INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

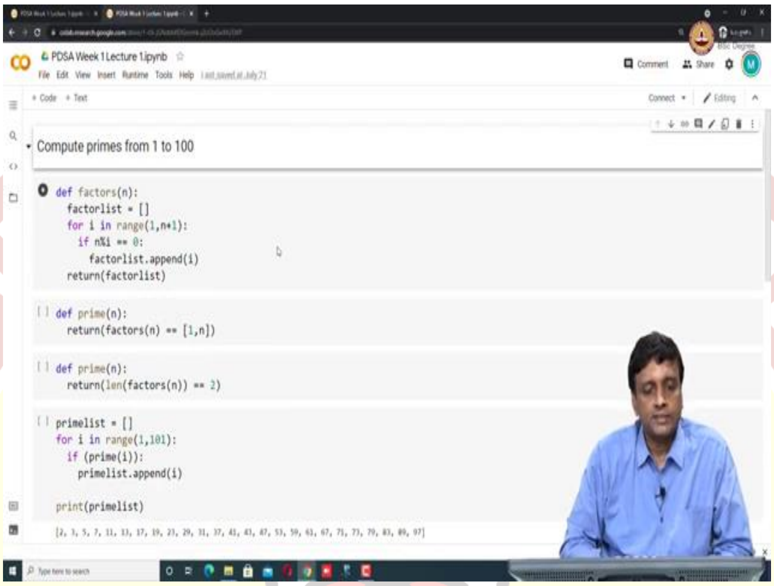
IIT Madras

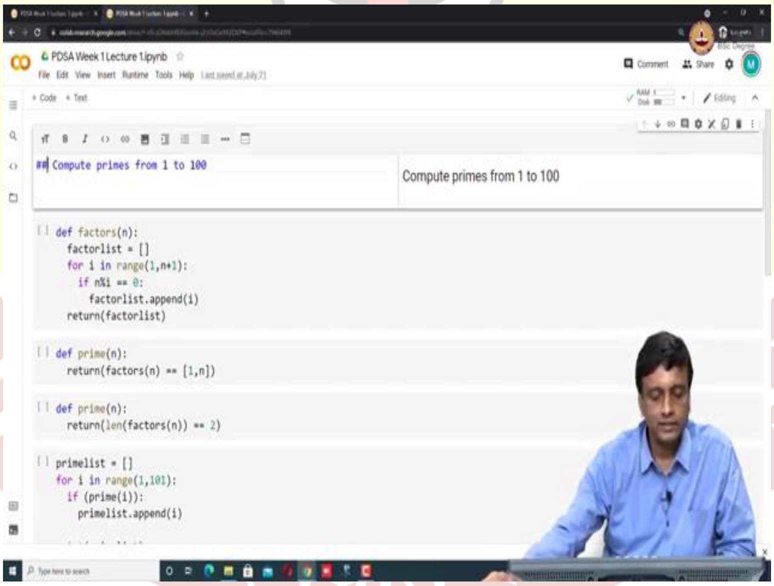
ONLINE DEGREE

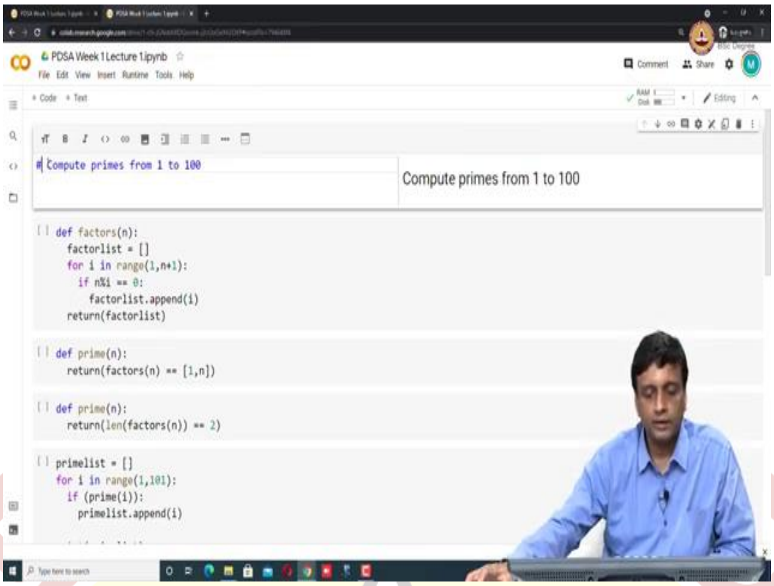
Programming Data Structures and Algorithms using Python
Professor Madhavan Mukund
Implementation of Python Codes – Part 01

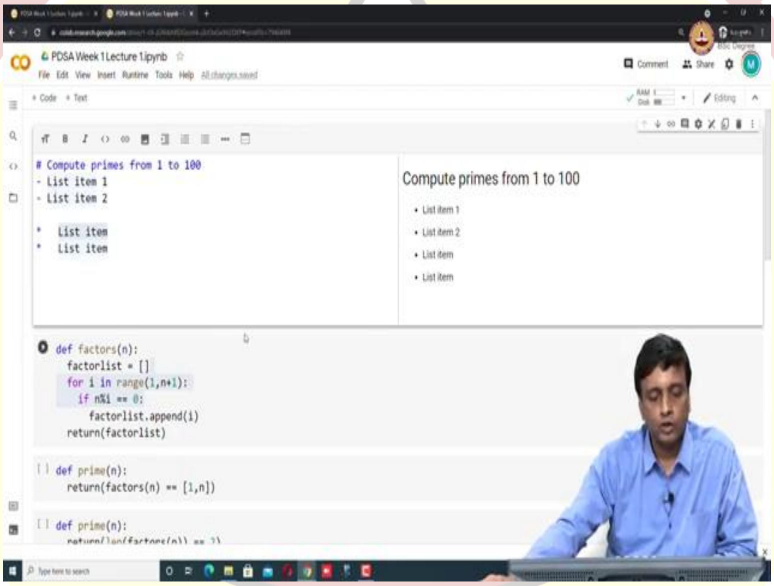
So, let us take a look at how this Jupyter notebook works in Colab.

(Refer Slide Time: 0:14)









So, if you go to this Colab url, colab.research.google.com and you have logged in, then it might show you what you have seen, done before or it will give you an empty menu and ask you whether you want to import something say you can take something from your Google Drive or you can upload something from your computer and so on.

So, here we have already, I have already done something, so let me just open something that we have, I have already used. So, this is the code which I was, had used as the backdrop of the slides of the first lecture. So, notice now that we have cells. So, we have a cell here, we have a cell here, we have a cell here and so on.

So, this is that one dimensional kind of spreadsheet structure that I talked about and if you look at the left of each cell there is a kind of bracket there, so this indicates whether cell has been run or not. So, at the moment none of these cells have been executed. The other thing is that we have this text over here and we have code over here, so these are these two different types of cell and Colab allows you to add new cells.

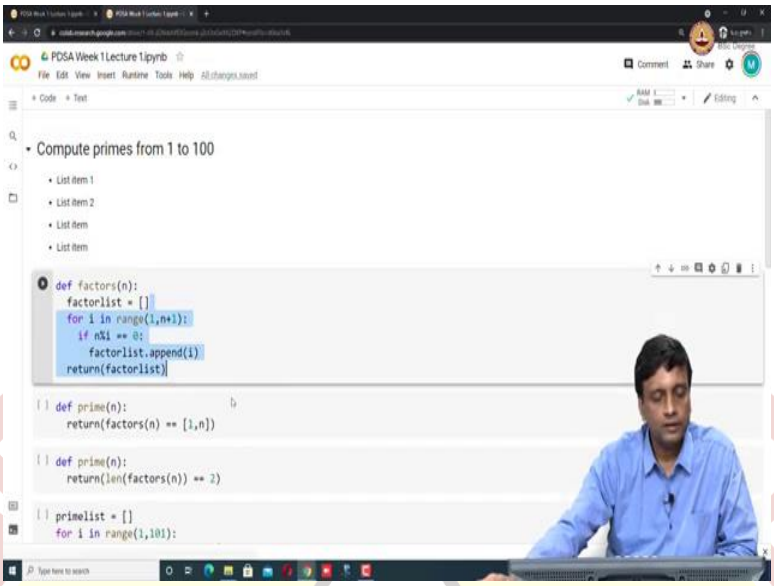
So, if I say plus code on the top it adds a cell in which I can type code and if I say plus text it will add a cell in which I can do documentation. So, now if I go to a text cell for example, and I edit it right then you see this Markdown syntax. So, this Markdown syntax is basically saying that this double hash is giving me this kind of a heading.

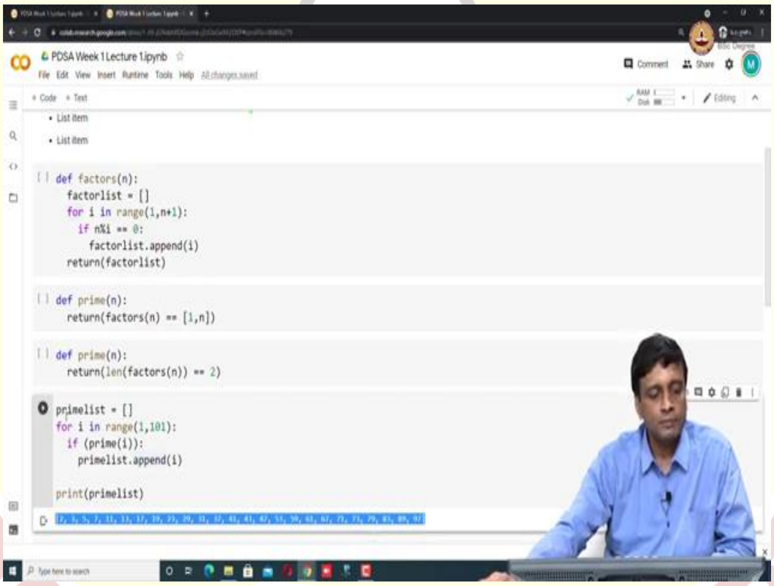
If I use a single hash, you will see that this thing will actually become bigger, so you notice that it actually displaying, Colab displays this, Jupyter notebook will not display, you have to execute the cell, but if I come out of the cell now it has become bigger. The other things I can do here for instance is I can kind of create, so I said that you can use text like formatting.

So, you can say, so you can use formatting like this, this hyphen and this will automatically come out as you can see on the right as some bulleted items in the list, so of course, now in this Colab environment this is not true in the normal Jupyter notebook that you would run on your laptop, it actually allows you to do this using this interface where you can say that I want to kind of add, say a bulleted item and so on.

So, you can actually do this directly, but normally you have to use this Markdown. So, it is still doing it in Markdown, as you can see it is using star, so minus gives you a bulleted item, star also so, Markdown is flexible, so you get this kind of text and it gives you documentation.

(Refer Slide Time: 2:56)





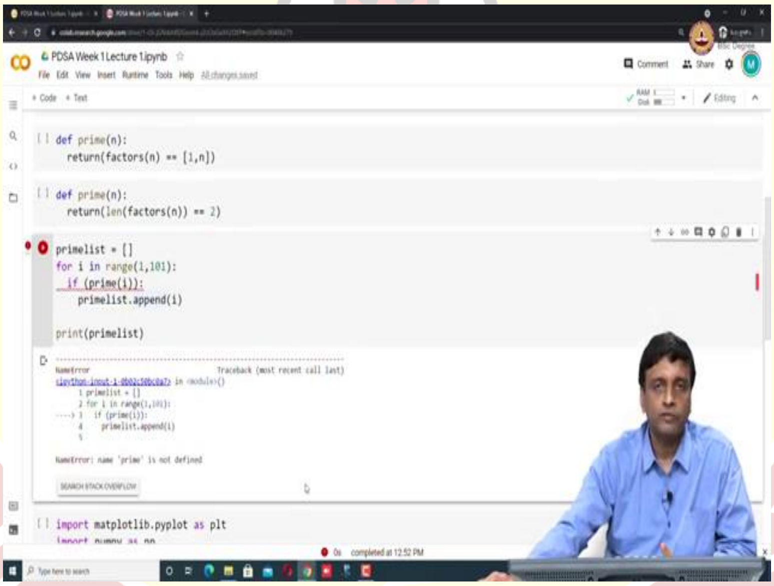
Now, when you come down to the code for instance, now at the moment none of these have been run. So, here I have, so this code let me explain what is doing. So, this first function factors is creating the list of factors of a number n, so it is running from 1 to n and it is checking all the numbers, which divide n and putting them into a list of factors. So, that is factors of n.

And then there are two definitions of prime, the first definition of prime checks whether the factors of n is precisely 1 comma n, the second definition of prime checks whether the length of this factors list is exactly 2, both are the same, because if there are exactly two factors there must be one comma n, but these are just two different to show you that you can have the same function defined in two different ways coexisting inside the notebook.

At the moment neither of these is active because I have not run anything. And finally here is a function a piece of code, it is not a function actually. It is a piece of code which actually computes all the primes from 1 to 100. So, it takes i in the range 100, and if i is a prime then it appends it to the prime list and finally it prints the prime list. Now, see that there is some output at the bottom. So, this code actually was run before.

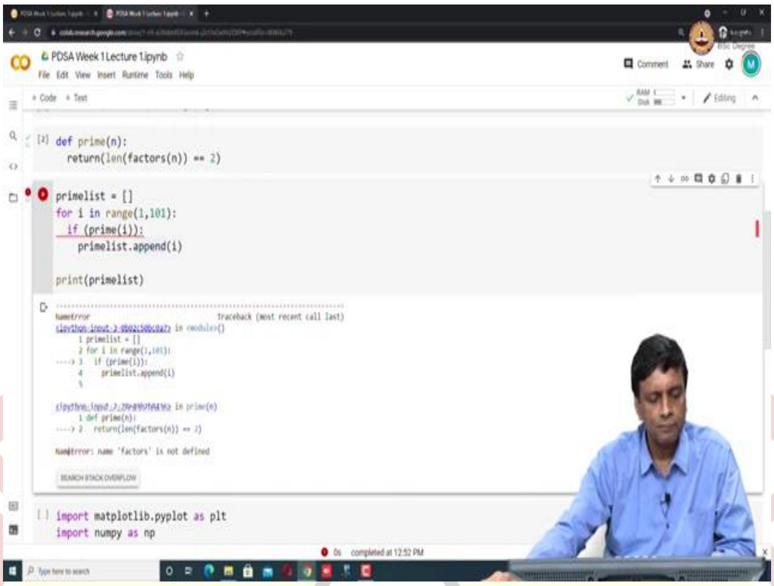
And this is what I meant earlier when I said you can export the notebook with the output, so even though I have not run the code even once, the output is there so that you can show it to somebody, saying this is what happened when I ran the code the last time right, so the output is preserved when you save this file or in Colab it is automatically saved like a Google document. So, when you reload it or you send it to somebody they will be able to open it and see the same output that you saw when you ran it.

(Refer Slide Time: 4:33)



So, now let us see what happens if I try to run this code. So, if I kind of hover over these things I get this run symbol, so if I run this code now it is going to complain because I have not run the code before it, so in the current environment the functions prime and factors which are used here are not there, so in particular I get the standard Python error which says name error, the name prime is not defined.

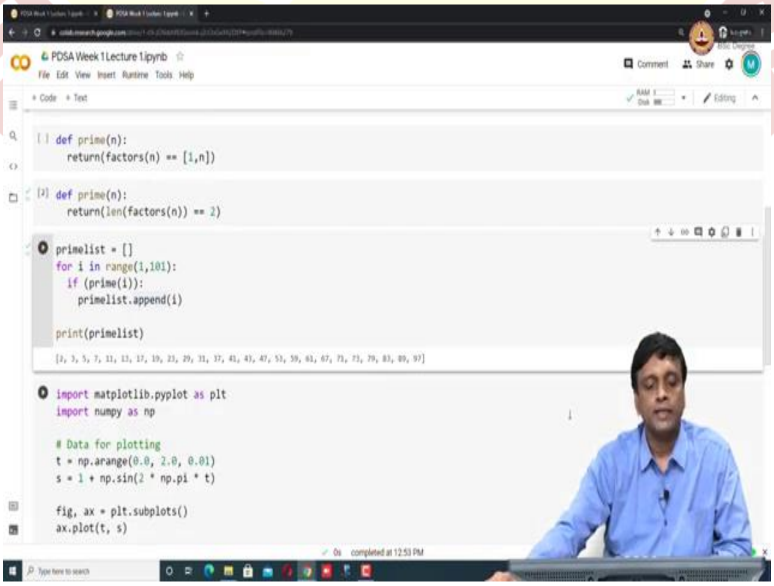
(Refer Slide Time: 5:00)



So, that means IS must go and define one of these two prime, so let me say take the second one. Now if I try to do that then it goes through, but now if I try to run this thing, now prime is defined but I am going to get a different error saying that inside prime the name factors is not defined. So, these are all things which are only available at runtime.

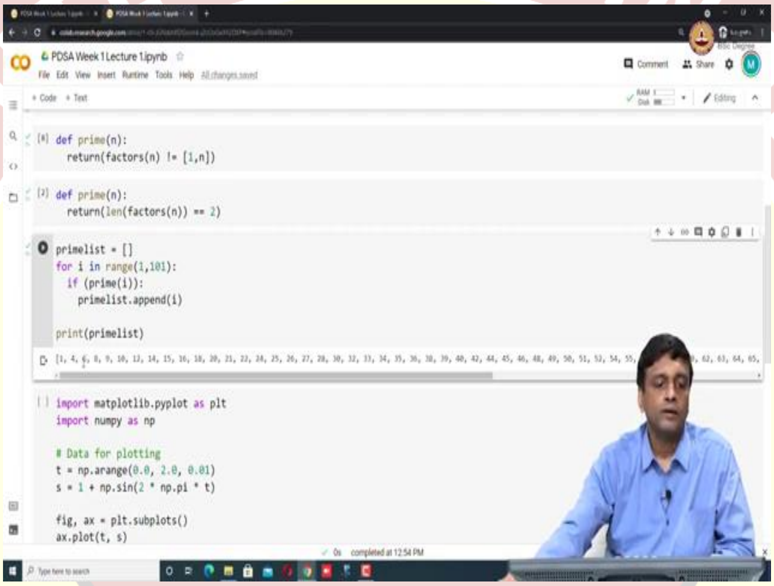
So, when I compile a function in some sense, when I run a function definition it does not check that those values at the function definition uses are defined because it will not know that, few factors could have been some kind of a global variable as far as the function is concerned, so it has no idea what to do with that factors.

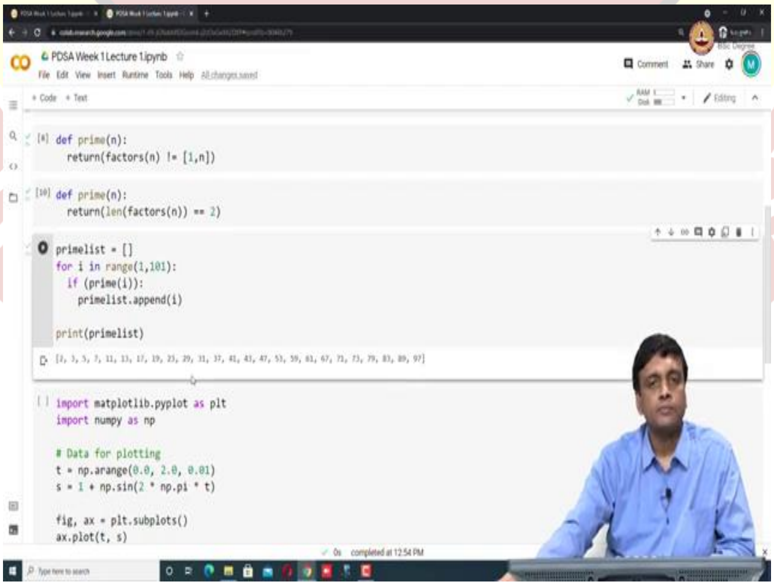
(Refer Slide Time: 5:37)



So, now that I know that it still does not run I go back and I run factors right and now that I have run factors this prime list will now run, now I can run prime list and this time it will execute and produce the same output that I saw before. So, two things to understand, one is that this output was there before, but these functions were not run and now if you look carefully, it is not easy to see, but there are numbers inside these boxes. So, it is, actually I executed these cells in a particular order, so there is some one, two, and this is called four, because I ran this a third time in between.

(Refer Slide Time: 6:16)





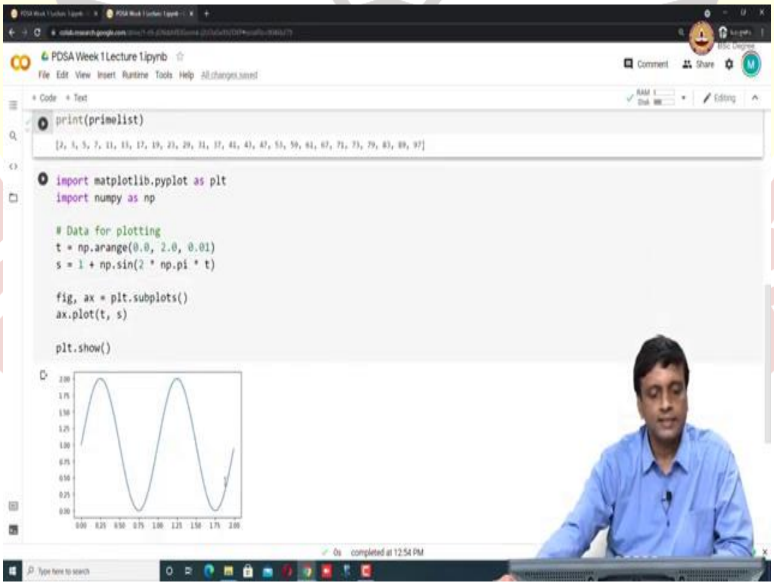
Now, I can now for instance take this function, this definition of prime and run it. What this does is it takes the previous definition of prime and replaces it by this definition. So, the most recently defined version of prime is available in manual, so if I run this code again, of course, in this case there is no big difference because it is going to be the same thing, is going to produce the same output.

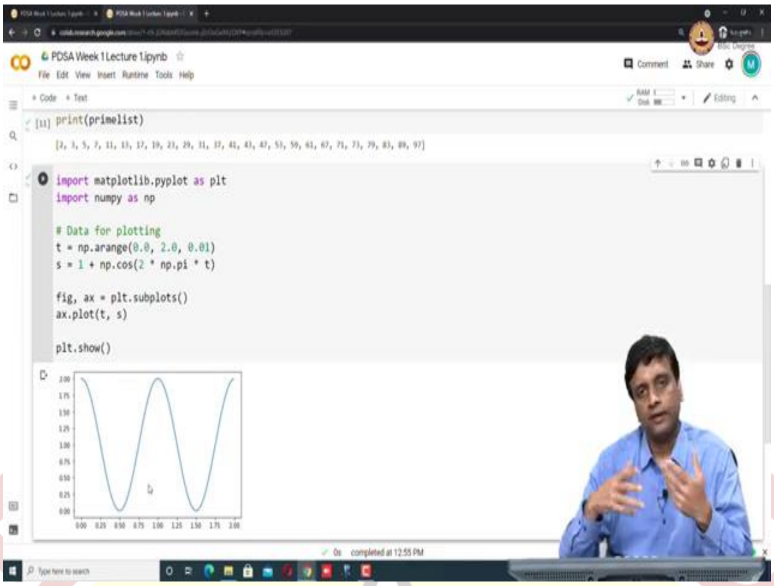
But supposing I make a mistake, so supposing I, instead of equal to equal to I say not equal to, so supposing I say a number is prime if the list of factors is not one comma n, in other words I have exactly negated the thing, so I have got factors other than one comma n. Now, supposing I call this my definition of primes and now I run this then what I will get is?

I will get a list of non-primes, I get 2 and 3 are skipped, I get 1, which is not a prime, 2 and 3 are skipped, which are primes, I get 4, I skip 5, I get 6 and so on. Now, if I go back and I take the correct definition of prime which I have not changed and I run this again, then I get the primes again. So, this is the nice thing about this notebook that you can dynamically…

So, in this case I made a mistake, but you can have two different definitions of a function, you can kind of interplay between them and you can come back and run your code, without having to go back and do a lot of copying and pasting and editing and all that, so some subset of these cells are available and the current state of the cells is what is basically according to the dynamics of how you ran them.

(Refer Slide Time: 7:45)





So, let us look at this other example. So, here is that sine curve which I had shown in the slide. So, the sine curve, again the previous plot is available, it is not that it is run, so now if change, for example, this function sin to cosine, so if you know your trigonometry, you will know that cosine is basically going to be the mirror image of sin.

So, this particular thing goes up and then down, so cosine will go down and then up. So, if I run this code now, it is going to recompute that plot and now it is going to start from above and go down, so I get a kind of flipped version of the sin thing because sin and cosine are flipped. So, this is a kind of dynamic environment in which I can, now if I save, so this file now will have, at this juncture will have whatever changes I made.

So, if I share this with you now, you will see whatever updates I made now and in general I can change it yet again and present you a version which is instructive in terms of text and in terms of outputs and then you can take it and further run it, so that is the advantage of using this Jupyter notebook.

सिद्धिर्भवति कर्मजा