1. Introduction to Networkx

A module in Python for studying graphs.

In [1]:

```python
# import networkx module before use
import networkx as nx

# create a Graph object
G0 = nx.Graph()
```

In [2]:

```python
help(nx.Graph)
```

...

In [3]:

```python
# use the graph object to create nodes
G0.add_node(1)
G0.add_node(2)
G0.add_node(3)
```

In [4]:

```python
# to create an edge between a pair of nodes
G0.add_edge(1,2)
G0.add_edge(2,3)
```

In [5]:

```python
# to list nodes
print ('nodes =', G0.nodes())
```

```
nodes = [1, 2, 3]
```

In [6]:

```python
help(nx.nodes)
```

```
Help on function nodes in module networkx.classes.function:

nodes(G)
    Return an iterator over the graph nodes.
```

In [7]:

```python
# to list edges
print ('edges =', G0.edges())
```

```
edges = [(1, 2), (2, 3)]
```

In [8]:

```
help(nx.edges)
```

```
Help on function edges in module networkx.classes.function:

edges(G, nbunch=None)
    Return an edge view of edges incident to nodes in nbunch.

    Return all edges if nbunch is unspecified or nbunch=None.

    For digraphs, edges=out_edges
```

In [9]:

```
G1 = nx.Graph()

# a node whose id is a string
G1.add_node("michael")

# a node whose id is even a float, but be careful (not recommended)
G1.add_node(0.3)

G1.add_edge("michael", 34)     # node 34 has not yet been created.
G1.add_edge(0.3, 34)

print ('nodes =', G1.nodes())
print ('edges =', G1.edges())
```

```
nodes = ['michael', 0.3, 34]
edges = [('michael', 34), (0.3, 34)]
```

What-you-see may not be what-you-get

In [10]:

```
G2 = nx.Graph()

G2.add_node(0.3)
G2.add_node(1)

x = 0.1 + 0.2
print ("x =", x)

G2.add_edge(x, 1)
print ('nodes =', G2.nodes())
```

```
x = 0.30000000000000004
nodes = [0.3, 1, 0.30000000000000004]
```

Add multiple nodes and multple edges at once

In [11]:

```
G33 = nx.Graph()

G33.add_edges_from([(1,2), (1,3), (4,5), (2,4)])
G33.add_node(1)

print ('V = ', G33.nodes())
print ('E = ', G33.edges())
print ('|V| =', G33.number_of_nodes())
print ('|E| =', G33.number_of_edges())
```

```
V =  [1, 2, 3, 4, 5]
E =  [(1, 2), (1, 3), (2, 4), (4, 5)]
|V| = 5
|E| = 4
```

Remove nodes and edges

In [12]:

```
G3 = nx.Graph()
G3.add_nodes_from([1,2,3,4,5])
G3.add_edges_from([(1,2), (1,3), (4,5), (2,4)])
print ('V = ', G3.nodes())
print ('E = ', G3.edges())

# edges are removed by a node
G3.remove_node(1)
print ('After removing node 1')
print ('V = ', G3.nodes())
print ('E = ', G3.edges())
```

```
V =  [1, 2, 3, 4, 5]
E =  [(1, 2), (1, 3), (2, 4), (4, 5)]
After removing node 1
V =  [2, 3, 4, 5]
E =  [(2, 4), (4, 5)]
```

In [13]:

```
G3.remove_edge(2,4)
print ('After removing edge (2,4)')
print ('V = ', G3.nodes())
print ('E = ', G3.edges())
```

```
After removing edge (2,4)
V =  [2, 3, 4, 5]
E =  [(4, 5)]
```

Graph attributes

In [14]:

```python
G = nx.Graph(name='IoT', date='2016/2/10')
print (G.graph)
print (G.graph['name'])
print (G.graph['date'])

# you can modify attributes later using dictionary notation
G.graph['date'] = '2016/3/1'
print (G.graph)
```

```
{'name': 'IoT', 'date': '2016/2/10'}
IoT
2016/2/10
{'name': 'IoT', 'date': '2016/3/1'}
```

Node attributes

In [15]:

```python
G = nx.Graph()
G.add_node(1, time='5pm')
G.add_nodes_from([2,3], time='2pm')
print ('nodes =', G.nodes(data=True))

print ('Add room attribute to node 1')
G.node[1]['room'] = 714
print ('nodes =', G.nodes(data=True))
print ('node 1 =', G.node[1])
print (G.node[1]['room'])
print (G.node[1]['time'])

```

```
nodes = [(1, {'time': '5pm'}), (2, {'time': '2pm'}), (3, {'time': '2pm'})]
Add room attribute to node 1
nodes = [(1, {'time': '5pm', 'room': 714}), (2, {'time': '2pm'}), (3, {'time':
'2pm'})]
node 1 = {'time': '5pm', 'room': 714}
714
5pm
```

Edge attributes

In [16]:

```python
G4 = nx.Graph()
# create nodes when edges were added
G4.add_edge(1, 2, action = 'teach')
G4.add_edge(2, 3, distance = '5nm')
G4.add_edge(2, 4, action = 'send', distance = '6mile')
G4.add_edge(3, 5)
print ('nodes =', G4.nodes(data=True))
print ('edges =', G4.edges(data=True))
```

```
nodes = [(1, {}), (2, {}), (3, {}), (4, {}), (5, {})]
edges = [(1, 2, {'action': 'teach'}), (2, 3, {'distance': '5nm'}), (2, 4, {'act
ion': 'send', 'distance': '6mile'}), (3, 5, {})]
```

In [17]:

```python
# get neighbors
print ('neighbors of node 2 =', G4[2])
# get an edge directly
print ('attribute of edge (2,4) =', G4[2][4])

# you can set or modify edge attribute by using the graph index
G4[2][4]['action'] = 'learn'
G4[2][4]['distance'] = '100km'

print ('attribute of edge (2,4) =', G4[2][4])
```

```
neighbors of node 2 = {1: {'action': 'teach'}, 3: {'distance': '5nm'}, 4: {'act
ion': 'send', 'distance': '6mile'}}
attribute of edge (2,4) = {'action': 'send', 'distance': '6mile'}
attribute of edge (2,4) = {'action': 'learn', 'distance': '100km'}
```

Graph generation

In [18]:

```python
k5 = nx.complete_graph(5)
print (k5.nodes())
print (k5.edges())
```

```
[0, 1, 2, 3, 4]
[(0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3,
4)]
```

In [19]:

```python
k32 = nx.complete_bipartite_graph(3,2)
print (k32.nodes())
print (k32.edges())
```

```
[0, 1, 2, 3, 4]
[(0, 3), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4)]
```

In [20]:

```python
peter = nx.petersen_graph()
print (peter.nodes())
print (peter.edges())
```

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

There are many functions for generating other graphs. Look for Networkx reference manual for details.

Graph operations

In [21]:

```
k6 = nx.complete_graph(6)
k4 = nx.subgraph(k6, [1,2,3,4])
print (k4.nodes())
print (k4.edges())
```

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

In [22]:

```
help(nx.subgraph)
```

...

In [23]:

```
k5 = nx.complete_graph(5)
print ("origin k5   =", k5.edges())
k5.remove_edges_from([(1,2), (2,3)])
print ("modified k5 =", k5.edges())
c6 = nx.complement(k5)
print ("compl'ed c6 =", c6.edges())
```

```
origin k5   = [(0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3),
(2, 4), (3, 4)]
modified k5 = [(0, 1), (0, 2), (0, 3), (0, 4), (1, 3), (1, 4), (2, 4), (3, 4)]
compl'ed c6 = [(1, 2), (2, 3)]
```

In [24]:

```
help(nx.complement)
```

...

Look for Networkx reference manual for details.

Read and write graph

```
graph [
  node [ id 0 label "0" ]
  node [ id 1 label "1" ]
  node [ id 2 label "2" ]
  node [ id 3 label "3" ]
  node [ id 4 label "4" ]
  node [ id 5 label "5" ]
  node [ id 6 label "6" ]
  edge [ source 0 target 1 weight 1 ]
  edge [ source 0 target 4 weight 3 ]
  edge [ source 0 target 6 weight 2 ]
  edge [ source 1 target 2 weight 5 ]
  edge [ source 1 target 3 weight 2 ]
  edge [ source 2 target 5 weight 2 ]
  edge [ source 2 target 6 weight 3 ]
  edge [ source 4 target 6 weight 1 ]
  edge [ source 5 target 6 weight 5 ]
]
```

In [25]:

```
mygraph=nx.read_gml("Graphs/g35.gml", label='id')
print (mygraph.nodes())
print (mygraph.edges())
```

```
[0, 1, 2, 3, 4, 5, 6]
[(0, 1), (0, 4), (0, 6), (1, 2), (1, 3), (2, 5), (2, 6), (4, 6), (5, 6)]
```

In [26]:

```
help(nx.read_gml)
```

...

In [27]:

```
mygraph=nx.read_gml("Graphs/g35.gml")
nx.write_gexf(mygraph, "test.gexf")
```

In [28]:

```
help(nx.write_gexf)
```

...

Analyzing algorithms

In [29]:

```
G=nx.Graph()
G.add_edges_from([(1,2),(1,3)])
G.add_node("spam")
print ('degree =', nx.degree(G))
```

```
degree = [(1, 2), (2, 1), (3, 1), ('spam', 0)]
```

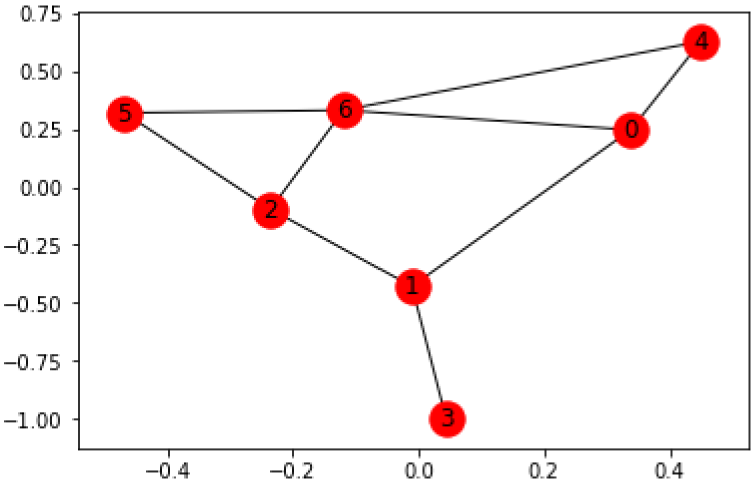
Drawing graphs

In [30]:

```
%matplotlib inline
```

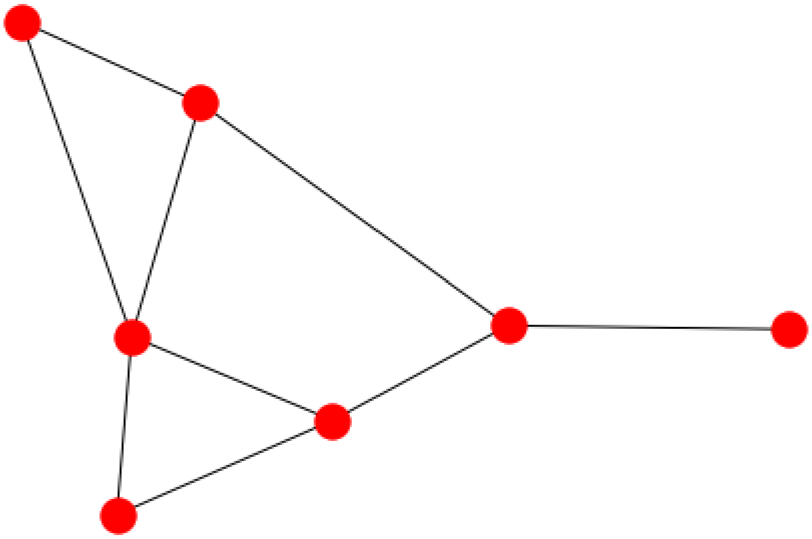
In [31]:

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.read_gml("Graphs/g35.gml")
nx.draw_networkx(G)
plt.show()
```



In [32]:

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.read_gml("Graphs/g35.gml")
nx.draw(G)
plt.savefig('g35.png')
```



In [33]:

```
# clean temporary files for this lecture
import os
try:
    os.remove('test.gexf')
    os.remove('g35.png')
except WindowsError:
    pass
```

In [ ]:

```

```