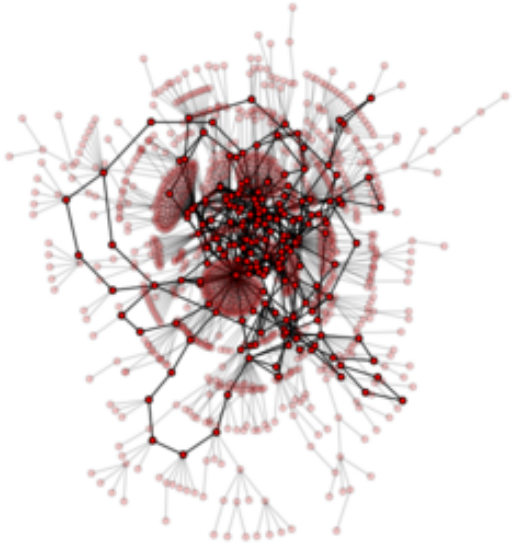
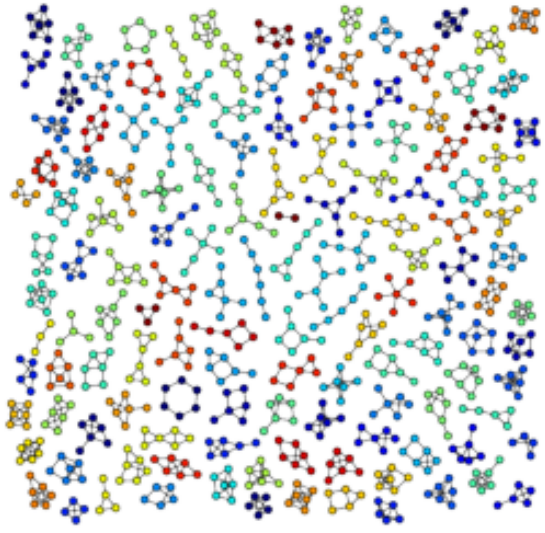
NetworkX in one slide

- Python language package for exploration and analysis of networks and network algorithms
- Data structures for representing many types of networks, or graphs, (simple graphs, directed graphs, and graphs with parallel edges and self loops)
- Nodes can be any (hashable) Python object
- Edges can contain arbitrary data
- Flexibility ideal for representing networks found in many different fields
- Many unit and functional tests
- Online up-to-date documentation





Simple use, adding nodes

Start Python
Import NetworkX using **nx** as a short name

```
>>> import networkx as nx
```

The basic *Graph* class is used to hold the network information. Nodes can be added as follows:

```
>>> G=nx.Graph()
>>> G.add_node(1) # integer
>>> G.add_node('a') # string
>>> print G.nodes()
['a', 1]
```

Graph object at a glance

Graph

```
>>>  G.vertex_count()
>>>  G.vertices()
>>>  G.edge_count()
>>>  G.edges()
>>>  G.get_edge(u, v)
>>>  G.degree(v, out=True)
>>>  G.incident_edges( v, out=True)
>>>  G.insert_vertex(v)
>>>  G.insert_edge(u, v, x=None)
>>>  G.remove_vertex(v)
>>>  G.remove_edge(e)
```

Nodes can be anything

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

```
>>> import math
>>> G.add_node(math.cos) # cosine function
>>> fh=open('tmp.txt','w')
>>> G.add_node(fh) # file handle
>>> print G.nodes()
[<built-in function cos>,
<open file 'tmp.txt', mode 'w' at 0x30dc38>]
>>> G.add_node(1,name = ‘Apple’) #Adding attributes
```

Edges are just pairs of nodes

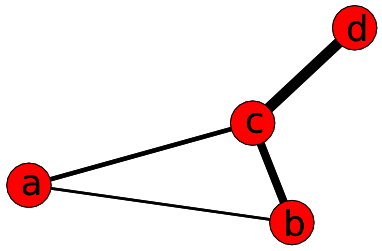
Edges, or links, between nodes are represented as tuples of nodes. They can be added simply

```
>>> G.add_edge(1,'a')
>>> G.add_edge('b',math.cos)
>>> print G.edges()
[('b', <built-in function cos>), ('a', 1)]
>>> G.add_node([1 ,2])
Traceback ( most recent call last ) :
File "<stdin>",line 1,in <module>
File "/usr/lib/pymodules/python2.7/networkx/classes/graph.py",
line 377, in add_node
if n not in self.adj:
TypeError:unhashable type:'list'
```

If the nodes do not already exist they are automatically added to the graph.

Edge can hold arbitrary data

Any Python object is allowed as edge data (e.g. number, string, image, file, ip address)
Edge data assigned and stored in a Python dictionary (default empty).



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']
```

Simple properties

Number of nodes

```
>>> len(G)
>>> G.number_of_nodes()
>>> G.order()
```

Number of edges

```
>>> G.number_of_edges()
```

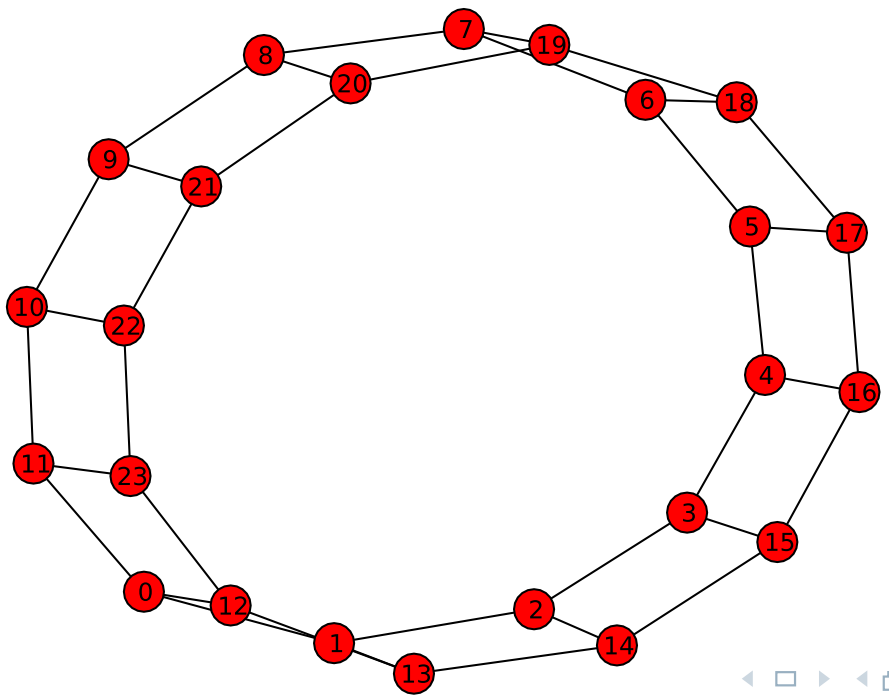
Iterating over edges

```
>>> G = nx.Graph()
>>> G.add_path([0 ,1 ,2 ,3])
>>> [e for e in G.edges_iter()]
[(0 , 1) , (1 , 2) , (2 , 3) ]
```

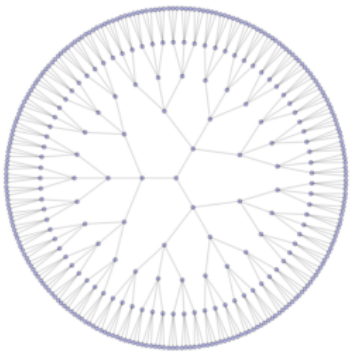
Drawing

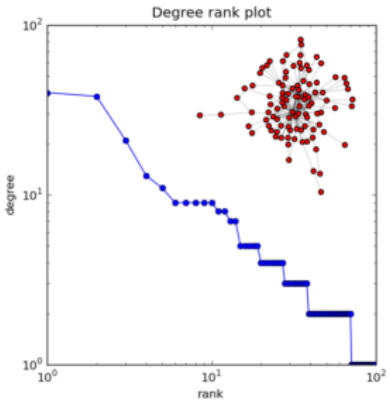
Built-in interface to Matplotlib plotting package
Node positioning algorithms based on force-directed, spectral, and geometric methods

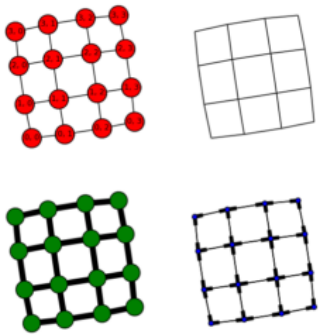
```
>>> G = nx.circular_ladder_graph(12)
>>> nx.draw(G) # Matplotlib under the hood
```

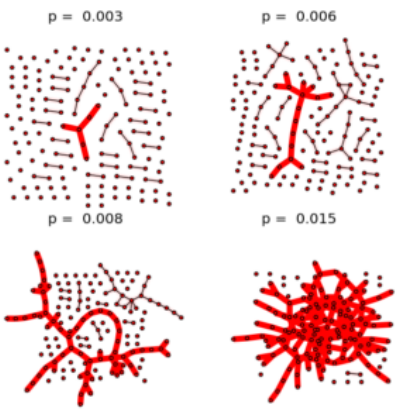


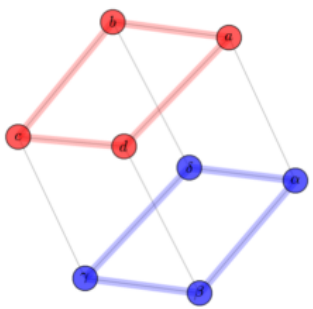
Drawing with Matplotlib

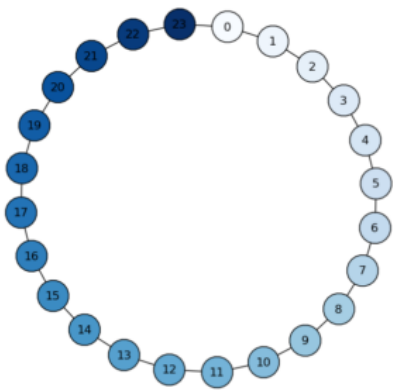


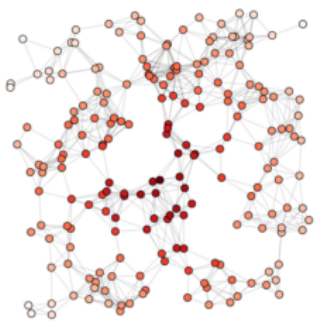












Design decisions

NetworkX defines no custom node objects or edge objects

- *Node-centric* view of network
- Nodes: whatever you put in (hashable)
- Edges: tuples with optional edge data (stored in dictionary)
- Edge data is arbitrary and users can define custom node types

NetworkX is all Python

- Focus on computational network modeling not software tool development
- Move fast to design new algorithms or models

Graph generators

```
>>> nx.complete_graph(5)
>>> nx.complete_bipartite_graph(n1,n2)
>>> nx.barabasi_albert_graph(n,m)
>>> nx.watts_strogatz_graph(n,k,p)
>>> nx.hypercube_graph(n)
>>> nx.lollipop_graph(n)
>>> nx.star_graph(n)
```

Using the help module

```
>>>import networkx as nx
>>>help(nx.algorithms)
```