3 - Getting Started with NetworkX

Drew Conway and Aric Hagberg

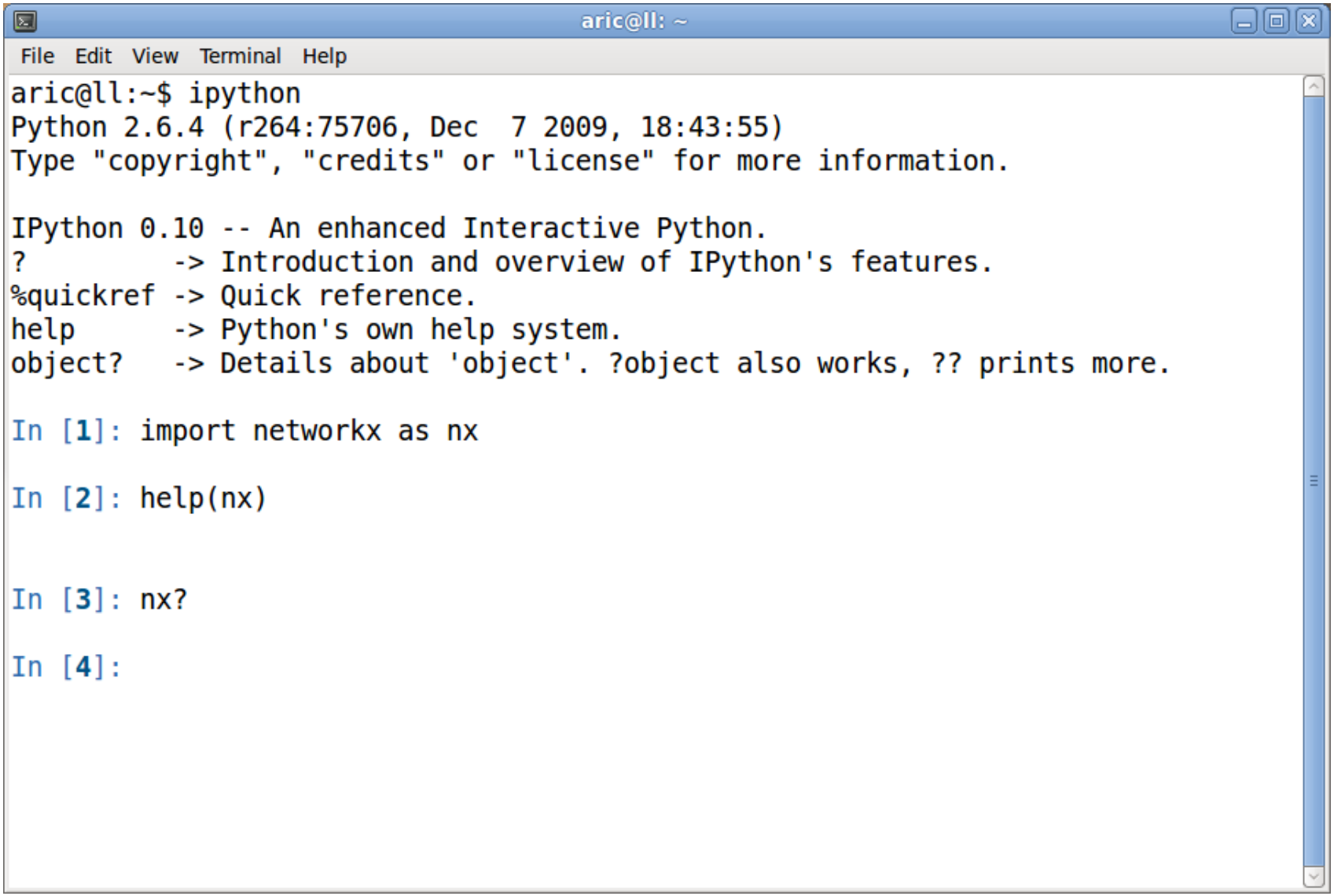
June 29, 2010

Outline

- 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

Running Python and loading NetworkX

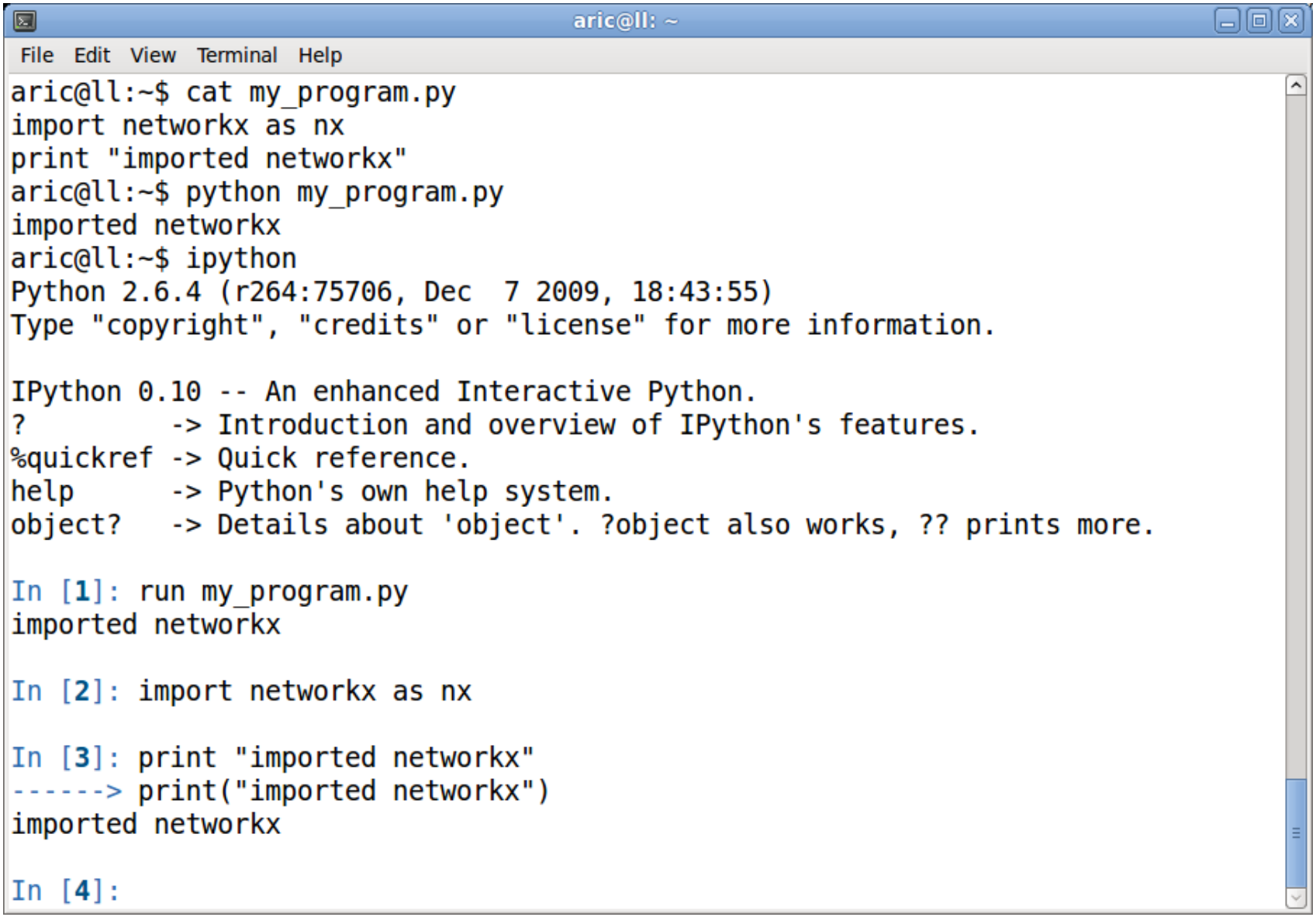
IPython Command line



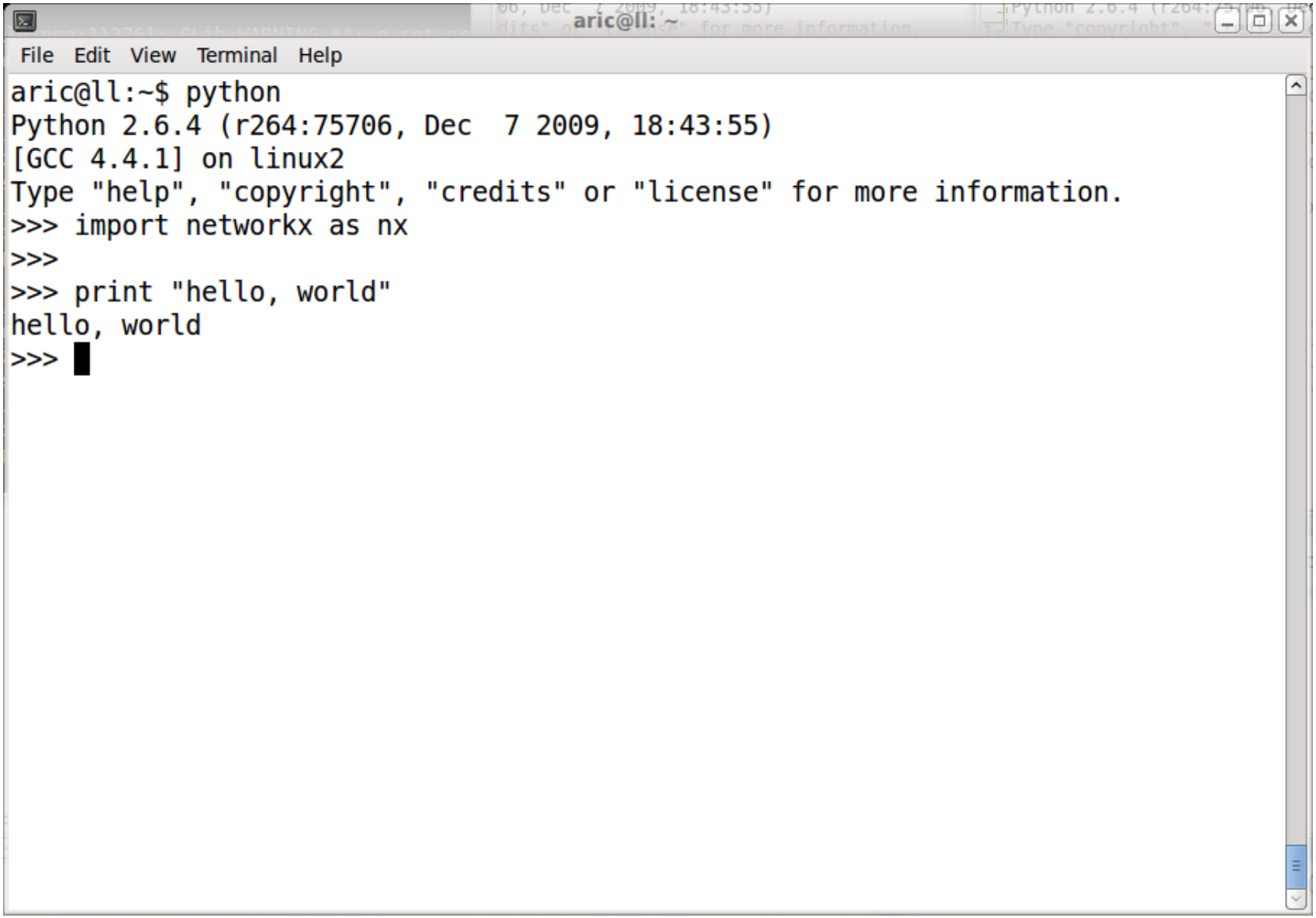
No GUI http://www.cryptonomicon.com/beginning.html

Command line vs executing file

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



The > > > (doctests)



Creating a graph

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

```
>>> import networkx as nx

>>> 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.

Adding nodes

```
# One node at a time
>>> G.add_node(1)  # "method" of G

# A list of nodes
>>> G.add_nodes_from([2,3])

# A container of nodes
>>> H=nx.path_graph(10)
>>> G.add_nodes_from(H) # G now contains the nodes of H

# In contrast, you could use the graph H as a node in G.
>>> 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.

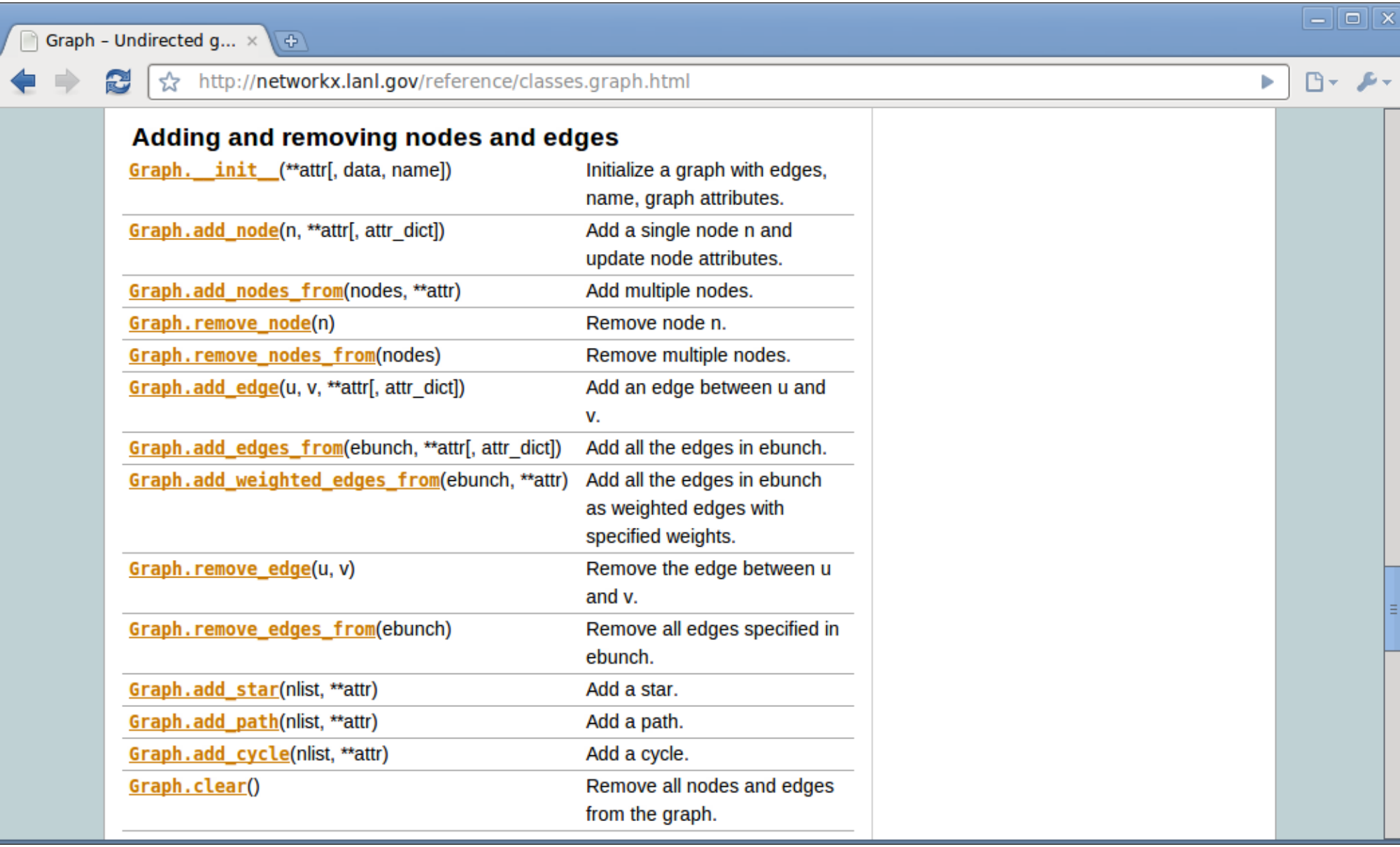
```
# Single edge
>>> G.add_edge(1,2)
>>> e=(2,3)
>>> G.add_edge(*e) # unpack edge tuple*

# List of edges

>>> G.add_edges_from([(1,2),(1,3)])

# Container of edges
>>> 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?

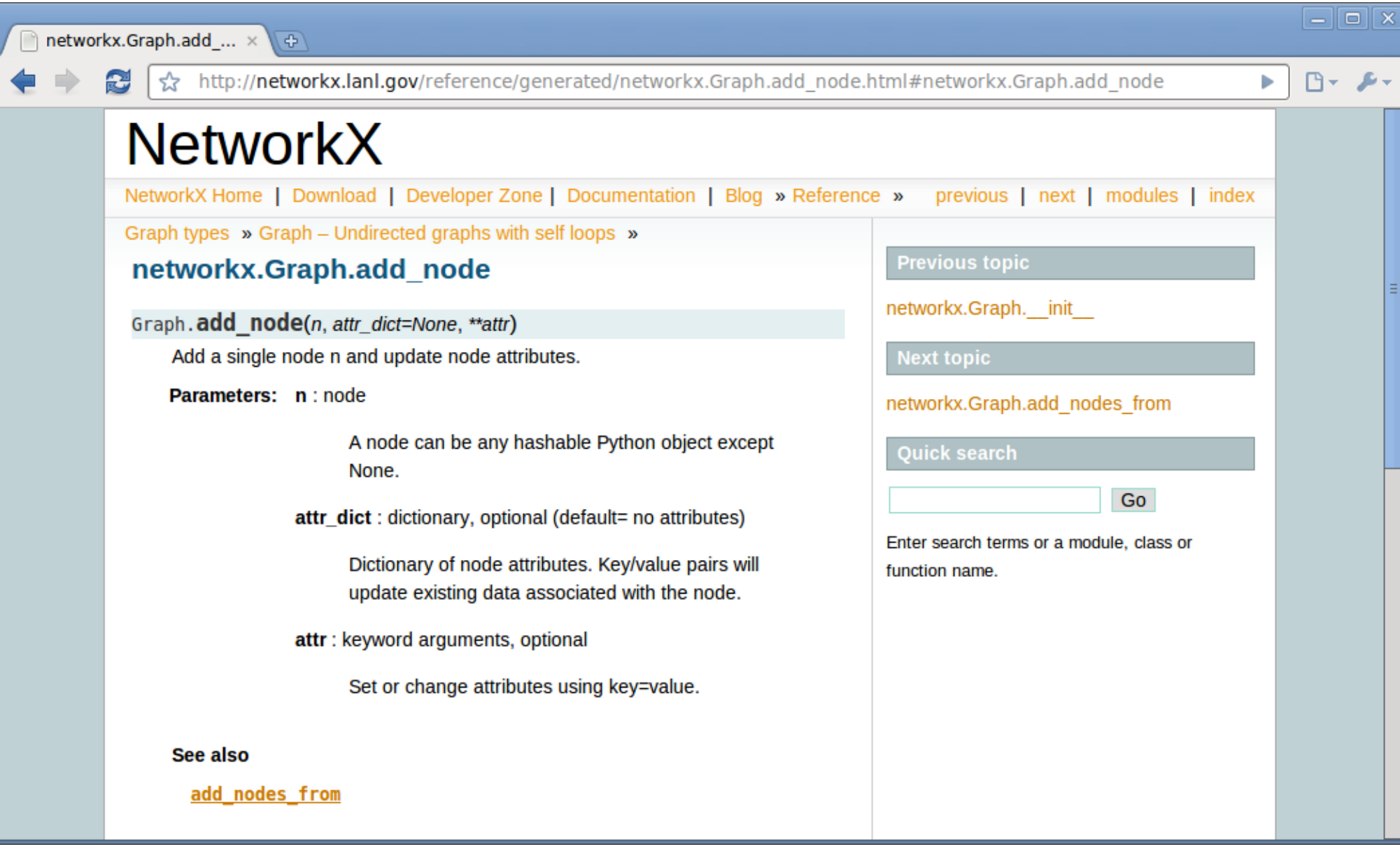
What's in NetworkX?



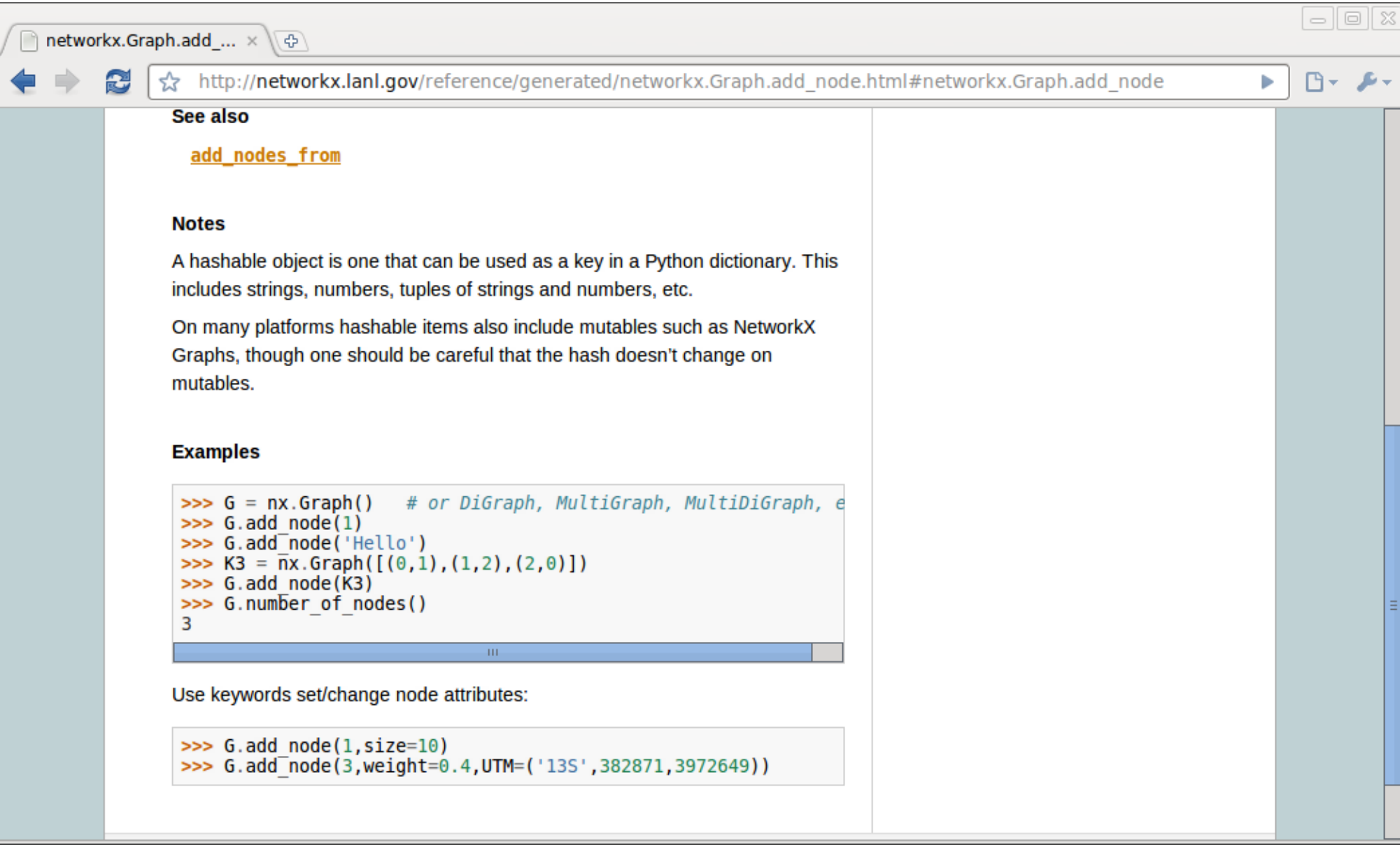
Adding and removing nodes and edges

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

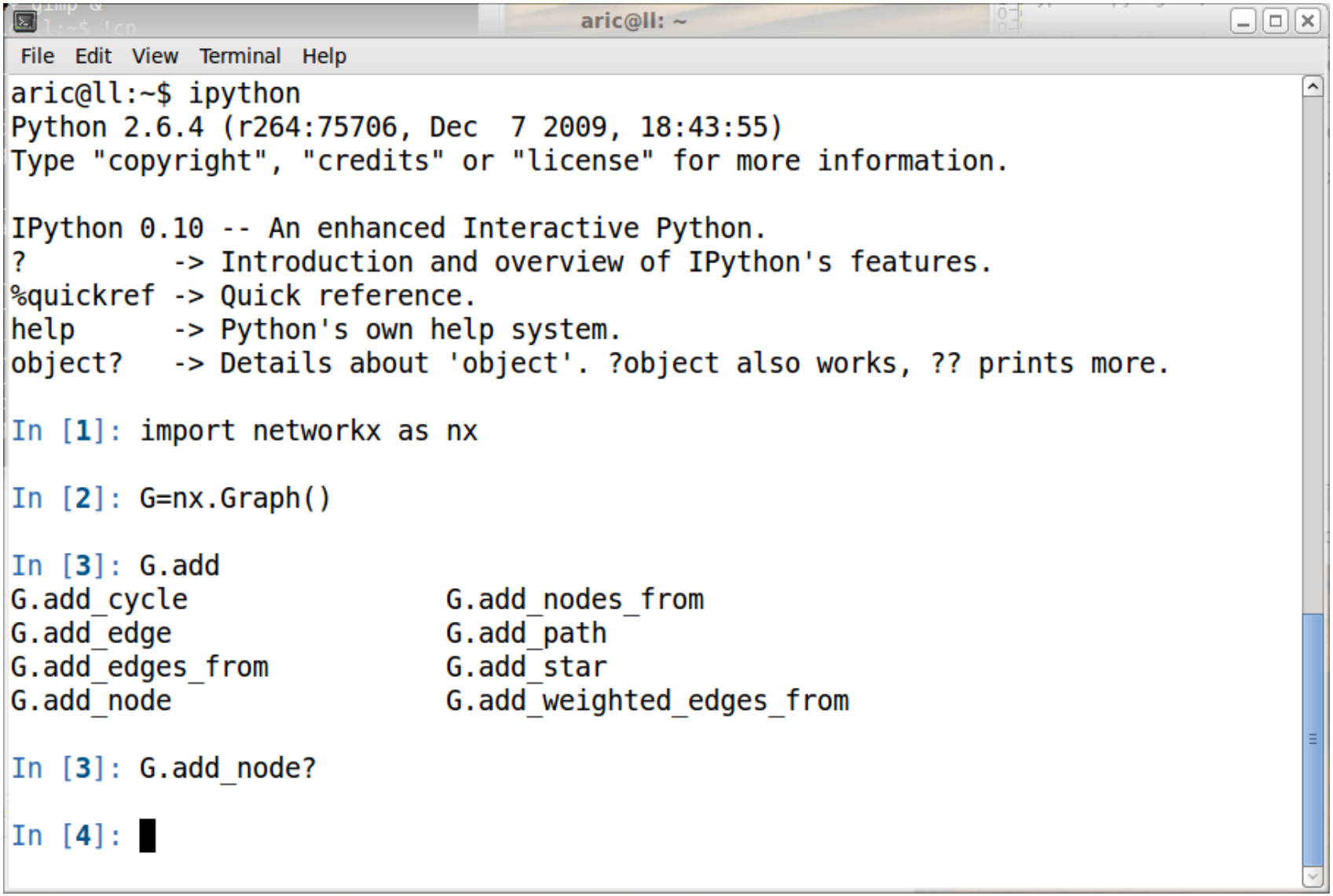
What's in NetworkX?



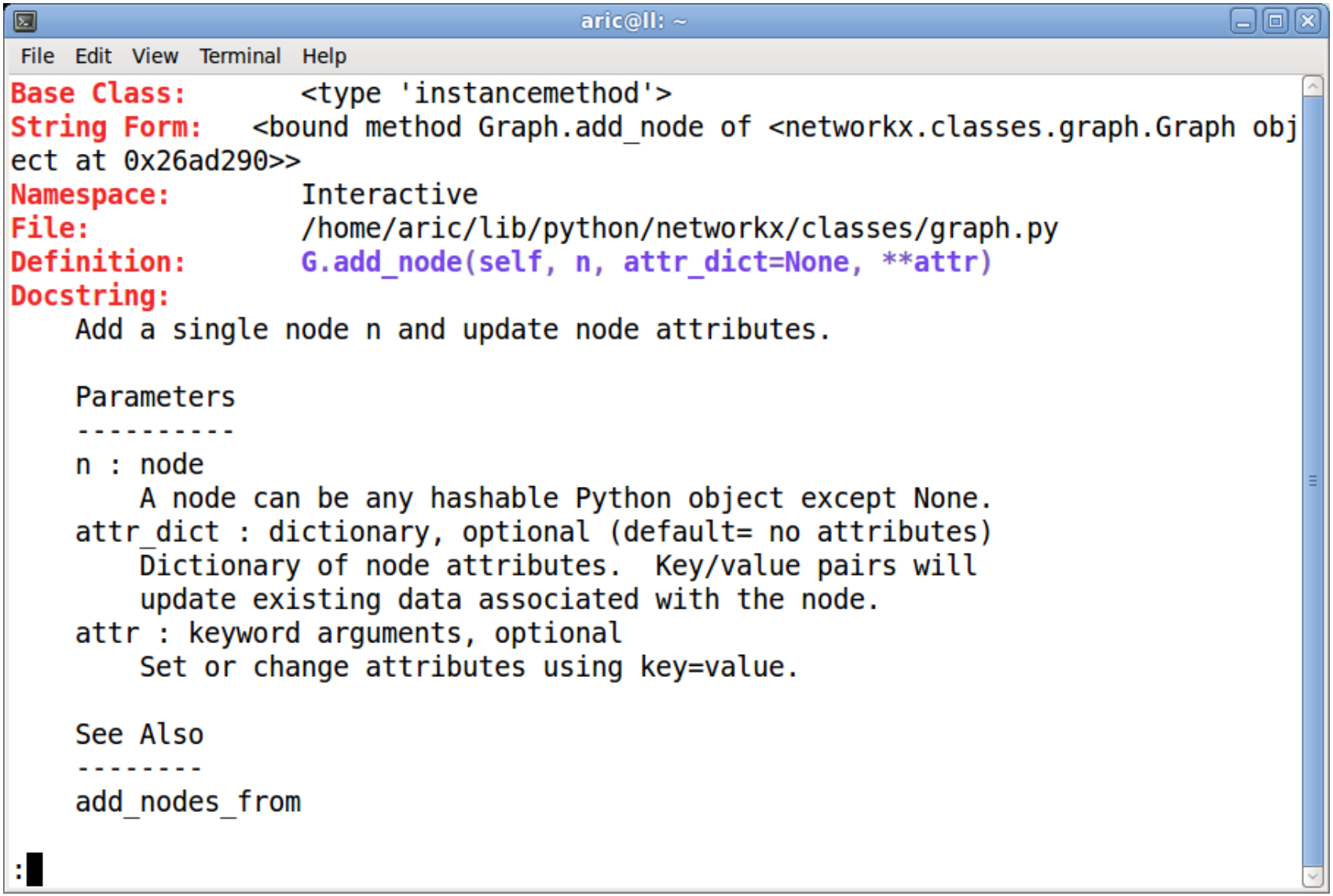
What's in Networkx?



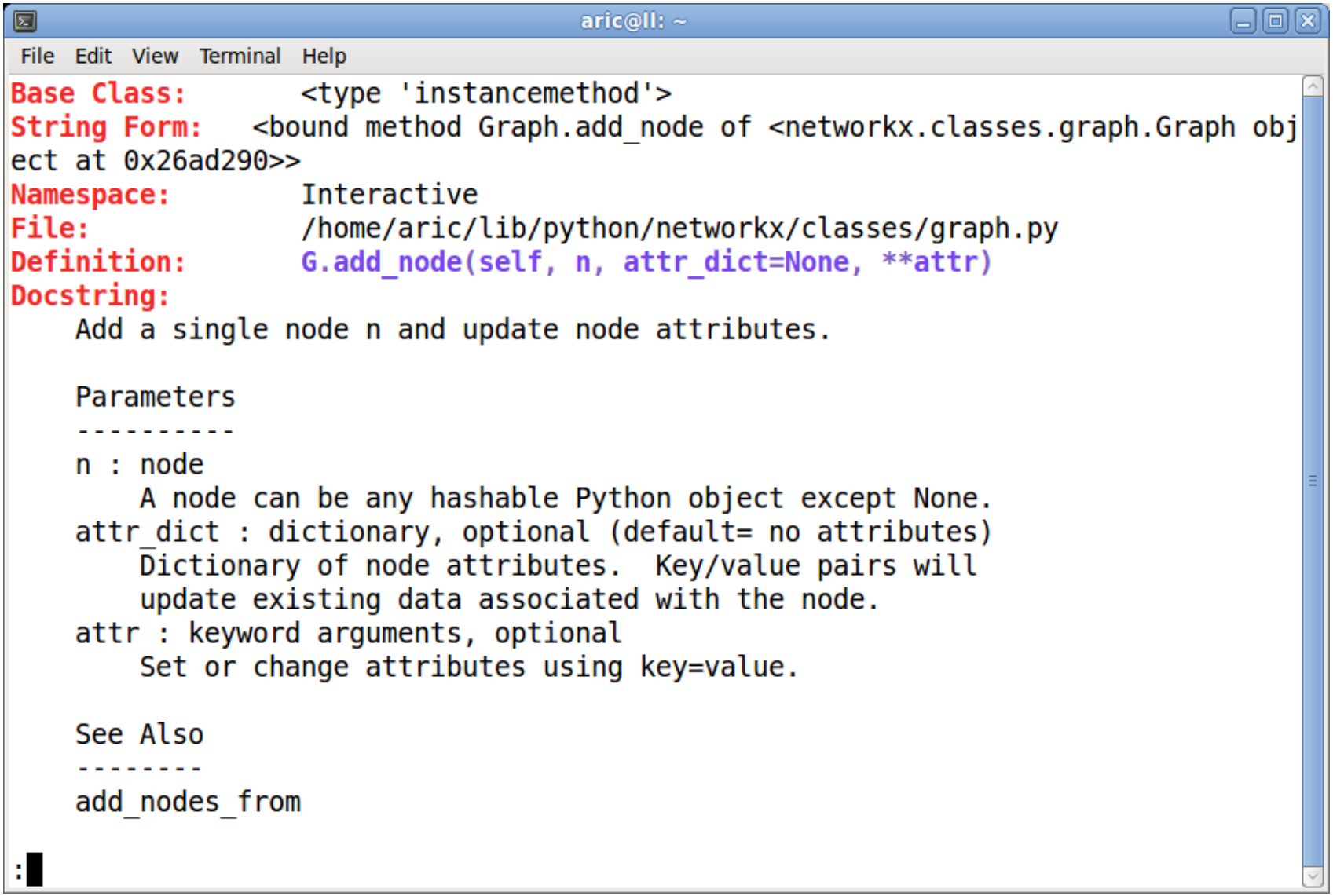
What's in Networkx?



What's in Networkx?



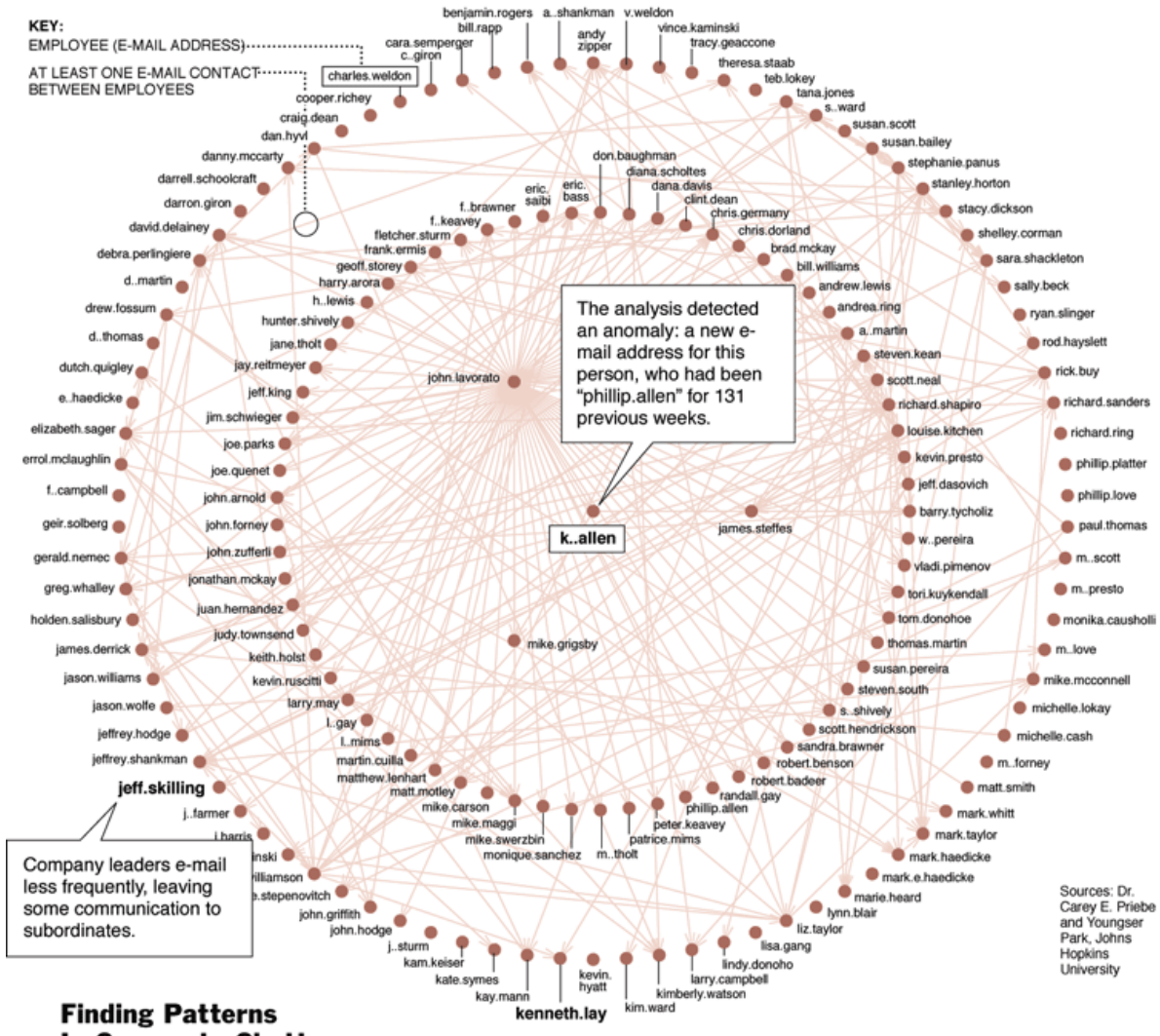
What's in Networkx?



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



Graph attributes

```
>>> import networkx as nx
# Assign graph attributes when creating a new graph

>>> G = nx.Graph(day="Friday")
>>> G.graph
{'day': 'Friday'} # Python dictionary

# Or you can modify attributes later

>>> G.graph['day']='Monday'
>>> G.graph
{'day': 'Monday'}
```

Node attributes

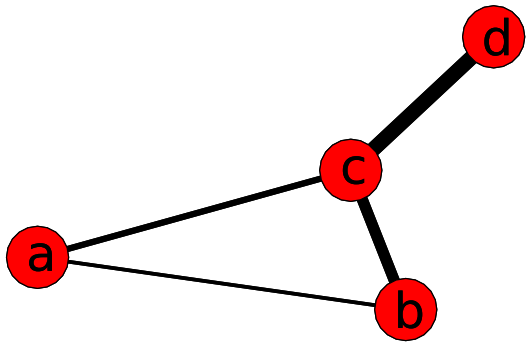
```
13
14 # Add node attributes using add_node(), add_nodes_from() or G.node
15 >>> G.add_node(1, time='5pm')
16 >>> G.node[1]['time']
17  '5pm'
18 >>> G.node[1] # Python dictionray
19 {'time': '5pm'}
20
21 >>> G.add_nodes_from([3], time='2pm') # multiple nodes
22 >>> G.node[1]['room'] = 714 # add new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

Edge attributes

```
# Add edge attributes using add_edge(), add_edges_from(),
# subscript notation, or G.edge.
>>> G.add_edge(1, 2, weight=4.0 )
>>> G[1][2]['weight'] = 4.0 # edge already added
>>> G.edge[1][2]['weight'] = 4.0 # edge already added

>>> G[1][2]['weight']
 4.0
>>> G[1][2]
{'weight': 4.0}

>>> G.add_edges_from([(3,4),(4,5)], color='red')
>>> G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])

>>> G.edges()
[(1, 2), (2, 3), (3, 4), (4, 5)]
>>> G.edges(data=True)
[(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:




```
>>> G=nx.Graph()
>>> G.add_edge('a','b',weight=0.3)
>>> G.add_edge('b','c',weight=0.5)
>>> G.add_edge('a','c',weight=2.0)
>>> G.add_edge('c','d',weight=1.0)
>>> print nx.shortest_path(G,'a','d')
['a', 'c', 'd']
>>> print nx.shortest_path(G,'a','d',weighted=True)
['a', 'b', 'c', 'd']
```

More ways to build graphs: operators and generators

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

```
# small graphs
petersen=nx.petersen_graph()
tutte=nx.tutte_graph()
maze=nx.sedgewick_maze_graph()
tet=nx.tetrahedral_graph()

# classic graphs
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)

# random graphs
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)
```

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

- edge lists
- adjacency lists
- GML
- GraphML
- Pajek
- LEDA

Basic analysis of graphs

```
>>> G=nx.Graph()
>>> G.add_edges_from([(1,2),(1,3)])
>>> G.add_node("spam")

# Structure of G can be analyzed using various
# graph-theoretic functions
>>> nx.connected_components(G)
[[1, 2, 3], ['spam']]

# Functions that return node properties return
# dictionaries keyed by node label.
>>> nx.degree(G)
{1: 2, 2: 1, 3: 1, 'spam': 0}

>>> sorted(nx.degree(G).values())
[0, 1, 1, 2]

>>> nx.clustering(G)
{1: 0.0, 2: 0.0, 3: 0.0, 'spam': 0.0}
```