

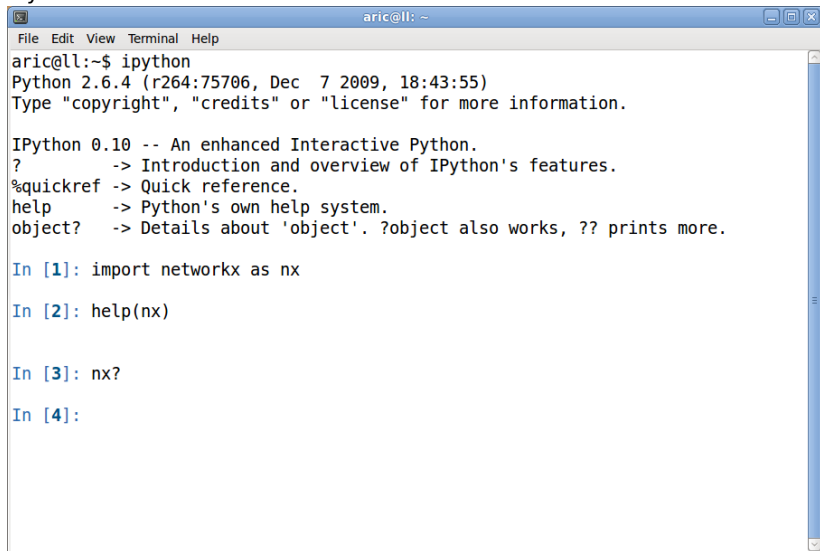
3 - Getting Started with NetworkX

Drew Conway and Aric Hagberg

June 29, 2010

- ▶ Running Python and loading NetworkX
- ▶ Creating a Graph, adding nodes and edges
- ▶ Finding what is in NetworkX
- ▶ Interacting with NetworkX graphs
- ▶ Graph generators and operators
- ▶ Basic analysis of graphs

IPython Command line



The screenshot shows a terminal window titled "aric@ll: ~". The menu bar includes "File", "Edit", "View", "Terminal", and "Help". The terminal content is as follows:

```
aric@ll:~$ ipython
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: import networkx as nx

In [2]: help(nx)

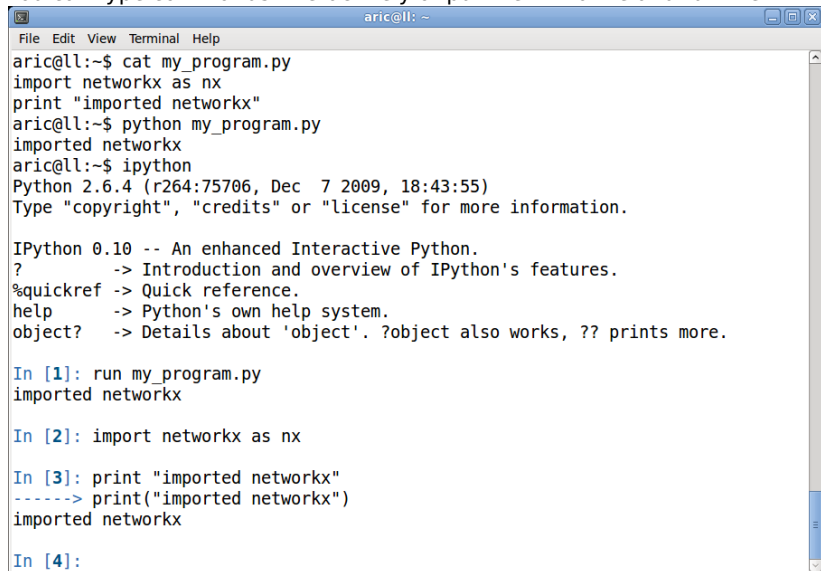
In [3]: nx?

In [4]:
```

No GUI <http://www.cryptonomicon.com/beginning.html>

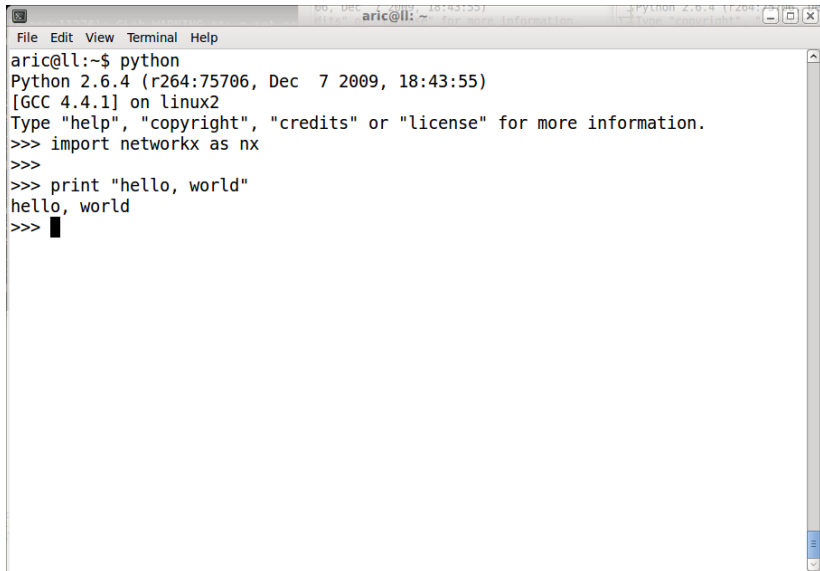
Command line vs executing file

You can type commands interactively or put them in a file and run them.



```
aric@ll: ~  
File Edit View Terminal Help  
aric@ll:~$ cat my_program.py  
import networkx as nx  
print "imported networkx"  
aric@ll:~$ python my_program.py  
imported networkx  
aric@ll:~$ ipython  
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 0.10 -- An enhanced Interactive Python.  
?          -> Introduction and overview of IPython's features.  
%quickref  -> Quick reference.  
help       -> Python's own help system.  
object?    -> Details about 'object'. ?object also works, ?? prints more.  
  
In [1]: run my_program.py  
imported networkx  
  
In [2]: import networkx as nx  
  
In [3]: print "imported networkx"  
-----> print("imported networkx")  
imported networkx  
  
In [4]:
```

The > > > (doctests)



The screenshot shows a terminal window with a title bar containing the text "aric@ll: ~" and a timestamp "60, Dec 7 2009, 18:43:55". The terminal has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The session starts with the command "python", which outputs the Python version "2.6.4 (r264:75706, Dec 7 2009, 18:43:55)", the compiler "[GCC 4.4.1] on linux2", and a prompt to type "help", "copyright", "credits", or "license" for more information. The user then enters a series of commands: "import networkx as nx", followed by three blank lines, then "print 'hello, world'", which outputs "hello, world", and finally another blank line.

```
aric@ll:~$ python
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import networkx as nx
>>>
>>> print "hello, world"
hello, world
>>>
```

The basic *Graph* object is used to hold the network information.
Create an empty graph with no nodes and no edges:

```
1 >>> import networkx as nx
2
3 >>> G=nx.Graph()
```

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.

```
5 # One node at a time
6 >>> G.add_node(1) # "method" of G
7
8 # A list of nodes
9 >>> G.add_nodes_from([2,3])
10
11 # A container of nodes
12 >>> H=nx.path_graph(10)
13 >>> G.add_nodes_from(H) # G now contains the nodes of H
14
15 # In contrast, you could use the graph H as a node in G.
16 >>> G.add_node(H) # G now contains Graph H as a node
```

Nodes can be any hashable object such as strings, numbers, files, functions, and more.

G can also be grown by adding edges.

```
18 # Single edge
19 >>> G.add_edge(1,2)
20 >>> e=(2,3)
21 >>> G.add_edge(*e) # unpack edge tuple*
22
23 # List of edges
24
25 >>> G.add_edges_from([(1,2),(1,3)])
26
27 # Container of edges
28 >>> G.add_edges_from(H.edges())
```

If the nodes do not already exist they are automatically added to the graph.
You can demolish the graph similarly with

G.remove_node, G.remove_nodes_from,
G.remove_edge, G.remove_edges_from.

- ▶ How do I find out the names of the methods like `add_edge`?
- ▶ How do I see what is in my graph?

What's in NetworkX?

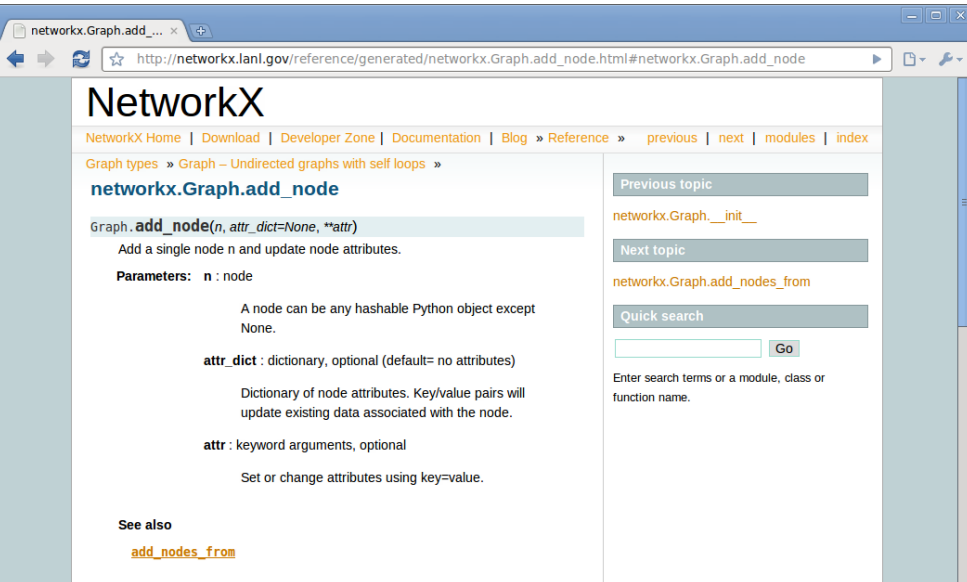
Graph - Undirected g... x

http://networkx.lanl.gov/reference/classes.graph.html

Adding and removing nodes and edges

<code>Graph.__init__(**attr[, data, name])</code>	Initialize a graph with edges, name, graph attributes.
<code>Graph.add_node(n, **attr[, attr_dict])</code>	Add a single node n and update node attributes.
<code>Graph.add_nodes_from(nodes, **attr)</code>	Add multiple nodes.
<code>Graph.remove_node(n)</code>	Remove node n.
<code>Graph.remove_nodes_from(nodes)</code>	Remove multiple nodes.
<code>Graph.add_edge(u, v, **attr[, attr_dict])</code>	Add an edge between u and v.
<code>Graph.add_edges_from(ebunch, **attr[, attr_dict])</code>	Add all the edges in ebunch.
<code>Graph.add_weighted_edges_from(ebunch, **attr)</code>	Add all the edges in ebunch as weighted edges with specified weights.
<code>Graph.remove_edge(u, v)</code>	Remove the edge between u and v.
<code>Graph.remove_edges_from(ebunch)</code>	Remove all edges specified in ebunch.
<code>Graph.add_star(nlist, **attr)</code>	Add a star.
<code>Graph.add_path(nlist, **attr)</code>	Add a path.
<code>Graph.add_cycle(nlist, **attr)</code>	Add a cycle.
<code>Graph.clear()</code>	Remove all nodes and edges from the graph.

What's in NetworkX?



The screenshot shows a web browser window with the address bar displaying `http://networkx.lanl.gov/reference/generated/networkx.Graph.add_node.html#networkx.Graph.add_node`. The page title is "NetworkX". Below the title is a navigation bar with links: [NetworkX Home](#) | [Download](#) | [Developer Zone](#) | [Documentation](#) | [Blog](#) » [Reference](#) » [previous](#) | [next](#) | [modules](#) | [index](#). Below the navigation bar is a breadcrumb trail: [Graph types](#) » [Graph – Undirected graphs with self loops](#) » [networkx.Graph.add_node](#). The main content area has the heading `networkx.Graph.add_node` in blue. Below it is the function signature: `Graph.add_node(n, attr_dict=None, **attr)`. The description says: "Add a single node n and update node attributes." The parameters section lists: **Parameters:** `n` : node. A node can be any hashable Python object except None. `attr_dict` : dictionary, optional (default= no attributes). Dictionary of node attributes. Key/value pairs will update existing data associated with the node. `attr` : keyword arguments, optional. Set or change attributes using key=value. The "See also" section lists [add_nodes_from](#). On the right side, there are three sections: "Previous topic" with a link to `networkx.Graph.__init__`, "Next topic" with a link to `networkx.Graph.add_nodes_from`, and "Quick search" with a search input field and a "Go" button. Below the search bar, it says: "Enter search terms or a module, class or function name."

networkx.Graph.add_node

NetworkX Home | Download | Developer Zone | Documentation | Blog » Reference » previous | next | modules | index

Graph types » Graph – Undirected graphs with self loops » [networkx.Graph.add_node](#)

networkx.Graph.add_node

`Graph.add_node(n, attr_dict=None, **attr)`

Add a single node n and update node attributes.

Parameters: `n` : node

A node can be any hashable Python object except None.

attr_dict : dictionary, optional (default= no attributes)

Dictionary of node attributes. Key/value pairs will update existing data associated with the node.

attr : keyword arguments, optional

Set or change attributes using key=value.

See also

[add_nodes_from](#)

Previous topic

[networkx.Graph.__init__](#)

Next topic

[networkx.Graph.add_nodes_from](#)

Quick search

Enter search terms or a module, class or function name.

What's in NetworkX?



The screenshot shows a web browser window with the address bar displaying `http://networkx.lanl.gov/reference/generated/networkx.Graph.add_node.html#networkx.Graph.add_node`. The page content includes a 'See also' section with a link to `add_nodes_from`, a 'Notes' section explaining hashable objects, an 'Examples' section with a code block, and a note about node attributes.

See also

[add_nodes_from](#)

Notes

A hashable object is one that can be used as a key in a Python dictionary. This includes strings, numbers, tuples of strings and numbers, etc.

On many platforms hashable items also include mutables such as NetworkX Graphs, though one should be careful that the hash doesn't change on mutables.

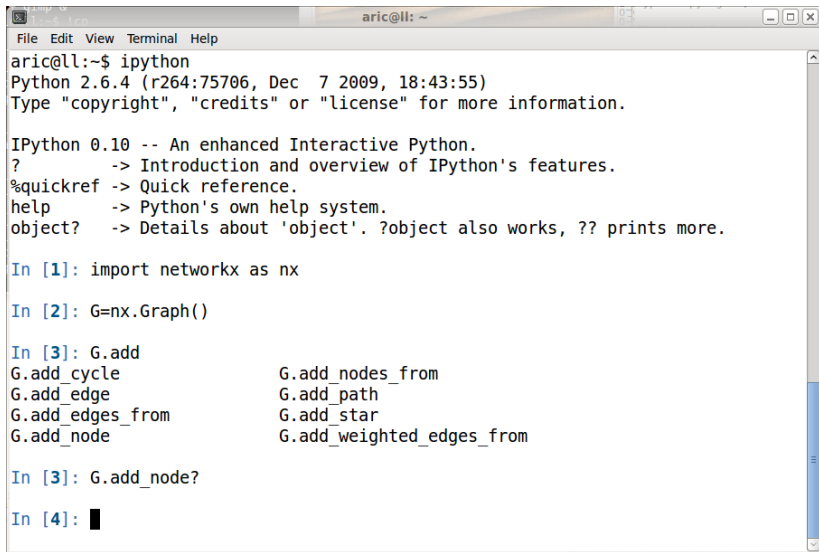
Examples

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_node(1)
>>> G.add_node('Hello')
>>> K3 = nx.Graph([(0,1),(1,2),(2,0)])
>>> G.add_node(K3)
>>> G.number_of_nodes()
3
```

Use keywords set/change node attributes:

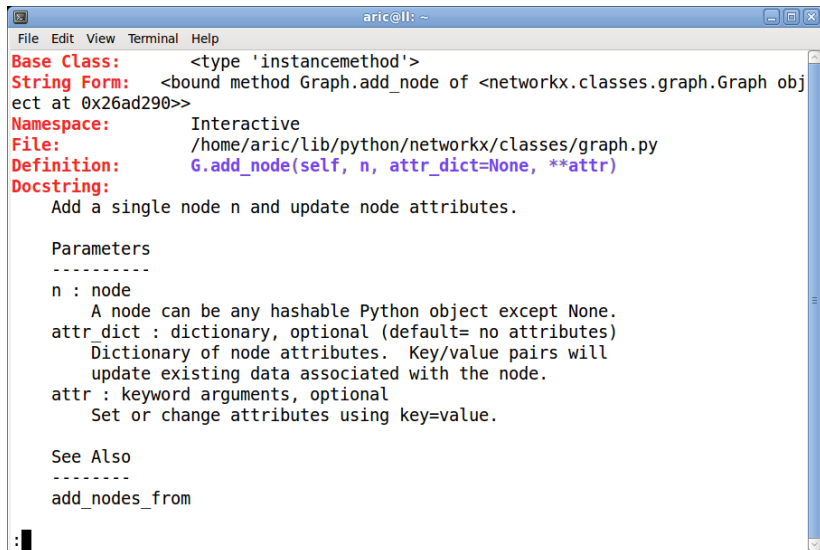
```
>>> G.add_node(1,size=10)
>>> G.add_node(3,weight=0.4,UTM=('13S',382871,3972649))
```

What's in Networkx?



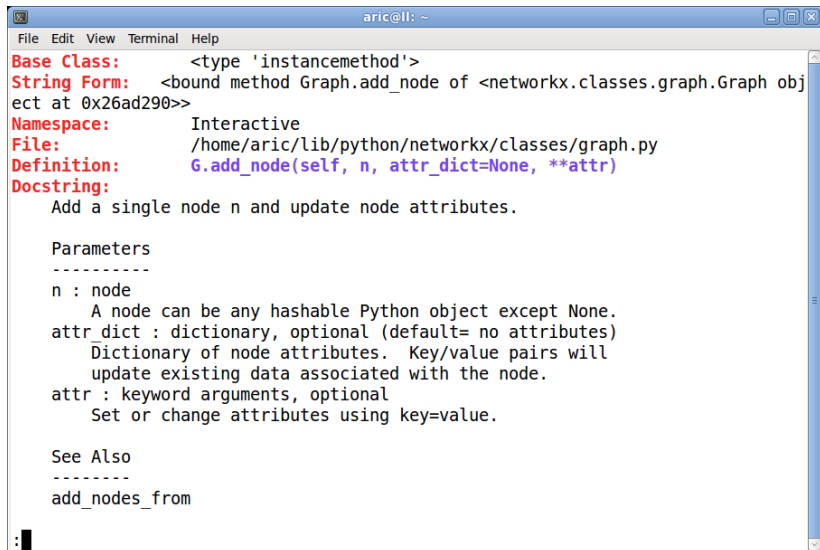
```
aric@ll: ~  
File Edit View Terminal Help  
aric@ll:~$ ipython  
Python 2.6.4 (r264:75706, Dec  7 2009, 18:43:55)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 0.10 -- An enhanced Interactive Python.  
?          -> Introduction and overview of IPython's features.  
%quickref  -> Quick reference.  
help       -> Python's own help system.  
object?    -> Details about 'object'. ?object also works, ?? prints more.  
  
In [1]: import networkx as nx  
  
In [2]: G=nx.Graph()  
  
In [3]: G.add  
G.add_cycle          G.add_nodes_from  
G.add_edge           G.add_path  
G.add_edges_from     G.add_star  
G.add_node           G.add_weighted_edges_from  
  
In [3]: G.add_node?  
  
In [4]: █
```

What's in Networkx?



```
aric@ll: ~  
File Edit View Terminal Help  
Base Class: <type 'instancemethod'>  
String Form: <bound method Graph.add_node of <networkx.classes.graph.Graph object at 0x26ad290>>  
Namespace: Interactive  
File: /home/aric/lib/python/networkx/classes/graph.py  
Definition: G.add_node(self, n, attr_dict=None, **attr)  
Docstring:  
    Add a single node n and update node attributes.  
  
    Parameters  
    -----  
    n : node  
        A node can be any hashable Python object except None.  
    attr_dict : dictionary, optional (default= no attributes)  
        Dictionary of node attributes. Key/value pairs will  
        update existing data associated with the node.  
    attr : keyword arguments, optional  
        Set or change attributes using key=value.  
  
    See Also  
    -----  
    add_nodes_from  
:  
█
```

What's in Networkx?

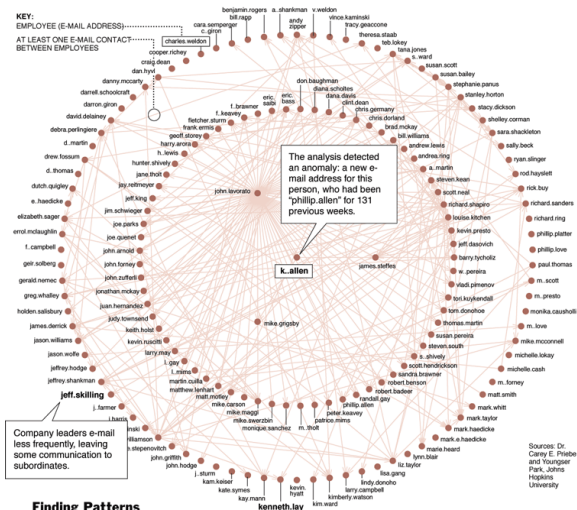


```
aric@ll: ~  
File Edit View Terminal Help  
Base Class: <type 'instancemethod'>  
String Form: <bound method Graph.add_node of <networkx.classes.graph.Graph object at 0x26ad290>>  
Namespace: Interactive  
File: /home/aric/lib/python/networkx/classes/graph.py  
Definition: G.add_node(self, n, attr_dict=None, **attr)  
Docstring:  
    Add a single node n and update node attributes.  
  
    Parameters  
    -----  
    n : node  
        A node can be any hashable Python object except None.  
    attr_dict : dictionary, optional (default= no attributes)  
        Dictionary of node attributes. Key/value pairs will  
        update existing data associated with the node.  
    attr : keyword arguments, optional  
        Set or change attributes using key=value.  
  
    See Also  
    -----  
    add_nodes_from  
:  
█
```

Adding attributes to graphs, nodes, and edges

(Almost) any Python object is allowed as graph, node, and edge data.

- ▶ number
- ▶ string
- ▶ image
- ▶ IP address
- ▶ email address




```
1 >>> import networkx as nx
2 # Assign graph attributes when creating a new graph
3
4 >>> G = nx.Graph(day="Friday")
5 >>> G.graph
6 {'day': 'Friday'} # Python dictionary
7
8 # Or you can modify attributes later
9
10 >>> G.graph['day'] = 'Monday'
11 >>> G.graph
12 {'day': 'Monday'}
```

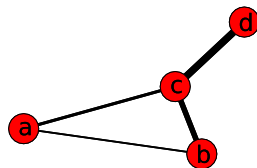
```
13
14 # Add node attributes using add_node(), add_nodes_from() or G.node
15 >>> G.add_node(1, time='5pm')
16 >>> G.node[1]['time']
17 '5pm'
18 >>> G.node[1] # Python dictionary
19 {'time': '5pm'}
20
21 >>> G.add_nodes_from([3], time='2pm') # multiple nodes
22 >>> G.node[1]['room'] = 714 # add new attribute
23
24 >>> G.nodes(data=True)
25 [(1, {'room': 714, 'time': '5pm'}), (3, {'time': '2pm'})]
```

```
27 # Add edge attributes using add_edge(), add_edges_from(),  
28 # subscript notation, or G.edge.  
29 >>> G.add_edge(1, 2, weight=4.0 )  
30 >>> G[1][2]['weight'] = 4.0 # edge already added  
31 >>> G.edge[1][2]['weight'] = 4.0 # edge already added  
32  
33 >>> G[1][2]['weight']  
34 4.0  
35 >>> G[1][2]  
36 {'weight': 4.0}  
37  
38 >>> G.add_edges_from([(3,4),(4,5)], color='red')  
39 >>> G.add_edges_from([(1,2,{'color': 'blue'}), (2,3,{'weight':8})])  
40  
41 >>> G.edges()  
42 [(1, 2), (2, 3), (3, 4), (4, 5)]  
43 >>> G.edges(data=True)  
44 [(1, 2, {'color': 'blue', 'weight': 4.0}), (2, 3, {'weight': 8}), (3,
```

Weighted graph example

The special attribute 'weight' should be numeric and holds values used by algorithms requiring weighted edges.

Use Dijkstra's algorithm to find the shortest path:



```
1 >>> G=nx.Graph()
2 >>> G.add_edge('a','b',weight=0.3)
3 >>> G.add_edge('b','c',weight=0.5)
4 >>> G.add_edge('a','c',weight=2.0)
5 >>> G.add_edge('c','d',weight=1.0)
6 >>> print nx.shortest_path(G,'a','d')
7 ['a', 'c', 'd']
8 >>> print nx.shortest_path(G,'a','d',weighted=True)
9 ['a', 'b', 'c', 'd']
```

Applying classic graph operations

`subgraph(G, nbunch)` - induce subgraph of G on nodes in nbunch

`union(G1,G2)` - graph union

`disjoint_union(G1,G2)` - graph union assuming all nodes are different

`cartesian_product(G1,G2)` - return Cartesian product graph

`compose(G1,G2)` - combine graphs identifying nodes common to both

`complement(G)` - graph complement

`create_empty_copy(G)` - return an empty copy of the same graph class

`convert_to_undirected(G)` - return an undirected representation of G

`convert_to_directed(G)` - return a directed representation of G

Call a graph generator

```
2 # small graphs
3 petersen=nx.petersen_graph()
4 tutte=nx.tutte_graph()
5 maze=nx.sedgewick_maze_graph()
6 tet=nx.tetrahedral_graph()
7
8 # classic graphs
9 K_5=nx.complete_graph(5)
10 K_3_5=nx.complete_bipartite_graph(3,5)
11 barbell=nx.barbell_graph(10,10)
12 lollipop=nx.lollipop_graph(10,20)
13
14 # random graphs
15 er=nx.erdos_renyi_graph(100,0.15)
16 ws=nx.watts_strogatz_graph(30,3,0.1)
17 ba=nx.barabasi_albert_graph(100,5)
18 red=nx.random_lobster(100,0.9,0.9)
```

Read a graph stored in a file using common graph formats.

- edge lists

- adjacency lists

 - GML

 - GraphML

 - Pajek

 - LEDA

```
2 >>> G=nx.Graph()
3 >>> G.add_edges_from([(1,2),(1,3)])
4 >>> G.add_node("spam")
5
6 # Structure of G can be analyzed using various
7 # graph-theoretic functions
8 >>> nx.connected_components(G)
9 [[1, 2, 3], ['spam']]
10
11 # Functions that return node properties return
12 # dictionaries keyed by node label.
13 >>> nx.degree(G)
14 {1: 2, 2: 1, 3: 1, 'spam': 0}
15
16 >>> sorted(nx.degree(G).values())
17 [0, 1, 1, 2]
18
19 >>> nx.clustering(G)
20 {1: 0.0, 2: 0.0, 3: 0.0, 'spam': 0.0}
```