1.add_node(3)
G1.add_node(4)
G1.add_node(5)
G1.add_node(6)

G1.add_edges_from([(1,2 ), (1,3), (2,4), (4,3), (4,6), (3,5), (5,4) ])
```

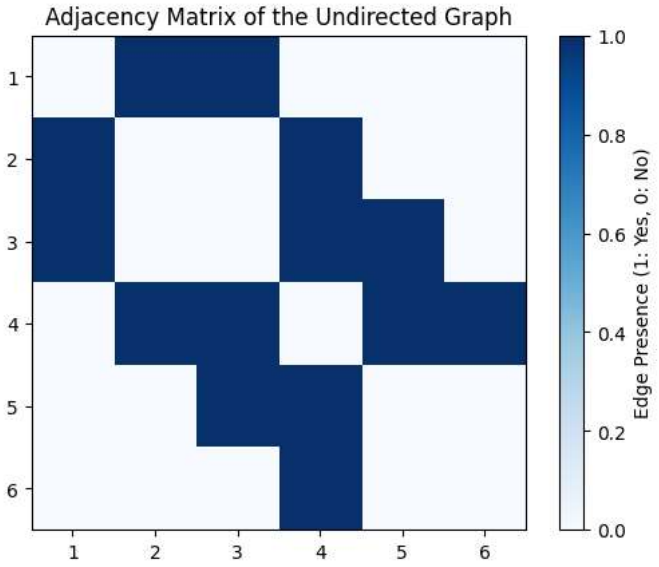
```
import matplotlib.pyplot as plt
```

```
nx.draw(G1,with_labels=True)
```



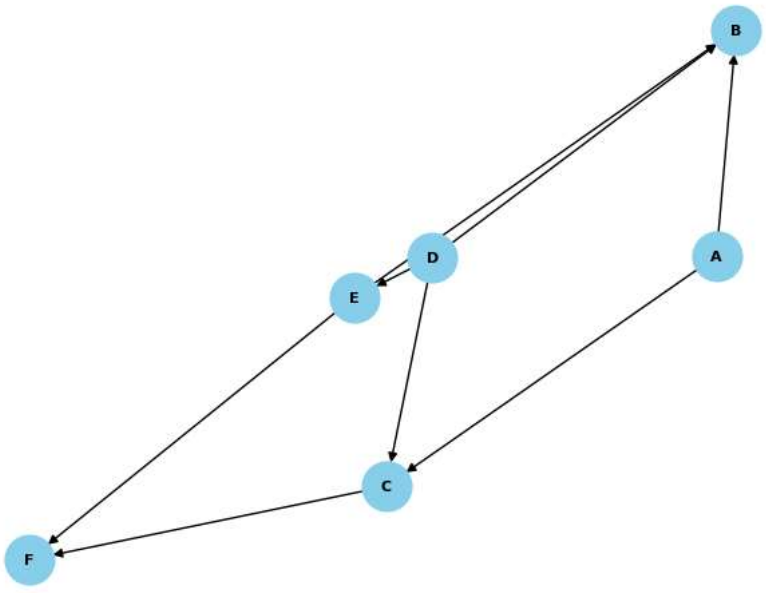

```
nodes = list(G1.nodes())
num_nodes = len(nodes)

# Initialize the adjacency matrix with zeros
adj_matrix = np.zeros((num_nodes, num_nodes))

# Fill the matrix for undirected edges
for edge in G1.edges():
    source, target = edge
    source_idx, target_idx = nodes.index(source), nodes.index(target)
    adj_matrix[source_idx, target_idx] = 1
    adj_matrix[target_idx, source_idx] = 1  # Symmetric entry

# Display the adjacency matrix
print("Adjacency Matrix:")
print(adj_matrix.astype(int))

# Draw the heatmap
plt.imshow(adj_matrix, cmap="Blues", interpolation="none")
plt.colorbar(label="Edge Presence (1: Yes, 0: No)")
plt.xticks(np.arange(num_nodes), nodes)
plt.yticks(np.arange(num_nodes), nodes)
plt.title("Adjacency Matrix of the Undirected Graph")
plt.show()
```

```
Adjacency Matrix:
[[0 1 1 0 0 0]
 [1 0 0 1 0 0]
 [1 0 0 1 1 0]
 [0 1 1 0 1 1]
 [0 0 1 1 0 0]
 [0 0 0 1 0 0]]
```



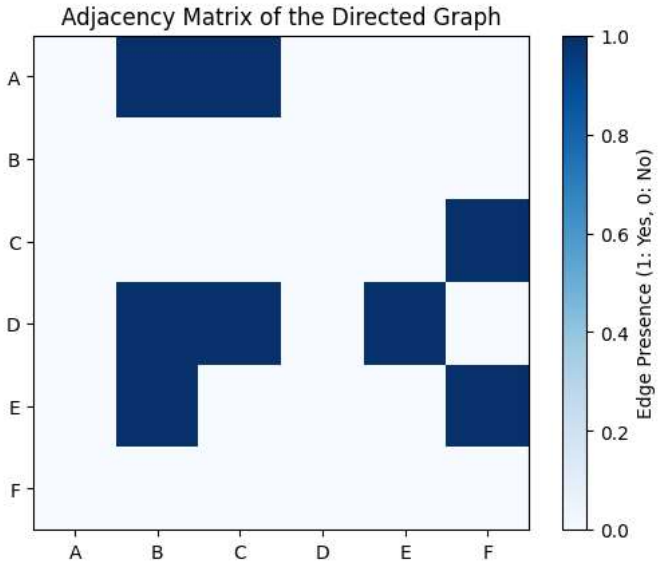
Que 2

```
G2 = nx.DiGraph()

G2.add_node("A")
G2.add_node("B")
G2.add_node("C")
G2.add_node("D")
G2.add_node("E")
G2.add_node("F")

G2.add_edges_from([("A","B" ), ("A","C"), ("D","B"), ("D","C"), ("D","E"), ("E","B"), ("E","F") ,("C","F")])


nx.draw(G2,with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_color='black', font_size=8, arrowsize=10)
```



```
# Get all nodes in the order of appearance
nodes = list(G2.nodes())
num_nodes = len(nodes)

# Initialize the adjacency matrix with zeros
adj_matrix = np.zeros((num_nodes, num_nodes))

# Fill the matrix with 1 for directed edges
for edge in G2.edges():
    source, target = edge
    source_idx, target_idx = nodes.index(source), nodes.index(target)
    adj_matrix[source_idx, target_idx] = 1

# Display the adjacency matrix
print("Adjacency Matrix:")
print(adj_matrix.astype(int))

# Draw the heatmap
plt.imshow(adj_matrix, cmap="Blues", interpolation="none")
plt.colorbar(label="Edge Presence (1: Yes, 0: No)")
plt.xticks(np.arange(num_nodes), nodes)
plt.yticks(np.arange(num_nodes), nodes)
plt.title("Adjacency Matrix of the Directed Graph")
plt.show()
```

```
Adjacency Matrix:
[[0 1 1 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 1]
 [0 1 1 0 1 0]
 [0 1 0 0 0 1]
 [0 0 0 0 0 0]]
```

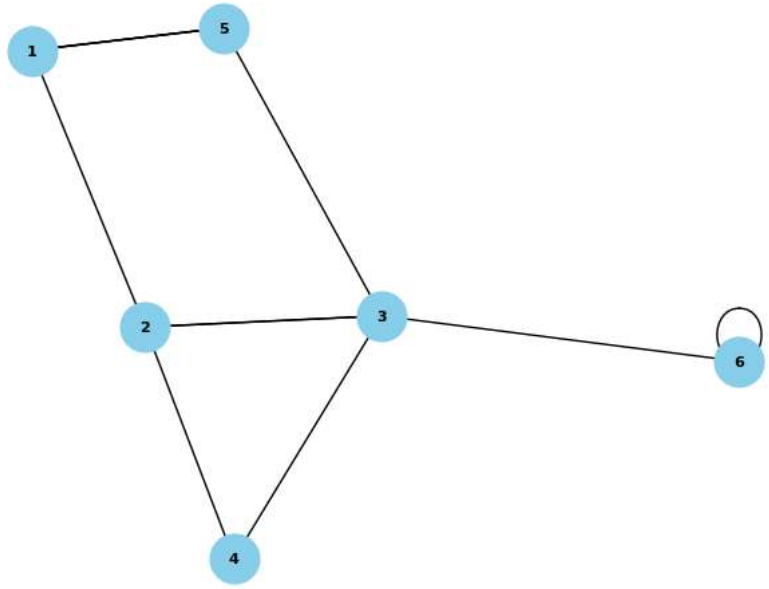


QUE 3 MULTI GRAPH

```
G3 = nx.MultiGraph()

G3.add_node(1)
G3.add_node(2)
G3.add_node(3)
G3.add_node(4)
G3.add_node(5)
G3.add_node(6)

G3.add_edges_from([(1,2 ), (2,3),(2,3), (1,5),(1,5),(1,5),  (5,3), (4,2), (3,4), (3,6), (6,6) ])
```

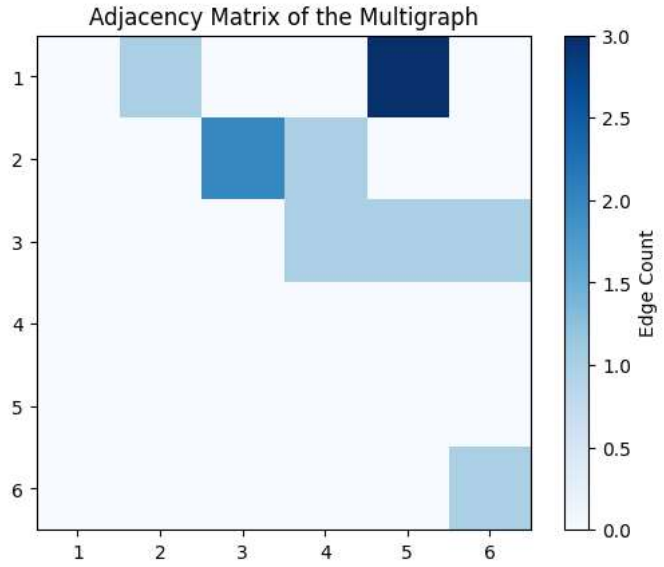
```
[0, 0, 1, 0, 1, 2, 0, 0, 0, 0, 0]
```

```
nx.draw(G3,with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_color='black', font_size=8, arrowsize=10)
```



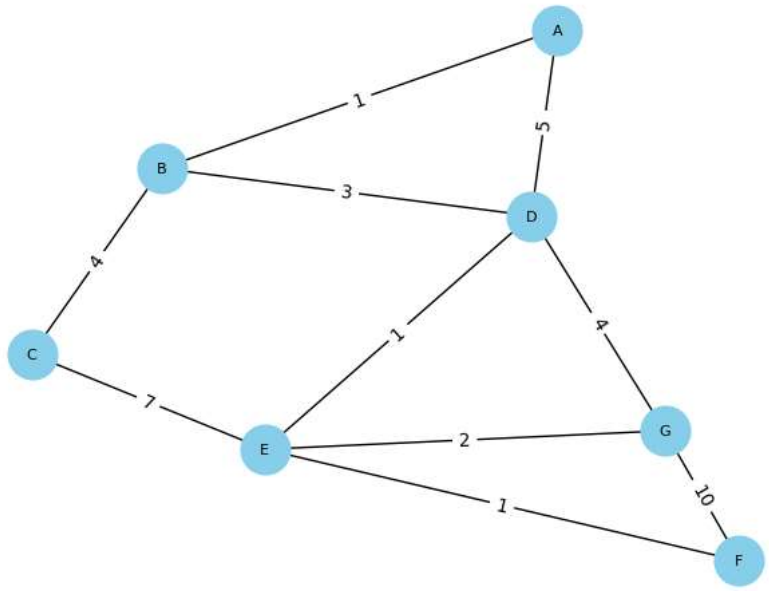

```
nodes = list(G3.nodes())
num_nodes = len(nodes)

# Initialize the adjacency matrix with zeros
adj_matrix = np.zeros((num_nodes, num_nodes))

# Fill the matrix with 1 if an edge exists, or the number of edges for multigraphs
for edge in G3.edges(data=True, keys=True):
    source, target, key = edge[:3]
    source_idx, target_idx = nodes.index(source), nodes.index(target)
    adj_matrix[source_idx, target_idx] += 1

# Display the adjacency matrix
print("Adjacency Matrix:")
print(adj_matrix.astype(int))

# Draw the heatmap
plt.imshow(adj_matrix, cmap="Blues", interpolation="none")
plt.colorbar(label="Edge Count")
plt.xticks(np.arange(num_nodes), nodes)
plt.yticks(np.arange(num_nodes), nodes)
plt.title("Adjacency Matrix of the Multigraph")
plt.show()
```

```
Adjacency Matrix:
[[0 1 0 0 3 0]
 [0 0 2 1 0 0]
 [0 0 0 1 1 1]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 1]]
```



QUE:4 WEIGHTED GRAPH

```
G4 = nx.MultiGraph()

G4.add_node("A")
G4.add_node("B")
G4.add_node("C")
G4.add_node("D")
G4.add_node("E")
G4.add_node("F")
G4.add_node("G")

G4.add_weighted_edges_from([("A", "B", 1), ("A", "D", 5), ("B", "D", 3),("B", "C", 4), ("C", "E", 7), ("D", "E", 1), ("D", "G", 4), ("G
                            ("G", "F", 10), ("E", "F", 1)])



pos = nx.spring_layout(G4)  # You can try other layout algorithms

edge_labels = {(u, v): d['weight'] for u, v, d in G4.edges(data=True)}

nx.draw(G4, pos, with_labels=True, node_size=700, node_color="skyblue", font_size=8, font_color="black")
nx.draw_networkx_edge_labels(G4, pos, edge_labels=edge_labels)

plt.show()
```




```
# Get all nodes in the order of appearance
nodes = list(G4.nodes())
num_nodes = len(nodes)

# Initialize the adjacency matrix with zeros
adj_matrix = np.zeros((num_nodes, num_nodes))

# Fill the matrix with edge weights
for edge in G4.edges(data=True):
    source, target, weight = edge
    source_idx, target_idx = nodes.index(source), nodes.index(target)
    adj_matrix[source_idx, target_idx] = weight['weight']
    adj_matrix[target_idx, source_idx] = weight['weight']  # Symmetric entry

# Display the adjacency matrix
print("Adjacency Matrix:")
print(adj_matrix)

# Draw the heatmap
plt.imshow(adj_matrix, cmap="Blues", interpolation="none")
plt.colorbar(label="Edge Weight")
plt.xticks(np.arange(num_nodes), nodes)
plt.yticks(np.arange(num_nodes), nodes)
plt.title("Adjacency Matrix of the Weighted Graph")
plt.show()
```

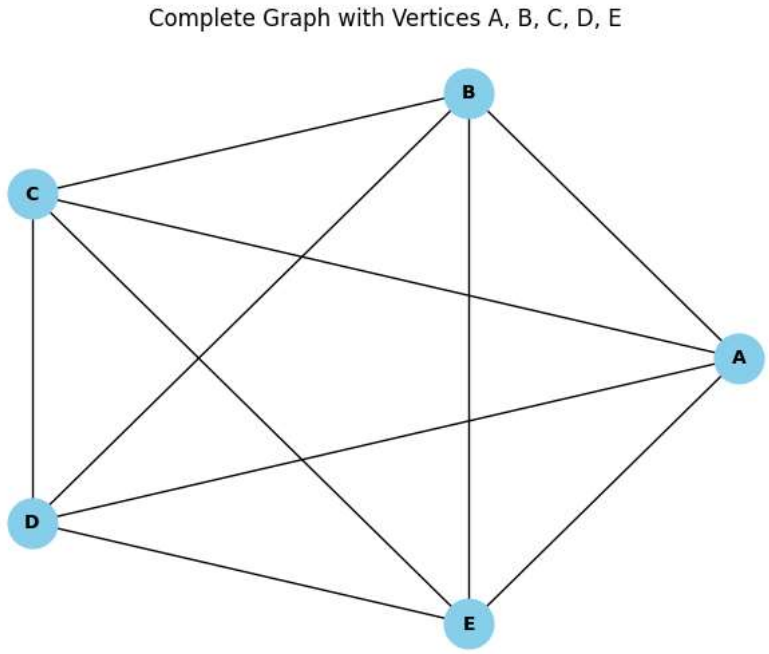
```
Adjacency Matrix:
[[ 0.  1.  0.  5.  0.  0.  0.]
 [ 1.  0.  4.  3.  0.  0.  0.]
 [ 0.  4.  0.  0.  7.  0.  0.]
```

QUE 5: COMPLETE GRAPH

```
 [ 0.  0.  0.  4.  2. 10.  0.]]
```

```
G5= nx.complete_graph(5)

mapping = {0: 'A', 1:'B', 2:'C', 3:'D', 4:'E'}

G5= nx.relabel_nodes(G5, mapping)
```

```
pos = nx.circular_layout(G5)
nx.draw(G5, pos, with_labels=True, node_size=700, node_color="skyblue", font_size=10, font_color="black", font_weight="bold", edge_colo

plt.title("Complete Graph with Vertices A, B, C, D, E")
plt.show()
```




```
nodes = list(G5.nodes())
num_nodes = len(nodes)
```