# NetworkX

## High−productivity software for complex networks

*Speaker : Lo Pang-Yun Ting*

# Outline

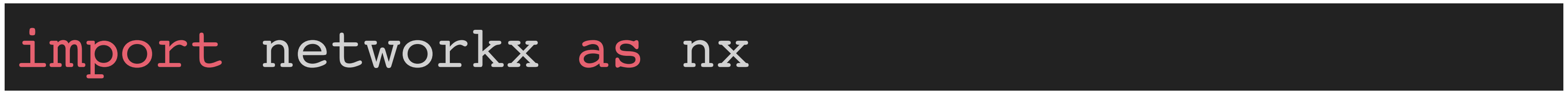
- Introduction
- Starting
- Attributes
- Directed graphs
- Graph operators
- Reading and writing graphs
- Algorithms
- Drawing graphs

# Introduction

- NetworkX is a Python library for studying graphs and networks
  - Classes for various networks, e.g. undirected, directed …
  - Conversion of graphs to and from several formats
  - Lots of graphs algorithms
  - Flexible data structure

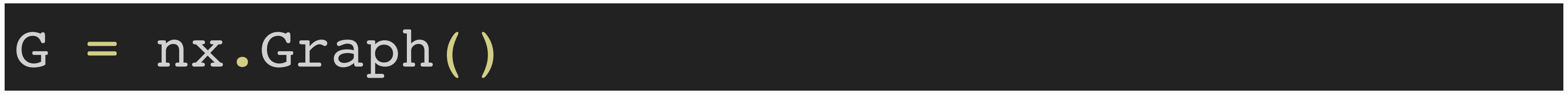
# Starting

- Creating a graph
  - Import network



```
import networkx as nx
```

  - Create a undirected graph

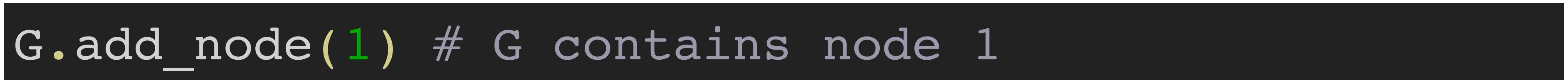


```
G = nx.Graph()
```

G = nx.DiGraph()  *# directed graph*
G = nx.MultiGraph()  *# undirected graph that can store multi edges*
G = nx.MultiDigraph()  *# directed graph that can store multi edges*

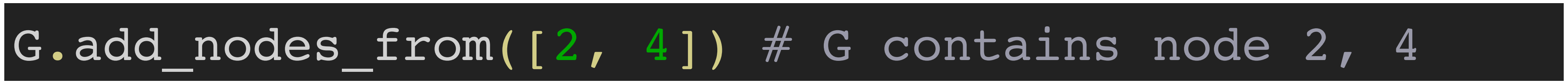
# Starting

- Nodes
  - Add one node



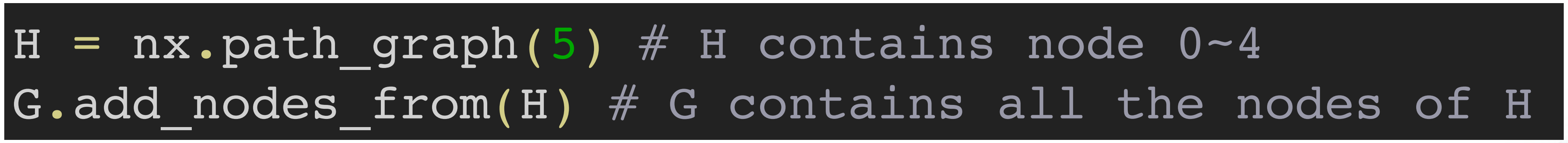
```
G.add_node(1) # G contains node 1
```

  - Add a list of nodes



```
G.add_nodes_from([2, 4]) # G contains node 2, 4
```

  - Add a container of nodes

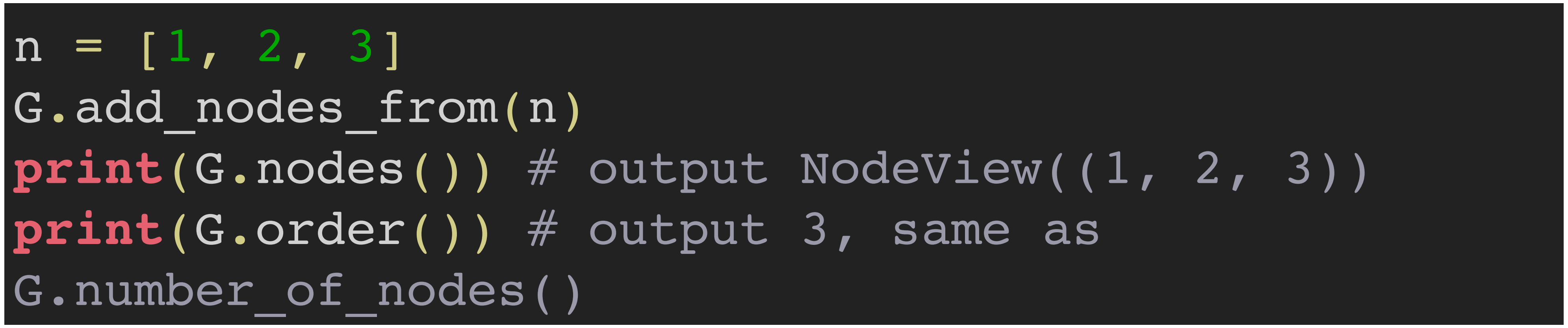


```
H = nx.path_graph(5) # H contains node 0~4
G.add_nodes_from(H) # G contains all the nodes of H
```

G.add_node(‘sna’) *# add node ‘sna’*
G.add_nodes_from(‘sna’) *# add nodes ’s’, ’n’, ‘a’*

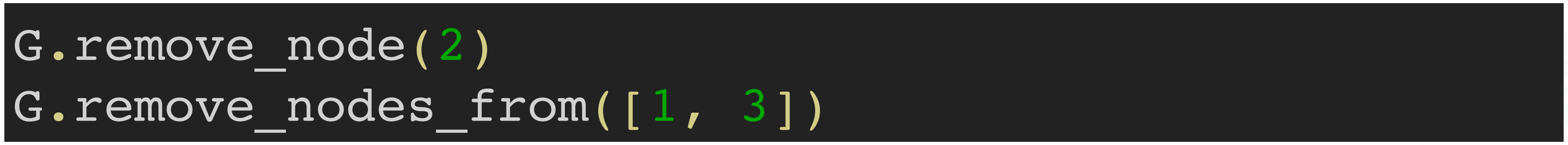
# Starting

- Nodes
  - Access all the nodes and number of nodes



```
n = [1, 2, 3]
G.add_nodes_from(n)
print(G.nodes()) # output NodeView((1, 2, 3))
print(G.order()) # output 3, same as
G.number_of_nodes()
```

  - Remove nodes



```
G.remove_node(2)
G.remove_nodes_from([1, 3])
```

# Starting

- Edges
  - Add single edge

```
G.add_edge(1, 2)
```

  - Add a list of edges

```
e = [(1, 2), (3, 4), (5, 6)]
G.add_edges_from(e)
```

  - Add a container of edges

```
G.add_edges_from(H.edges())
```

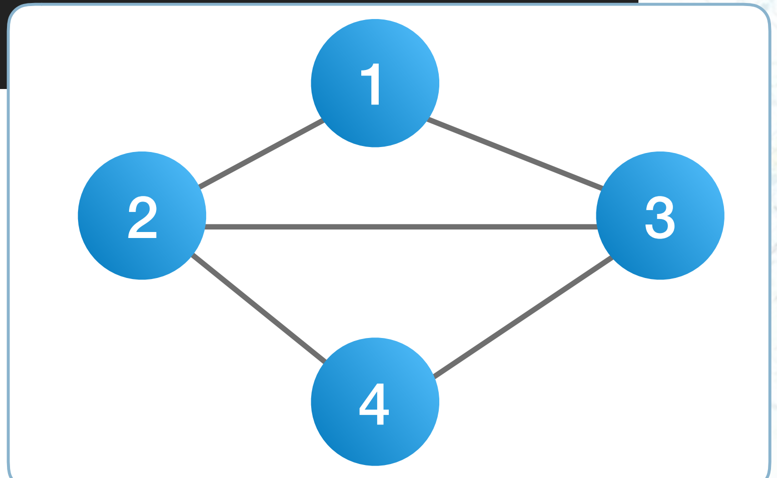
# Starting

- Edges
  - Access all the edges and number of edges

```
e = [(1, 2), (3, 4), (5, 6)]
G.add_edges_from(e)
print(G.edges()) # output EdgeView([(1, 2), (3, 4), (5,
6)])
print(G.size()) # output 3,  same as G.number_of_edges()
```

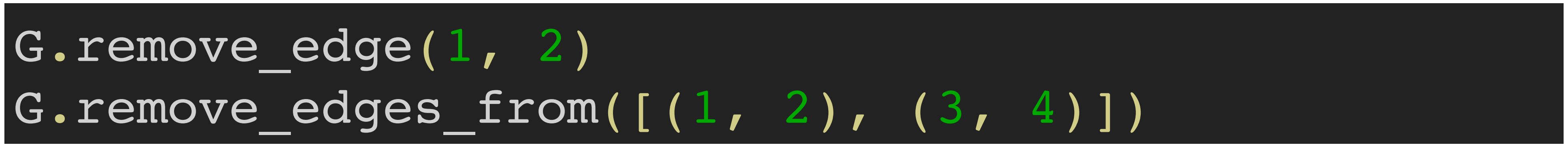
  - Access neighbors and degree

```
e = [(1, 2),(2, 3),(1, 3),(2, 4),(3, 4)]
G.add_edges_from(e)
print(list(G.neighbors(2))) # output [1, 3, 4]
print(G.degree(2)) # output 3
```



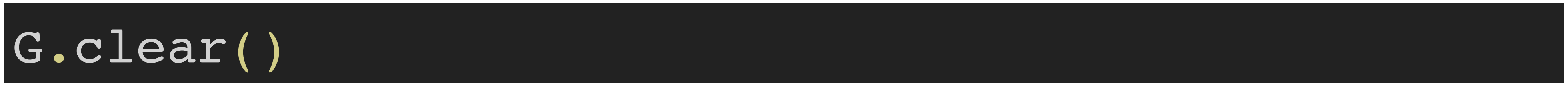
# Starting

- Edges
    - Remove edges



```
G.remove_edge(1, 2)
G.remove_edges_from([(1, 2), (3, 4)])
```

- Clear all the edges and nodes



```
G.clear()
```

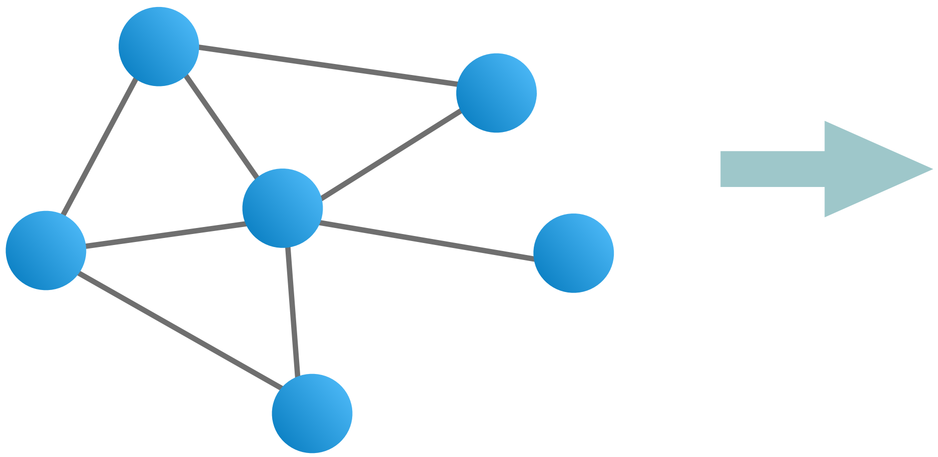
# Attributes

- Graph attributes
  - Assign graph attributes when creating a graph

```
G = nx.Graph(gender = 'girls')
```

  - Modify graph attributes later

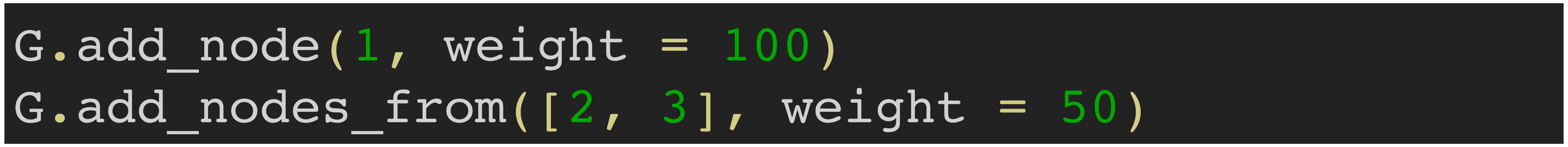
```
G.graph['gender'] = 'boys'
G.graph['college'] = 'ncku'
print(G.graph)
# output {'gender': 'boys', 'college': 'ncku'}
```



*gender: boys*
*college: ncku*

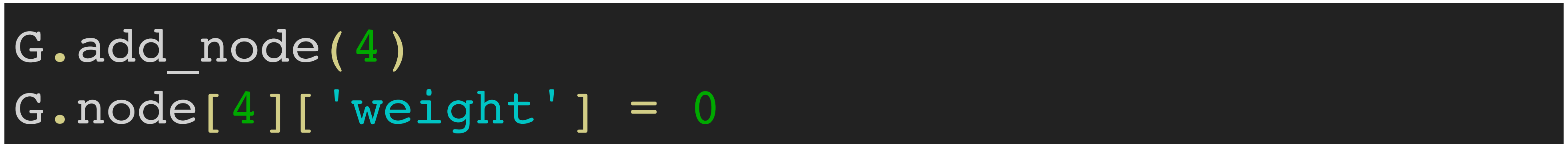
# Attributes

- Node attributes
  - Assign node attributes when adding nodes



```
G.add_node(1, weight = 100)
G.add_nodes_from([2, 3], weight = 50)
```

  - Modify node attributes later



```
G.add_node(4)
G.node[4]['weight'] = 0
```

# Attributes

- Node attributes

```
# show all the nodes
>>> G.nodes()
NodeView((1, 2, 3, 4))

# show all the nodes attributes
>>> G.nodes(data = True)
NodeDataView({1: {'sex': ‘female'}, …})

# show all the nodes attributes
>>> G.node
{1: {'sex': 'female'}, … }

# show the sepecified node attributes (dictionary)
>>> G.node[1]
{'sex': 'female'}
```

# Attributes

- Edge attributes
  - Assign edge attributes when adding edges

```
G.add_edge(1, 2, weight = 3.5)

G.add_edges_from([(3, 4), (4, 5)], weight = 2.0)
G.add_edges_from([(3, 4, {'weight': 2.0}), (4, 5,
{'weight': 3.2})])
```

  - Modify edge attributes later

```
G[1][2]['weight'] = 4
G.edges[1, 2]['weight'] = 4
```

# Attributes

- Edge attributes

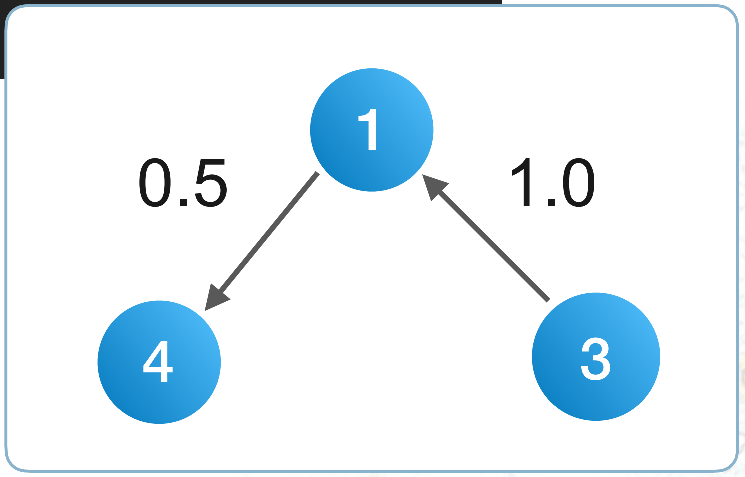
```
# show all the edges
>>> G.edges()
EdgeView([(1, 2), …])


# show all the edges attributes
>>> G.edges(data = True)
EdgeDataView([(1, 2, {‘weight': 4}),…)]) each pair of edge


# show all the edges attributes
>>> G.edge
{1: {2: {’weight’: 4}}, … }  each node


# show the sepecified edge attributes
>>> G.edge[1]
{2: {’weight’: 4}}
```

# Directed Graphs

- Additional properties to directed graphs

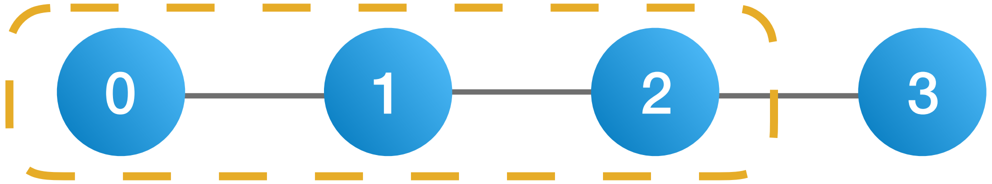
```
DG = nx.DiGraph()
DG.add_weighted_edges_from([(1, 4, 0.5), (3, 1,
1.0)])

print(DG.in_degree(1, weight = 'weight')) # 1.0
print(DG.out_degree(1, weight = 'weight')) # 0.5

print(list(DG.successors(1))) # [4]
print(list(DG.predecessors(1))) # [3]
```

degree() = in_degree() + out_degree()
neighbors() = successors()



# Graph operators

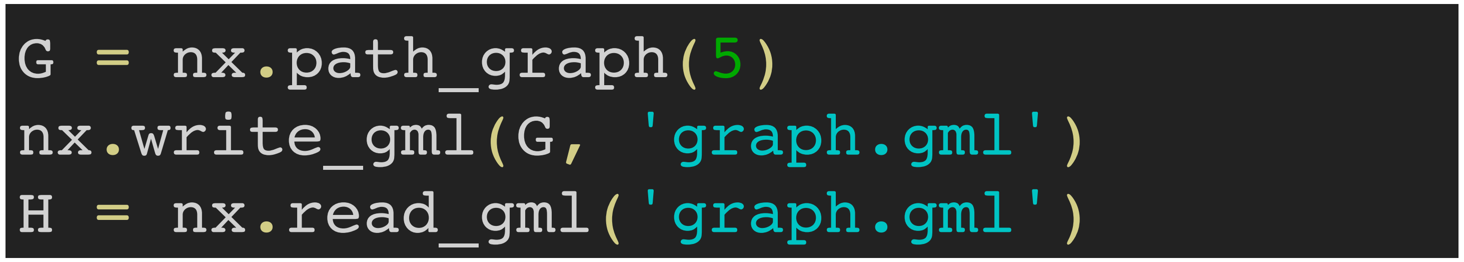
- **subgraph(G, nbunch)** - induce subgraph of G on nodes in bunch
    - E.g. H = nx.subgraph(G, [0, 1, 2])



- **complement(G)** - graph complement
- **create_empty_copy(G)** - return an empty of the same graph class
- **convert_to_undirected(G)** - return an undirected representation of G
    - E.g. H = nx.convert.convert_to_undirected(G) ***# H = G.to_undirected()***
- **convert_to_directed(G)** - return a directed representation of G

# Reading and writing graphs

- Can read/write graphs in GML, GraphML, Pajek … … format
  - Example: read/write GML

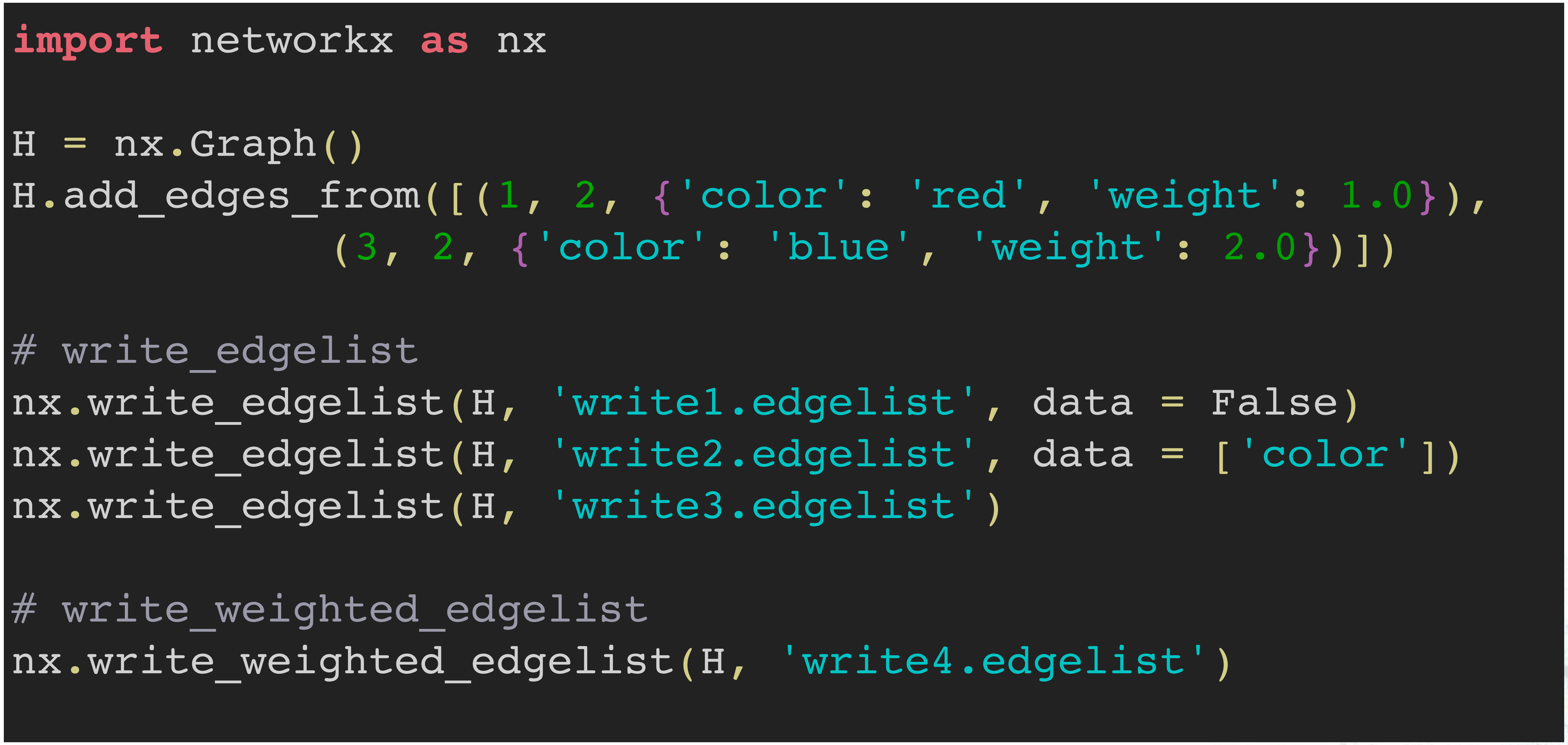


```
G = nx.path_graph(5)
nx.write_gml(G, 'graph.gml')
H = nx.read_gml('graph.gml')
```

- Read/write Adjacency list, Multiline Adjacency List, Edge list

# Reading and writing graphs

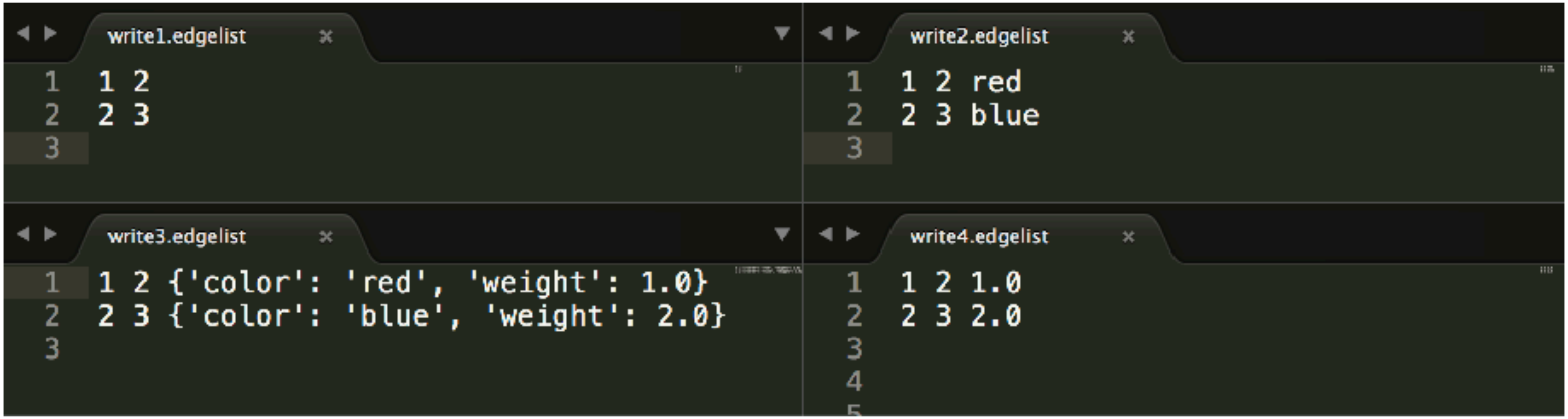
- Example: Write edge list to a file



```python
import networkx as nx

H = nx.Graph()
H.add_edges_from([(1, 2, {'color': 'red', 'weight': 1.0}),
                 (3, 2, {'color': 'blue', 'weight': 2.0})])

# write_edgelist
nx.write_edgelist(H, 'write1.edgelist', data = False)
nx.write_edgelist(H, 'write2.edgelist', data = ['color'])
nx.write_edgelist(H, 'write3.edgelist')

# write_weighted_edgelist
nx.write_weighted_edgelist(H, 'write4.edgelist')
```

# Reading and writing graphs

- Example: Write edge list to a file



```
1 2
2 3
```

```
1 2 red
2 3 blue
```

```
1 2 {'color': 'red', 'weight': 1.0}
2 3 {'color': 'blue', 'weight': 2.0}
```

```
1 2 1.0
2 3 2.0
```

# Reading and writing graphs

- Example: Read edge list from a file

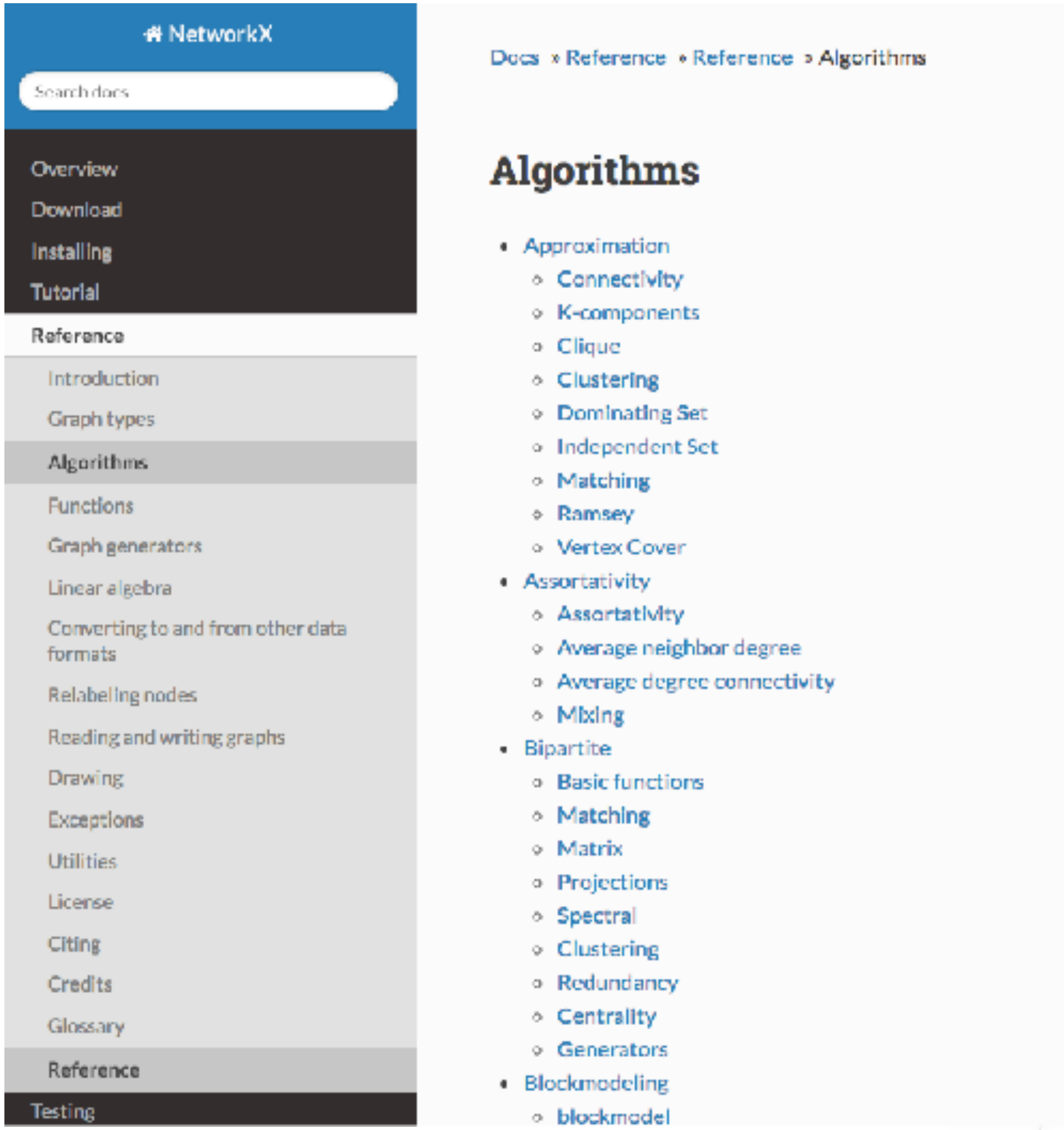
*data.csv*

```
3,4,purple,5.2
1,2,pink,4.6
```

```
import networkx as nx

fpr = open('data.csv', 'rb')

G = nx.read_edgelist(fpr, delimiter = ',', nodetype = int,
             data = (('color', str), ('weight', float)))

print(G.edges(data = True))
#EdgeDataView([(3, 4, {'color': 'purple', 'weight': 5.2}),
(1, 2, {'color': 'pink', 'weight': 4.6})])
```

# Algorithms

- Approximation, Biparitite, Centrality, Clustering …



## Algorithms

- Approximation
  - Connectivity
  - K-components
  - Clique
  - Clustering
  - Dominating Set
  - Independent Set
  - Matching
  - Ramsey
  - Vertex Cover
- Assortativity
  - Assortativity
  - Average neighbor degree
  - Average degree connectivity
  - Mixing
- Bipartite
  - Basic functions
  - Matching
  - Matrix
  - Projections
  - Spectral
  - Clustering
  - Redundancy
  - Centrality
  - Generators
- Blockmodeling
  - blockmodel

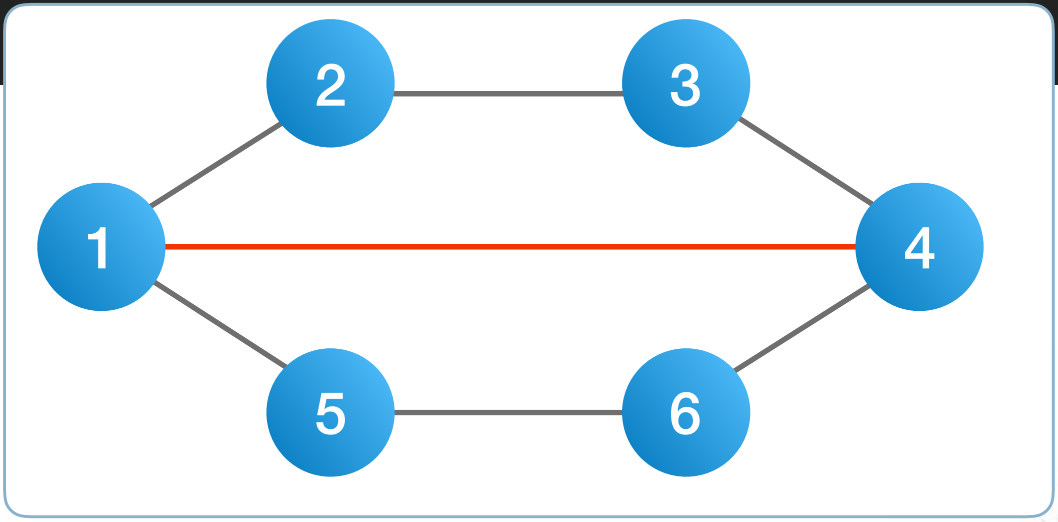
# Algorithms

- Example1: Shortest path

```
import networkx as nx

fpr = open('input.csv', 'rb')
G = nx.read_edgelist(fpr, delimiter = ',',
                               nodetype = int)

p = nx.shortest_path(G, source = 1, target = 4)

print(p) # output [1, 4]
```

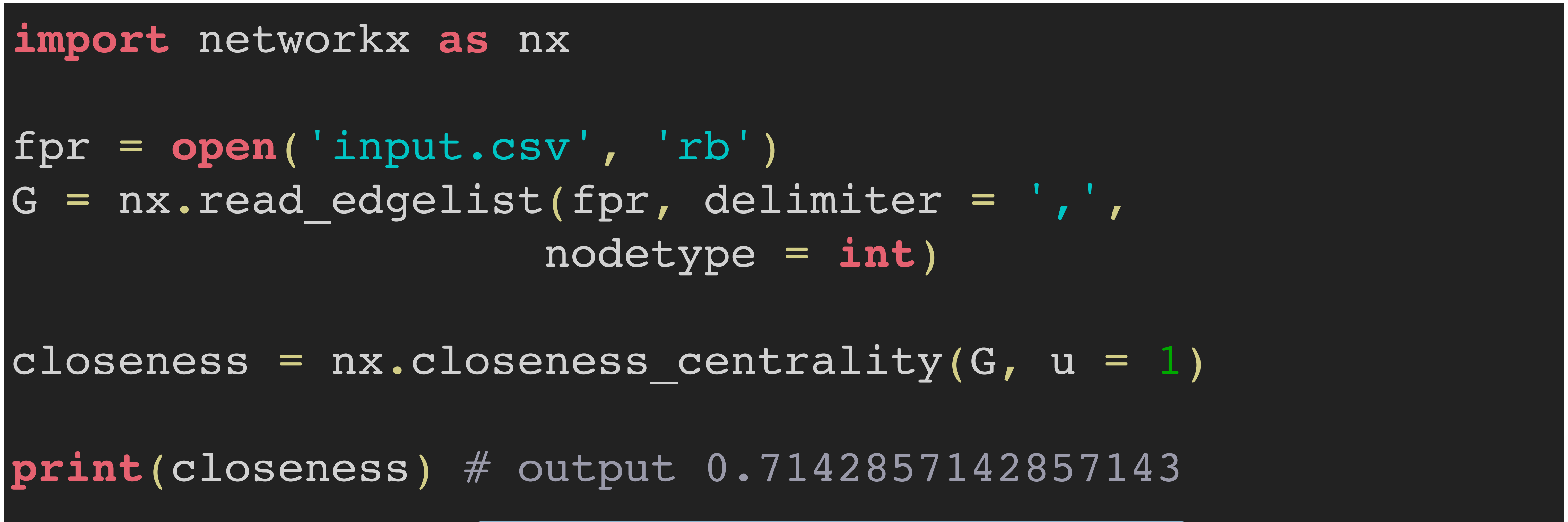
*input.csv*

```
1,2
1,4
1,5
2,3
3,4
4,6
5,6
```

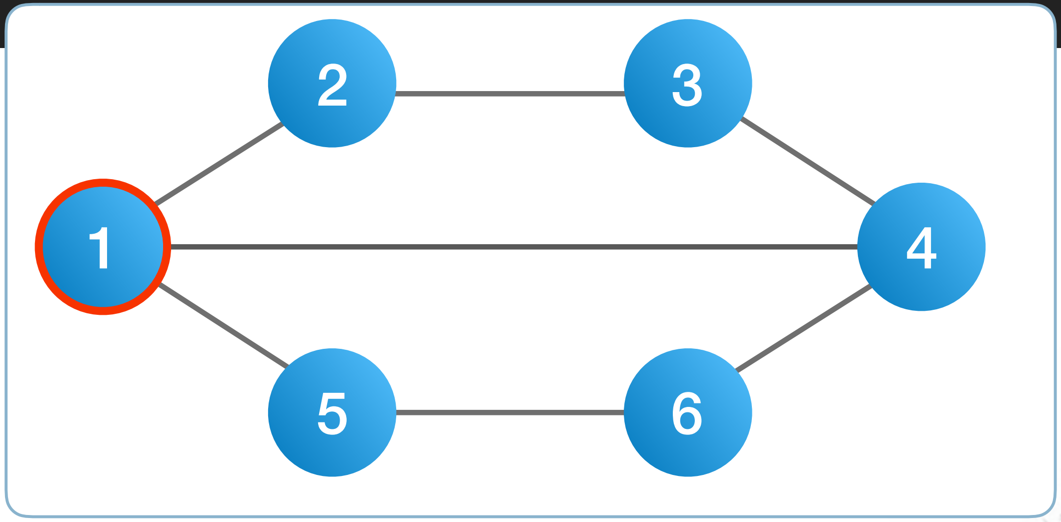


# Algorithms

- Example2: Closeness

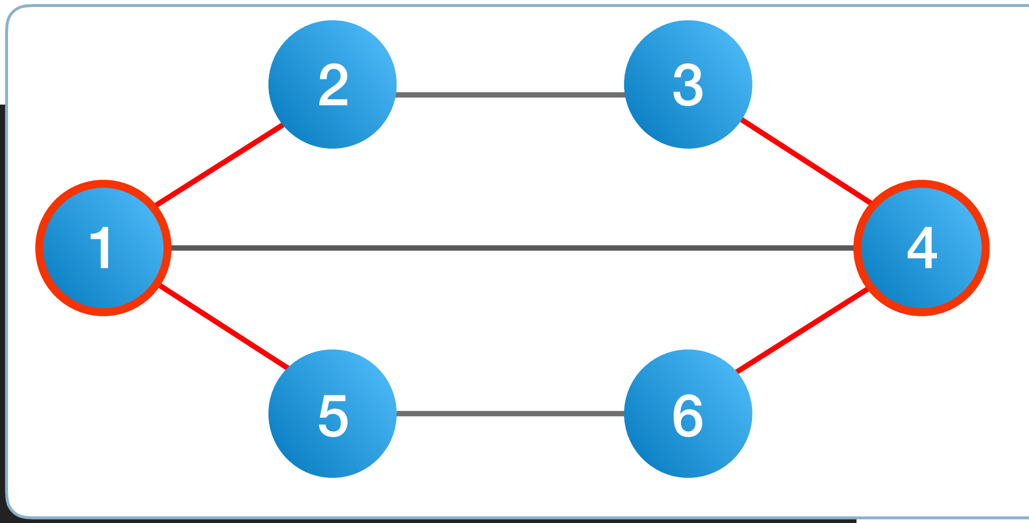


```python
import networkx as nx

fpr = open('input.csv', 'rb')
G = nx.read_edgelist(fpr, delimiter = ',',
                    nodetype = int)

closeness = nx.closeness_centrality(G, u = 1)

print(closeness) # output 0.7142857142857143
```



# Algorithms

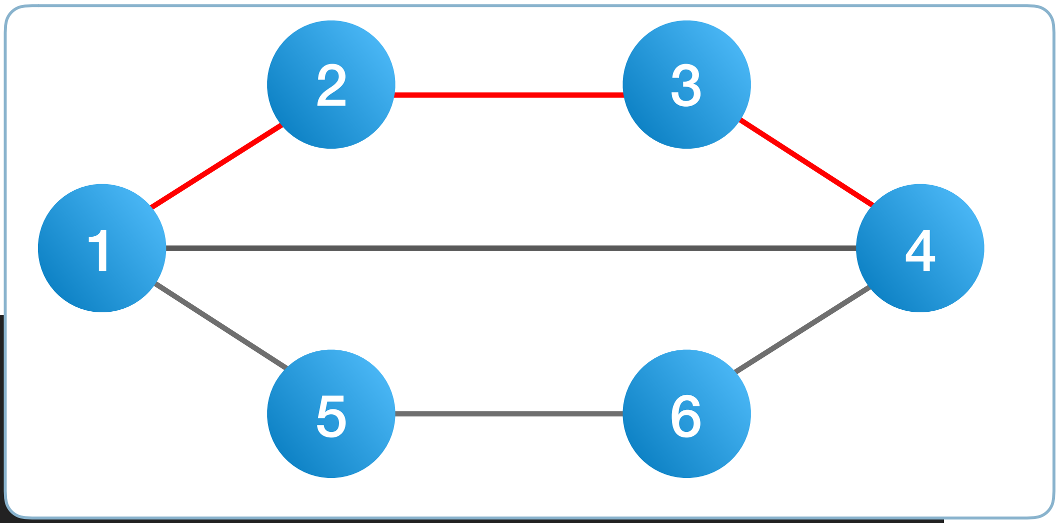
- Example3: Betweenness



```python
import networkx as nx

fpr = open('input.csv', 'rb')
G = nx.read_edgelist(fpr, delimiter = ',',
                        nodetype = int)

bc = nx.betweenness_centrality(G)
edge_bc = nx.edge_betweenness_centrality(G)

print('betweenness centrality:\n', bc)
print('edge betweenness centrality:\n', edge_bc)
'''
betweenness centrality:
 {1: 0.33333333333333337, 2: 0.08333333333333333, 4:
0.3333333333333333, 5: 0.08333333333333333, 3: 0.08333333333333333,
6: 0.08333333333333333}
edge betweenness centrality:
 {(1, 2): 0.26666666666666666, (1, 4): 0.24444444444444444, (1, 5):
0.26666666666666666, (2, 3): 0.17777777777777776, (4, 3):
0.2666666666666666, (4, 6): 0.26666666666666666, (5, 6):
0.17777777777777776}
'''
```

# Algorithms

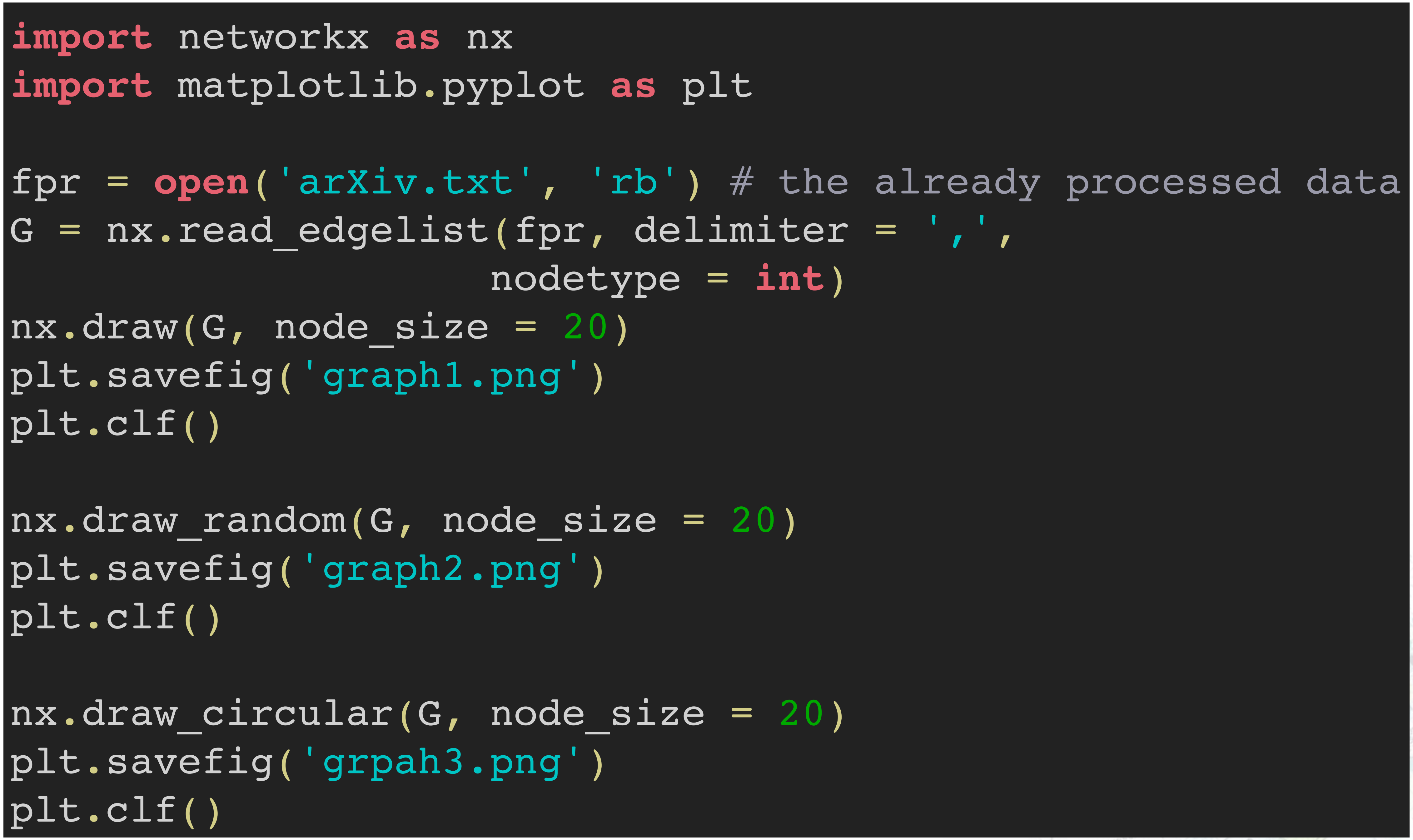
- Example4: Diameter, Density



```
import networkx as nx

fpr = open('input.csv', 'rb')
G = nx.read_edgelist(fpr, delimiter = ',',
                        nodetype = int)

diam = nx.diameter(G)
dens = nx.density(G)

print('diameter:', diam) # diameter: 3
print('density:', dens) # density: 0.4666666666666667
```

# Drawing graphs

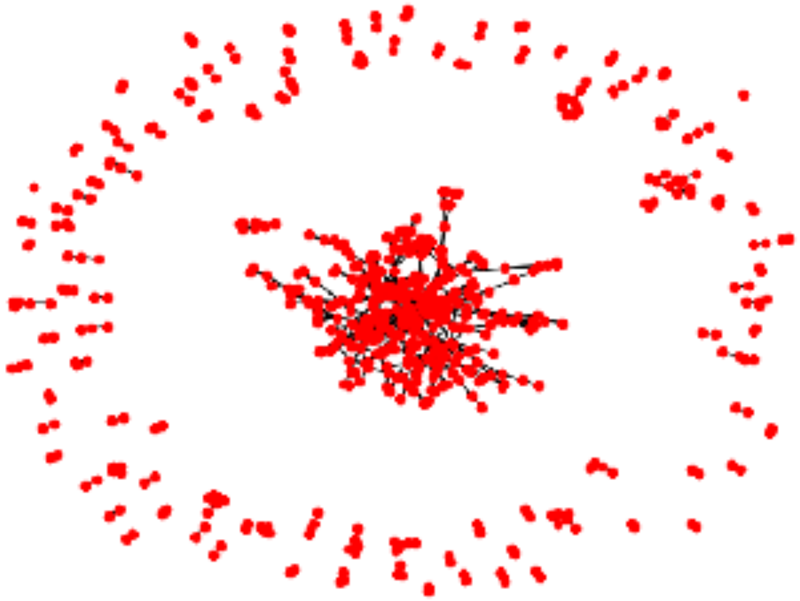
- Example: use arXiv dataset



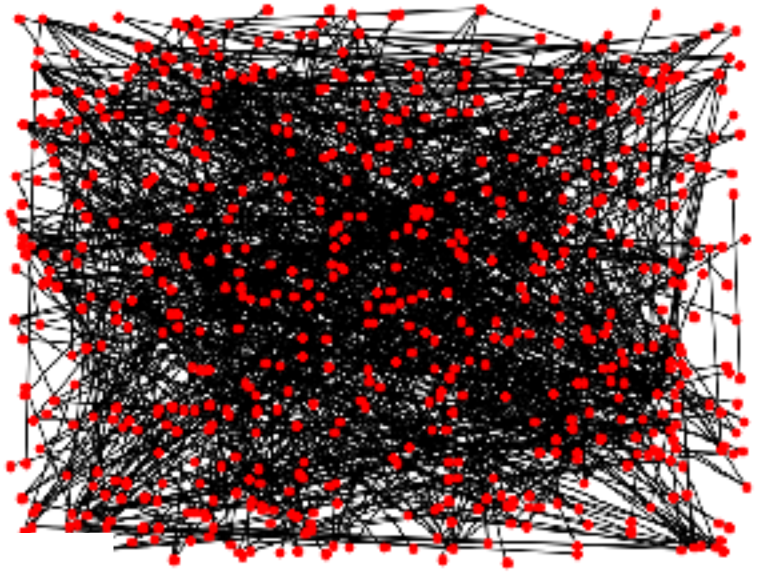
```python
import networkx as nx
import matplotlib.pyplot as plt

fpr = open('arXiv.txt', 'rb') # the already processed data
G = nx.read_edgelist(fpr, delimiter = ',',
                    nodetype = int)
nx.draw(G, node_size = 20)
plt.savefig('graph1.png')
plt.clf()

nx.draw_random(G, node_size = 20)
plt.savefig('graph2.png')
plt.clf()

nx.draw_circular(G, node_size = 20)
plt.savefig('grpah3.png')
plt.clf()
```

# Drawing graphs

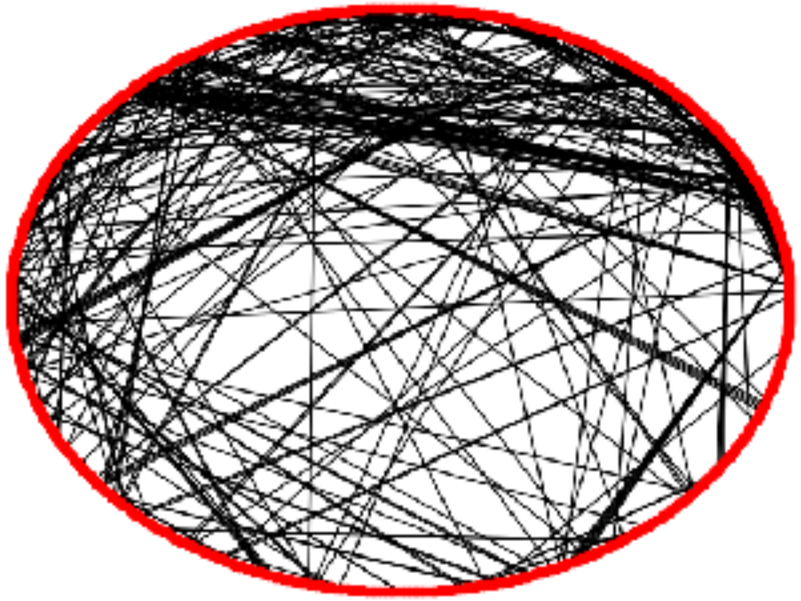
- Example: use arXiv dataset



***graph1.png***



***graph2.png***



***graph3.png***

# More about NetworkX

- NetworkX official website