

NetworkX graph and Network Visualisation :

Introduction:

NetworkX is a Python Library which is used for working on the Graphs. It helps to create the graphs and also helps to analyze of it. NetworkX explains only a very basic way using syntax like `nx.draw()` to show the graphs using `matplotlib`. NetworkX is simple but not very efficient. These are the key features of NetworkX :

- 1.Graph Creation: Networkx helps us to built the graph by using the given data
- 2.Graph Representation: It helps us to to understand the way of connections in the given data.
- 3.Graph Algorithms: It helps us to conduct the tasks on the graph and helps to solve the problems
- 4.Graph Visualization:It helps us to Visualisation and helps to analyze the data

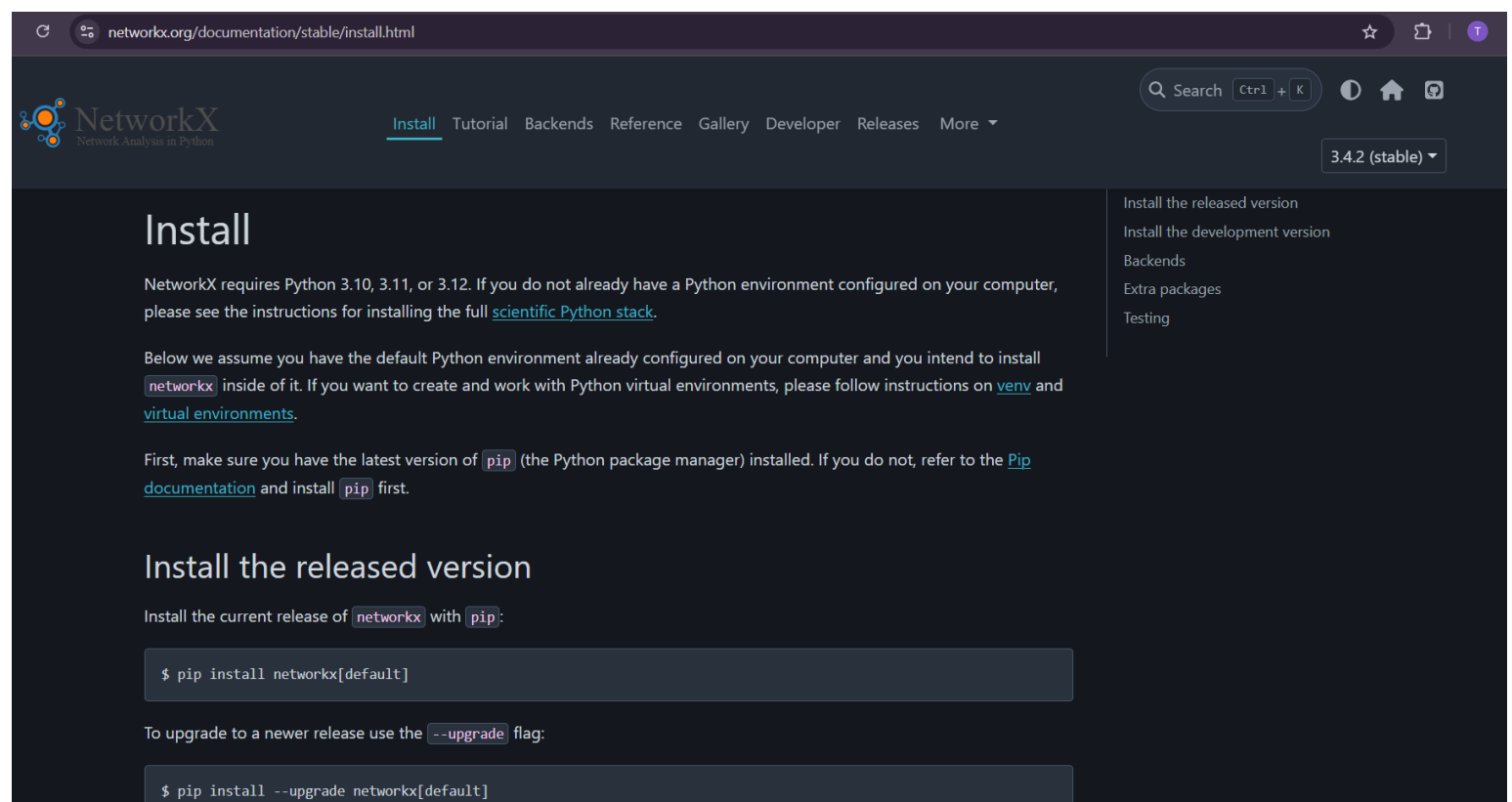
These are the main features of NetworkX

Installing the NetworkX:

The way of installing the NetworkX in the windows:

For installing the NetworkX in the windows.Open Command prompt and run code "`pip install networkx`". For verifying it open python and execute code NetworkX. If codes run without problem then we can conform that NetworkX is successfully installed in the windows.

For upgrading the NetworkX use command of "**`pip install --upgrade networkx`**"

The image is a screenshot of the NetworkX documentation website, specifically the installation page. The browser's address bar shows the URL 'networkx.org/documentation/stable/install.html'. The page has a dark theme. At the top, there is a navigation bar with links for 'Install', 'Tutorial', 'Backends', 'Reference', 'Gallery', 'Developer', 'Releases', and 'More'. A search bar is also present. The main content area is titled 'Install' and contains text explaining the requirements for Python (3.10, 3.11, or 3.12) and the installation process using pip. It includes code blocks for installing the current release and upgrading. A sidebar on the right lists other documentation sections like 'Install the released version', 'Install the development version', 'Backends', 'Extra packages', and 'Testing'.

Key features and Explanation

The features used to create the graphs:

- 1.For creating the Undirected Graphs by using the syntax **`nx.Graph`**
- 2.For creating the directed Graphs by using the syntax **`nx.DiGraph`**
- 3.For creating the Multigraphs Graphs by using the syntax **`nx.MultiGraph`**

The differences between undirected and directed graphs is :

Undirected Graphs:

These are the graphs that has no directions on edges. You can traverse the edge in either direction.

- Simple pair node
- Connection is symmetric
- Each vertex has a degree as number of edges connected to it. Since the edges are bidirectional, the degree count is total number of edges connected to a node
- These are often used to model relationships where direction doesn't matter, such as in social networks (friendships), computer networks (connected devices), or undirected physical connections like roads or railways.

Directed Graphs:

These are the graphs that has specific direction, it means that each edge points from one node to another in the given direction only.

- Ordered pair node
- Connection is assymetric
- Each vertex has two degree: Indegree(the number of edges into the vertex), Outdegree(the number of edges out the vertex)
- These are used for modeling systems where direction matters, such as web pages (where hyperlinks go from one page to another), traffic flow (one-way streets), or dependencies in task scheduling.

MultiGraph:

A MultiGraph is a specialized type of graph used in graph theory and computer science where multiple edges (or connections) can exist between any two nodes (vertices). This contrasts with the typical graph structure, where there is at most one edge between two vertices.

For other types of graphs

Different graph types and plotting can be done using `networkx` drawing and `matplotlib`.

Note** : Here **keywords** is referred to optional keywords that we can mention use to format the graph plotting. Some of the general graph layouts are :

1. **`draw_circular(G, keywords)`** : This gives circular layout of the graph *G*.
2. **`draw_planar(G, keywords)`** :] This gives a planar layout of a planar *networkx* graph *G*.
3. **`draw_random(G, keywords)`** : This gives a random layout of the graph *G*.
4. **`draw_spectral(G, keywords)`** : This gives a spectral 2D layout of the graph *G*.
5. **`draw_spring(G, keywords)`** : This gives a spring layout of the graph *G*.
6. **`draw_shell(G, keywords)`** : This gives a shell layout of the graph *G*.

Graph visualisation

For this purpose there are many libraries that can be used for this purpose like Matplotlib integration, layout algorithms.

The built-in `nx.draw()` function can draw a graph and customize node sizes, colors, edge widths, etc.

It also supports various formats to import/export graphs, including GraphML, GML, Pajek, Adjacency List/Matrix

Python | Visualize graphs generated in NetworkX using Matplotlib

Last Updated : 14 Aug, 2021



Prerequisites: [Generating Graph using Network X](#), [Matplotlib Intro](#)

In this article, we will be discussing how to plot a graph generated by NetworkX in Python using Matplotlib. **NetworkX** is not a graph visualizing package but basic drawing with Matplotlib is included in the software package.

Step 1 : Import networkx and matplotlib.pyplot in the project file.

Python3



```
# importing networkx
import networkx as nx

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



Step 2 : Generate a graph using networkx.

Step 3 : Now use draw() function of networkx.drawing to draw the graph.

Step 4 : Use savefig("filename.png") function of matplotlib.pyplot to save the drawing of graph in filename.png file.

Network Metrics and Statistics

It helps for various types

Degree Distribution: Calculate the degree (number of connections) for nodes in the graph

Path Lengths: Functions to compute the average shortest path length, diameter, etc., of the graph

Clustering Coefficient: Measures the tendency of nodes to cluster together in the graph

Assortativity: Measures the correlation of node degrees in the graph.

These are the key features of NetworkX

Saving the Networkx graph in gexf format

Saving the NetworkX graph in gexf format

To achieve this we will be employing `write_gexf()` function which as the name suggests saves a networkx graph into gexf format easily.

Syntax:

```
networkx.write_gexf( G , path )
```

Parameter:

- ***G:*** In this argument NetworkX graph object or simply the graph is sent as parameter.
- ***path:*** In this argument a valid path for saving the graph is specified.

Approach:

- Import module
- Create a networkx graph
- Save this graph in gexf format

Codes regarding NetworkX

In [1]:

#1. Simple Graph Example

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

Create a graph object

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

Add nodes

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

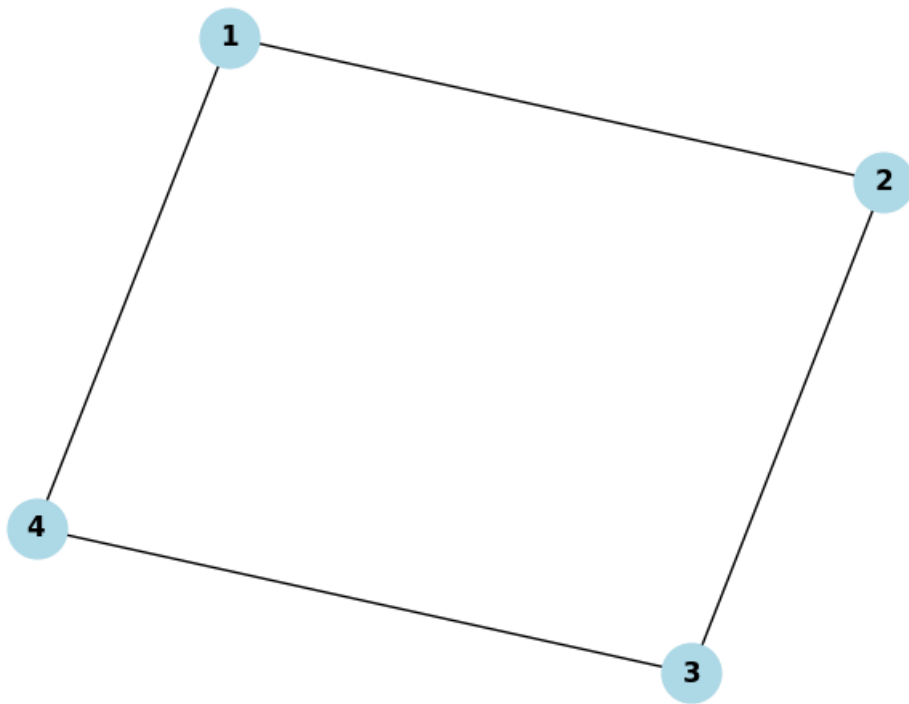
Add edges between nodes

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

Draw the graph

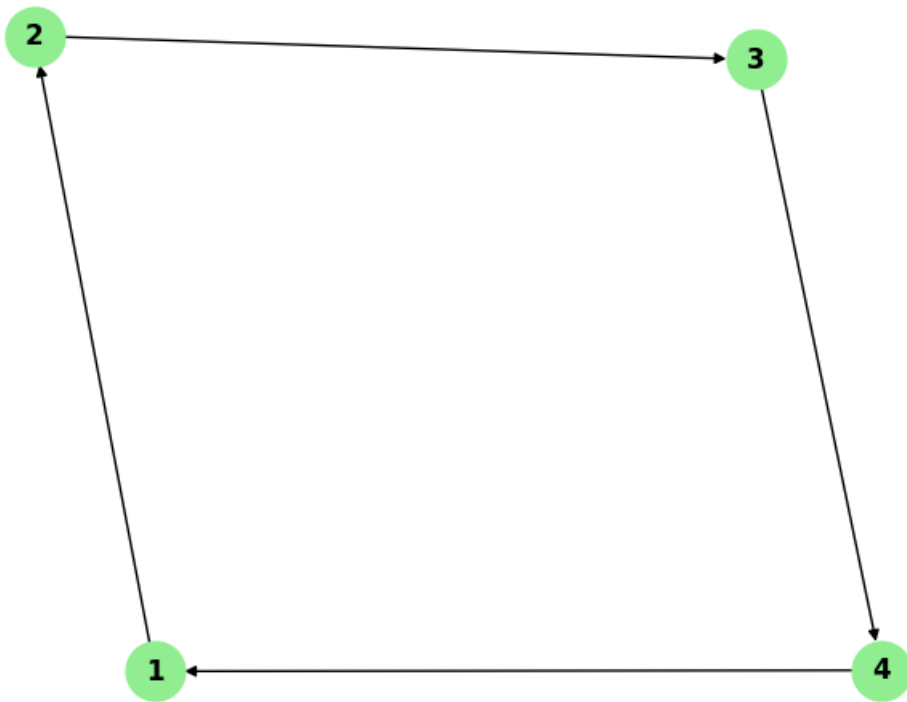
```
nx.draw(G, with_labels=True, node_color='lightblue', font_weight='bold', node_size=700)  
plt.title("Simple Graph")  
plt.show()
```

Simple Graph



```
In [2]:  
#2. Directed Graph Example  
#This example shows how to create a directed graph (with arrows pointing in a specific direction).  
  
import networkx as nx  
import matplotlib.pyplot as plt  
  
# Create a directed graph  
G = nx.DiGraph()  
  
# Add nodes  
G.add_nodes_from([1, 2, 3, 4])  
  
# Add directed edges (edges with direction)  
G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1)])  
  
# Draw the directed graph  
nx.draw(G, with_labels=True, node_color='lightgreen', font_weight='bold', node_size=700, arrows=True)  
plt.title("Directed Graph")  
plt.show()
```

Directed Graph



```
In [3]:
#3. Weighted Graph Example
#In this example, we add weights to the edges of the graph.

import networkx as nx
import matplotlib.pyplot as plt

# Create a weighted graph
G = nx.Graph()

# Add nodes
G.add_nodes_from([1, 2, 3, 4])

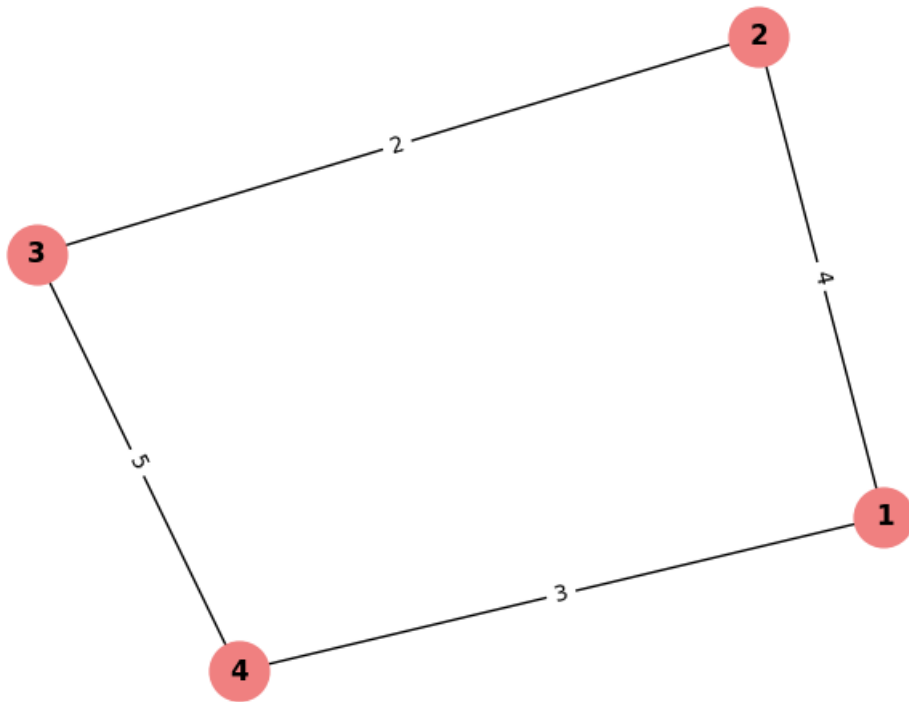
# Add weighted edges
G.add_edge(1, 2, weight=4)
G.add_edge(2, 3, weight=2)
G.add_edge(3, 4, weight=5)
G.add_edge(4, 1, weight=3)

# Draw the graph with edge weights
pos = nx.spring_layout(G) # positions for nodes
nx.draw(G, pos, with_labels=True, node_color='lightcoral', node_size=700, font_weight='bold')

# Draw edge labels (weights)
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

plt.title("Weighted Graph")
plt.show()
```

Weighted Graph



```
In [4]:
#4. Bipartite Graph Example
#This example shows how to create a bipartite graph, where nodes are divided into two sets.
import networkx as nx
import matplotlib.pyplot as plt

# Create a bipartite graph
B = nx.Graph()

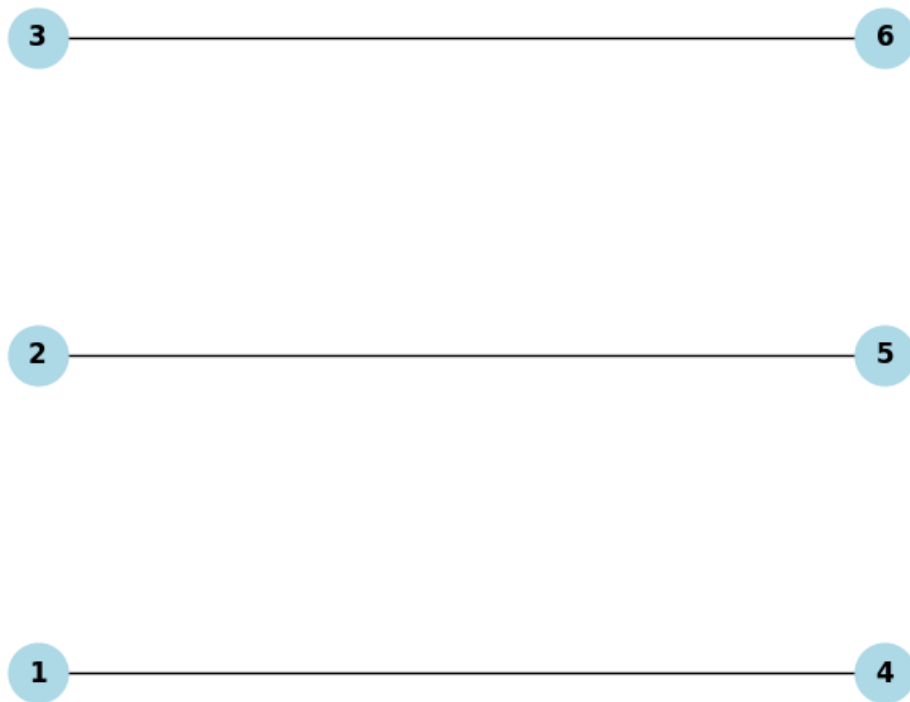
# Add nodes with a bipartite attribute
B.add_nodes_from([1, 2, 3], bipartite=0) # Set 0
B.add_nodes_from([4, 5, 6], bipartite=1) # Set 1

# Add edges between the two sets
B.add_edges_from([(1, 4), (2, 5), (3, 6)])

# Draw the bipartite graph
pos = nx.bipartite_layout(B, nodes=[1, 2, 3]) # Position nodes of set 0 on top
nx.draw(B, pos, with_labels=True, node_color='lightblue', node_size=700, font_weight='bold')

plt.title("Bipartite Graph")
plt.show()
```

Bipartite Graph



In [5]:
#5. Random Graph Example
#Here's an example where we generate a random graph using the erdos_renyi_graph method.

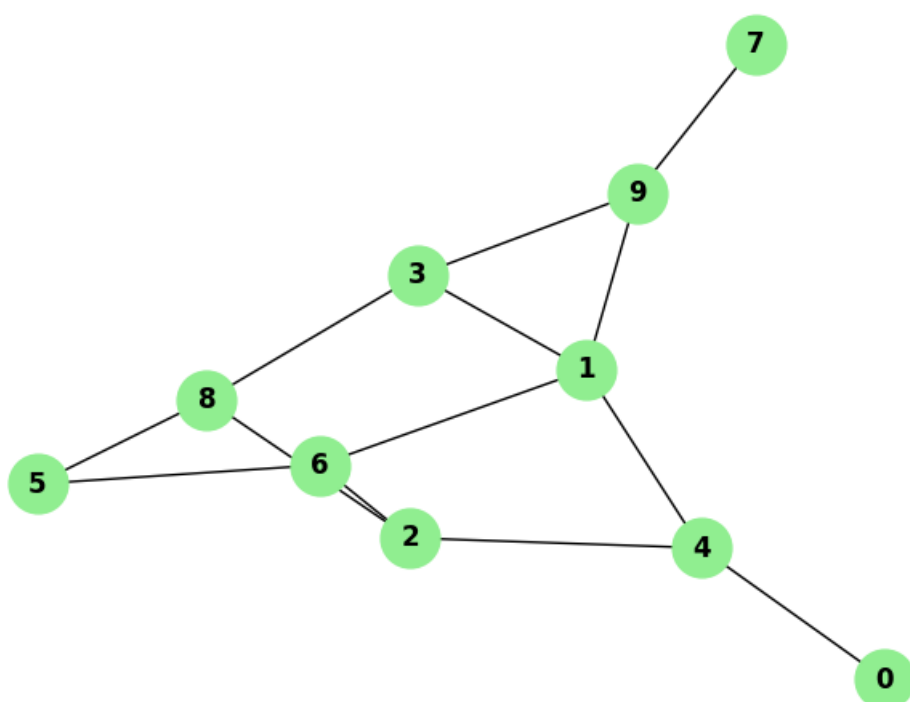
```

import networkx as nx
import matplotlib.pyplot as plt

# Generate a random graph (Erdős-Rényi model)
G = nx.erdos_renyi_graph(n=10, p=0.3) # n=10 nodes, p=0.3 probability of edge creation

# Draw the random graph
nx.draw(G, with_labels=True, node_color='lightgreen', font_weight='bold', node_size=700)
plt.title("Random Graph")
plt.show()
  
```

Random Graph



The real life examples for the usage of NetworkX

These are the usages of NetworkX in real life:

1. In social media platforms:

It can be used as following networks of users can be represented as graphs, where users are nodes, and their interactions (like friendships or followings) are edges. One of the most common tasks is to detect communities

2. Transportation and Logistics

In transportation networks, you can use graphs to find the shortest or most efficient routes between two points.

3. Biology

In biology, networks of genes or proteins can be represented as graphs, where nodes represent genes and edges represent interactions between them.

4. Network Traffic

In computer networks, routers and switches can be represented as nodes, and network links can be represented as edges.

5. Supply Chain and Inventory Management

In supply chains, inventory systems can be modeled as graphs where products and warehouses are nodes, and edges represent the flow of goods.

Syntaxes regarding NetworkX

1. Undirected Graph: `nx.Graph()`
2. Directed Graph: `nx.DiGraph()`
3. MultiGraph: `nx.MultiGraph()`
4. MultiDiGraph: `nx.MultiDiGraph()`
5. Add nodes: `G.add_nodes_from([2, 3, 4])`
6. Adding edges: `G.add_edges_from([(1, 2), (2, 3)])` *italicized text*
7. Shortest path length: `nx.shortest_path_length`
8. Graph Algorithms: `components = list(nx.connected_components(G))` , `scc = list(nx.strongly_connected_components(G))` , `clustering = nx.clustering(G, 1)`
9. Pathfinding Algorithms: `path = nx.dijkstra_path(G, source=1, target=4, weight='weight')` , `path = nx.astar_path(G, source=1, target=4, heuristic=None, weight='weight')`
10. Subgraphs: `subgraph = G.subgraph([1, 2, 3])`
11. Graph Metrics: `centrality = nx.betweenness_centrality(G)` , `centrality = nx.closeness_centrality(G)` , `pagerank = nx.pagerank(G)`

The screenshot shows the NetworkX documentation website. The browser address bar displays `networkx.org/documentation/stable/tutorial.html`. The page header includes the NetworkX logo, navigation links (Install, Tutorial, Backends, Reference, Gallery, Developer, Releases, More), a search bar, and a version indicator (3.4.2). The main content area is titled 'Nodes #' and explains how to create a graph. It provides code examples for adding nodes and edges, and a sidebar on the right lists various graph-related topics.

Nodes #

The graph `G` can be grown in several ways. NetworkX includes many [graph generator functions](#) and [facilities to read and write graphs in many formats](#). To get started though we'll look at simple manipulations. You can add one node at a time,

```
G.add_node(1)
```

or add nodes from any [iterable](#) container, such as a list

```
G.add_nodes_from([2, 3])
```

You can also add nodes along with node attributes if your container yields 2-tuples of the form `(node, node_attribute_dict)`:

```
G.add_nodes_from([(4, {"color": "red"}), (5, {"color": "green"})])
```

Node attributes are discussed further [below](#).

Nodes from one graph can be incorporated into another:

```
H = nx.path_graph(10)
G.add_nodes_from(H)
```

Creating a graph

- Nodes
- Edges
- Examining elements of a graph
- Removing elements from a graph
- Using the graph constructors
- What to use as nodes and edges
- Accessing edges and neighbors
- Adding attributes to graphs, nodes, and edges
- Directed graphs
- Multigraphs
- Graph generators and graph operations
- Analyzing graphs
- Using NetworkX backends
- Drawing graphs
- NX-Guides

NetworkX : Python software package for study of complex networks

Last Updated : 11 Jul, 2022



NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks. It is used to study large complex networks represented in form of graphs with nodes and edges. Using networkx we can load and store complex networks. We can generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms and draw networks.

Installation of the package:

```
pip install networkx
```

Creating Nodes

Add one node at a time:

```
G.add_node(1)
```

Add a list of nodes:

```
G.add_nodes_from([2,3])
```

Removing Nodes and Edges:

One can demolish the graph using any of these functions:

```
Graph.remove_node(), Graph.remove_nodes_from(),  
Graph.remove_edge() and Graph.remove_edges_from()
```

Scrapy Project

A Scrapy project is a framework for building web crawlers and scrapers in Python. It allows you to extract and process data from websites in an automated way. Scrapy is highly efficient and is widely used for web scraping, data mining, and extracting information from websites for various purposes, such as research, business intelligence, and machine learning datasets.

Step 1: Create scrapy project

Execute the following command, at the terminal, to create a Scrapy project –

```
scrapy startproject gfg_friendshipquotes
```

This will create a new directory, called “gfg_friendshipquotes”, in your current directory. Now change the directory, to the newly created folder.

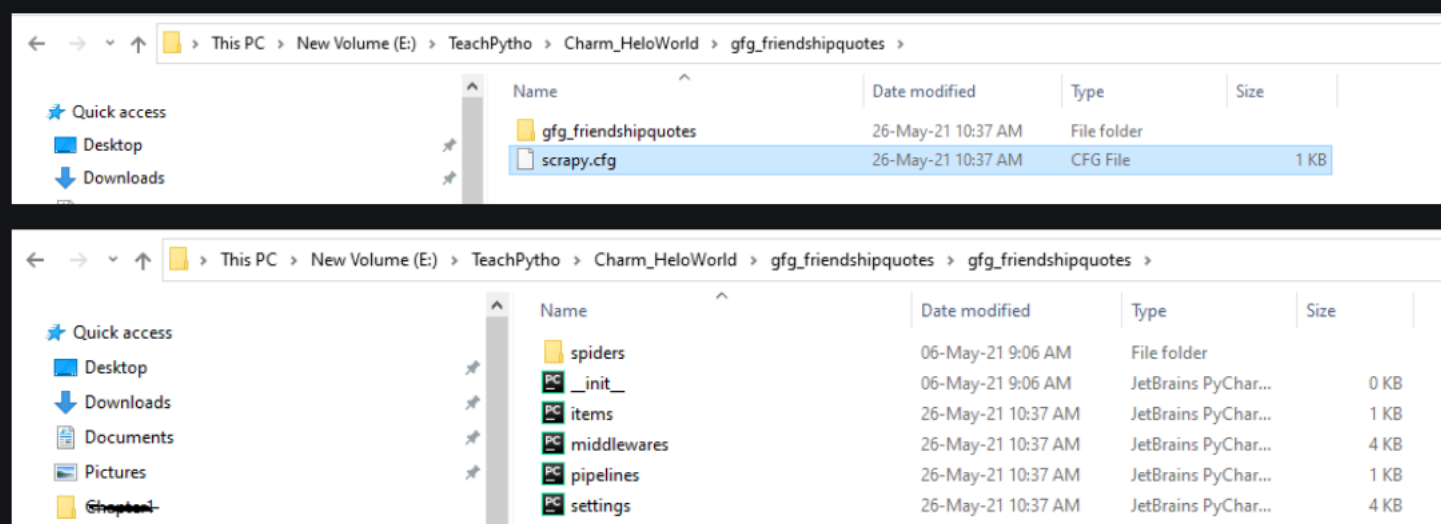
```
Terminal: Local < +
(venv) E:\TeachPytho\Charm_HeloWorld> scrapy startproject gfg_friendshipquotes
New Scrapy project 'gfg_friendshipquotes', using template directory 'e:\teachpytho\charm_heloworld\venv\lib\site-packages\scrapy\templates\project', created in:
E:\TeachPytho\Charm_HeloWorld\gfg_friendshipquotes

You can start your first spider with:
cd gfg_friendshipquotes
scrapy genspider example example.com

(venv) E:\TeachPytho\Charm_HeloWorld> cd gfg_friendshipquotes

(venv) E:\TeachPytho\Charm_HeloWorld\gfg_friendshipquotes>
```

The folder structure of ‘gfg_friendshipquotes’ is as displayed below. Keep the contents of the configuration files as they are, currently.



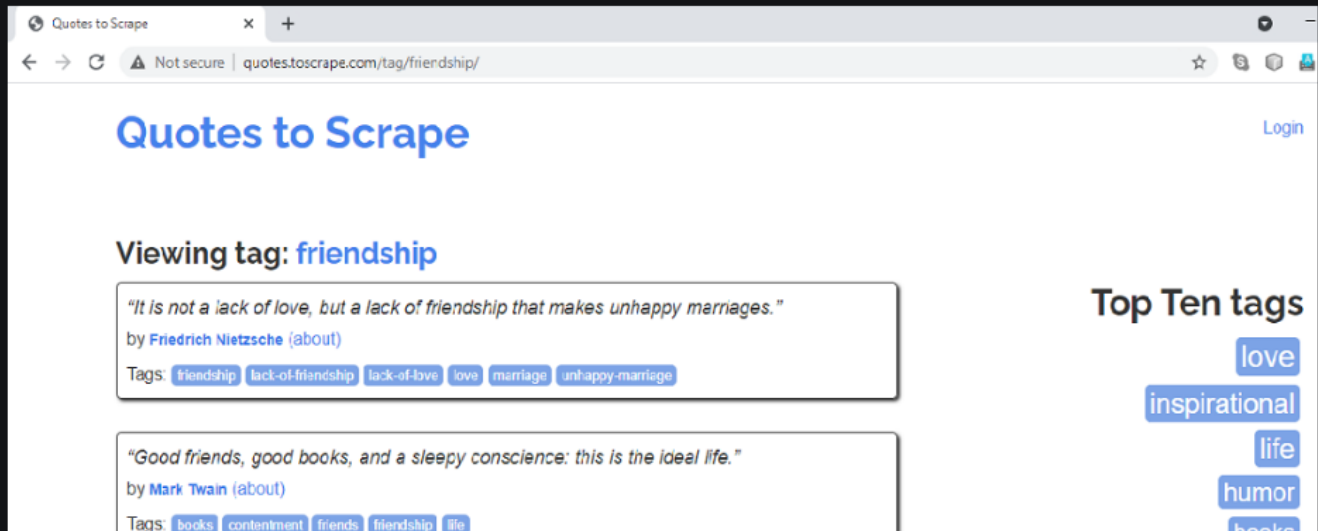
Step 2: To create a spider file, we use the command ‘genspider’. Please see that genspider command is executed at the same directory level, where scrapy.cfg file is present. The command is –

```
scrapy genspider spider_filename "url_of_page_to_scrape"
```

Now, execute the following at the terminal:

```
scrapy genspider gfg_friendquotes "quotes.toscrape.com/tag/friendship/"
```


Step 3: Now, let's analyze the XPath expressions for required elements. If you visit the link, <http://quotes.toscrape.com/tag/friendship/> it looks as follows:



URL of the page that we will scrape

We are going to scrape the friendship quotes titles, authors, and tags. When you right-click on Quotes, block it, and select Inspect option, one can notice they belong to class “quote”. As you hover over the rest of the quote blocks, one can notice that all the quotes, in the webpage, have the CSS class attribute as “quote”.

Creating CSV file:

For storing the data in a CSV file, one can follow any of the methods mentioned below.

Write the following command at the terminal:

```
scrapy crawl gfg_friendquotes -o friendshipquotes.csv
```

Conclusion

NetworkX is a powerful and versatile library for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. With its simple and intuitive API, it allows users to model various types of networks, including graphs, trees, and directed acyclic graphs (DAGs), and perform tasks like graph traversal, pathfinding, clustering, and visualization. Additionally, it supports the analysis of both small and large-scale networks, integrating seamlessly with other Python libraries for scientific computing like NumPy and SciPy

NetworkX provides the tools needed to explore and understand networks effectively

References and further information:

These are the further websites for more information of **NetworkX**

<https://networkx.org/documentation/stable/install.html>

<https://github.com/networkx/networkx>

<https://www.geeksforgeeks.org/networkx-python-software-package-study-complex-networks/>