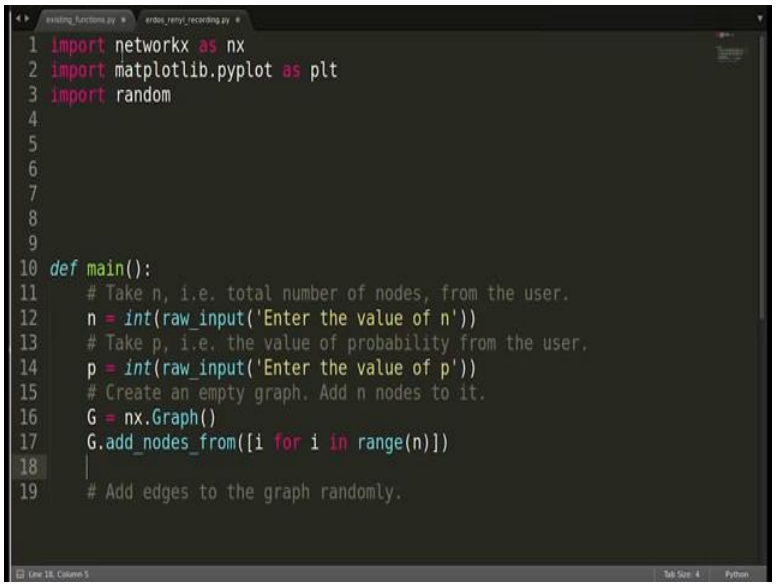
Social Networks

Prof. S. R. S. Iyengar
Prof. Anamika Chhabra
Department of Computer Science
Indian Institute of Technology, Ropar

Rich Get Richer Phenomenon
Lecture - 124
Implementing a Random Graph (Erdos- Renyi Model)-2

In the previous video we discussed Erdos Renyi model to create Random networks. We also discussed the sequence of steps that we will be following for the implementation. In this video we will do the implementation.

(Refer Slide Time: 00:21)



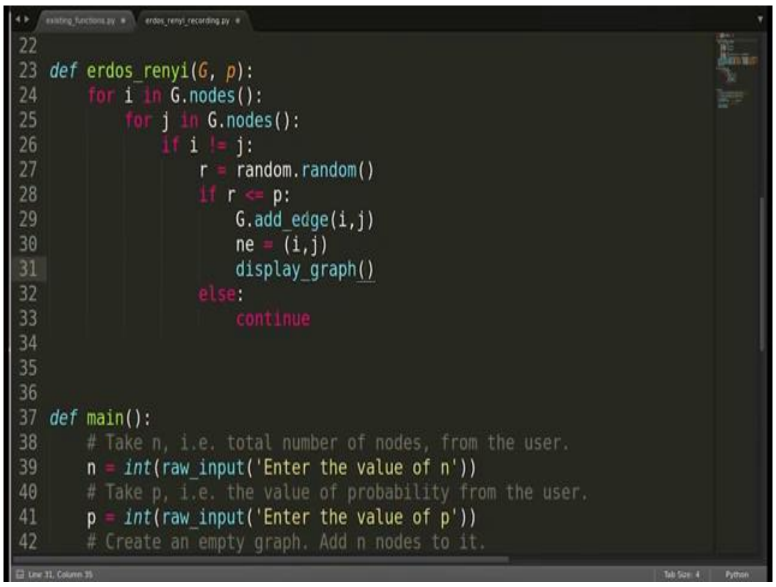
```
import networkx as nx
import matplotlib.pyplot as plt
import random






def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
    # Take p, i.e. the value of probability from the user.
    p = int(raw_input('Enter the value of p'))
    # Create an empty graph. Add n nodes to it.
    G = nx.Graph()
    G.add_nodes_from([i for i in range(n)])

    # Add edges to the graph randomly.
```

I have pasted the sequence of steps that we discussed in the previous video over here and we will start implementing them. To start with let me import the packages that we are going to need. We will be needing networkx, I am sorry we need to display the graph, so we will need matplotlib. We also need random package ok. Let us start the main. Let us take these steps 1 by 1. So, we have to take the value of n from the user and also we have to take the value of p that is probability from the user.

We can use the function draw input for that. So, we can write n is equal to draw input returns in string format. So, we need to convert it to int. Similarly, we will take the value of probability ok. The next step is to create an empty graph and add n nodes to it. Will create an empty graph, we have to now add nodes to it n nodes to it, we will use the function G.add_nodes from here we can pass a list. So, we must add n nodes, what we can do is will write i from i for i in range n is the total number of nodes.

So, 0 to n - 1 nodes will be added to the graph. At this point we can also display the graph we had already created this (Refer Time: 02:49) graph function in the previous video. So, we are going to reuse it. Let me copy paste that function from the previous videos content and we will use it as it is.

(Refer Slide Time: 03:05)



```python

def erdos_renyi(G, p):
    for i in G.nodes():
        for j in G.nodes():
            if i != j:
                r = random.random()
                if r <= p:
                    G.add_edge(i,j)
                    ne = (i,j)
                    display_graph()
                else:
                    continue



def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
    # Take p, i.e. the value of probability from the user.
    p = int(raw_input('Enter the value of p'))
    # Create an empty graph. Add n nodes to it.
```

So, initially when we display the graph it be all empty only the nodes will be there. The next step is to add the edges into this graph which only has nodes as of now, we will create a function for that.

We have to add the edges randomly as per Erdos Renyi model. So, let us create that function. Will pass the graph and the other thing that must be passed is p. So, in Erdos Renyi basically we pass 2 parameters n and p. We pass n here or we can compute the number of nodes from the graph itself. So, I am going to pass only these 2 parameters.

Let us create this function, now, if you remember from the previous video, we have to add the edges randomly. So, basically, we will take one pair of nodes and based on the probability p we will add an edge, or we will skip it.

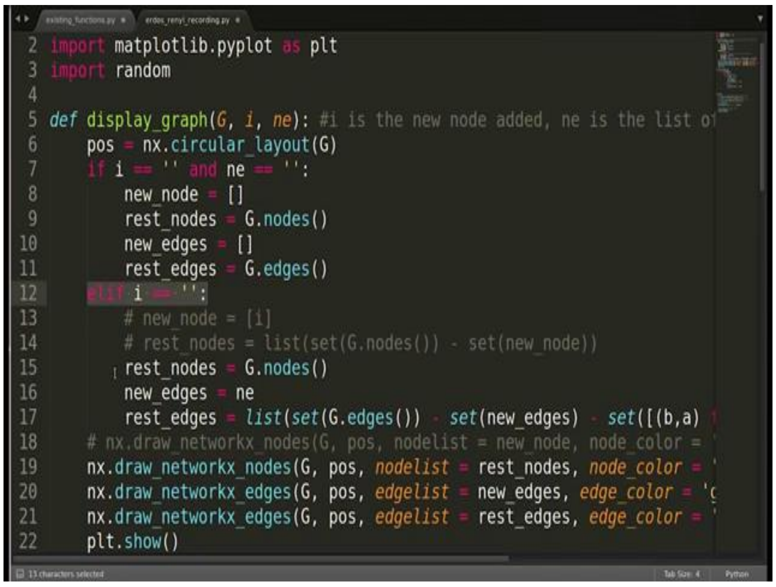
So, let us take all pair of nodes, how can we do that? We will start we will use 2 for loops write for i in G.nodes. We got all the nodes here as the first element in the pair and we will write for j in G.nodes. We do not have to add the edges to from we do not have to add the self edges from one node to itself.

So, we will put a condition here, if i != j only then we will add the edge. We have to take a random number here and will compare that random number with probability p. So, we will take r = random.random, which will return us a value between 0 and 1.

So, if r <= p, we will add that edge. So, we will write G.add_edge. The edge is going to be i comma j and in case r is greater than p we will not add that edge. So, we will just write continue. So, these are the main things to add the edges. Now we also want to display the graph and we also want to keep track of the edges which are getting added at each iteration. So, let me do one more thing here.

So, we will keep track of the edges which are being added into this variable i comma j ok. And then I would like to display the graph, so let me call that function display graph which we have already added. Display graph has two parameter 3 parameters the graph as you can see here, the graph and the node which is being added and the edge which is being added.

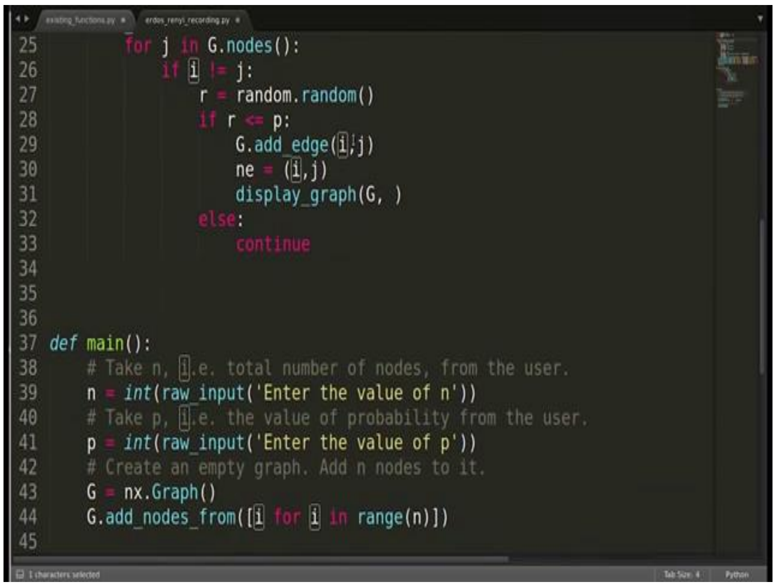
(Refer Slide Time: 05:58)



```
import matplotlib.pyplot as plt
import random

def display_graph(G, i, ne): #i is the new node added, ne is the list of
    pos = nx.circular_layout(G)
    if i == '' and ne == '':
        new_node = []
        rest_nodes = G.nodes()
        new_edges = []
        rest_edges = G.edges()
    elif i == '':
        # new_node = [i]
        # rest_nodes = list(set(G.nodes()) - set(new_node))
        rest_nodes = G.nodes()
        new_edges = ne
        rest_edges = list(set(G.edges()) - set(new_edges) - set([(b,a)
    # nx.draw_networkx_nodes(G, pos, nodelist = new_node, node_color =
    nx.draw_networkx_nodes(G, pos, nodelist = rest_nodes, node_color = '
    nx.draw_networkx_edges(G, pos, edgelist = new_edges, edge_color = 'g
    nx.draw_networkx_edges(G, pos, edgelist = rest_edges, edge_color = '
    plt.show()
```

So, here nodes are not being added because, all the nodes are added to the graph initially only the edges are getting added at each iteration. So, if you note here and if you remember ne is a list of edges right and i is a single node.

(Refer Slide Time: 06:24)

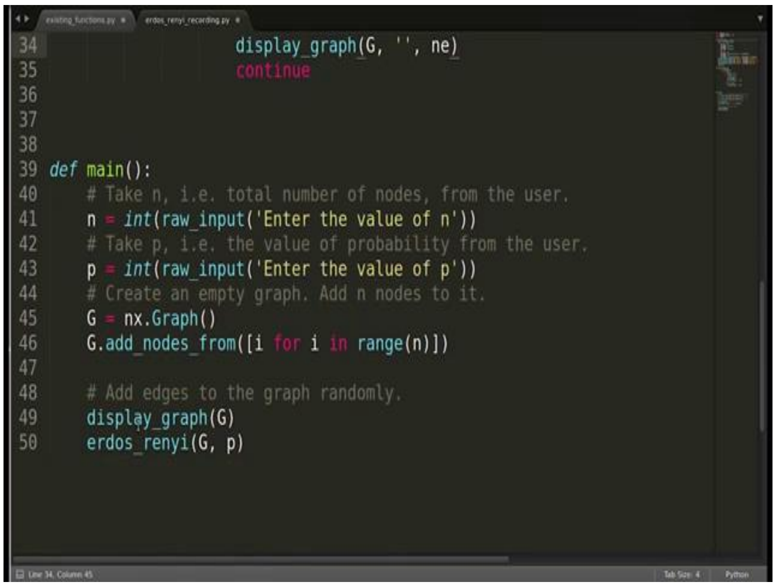


```
        for j in G.nodes():
            if i != j:
                r = random.random()
                if r <= p:
                    G.add_edge(i,j)
                    ne = (i,j)
                    display_graph(G, )
                else:
                    continue



def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
    # Take p, i.e. the value of probability from the user.
    p = int(raw_input('Enter the value of p'))
    # Create an empty graph. Add n nodes to it.
    G = nx.Graph()
    G.add_nodes_from([i for i in range(n)])
```

So, we have to keep we have to take care of that while we are passing the things here. Now we do not want to change the colour of the nodes, we will just pass an empty string here and we have to pass the edge which is being added in this iteration as a list. So, let me convert this into list and will what I will be passing.

And in case we are not adding any edge we want to display the graph at that moment as well because, we want to see whether in this particular iteration any edge was added or not, we want to display the graph there also. So, let me copy this in that case there will not be any new edge. So, we will keep this empty and everything else should remain same. Now let us go back to main and see if we can start executing this code.

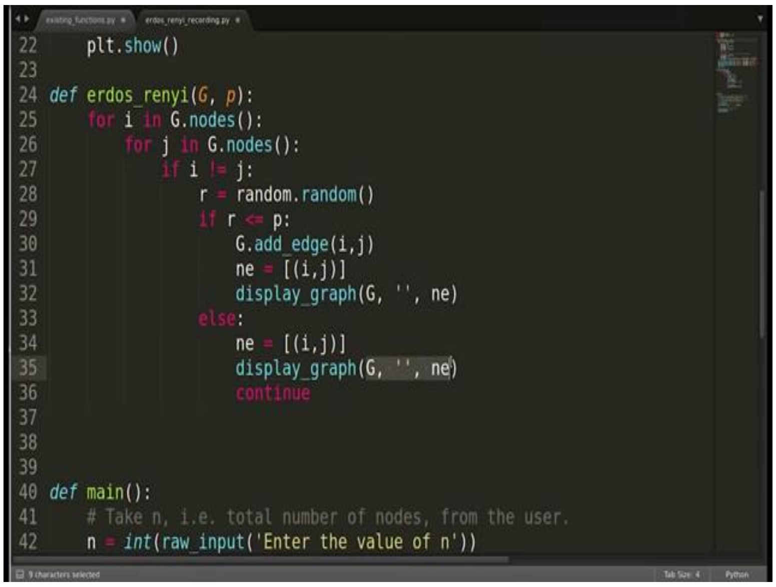
(Refer Slide Time: 07:19)



```
                display_graph(G, '', ne)
                continue



def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
    # Take p, i.e. the value of probability from the user.
    p = int(raw_input('Enter the value of p'))
    # Create an empty graph. Add n nodes to it.
    G = nx.Graph()
    G.add_nodes_from([i for i in range(n)])

    # Add edges to the graph randomly.
    display_graph(G)
    erdos_renyi(G, p)
```

We are calling this function Erdos Renyi, we are calling display graph. There are 3 parameters to this function and in main we are passing only 1 parameter. So, let me pass the rest 2 parameters. So, this is the initial displaying of the graph where there are no new nodes, no new edges. Let us pass the empty strings there. So, the initial graph should be created.

(Refer Slide Time: 07:46)



```
    plt.show()

def erdos_renyi(G, p):
    for i in G.nodes():
        for j in G.nodes():
            if i != j:
                r = random.random()
                if r <= p:
                    G.add_edge(i,j)
                    ne = [(i,j)]
                    display_graph(G, '', ne)
                else:
                    ne = [(i,j)]
                    display_graph(G, '', ne)
                    continue



def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
```

And when we are calling display graph from this function, we are passing G comma, we passing G and we passing empty string and a list ok. Let us see whether this function were surprise for that. So, the initial condition this condition cannot be used because, both of these are not empty and if you go to the else part or i is empty here.

So, we see that we need to make some changes into this function in order to be able to use this for our set of conditions in this particular model. So, maybe we can write elif i = empty because this is the case that will be followed when we call display graph from Erdos Renyi function. Now we basically do not know want to change the colours of nodes because, every node is added initially. So, maybe we can let go off this thing ok.

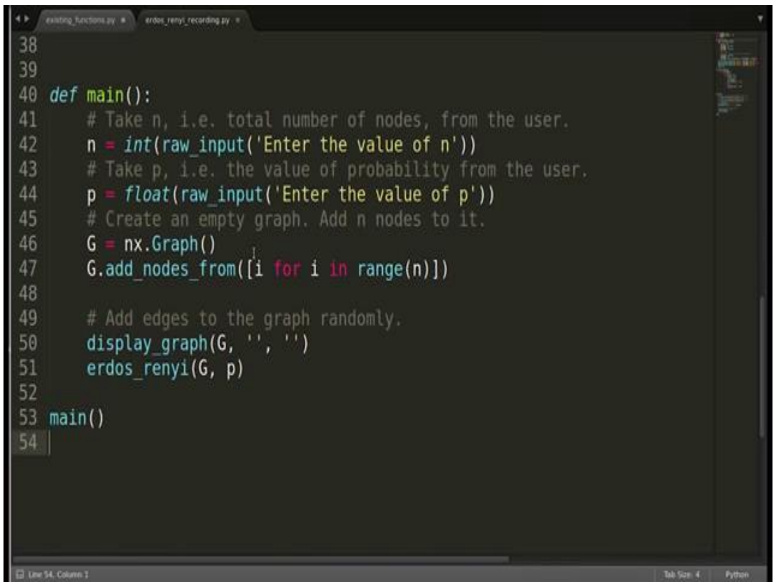
Only the edges coloured has to be changed. So, this is fine, and this is fine and this should be fine let me see, we want to display the nodes right. So, let me remove the new nodes statement from here and the rest of the nodes will be displayed in red colour. Rest of the nodes is what we have to define here.

So, I will write rest and I will just copy from here ok. I am just playing around you will be able to do it yourself as well or you can write the function once again. I will just explain after I am done with this. I think they should do ok.

So, in our case when we will calling display graph function from the set of Renyi function, we are passing 3 things and in all the cases G is there and empty string is passed as i and the third one is the list. In case we are not adding the edge, this list is going to be empty.

So, let us see which condition will be followed. This condition will not be followed because, ne is not empty. We will be going into this and rest of the nodes is are going to be all the nodes and new edges yes we are passing a list there, yes this should work.

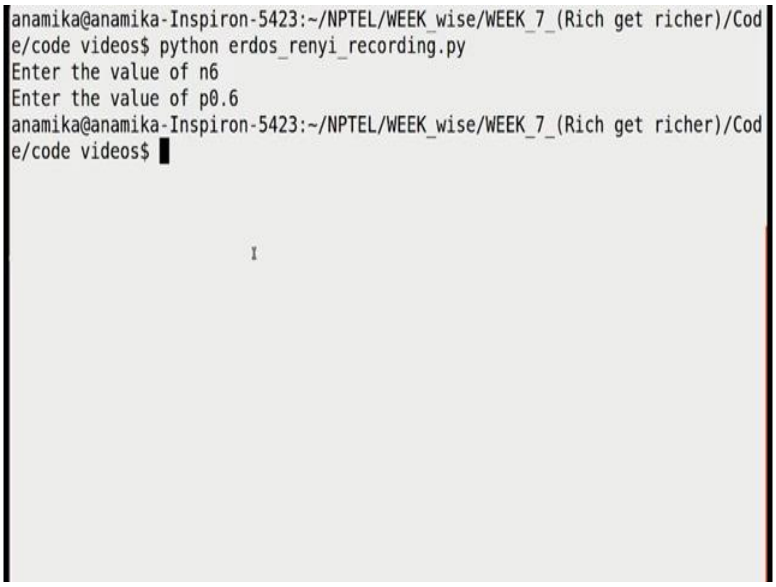
(Refer Slide Time: 10:27)



```python
def main():
    # Take n, i.e. total number of nodes, from the user.
    n = int(raw_input('Enter the value of n'))
    # Take p, i.e. the value of probability from the user.
    p = float(raw_input('Enter the value of p'))
    # Create an empty graph. Add n nodes to it.
    G = nx.Graph()
    G.add_nodes_from([i for i in range(n)])

    # Add edges to the graph randomly.
    display_graph(G, '', '')
    erdos_renyi(G, p)

main()
```

Let us go to the main and the value of n is going to be integer right and the value of probability is going to be between 0 and 1. So, that is actually no going to be integer. So, let us add a float there. Now let us call this main function. Now we can now check the functionality of this code.

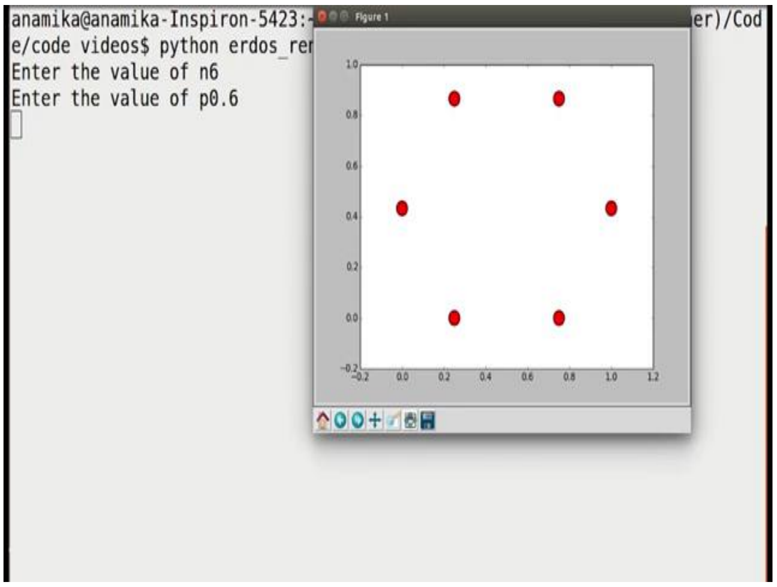
(Refer Slide Time: 10:53)



```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod
e/code videos$ python erdos_renyi_recording.py
Enter the value of n6
Enter the value of p0.6
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod
e/code videos$ 
```

So, let us check how it works. Enter the value of n, so let me pass a small value so that we can check the graphs at every iteration let me pass 6 for example. The value of p let me add 0.6 here we can pass any value between 0 and 1.

(Refer Slide Time: 11:15)



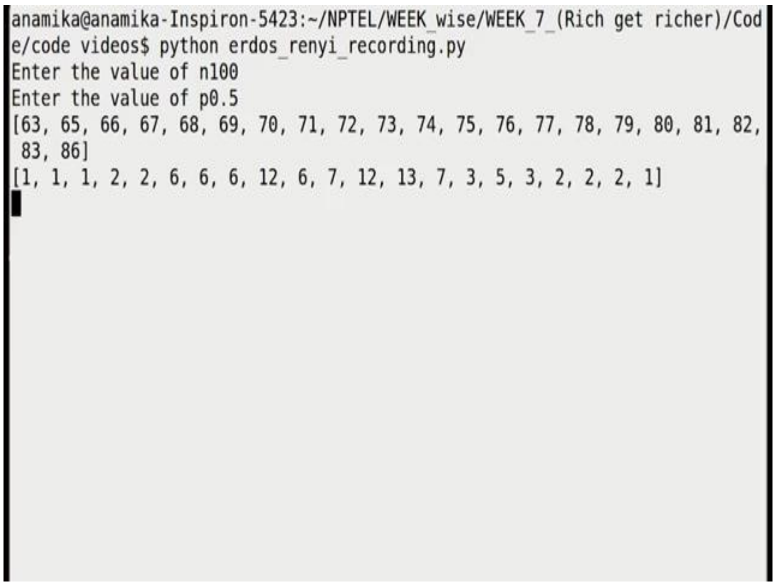
```
anamika@anamika-Inspiron-5423:~
e/code videos$ python erdos_re
Enter the value of n6
Enter the value of p0.6
```

So, this is the initial graph where there are only nodes, 6 nodes are there, there are no edges. Now I am closing this and in the next iteration we are getting this edge between these 2 nodes and in the next iteration again we are getting this edge, we are getting an edge and in the 4th iterations as you can see we are not getting any edge. We got this edge and so in some iterations as you can see, we are getting some edges one edge and in some iterations, we are not getting.

So, I think I have to do this for 6 into 6, 36 times right. So, as you can see 36 times this loop will run and in some of the iterations i might be equal to j, so no edge might be added. In some of the iterations the r might be greater than p, so h might not be added. So, this was the functioning of Erdos Renyi model that we created. I also told you that the degree distribution that Erdos Renyi model follows is not power law rather it is normal distribution. We had created a function for plotting the degree distribution in one of previous video.

So, I am going to just use that function as it is, let me copy that and we will use this function plot, degree distribution, so let us use it here. In order to show you the degree distribution I will take some good number of nodes so that we can see a nice plot with a with many values. So, I will remove the displaying of the graph so that, I do not need to press all tab 4 again and again. So, I am just commenting this displaying of graph here. Now let us check the functionality of this code.

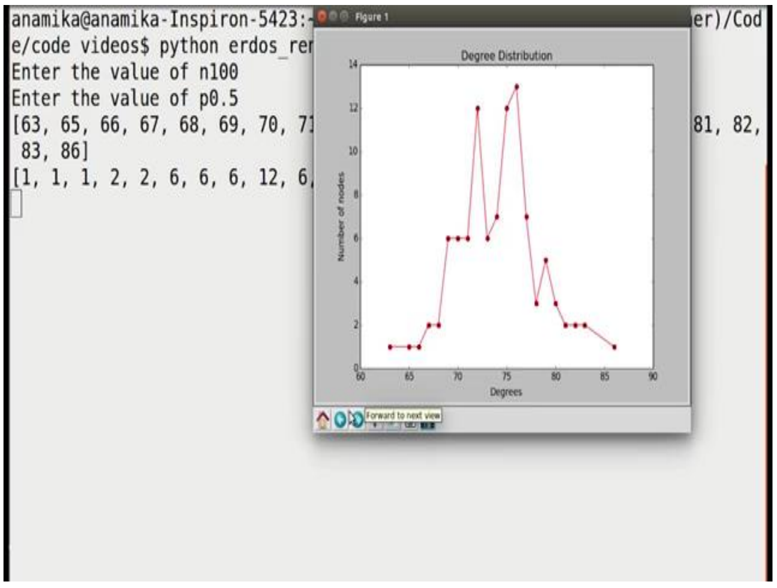
(Refer Slide Time: 13:23)



```
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_7_(Rich get richer)/Cod
e/code videos$ python erdos_renyi_recording.py
Enter the value of n100
Enter the value of p0.5
[63, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
 83, 86]
[1, 1, 1, 2, 2, 6, 6, 6, 12, 6, 7, 12, 13, 7, 3, 5, 3, 2, 2, 2, 1]
```

So, we have to observe the degree distribution for an Erdos Renyi network. Let me add 100 nodes and let me add the probability to be 0.5.

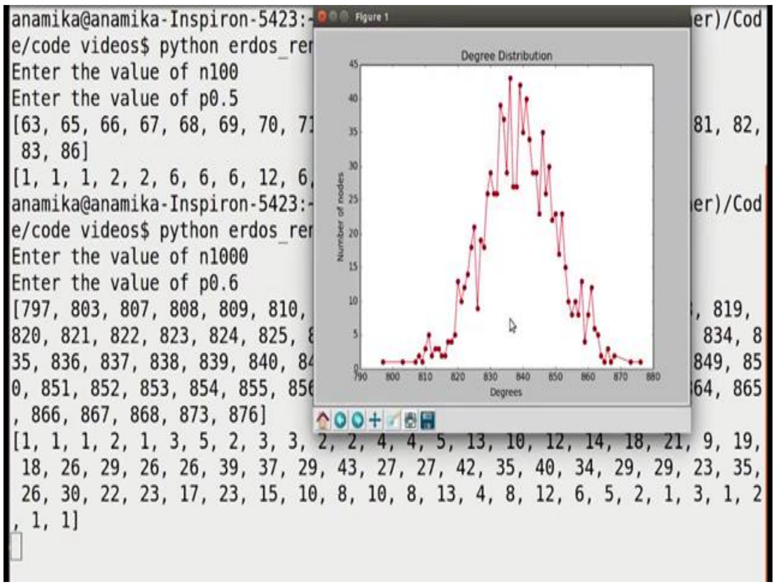
(Refer Slide Time: 13:33)



```
anamika@anamika-Inspiron-5423:~                                     er)/Cod
e/code videos$ python erdos_rer
Enter the value of n100
Enter the value of p0.5
[63, 65, 66, 67, 68, 69, 70, 7                                      81, 82,
 83, 86]
[1, 1, 1, 2, 2, 6, 6, 6, 12, 6,
```

So, this is the sort of curve that we obtain. So, you can see since we were adding the edges randomly, there were no preferential attachment. We can see that there are very less nodes with very less degree and there are very less nodes with very high degree and there are very large number of nodes with medium degree. So, that is a sort of distribution that we obtain, sort of normal distribution that we get in the case of a random network.

(Refer Slide Time: 14:08)



```
anamika@anamika-Inspiron-5423:-                                        er)/Cod
e/code videos$ python erdos_rer
Enter the value of n100
Enter the value of p0.5
[63, 65, 66, 67, 68, 69, 70, 71                                        81, 82,
 83, 86]
[1, 1, 1, 2, 2, 6, 6, 6, 12, 6,
anamika@anamika-Inspiron-5423:-                                        er)/Cod
e/code videos$ python erdos_rer
Enter the value of n1000
Enter the value of p0.6
[797, 803, 807, 808, 809, 810,                                         , 819,
820, 821, 822, 823, 824, 825, 8                                         834, 8
35, 836, 837, 838, 839, 840, 84                                        849, 85
0, 851, 852, 853, 854, 855, 856                                        64, 865
, 866, 867, 868, 873, 876]
[1, 1, 1, 2, 1, 3, 5, 2, 3, 3, 2, 2, 4, 4, 5, 13, 10, 12, 14, 18, 21, 9, 19,
 18, 26, 29, 26, 26, 39, 37, 29, 43, 27, 27, 42, 35, 40, 34, 29, 29, 23, 35,
 26, 30, 22, 23, 17, 23, 15, 10, 8, 10, 8, 13, 4, 8, 12, 6, 5, 2, 1, 3, 1, 2
, 1, 1]
```

Let me try it once again for a greater number of nodes. Maybe I will add some 1000 nodes and the probability to be 0.6. So, again this is nice curve that we obtain. So, as you can see as you keep increasing the number of nodes, you can see the nice distribution that it takes ok. So, that was about the Erdos Renyi network to Erdos Renyi model to create the random networks and it follows the normal distribution.