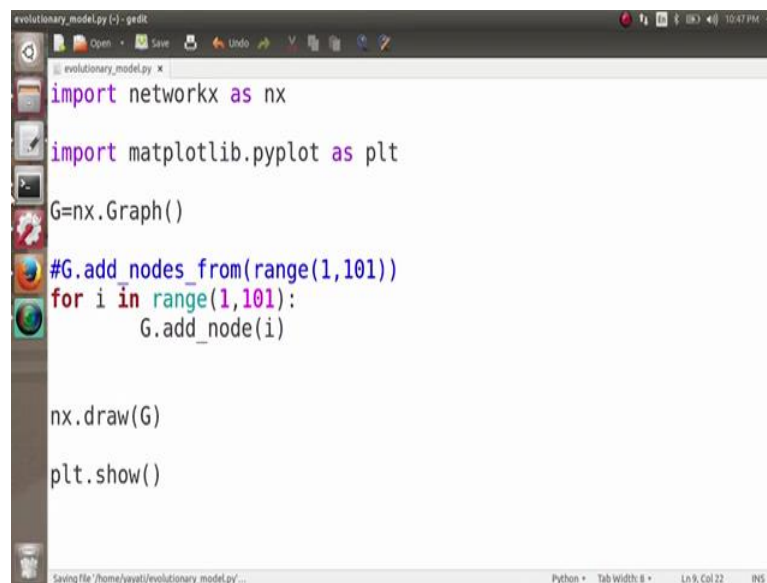**Social Networks**
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture - 47
Strong and Weak Relationships (Continued) & Homophily
Fatman Evolutionary Model - The Base Code (Adding people)

(Refer Slide Time: 00:05)

So, what do we do first is we have to create a graph on 100 nodes where everybody has a different BMI? So, mainly we have this city and the city has 100 people which we depict as 100 nodes. So, to make this graph first be import a function sorry be import a package import networkx as nx and let us save this file, let us save this file as let say evolutionary model.py. So, we save this file a evolutionary model.py. The import networkx is nx, now we have to create a graph on 100 nodes which we know is very simple we have already created many graphs.

So, what we simply need to do is first of all create a graph G = nx.Graph and then we have to at 100 nodes here. So, we write G dot add_nodes from and this function demands a list here. So, what is the list here? Range 1 to 101. So, what will this function do it will add 100 nodes to this graph with the indices being 1 2 3 4 up to 101. So, this is one we do will get will soon look at another wave of doing the same thing.

And next I want to plot this graph and see how does it look like, so far as we know for plotting we need import mat plot lib dot pyplot as plt and then we can use a function and next dot draw G and then plt.show.

(Refer Slide Time: 02:07)
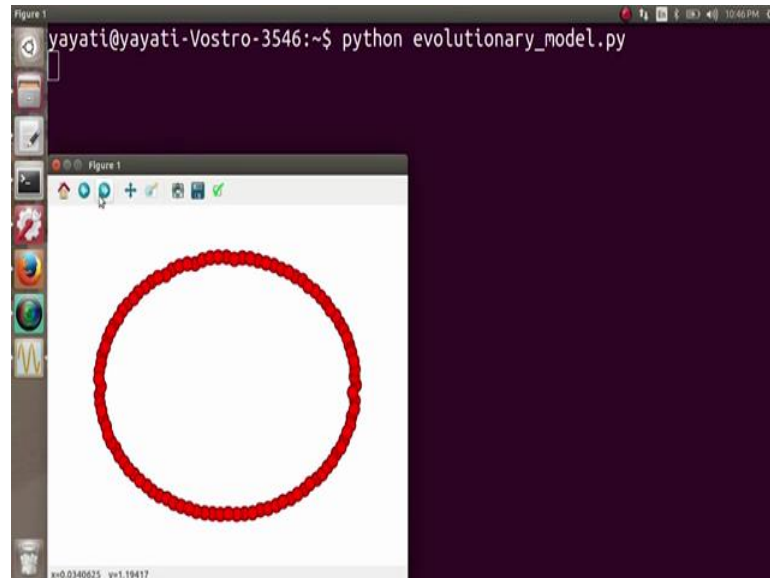
I save this file let us run it and see, python evolutionary model.py and here we can see a graph with 100 nodes. Let us quickly look at another way of doing the same thing for you might be already knowing it I am just repeating it.
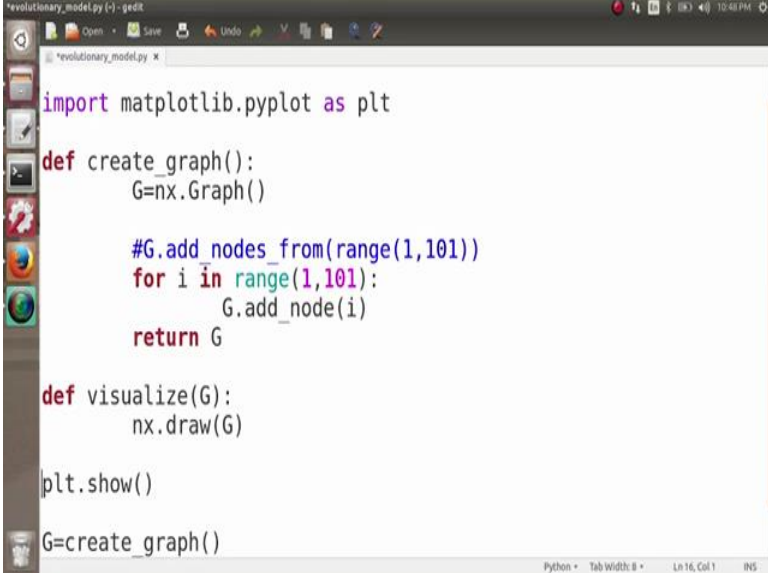
(Refer Slide Time: 02:32)
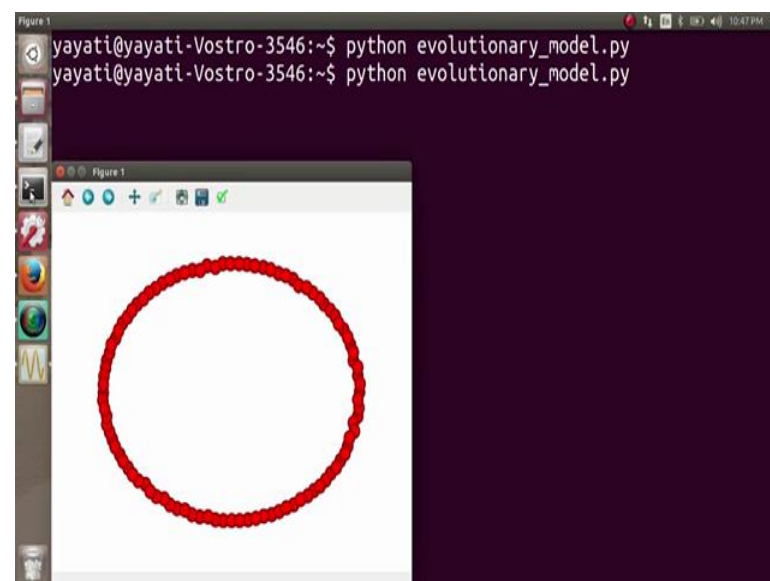
So, here what we have done is we have made a list here and then passed it what we can do is have a four loop here. So, we can do for i in range 1 to 101 what do we do G. add_node(i), and it creates the same output as before. So, when we run this file and we do python evolutionary underscore model.py we get the same graph perfect.

Next what we want to do is we want to a see some random BMIs to these people. So, for assigning random BMIs to these people what we will do is. So, we will be first before doing that we want to use a good use of we want to make a good use of functions here. So, whatever will be doing will keep writing it everything in functions only. So, this particular piece of code G = nx.Graph to add_node(i) was for creating our graph.

(Refer Slide Time: 03:44)

So, what we do here is we simply pass a function here G = create_graph and this entire thing we shift under the function create underscore graph. So, we have this function here - define create underscore graph and what this function does is it creates a graph haven 100 nodes and it returns the graph G.

And these two statements nx.draw(G) and plt.show we are using it to visualize our graph. So, what do we do is we define another function here define and then we name this function as visualize, define visualize G what it does is it will draw or graph and showing?
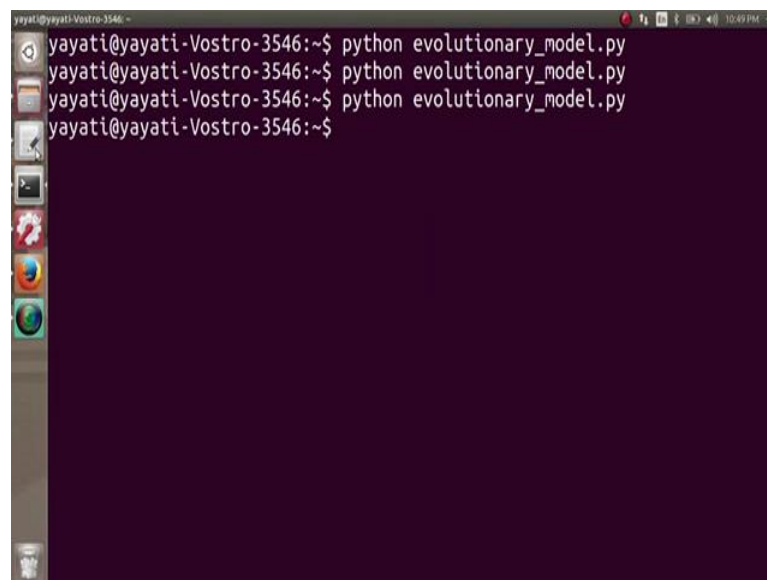
(Refer Slide Time: 04:39)

So, we pass both of these statements inside this function and whenever we have to see this graphic simply called visualize G. So, let us run it and see quickly.

(Refer Slide Time: 04:47)
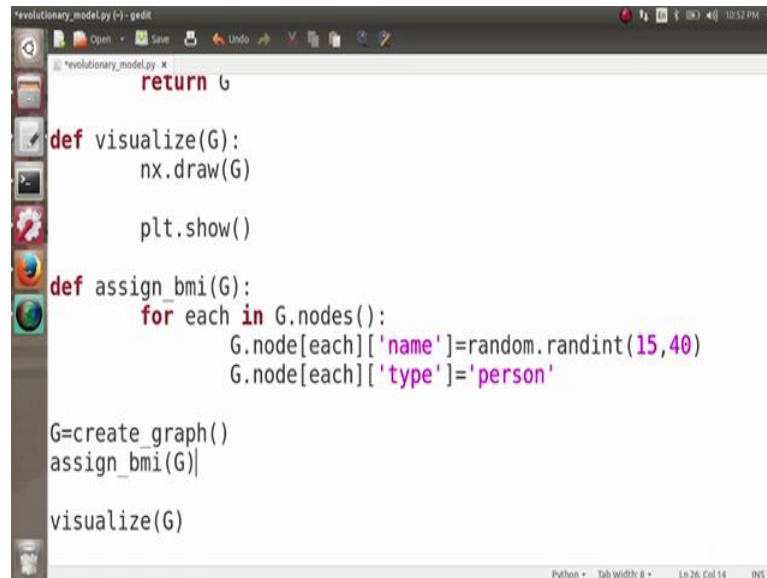
So, we get here the output which we want it. So, till now everything is perfect. So, we have a graph having 100 nodes.

Next what we want to do is we want to assign a body mass index to every person here in this graph. So, here I have 100 people and you want to assign a BMI to every person. So,

for that again we take the help of a function and we call the function assign BMI and we pass the graph G in this function.

(Refer Slide Time: 05:16)
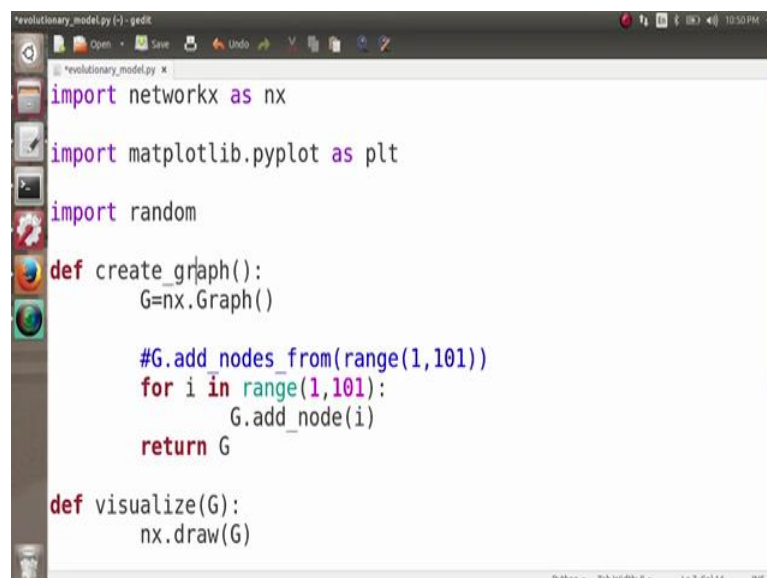
So, let us write down the body of this function, we have define, assign, underscore BMI and we have a graph G here. How do we assign a BMI, so we know that BMI is going to be a number from 15 to 40? So, what we are going to do is for every node in the network we generate a random number from 15 to 40 and assign that number as the BMI of this node.

(Refer Slide Time: 05:56)

So, since we need a random number, we need derive package random here. So, we import our package random here, and we come back to our function assign underscore BMI. How do we assign a BMI to our node? So, probably you will remember we have used the attributes of the node, we have learnt about the attributes of the node. So, we are going to make the use of the attributes of the nodes. So, how do we do it for each in G dot nodes? So, we have an iterator which iterates you every node in G. So, for each in G dot nodes what do we do is we add an attribute G.node each and let us have the name of this attribute as name. So, better option was having the BMI here, but we are using the name of the attribute to be name itself have a reason why I am using the attribute name to be name here which you will understand as you see more of this code.

So, every person has here, and attribute and the attributes name are name. So, the attribute is name and the value of the attribute is random number, random integers from 15 to 40 what I do is random.randint and it should be from 15 to 40. In addition to an attribute name I want one more attribute associated with each node here and you guess what that attribute is going to be. I told you that we are going to have two types of nodes in this network. So, one type of nodes is corresponding to the people in the network and another type of node is corresponding to the social for kind. So, will need to distinguish between these two kinds of nodes, so hence I take an attribute here G dot node each and I name this attribute as type.

So, G dot node each type equals to and these are 100 people. So, I name this as I keep the value as person. So, for each of these 100 nodes there type is equal to person. So, we have assigned a BMI to every node and let us put this assign BMI function before visualizing the graph.
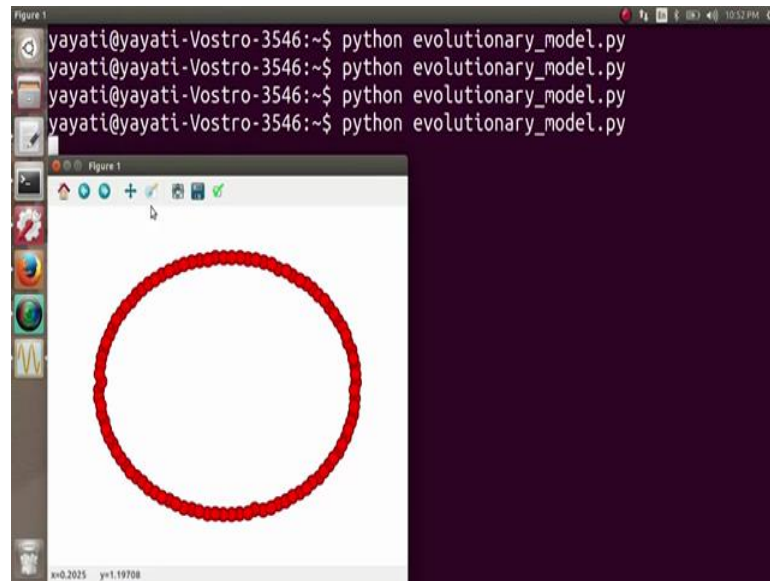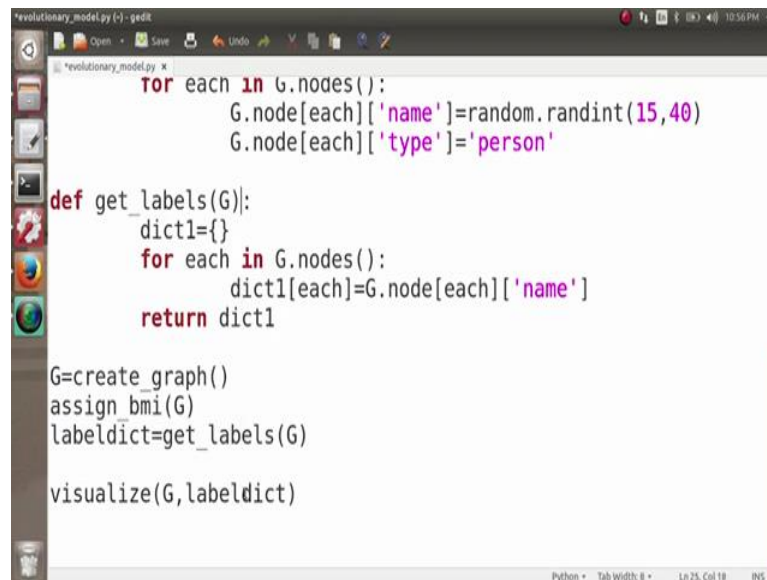
And let us run the code and see it now run the code and we see it so, but the graph looks like the previous graph only so every node here has a BMI, but we do not see any BMI here why, because currently or nodes are having no labels. So, we need to label our nodes and this particular labeling which we want is the BMI. So, every node should be labeled with its BMI. Every node should have a number written on it and that number should be the BMI of this node.

So, how do we do that? After assigning BMI is to these nodes, we want these as labels. So, mainly we need a dictionary for all the labels which can be displayed.

```python
        for each in G.nodes():
                G.node[each]['name']=random.randint(15,40)
                G.node[each]['type']='person'

def get_labels(G):
        dict1={}
        for each in G.nodes():
                dict1[each]=G.node[each]['name']
        return dict1

G=create_graph()
assign_bmi(G)
labeldict=get_labels(G)

visualize(G,labeldict)
```
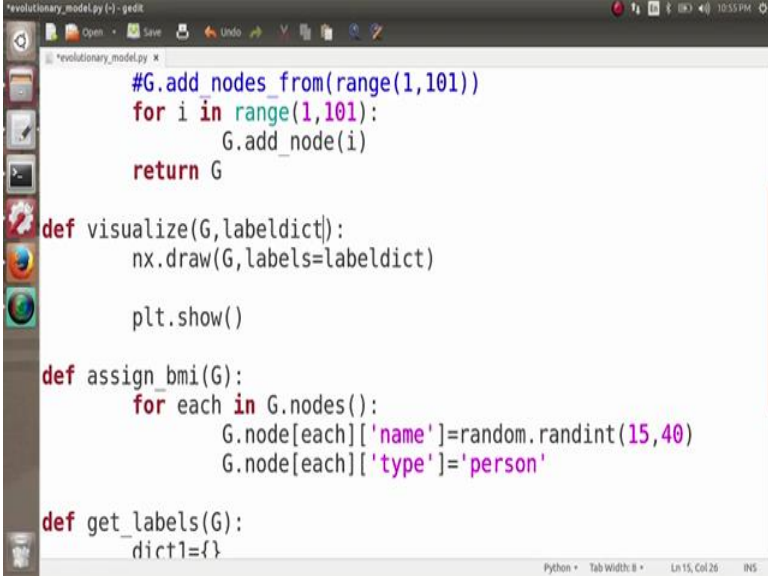
We want the dictionary here let us name this dictionary as labeldict and to get this dictionary labeldict we use a function get labels from G and this is not an implicit function get labels we need to define these function. So, what do we do? We define this function get underscore labels G and what this function is going to do first of all we create a dictionary we have a dictionary here let us name it as dict to 1 and what will add to this dictionary is the label for every node.

Hence for each in G dot nodes what we are going to do is dict1 and the value in the value for each, so each is the key here, dict to 1 of each. So, each is am node here dict1 of each should be equal to, what it should be equal to the BMI of this person, the BMI of each. So, how to access the BMI of each BMI is are attribute we do G.node each under BMI was stored in the attribute named name and at the end we return our dictionary dict to 1.

So, here you see we get a dictionary labeled dict equals to get underscore labels G and this dictionary has all the labels. Now to see these labels in our graph we look at this function visualize G, we go above will see this function visualized G. So, it was having nx dot draw G we need to add a small code here, we want the labels with the each node.

So, we add labels = and what should be the labels = a dictionary and what was the dictionary label dict. Do you see a problem here? I will see a problem here, we are using labels = labeldict how does my code node what is, does my code node what is label dict. So, you need to pass labeldict as an argument here. So, we pass this labeldict as an argument here and similarly here while visualizing the graph G we need to pass labeldict here. So, I hope that the code is clear do you.

So, I will just quickly recap what did we do is after assigning the BMI we created a dictionary the name of the dictionary is labeldict and what is the value what is this dictionary carrying is this dictionary is having one key for each of the nodes in the network and the value corresponding to that key, corresponding to that node is the BMI for that node and then we written this dictionary dict1 and its collected in label dict. Now we have a dictionary for all the labels in the graph to do visualize the graph we pass this labeled dict and here by visualizing we have an extra parameter here labels which is equal to label dict.
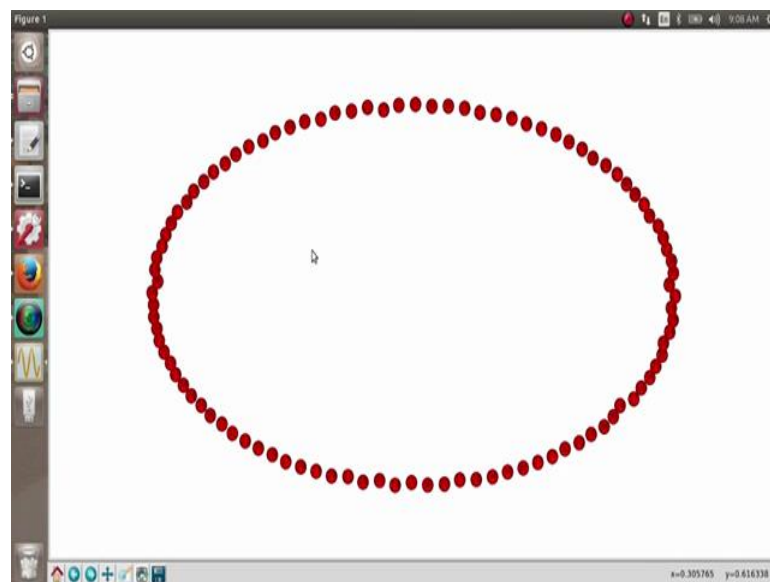
So, now we execute this code and see our graph. So, I execute our code and you see the graph here.
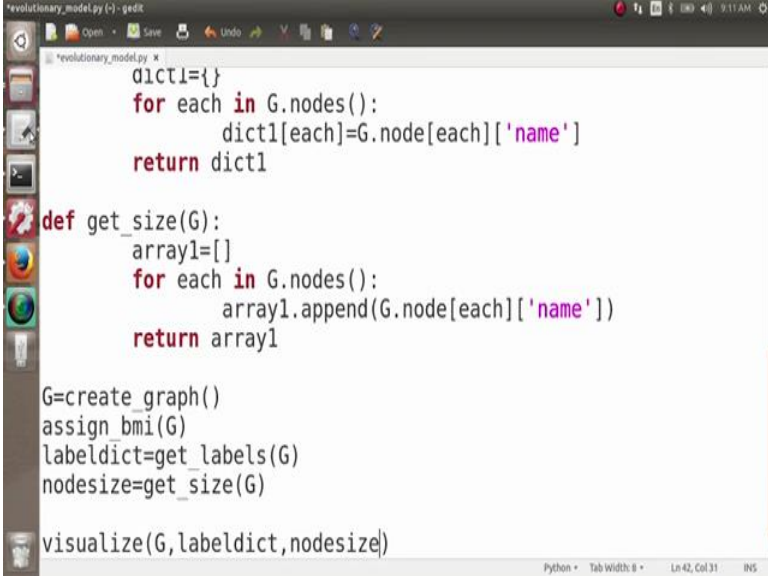
(Refer Slide Time: 12:42)
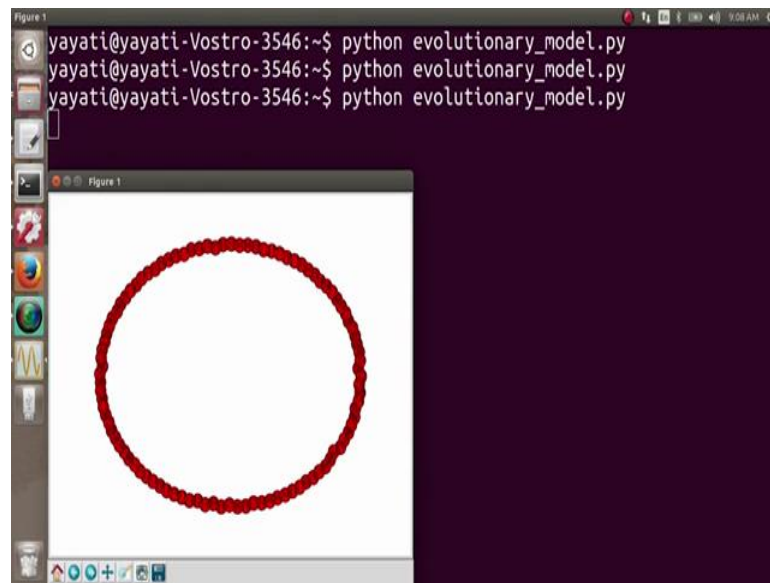
So, when we look here at this graph you can see that every node is marked with the label and the label is its BMI.

(Refer Slide Time: 12:51)

So, these are 100 nodes if you look at the labels for all these nodes the label the range from BMI that is from 15 to 40 and then you can also see that there are two nodes it is having a label 31 and this is also label 31. So, these two nodes they are having the same BMI. So, here we get the graph where we get different people and every person is labeled with his or her own BMI.
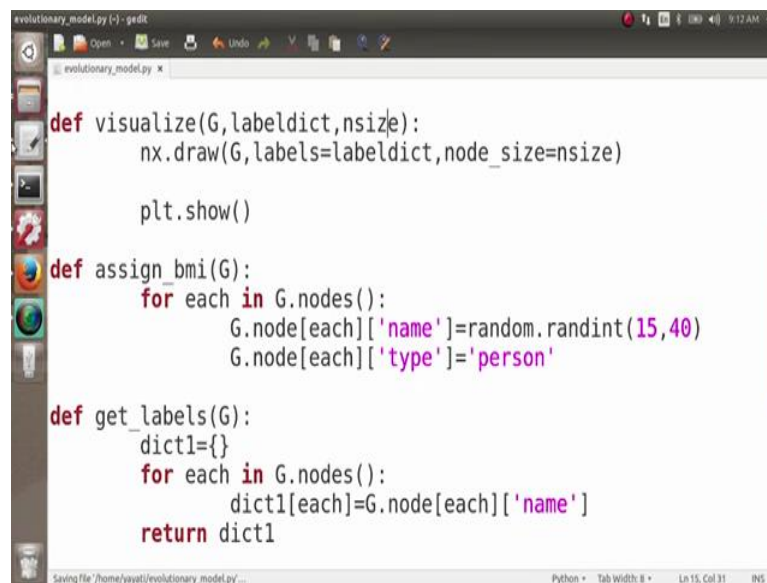
(Refer Slide Time: 13:29)

Next what am I interested in, what I want in the graph. Suppose that we want to see this difference in the form of the node size that is I want a person having a higher BMI to appear larger as compared to a person who is having a lower BMI. So, what do we do for that is I want to assign a size to every node and the size of the node should be equal to or proportional to the size of a node should be proportional do its BMI? So, what do we do for that is, I want an array here and this array consist of, this array consists of these sizes of different node? So, I take an array here node size and the value of this node size is to get the value I call a function get underscore size G.

And again, this function gets underscore size we have to define. So, we define this function here - define get underscore size G and what this function should output is an array of the sizes of different nodes whatever size we want to keep in the graph. I create an array here let us say array one and then for each in G.nodes. So, I put an iterator for all the nodes in the graph for each in G dot nodes what do we do? We do array one dot append and what do we append in this array in this array we append the BMI value extract that node and as we know how do we extract the BMI value of a node is G dot node each and then we have here name. This gives us the BMI value of each node and at the end we return array1. So, we get here and array node size which has the size of different nodes.

We want to see it in the graph. So, to see it in the graph we first of all pass it in the function visualize, so in the function visualize we also pass here the array node size. And then when we are visualizing the graph what I do here is I include a new parameter which says node underscore size. So, and the value for every node is to be extracted from the array node size and also, we need to pass this array here. So, let me pass node size here.

(Refer Slide Time: 15:42)


```python
def visualize(G,labeldict,nsize):
        nx.draw(G,labels=labeldict,node_size=nsize)

        plt.show()

def assign_bmi(G):
        for each in G.nodes():
                G.node[each]['name']=random.randint(15,40)
                G.node[each]['type']='person'

def get_labels(G):
        dict1={}
        for each in G.nodes():
                dict1[each]=G.node[each]['name']
        return dict1
```
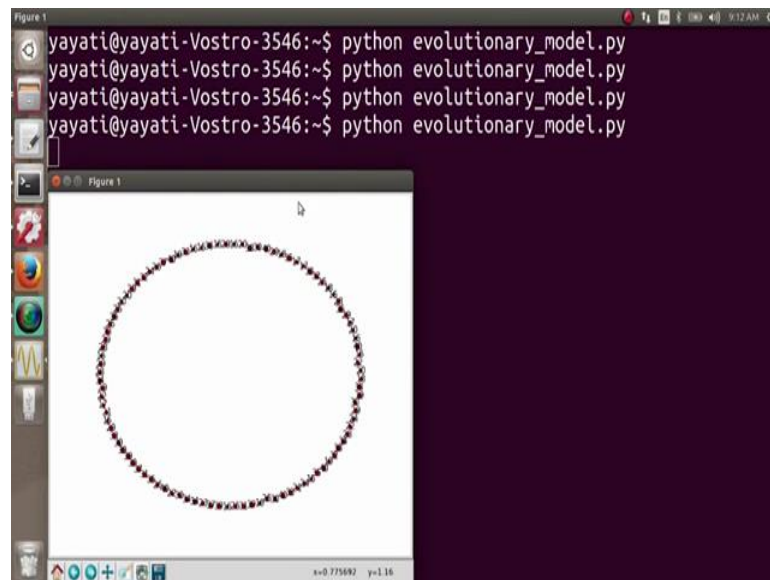
You can also give different names for like we pass there the argument node size. So, here I can actually keep it nsize and at the end I pass here n size. Just doing it this way for the sake of clarity, we can also have, we can keep the same name for the argument and we parameters and we can also give different names. Let us go back execute this code and see.
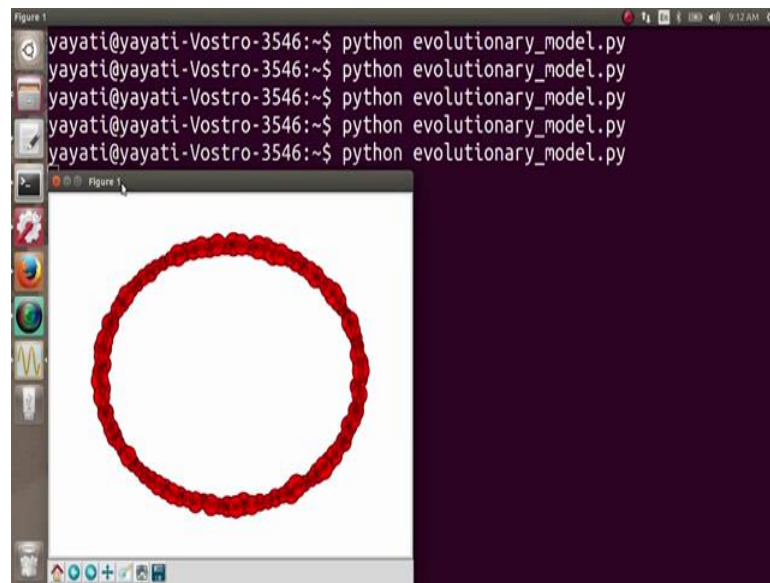
(Refer Slide Time: 16:28)


(Refer Slide Time: 16:33)
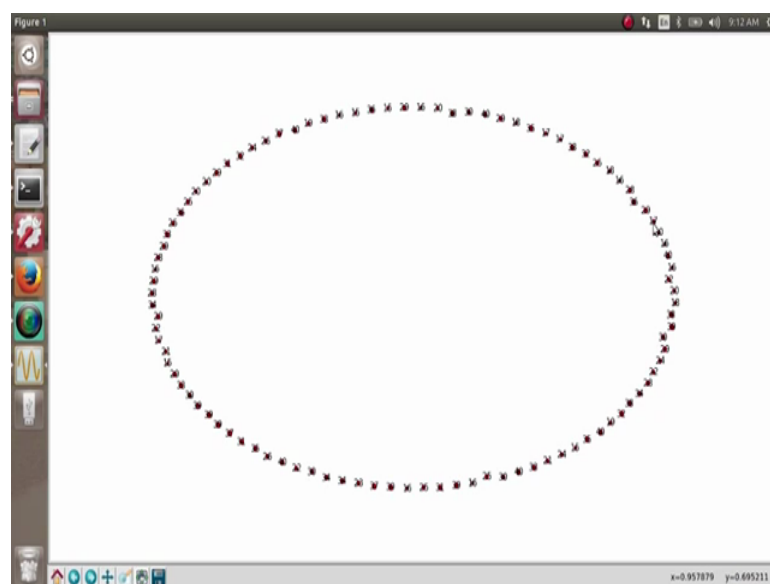
Let us execute this code and what we see here is the size of the nodes have changed, but we are not being able to. So, you can see here that some nodes are of smaller size some nodes are of larger size, but still the difference is not very clear the graph does not look very need. So, what I do is I go back to the code and for and for each value way wherever we are collecting the size here. So, for each value what I do? I multiply it by 20. So, I scale the value for the size of every node by 20. So, that the graph looks cleaner.

(Refer Slide Time: 17:10)

And then we go back and then we execute this code. So, you can see here this looks more beautiful from the previous graph.
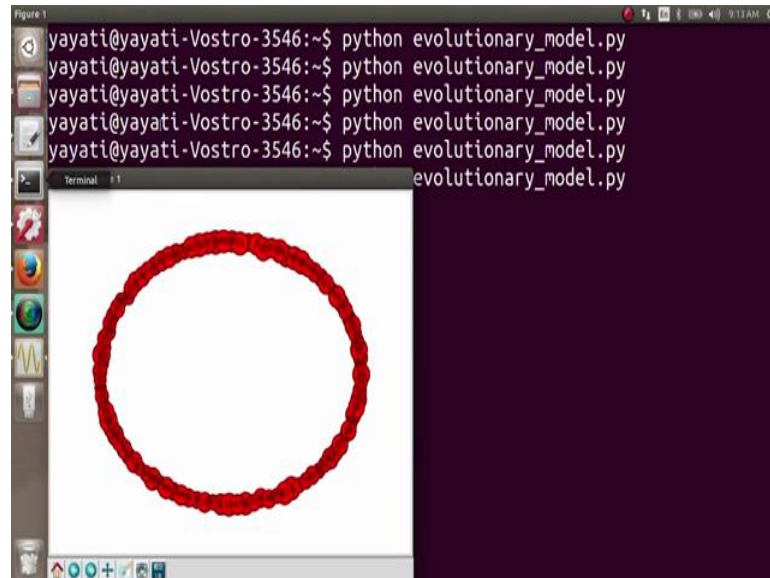
(Refer Slide Time: 17:14)
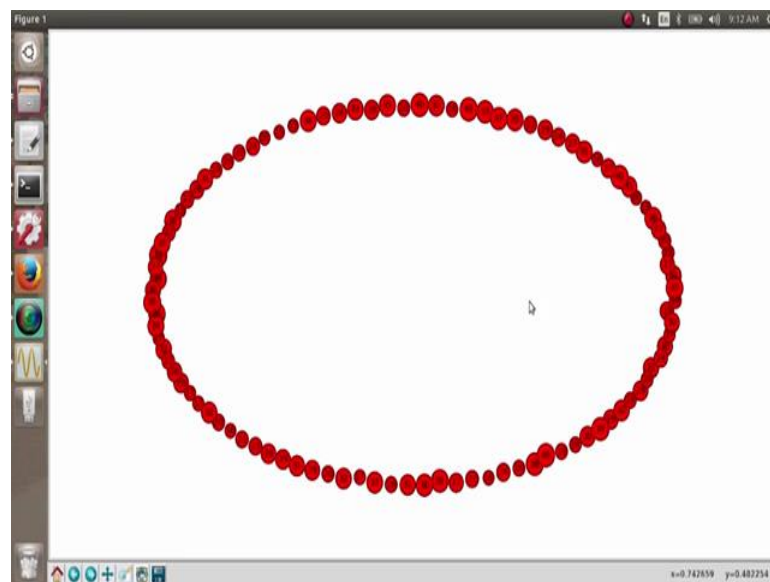
So, you can say here the nodes which are having a high BMI such as 40 they appear in the biggest; they appeared biggest in size and the nodes which are having a lower BMI such as 15 they appear smaller in size.

So, till now we have just created a network on 100 people where every person has a different BMI and the size of every person is proportional to his or her BMI when we

visualize this network. So, for the time being what I want to do is while visualizing my graph I will now remove the labels. So, these are the labels which shows the BMI in our graph, I will just I will remove the labels.

(Refer Slide Time: 18:04)

(Refer Slide Time: 18:07)
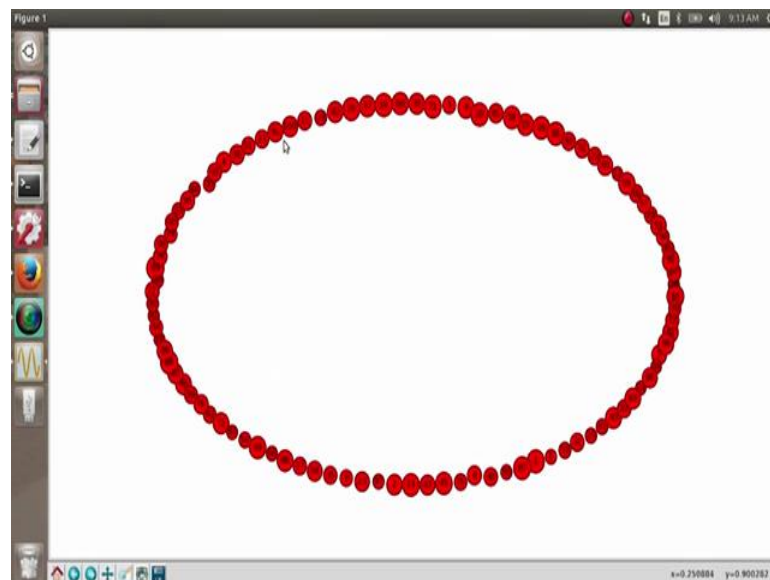
So, here we say that the node 10. So, can see that the node 10 is of the is having a very high BMI, node 87, node 13 these are all having very high BMI and the nodes like 83, 64, 66 these are having smaller BMI.