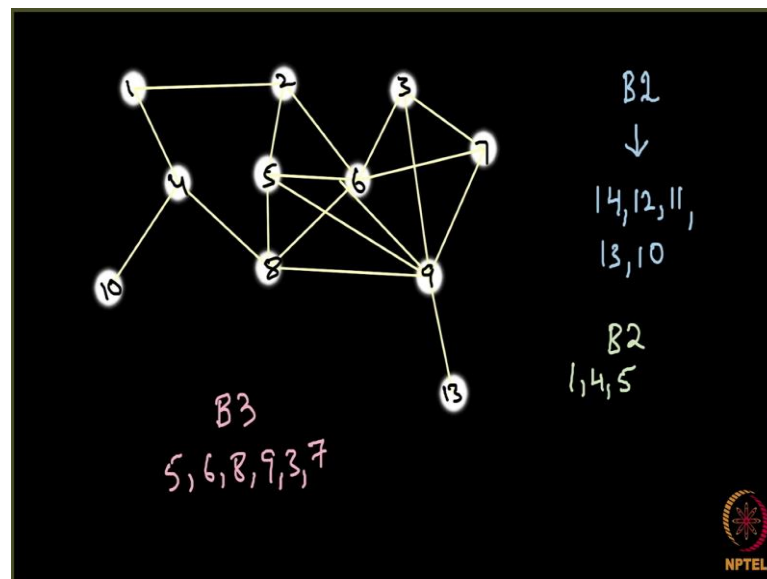


**Social Networks**  
**Prof. S. R. S. Iyengar**  
**Department of Computer Science**  
**Indian Institute of Technology, Ropar**

**How to go Viral on Web**  
**Lecture - 163**  
**Coding K-Shell Decomposition**

(Refer Slide Time: 00:05)



In the next programming screen cast we are going to implement facial Decomposition. So, I hope you remember you remember what was the algorithm for facial decomposition. First of all, we take bucket 1 and then after taking bucket 1 we start removing the nodes having degree 1.

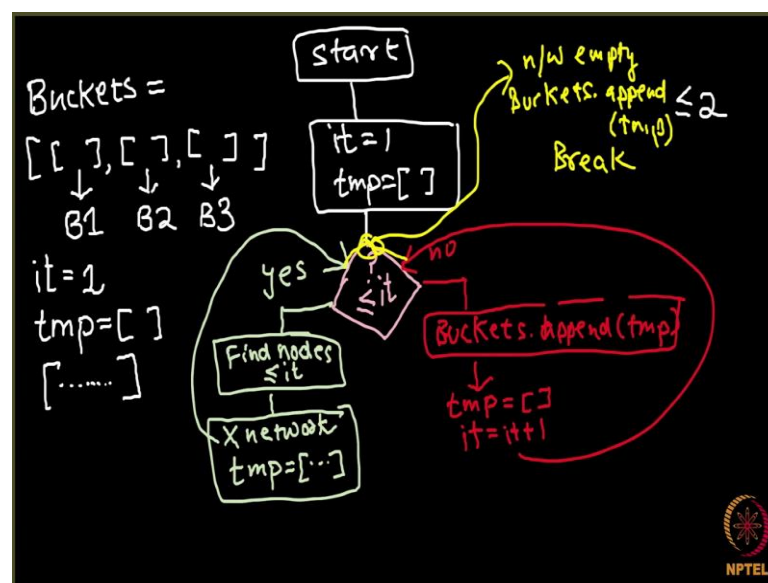
So, we see that this node over here these nodes have degree 1, this node has degree 2. So, these 3 nodes have degree 1, so what we are going to do, we are going to remove these 3 nodes and put them in bucket 1. So, let me number these nodes also. So, what I am going to do is let me number these nodes first ok. So, we are going to take bucket 1 and in bucket 1 will recursively put the nodes who have degree 1.

So, first of all nodes 14, 12 and 11 go to bucket 1 because, they have degree 1 and then we delete these nodes from this network. So, we cut node 14, 12 and 11 from this network and we put them. And next again we see that there is degree 1 in this network and then we recursively remove these nodes also from the network and put them in

bucket 1; similarly, we determine bucket 2 and bucket 3 in this network. So, bucket 2 will comprise of the nodes 1, 4 and 5 and bucket 3 will comprise of the nodes 5, 6, 8, 9, 3 and 7.

So, how do we write a code for this? So, before writing the code here I am going to tell you the structure of the code, so that it will be easy for you. Decode is a little bit involved, so it is important to understand its structure before writing the piece of code ok. So, let us take this network and we do not even need this network ok.

(Refer Slide Time: 02:15)



So, what we will be doing is let us say that here a good start. So, this is starting. What we are going to do is we know that we have to recursively remove the nodes having degree 1 or less than 1 till there are no degree 1 nodes in the network. Next, we have to recursively remove the nodes having degree less than or equal to 2 from the network till there are no nodes having degree less than or equal to 2 in the network so on and so forth.

So, in the beginning I am going to take some variables. So, I have a first of all I will take a list here and I name this list as buckets. What is this list bucket? This list buckets is going to be a lists of lists. So, it will be further consisting of many lists and this will be the list corresponding to my bucket 1. This will be my list corresponding to bucket 2, bucket 1 here means bucket corresponding to pour 1, bucket corresponding to pour 2, and bucket corresponding to pour 3.

So, this is my array buckets and the aim of our code is to fill this array buckets. And then I am going to take a variable let us say it which is initially 1. What this variable tells me? This variable tells me at a particular time in a code which degree nodes I am removing. So, if the value of it is k let us say it means that I am removing the nodes having degree less than or equals to k.

So, that is my variable it. And then we have a array or we have a list tmp. What this list tmp is going to hold is, so we are going to determine 1 of bucket at a time. So, I might be filling this bucket at a particular time, I might be filling this bucket or this bucket. So, t is going to temporarily hold this bucket. So, initially the value of tmp is going to be initially, so tmp will start with empty and then whatever values will put in tmp will be the values in B1.

So, we will fill tmp, we will keep filling tmp and a point will come where all the nodes have been degree less than or equals to 1 will be removed from this network. At that time we are going to close tmp and we are saying that we will say that this tmp is nothing but bucket 1 and we will append tmp here, it will be further clear ok.

So, what we are going to do is we have a start here. After start we are going to check a condition. So, initially we have it equals to 1 and tmp is nothing, but empty ok. Next what we are going to do? We are going to check a condition and what is this condition? This condition is whether there are nodes of degree greater than sorry, whether there are nodes of degree less than or equal to it; in this network whether, there are such nodes in the network and the answer can be yes or no.

So, let us say here the answer is yes and here the answer is no. So, if the answer is yes what we are going to do? If the answer is yes it means that, still there are nodes of degree less than or equals to it in this network. So, we have to prune those nodes from the network. So, what we will be doing is what we will do is find those nodes.

So, we find these nodes which have the degree less than or equals to it and what do we do? We remove these nodes from our network. So, we remove these nodes from our network and we append these nodes in our array tmp, in our list tmp which is the currently being filled bucket.

So, we append these nodes in tmp. That is what we do and after doing this we return back to here. So, when the value of it was 1 here what do you do, you find the nodes which have degree less than or equals to 1, you delete those nodes from the network and put them in your bucket and after that you again see whether you; I am very sorry, you come here you come here ok.

So after that, after removing these nodes having degree 1, you again see whether still there are nodes having degree less than or equal to 1 in your network and again if the answer is yes you remove those nodes also and you keep doing this till there are no nodes having degree less than or equals to 1 in the network. So, if the answer is no what will you do? It says that your bucket is over.

The bucket which you are currently filling is over, so what you do? At this point if the answer is no you append your tmp to your buckets. So, `buckets.append(tmp)` that is your current bucket is full, so you do this and what else you do. Now we have to start filling a new bucket. So, we set back tmp to empty and also, we have removed all the nodes having degree it. Now we have to start removing the nodes having degree it plus 1. And after doing this we go back to these things.

So, I hope the code structure is clear now. So, this loop will keep running. So, it will check for the value of it equals less than or equal it will be 1. So, we check whether there are nodes of degree less than or equals to 1. We keep doing this, we keep coming back, keep removing nodes having degree less than or equals to 1 till there are no such nodes in the network. And when there are no such nodes in the network, we change it to 2 and then we check the nodes having degree less than or equals to 2, we will keep doing it.

Now do you see this code will keep running forever, there is some problem with this code. At one time all the nodes from this network will be removed and that is the time where we should stop. So, what we are actually going to do is before coming to this step, before coming to this step and before checking whether there are such nodes in the network we can actually put a small condition here.

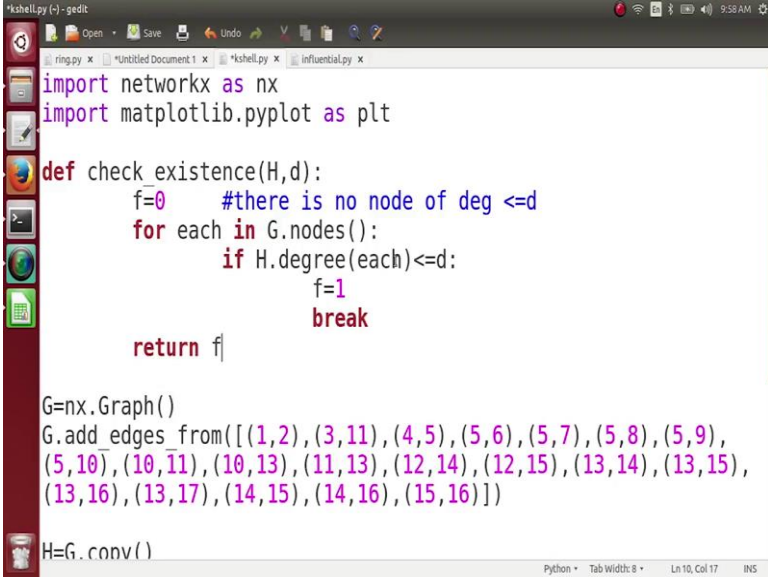
So, before doing this we can actually put a small condition there and this condition will check whether my network is empty, whether my network has become empty. Whether I have removed all the nodes from this network and if I have removed all the nodes from this network what I will do? I will simply append my temp to buckets.

So, at this time we will append tmp to buckets `buckets.append(temp)` and you break. Now we will be implementing this and I am sure this will be further clear when you look at the code, you might want to keep this in front of you while writing the code which will make your process easier ok. So, next we are going to write the next we are going to see the code for facial decomposition and please note we need a little bit more complicated network.

So, for doing facial decomposition and for all the further codes we will be going to use this particular network which is the slightly a little bit complicated. So, here if you look at the nodes present in nodes which will be removed in the bucket 1, so in bucket 1 these are the nodes which should be removed. So, first you remove this node 2 and 2 and all these nodes 4, 8, 6, 7, 9 and 17 over here and 3 over here and in the next iteration you will remove one and five as well and then your bucket 1 will be over.

And then in your bucket 2 will be going to remove these nodes 11 over here and 12 over here. And then once you remove 11, you will have to remove this node 10 over here also. So, these nodes should be in your bucket 2 and finally, your bucket 3 should comprise of these 4 nodes over here. So, we will be using this network for our programming screen casts. So, now, we are going to write the code for facial decomposition.

(Refer Slide Time: 11:13)



```
*kshell.py (-) - gedit
import networkx as nx
import matplotlib.pyplot as plt

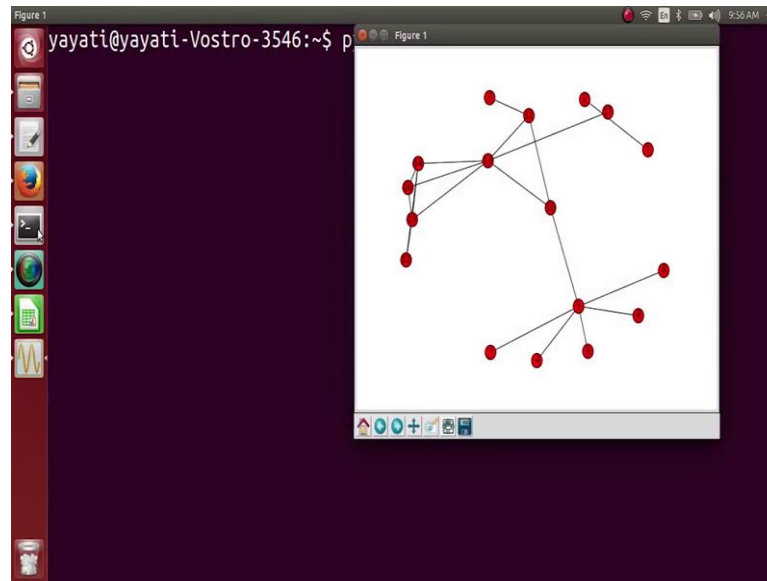
def check_existence(H,d):
    f=0 #there is no node of deg <=d
    for each in G.nodes():
        if H.degree(each)<=d:
            f=1
            break
    return f

G=nx.Graph()
G.add_edges_from([(1,2),(3,11),(4,5),(5,6),(5,7),(5,8),(5,9),
(5,10),(10,11),(10,13),(11,13),(12,14),(12,15),(13,14),(13,15),
(13,16),(13,17),(14,15),(14,16),(15,16)])

H=G.conv()
```

So, as usual we import network x as nx and we import matplotlib pyplot as plt. What do we do next is we create a graph  $G = nx$  graph and we add some edges to this graph? So, this graph which I am going to take is the same graph which we have seen previously in this video. So, the graph which I am going to take is this. You can actually verify and see that it is the same graph. So, we can also draw it and see `nx.draw(G)` and then `plt.show` and then we can execute and see it.

(Refer Slide Time: 12:02)



So, this is the network over here and we have to look at various shells in this network ok. What we do? Next is we have already discussed what we are going to do is first of all we are going to create a copy of this network  $H = G.copy$  and whatever we are going to do? We are going to do it with  $H$  and then we take our variable iteration corresponding to the degree which is initially 1. And then we have a list  $tmp$  corresponding to the bucket which we are filling currently.

So, it is corresponding to the let me write it down here. It is for the bucket being filled currently ok. And then we have our overall added bucket which is going to be a list of lists and each list here is a bucket ok. Next what we are going to do is we are going to run a while loop while 1.

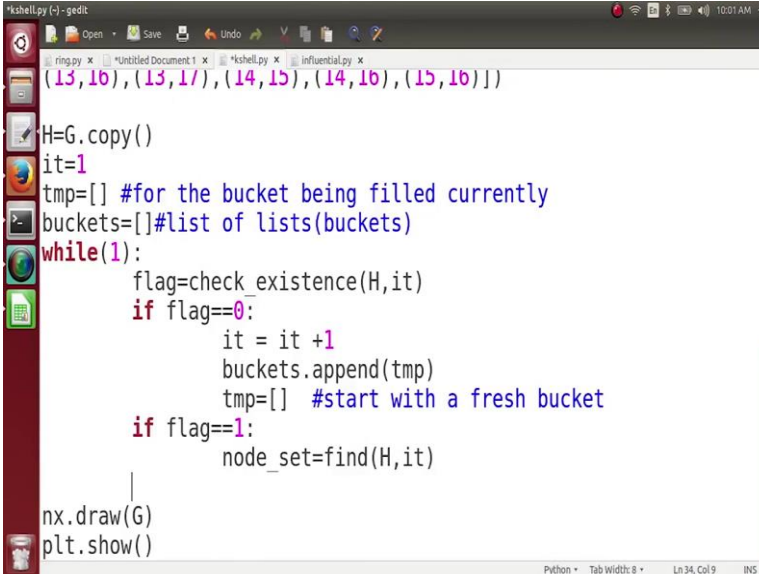
So, this loop will keep running till a terminating condition comes and Now, we put a check condition whether there are nodes of degree less than or equals to it in this network. So, for that I use a function and let us call this function as check existence. So, flag equals to check existence and what do I pass here a graph  $h$  and the degree. And let us define this function first here. Define check existence, where I have passed the network  $H$  and the particular degree and how do I define it? It is very simple.

So, what do I do? Initially I said  $f$  equals to 0 which means that  $f$  is going to be 0 till the end of this function if in the case there is no node of degree less than or equals to  $d$  in the network right. And then what we are going to do for each in  $G.nodes$  we check each and

every node in this network and as soon as we find a node in this network whose degree is less than or equals to  $d$ , it means that still there is a node having degree less than or equals to  $d$  in the network immediately we set  $f$  to 1 and break this loop right and then we return  $f$ .

So, this  $f$  is going to be 0 till the end of this code if there is no node of degree less than equals to  $d$  and it is 1 if there is even 1 node of degree less than or equals to  $d$  ok.

(Refer Slide Time: 14:53)



```
*kshell.py (-) - gedit
(13,16),(13,17),(14,15),(14,16),(15,16)])
H=G.copy()
it=1
tmp=[] #for the bucket being filled currently
buckets=[]#list of lists(buckets)
while(1):
    flag=check_existence(H,it)
    if flag==0:
        it = it +1
        buckets.append(tmp)
        tmp=[] #start with a fresh bucket
    if flag==1:
        node_set=find(H,it)
nx.draw(G)
plt.show()
```

So,  $flag$  equals to check existence  $h$  comma  $it$ . And next what we have seen,  $flag$  can be 0 or 1. If  $flag$  is 0, it means that we have looked at all the nodes having degree less than or equals to  $it$ , even recursively we have seen we have pruned again and again and there is no node of degree less than or equals to  $it$ . In that case what we have to do? We have to mark our bucket as complete.

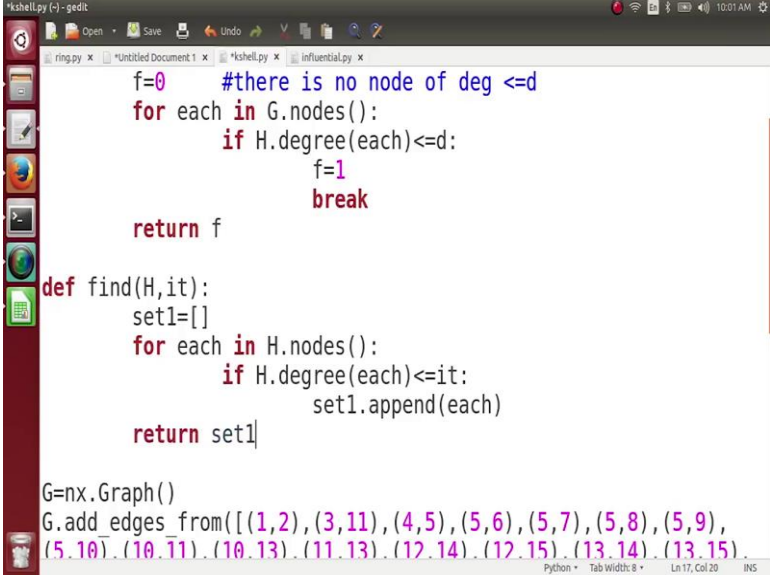
So, if  $flag$  is equals to 0, what do we do? We move on to the next degree, we set  $it$  equals to  $it$  plus 1 and we say that the current bucket is complete. So, we append our temp to buckets, mark it as complete and now we start with a fresh bucket.

So, this is nothing, but start with a fresh bucket right ok. And in another case if  $flag$  was 1, what we are going to do? If  $flag$  is one it means that we have to keep pruning the nodes having degree less than or equals to  $it$ . In that case first of all we should find out such nodes in the current network we have to find the nodes which are having degree



less than or equal to it. Let us call a function for that and let us name this function as find and so in the graph H finally, nodes having degree less than or equals to it and we can quickly write code for this function define find h comma it and how do I define it.

(Refer Slide Time: 16:26)



```
*kshell.py (-) - gedit
ring.py x  Untitled Document 1 x  *kshell.py x  influential.py x

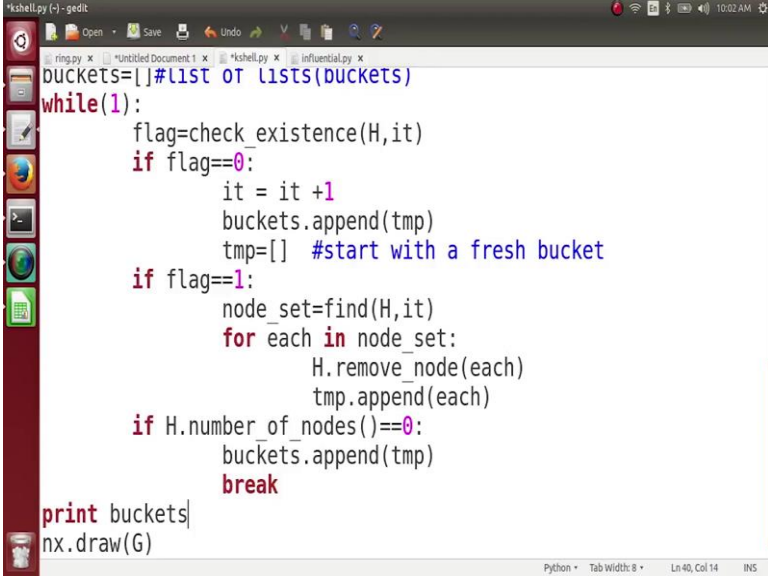
f=0 #there is no node of deg <=d
for each in G.nodes():
    if H.degree(each)<=d:
        f=1
        break
return f

def find(H,it):
    set1=[]
    for each in H.nodes():
        if H.degree(each)<=it:
            set1.append(each)
    return set1

G=nx.Graph()
G.add edges from([(1,2),(3,11),(4,5),(5,6),(5,7),(5,8),(5,9),
(5,10),(10,11),(10,13),(11,13),(12,14),(12,15),(13,14),(13,15)]
```

So, initially this set is empty, and it is simple. For each in H both nodes if H.degree(each), if H.degree of H is less than or equals to it what we are going to do is set 1.append each and then return set 1 right. So, we have find out we have found out all the nodes having degree less than or equals to it. Next, we have to prune these nodes on the network and add them to our bucket. So, what do we do? Prune these nodes from our network and add them to the bucket.

(Refer Slide Time: 17:23)



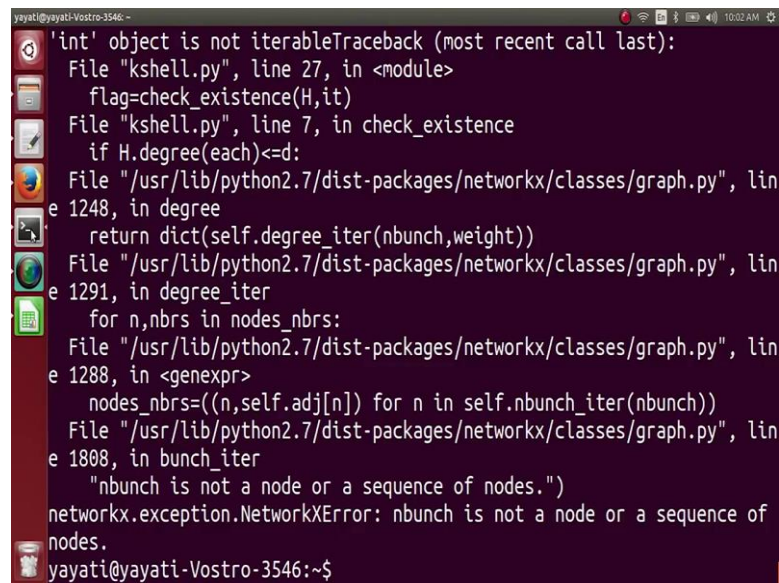
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

buckets=[]#list of lists(buckets)
while(1):
    flag=check_existence(H,it)
    if flag==0:
        it = it +1
        buckets.append(tmp)
        tmp=[] #start with a fresh bucket
    if flag==1:
        node_set=find(H,it)
        for each in node_set:
            H.remove_node(each)
            tmp.append(each)
        if H.number_of_nodes()==0:
            buckets.append(tmp)
            break
print buckets
nx.draw(G)
```

So, for each in node underscores set what we are going to do is `H.remove_node(each)`, we have removed it from `H` and we have to put it in our current bucket which is being filled which is nothing, but `tmp.append(each)` right. In this case we will keep pruning the nodes and the last part of the code is a terminating condition. So, if by pruning these nodes our network it becomes empty what we have to do?

So, if network becomes empty is if `H.number_of_nodes` if it becomes equals to 0 what we have to do? What we have to mark the current bucket as filled. So, `buckets.append(tmp)` and then `break` ok. And now we want to see whether our buckets are correct or not. So, for that what we can do is we can simply print buckets. It tells us the different buckets in our network.

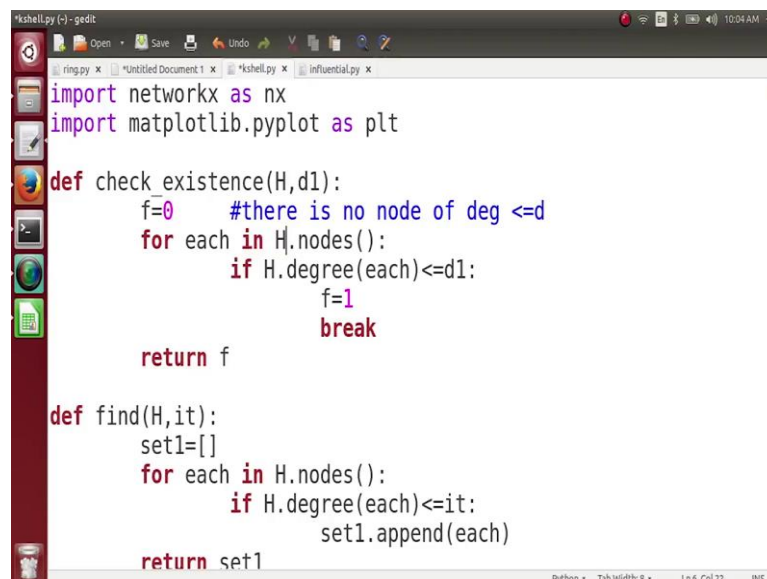
(Refer Slide Time: 18:44)



```
yayati@yayati-Vostro-3546:~$  
'int' object is not iterableTraceback (most recent call last):  
  File "kshell.py", line 27, in <module>  
    flag=check_existence(H,it)  
  File "kshell.py", line 7, in check_existence  
    if H.degree(each)<=d:  
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", line 1248, in degree  
    return dict(self.degree_iter(nbunch,weight))  
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", line 1291, in degree_iter  
    for n,nbrs in nodes_nbrs:  
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", line 1288, in <genexpr>  
    nodes_nbrs=((n,self.adj[n]) for n in self.nbunch_iter(nbunch))  
  File "/usr/lib/python2.7/dist-packages/networkx/classes/graph.py", line 1808, in bunch_iter  
    "nbunch is not a node or a sequence of nodes.")  
networkx.exception.NetworkXError: nbunch is not a node or a sequence of nodes.  
yayati@yayati-Vostro-3546:~$
```

So, let us execute this code and see. An adder at line number 27. Check underscore existence H comma it ok. If it was 0 for each in G.nodes, so if I do not think this (Refer Time: 19:20) ok. Some error let us try to figure it out, some very small error. So, it what is it here? It is a number right a particular degree d for each in let us use some other variable here d1 ok.

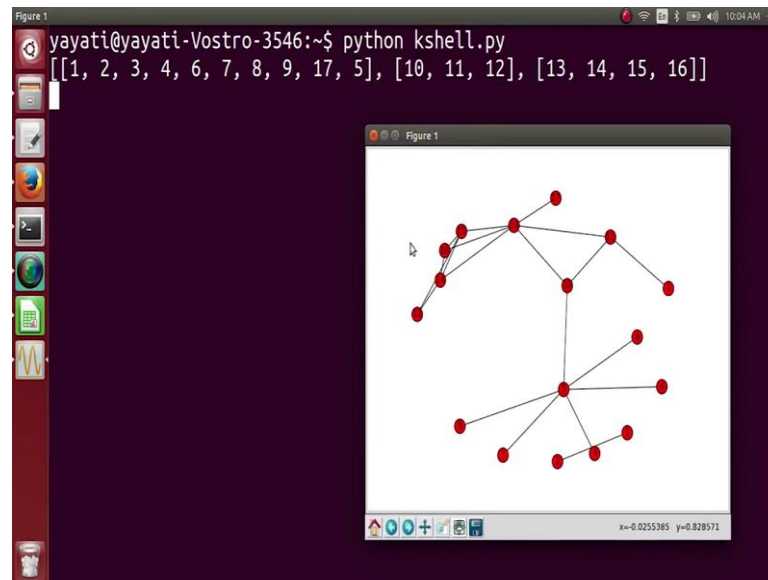
(Refer Slide Time: 20:11)



```
*kshell.py (-) - gedit  
import networkx as nx  
import matplotlib.pyplot as plt  
  
def check_existence(H,d1):  
    f=0    #there is no node of deg <=d  
    for each in H.nodes():  
        if H.degree(each)<=d1:  
            f=1  
            break  
    return f  
  
def find(H,it):  
    set1=[]  
    for each in H.nodes():  
        if H.degree(each)<=it:  
            set1.append(each)  
    return set1
```

See a mistake here; here we have used G.nodes, so here we have to use H.nodes because, if we are using G.nodes here we might here be talking about a node which is not there in H, H has less nodes than G So, it should be H.nodes here and let us execute it ok.

(Refer Slide Time: 20:32)



So, you see it here, your 1 core is the first bucket is from 1, 2, 3, 4, 6, 7, 8, 9, 17, second is this and third is this and that is the result.

So, we have implemented facial decomposition, if you want you can also color these nodes with different colors to see the differentials and so on.