

Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture – 58
Homophily (Continued) & Positive and Negative Relationships
Schelling Model Implementation – Base Code

Before we start the implementation in the editor I want to show you certain commands in the ipython console, so that it becomes easier for you to understand these commands when I write there in the editor I basically want to show you some intermediate results. So, I will be easier for you to understand. So, let me ask you what is our aim our aim is to how a grid then we have to put people on the cells of the script then we have to identify the nodes which are the people which are unsatisfied and then we have to move these are unsatisfied people to different locations that is what is the cracks. Now what is the first step? The first step is you want to grid, right.

So, how do we get the grid is your any network expansion that can help us to get the grid. So, apparently we have this function grid to be graph which we can make use of although not to suit our complete purpose, but we can make changes into that and then basically we can manipulate that.

(Refer Slide Time: 01:16)

```
anamika@anamika-Inspiron-5423:~$ ipython
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: import networkx as nx

In [2]: N = 10

In [3]: G = nx.grid_2d_graph(N,N)

In [4]: import matplotlib.pyplot as plt

In [5]: nx.draw(G)

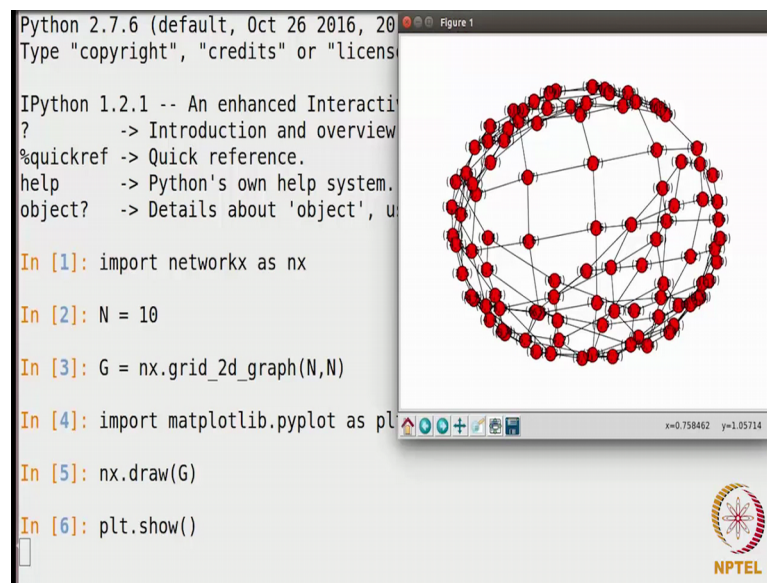
In [6]: plt.show()
```



So, let us get started I will start my ipython console and lets import. So, let us create a 10 cross 10 grid. So, I will take N to be 10 and then I will use the function. So, I will write `nx.grid_2d_graph` and we want to N to be N cross and that is a 10 cross 10 in this case.

So, this is the function that we will use let me show how it looks like. So, I need to show I need to show that. So, I need to import `matplotlib.pyplot` as `plt`. So, I will write `nx.grid_2d_graph` and `plt.show`.

(Refer Slide Time: 02:17)




So, this is how it looks like; obviously, it is not looking like a grid of, but it is a grid although it is not looking like the way we want to show it. So, we might need to make some changes into it let us see what kind of changes we can make here I am going to close the graph is not looking. So, nice because the nodes are not displayed in the form of a grid apparently we do not have an in grid layout a networkx.

So, we might need to manually assign the positions to the grid to the nodes now how can we do that before I tell you that let me show you the nodes of this graph.

(Refer Slide Time: 03:05)

```
In [2]: N = 10
In [3]: G = nx.grid_2d_graph(N,N)
In [4]: import matplotlib.pyplot as plt
In [5]: nx.draw(G)
In [6]: plt.show()
In [7]: G.nodes()
Out[7]:
[(7, 3),
 (6, 9),
 (0, 7),
 (1, 6),
 (3, 7),
 (2, 5),
 (8, 5),
 (5, 8),
 (4, 0),
```




So, I will write `G.nodes()` now this something different here in the other kinds of graphs we get either some number or some use a define string as the label of the nodes, but here see I wrote `G.nodes()` and what I am getting is and getting a tuple for every node and the tuple consist in 2 values which is basically the row number and the column number of that node. So, basically it is like a grid, but it just that is not looking like a grid.

So, we might need to display it accordingly we might need to assign appropriate positioning to every node. So, I will show you how we can do that please note that we are getting both x and y positions which are relative positions of these nodes with respect to each other.

(Refer Slide Time: 03:58)

```
(5, 6),
(7, 7),
(0, 3),
(1, 2),
(4, 9),
(3, 3),
(2, 9),
(8, 1),
(4, 4),
(6, 3),
(0, 0),
(7, 9),
(3, 8),
(2, 0),
(1, 8),
(8, 8),
(4, 3),
(9, 5),
(5, 2)]


In [8]: █
```



(Refer Slide Time: 04:00)

```
(4, 9),
(3, 3),
(2, 9),
(8, 1),
(4, 4),
(6, 3),
(0, 0),
(7, 9),
(3, 8),
(2, 0),
(1, 8),
(8, 8),
(4, 3),
(9, 5),
(5, 2)]

In [8]: pos = dict((n, pos) for n in G.nodes())
In [9]: nx.draw(G, pos)
In [10]: plt.show() █
```



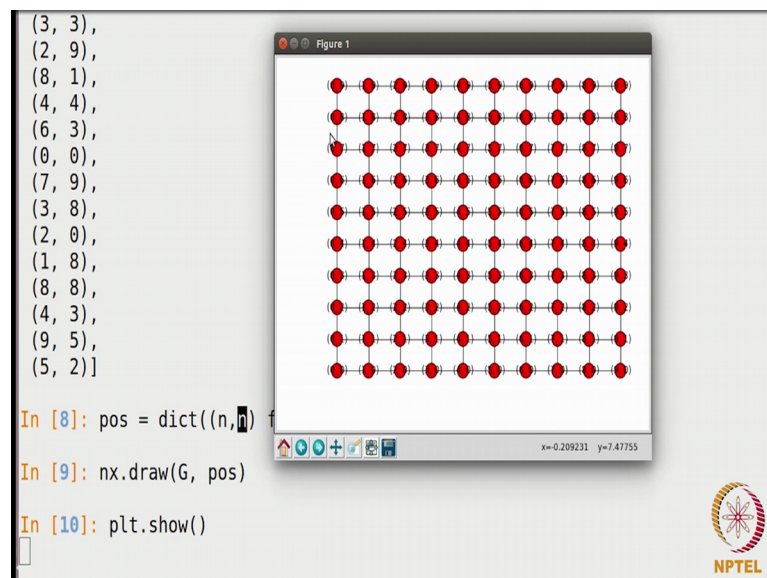
So, what we can do is I will write N comma N for N in G dot nodes or since we have used N capital N already maybe I should use small n . So, what I will write is I will convert this into a dictionary and I will assign that dictionary to a variable say pos .

Let me explain you what I have done. So, these are G dot nodes as you can see please note the pattern of the nodes are we getting here. So, this 8 comma 8 for example, will be N will be 1 N . So, what I am doing here for N in G dot nodes that is for all these tuples I am writing N comma n . So, I am creating tuple where there are 2 N s N comma N . So, it

will be like 8 8 comma 8 8 what I am doing is for node I am fixing its position which is the same as its label, right so, and after that I am converting that into dictionary because that is how we pass the parameter that we pass should be in the form of dictionary. So, this dict function this is the function in python what it does is it converts a tuple into a dictionary where the first value of the tuple will be the key and the second value of the tuple will be the value of the dictionary.

So, this pos this is the positioning that we have fixed and we can assign these positions to nodes and how we can do that is we call this function and it is not draw G we can pass this positioning here and then we can write plt.show.

(Refer Slide Time: 05:47)




Alright, this looks beautiful now listen it right. So, as you can see we are getting the grid and now other thing that we have to node here is that the labels are not in. So, nice one thing that you should note here is that 0 0 is here and the bottom left corner and it goes like this 0 0 0 1 0 2 0 3 and then comes back from 1 0 1 1 and so on. So, this is the default sequencing of the nodes that this function provides that we used what grid 2 d graph if you want you can change this label in such a way that 0 0 pairs here, but I do not think we need that. So, we will let it be like this, but one thing that you would want to do is that we would want to change the label of basically we have 10 cross 10 that is hundred nodes here and it will be nice if we assign the no labels like 0 2 0 1 2 3 up to 99.

So, that is one thing that we would like to do here now let me show you how we can do that again we will assign different label to every node now the name of the node itself carries the row and column number that is what we will be using in order to assign the label. So, the 0 0 should be should be having the label 0 0 1 should be having label 1 and so on 0 1 2 3 4 up to 10; now 0 up to 9 and then 10 should be here than 11 and so on. So, this is how we have to assign the labels how can we do that if you know a little math you should be able to understand what I am doing.

(Refer Slide Time: 07:35)

```
(0, 0),
(7, 9),
(3, 8),
(2, 0),
(1, 8),
(8, 8),
(4, 3),
(9, 5),
(5, 2)]

In [8]: pos = dict((n,n) for n in G.nodes())
In [9]: nx.draw(G, pos)
In [10]: plt.show()
In [11]: labels = dict((i,j), i*10+j) for i,j in G.nodes()
In [12]: nx.draw(G, pos, with_labels = False)
In [13]: plt.show()
```



So, what I will do is i comma j to i comma j what I am assigning is i into 10 plus j this is what I am assigning to first node and to the i j th node this I will do for every node. So, I will write for i j in G dot nodes and again I will convert this into dictionary. So, I will use the same function and let me call this dictionary labels or when they should go.


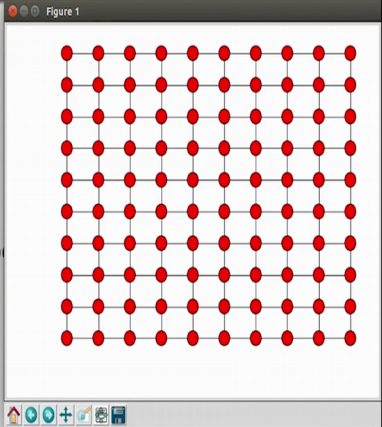
So, what I am doing here is that to every node which is i j like here like 4 3 I am assigning label which is i into 10 plus j in the case 4 comma 3 it will be 43. So, that label I am assigning to every node and that information I am keeping in the form of a dictionary in this dictionary called labels which I am going to use. So, let me show you how we can do that. So, we used nx dot draw a previously which display some labels by default which is the labels that I showed you just now we do not want those labels to be displayed. So, what I will write is with_labels is equal false right. So, the labels that it

will showing it will not be visible you can check that is well if you I can pretty show you.

(Refer Slide Time: 09:02)

```
(7, 9),
(3, 8),
(2, 0),
(1, 8),
(8, 8),
(4, 3),
(9, 5),
(5, 2)]


In [8]: pos = dict((n,n) for n in G.nodes())
In [9]: nx.draw(G, pos)
In [10]: plt.show()
In [11]: labels = dict((i,j), i*10+j)
In [12]: nx.draw(G, pos, with_labels = False)
In [13]: plt.show()
```



(Refer Slide Time: 09:04)

```
(8, 8),
(4, 3),
(9, 5),
(5, 2)]

In [8]: pos = dict((n,n) for n in G.nodes())
In [9]: nx.draw(G, pos)
In [10]: plt.show()
In [11]: labels = dict((i,j), i*10+j) for i,j in G.nodes()
In [12]: nx.draw(G, pos, with_labels = False)
In [13]: plt.show()
In [14]: nx.draw(G, pos, with_labels = False)
In [15]: nx.draw_networkx_labels(G, pos, labels = labels)
```




So, there are no label displayed here; here now we want to display our own labels for that we have a function `nx.draw_networkx_labels`. Now again we have to pos the graph we have to pos the positioning and then we have to pos the labels dictionary labels dictionary.

So, this is how we write labels is it would labels this should work now the labels have the new labels have been assigned that is display them oh, yeah, this looks nice.

(Refer Slide Time: 09:28)

```
(8, 1): <matplotlib.text.Text at 0x7fd4668fd8d0>,  
(8, 2): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 3): <matplotlib.text.Text at 0x7fd465f48a10>,  
(8, 4): <matplotlib.text.Text at 0x7fd466958510>,  
(8, 5): <matplotlib.text.Text at 0x7fd465f44510>,  
(8, 6): <matplotlib.text.Text at 0x7fd465f7cdd0>,  
(8, 7): <matplotlib.text.Text at 0x7fd4660de150>,  
(8, 8): <matplotlib.text.Text at 0x7fd466914b50>,  
(8, 9): <matplotlib.text.Text at 0x7fd4660fcf90>,  
(9, 0): <matplotlib.text.Text at 0x7fd46611c090>,  
(9, 1): <matplotlib.text.Text at 0x7fd466963210>,  
(9, 2): <matplotlib.text.Text at 0x7fd465f967d0>,  
(9, 3): <matplotlib.text.Text at 0x7fd466118d10>,  
(9, 4): <matplotlib.text.Text at 0x7fd465f78410>,  
(9, 5): <matplotlib.text.Text at 0x7fd466921410>,  
(9, 6): <matplotlib.text.Text at 0x7fd4660de9d0>,  
(9, 7): <matplotlib.text.Text at 0x7fd465fa3690>,  
(9, 8): <matplotlib.text.Text at 0x7fd4660b6e90>,  
(9, 9): <matplotlib.text.Text at 0x7fd4660fc2d0>}
```

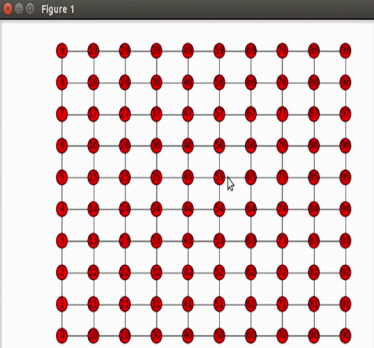
In [16]: plt.show()




Now, So, we have the labels from 0 1 2 3 up to 99. So, it will be nice to refer to them when we do the later manipulations.

(Refer Slide Time: 09:35)

```
(8, 2): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 3): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 4): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 5): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 6): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 7): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 8): <matplotlib.text.Text at 0x7fd466118490>,  
(8, 9): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 0): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 1): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 2): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 3): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 4): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 5): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 6): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 7): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 8): <matplotlib.text.Text at 0x7fd466118490>,  
(9, 9): <matplotlib.text.Text at 0x7fd466118490>}
```



In [16]: plt.show()



Now, one thing that is important to note here is that I told in the explanation of scaling model that every node which is an internal node that is node on the boundary should have 8 neighbours right. Now here if you look at this node number 45 it has 3; 35, 46,

55, 44; it has only 4 neighbours it is not neighbours with the diagonal nodes as you can see. So, this is the kind of grid that the function provides as by default.

But this is not serving our purpose our purpose is that this 45 should be connected to 56 as well as an all these 4 nodes 36, 34, 54 and 56 now how do we do that. So, we would have to add them the edges manually which are not already present here now how can we do that I want to show you.

(Refer Slide Time: 10:40)


```
(8, 8): <matplotlib.text.Text at 0x7fd466914b50>,  
(8, 9): <matplotlib.text.Text at 0x7fd4660fcf90>,  
(9, 0): <matplotlib.text.Text at 0x7fd46611c090>,  
(9, 1): <matplotlib.text.Text at 0x7fd466963210>,  
(9, 2): <matplotlib.text.Text at 0x7fd465f967d0>,  
(9, 3): <matplotlib.text.Text at 0x7fd466118d10>,  
(9, 4): <matplotlib.text.Text at 0x7fd465f78410>,  
(9, 5): <matplotlib.text.Text at 0x7fd466921410>,  
(9, 6): <matplotlib.text.Text at 0x7fd4660de9d0>,  
(9, 7): <matplotlib.text.Text at 0x7fd465fa3690>,  
(9, 8): <matplotlib.text.Text at 0x7fd4660b6e90>,  
(9, 9): <matplotlib.text.Text at 0x7fd4660fc2d0>}
```

```
In [16]: plt.show()
```

```
In [17]: nx.draw_networkx_labels(G, pos, labels = labels)  
%%latex labels labels= lambda
```

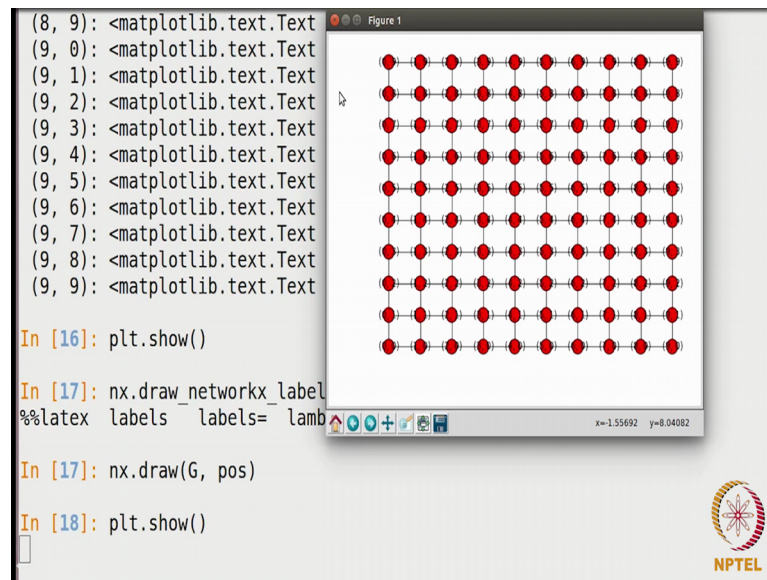
```
In [17]: nx.draw(G, pos)
```

```
In [18]: plt.show()
```



The previous representation of the graph because that will help us understand how we can add the edges, so, what I am going to do is I am not going to use this command `nx dot draw` the usual function that we will using and this position we were using I would like to display this only.

(Refer Slide Time: 11:06)



I am showing you the previous version of the graph for a reason because I want to show you how can are the edges with diagonal edges now if you look at this 1 1 node we have to connected to 2 2 this 2 2 has to be connected to 3 3 and if you look at some other node this 2 6 has to be connected to 3 7 4 8 3 7 has to be connected to 4 8.

So, if you observe carefully the node $i j$ has to be connected to $i + 1 j + 1$ if you look at 6 7 it has to be connected to 7 8 which is again $i + 1 j + 1$. So, we will take all the nodes $i j$ and we will connect them to the nodes $i + 1 j + 1$ that should be half of the work done. So, let us see one thing that we have to node here again is that when we are doing $i + 1 j + 1$ it should never cross the 9 right we cannot connect 9 9 to 10 10. So, that limit we have to take care of, so that we will take care of that thing.

(Refer Slide Time: 12:15)

```
(9, 7): <matplotlib.text.Text at 0x7fd465fa3690>,
(9, 8): <matplotlib.text.Text at 0x7fd4660b6e90>,
(9, 9): <matplotlib.text.Text at 0x7fd4660fc2d0>}

In [16]: plt.show()

In [17]: nx.draw_networkx_labels(G, pos, labels = labels)
%%latex labels labels= lambda


In [17]: nx.draw(G, pos)

In [18]: plt.show()

In [19]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v+1 <= N-1):
.....:         G.add_edge((u,v),(u+1,v+1))
.....:

In [20]: nx.draw(G, pos)

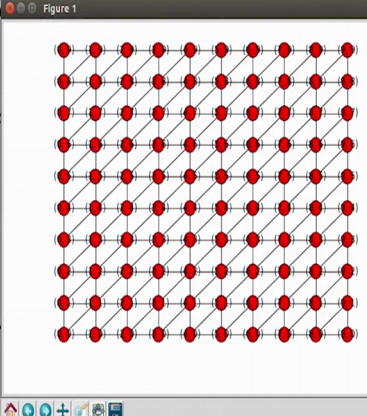
In [21]: nx.draw(G, pos)
```



So, I will write for u comma v let say for u, v in G dot nodes what do we want to do u, v has to be connected to $u + 1, v + 1$, but again we have to take care that $u + 1$ and $v + 1$ does not exist does not exceed $N - 1$. So, I will write if $u + 1$ is less than equal to $N - 1$ and we say to be done for v . So, I will write $v + 1$ is less than equal to $N - 1$ then only we will do this otherwise you will not add that edge. So, I will write G dot add edge which edge u, v edge to $u + 1, v + 1$ edge this should work let us see let me show you in the graph at this stage I will write `nx.draw` and then (Refer Time: 13:16) `plt.show`, yeah.

(Refer Slide Time: 13:16)

```
(9, 9): <matplotlib.text.Text at 0x7f... Figure 1
```



```
In [16]: plt.show()

In [17]: nx.draw_networkx_labels(G, pos, labels = labels)
%%latex labels labels= lambda


In [17]: nx.draw(G, pos)

In [18]: plt.show()

In [19]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v+1 <= N-1):
.....:         G.add_edge((u,v),(u+1,v+1))
.....:

In [20]: nx.draw(G, pos)

In [21]: plt.show()
```




So, we have 1 p of half of the diagonal edges in corporate already the other direction of the diagonal edges are still yet to be added for example, we have to connect this 3 9 2 4 8 4 8 to 5 7. So, that has to be done.

Now, if you a observe carefully 4 6 has to be connected to 5 5 which means $u v$ has to be connected to $u + 1 v - 1$ yeah that that wholes true for every node. Now again one thing that we have to take care of here is that while doing $v - 1$ it should not we come less than 0, so that part we will take care of let see how we can do that.

(Refer Slide Time: 14:01)

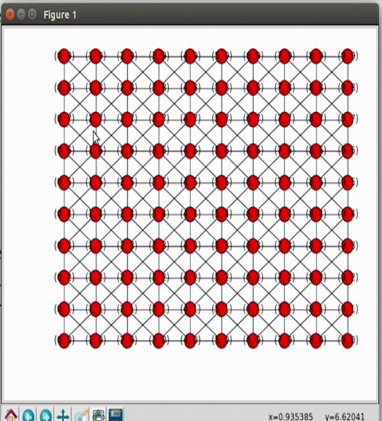
```
In [18]: plt.show()
In [19]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v+1 <= N-1):
.....:         G.add_edge((u,v),(u+1,v+1))
.....:
In [20]: nx.draw(G, pos)
In [21]: plt.show()
In [22]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v-1 >= 0):
.....:         G.add_edge((u,v),(u+1,v-1))
.....:
In [23]: nx.draw(G, pos)
In [24]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v-1 >= 0):
.....:         G.add_edge((u,v),(u+1,v-1))
```



I am going to make changes here itself or yeah. So, what I will do is I will go back here and if $u + 1$. So, $u v$ has to be connected to $u + 1 v - 1$. So, $u + 1$ should be this and $v - 1$ should be less than equal to now should be greater than equal to 0 it should not be less than equal to. So, this we have to check and if this is true than we will have N h from $u v$ to $u + 1 v - 1$; they should work.

(Refer Slide Time: 14:46)

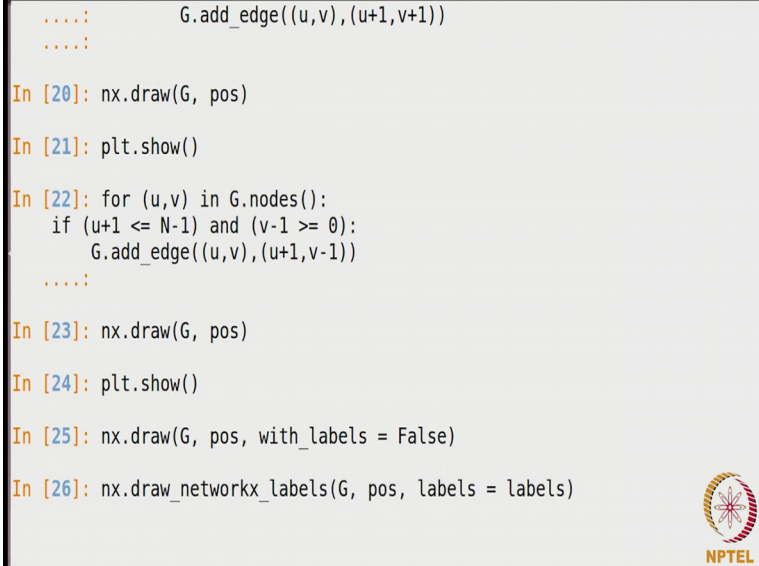
```
In [18]: plt.show()
In [19]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1)
.....:         G.add_edge(u,v,(u+1,v))
.....:
In [20]: nx.draw(G, pos)
In [21]: plt.show()
In [22]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v-1 >= 0)
.....:         G.add_edge((u,v),(u+1,v-1))
.....:
In [23]: nx.draw(G, pos)
In [24]: plt.show()
```



Doubtful about the in indentation, but let me see if there is work if it works fine, yeah. So, here you see all the edges have been incorporated all the diagonal edges are there now the graph seems complete.

(Refer Slide Time: 14:58)

```
.....:         G.add_edge((u,v),(u+1,v+1))
.....:
In [20]: nx.draw(G, pos)
In [21]: plt.show()
In [22]: for (u,v) in G.nodes():
.....:     if (u+1 <= N-1) and (v-1 >= 0):
.....:         G.add_edge((u,v),(u+1,v-1))
.....:
In [23]: nx.draw(G, pos)
In [24]: plt.show()
In [25]: nx.draw(G, pos, with_labels = False)
In [26]: nx.draw_networkx_labels(G, pos, labels = labels)
```




Let me just show you the labels they updated labels that we recently showed which command it will be use yeah we use this and then we used this one yeah let us see.

(Refer Slide Time: 15:11)

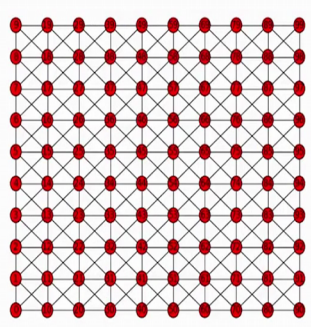
```
(8, 1): <matplotlib.text.Text at 0x7fd465d03f90>,  
(8, 2): <matplotlib.text.Text at 0x7fd465c6db50>,  
(8, 3): <matplotlib.text.Text at 0x7fd465dbd610>,  
(8, 4): <matplotlib.text.Text at 0x7fd465d5ebd0>,  
(8, 5): <matplotlib.text.Text at 0x7fd465c4abd0>,  
(8, 6): <matplotlib.text.Text at 0x7fd465c72250>,  
(8, 7): <matplotlib.text.Text at 0x7fd465d3b810>,  
(8, 8): <matplotlib.text.Text at 0x7fd465d27250>,  
(8, 9): <matplotlib.text.Text at 0x7fd465ed0690>,  
(9, 0): <matplotlib.text.Text at 0x7fd465c568d0>,  
(9, 1): <matplotlib.text.Text at 0x7fd465d6c8d0>,  
(9, 2): <matplotlib.text.Text at 0x7fd465dbde90>,  
(9, 3): <matplotlib.text.Text at 0x7fd465eac410>,  
(9, 4): <matplotlib.text.Text at 0x7fd465ed0ad0>,  
(9, 5): <matplotlib.text.Text at 0x7fd465d27ad0>,  
(9, 6): <matplotlib.text.Text at 0x7fd465d460d0>,  
(9, 7): <matplotlib.text.Text at 0x7fd465c7a7d0>,  
(9, 8): <matplotlib.text.Text at 0x7fd465de2590>,  
(9, 9): <matplotlib.text.Text at 0x7fd465ed9990>}
```

In [27]: █




(Refer Slide Time: 15:14)

```
(8, 2): <matplotlib.text.Text at 0x7f... Figure 1  
(8, 3): <matplotlib.text.Text at 0x7f...  
(8, 4): <matplotlib.text.Text at 0x7f...  
(8, 5): <matplotlib.text.Text at 0x7f...  
(8, 6): <matplotlib.text.Text at 0x7f...  
(8, 7): <matplotlib.text.Text at 0x7f...  
(8, 8): <matplotlib.text.Text at 0x7f...  
(8, 9): <matplotlib.text.Text at 0x7f...  
(9, 0): <matplotlib.text.Text at 0x7f...  
(9, 1): <matplotlib.text.Text at 0x7f...  
(9, 2): <matplotlib.text.Text at 0x7f...  
(9, 3): <matplotlib.text.Text at 0x7f...  
(9, 4): <matplotlib.text.Text at 0x7f...  
(9, 5): <matplotlib.text.Text at 0x7f...  
(9, 6): <matplotlib.text.Text at 0x7f...  
(9, 7): <matplotlib.text.Text at 0x7f...  
(9, 8): <matplotlib.text.Text at 0x7fd465de2590>,  
(9, 9): <matplotlib.text.Text at 0x7fd465ed9990>}
```



In [27]: plt.show()



This looks nice. So, we have the great we have the diagonal edges we have the nodes now we have said the playground for playing with these nodes where we will put the people of different types there will be some people of type 0, there will be some people of type 1; there will be some nodes its will be empty we will not put any people there. So, that is how we are going start in the next video I will be using all these commands that I would show shown you already in the ipython console. So, I will use them in the editor and then we will get started with manipulating the nodes.