Networkx python package explanation

- **NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks**

- **the structure of NetworkX can be seen by the organization of its source code. The package provides classes for graph objects, generators to create standard graphs, IO routines for reading in existing datasets, algorithms to analyze the resulting networks and some basic drawing tools.**

- **Source code: https://github.com/networkx/networkx**

- **import the package:**

```
import networkx as nx
```

The following basic graph types are provided as Python classes:

- **Graph**

This class implements an undirected graph. It ignores multiple edges between two nodes. It does allow self-loop edges between a node and itself.

- **DiGraph**

Directed graphs, that is, graphs with directed edges. Provides operations common to directed graphs, (a subclass of Graph).

- **MultiGraph**

A flexible graph class that allows multiple undirected edges between pairs of nodes. The additional flexibility leads to some degradation in performance, though usually not significant.

- **MultiDiGraph**

A directed version of a MultiGraph.

- **So, we use Graph:**

```
g = nx.Graph()
```

- **Graph Creation**

NetworkX graph objects can be created in one of three ways:

- Graph generators—standard algorithms to create network topologies.
- Importing data from pre-existing (usually file) sources.
- Adding edges and nodes explicitly.

Networkx python package explanation

- **So, we used Adding edges and nodes explicitly:**
- create company node in graph:

```
g.add_node(company name,color= 'Cyan',pos=(1,2))
```

- add all users in company
- and add edges between each user node and company node

I need to mention that w is a list have all users data (name, state)

```
    p=1
    #create nodes for all user and set color node as state color.
    for i in w:
        g.add_node(i[0] ,color=i[1],pos=(p,p))
        g.add_edge(i[0], l[7])
        p=p+1
        nx.draw_networkx_edge_labels(g,pos=nx.spring_layout(g))
```

- **Graph.add_node**

Add a single node node_for_adding and update node attributes.

Parameters

node_for_adding: [node] A node can be any hash able Python object except None.

Attr : [keyword arguments, optional] Set or change node attributes using key=value

- **Graph.add_edge**

Add an edge between u and v.

The nodes u and v will be automatically added if they are not already in the graph.

Edge attributes can be specified with keywords or by directly accessing the edge's attribute dictionary.

Parameters

u_of_edge, v_of_edge: [nodes] Nodes can be, for example, strings or numbers. Nodes must be hash able (and not None) Python objects.

Attr: [keyword arguments, optional] Edge data (or labels or objects) can be assigned using keyword arguments.

- **draw_networkx_labels**

Draw node labels on the graph g.

Parameters

g: [graph] A networkx graph

Pos: [dictionary] A dictionary with nodes as keys and positions as values. Positions should be sequences of length 2.

Networkx python package explanation

- **Drawing**

The basic drawing functions essentially place the nodes on a scatterplot using the positions you provide via a dictionary, or the positions are computed with a layout function. The edges are lines between those dots.

- **get_node_attributes**

Get node attributes from graph

Parameters:

- G: [NetworkX Graph]

- Name: [string] Attribute name

Returns:

Dictionary of attributes keyed by node

```
color=nx.get_node_attributes(g,'color')
pos=nx.get_node_attributes(g,'pos')
```

- now we have color and position for each node.

```
nx.draw(g, pos,with_labels = True,node_size=4000,node_color=color)
```

- draw_networkx

Draw the graph G using Matplotlib.

Draw the graph with Matplotlib with options for node positions, labeling, titles, and many other drawing features.

See draw () for simple drawing without labels or axes.

Parameters

G: [graph] A networkx graph

Pos: [dictionary, optional] A dictionary with nodes as keys and positions as values. If not specified a spring layout positioning will be computed. using networkx.drawing.layout for functions that compute node positions

with_labels: [bool (default=True)] Set to True to draw labels on the nodes.

node_size: [scalar or array (default=300)] Size of nodes. If an array is specified it must be the same length as nodelist.

node_color: [color or array of colors (default='#1f78b4')] Node color. Can be a single color or a sequence of colors with the same length as nodelist. Color can be string or rgb (or rgba) tuple of floats

from 0-1. If numeric values are specified, they will be mapped to colors using the cmap and vmin, vmax parameters. See matplotlib. scatter for more details.

- **Finally save graph as image to visualize it in html page:**

```
plt.savefig(os.getcwd()+"\static\img\Graph.png", format="PNG")
```

****