

The basics

Import library	<code>import networkx as nx</code>
Import matplotlib for graph drawing	<code>import matplotlib.pyplot as plt</code>
Initialize (Di)graphs	<code>G = nx.Graph(), G = nx.DiGraph()</code>
Create a (Di)graph from another graph	<code>G = nx.Graph(H), G = nx.DiGraph(H)</code>
Copy existing graphs	<code>G2 = G1.copy()</code>

Unweighted node and edge creation/deletion

Add nodes	<code>G.add_node(1)</code> <code>G.add_nodes_from(['a', 'b'])</code>
Add (directed) edges	<code>G.add_edge(1, 2)</code> <code>G.add_edges_from([(1, 'a'), (1, 'c')])</code>
Delete nodes	<code>G.remove_node(1)</code> <code>G.remove_nodes_from(['a', 'b'])</code>
Delete edges	<code>G.remove_edge(1, 2)</code> <code>G.remove_edges_from([(1, 'a'), (1, 'c')])</code>

If G is directed, then these take into account edge direction.

Weights and attributes

Add nodes with attribute	<code>G.add_nodes_from([1, 2], color = 'blue')</code>
Change node attributes	<code>G.nodes[1]['color'] = 'red'</code>
Add edges with attribute	<code>G.add_edges_from([(1,2), (2,4)], weight=2)</code>
Change edge attributes	<code>G.edges[1,2]['weight'] = 2</code>
Set attributes from dictionary	<code>nx.set_node_attributes(G, {1: {'city': 'd'}})</code>

Attributes can have any name (color, timestamp, weight, etc). However for edge weights they should have 'weight', as many functions for weighted graphs assume it.

Basic graph properties

Number of nodes	<code>N = len(G.nodes())</code>
Number of edges	<code>E = len(G.edges())</code>
Node-degree dictionary	<code>degreedict = G.degrees</code>
Directed-ness	<code>G.is_directed(), G.is_undirected()</code>
Planarity	<code>G.is_planar()</code>
Diameter, radius	<code>d = nx.diameter(G), r = nx.radius(G)</code>
Average shortest path length	<code>aspl = nx.average_shortest_path_length(G)</code>

Iterate over a graph

Iterate over nodes/-edges

```
odelist = [n for n in G.nodes()]
edgelist = [n for n in G.edges()]
```

Iterate over node/edge attributes

```
node_attrs = [n for n in G.nodes(data=True)]
edge_attrs = [n for n in G.edges(data=True)]
```

Iterate over in/out edges

```
in_list = [e for e in G.in_edges()]
out_list = [e for e in G.out_edges()]
```

Iterate over node's neighbors

```
neighs = [n for n in G.neighbors(node)]
```

Iterate over successors/pred-ecessors

```
succ = [n for n in G.successors(node)]
pred = [n for n in G.predecessors(node)]
```

Connectivity properties

Connectedness

```
G.is_connected()
G.is_weakly_connected(),
G.is_strongly_connected()
```

Contains node or edge?

```
G.has_node(node), G.has_edge(*edge)
```

Connectivity properties (cont)

List of connected components

```
cc = nx.connected_components(G)
wcc = nx.weakly_connected_components(G)
scc = nx.strongly_connected_components(G)
```

Largest (simply) connected component

```
largest_cc = max(nx.connected_components(G), key=len)
LCC = G.subgraph(largest_cc).copy()
```

Some matricial representations

Adjacency matrix (standard)

```
A = nx.to_numpy_array(G, dtype=int)
```

Adjacency matrix (sparse)

```
A = nx.to_scipy_sparse_array(G)
```

Adjacency matrix eigenvalues

```
eigs = nx.adjacency_spectrum(G)
```

Graph from adjacency matrix

```
G = nx.from_numpy_array(A)
G = nx.from_scipy_sparse_array(A)
```

Incidence matrix (sparse)

```
I = nx.incidence_matrix(G)
```

Laplacian matrix (sparse)

```
L = nx.laplacian_matrix(G)
```



By **gonz95alo**
cheatography.com/gonz95alo/

Published 7th September, 2022.
 Last updated 23rd October, 2022.
 Page 2 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Some matricial representations (cont)

Laplacian matrix eigenvalues `eigs = nx.laplacian_spectrum(G)`

Google matrix (standard) `Goog = nx.google_matrix(G)`

Sparse matrices/arrays are memory efficient. See the Scipy page on them for their methods, or convert them to numpy arrays with `sparse_array.to_dense()`

Graph plotting

Plotting command, standard options `nx.draw(G, pos=pos, with_labels=True, ax=axis)`

Node positioning options (I) `pos = [nx.circular_layout(G), nx.planar_layout(G)]`

Node positioning options (II) `pos = [nx.shell_layout(G), nx.spring_layout(G)]`

Node colors and sizes in `nx.draw()` `node_color = colorlist`
`node_size = sizelist`

Edge colors and widths in `nx.draw()` `edge_color = colorlist`
`width = widthlist`

Node fine-tuning `nx.draw_networkx_nodes(G, other options)`

Edge fine-tuning `nx.draw_networkx_edges(G, other options)`

Centrality measures

Degree `C = nx.degree_centrality(G)`

Betweenness `C = nx.betweenness_centrality(G)`

Closeness `C = nx.closeness_centrality(G)`

Eigenvector `C = nx.eigenvector_centrality_numpy(G)`

Katz `C = nx.katz_centrality_numpy(G, alpha=1.0)`

PageRank `C = nx.pagerank(G, alpha=0.1, personal=None)`

HITS `C = nx.hits(G)`

All measures return a dictionary {node: value}. They do not take into account weights, one needs to provide an extra argument `weight='weight'` in order to consider them.

Undirected graph generators

Path graph `G = nx.path_graph(N)`

Cycle graph `G = nx.cycle_graph(N)`

Star graph `G = nx.star_graph(N)`

Complete graph `G = nx.complete_graph(N)`

Erdős-Renyi (random) `G = nx.erdos_renyi_graph(N, p)`

Barabasi-Albert (scale-free) `G = nx.barabasi_albert_graph(N, m)`

Watts-Strogatz (small world) `G = nx.watts_strogatz_graph(N, p)`