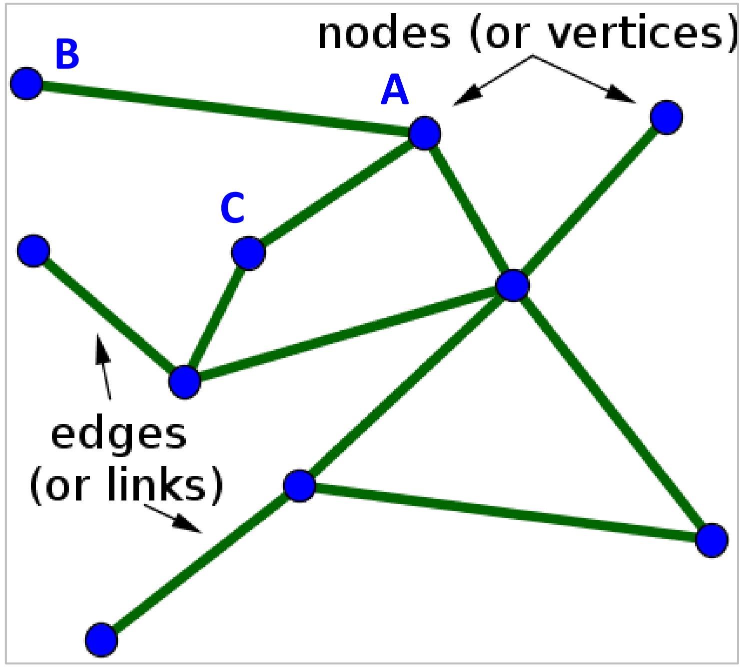
Network Definition

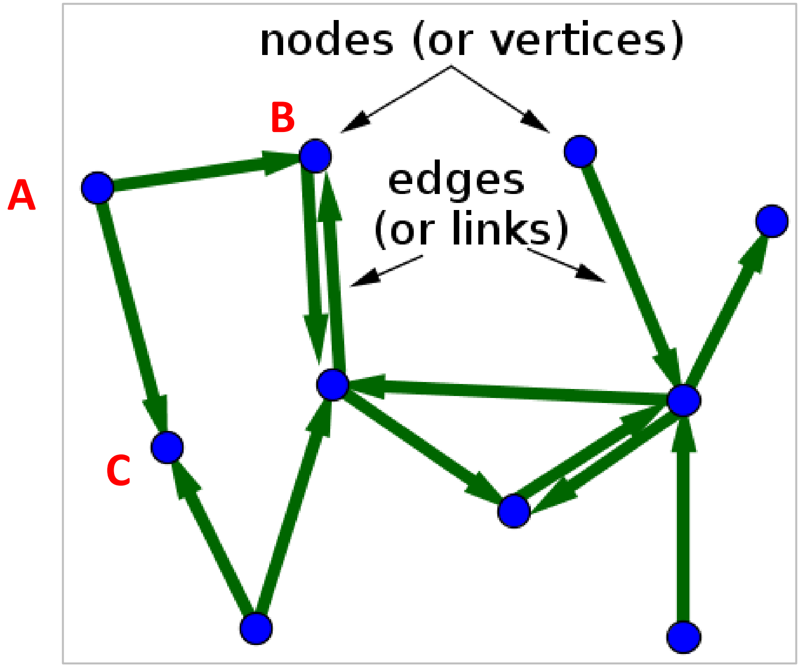
An undirected graph is defined as ***G*** = (***N***, ***E***), consisting of the set ***N*** of nodes and the set ***E*** of edges, which are **unordered pairs** of elements of ***N***. The formal definition of a directed graph is similar, the only difference is that the set ***E*** contains **ordered pairs** of elements of ***N***. In mathematics, networks are often referred to as **graphs**.

N **= { A, B, C, … }**
E **= { (A, B), (A, C), …}**



If all edges are bidirectional, or undirected, the network is an **undirected network** (or undirected graph).

N **= { A, B, C, … }**
E **= { < A, B >, < A, C >, …}**

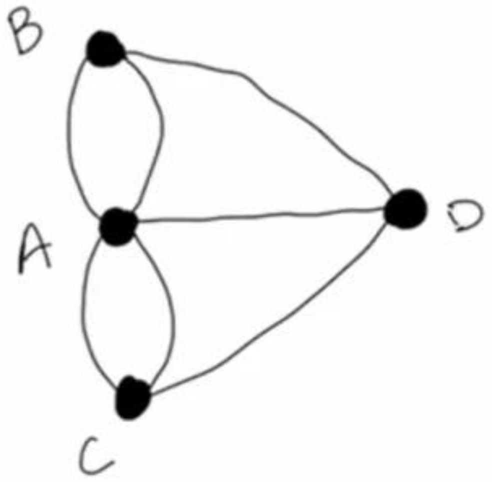


If the edges in a network are directed, i.e., pointing in only one direction, the network is called a **directed network**.

https://mathinsight.org/definition/network

Data Structure to Represent a Network

The Adjacency List

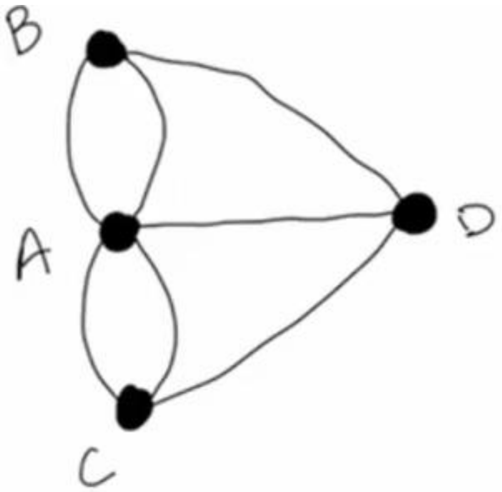


Hashed

Conceptual:

A: B, B, C, C, D
B: A, A, D
C: A, A, D
D: A, B, C

The Adjacency List

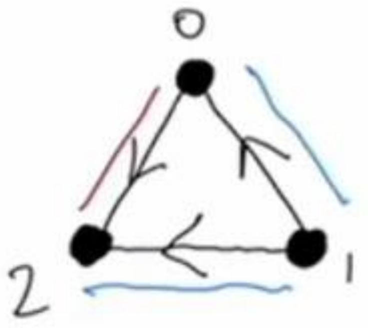


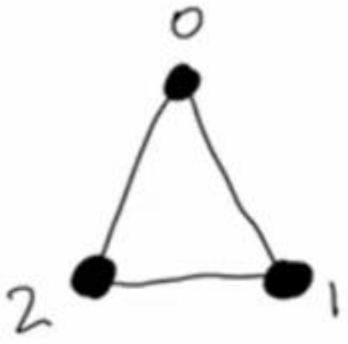
Programmed:

{ A: [B, B, C, C, D]
B: [A, A, D]
C: [A, A, D]
D: [A, B, C] }

https://www.youtube.com/watch?v=ukFNELi_U88

Directed vs. Undirected Graphs





Adjacency List

Directed:

0: 2
1: 0, 2
2:

Undirected:

0: 1, 2
1: 0, 2
2: 0, 1

Adjacency Matrix

Directed:

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

Undirected:

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

https://www.youtube.com/watch?v=ukFNELi_U88

NetworkX

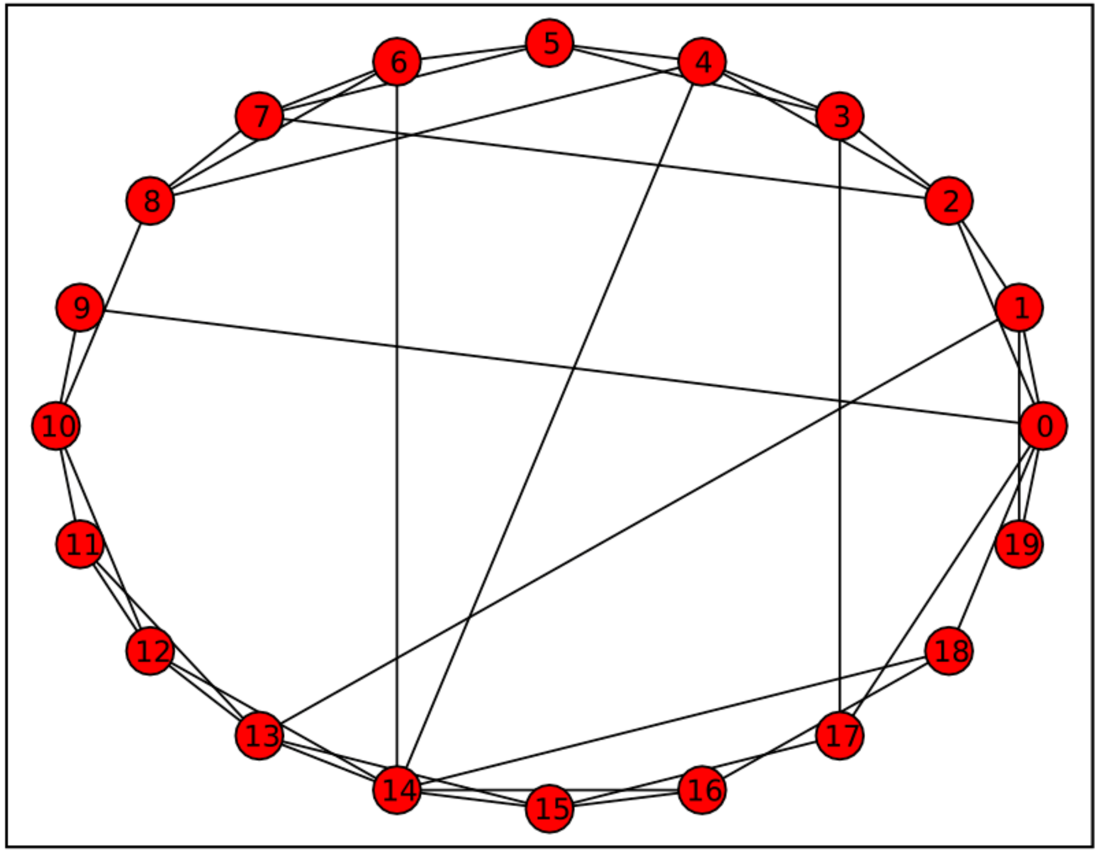
NetworkX is a Python library for studying graphs and networks. NetworkX is free software released under the BSD-new license.

Suitability

NetworkX is suitable for operation on large real-world graphs: e.g**.,** **graphs in excess of 10 million nodes and 100 million edges**.[clarification needed][4]

Due to its dependence on a pure-Python "dictionary of dictionary" data structure, NetworkX is a reasonably efficient, **very scalable, highly portable framework** for network and social network analysis.[5]

Watts-Strogatz model N=20, K=4, β=0.2



The model constructs an undirected graph with ***N*** nodes and **N*K / 2** edges with $0 \leq \beta \leq 1$ → **model a "small world"**.

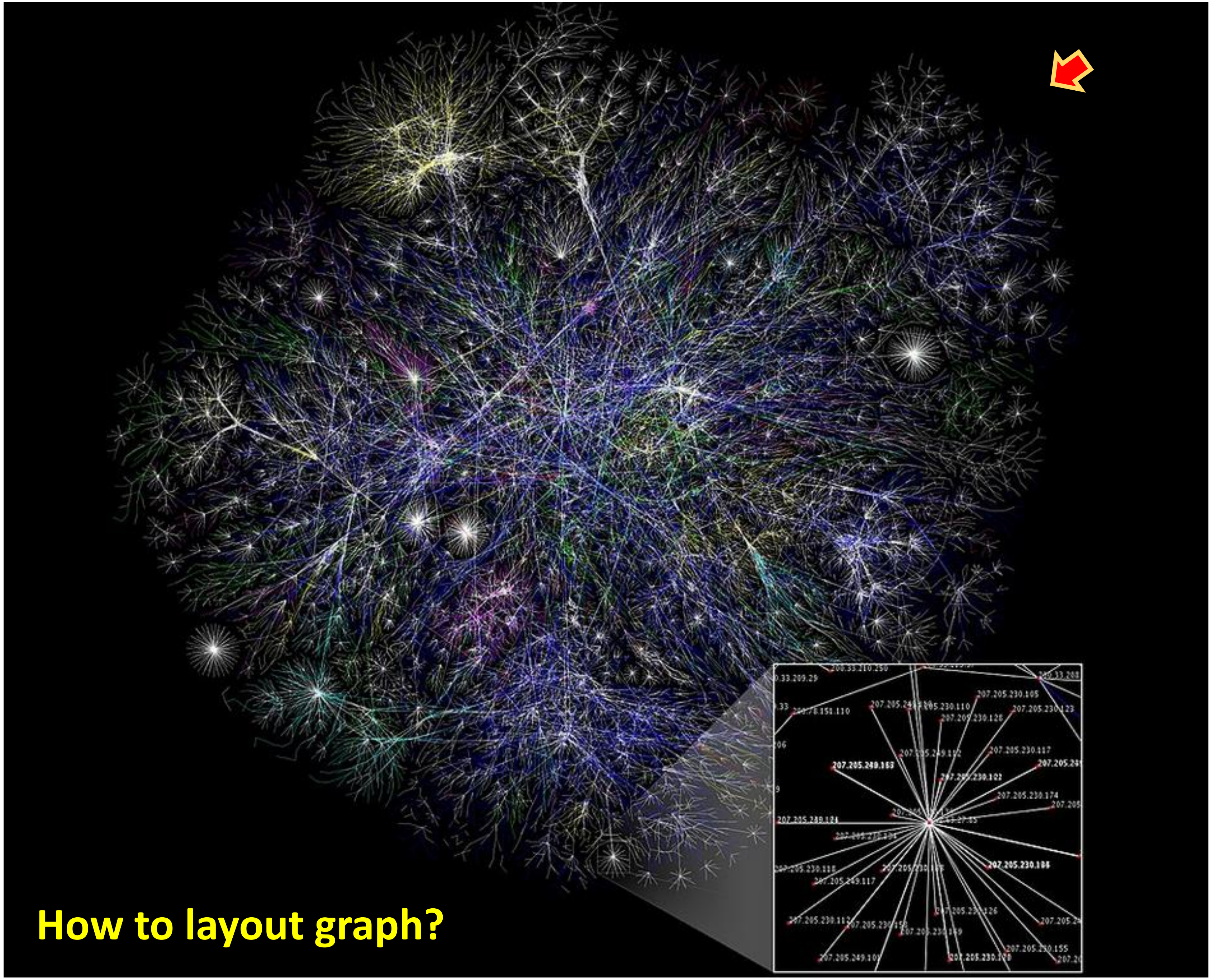
https://en.wikipedia.org/wiki/NetworkX

What do *N* and *E* represent?

Partial map of the Internet based on the January 15, 2005 data found on opte.org. **Each line is drawn between two nodes, representing two IP addresses.**

The length of the lines are indicative of the delay between those two nodes. This graph represents less than 30% of the Class C networks reachable by the data collection program in early 2005.

Lines are color-coded according to their corresponding RFC 1918 allocation as follows: Dark blue: net, ca, us Green: com, org Red: mil, gov, edu Yellow: jp, cn, tw, au, de Magenta: uk, it, pl, fr Gold: br, kr, nl White: unknown

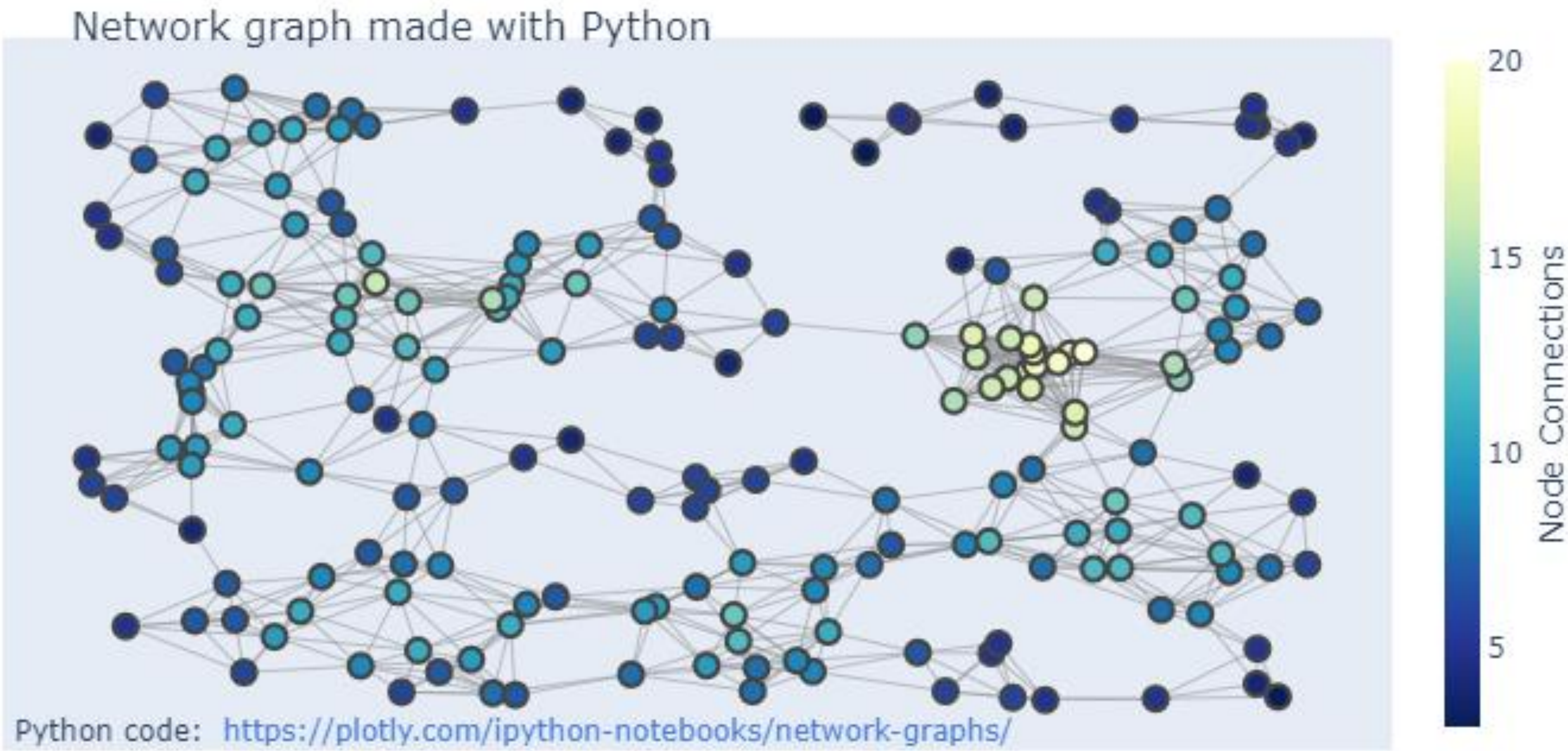


https://en.wikipedia.org/wiki/NetworkX

Create random graph with NetworkX

```
import plotly.graph_objects as go
import networkx as nx
G = nx.random_geometric_graph(200, 0.125)
```

NetworkX is a Python library for studying graphs and networks.
NetworkX is free software released under the BSD-new license.



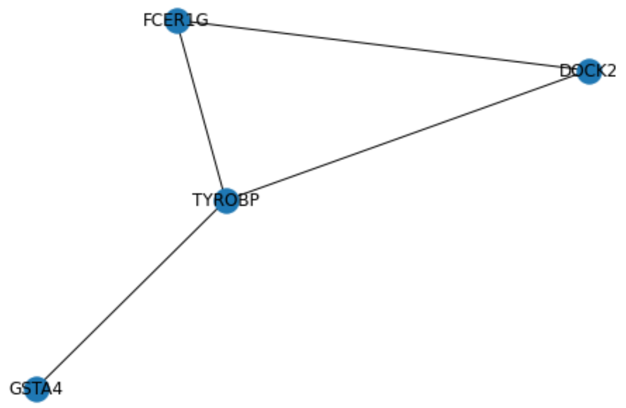
https://plotly.com/python/network-graphs/

NetworkX vs. pyvis –Static vs. Interactive

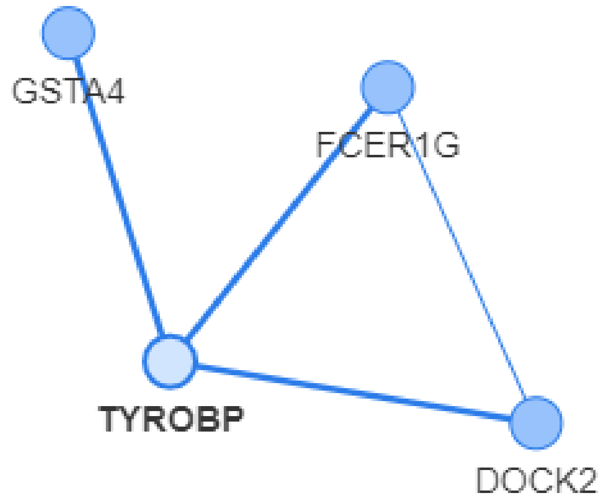
Consider the LOAD data set

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

G = nx.Graph()
G.add_edge("TYROBP", "DOCK2")
G.add_edge("TYROBP", "FCER1G")
G.add_edge("TYROBP", "GSTA4")
G.add_edge("DOCK2", "FCER1G")
nx.draw(G, with_labels=True)
plt.show()
```

```
# Intall pyvis using pip install pyvis
import pyvis
from pyvis.network import Network

# create vis network
net = Network(notebook=True, width=1000, height=600)
# load the networkx graph
net.from_nx(G)
# show – example.html is just a given name for the diagram
net.show("example.html")
```



Static image

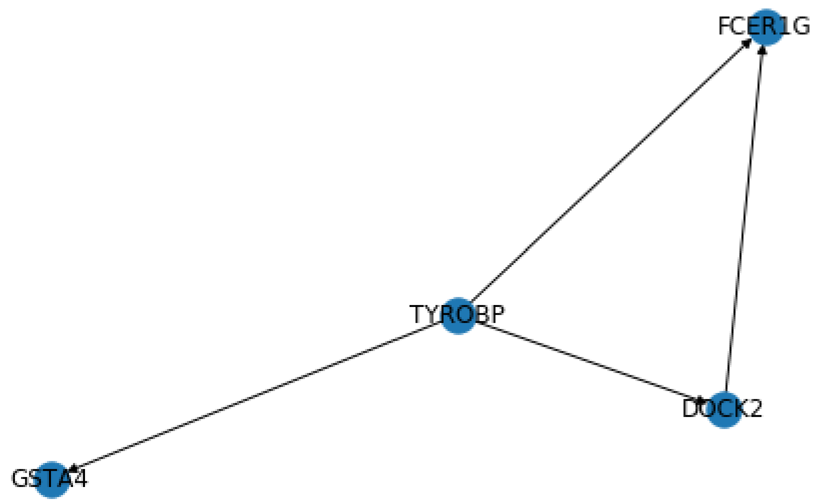


Interactive network

Example – Directed vs Undirected

Consider the LOAD data set

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

G = nx.DiGraph()
G.add_edge("TYROBP", "DOCK2")
G.add_edge("TYROBP", "FCER1G")
G.add_edge("TYROBP", "GSTA4")
G.add_edge("DOCK2", "FCER1G")
nx.draw(G, with_labels=True)
plt.show()
```



Directed Edges; Static image

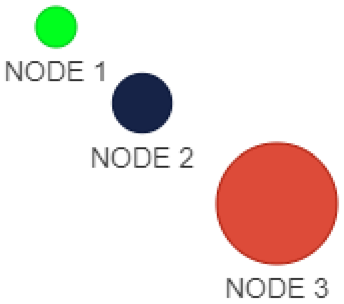
```
# Intall pyvis using pip install pyvis
import pyvis
from pyvis.network import Network

# create vis network
net = Network(notebook=True, width=1000, height=600)
# load the networkx graph
net.from_nx(G)
# show – example.html is just a given name for the diagram
net.show("example.html")
```

Interactive network

Example – Node size, coloring, labeling and edge direction

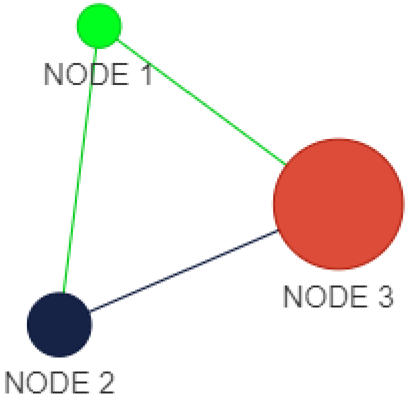
Directly construct the network using pyvis.

```
import pyvis
from pyvis.network import Network

g = Network(notebook=True)
g.add_nodes([1,2,3], value=[10, 100, 400],
            title=['I am node 1', 'node 2 here', 'and im node 3'],
            x=[11.2, 54.2, 121.4],
            y=[12.1, 50.54, 100.2],
            label=['NODE 1', 'NODE 2', 'NODE 3'],
            color=['#00ff1e', '#162347', '#dd4b39'])
g.show("example.html")
```

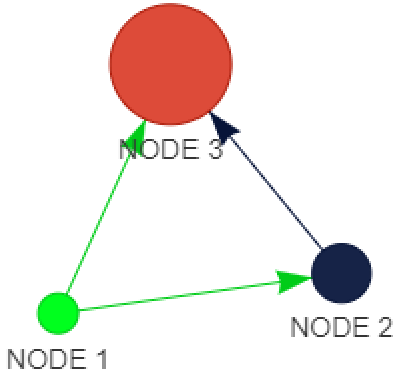



```
g.add_edge(1, 2, weight=.30)
g.add_edge(2, 3, weight=.87)
g.add_edge(1, 3, weight=.90)
g.show("example.html")
```

Are x,y values ignored when edges are added?




```
g = Network(directed=True, notebook=True)
```

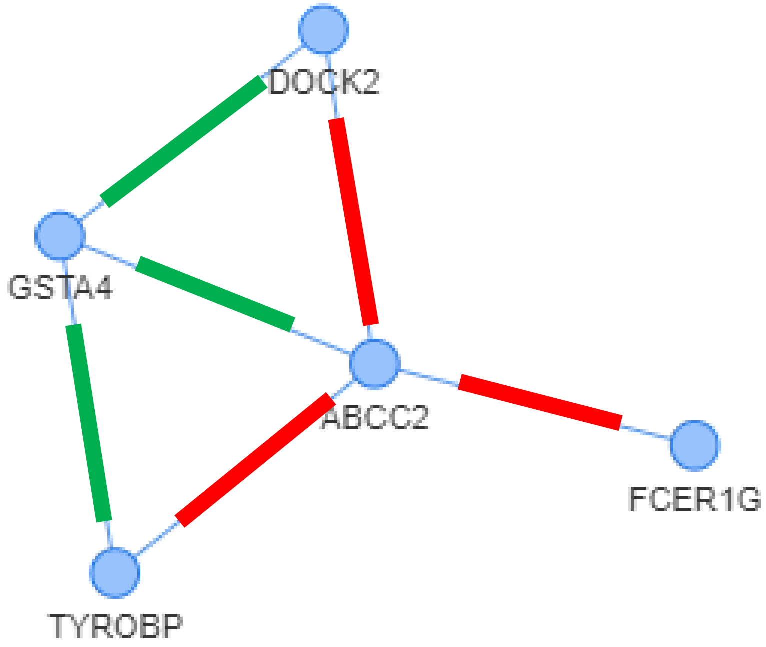


https://pyvis.readthedocs.io/en/latest/tutorial.html

Example:
Can you construct a gene “correlation” network from the LOAD data?

Consider the LOAD data set with r values.

	node1	node2	r_val
2	TYROBP	GSTA4	-0.700
3	TYROBP	ABCC2	0.028
5	DOCK2	GSTA4	-0.647
6	DOCK2	ABCC2	0.071
8	FCER1G	ABCC2	0.206
9	GSTA4	ABCC2	-0.085



Edges are colored based on r values.

r_val > 0

r_val < 0

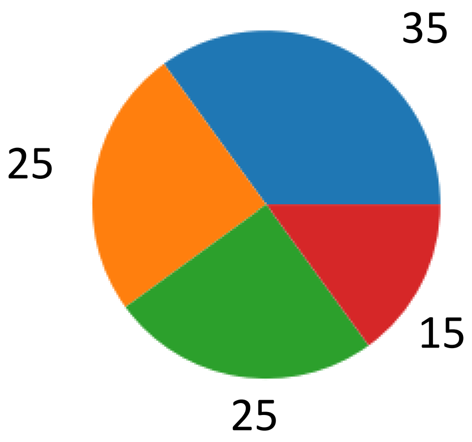
What if we want to create such a network from only highly correlate pairs of genes?

Can edge thickness reflect r strength?

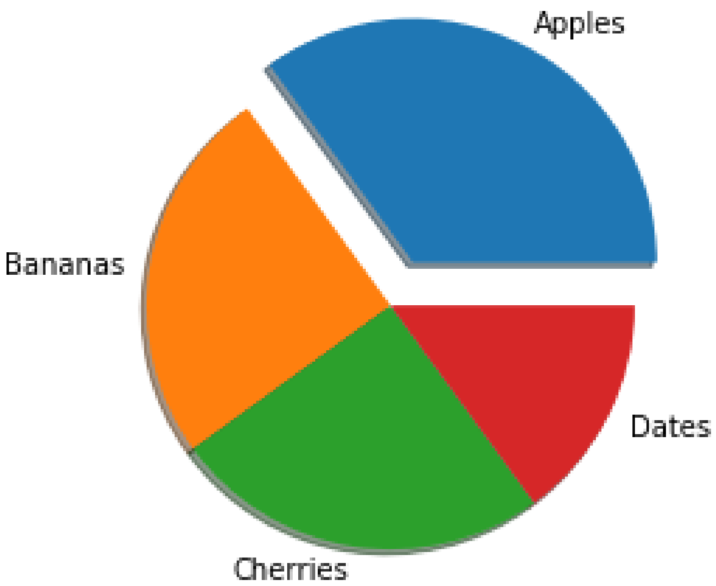
Matplotlib Pie Charts -- Creating Pie Charts

With Pyplot, you can use the pie() function to draw pie charts:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```

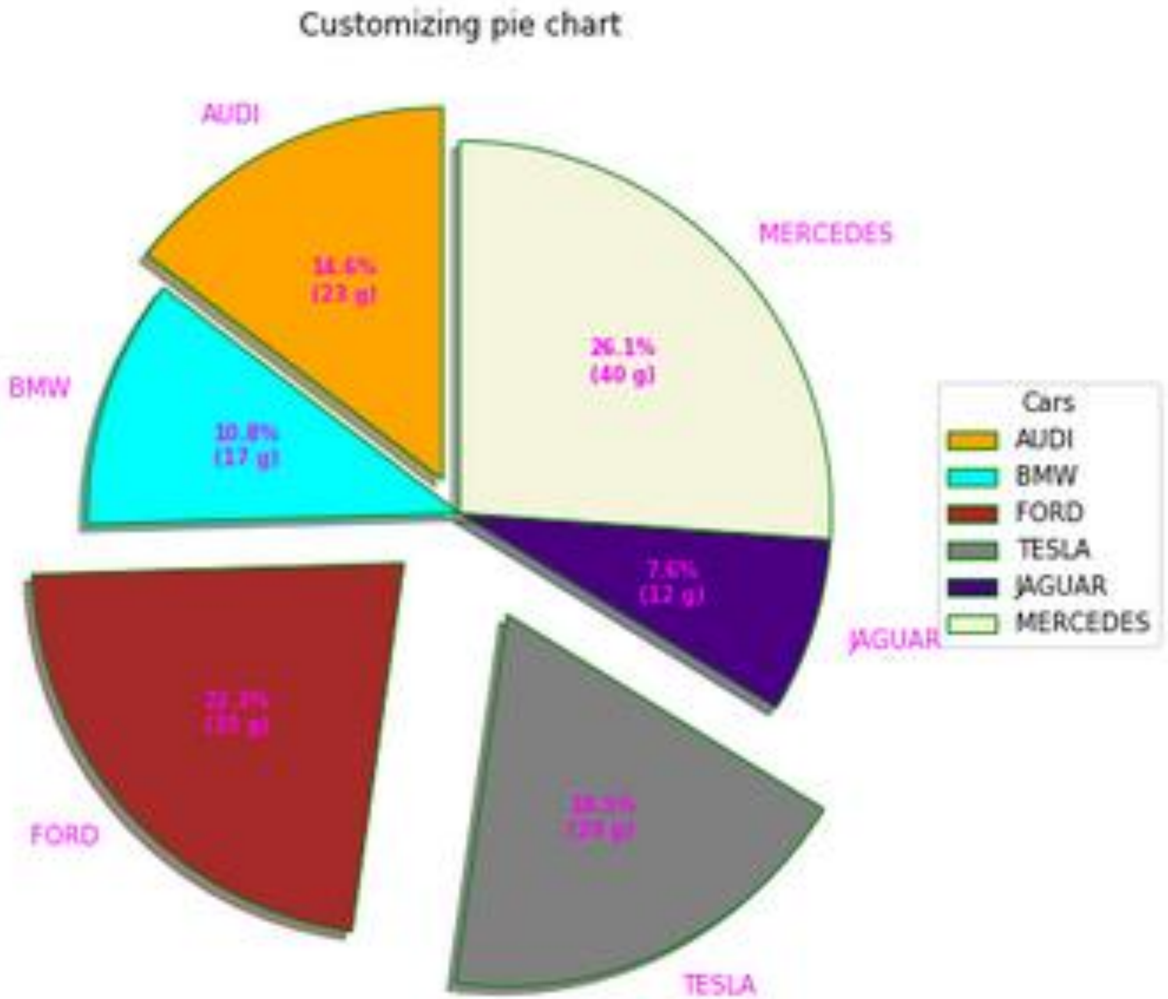


https://www.w3schools.com/python/matplotlib_pie_charts.asp

Customizing Pie Chart

startangle   wedgeprop   frame=True   chart.autopct

Example 1:

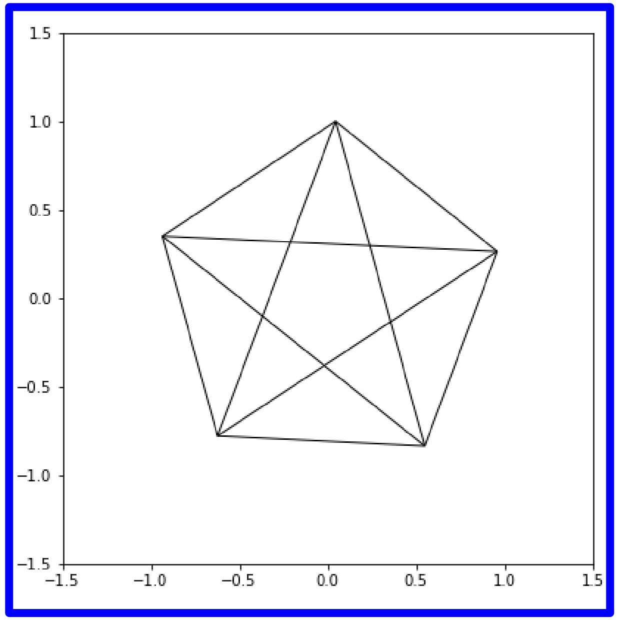


Example 2:

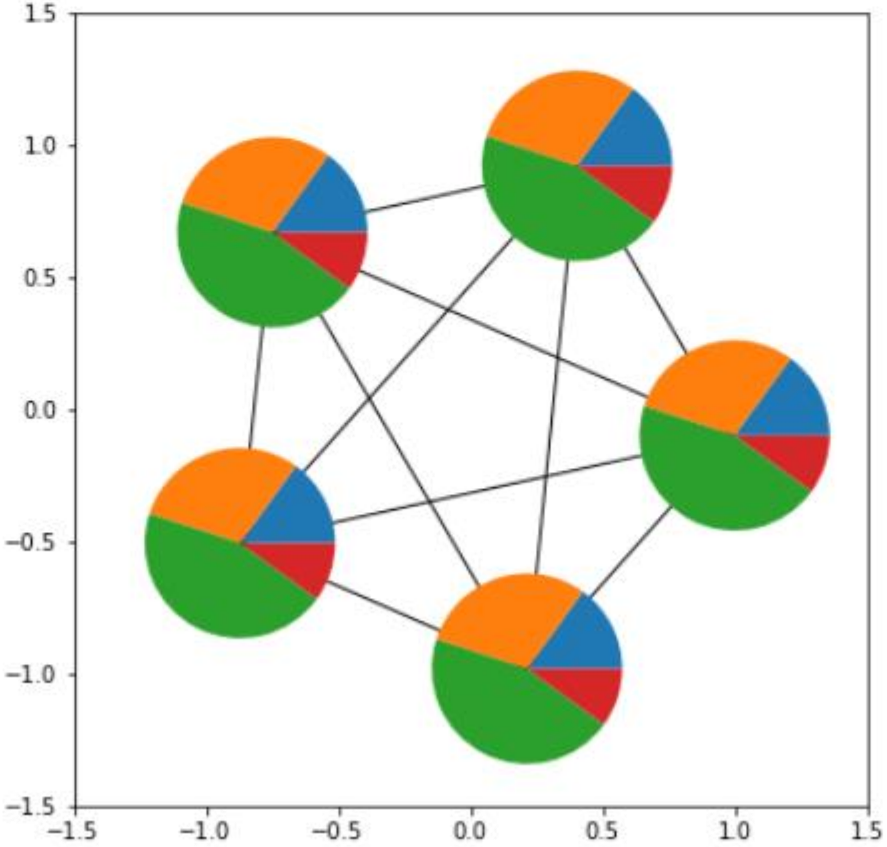
https://www.geeksforgeeks.org/plot-a-pie-chart-in-python-using-matplotlib/

Network of Pie Charts

```
# Make sure: pip install decorator==4.3
import networkx as nx
import matplotlib.pyplot as plt
G=nx.complete_graph(5)
pos=nx.spring_layout(G)

fig=plt.figure(figsize=(5,5))
ax=plt.axes([0,0,1,1])
ax.set_aspect('equal')
nx.draw_networkx_edges(G,pos,ax=ax)

plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)

trans=ax.transData.transform
trans2=fig.transFigure.inverted().transform

piesize=0.3
p2=piesize/2.0
for n in G:
    xx,yy=trans(pos[n]) # figure coordinates
    xa,ya=trans2((xx,yy)) # axes coordinates
    a = plt.axes([xa-p2,ya-p2, piesize, piesize])
    a.set_aspect('equal')
    fracs = [15,30,45, 10]
    a.pie(fracs)

plt.savefig('pc.png')
```

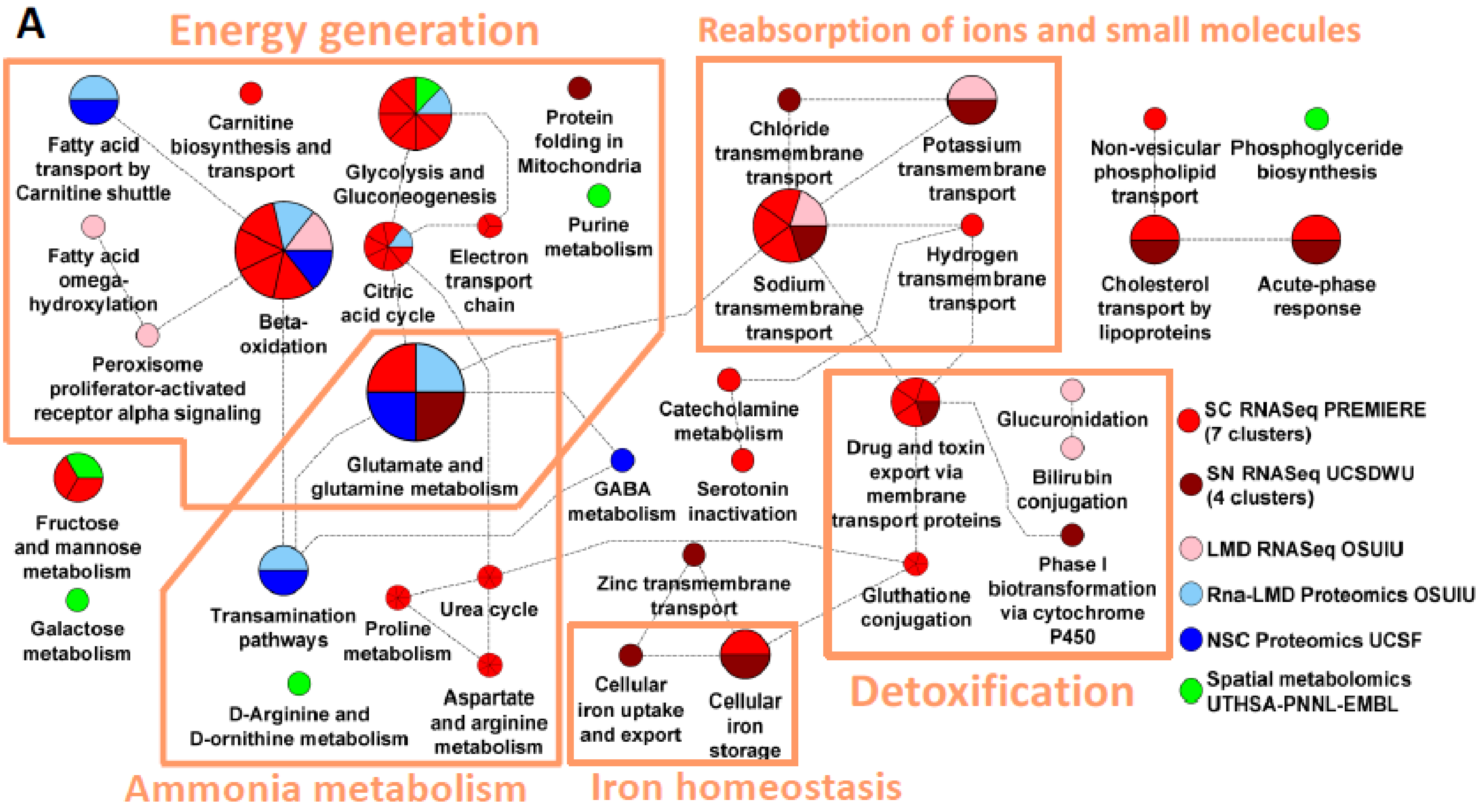


Superimposing pie charts over the network nodes.



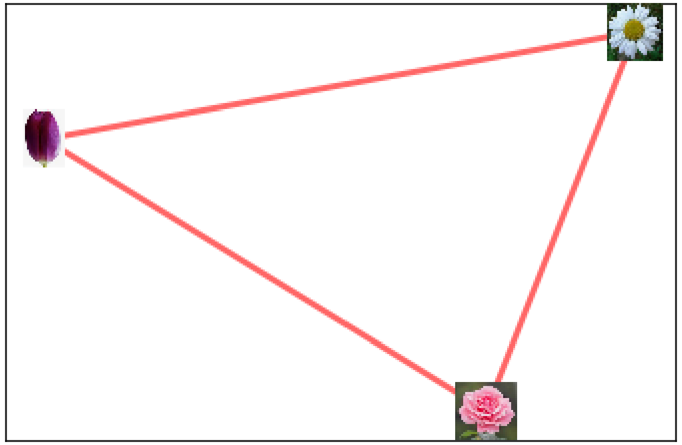
https://stackoverflow.com/questions/26714730/pie-chart-as-nodes-in-networkx

An Interesting Pie Chart Network Visualization

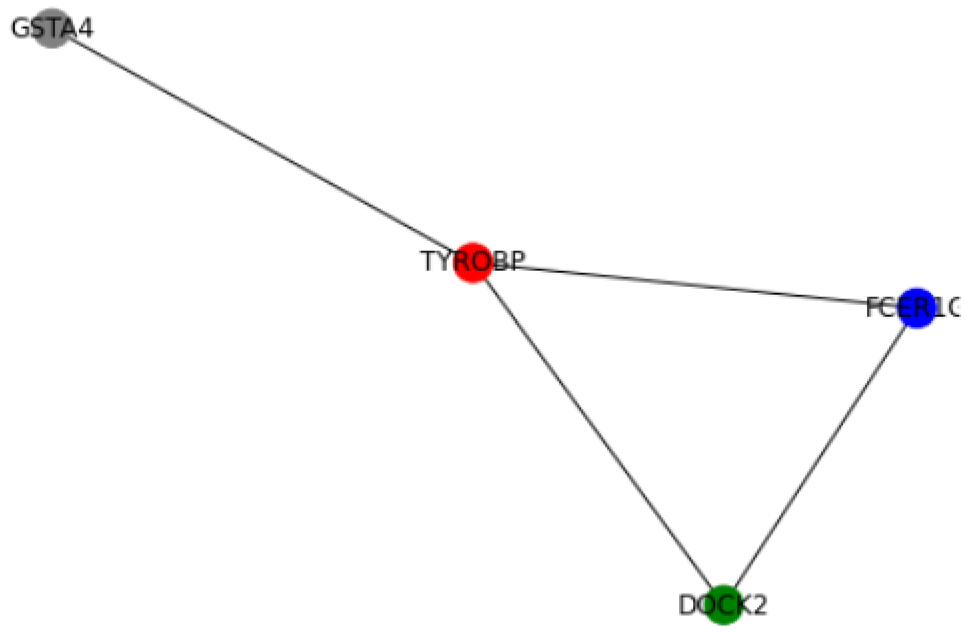


bioRxiv preprint doi: https://doi.org/10.1101/2020.07.23.216507

Example:
Can nodes of a network be images? Can they be a subnetwork?



```
nx.draw(G, node_color=["red", "green", "blue", "gray"], with_labels=True)
```



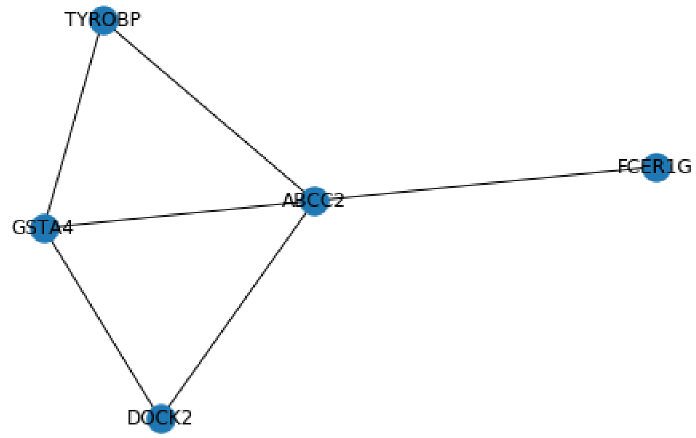
NetworkX

Two different Look & Feel

```
G_n = nx.from_pandas_edgelist(df_n,
                              source='node1',
                              target='node2',
                              edge_attr='r_val')
print("No of unique characters:", len(G_n.nodes))
print("No of connections:", len(G_n.edges))
nx.draw(G_n, with_labels=True)
plt.show()
```

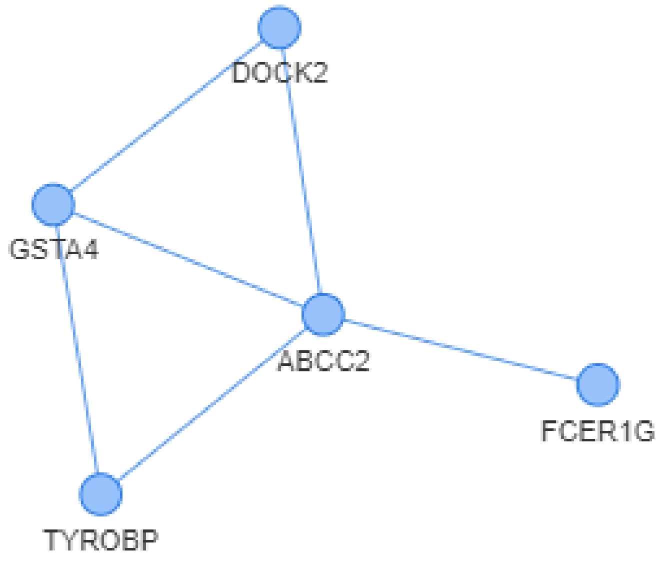
```
# Intall pyvis using pip install pyvis
# import pyvis
from pyvis.network import Network
# create vis network
net = Network(notebook=True, width=1000, height=600)
# load the networkx graph
net.from_nx(G_n)
# show – example.html is just a given name for the diagram
net.show("example.html")
```

```
No of unique characters: 5
No of connections: 6
```



df_n = df.loc[df['r_val']< 0.5, :]

	node1	node2	r_val
2	TYROBP	GSTA4	-0.700
3	TYROBP	ABCC2	0.028
5	DOCK2	GSTA4	-0.647
6	DOCK2	ABCC2	0.071
8	FCER1G	ABCC2	0.206
9	GSTA4	ABCC2	-0.085



https://en.wikipedia.org/wiki/NetworkX

Network Visualization

Visualizing Networks in Python

Code and Data Available

https://github.com/imohitmayank/got_network_viz_python

```
# Install network using pip at CMD if it is not installed.
pip install networkx
```

```
# Import
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# load data
df = pd.read_csv("F:/Data/data/book1.csv")
# pick only important weights (hard threshold)
df = df.loc[df['weight']>10, :]
df
```

```
# load pandas df as networkx graph
G = nx.from_pandas_edgelist(df
```

Option 2: PyVis

PyVis is an interactive network visualization python package which takes the NetworkX graph as input. It also provides multiple styling options to customize the nodes, edges and even the complete layout. And the best part, it can be done on-the-go using a setting pane where you can play with the various options and export the final settings in form of a python dictionary. This dictionary can later be passed as config while calling the function, resulting in as-it-was drawing of the network. Apart from this, in terms of visualization, you have the basic option of zooming, selecting, hover, among others. Cool isn’t it! 😉

Use STRING DB data for the network drawing.

```
df = pd.read_csv("F:/Data/string_interactions.csv")
```

Tutorial

This guide can help you start working with NetworkX.

Creating a graph

```
import networkx as nx
G = nx.Graph()
```

By definition, a Graph is a collection of nodes (vertices) along with identified pairs of nodes (called edges, links, etc).

Nodes can be any hashable object e.g., a text string, an image, an XML object, another Graph, a customized node object, etc.

https://networkx.org/documentation/stable/tutorial.html

Matplotlib Pie Charts
Creating Pie Charts

With Pyplot, you can use the pie() function to draw pie charts:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.Graph()
G.add_edge("TYROBP", "DOCK2")
G.add_edge("TYROBP", "FCER1G")
G.add_edge("TYROBP", "GSTA4")
G.add_edge("DOCK2", "FCER1G")
nx.draw(G, with_labels=True)
plt.show()
```

https://www.w3schools.com/python/matplotlib_pie_charts.asp

weighted edges

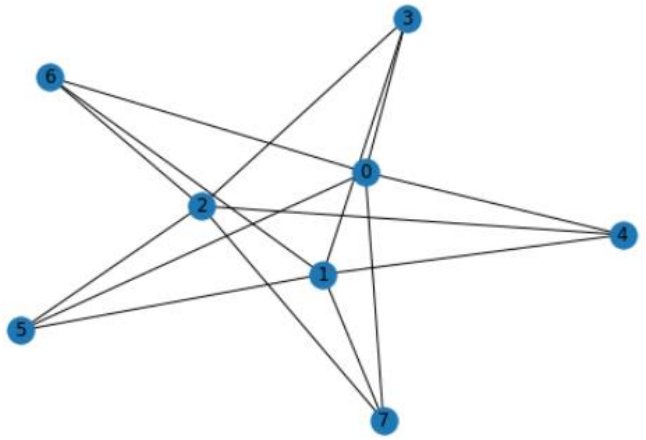
```
G.add_edge(1, 2, weight=4.7 )
G.add_edges_from([(3, 4), (4, 5)],
color='red')
G.add_edges_from([(1, 2, {'color':
'blue'}), (2, 3, {'weight': 8})])
G[1][2]['weight'] = 4.7
G.edges[3, 4]['weight'] = 4.2
G.edges.data()
```

The special attribute `weight` should be numeric as it is used by algorithms requiring weighted edges.

Multigraphs

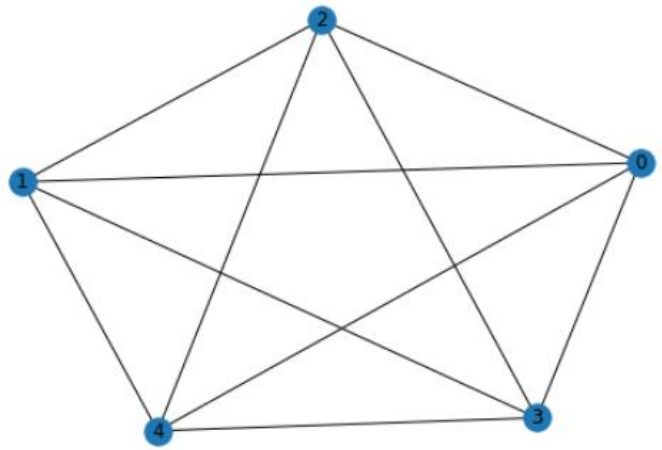
```
MG = nx.MultiGraph()

MG.add_weighted_edges_from([(1, 2, 0.5), (1, 2, 0.75), (2, 3, 0.5)])

dict(MG.degree(weight='weight'))
{1: 1.25, 2: 1.75, 3: 0.5}

GG = nx.Graph()

for n, nbrs in MG.adjacency():

   for nbr, edict in nbrs.items():

       minvalue = min([d['weight'] for d in edict.values()])

       GG.add_edge(n, nbr, weight = minvalue)


nx.shortest_path(GG, 1, 3)
```

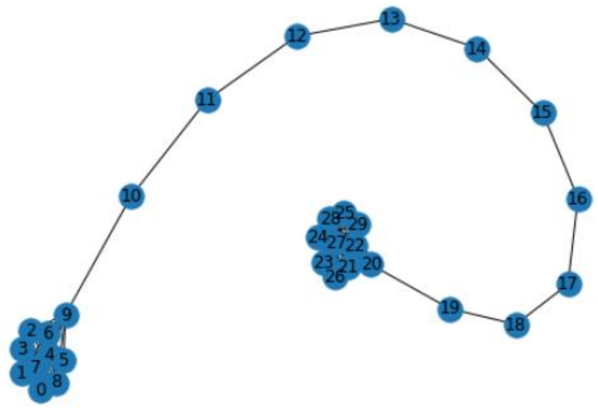
Using a (constructive) generator for a classic graph

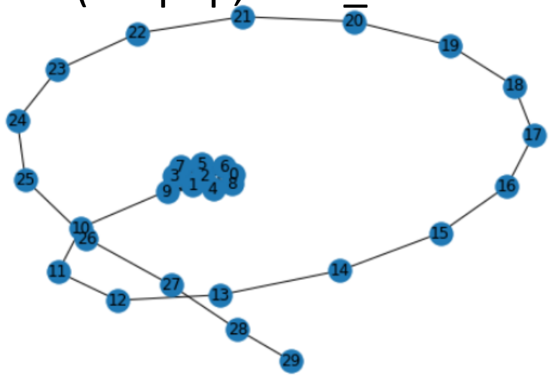



```
K_5 = nx.complete_graph(5)

K_3_5 = nx.complete_bipartite_graph(3, 5)

barbell = nx.barbell_graph(10, 10)

lollipop = nx.lollipop_graph(10, 20)
```

nx.draw(lollipop, with_labels=True)

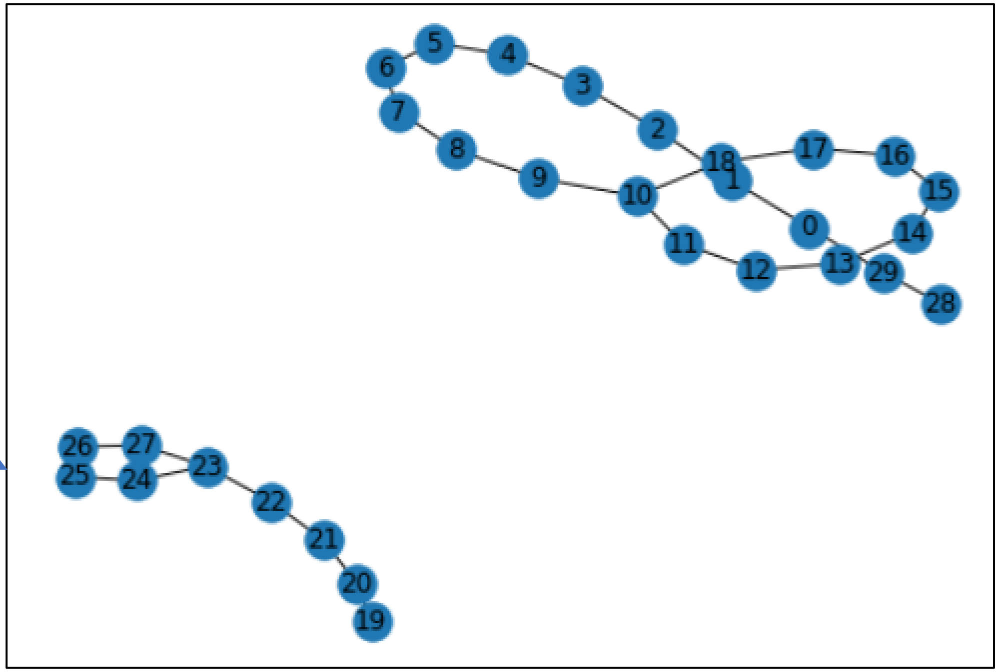


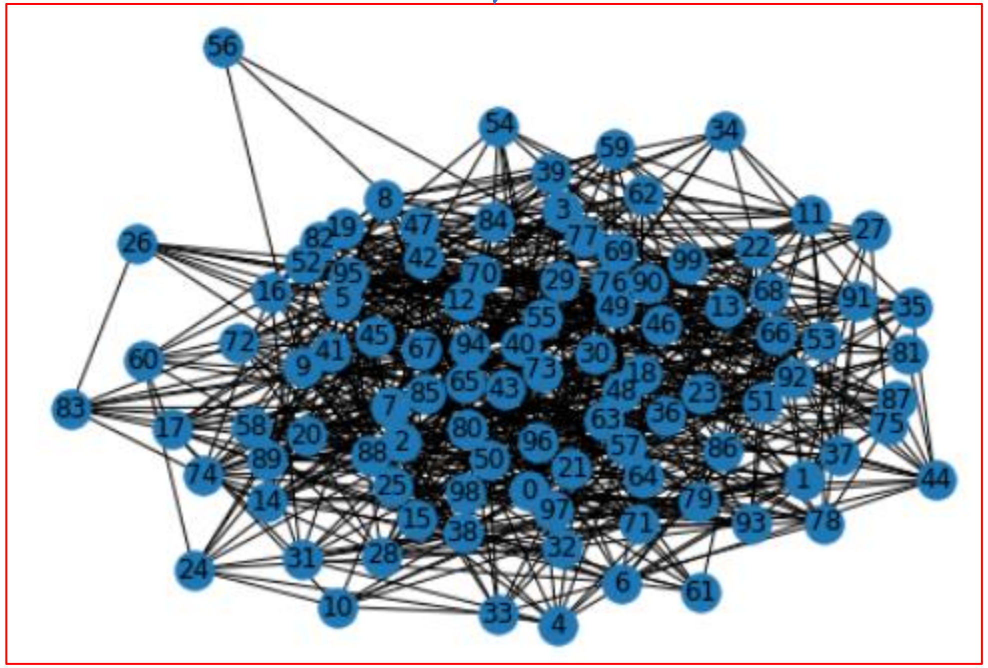


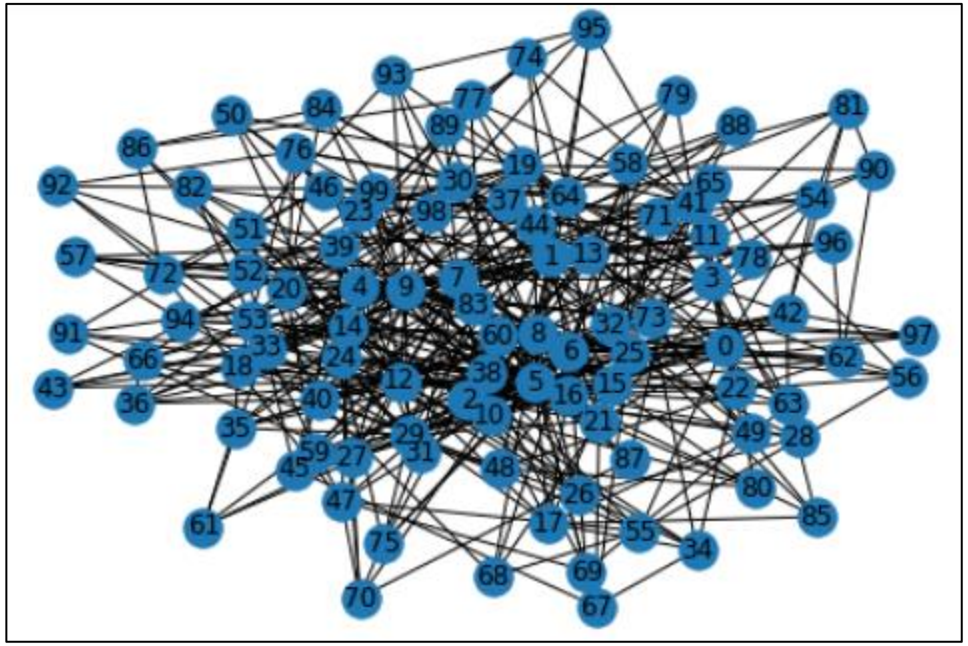


Using a stochastic graph generator, e.g,

```
er = nx.erdos_renyi_graph(100, 0.15)

ws = nx.watts_strogatz_graph(30, 3, 0.1)

ba = nx.barabasi_albert_graph(100, 5)

red = nx.random_lobster(100, 0.9, 0.9)
```

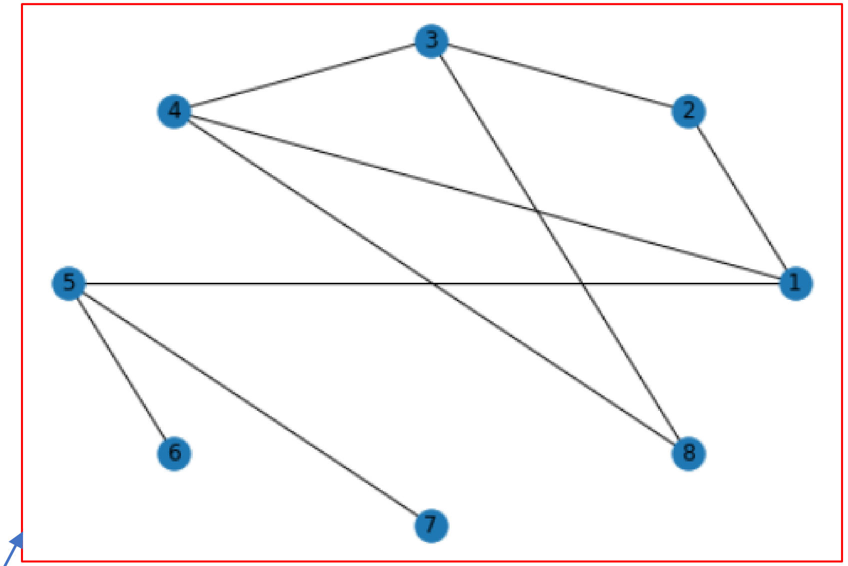


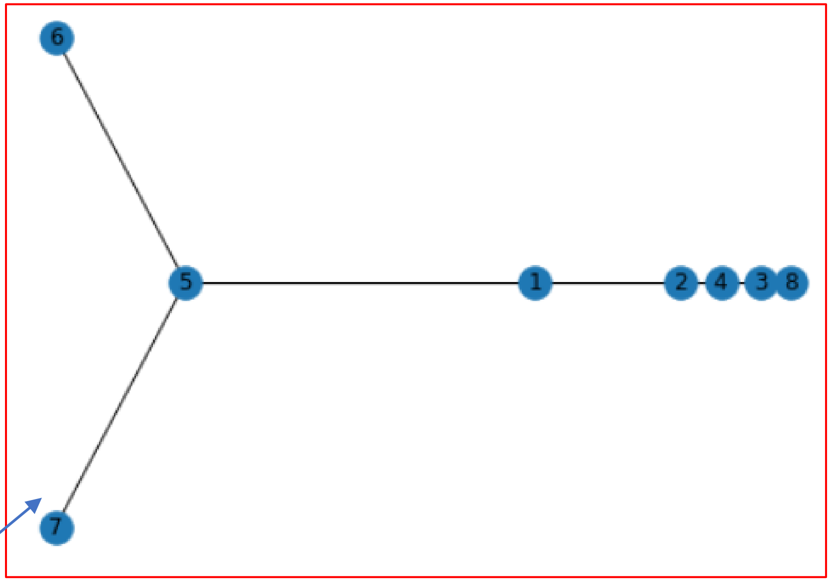




Python | Visualize graphs generated in NetworkX using Matplotlib

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

g = nx.Graph()

g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 4)
g.add_edge(1, 4)
g.add_edge(1, 5)
g.add_edge(5, 6)
g.add_edge(5, 7)
g.add_edge(4, 8)
g.add_edge(3, 8)

# drawing in circular layout
nx.draw_circular(g, with_labels = True)
plt.savefig("filename1.png")
```



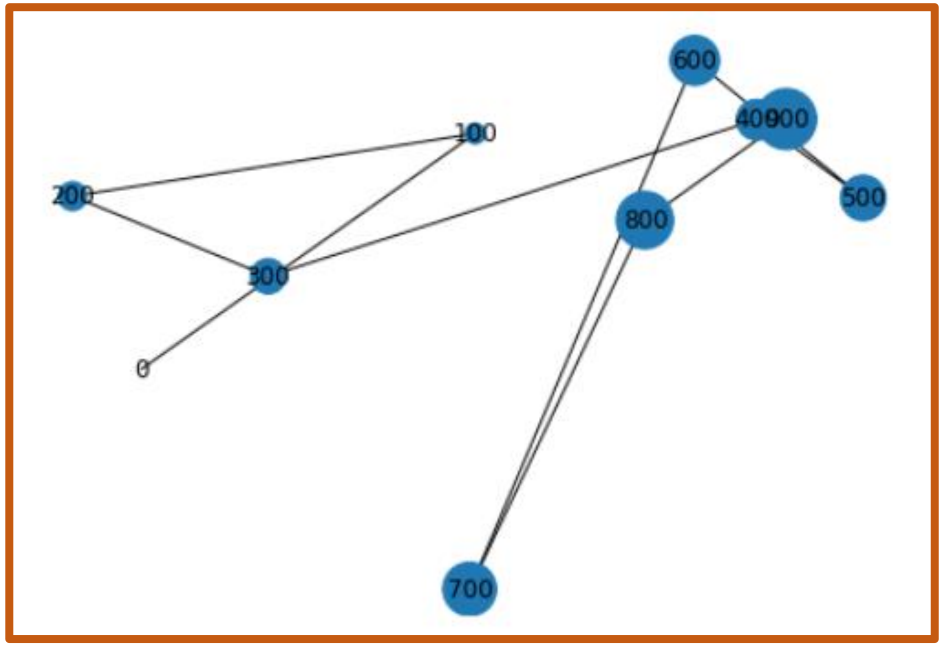


nx.draw_spectral(g, with_labels = True)

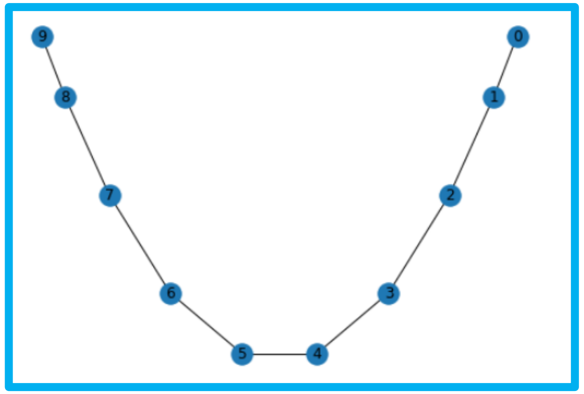
https://www.geeksforgeeks.org/python-visualize-graphs-generated-in-networkx-using-matplotlib/

Changing node size: node_size

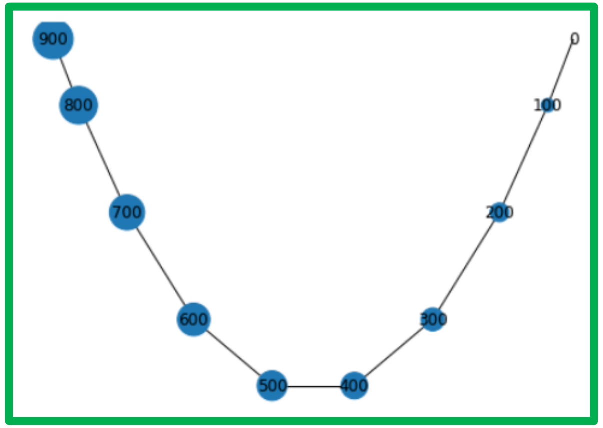
```
import sys, networkx as nx, matplotlib.pyplot as plt

# Create a list of 10 nodes numbered [0, 9]
nodes = range(10)
node_sizes = []
labels = {}
for n in nodes:
      node_sizes.append( 100 * n )
      labels[n] = 100 * n
# Node sizes: [0, 100, 200, 300, 400, 500, 600, 700, 800, 900]

# Connect each node to its successor
edges = [ (i, i+1) for i in range(len(nodes)-1) ]

# Create the graph and draw it with the node labels
g = nx.Graph()
g.add_nodes_from(nodes)
g.add_edges_from(edges)

nx.draw_random(g, node_size = node_sizes, labels=labels, with_labels=True)
plt.show()
```




```
nx.draw_spectral(g, with_labels = True)
```




```
nx.draw_spectral(g, node_size = node_sizes, labels=labels)
```



https://stackoverflow.com/questions/24636015/networkx-change-node-size-based-on-list-or-dictionary-value

Visualizing Graphs in Python With pyvis | Graph Theory With Python #3 by David Amos (45:41)

In this video, you'll learn how to visualize graphs in Python using the pyvis package. You'll also learn about four families of graphs — paths, cycles, complete graphs, and stars — and how to write Python functions that generate these graphs for you. This is an important step for providing tooling to quickly create examples of graphs that will help us as we investigate Graph Theory with Python in future videos.

https://www.youtube.com/watch?v=7MQ19mADAV8