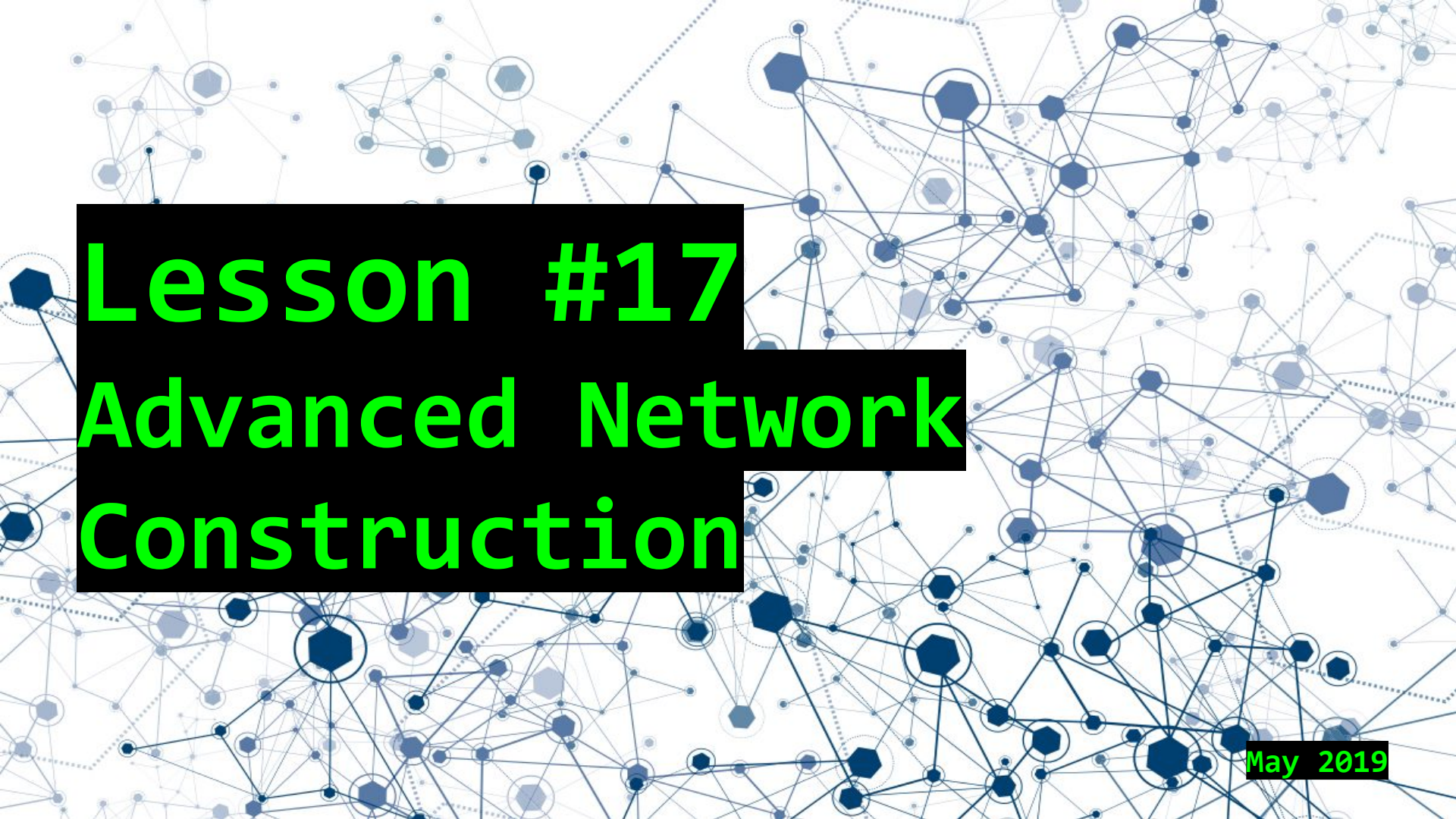
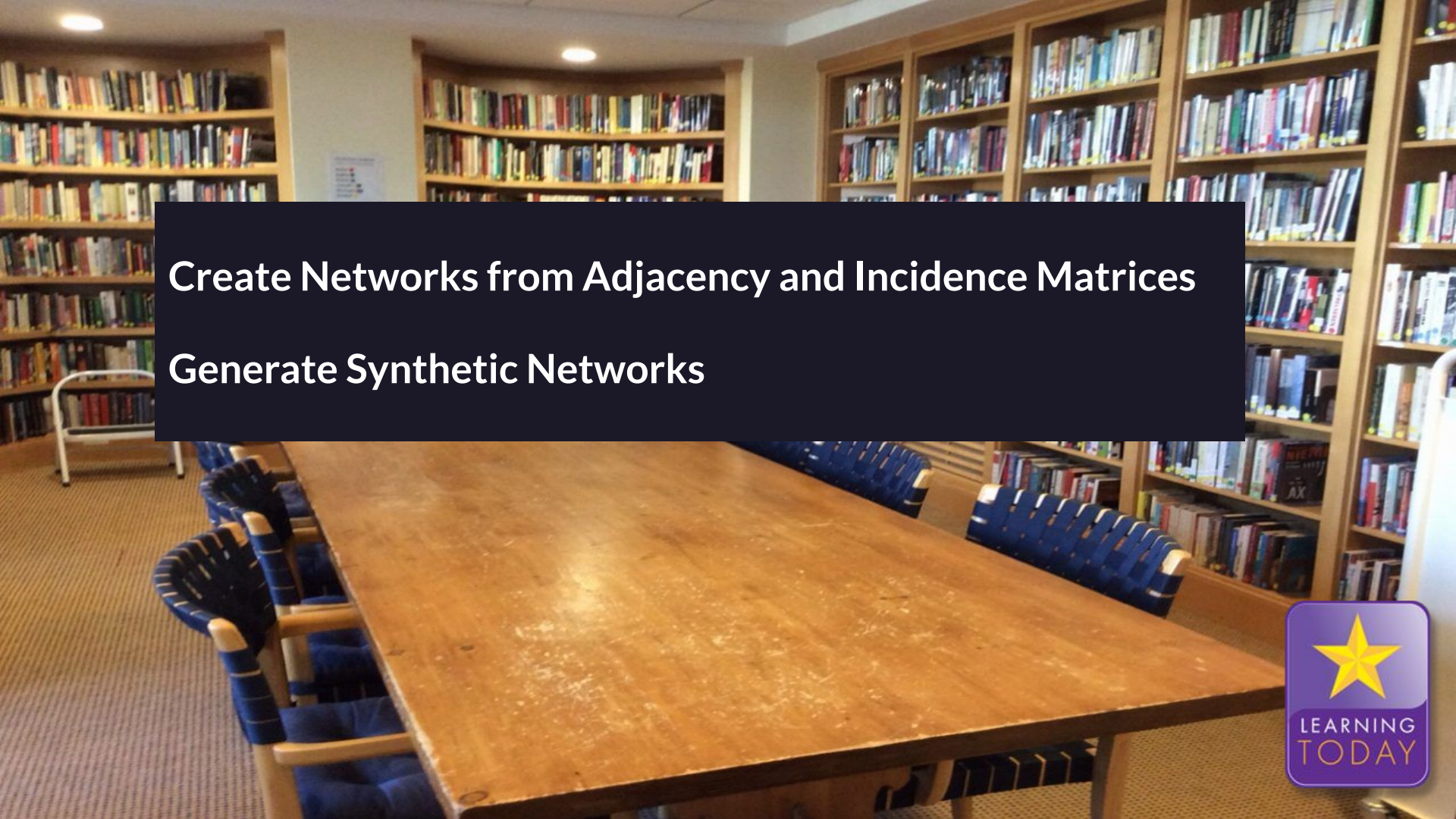

Lesson #17
Advanced Network Construction
May 2019


Create Networks from Adjacency and Incidence Matrices
Generate Synthetic Networks
LEARNING
TODAY

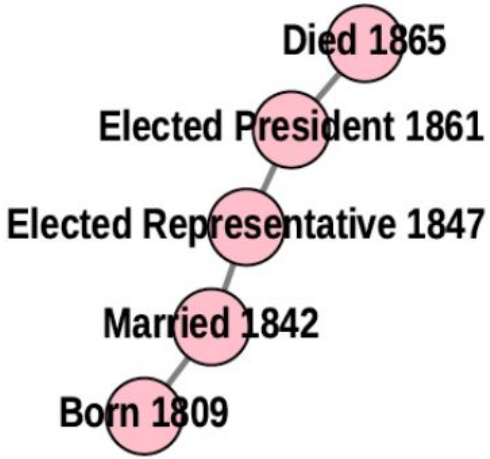
Update from repository

```
git clone https://github.com/ivanovitchm/datascience_one_2019_1
```

Or ....

```
git pull
```

Create Networks from Adjacency and Incidence Matrices



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Incidence Matrix

```
0 1 0 0 0   0 1 0 0 0
0 0 1 0 0   0 0 1 0 0
0 0 0 1 0   0 0 0 1 0
0 0 0 0 1   0 0 0 0 1
0 0 0 0 0   1 0 0 0 0
```

Adjacency Matrix

Adjacency Matrix

- Python way
- Numpy way
- Pandas way

Network type	Adjacency matrix	Interpretation
Simple	Only 0s or 1s, and no 1s on the main diagonal	Absence/presence of an edge
Weighted	At least one floating-point number	Numbers are edge weights
With self-loops	At least one non-zero on the main diagonal	Same as above
Signed	At least one negative number	Same as above
Undirected	Symmetric	Same as above
Multigraph	Not possible	Cannot be represented as an adjacency matrix

Adjacency Matrix

- Python way

```
A = [[0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 0, 0, 0, 1],
     [.1, 0, 0, 0, 0]]
```

```
# directed graph
G = nx.DiGraph()

# nested list comprehension
edges = [(i,j) for i,row in enumerate(A)
               for j,column in enumerate(row) if A[i][j]]

# add edges
G.add_edges_from(edges)
```

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

Adjacency Matrix

- Python way

```
A = [[0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 0, 0, 0, 1],
     [.1, 0, 0, 0, 0]]
```

```
# directed graph
G = nx.DiGraph()

# nested list comprehension
edges = [(i,j, {"weight": A[i][j]}) for i,row in enumerate(A)
         for j,column in enumerate(row) if A[i][j]]

# add edges
G.add_edges_from(edges)
```

```
[(0, 1, {'weight': 1}), (1, 2, {'weight': 1}), (2, 3, {'weight': 1}),
 (3, 4, {'weight': 1}), (4, 0, {'weight': 0.1})]
```

Adjacency Matrix

- Numpy way

```
A = [[0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 0, 0, 0, 1],
     [.1, 0, 0, 0, 0]]
```

```
import numpy as np

A_mtx = np.matrix(A)
G = nx.from_numpy_matrix(A_mtx,
                         create_using=nx.DiGraph())
```

```
[(0, 1, {'weight': 1.0}), (1, 2, {'weight': 1.0}),
 (2, 3, {'weight': 1.0}), (3, 4, {'weight': 1.0}), (4, 0, {'weight': 0.1})]
```

Adjacency Matrix

- Pandas way

	Born	Married	Elected Rep	Elected Pres	Died
Born	0.0	1.0	0.0	0.0	0.0
Married	0.0	0.0	1.0	0.0	0.0
Elected Rep	0.0	0.0	0.0	1.0	0.0
Elected Pres	0.0	0.0	0.0	0.0	1.0
Died	0.1	0.0	0.0	0.0	0.0

```
labels = "Born", "Married", "Elected Rep", "Elected Pres", "Died"

# change nodes label inplace
nx.relabel_nodes(G, dict(enumerate(labels)), copy=False)

# create a dataframe from a networkx graph
df = nx.to_pandas_adjacency(G)
```

Adjacency Matrix

- Pandas way

```
G = nx.from_pandas_edgelist(edges,*edges.columns)
G.edges(data=True)

EdgeDataView([('Born', 'Married', {'weight': 1.0}),
              ('Born', 'Died', {'weight': 0.1}),
              ('Married', 'Elected Rep', {'weight': 1.0}),
              ('Elected Rep', 'Elected Pres', {'weight': 1.0}),
              ('Elected Pres', 'Died', {'weight': 1.0})])
```

	source	target	weight
0	Born	Married	1.0
1	Married	Elected Rep	1.0
2	Elected Rep	Elected Pres	1.0
3	Elected Pres	Died	1.0
4	Died	Born	0.1

```
edges = nx.to_pandas_edgelist(G)
```

Incidence Matrices


```
J = nx.incidence_matrix(G, oriented=True).todense()
```

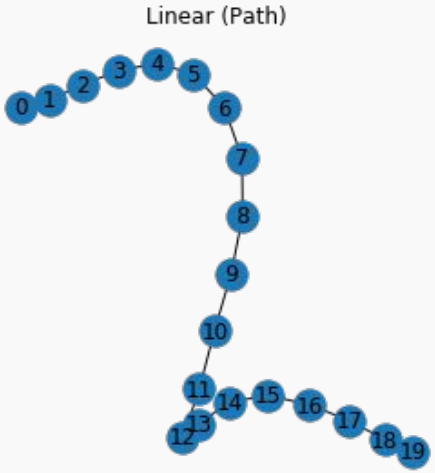
```
matrix([[-1., -1.,  0.,  0.,  0.],
        [ 1.,  0., -1.,  0.,  0.],
        [ 0.,  0.,  0., -1., -1.],
        [ 0.,  1.,  0.,  1.,  0.],
        [ 0.,  0.,  1.,  0.,  1.]])
```

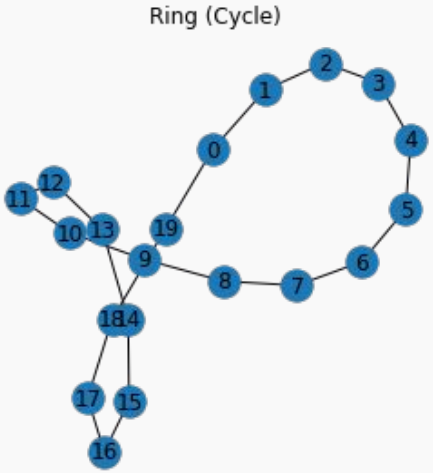
```
import sys
print(sys.getsizeof(nx.incidence_matrix(G, oriented=True)))
print(sys.getsizeof(nx.incidence_matrix(G, oriented=True).todense()))
```

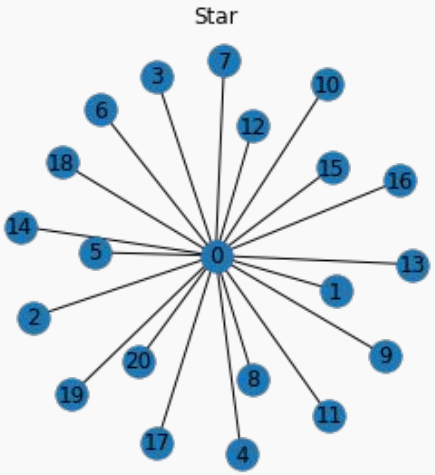
```
56
136
```

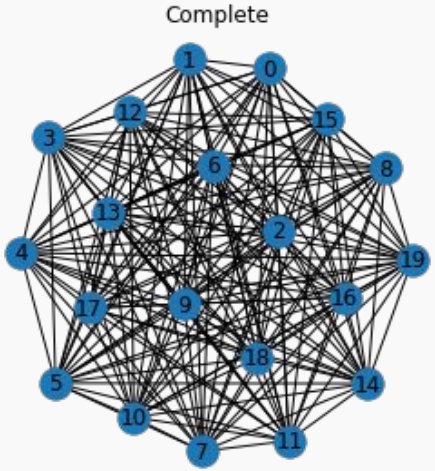
Generate Synthetic Networks

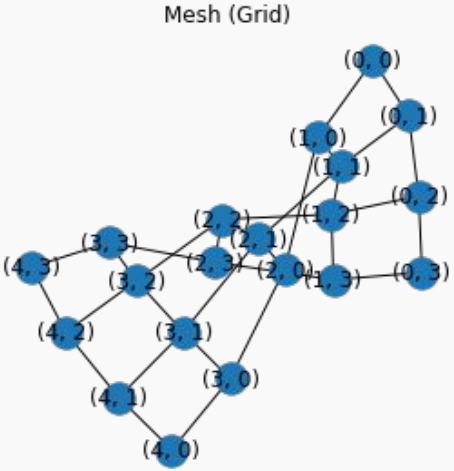
Classic Networks











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

fig, plot = plt.subplots(2, 3,figsize=(15,10))
subplots = plot.reshape(1, 6)[0]

# Generate and draw classic networks
G0 = nx.path_graph(20)
G1 = nx.cycle_graph(20)
G2 = nx.star_graph(20)
G3 = nx.complete_graph(20)
G4 = nx.balanced_tree(2, 5)
G5 = nx.grid_2d_graph(5, 4)

graphs = [G0,G1,G2,G3,G4,G5]

names = ["Linear (Path)", "Ring (Cycle)",
         "Star", "Complete", "Balanced Tree", "Mesh (Grid)"]

for graph,title,plot in zip(graphs,names,subplots):
  nx.draw_networkx(graph, ax=plot)
  plot.set_title(title)
  plot.axis("off")
```

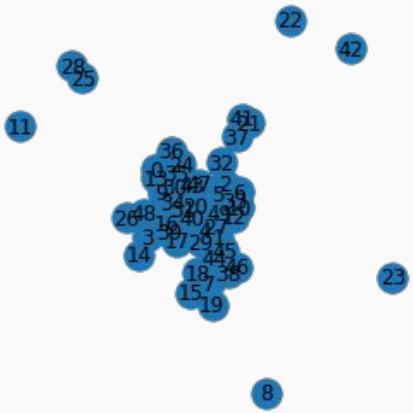
Random Graphs

- **Erdös–Rényi**
  a. "binomial graph", contains N nodes, but each edges occur with a probability P.
- **Watts–Strogatz**
  a. "Six degree of separation", create a illusion of small world.
- **Barabási–Albert**
  a. principle of preferential attachment, "celebrity" nodes
- **Holme–Kim**
  a. the same as above, plus the probability of adding a triangle for each added edge, making the synthetic network even more clustered and lifelike.

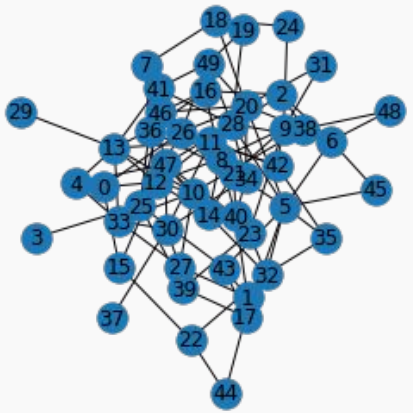
Erdos (n,p)

erdos(50,0)

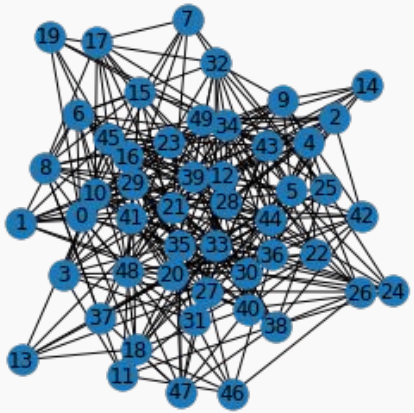
erdos(50,0.05)



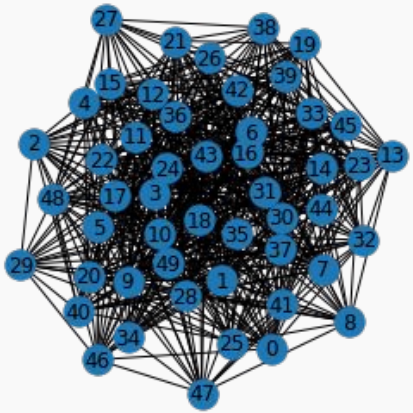
erdos(50,0.1)



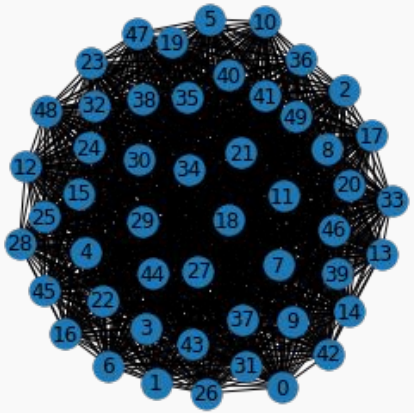
erdos(50,0.25)



erdos(50,0.5)



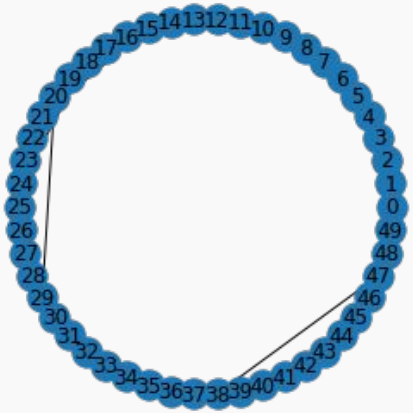
erdos(50,1)



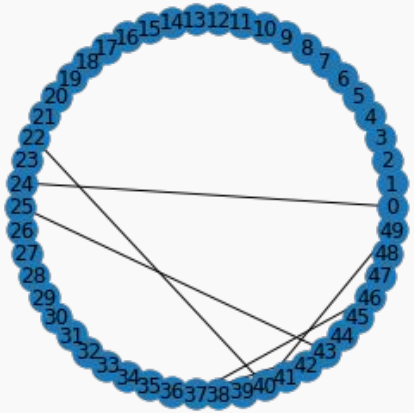
Watts Strogatz (n,k,p)

watts_strogatz(50,4,0)

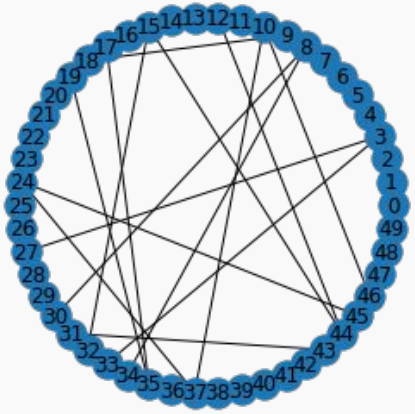
watts_strogatz(50,4,0.05)



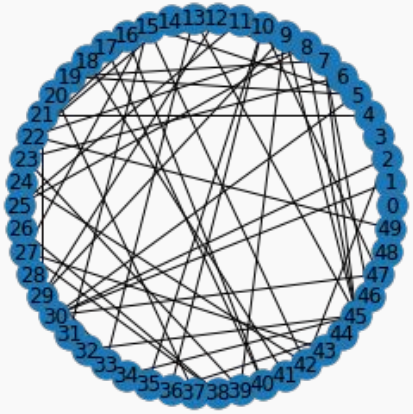
watts_strogatz(50,4,0.1)



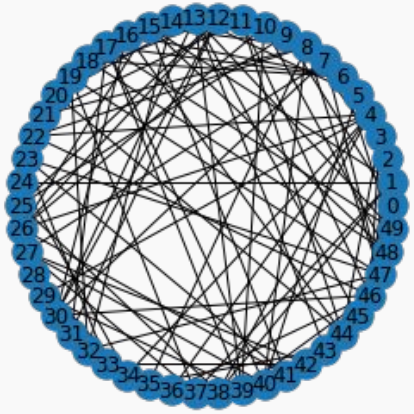
watts_strogatz(50,4,0.25)



watts_strogatz(50,4,0.5)

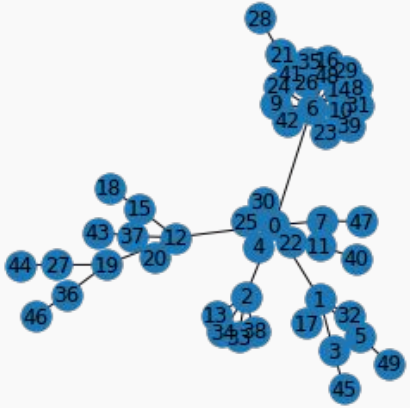


watts_strogatz(50,4,1)

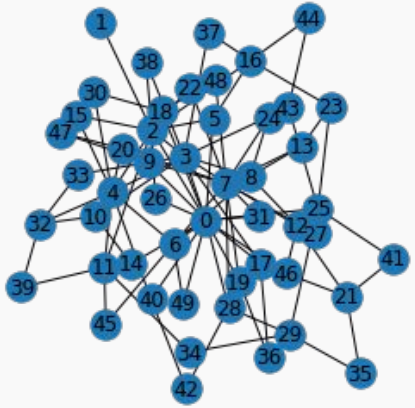


Barabasi Albert (n,k)

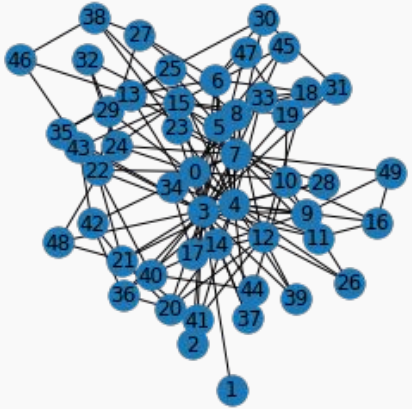
barabasi(50,1)



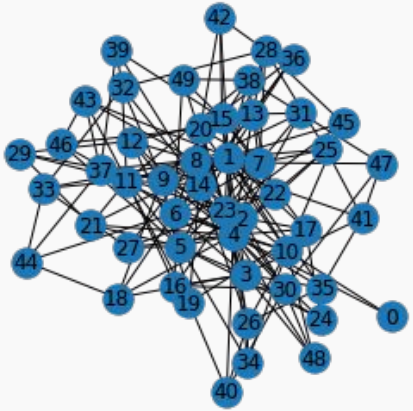
barabasi(50,2)



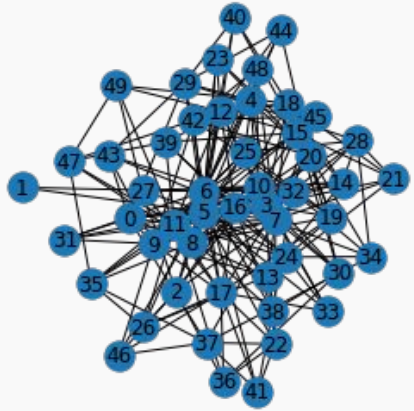
barabasi(50,3)



barabasi(50,4)



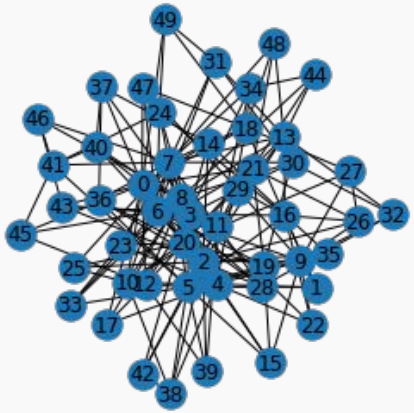
barabasi(50,5)



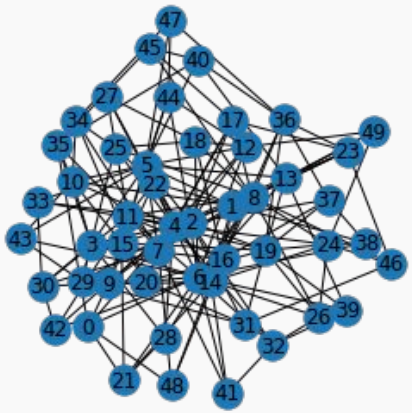
barabasi(50,10)

Holme Kim (n,k,p)

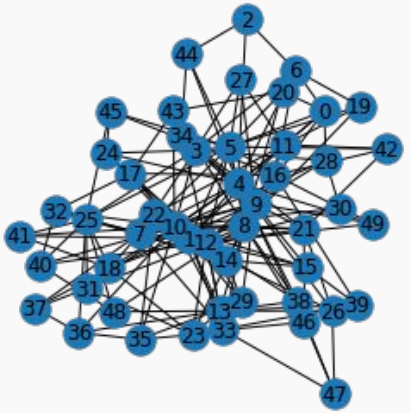
holme(50,4,0)



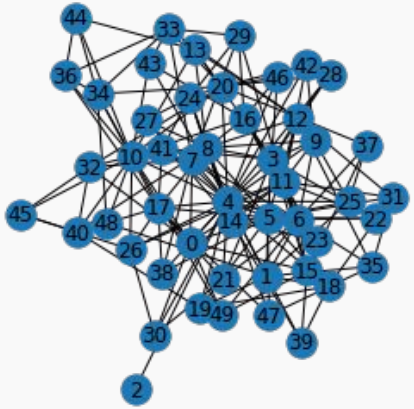
holme(50,4,0.05)



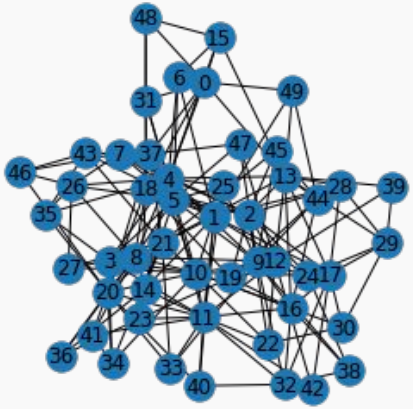
holme(50,4,0.1)



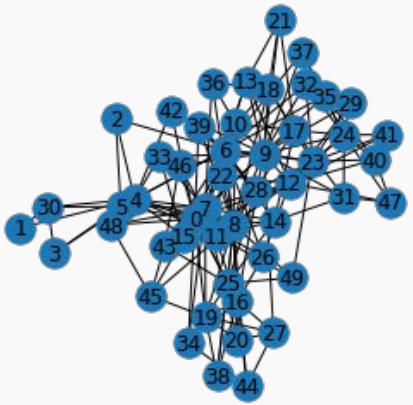
holme(50,4,0.25)



holme(50,4,0.5)



holme(50,4,1)



Code!!!