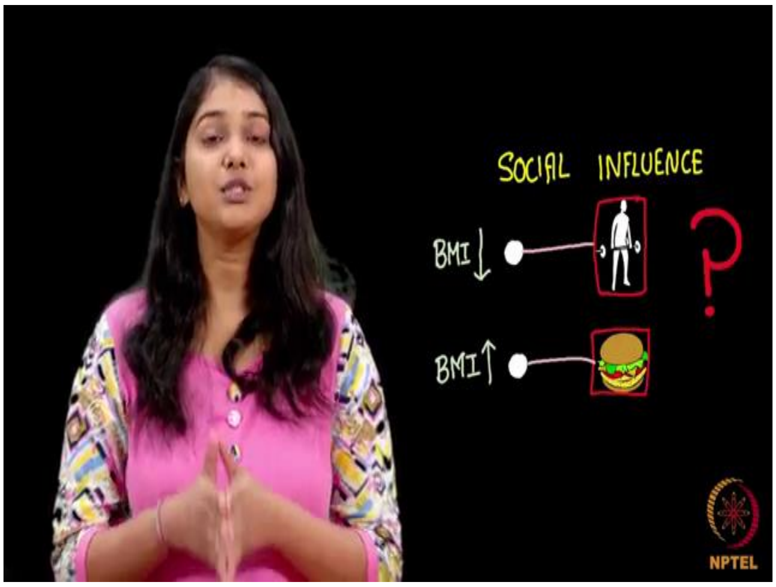
Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Lecture – 52
Fatman Evolutionary Model – Implementing Social Influence

Finally, we have implemented homophily, we have seen how do implement three kinds of closures scenario evolutionary model and the last and the most interesting thing which remains is the social influence and it is actually the easiest. We need not do anything here in this model to achieve social influence, as I have told before.

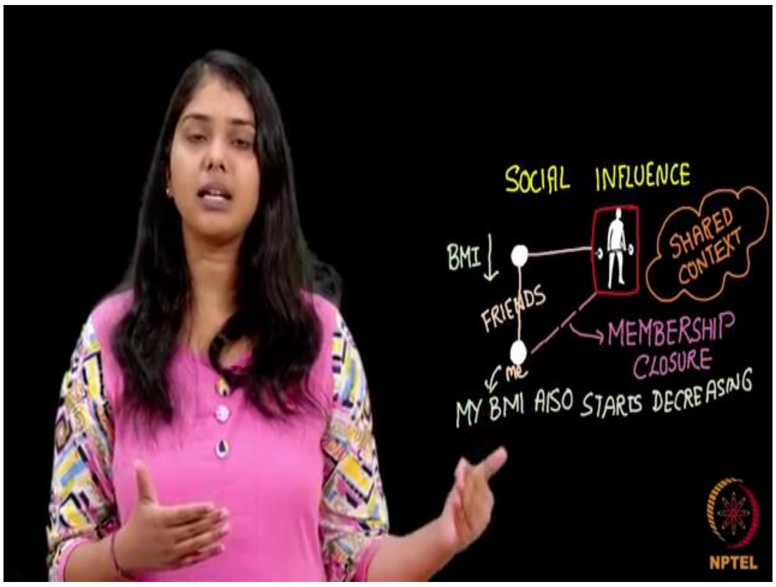
(Refer Slide Time: 00:20)



So, there is a very subtle way in which we capture these social influence happening; how do we do that is, assume that if a person is connected to a gym so; obviously, he is being connected to gym should result in a decrease in his BMI and if a person is connected to an eat out place, they should result in increase in the BMI of this person.

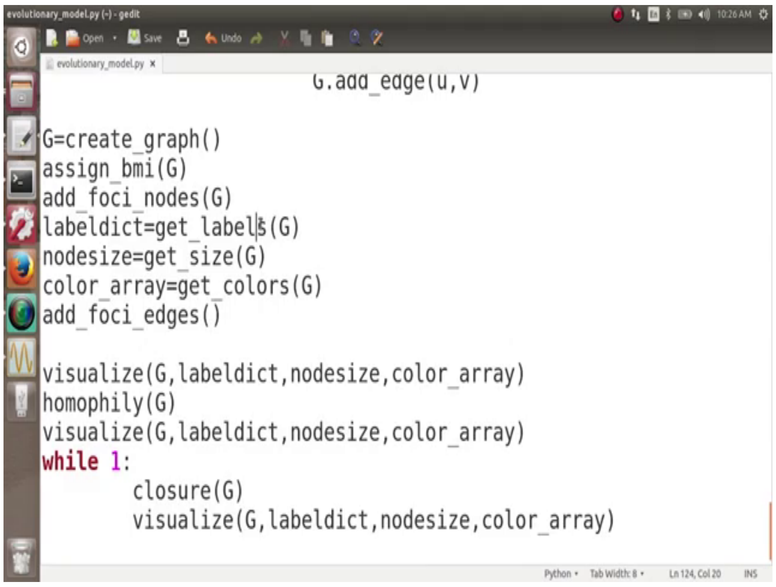
So, what do we do; for every node which is connected to a gym, we reduce its BMI by 1; every iteration and every node which is connected to an eat out place, we increase its BMI by one every iteration and we just do this and the social influences captured; How? So, as I have told you; we have seen here now, when a person gets connected to a gym; he is losing weight. Now, I have a friend and this friend is going to the gym; now because of membership closure, I tend to become a part of the gym and when I become a part of the gym; as you saw I will start losing weight.

(Refer Slide Time: 01:22)



So, this is one aspect of social influence, so we have mainly looked at three reasons why in which social influence can happen, but we capture one of those here and that reason is the sheared context. When 2 people are the part of the same social foci, it leads to an influence. So, we have seen that it is like; if 2 people both have a brilliant teacher, both of them will start having good marks. Similarly, if 2 people here both of them are part of gym; so one person who was going to gym, another also started going to gym will start losing weight and similarly with an eat out place; the effect is reversed, so this is how we capture social influence here.

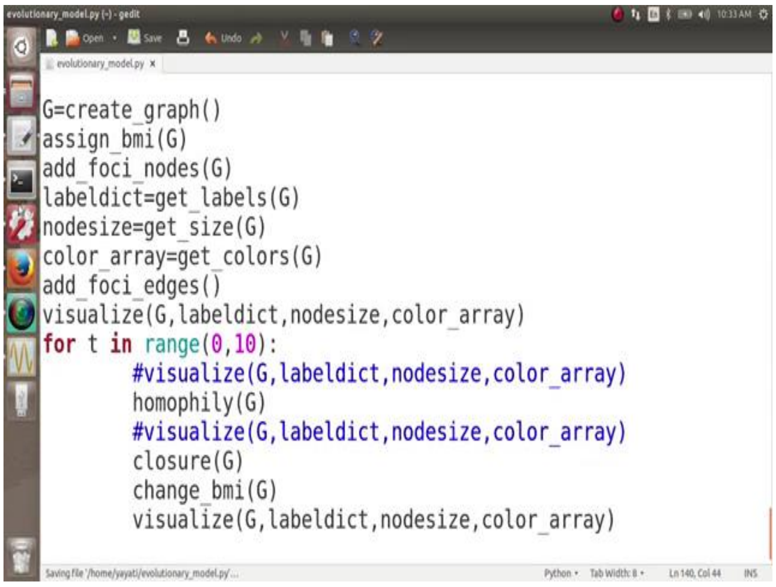
(Refer Slide Time: 02:25)



```
                        G.add_edge(u,v)

G=create_graph()
assign_bmi(G)
add_foci_nodes(G)
labeldict=get_labels(G)
nodesize=get_size(G)
color_array=get_colors(G)
add_foci_edges()

visualize(G,labeldict,nodesize,color_array)
homophily(G)
visualize(G,labeldict,nodesize,color_array)
while 1:
        closure(G)
        visualize(G,labeldict,nodesize,color_array)
```

Finally, what we want to do here is; we want to implement social influence. How we are implementing social influence? We have already discussed before; well it is an indirect implementation of social influence. So, it is mainly at every iteration a person who is associated with gym; lose his weight and a person will associated with an eat out, gains weight and then as we know that membership closure is happening. So, let us say I am a part of gym and then I have a friend, who is fat, then he looks at me and I have joined gym, he also joined gym and hence he started losing weight. So, this the kind of social influence we are going to implement here.

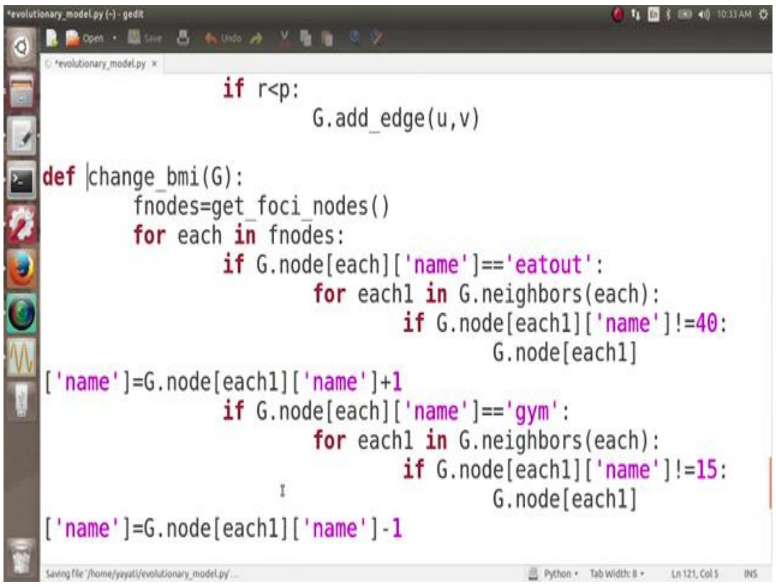
(Refer Slide Time: 03:18)



```
G=create_graph()
assign_bmi(G)
add_foci_nodes(G)
labeldict=get_labels(G)
nodesize=get_size(G)
color_array=get_colors(G)
add_foci_edges()
visualize(G,labeldict,nodesize,color_array)
for t in range(0,10):
        #visualize(G,labeldict,nodesize,color_array)
        homophily(G)
        #visualize(G,labeldict,nodesize,color_array)
        closure(G)
        change_bmi(G)
        visualize(G,labeldict,nodesize,color_array)
```

So, for meanwhile let us just comment everything and now what we want to do here is; we want to change weights, we want to change BMIs of people.

(Refer Slide Time: 03:41)



```
			if r<p:
				G.add_edge(u,v)

def change_bmi(G):
	fnodes=get_foci_nodes()
	for each in fnodes:
		if G.node[each]['name']=='eatout':
			for each1 in G.neighbors(each):
				if G.node[each1]['name']!=40:
					G.node[each1]
['name']=G.node[each1]['name']+1
		if G.node[each]['name']=='gym':
			for each1 in G.neighbors(each):
				if G.node[each1]['name']!=15:
					G.node[each1]
['name']=G.node[each1]['name']-1
```

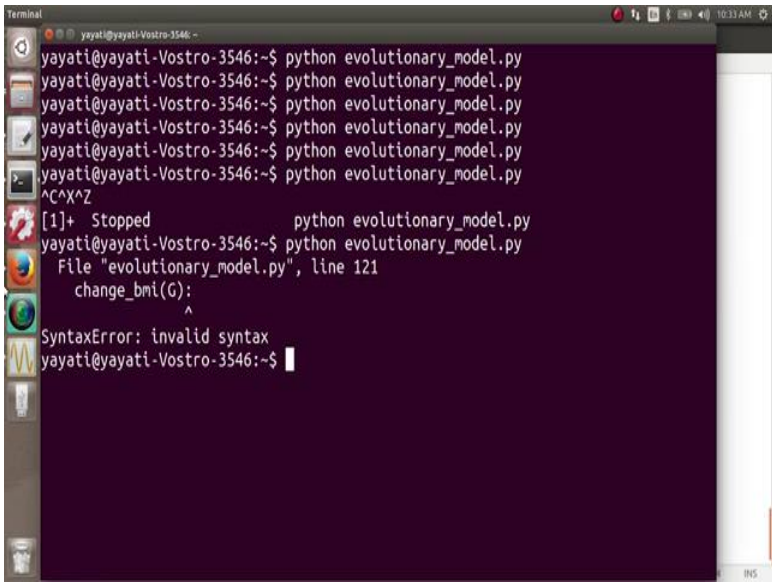
So, we called change_bmi(G) and what this function change_bmi(G) does is change_bmi(G). So, for each in f nodes; what do we do for each1 in G.neighbors(each); what do we do, if G.node[each1], but again we are doing it for the foci node which are either an eat out or a gym. So, we want to see that so for each in f nodes; we again need to put a node here, if G.node[each]. So, if this each node its name; its name == ‘eat out’.

If its name = ‘eatout’ only then we are going to insecure this code, so if its eat out we look at each of its neighbors and if each G.node[each1][‘name’]. So, now since we are going to increase the weight, first this name should not be equals to 40 because if it is equal to 40, we cannot further increase the BMI of this person. So, if each if G.node[each1]; name != 40; what do we do is we increase it by 1.

G.node[each1], name = G.node[each1][‘name’] + 1 and then we again do this same thing for gym. It is essential we have to copy paste the same code here, if G.node each, name = ‘gym’. So, if it is a gym we look at all of its neighbors and if for a neighbour this value should not be equals to 15; if this value is not equals to 15; we decrease its BMI by 1, so this is how we are going to change the BMI of people.

Now, what are we are going to do is; we have all three components in our network. We have homophily, we have closure and we have social influence. We are going to put all of these three together and we are going to visualize our network. So, let us take time for t in range; let us do it over a time period of; let us do it for 10-time instants. So, for t in range 0 to 10; what are we going to do is, we do homophily we apply (Refer Time: 07:00) closure, we apply social influence and then we visualize this graph and add the beginning when nothing is done that time also we want to visualize our graph. So, we have the statements here; now let us execute our code and see what is happening.

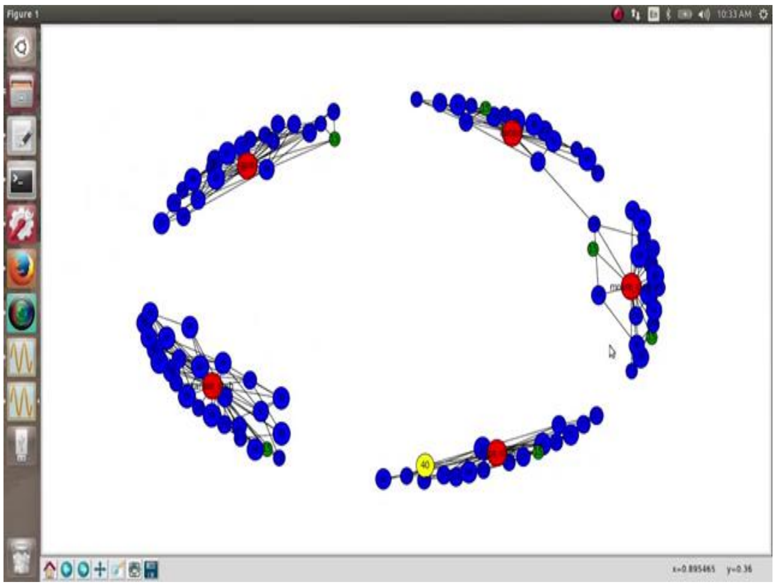
(Refer Slide Time: 07:16)



```
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
^C^X^Z
[1]+  Stopped                 python evolutionary_model.py
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
  File "evolutionary_model.py", line 121
    change_bmi(G):
                 ^
SyntaxError: invalid syntax
yayati@yayati-Vostro-3546:~$
```

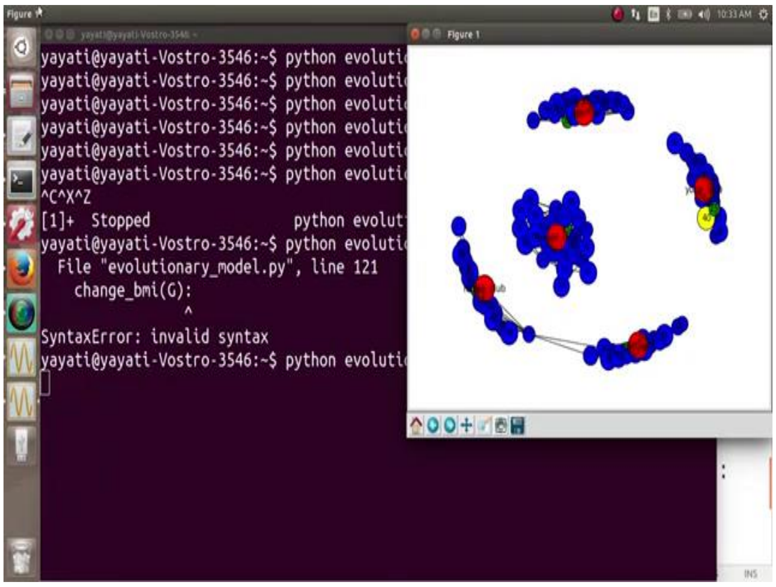
change_bmi(G); so, this is a function definition; so, we need a define here; go back and secure it. So, this is our initial network where we have five social focis and we have different nodes; I mean different BMIs.

(Refer Slide Time: 07:31)



Now, we see here that some nodes have become friends with each other and there are more number of edges between these nodes and let us look at the number of obese people in this network, so currently we have one obese node which is 40.

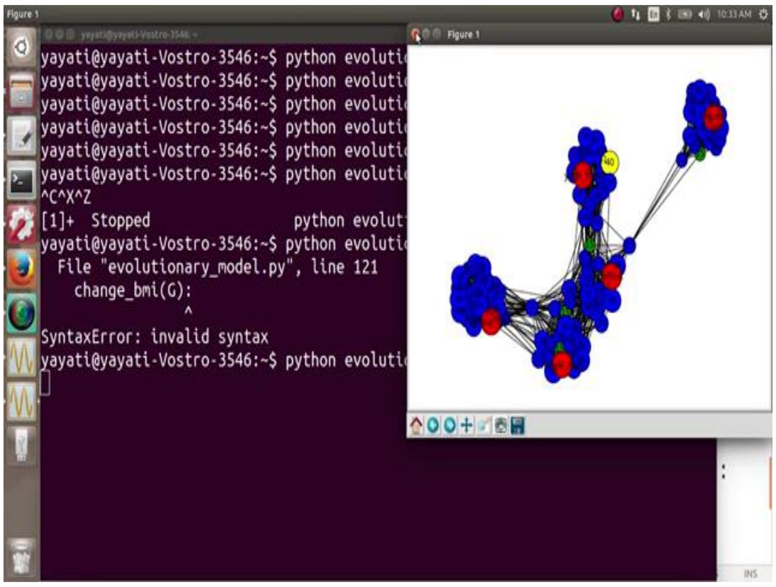
(Refer Slide Time: 08:00)



```
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
^C^X^Z
[1]+  Stopped                 python evolut
yayati@yayati-Vostro-3546:~$ python evoluti
  File "evolutionary_model.py", line 121
    change_bmi(G):
                 ^
SyntaxError: invalid syntax
yayati@yayati-Vostro-3546:~$ python evoluti
```

We again have one obese node; which is 40 and is a greater number of nodes are getting added; Do you see some problem in this code? I see some problem in this code.

(Refer Slide Time: 08:10)



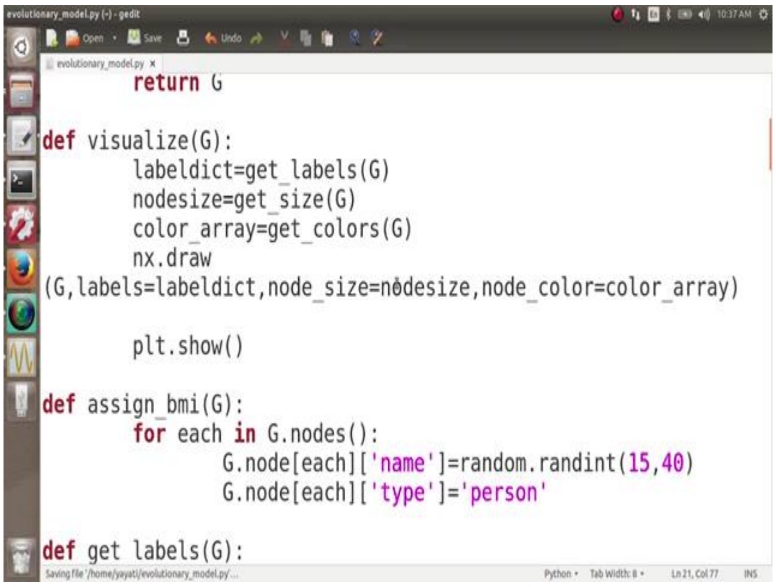
```
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
^C^X^Z
[1]+  Stopped                 python evolut
yayati@yayati-Vostro-3546:~$ python evoluti
  File "evolutionary_model.py", line 121
    change_bmi(G):
                 ^
SyntaxError: invalid syntax
yayati@yayati-Vostro-3546:~$ python evoluti
```

So, as we can see the number of obese people is now changing in this network that should actually happen because we have implemented social influence. So, what can be the problem with this code, so let us try to troubleshoot it together when we write a piece of code, we should be able to troubleshoot this code nicely? So, you will see here when we were visualizing a graph G here; we have passed these various arrays here; color array, node size, label dict. So, here when we have here when the BMI of people have changed and then the social influence has occurred. Do not you think we need to again calculate all these values?

So, it is like in the beginning we saw which nodes were yellow, we faded it here and we visualize we fed it here and we visualized it and now when we are running this graph; we are again and again using the same color array, the same size array; so, we need to change this. So, best way to change is to have these values; took all these functions inside our visualize.

So, this functions like let us say labeldict function and then the node size function and the color array function. So, all these three things need to be calculated in the visualize function because whenever you visualize it, these values are keeping, these value that changing. So, what we do is we just delete these values from here and we put all these values inside all these function calls; inside a visualize function. So, here is our function visualize; so, what we do here is we have all these values here.

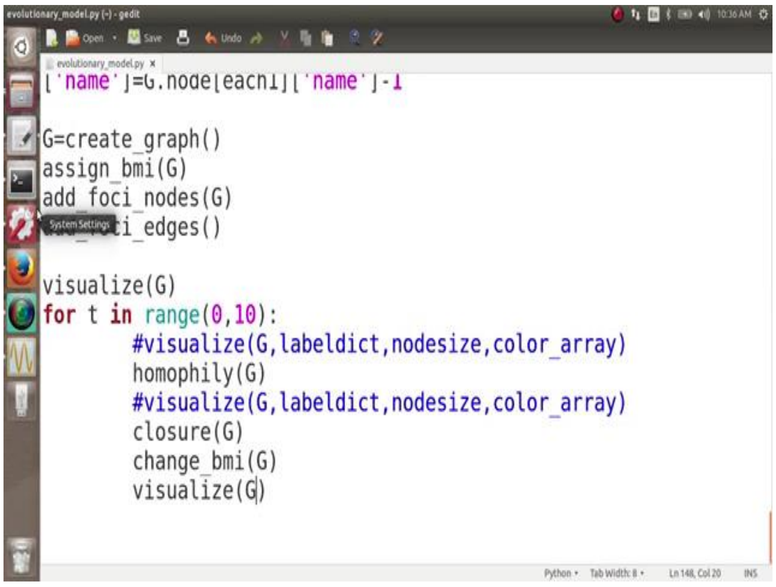
(Refer Slide Time: 10:04)



```
        return G

def visualize(G):
        labeldict=get_labels(G)
        nodesize=get_size(G)
        color_array=get_colors(G)
        nx.draw
(G,labels=labeldict,node_size=nodesize,node_color=color_array)

        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):
```

So, whenever we want to visualize a graph; we again look at the labels, we again look at the size of nodes and we again look at the colors and then we do not need to have all these things here, we just have a define visualize(G) and when we call these function towards the end, there also we just need a visualize(G). So, whenever you will try to code sometimes when weird things will happen. So, the best ways to keep printing your different parameters at different places and see what is going wrong where.

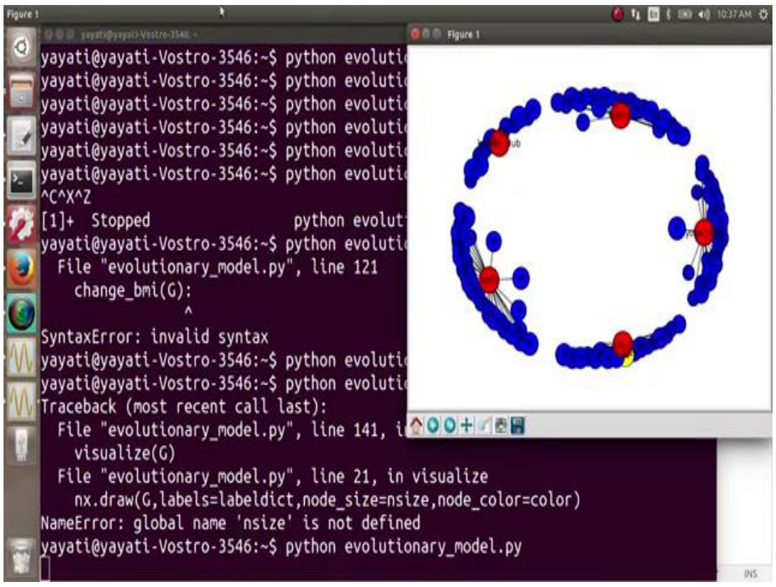
(Refer Slide Time: 10:32)



```
['name']=G.node[each]['name']-1

G=create_graph()
assign_bmi(G)
add_foci_nodes(G)
      ci_edges()

visualize(G)
for t in range(0,10):
        #visualize(G,labeldict,nodesize,color_array)
        homophily(G)
        #visualize(G,labeldict,nodesize,color_array)
        closure(G)
        change_bmi(G)
        visualize(G)
```

Now, let us again execute it.

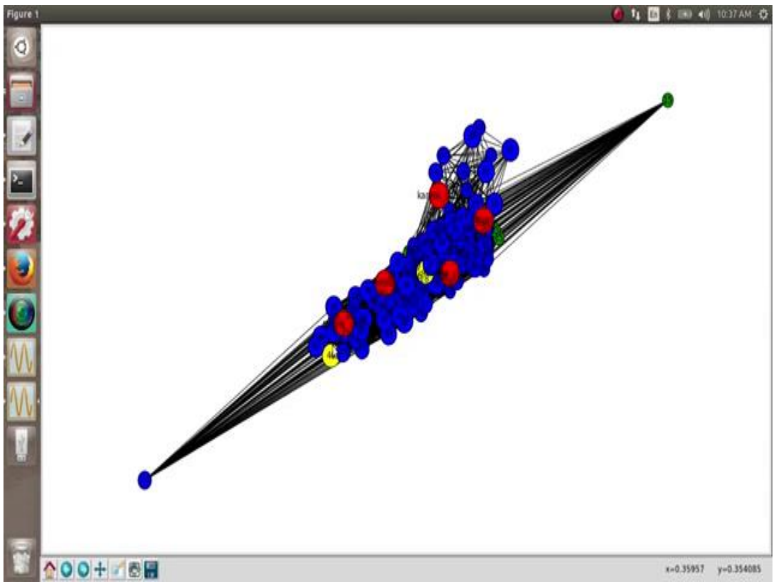
(Refer Slide Time: 10:21)



```
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
^C^X^Z
[1]+  Stopped                 python evolut
yayati@yayati-Vostro-3546:~$ python evoluti
  File "evolutionary_model.py", line 121
    change_bmi(G):
                 ^
SyntaxError: invalid syntax
yayati@yayati-Vostro-3546:~$ python evoluti
yayati@yayati-Vostro-3546:~$ python evoluti
Traceback (most recent call last):
  File "evolutionary_model.py", line 141, i
    visualize(G)
  File "evolutionary_model.py", line 21, in visualize
    nx.draw(G,labels=labeldict,node_size=nsize,node_color=color)
NameError: global name 'nsize' is not defined
yayati@yayati-Vostro-3546:~$ python evolutionary_model.py
```

Line 21; there seems to be a problem, so let us go to the line 21, so we have calculated labeldict here, we have calculated node size here. So, instead of n size here; it should node size and then node under scroll color is color under scroll array node color. Now, let us look at this graph, so this is our initial graph and then you see some 2 friendships have been added to this graph.

(Refer Slide Time: 11:56)

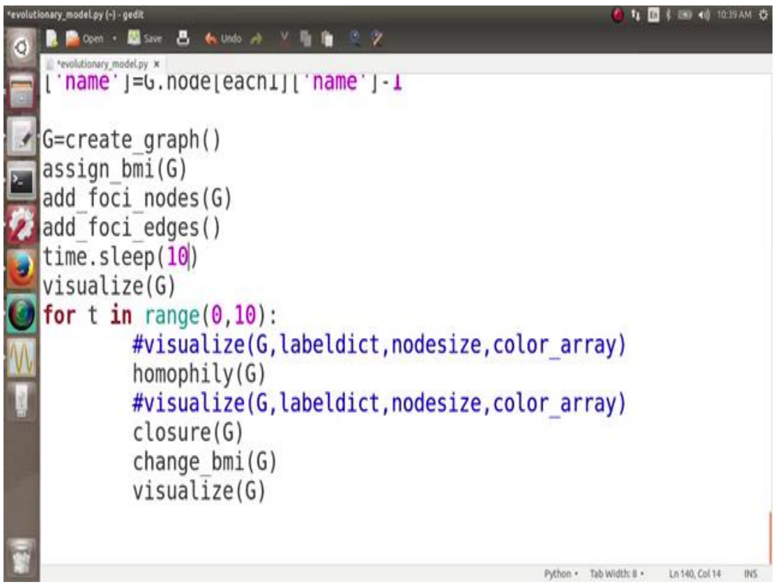


So, now you see there is one yellow node and there is another yellow node, so you see that the BMI of one node has been increased and it is hit the obesity parameter. So, this how a graph will keep changing; to have a battle visualization, we can actually change the parameters, we have used in the code; which you will see that with time more and more number of people will keep becoming obese because of social influence and yes people will use their weights also because of this.

So, let us quickly have a look on the parameter, so, this was for homophily and then this is for closure. So, let us keep it small for the time being let us keep it 0.01 and then we can again visualize this graph.

Now, let us visualize this graph in a very nice way, so what we are going to do for this visualization is we store this graph in a file jpg file and when we run this code at jpg file, we will keep changing with time and we observe that change in that jpg file. So, for that what do we do is; when you call this function visualize(G), this function will store your graph; will save your graph as a jpg file; instead of plotting it here.

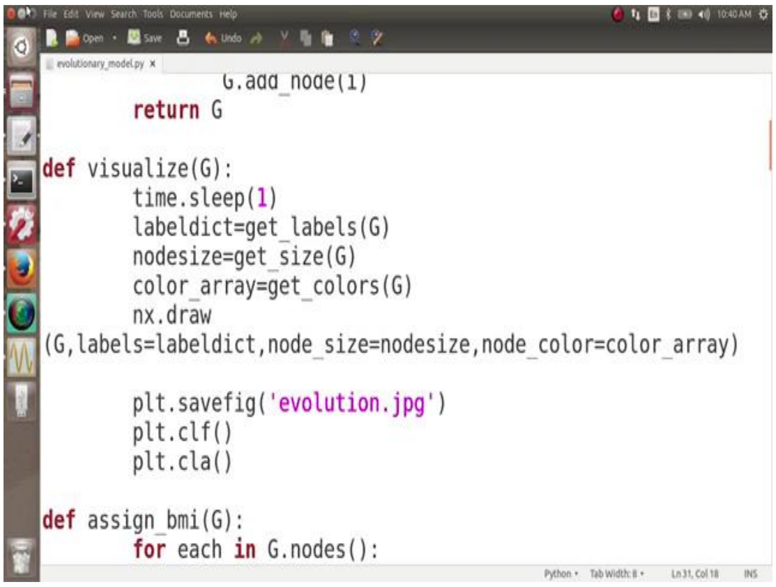
(Refer Slide Time: 13:19)



```
['name']=G.node[each1]['name']-1

G=create_graph()
assign_bmi(G)
add_foci_nodes(G)
add_foci_edges()
time.sleep(10)
visualize(G)
for t in range(0,10):
        #visualize(G,labeldict,nodesize,color_array)
        homophily(G)
        #visualize(G,labeldict,nodesize,color_array)
        closure(G)
        change_bmi(G)
        visualize(G)
```

Now, first let us put a sleep statement here so that before making your file, the code sleeps for some time.

(Refer Slide Time: 13:39)

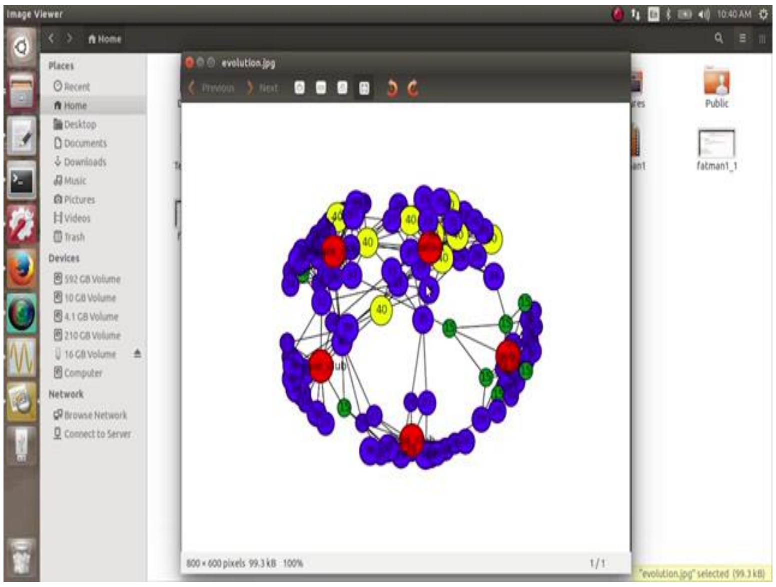


```
                G.add_node(1)
        return G

def visualize(G):
        time.sleep(1)
        labeldict=get_labels(G)
        nodesize=get_size(G)
        color_array=get_colors(G)
        nx.draw
(G,labels=labeldict,node_size=nodesize,node_color=color_array)

        plt.savefig('evolution.jpg')
        plt.clf()
        plt.cla()

def assign_bmi(G):
        for each in G.nodes():
```

So, let us say some 10 seconds and whenever you call your function; visualize(G), what this function is going to do is; it will wait for let us say 1 second and for this, use this sleep statement we need to import time. It may put some delays in your code, so time.sleep(1) and here instead of showing our plot, what do we do is; we save our figure plt.savefig and let us say evolution.jpg.

So, this plot is saved and once this plot is saved; we need to clear this plotting screen for the next plotting to occur. So, what we have done is; we have removed the plt dot show statement here and whatever the output graph is, we are having that output graph in a separate file and that separate file keeps changing with time.

(Refer Slide Time: 14:50)



Let us execute this code now and see what is happening. So, let us come back, so here we are and let us wait. So, here is our file evolution dot jpg let us open it and you can see that now this code has started changing with time. Now this figure has started changing with time and you can see that how more and more number of yellow nodes are popping up in the network; how more and more number of green nodes are popping up in the network, which shows that with time the number of obese people in this network is increasing and with time the number of underweight people in this network is also increasing, so it mainly depends on how you execute this code.

So, this was one way of looking at this animation; there actually milli nines python functions to look at this animations. So, for the sake of simplicity we have done it like this time, but next time we will try to use a better approach, we will try to use a inbuilt functions of python.