Basics

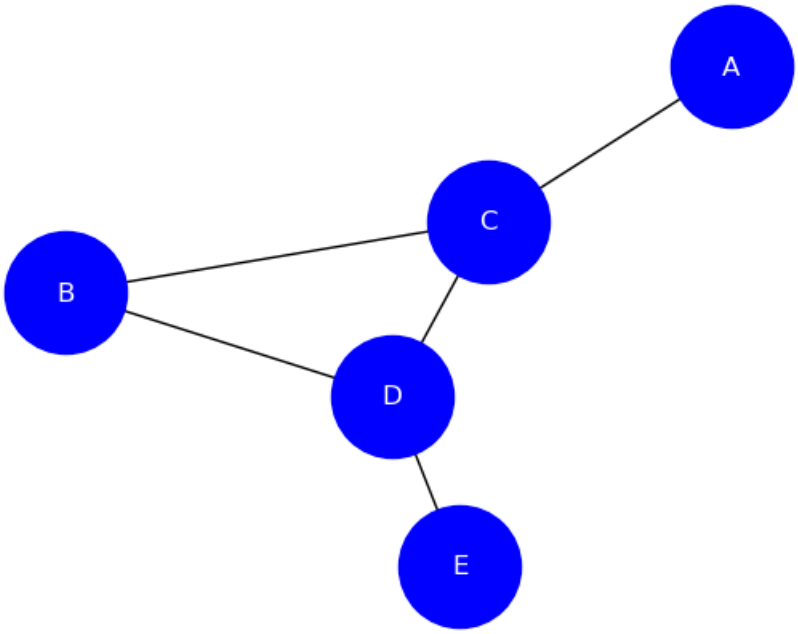
In [2]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import random
```

Create a graph

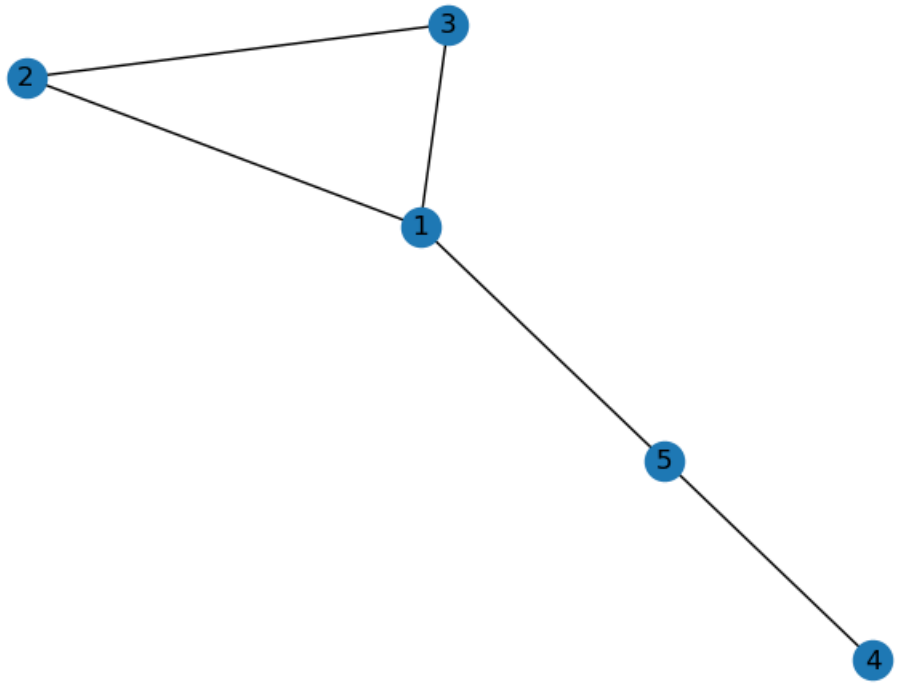
In [3]:
```
g = nx.Graph()
```

In [4]:
```
# Adding nodes
g.add_node('A')
g.add_node('B')
g.add_node('C')
g.add_node('D')
g.add_node('E')

# Adding edges
g.add_edge('A','C')
g.add_edge('B','C')
g.add_edge('B','D')
g.add_edge('C','D')
g.add_edge('D','E')
```

In [6]:
```
nx.draw(g,node_color = 'blue',node_size = 3000,with_labels = True,font_color= 'white')
plt.margins(0.2)
```



In [19]:
```
# Another simpler way of creating graphs
newg = nx.Graph()
newg.add_nodes_from([1,2,3,4,5])
newg.add_edges_from([(1,2),(1,3),(1,5),(3,2),(4,5)])

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



Getting node degree and node neighbour

In [22]:
```
print('Set of nodes',g.nodes)
print('Set of edges',g.edges)
```

```
Set of nodes ['A', 'B', 'C', 'D', 'E']
Set of edges [('A', 'C'), ('B', 'C'), ('B', 'D'), ('C', 'D'), ('D', 'E')]
```

In [26]:
```
for i in g.nodes:
    print(f"Degree of node({i}) = {g.degree(i)}")
```

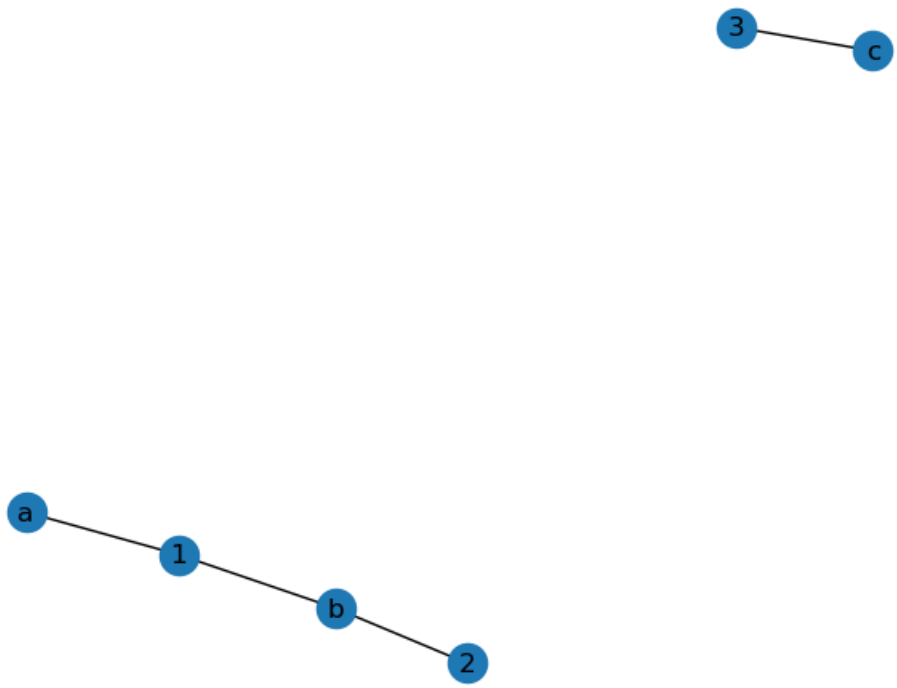
```
Degree of node(A) = 1
Degree of node(B) = 2
Degree of node(C) = 3
Degree of node(D) = 3
Degree of node(E) = 1
```

U3 : Bipartite matching

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms import bipartite
```

Creating a bipartite graph

In [2]:
```
B = nx.Graph()
top_nodes = [1,2,3]
bottom_nodes = ["a","b","c"]
edges = [(1, 'a'), (1, 'b'), (2, 'b'), (3, 'c')]
B.add_nodes_from(top_nodes,bipartite = 0)
B.add_nodes_from(bottom_nodes,bipartite = 1)
B.add_edges_from(edges)
nx.draw(B,with_labels = True)
```



In [3]:
```
def greedy_maximum_matching(B,top_nodes,bottom_nodes):
    matching = set()
    matched = set()
    for u,v in B.edges:
        if u not in matched and v not in matched:
            matching.add((u,v))
            matched.add(u)
            matched.add(v)
    return matching
```

Check if graph is bipartite or not?

In [4]:
```
print('The condition that graph is bipartite is',bipartite.is_bipartite(B))
```

```
The condition that graph is bipartite is True
```

Perform greedy max matching

In [8]:
```
greedy_matching = greedy_maximum_matching(B,top_nodes,bottom_nodes)
print(greedy_matching)
```

```
{(3, 'c'), (1, 'a'), (2, 'b')}
```

Visualizing Matches

In [9]:
```
pos = nx.bipartite_layout(B,top_nodes)
matching_edges = list(greedy_matching)
edge_colours = ['red' if (u,v) in matching_edges or (v,u) in matching_edges else 'black' for (u,v)
nx.draw(B,pos = pos,edge_color = edge_colours,with_labels = True)
plt.title("Bipartite Graph with Greedy Maximum Matching")
```

Out[9]:
```
Text(0.5, 1.0, 'Bipartite Graph with Greedy Maximum Matching')
```