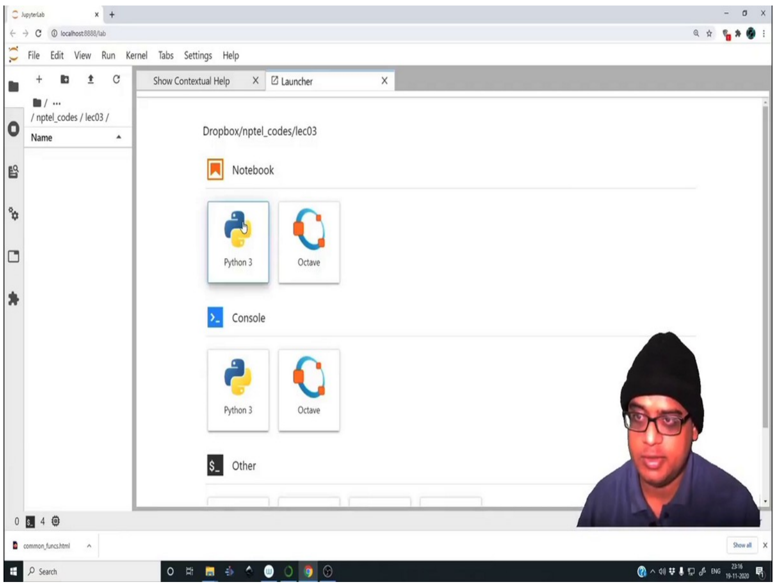
Tools in Scientific Computing

Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 03
Commonly used functions

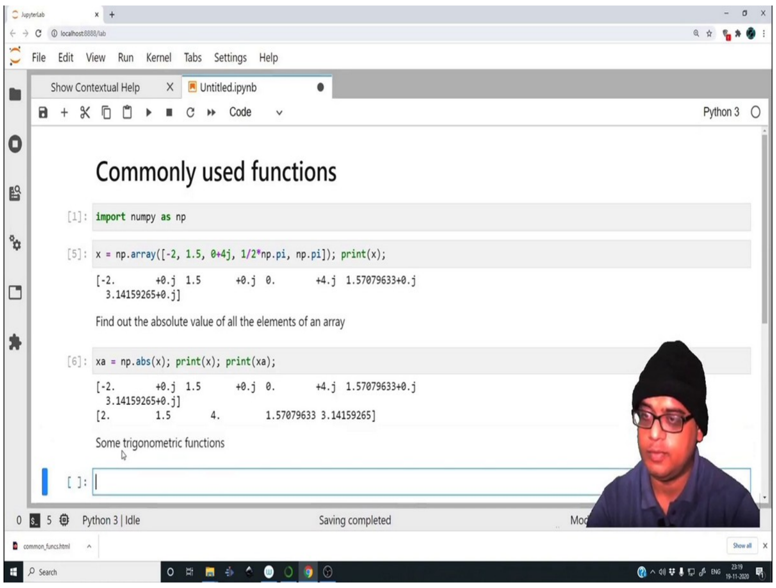
Hello and welcome to this 3rd lecture in which we will be covering some common functions. So, let us begin I have opened up JupyterLab over here. So, let me create a new Notebook.

(Refer Slide Time: 00:38)



So, I will select the kernel. You can select an Octave kernel if you want to program an Octave. You can select Python kernel if you want to program in Python. JupyterLabs provide provides you a good environment to write scripts in both. I will be focusing more on Python 3 in this particular session, but you will find the corresponding Octave file also in the website.

(Refer Slide Time: 01:15)



Commonly used functions

```
[1]: import numpy as np

[5]: x = np.array([-2, 1.5, 0+4j, 1/2*np.pi, np.pi]); print(x);

[-2.        +0.j  1.5       +0.j  0.        +4.j  1.57079633+0.j
  3.14159265+0.j]
```

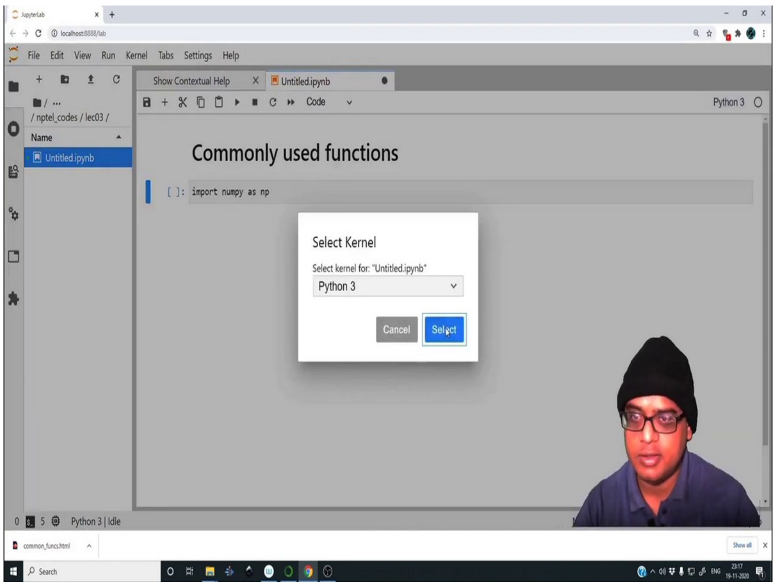
Find out the absolute value of all the elements of an array

```
[6]: xa = np.abs(x); print(x); print(xa);

[-2.        +0.j  1.5       +0.j  0.        +4.j  1.57079633+0.j
  3.14159265+0.j]
[2.         1.5        4.         1.57079633 3.14159265]
```

Some trigonometric functions

So, before starting let us quickly define what this file is going to be. So, it is a it says it is a notebook on commonly used functions. So, let us import numpy as np.

(Refer Slide Time: 01:34)



Commonly used functions

```
[ ]: import numpy as np
```

Select Kernel

Select kernel for: "Untitled.ipynb"

Python 3

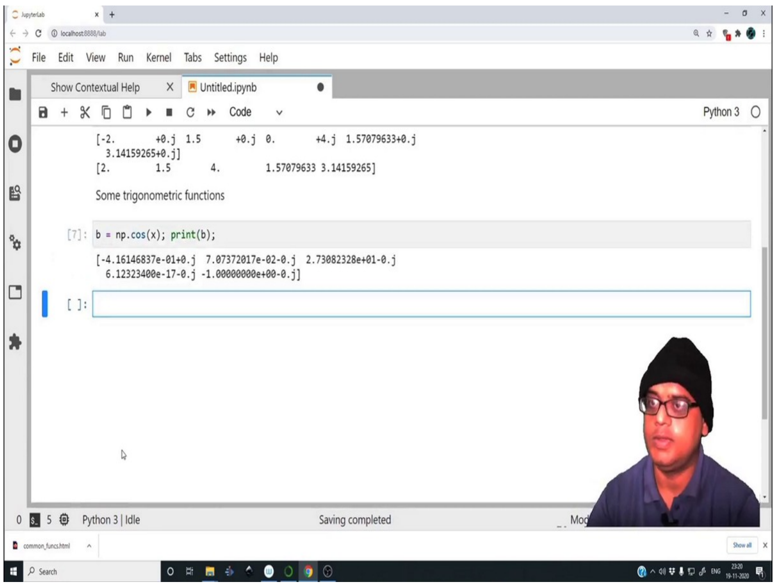
So, this is small bug in which I have to select the kernel again. So, if I press shift enter it evaluates the cell and it is done. So, let me create an array. So, x = np.array([-2, 1.5, 0, 1/2*np.pi, np.pi]) So, let me print out the elements. So, let me print x. So, these are the elements of the array we have just defined, its a user defined array.

So, now suppose I want to find out the absolute value. So, let me just go ahead and write xa. So, xa = np.abs(x). So, let me then print what x is and then correspondingly let me print what xa is. So, here the - 2 has been converted to 2. In fact, we can take complex number as well. So, 0+4j. So, in numpy complex numbers are usually represented by j rather than i and in electrical engineering as well this convention is followed.

So, let me evaluate this cell. So, now, we have -2 + 0 j, 1.5 + 0 j, 0 + 4 j and this. So, it just append 0 j because we have declared one of the variables as complex. So, it starts to treat all the variables inside x as complex with the 0 imaginary part of course. Let me find out the absolute. So, the absolute value of 0 + 4 j is 4 and it checks out ok. So, this is how we can find the absolute values.

And, this is quite an important function we will be making use of this function now and then. The other important function that we will be using is trigonometric functions like cosine, sine and all these things. So, let me declare this as markdown let me write over here some trigonometric functions ok.

(Refer Slide Time: 04:31)



```
[-2.        +0.j  1.5       +0.j  0.        +4.j  1.57079633+0.j
  3.14159265+0.j]
[2.         1.5        4.         1.57079633 3.14159265]
```

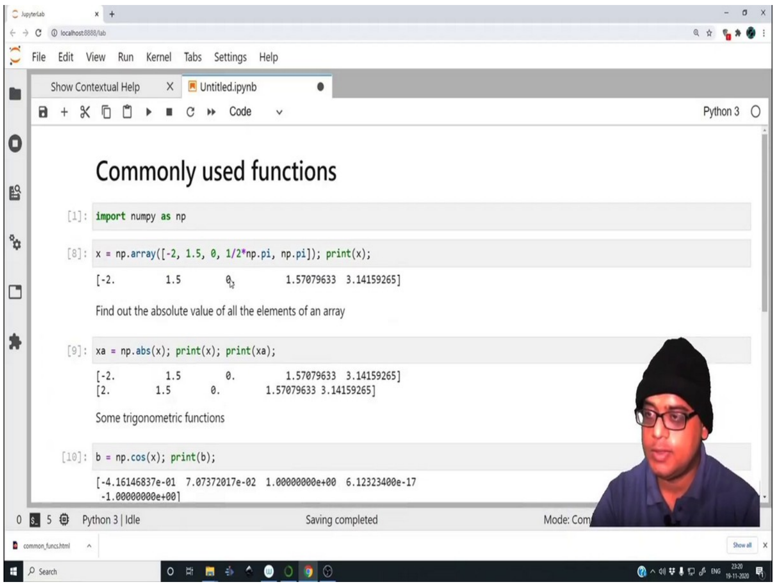
Some trigonometric functions

```
[7]: b = np.cos(x); print(b);
```

```
[-4.16146837e-01+0.j  7.07372017e-02-0.j  2.73082328e+01-0.j
  6.12323400e-17-0.j -1.00000000e+00-0.j]
```

So, let me define x as np.cos(x) let me define this as b let me print b. In fact, let me remove the complex part for simplicity ok.

(Refer Slide Time: 04:48)



Commonly used functions

```
[1]: import numpy as np

[8]: x = np.array([-2, 1.5, 0, 1/2*np.pi, np.pi]); print(x);

[-2.         1.5        0.         1.57079633  3.14159265]
```

Find out the absolute value of all the elements of an array

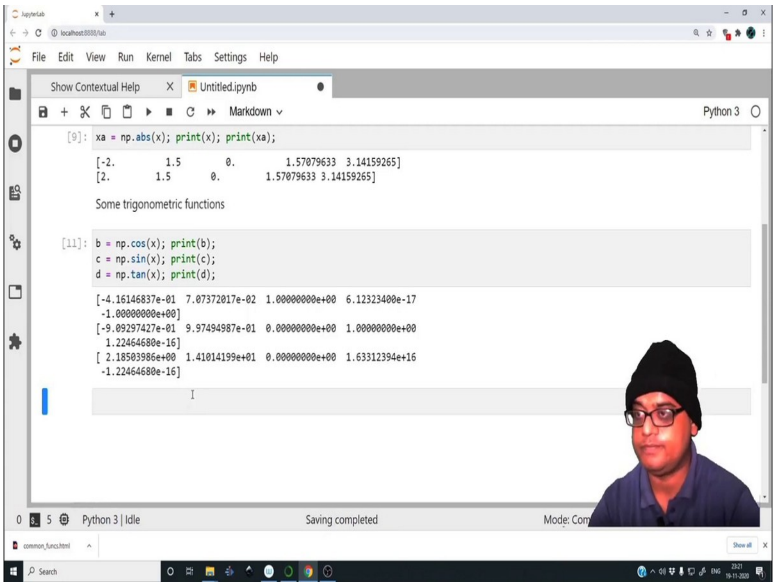
```
[9]: xa = np.abs(x); print(x); print(xa);

[-2.         1.5        0.         1.57079633  3.14159265]
[2.        1.5       0.        1.57079633 3.14159265]
```

Some trigonometric functions

```
[10]: b = np.cos(x); print(b);

[-4.16146837e-01  7.07372017e-02  1.00000000e+00  6.12323400e-17
 -1.00000000e+00]
```

So, x is this, absolute is this, b is equal to this ok. So, cos of so, whatever the input is it is in radians. So, cos (π) so, np.pi is equal to π is - 1; cos(π/2) is something which is very small it is close to 0; cos 0 is 1, then cos 1.5 radian is this and cos - 2 radian is this.

(Refer Slide Time: 05:24)



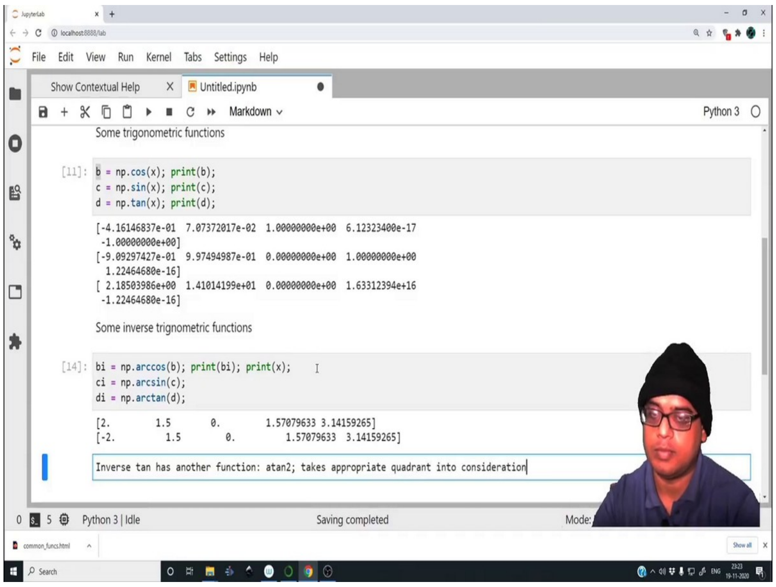
```
[9]: xa = np.abs(x); print(x); print(xa);

[-2.         1.5        0.         1.57079633  3.14159265]
[2.        1.5       0.        1.57079633 3.14159265]
```

Some trigonometric functions

```
[11]: b = np.cos(x); print(b);
      c = np.sin(x); print(c);
      d = np.tan(x); print(d);

[-4.16146837e-01  7.07372017e-02  1.00000000e+00  6.12323400e-17
 -1.00000000e+00]
[-9.09297427e-01  9.97494987e-01  0.00000000e+00  1.00000000e+00
  1.22464680e-16]
[ 2.18503986e+00  1.41014199e+01  0.00000000e+00  1.63312394e+16
 -1.22464680e-16]
```

We can similarly find sin(x) and we can find the tangent the tan ok. So, its a very straightforward exercise to do. So, tan(π/2) it is a very large number ok. So, for all practical purposes this is infinity. You do not want it to go to nan you do not want to overflow to nan otherwise all your computations will be ruined. Do need to take care of such kinds of large numbers appearing in your computation and that boils down to having experience on expecting small or large numbers.

(Refer Slide Time: 06:15)



Some trigonometric functions

```
[11]: b = np.cos(x); print(b);
      c = np.sin(x); print(c);
      d = np.tan(x); print(d);

[-4.16146837e-01  7.07372017e-02  1.00000000e+00  6.12323400e-17
 -1.00000000e+00]
[-9.09297427e-01  9.97494987e-01  0.00000000e+00  1.00000000e+00
  1.22464680e-16]
[ 2.18503986e+00  1.41014199e+01  0.00000000e+00  1.63312394e+16
 -1.22464680e-16]
```

Some inverse trignometric functions

```
[14]: bi = np.arccos(b); print(bi); print(x);
      ci = np.arcsin(c);
      di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

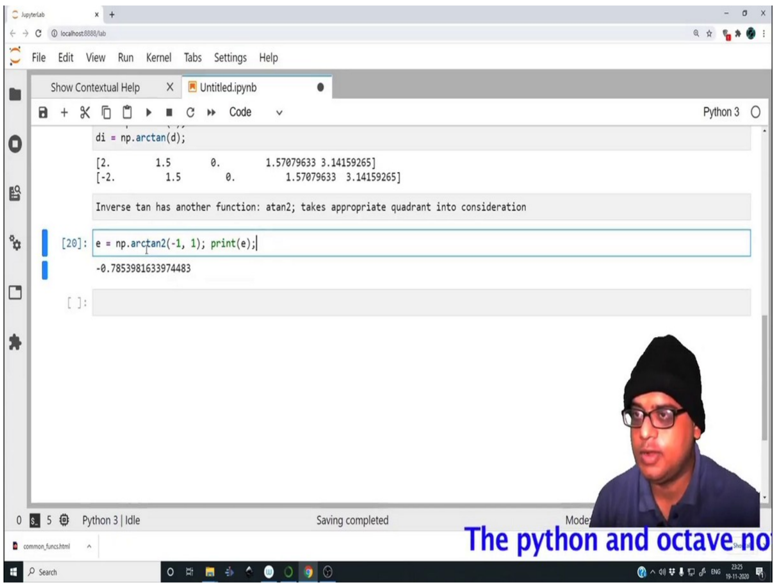
```
Inverse tan has another function: atan2; takes appropriate quadrant into consideration
```

Let us find out some inverse trigonometric functions. So, let me say bi = np.arccos(b) print bi. So, we have just taken whatever we have cosined over here, we have stored it in b and we are then taking a cos inverse of it. So, we expect to get back the same x array.

So, in fact, let me print out x as well to just make sure it is same and it is in fact, same ok. So, that takes care of the inverse. Similarly, ci = np.arcsin(c), di = np.arctan(d). So, this is how you evaluate inverses.

So, far what we have studied is quite easy and there is another tan inverse function ok. So, tan Inverse. So, it is atan2 in octave and all; in MATLAB also you will find this function atan2 and basically it takes the appropriate quadrant into consideration.

(Refer Slide Time: 07:53)



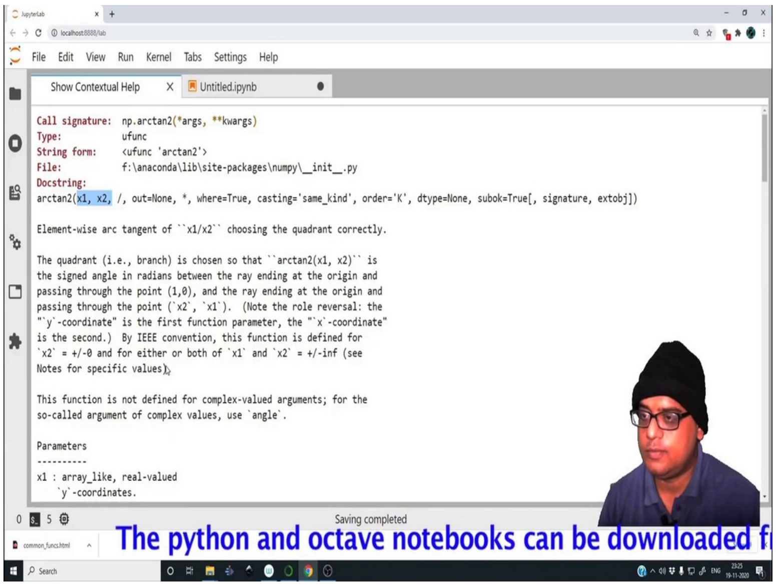
```
di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

Inverse tan has another function: atan2; takes appropriate quadrant into consideration

```
[20]: e = np.arctan2(-1, 1); print(e);

-0.7853981633974483
```

So, let us find out the Inverse tangent. So, np.arctan2(-1, 1).

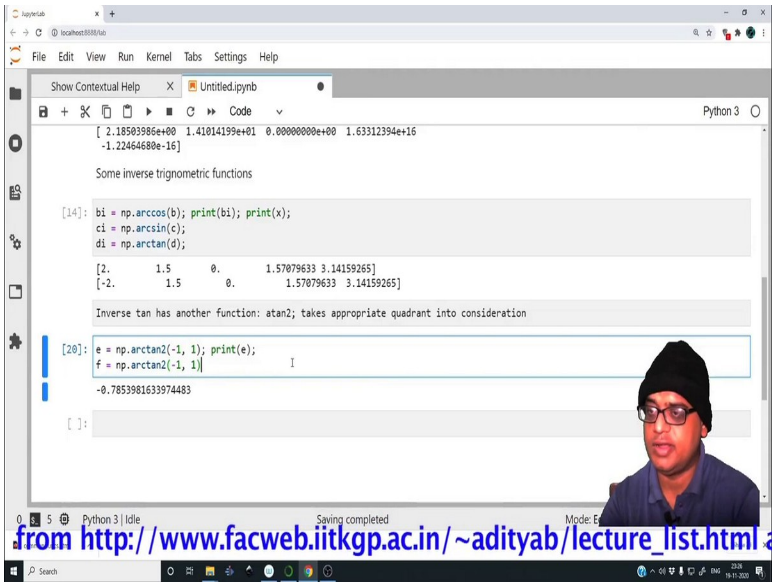
(Refer Slide Time: 08:02)



```
Call signature:  np.arctan2(*args, **kwargs)
Type:            ufunc
String form:     <ufunc 'arctan2'>
File:            f:\anaconda\lib\site-packages\numpy\__init__.py
Docstring:
arctan2(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])

Element-wise arc tangent of ``x1/x2`` choosing the quadrant correctly.

The quadrant (i.e., branch) is chosen so that ``arctan2(x1, x2)`` is
the signed angle in radians between the ray ending at the origin and
passing through the point (1,0), and the ray ending at the origin and
passing through the point (`x2`, `x1`).  (Note the role reversal: the
"`y`-coordinate" is the first function parameter, the "`x`-coordinate"
is the second.)  By IEEE convention, this function is defined for
`x2` = +/-0 and for either or both of `x1` and `x2` = +/-inf (see
Notes for specific values).

This function is not defined for complex-valued arguments; for the
so-called argument of complex values, use `angle`.

Parameters
----------
x1 : array_like, real-valued
    `y`-coordinates.
```

Let me define it to e let me print the value of e ok. So, the meaning of this function you can double click on this and go to the Contextual Help. So, it is x1, comma x2 arc tangent of x1 divided by x2 ok. So, just have a go through this.

(Refer Slide Time: 08:29)



```
[ 2.18503986e+00  1.41014199e+01  0.00000000e+00  1.63312394e+16
 -1.22464680e-16]
```

Some inverse trignometric functions

```
[14]: bi = np.arccos(b); print(bi); print(x);
      ci = np.arcsin(c);
      di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

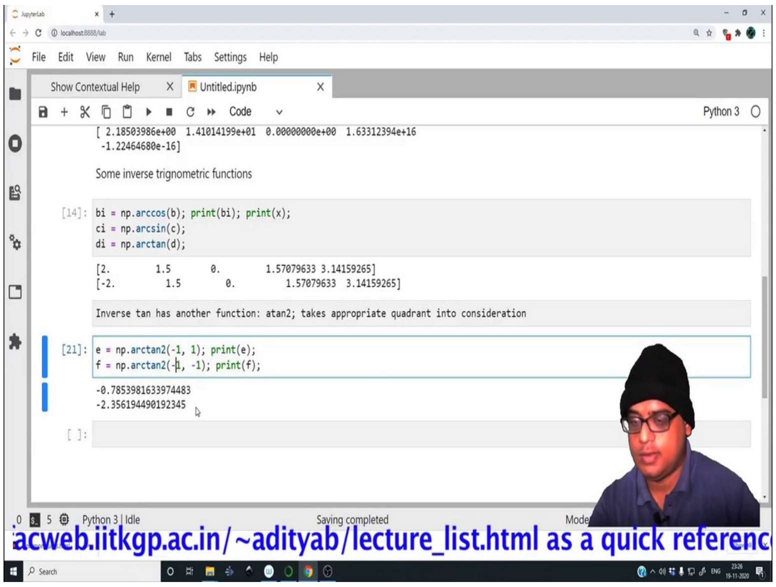
Inverse tan has another function: atan2; takes appropriate quadrant into consideration

```
[20]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(-1, 1)

-0.7853981633974483
```

from http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html

So, y-coordinate is the first function and the x-coordinate is the second function. So, why is this. So, -1 it belongs to the negative y-axis and x is the positive. So, it belongs in the fourth quadrant let me define f = np.arctan2(1, 1).

(Refer Slide Time: 08:59)



```
[ 2.18503986e+00  1.41014199e+01  0.00000000e+00  1.63312394e+16
 -1.22464680e-16]
```

Some inverse trignometric functions

```
[14]: bi = np.arccos(b); print(bi); print(x);
      ci = np.arcsin(c);
      di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

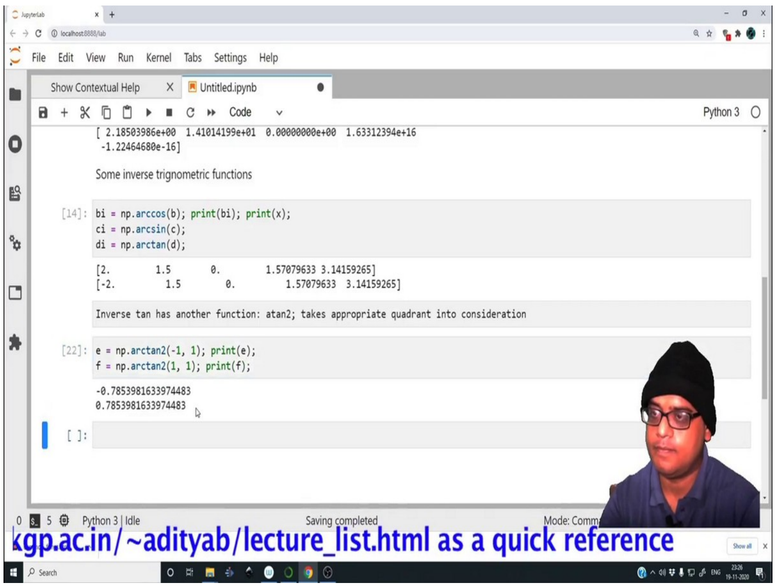
Inverse tan has another function: atan2; takes appropriate quadrant into consideration

```
[21]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(-1, -1); print(f);

-0.7853981633974483
-2.356194490192345
```

acweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick referenc

Let me take both as negative ok. Similarly, we can take both positive.

(Refer Slide Time: 09:09)



```
[ 2.18503986e+00  1.41014199e+01  0.00000000e+00  1.63312394e+16
 -1.22464680e-16]
```

Some inverse trignometric functions

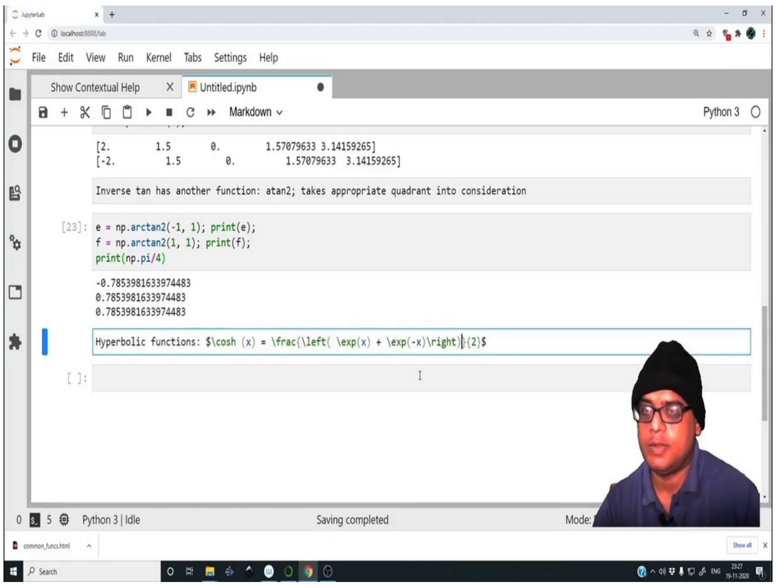
```
[14]: bi = np.arccos(b); print(bi); print(x);
      ci = np.arcsin(c);
      di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5         0.          1.57079633  3.14159265]
```

Inverse tan has another function: atan2; takes appropriate quadrant into consideration

```
[22]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(1, 1); print(f);

-0.7853981633974483
0.7853981633974483

[ ]:
```

So, this is how we. So, basically this is $\pi/4$ in case you wondering this is $\pi/4$ ok.

(Refer Slide Time: 09:18)



```
[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5         0.          1.57079633  3.14159265]
```

Inverse tan has another function: atan2; takes appropriate quadrant into consideration

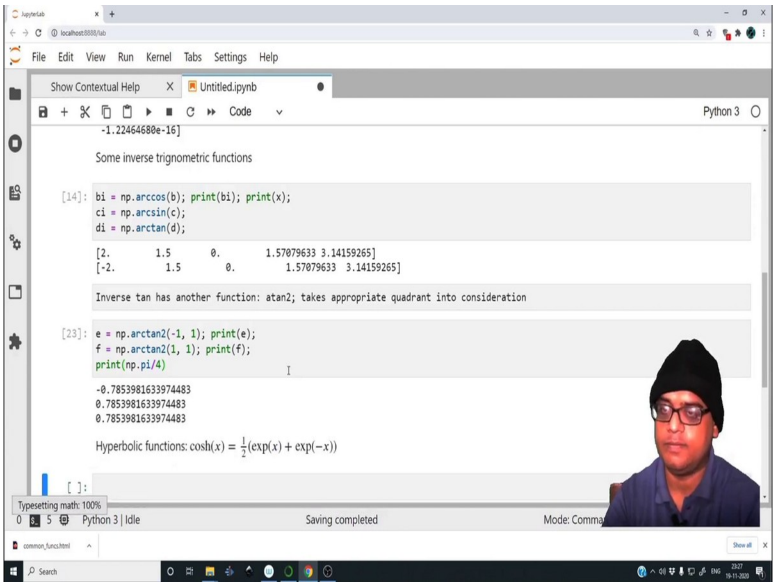
```
[23]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(1, 1); print(f);
      print(np.pi/4)

-0.7853981633974483
0.7853981633974483
0.7853981633974483
```

```
Hyperbolic functions: $\cosh (x) = \frac{\left( \exp(x) + \exp(-x)\right)}{2}$
```

```
[ ]:
```

So, this is how we find out the appropriate arc tangent. Similarly, there is a bunch of functions to find out the Hyperbolic sines and cosines. So, basically a hyperbolic function so, a function like cosh(x) defined as something like this.

(Refer Slide Time: 10:07)



```
-1.22464680e-16]
```

Some inverse trignometric functions

```
[14]: bi = np.arccos(b); print(bi); print(x);
      ci = np.arcsin(c);
      di = np.arctan(d);

[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

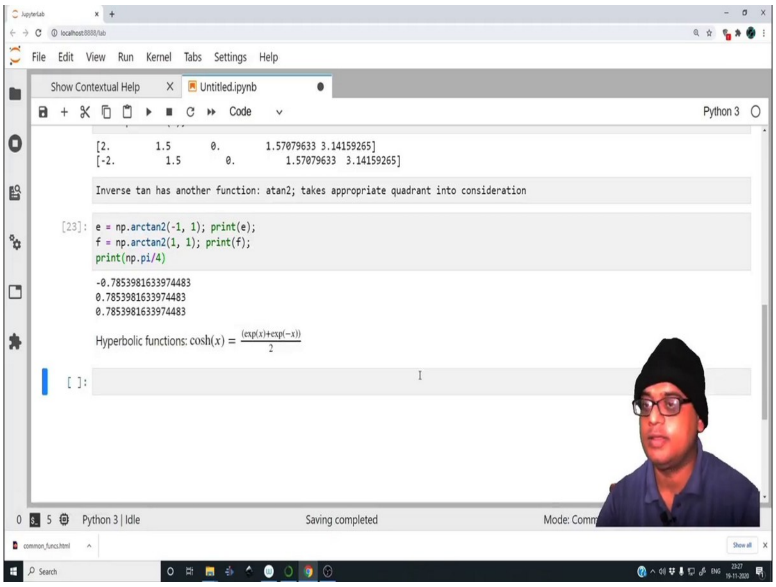
```
Inverse tan has another function: atan2; takes appropriate quadrant into consideration
```

```
[23]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(1, 1); print(f);
      print(np.pi/4)

-0.7853981633974483
0.7853981633974483
0.7853981633974483
```

Hyperbolic functions: $\cosh(x) = \frac{1}{2}(\exp(x) + \exp(-x))$

```
[ ]:
```

(Refer Slide Time: 10:16)



```
[2.         1.5        0.         1.57079633 3.14159265]
[-2.         1.5        0.         1.57079633  3.14159265]
```

```
Inverse tan has another function: atan2; takes appropriate quadrant into consideration
```

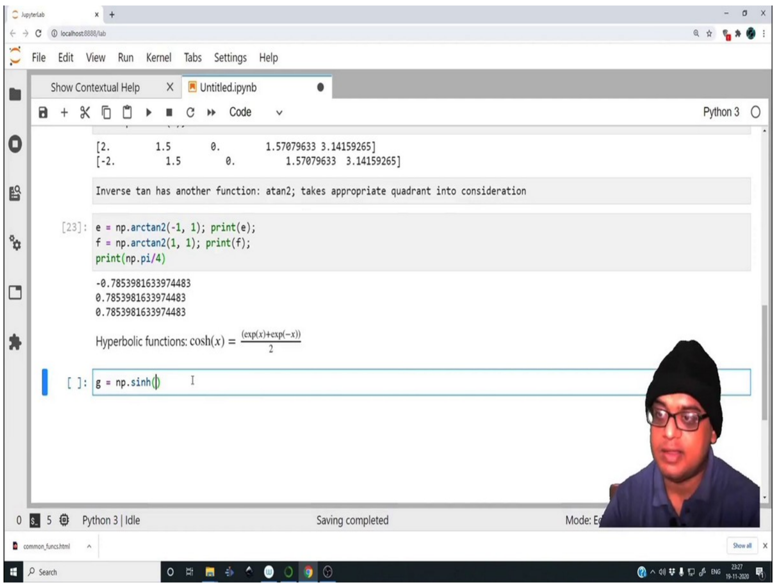
```
[23]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(1, 1); print(f);
      print(np.pi/4)

-0.7853981633974483
0.7853981633974483
0.7853981633974483
```

Hyperbolic functions: $\cosh(x) = \frac{(\exp(x)+\exp(-x))}{2}$

```
[ ]:
```

So, you can input latex commands inside the Markdown and so, you can render nice equations. So, this is a small tip if you make.

(Refer Slide Time: 10:29)



```
[2.        1.5       0.        1.57079633 3.14159265]
[-2.        1.5       0.        1.57079633  3.14159265]
```

Inverse tan has another function: atan2; takes appropriate quadrant into consideration

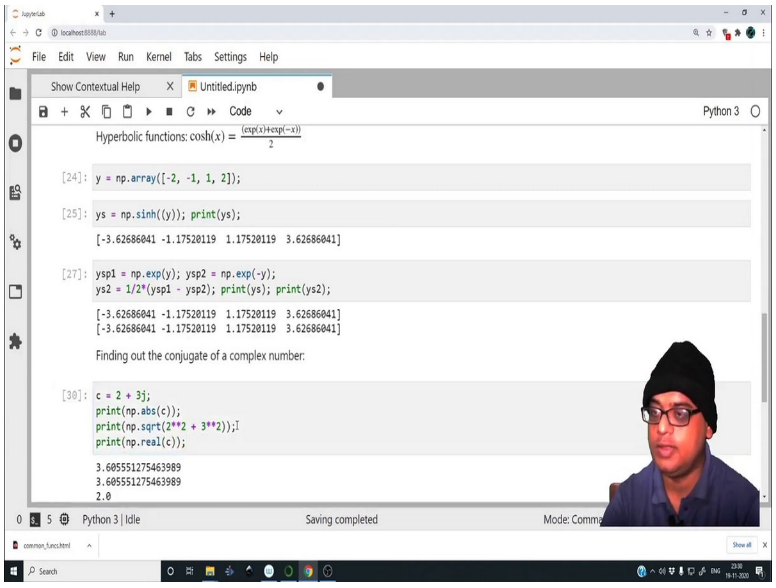
```
[23]: e = np.arctan2(-1, 1); print(e);
      f = np.arctan2(1, 1); print(f);
      print(np.pi/4)

-0.7853981633974483
0.7853981633974483
0.7853981633974483
```

Hyperbolic functions: $\cosh(x) = \frac{(\exp(x)+\exp(-x))}{2}$

```
[ ]: g = np.sinh()
```

So, its simple we just define g as np.sinh of whatever you want and we can pass arrays to this. So, like we have done for these functions we have passed down arrays, we have passed down x. So, the all these values of x were fed into the function cos and we obtained the corresponding array which corresponds to the cos of each element.

(Refer Slide Time: 10:58)



Hyperbolic functions: $\cosh(x) = \frac{(\exp(x)+\exp(-x))}{2}$

```
[24]: y = np.array([-2, -1, 1, 2]);

[25]: ys = np.sinh((y)); print(ys);

[-3.62686041 -1.17520119  1.17520119  3.62686041]

[27]: ysp1 = np.exp(y); ysp2 = np.exp(-y);
      ys2 = 1/2*(ysp1 - ysp2); print(ys); print(ys2);

[-3.62686041 -1.17520119  1.17520119  3.62686041]
[-3.62686041 -1.17520119  1.17520119  3.62686041]
```

Finding out the conjugate of a complex number:

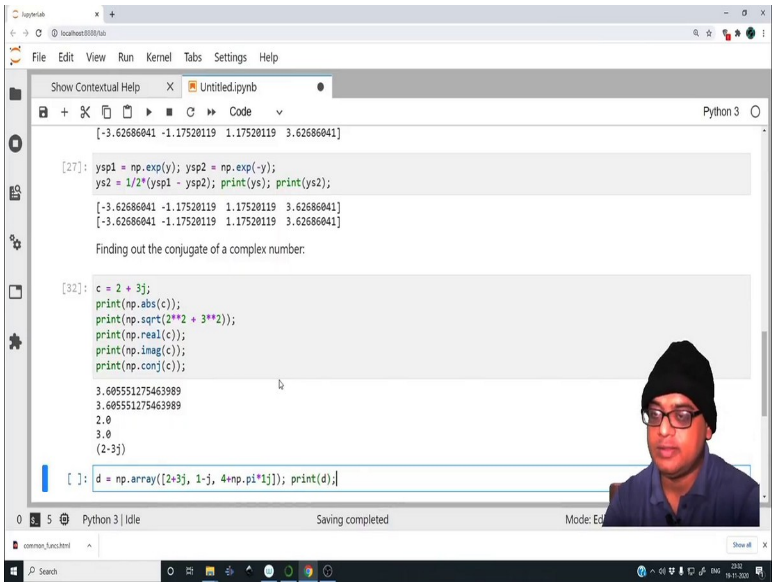
```
[30]: c = 2 + 3j;
      print(np.abs(c));
      print(np.sqrt(2**2 + 3**2));
      print(np.real(c));

3.605551275463989
3.605551275463989
2.0
```

Similarly, we can do this over here. So, let me define y = np.array([-2, -1, 1, 2]) something like this suppose. So, let me do ys = np.sinh((y)) print ys. In fact, let us confirm whether sinh is (exp(x)-exp(−x))/2. So, let me say ysp1 ys part 1 is np.exp(y), ysp2 is np.exp(-y) and ys2=1/2*(ysp1 - ysp2). So, let me print ys and let me print ys 2. So, this has to be a negative sign over here. So, they are indeed the same.

So, this is how you can find out sinh. So, there is a bunch of functions hyperbolic sin cos tan. There is also inverse hyperbolic functions arc cos h, arc sin h, arc tan h ok. So, this is how you do it. Let us look at finding out the conjugate. So, let us define c=2+3j. So, let us then define rather let us print |c| let us see what it is 3.6055. In fact, let us confirm whether it is true.

So, $\sqrt{(2^2+3^2)}$ ok. So, it does match. And it is not surprising that it matches I mean I just wanted to show you that it does match. So, what about the real part? So, let us print np.real(c).

(Refer Slide Time: 13:31)



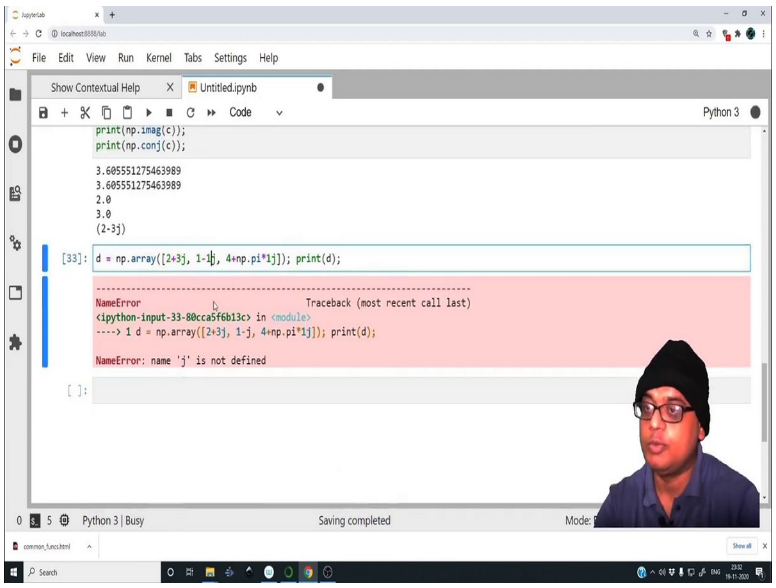
```
[-3.62686041 -1.17520119  1.17520119  3.62686041]

[27]: ysp1 = np.exp(y); ysp2 = np.exp(-y);
      ys2 = 1/2*(ysp1 - ysp2); print(ys); print(ys2);

[-3.62686041 -1.17520119  1.17520119  3.62686041]
[-3.62686041 -1.17520119  1.17520119  3.62686041]
```

Finding out the conjugate of a complex number:

```
[32]: c = 2 + 3j;
      print(np.abs(c));
      print(np.sqrt(2**2 + 3**2));
      print(np.real(c));
      print(np.imag(c));
      print(np.conj(c));

3.605551275463989
3.605551275463989
2.0
3.0
(2-3j)

[ ]: d = np.array([2+3j, 1-j, 4+np.pi*1j]); print(d);
```

So, it is np.imag(c) there you go similarly we can find out the conjugate of c. So, print np.conj(c). So, it is 2-3j. In fact, whatever functions we are writing for the complex numbers they are equally valid if you apply it for an array. So, let us define d = np.array([2+3j, 1-1j, 4+np.pi*1j]). So, you cannot just write j you have to always write 1j because j it can be a different variable, it can be a loop counter for example. So, you do not want to do that. You want to avoid writing j as a naked term you want to always write it 1j to signify it is a complex number.

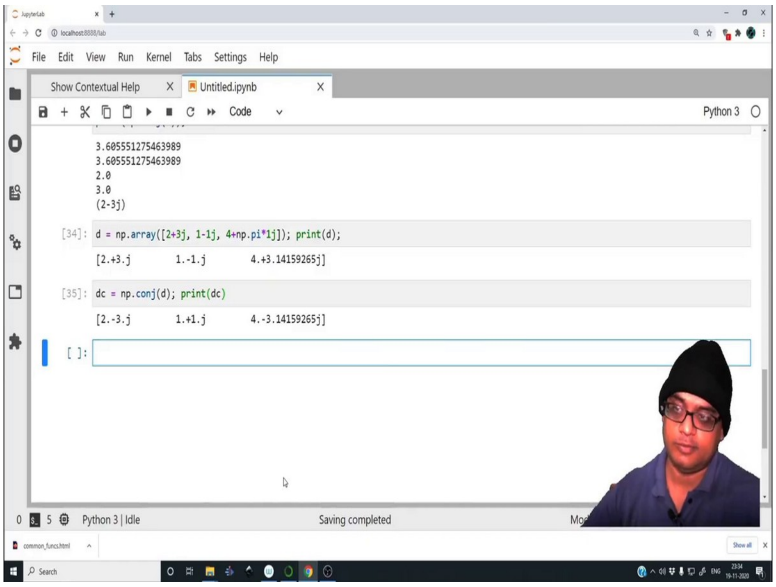
(Refer Slide Time: 14:40)



```
print(np.imag(c));
print(np.conj(c));

3.605551275463989
3.605551275463989
2.0
3.0
(2-3j)

[33]: d = np.array([2+3j, 1-1j, 4+np.pi*1j]); print(d);

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-80cca5f6b13c> in <module>
----> 1 d = np.array([2+3j, 1-j, 4+np.pi*1j]); print(d);

NameError: name 'j' is not defined

[ ]:
```

So, let us print d in this case. So, j is not defined great. So, this has to be 1j.

(Refer Slide Time: 14:45)

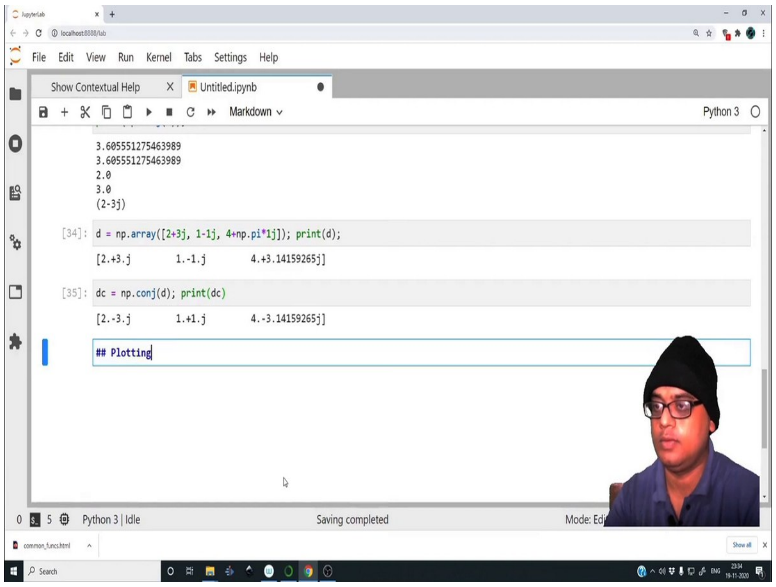


```
3.605551275463989
3.605551275463989
2.0
3.0
(2-3j)

[34]: d = np.array([2+3j, 1-1j, 4+np.pi*1j]); print(d);

[2.+3.j        1.-1.j        4.+3.14159265j]

[35]: dc = np.conj(d); print(dc)

[2.-3.j        1.+1.j        4.-3.14159265j]

[ ]:
```

So, now, we have a complex number. So, let us say d conjugate is equal to np.conj(d). So, let me print what dc is. So, it is the conjugate of 2 + 3j which is 2 - 3j conjugate of 1 - j, 1 + j and so on. So, this is how you can broadcast the functions to the entire array. In fact, it can be broadcast to an entire matrix as well.

It is an element wise operation, it is broadcasted, it is set to broadcast all over the elements of the array or the matrix. So, now, let us move on to something which you will use heavily in the course of this entire course for that matter your research work and that is plotting of functions ok.

(Refer Slide Time: 15:40)



```
3.605551275463989
3.605551275463989
2.0
3.0
(2-3j)
```

```
[34]: d = np.array([2+3j, 1-1j, 4+np.pi*1j]); print(d);
```

```
[2.+3.j        1.-1.j        4.+3.14159265j]
```

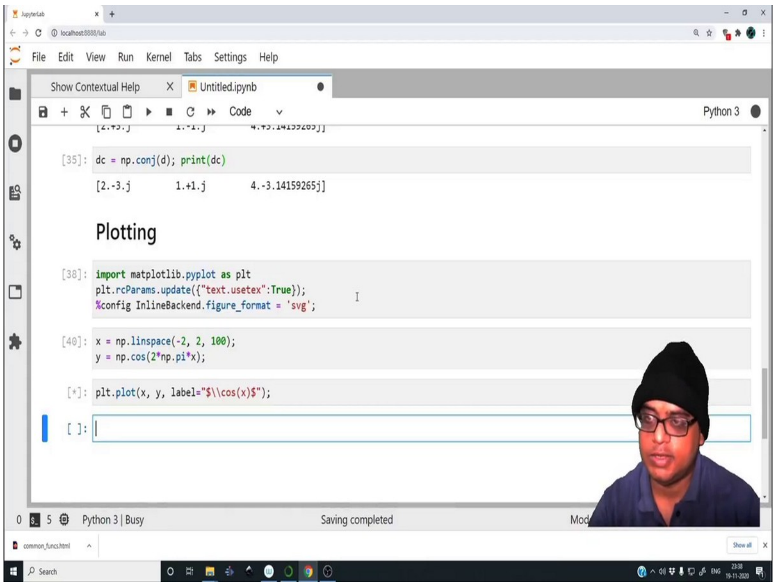
```
[35]: dc = np.conj(d); print(dc)
```

```
[2.-3.j        1.+1.j        4.-3.14159265j]
```

```
## Plotting
```

So, let us import the appropriate library for it or the module for it. So, Plotting.

(Refer Slide Time: 15:45)



```
[35]: dc = np.conj(d); print(dc)
```

```
[2.-3.j        1.+1.j        4.-3.14159265j]
```

Plotting

```
[38]: import matplotlib.pyplot as plt
      plt.rcParams.update({"text.usetex":True});
      %config InlineBackend.figure_format = 'svg';
```

```
[40]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);
```

```
[*]: plt.plot(x, y, label="$\\cos(x)$");
```

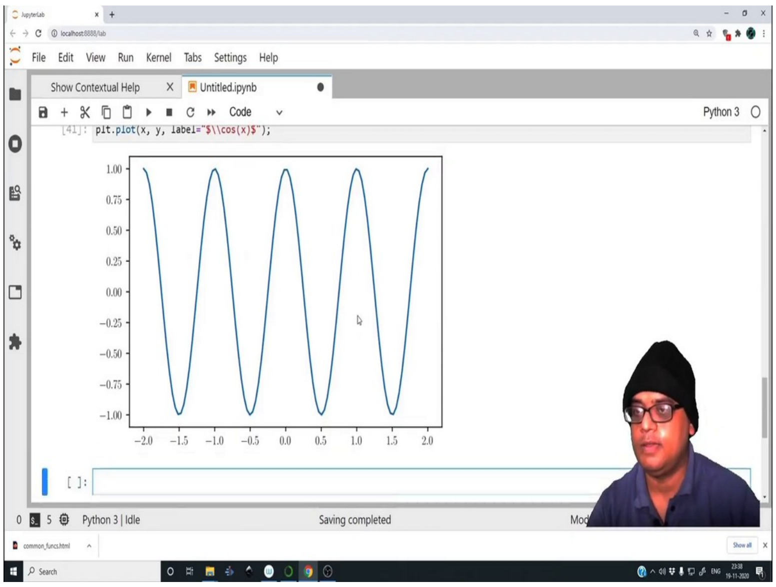
Let us import matplotlib. So, in fact, if you want to make line plots or xy plots or in fact, 2-dimensional plots the sub module pyplot is sufficient for your needs. So, import matplotlib dot pyplot and we want to import it as an alias plt ok. So, this is how you import matplotlib, but once you have imported it typically what I do is something like this.

I set the rc params. So, there is something which there is a parameter file inside matplotlib which sort of dictates how the feel of the plot is you do not need to do anything about it usually, but it is just to make plots appear slightly pretty. So, we have plt dot rcParams dot update and inside this we have a bunch of parameters. So, the first parameter is text dot usetex and we set it to be true alright then that is it I mean this is the important parameter, you do not need to really tweak with anything else.

Apart from this for showing the plots inline we will be using inline plotting in this particular notebook. So, in order to perform inline plots we will use this config InlineBackend dot figure underscore format and we will set it to svg. So, its scalable vector graphics. So, we evaluate the cell and now we are ready to plot. Now, suppose we want to plot sin x or cos x or whatever we have.

So, let us first define x. So, let us define x = np.linspace(-2, 2, 100). So, this is the linspace command that we have seen earlier then we will define we will do it in the same cell we will define y = np.cos(2*np.pi*x). So, now we evaluate the cell let us go over here. Let us do plt.plot(x,y) and in fact, let me label this function as we will set a label as cos(x). So, let us see what it returns. It takes a while initially.

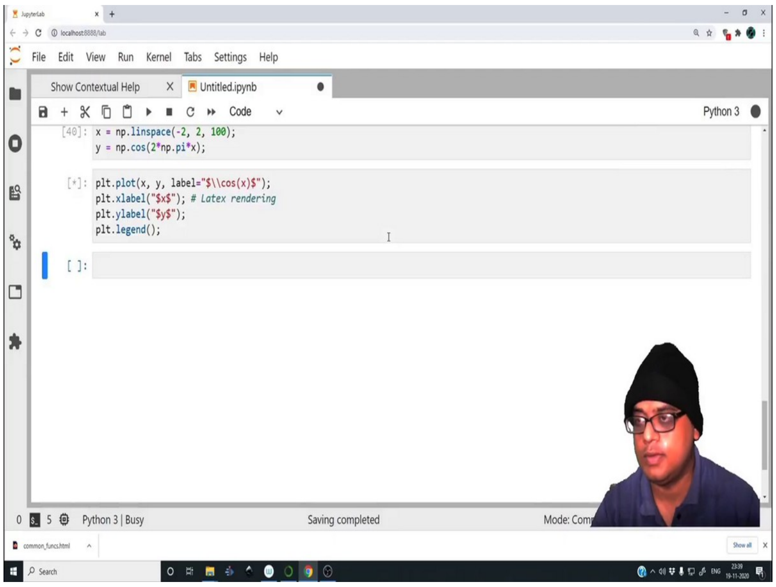
(Refer Slide Time: 19:13)



```
[41]: plt.plot(x, y, label="$\\cos(x)$");
```

There you go this is the plot ok. So, plot of cos(x). Now, I want to label the x and y axis.

(Refer Slide Time: 19:23)

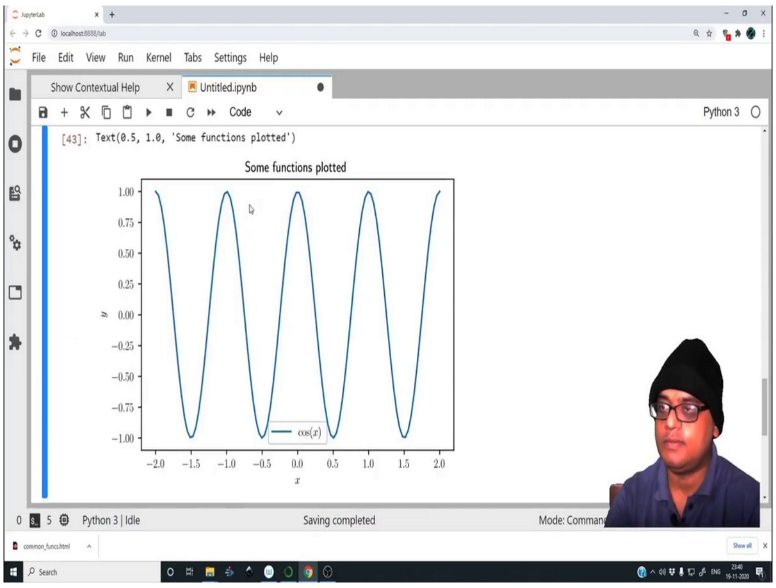


```
[40]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);

[*]:  plt.plot(x, y, label="$\\cos(x)$");
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();

[ ]:
```

So, I will say plt.xlabel and I will label it as x and the reason why I am putting things under double $s is that I am rendering the text in Latex. So, Latex if you like is a rendering or it is a typesetting software or a typesetting program where you can render mathematics much more easily than anything else.

So, there is a various there is a very specific syntax to render Latex and you have to put mathematics within $s. So, the reason why I am using two $s over here is the escape character backslash it does not get accepted in Python unless you put two backslashes. So, let me write plt.ylabel and do not worry too much about this you will get the hang of it once you start using it and say plt dot legend.

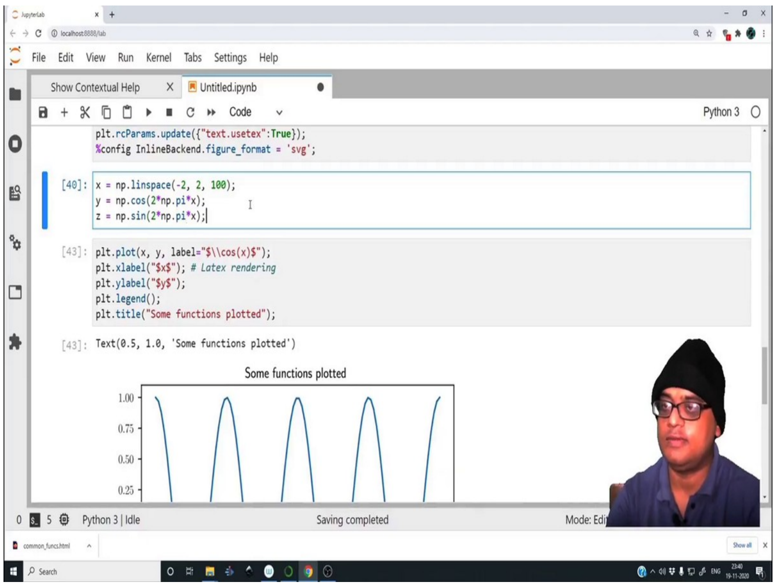
(Refer Slide Time: 20:28)



```
[43]: Text(0.5, 1.0, 'Some functions plotted')
```

So, the legend is placed over here. It is the function cos x that is being plotted and this is the function and on the y axis we have y as the label.

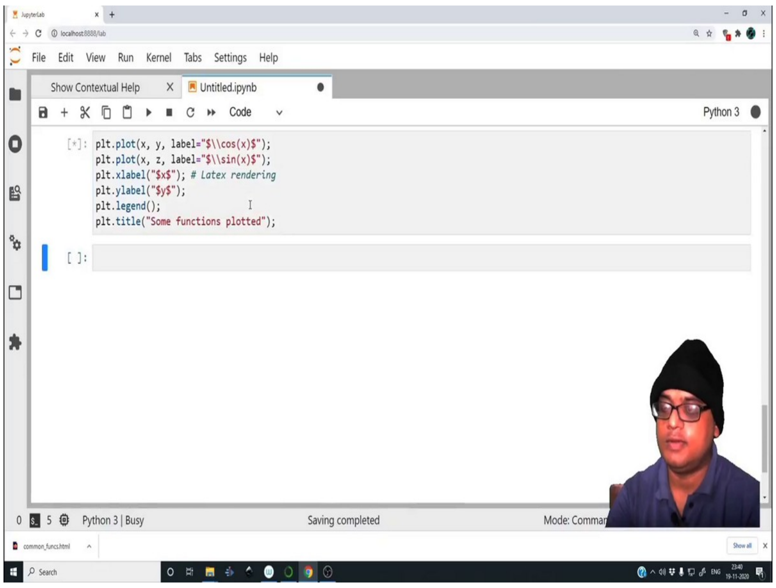
(Refer Slide Time: 20:40)



```
plt.rcParams.update({"text.usetex":True});
%config InlineBackend.figure_format = 'svg';

[40]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);
      z = np.sin(2*np.pi*x);

[43]: plt.plot(x, y, label="$\\cos(x)$");
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");

[43]: Text(0.5, 1.0, 'Some functions plotted')
```

We can in fact, put a title to this one this plot as well plt.title some functions plotted it shows a title over here as well. Now, we can add more plots to this we can saz = np.sin(2*np.pi*x).

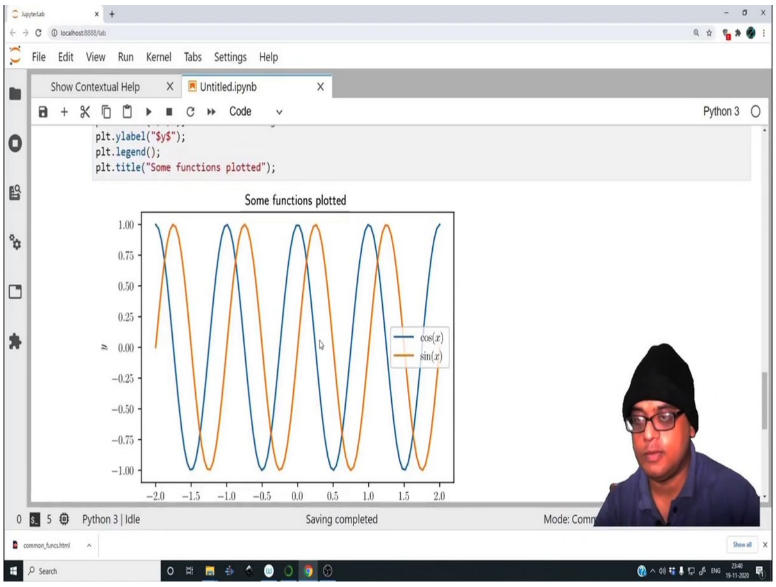
(Refer Slide Time: 21:05)



```
[*]: plt.plot(x, y, label="$\\cos(x)$");
     plt.plot(x, z, label="$\\sin(x)$");
     plt.xlabel("$x$"); # Latex rendering
     plt.ylabel("$y$");
     plt.legend();
     plt.title("Some functions plotted");

[ ]:
```

Then inside this we can add another plot. So, plt.plot (x,z) label is sin(x).

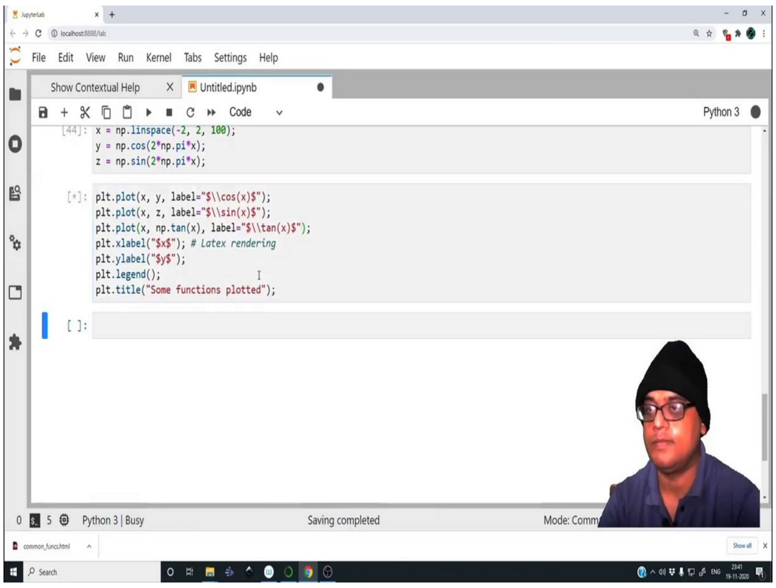
(Refer Slide Time: 21:18)



```
plt.ylabel("$y$");
plt.legend();
plt.title("Some functions plotted");
```

So, it should now plot two plots. So, the blue one is cos x and the orange one is sin x. So, then we can similarly, we can plot as many functions as we like. Suppose now I plot tan x on this suppose I plot tan x on this.

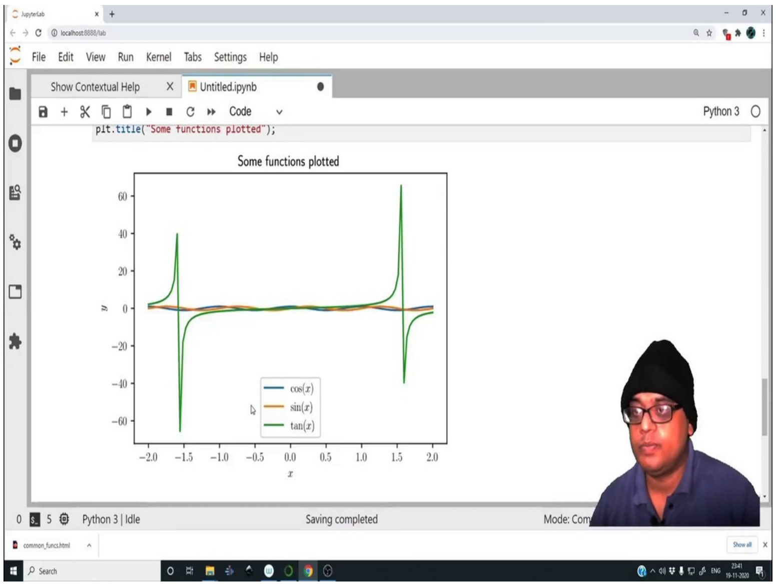
(Refer Slide Time: 21:43)



```
[44]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);
      z = np.sin(2*np.pi*x);

[*]:  plt.plot(x, y, label="$\\cos(x)$");
      plt.plot(x, z, label="$\\sin(x)$");
      plt.plot(x, np.tan(x), label="$\\tan(x)$");
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");

[ ]:
```

In fact, I can directly do the plotting without declaring it to a variable I can do it in line like this I do not need to declare it as a variable every time. So, the label to this will be tan(x) ok. So, what is the error? It should be label equal to ok.

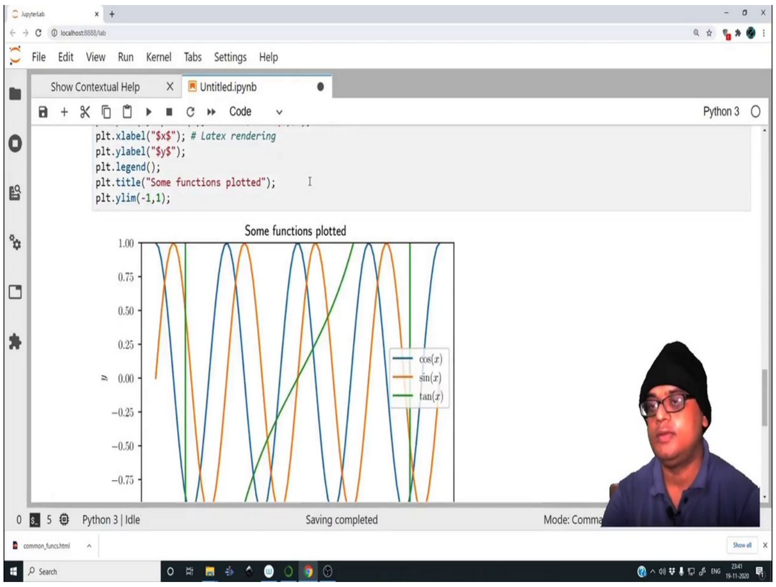
(Refer Slide Time: 22:07)



```
plt.title("Some functions plotted");
```

So, now, the tan x function because it becomes infinite at $\pi/2$. So, now, we want to reduce the y limit of this entire plot. So, how do we do that? So, right now it is blowing up at $\pi/2$ at -$\pi/2$. So, we need to restrict the y limit.

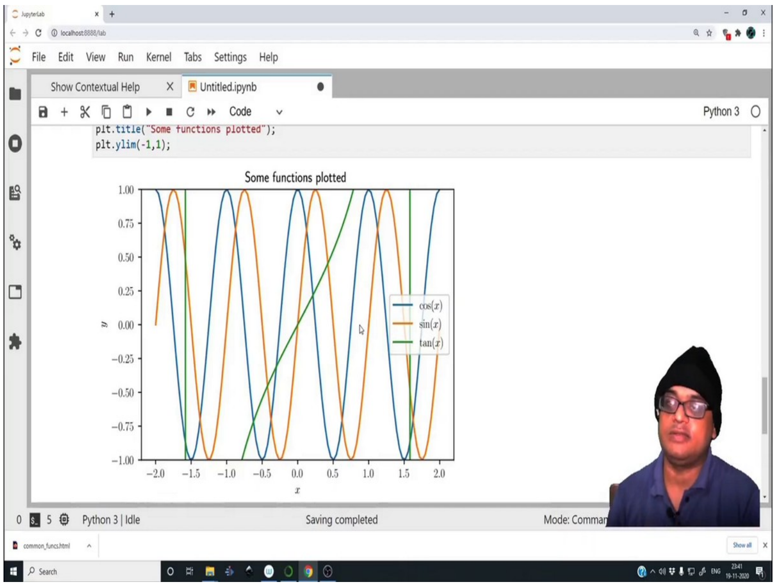
(Refer Slide Time: 22:29)



```
plt.xlabel("$x$"); # Latex rendering
plt.ylabel("$y$");
plt.legend();
plt.title("Some functions plotted");
plt.ylim(-1,1);
```

So, plt.ylim and within bracket we will say -1 to 1 ok.

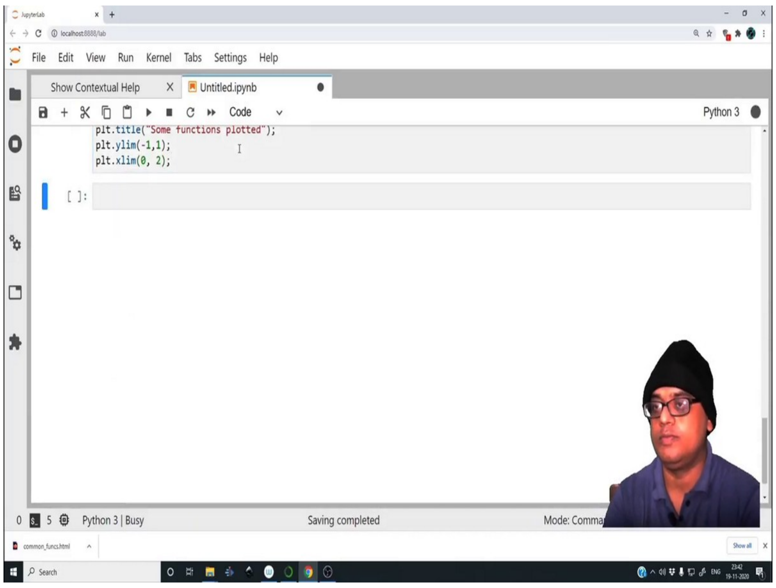
(Refer Slide Time: 22:37)



```
plt.title("Some functions plotted");
plt.ylim(-1,1);
```

So, then it restricts the plot between -1 to 1 in the y axis.

(Refer Slide Time: 22:45)

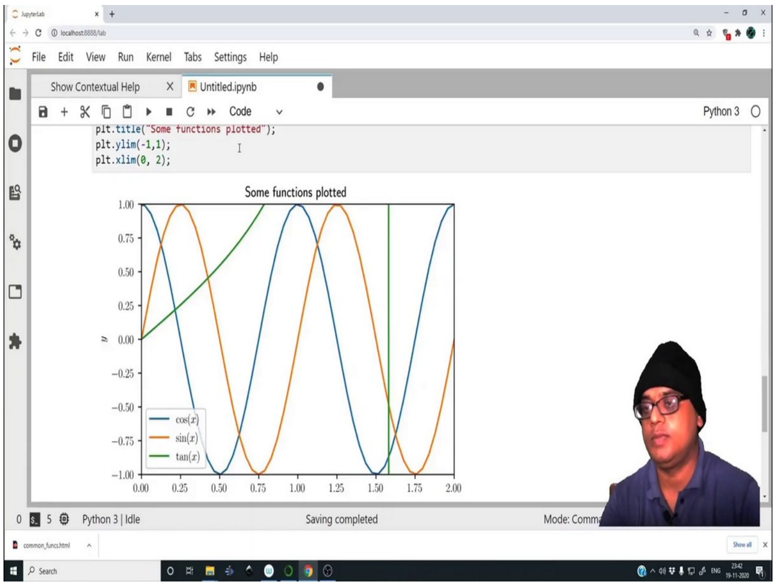


```
plt.title("Some functions plotted");
plt.ylim(-1,1);
plt.xlim(0, 2);
```

```
[ ]:
```

Similarly, if you if we want to only plot 0 to 2, we could have done plt.xlim(0, 2) ok.

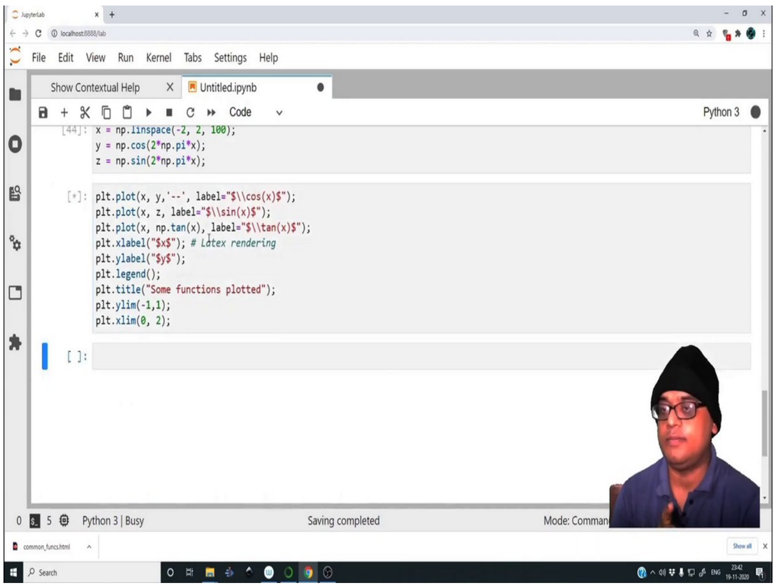
(Refer Slide Time: 22:50)



```
plt.title("Some functions plotted");
plt.ylim(-1,1);
plt.xlim(0, 2);
```

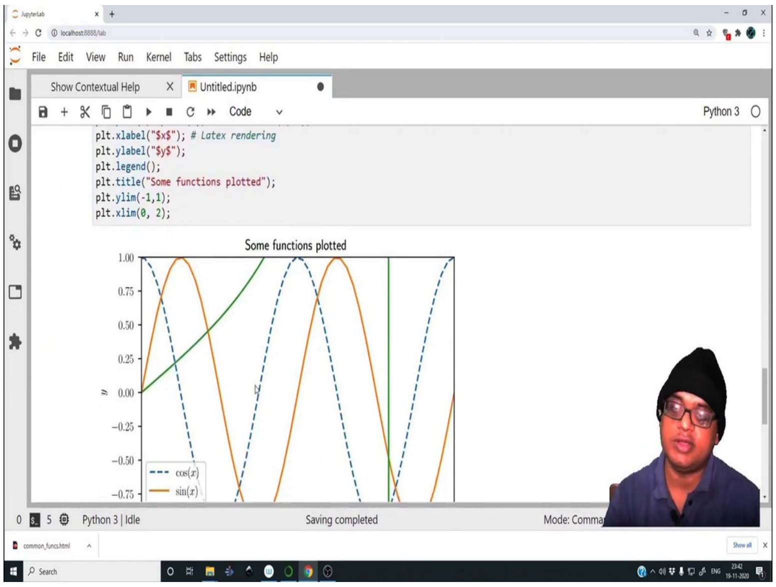
And, by default it is using a colour switcher each time you add a plot it will switch colours between the plot, but I can change the line style as well ok.

(Refer Slide Time: 23:05)



```
[44]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);
      z = np.sin(2*np.pi*x);

[*]:  plt.plot(x, y,'--', label="$\\cos(x)$");
      plt.plot(x, z, label="$\\sin(x)$");
      plt.plot(x, np.tan(x), label="$\\tan(x)$");
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");
      plt.ylim(-1,1);
      plt.xlim(0, 2);

[ ]:
```

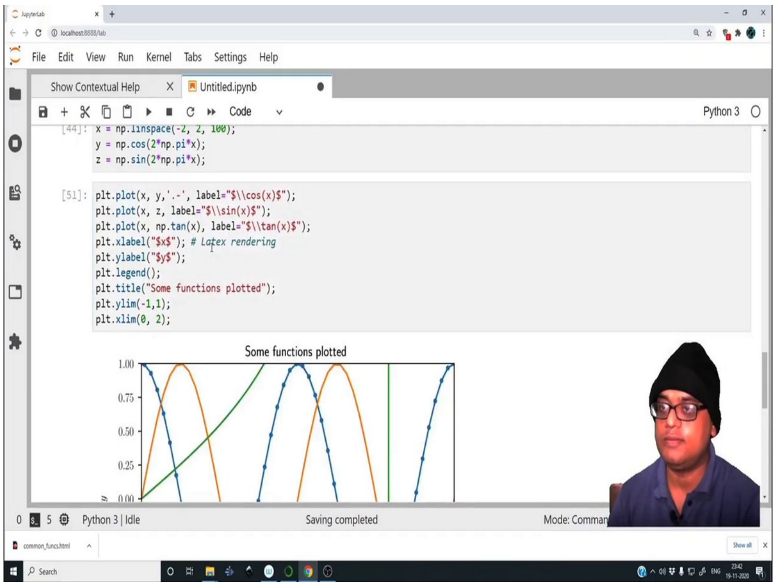
(Refer Slide Time: 23:12)



```
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");
      plt.ylim(-1,1);
      plt.xlim(0, 2);
```

Before the label I just need to specify something like this. So, now, it will plot cos x as a broken line. There you go. It is plotting it as a broken line.

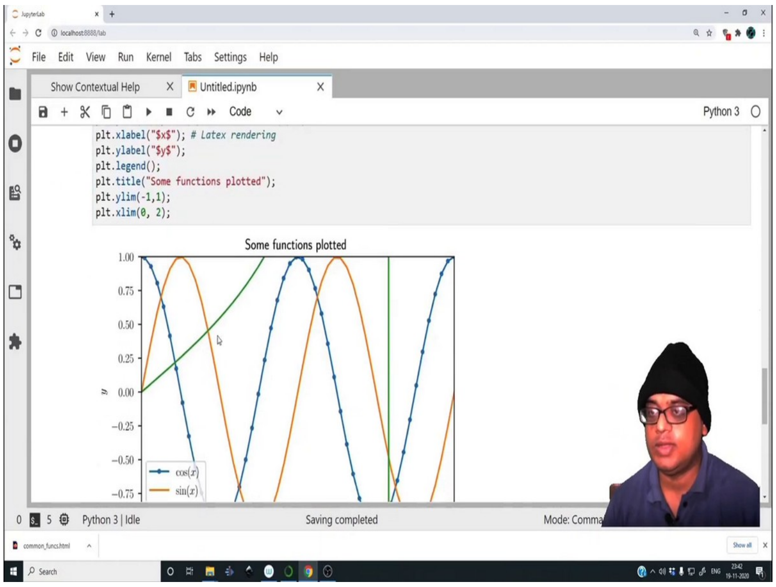
(Refer Slide Time: 23:18)



```
[44]: x = np.linspace(-2, 2, 100);
      y = np.cos(2*np.pi*x);
      z = np.sin(2*np.pi*x);

[51]: plt.plot(x, y,'.-', label="$\\cos(x)$");
      plt.plot(x, z, label="$\\sin(x)$");
      plt.plot(x, np.tan(x), label="$\\tan(x)$");
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");
      plt.ylim(-1,1);
      plt.xlim(0, 2);
```

You can have a dot dash line as well.

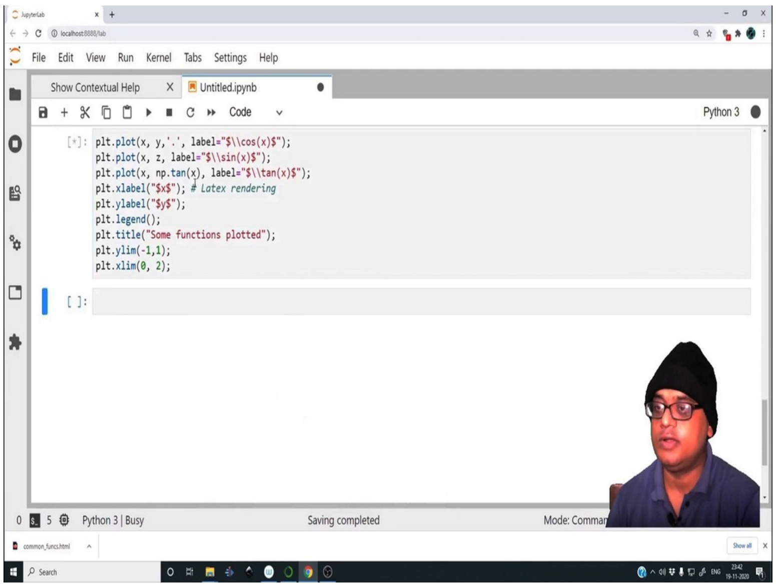
(Refer Slide Time: 23:21)



```
      plt.xlabel("$x$"); # Latex rendering
      plt.ylabel("$y$");
      plt.legend();
      plt.title("Some functions plotted");
      plt.ylim(-1,1);
      plt.xlim(0, 2);
```

Something like this.

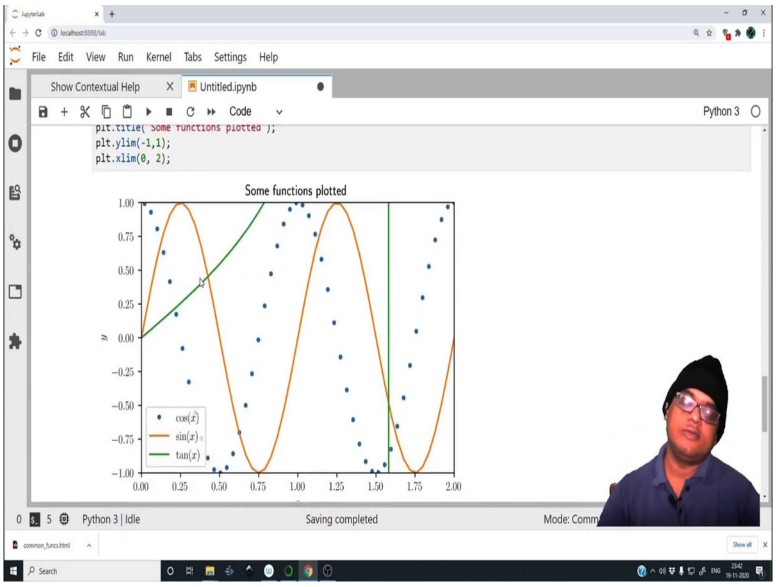
(Refer Slide Time: 23:26)



```
[*]: plt.plot(x, y,'.', label="$\\cos(x)$");
     plt.plot(x, z, label="$\\sin(x)$");
     plt.plot(x, np.tan(x), label="$\\tan(x)$");
     plt.xlabel("$x$"); # Latex rendering
     plt.ylabel("$y$");
     plt.legend();
     plt.title("Some functions plotted");
     plt.ylim(-1,1);
     plt.xlim(0, 2);

[ ]:
```

You can have single dot.

(Refer Slide Time: 23:27)

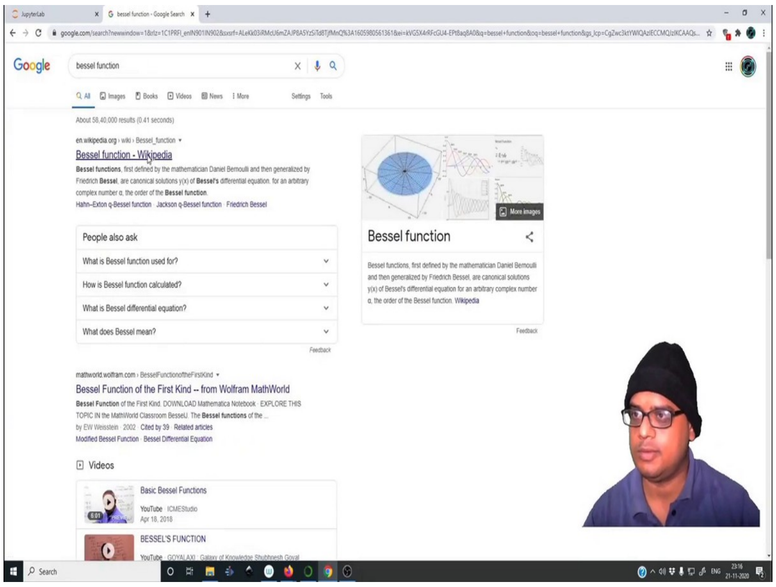


```
plt.title("Some functions plotted");
plt.ylim(-1,1);
plt.xlim(0, 2);
```

You can plot only the dots ok. So, there is various line styles that you can plot and I request you to have a look online because the matplotlib documentation is quite exhaustive its quite large and its not something which we really need to discuss in great details because all we care about is to plot whatever we want to. It is to get a feel; it is not to make something pretty as of now.

Eventually when you start working on a research problem you will try to make plots as descriptive as possible. So, I request you to start looking up the documentation and you eventually you will get the hang of it. So, let us now see how we can plot some special functions using Python in particular we will be plotting the Bessel function.

(Refer Slide Time: 24:20)



bessel function

About 58,40,000 results (0.41 seconds)

en.wikipedia.org › wiki › Bessel_function

Bessel function - Wikipedia

Bessel functions, first defined by the mathematician Daniel Bernoulli and then generalized by Friedrich **Bessel**, are canonical solutions y(x) of **Bessel's** differential equation. for an arbitrary complex number α, the order of the **Bessel function**.

Hahn–Exton q-Bessel function · Jackson q-Bessel function · Friedrich Bessel

People also ask

What is Bessel function used for?

How is Bessel function calculated?

What is Bessel differential equation?

What does Bessel mean?

mathworld.wolfram.com › BesselFunctionoftheFirstKind

Bessel Function of the First Kind -- from Wolfram MathWorld

Bessel Function of the First Kind. DOWNLOAD Mathematica Notebook · EXPLORE THIS TOPIC IN the MathWorld Classroom BesselJ. The **Bessel functions** of the ...

by EW Weisstein · 2002 · Cited by 39 · Related articles

Modified Bessel Function · Bessel Differential Equation

Videos

Basic Bessel Functions
YouTube · ICMEStudio
Apr 18, 2018

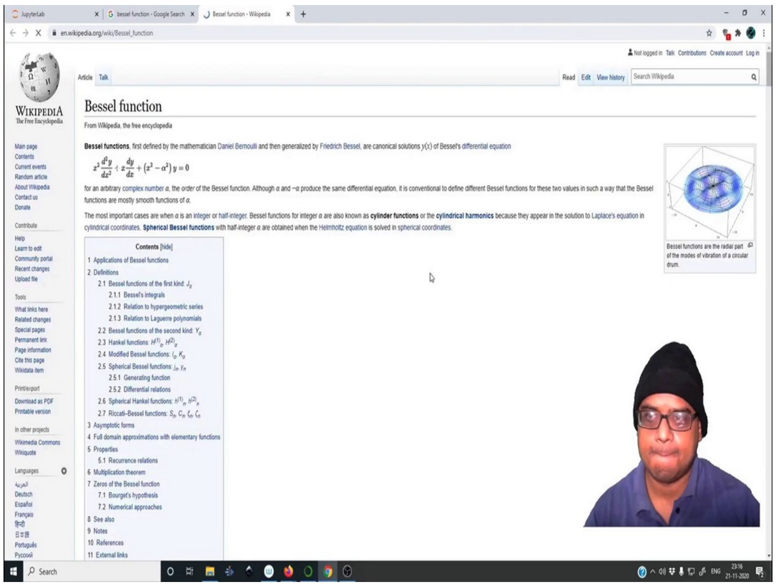
BESSEL'S FUNCTION
YouTube · GOYALAXI · Galaxy of Knowledge Shubhnesh Goyal

Bessel function

Bessel functions, first defined by the mathematician Daniel Bernoulli and then generalized by Friedrich Bessel, are canonical solutions y(x) of Bessel's differential equation for an arbitrary complex number α, the order of the Bessel function. Wikipedia

So, first of all let us see how the Bessel function looks like.

(Refer Slide Time: 24:28)



Bessel function

From Wikipedia, the free encyclopedia

Bessel functions, first defined by the mathematician Daniel Bernoulli and then generalized by Friedrich Bessel, are canonical solutions $y(x)$ of Bessel's differential equation

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-\alpha^2)y=0$$

for an arbitrary complex number α, the *order* of the Bessel function. Although α and $-\alpha$ produce the same differential equation, it is conventional to define different Bessel functions for these two values in such a way that the Bessel functions are mostly smooth functions of α.

The most important cases are when α is an integer or half-integer. Bessel functions for integer α are also known as **cylinder functions** or the **cylindrical harmonics** because they appear in the solution to Laplace's equation in cylindrical coordinates. **Spherical Bessel functions** with half-integer α are obtained when the Helmholtz equation is solved in spherical coordinates.

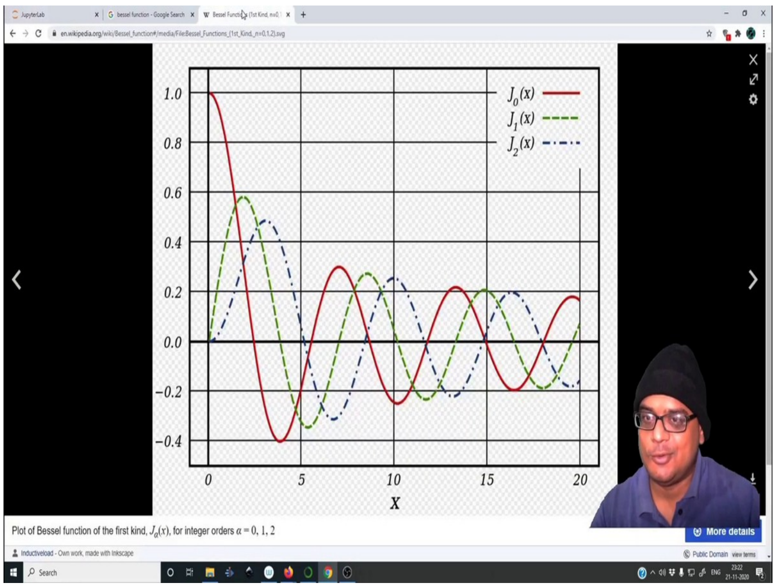
Bessel functions are the radial part of the modes of vibration of a circular drum.

Contents [hide]

1 Applications of Bessel functions
2 Definitions
2.1 Bessel functions of the first kind: J_α
2.1.1 Bessel's integrals
2.1.2 Relation to hypergeometric series
2.1.3 Relation to Laguerre polynomials
2.2 Bessel functions of the second kind: Y_α
2.3 Hankel functions: $H^{(1)}_\alpha$, $H^{(2)}_\alpha$
2.4 Modified Bessel functions: I_α, K_α
2.5 Spherical Bessel functions: j_n, y_n
2.5.1 Generating function
2.5.2 Differential relations
2.6 Spherical Hankel functions: $h^{(1)}_n$, $h^{(2)}_n$
2.7 Riccati–Bessel functions: S_n, C_n, ξ_n, ζ_n
3 Asymptotic forms
4 Full domain approximations with elementary functions
5 Properties
5.1 Recurrence relations
6 Multiplication theorem
7 Zeros of the Bessel function
7.1 Bourget's hypothesis
7.2 Numerical approaches
8 See also
9 Notes
10 References
11 External links

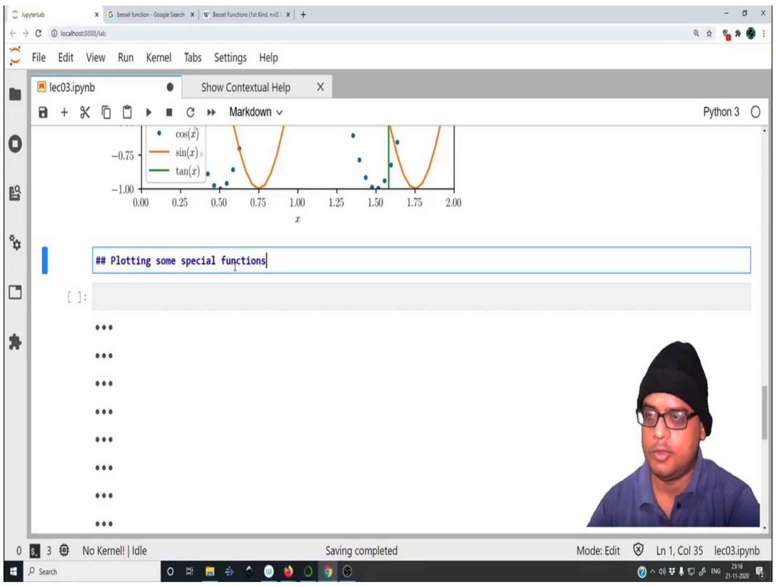
Rather let us go just have a look I mean ok.

(Refer Slide Time: 24:32)


$J_0(x)$
$J_1(x)$
$J_2(x)$
X
Plot of Bessel function of the first kind, $J_\alpha(x)$, for integer orders $\alpha = 0, 1, 2$
More details

So, this is the plot we are targeting; so, J 0, J 1, J 2. So, J is the Bessel function of the first kind and 0, 1, 2 they are the orders of the Bessel function.

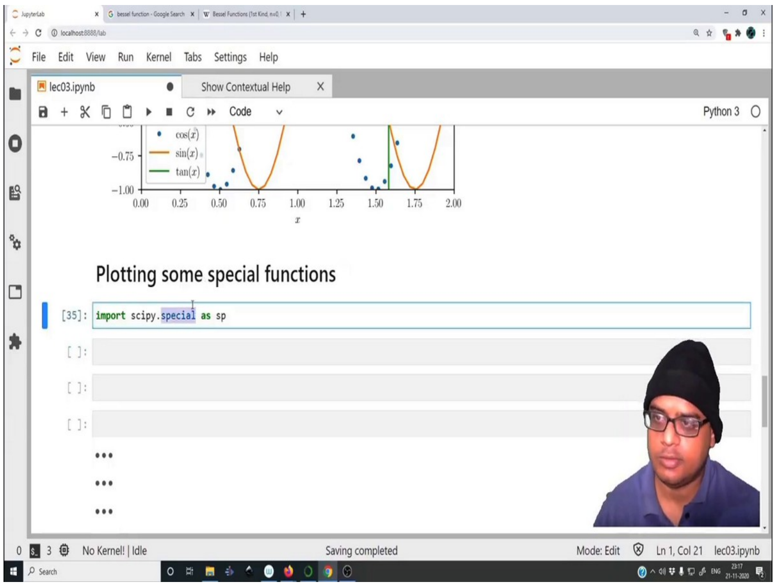
(Refer Slide Time: 24:52)



```
## Plotting some special functions
```

So, let us try to recreate this particular plot. So, first of all let me create a Markdown cell.

(Refer Slide Time: 25:00)

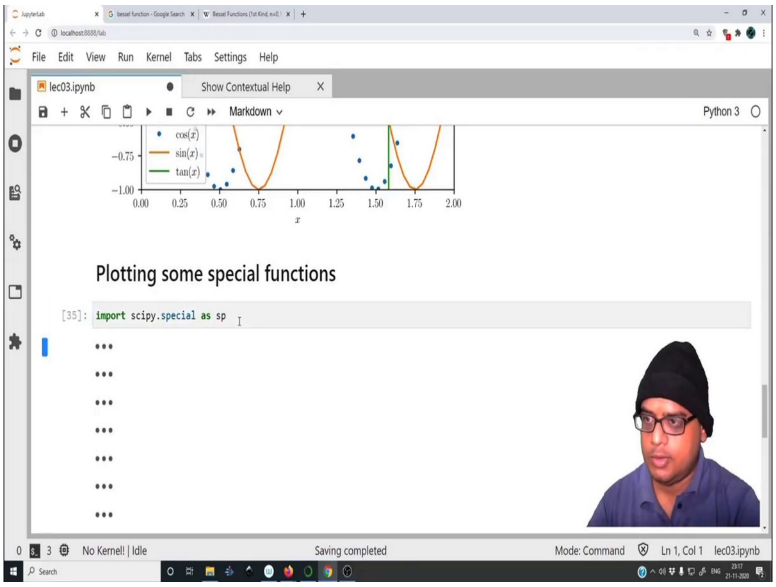


Plotting some special functions

```
[35]: import scipy.special as sp
```

So, the reason why I am calling it special function is because in Python these are all included in the module called as scipy. So, scipy is an abbreviation for scientific Python. And it contains a bunch of tools which you will find immensely useful for scientific computation. So, let us go ahead and import scipy dot special as sp.

(Refer Slide Time: 25:28)

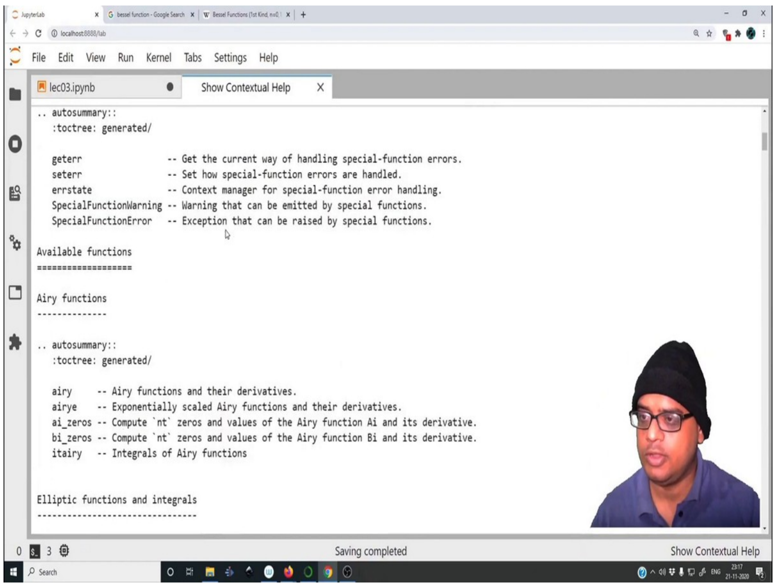


Plotting some special functions

```
[35]: import scipy.special as sp
```

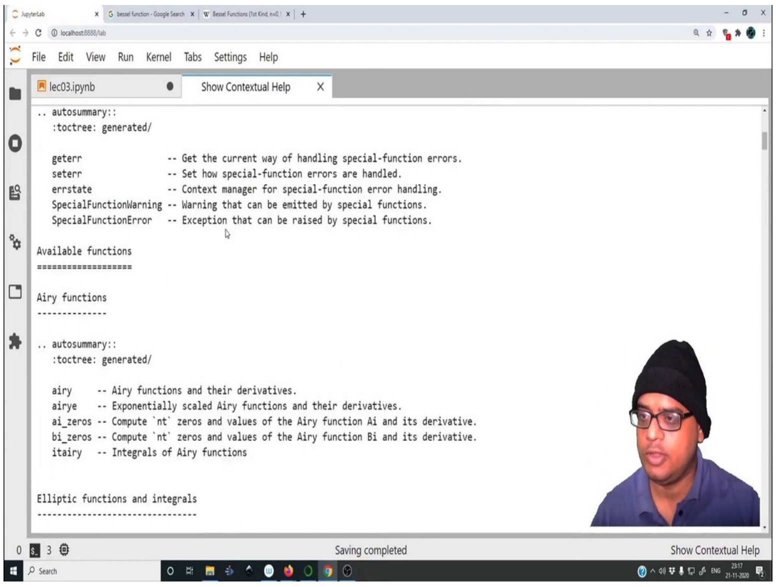
And, so, once we do that we can double click on this special.

(Refer Slide Time: 25:44)



```
.. autosummary::
   :toctree: generated/

   geterr                 -- Get the current way of handling special-function errors.
   seterr                 -- Set how special-function errors are handled.
   errstate               -- Context manager for special-function error handling.
   SpecialFunctionWarning -- Warning that can be emitted by special functions.
   SpecialFunctionError   -- Exception that can be raised by special functions.

Available functions
===================

Airy functions
--------------

.. autosummary::
   :toctree: generated/

   airy     -- Airy functions and their derivatives.
   airye    -- Exponentially scaled Airy functions and their derivatives.
   ai_zeros -- Compute `nt` zeros and values of the Airy function Ai and its derivative.
   bi_zeros -- Compute `nt` zeros and values of the Airy function Bi and its derivative.
   itairy   -- Integrals of Airy functions


Elliptic functions and integrals
--------------------------------
```

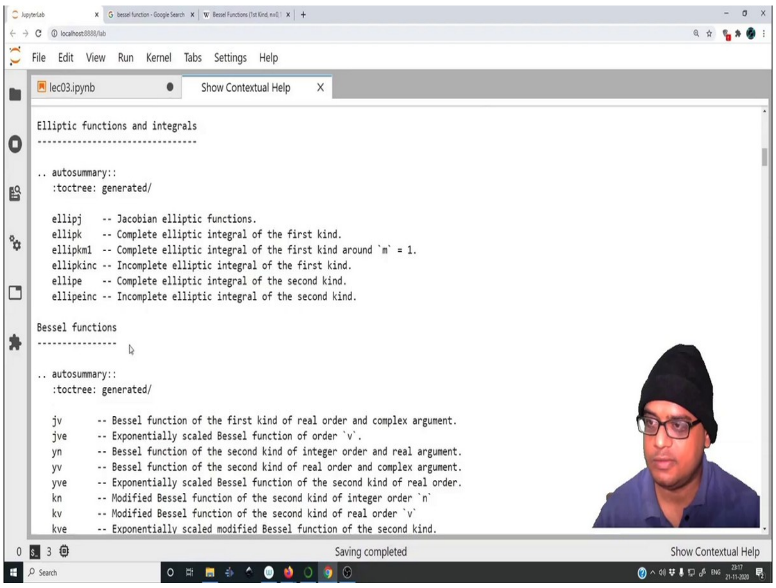
(Refer Slide Time: 25:46)



```
.. autosummary::
   :toctree: generated/

   geterr                 -- Get the current way of handling special-function errors.
   seterr                 -- Set how special-function errors are handled.
   errstate               -- Context manager for special-function error handling.
   SpecialFunctionWarning -- Warning that can be emitted by special functions.
   SpecialFunctionError   -- Exception that can be raised by special functions.

Available functions
===================

Airy functions
--------------

.. autosummary::
   :toctree: generated/

   airy     -- Airy functions and their derivatives.
   airye    -- Exponentially scaled Airy functions and their derivatives.
   ai_zeros -- Compute `nt` zeros and values of the Airy function Ai and its derivative.
   bi_zeros -- Compute `nt` zeros and values of the Airy function Bi and its derivative.
   itairy   -- Integrals of Airy functions


Elliptic functions and integrals
--------------------------------
```

And, we can look up the contextual help to see all the functions that are available to us to plot or that to use.

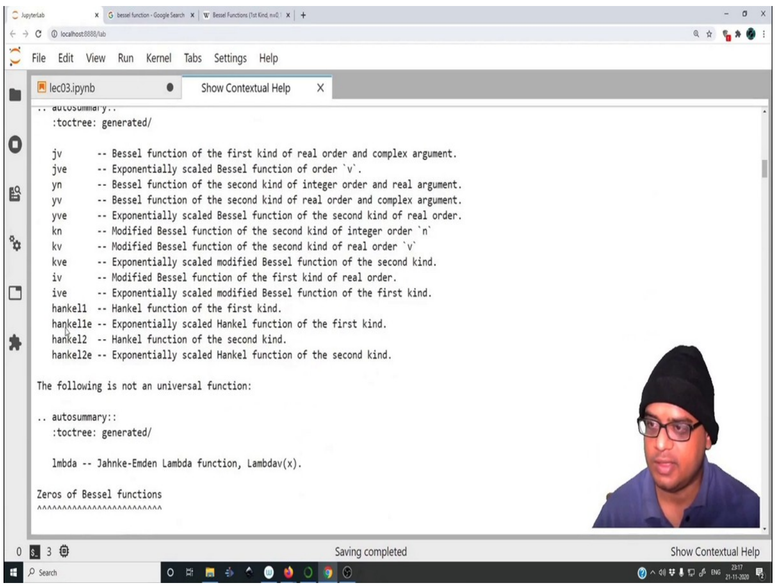
(Refer Slide Time: 25:53)



```
Elliptic functions and integrals
--------------------------------

.. autosummary::
   :toctree: generated/

   ellipj    -- Jacobian elliptic functions.
   ellipk    -- Complete elliptic integral of the first kind.
   ellipkm1  -- Complete elliptic integral of the first kind around `m` = 1.
   ellipkinc -- Incomplete elliptic integral of the first kind.
   ellipe    -- Complete elliptic integral of the second kind.
   ellipeinc -- Incomplete elliptic integral of the second kind.

Bessel functions
----------------

.. autosummary::
   :toctree: generated/

   jv       -- Bessel function of the first kind of real order and complex argument.
   jve      -- Exponentially scaled Bessel function of order `v`.
   yn       -- Bessel function of the second kind of integer order and real argument.
   yv       -- Bessel function of the second kind of real order and complex argument.
   yve      -- Exponentially scaled Bessel function of the second kind of real order.
   kn       -- Modified Bessel function of the second kind of integer order `n`
   kv       -- Modified Bessel function of the second kind of real order `v`
   kve      -- Exponentially scaled modified Bessel function of the second kind.
```

So, we have the array functions, elliptic functions, elliptic integrals, Bessel function.

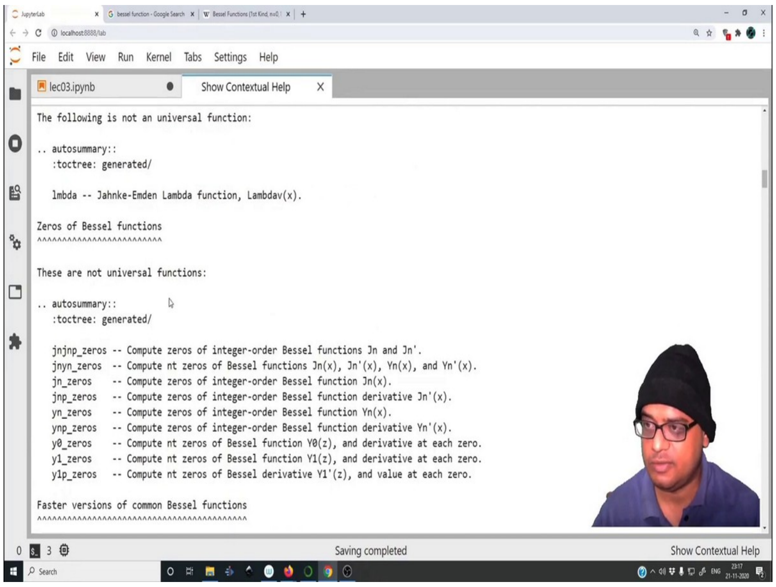
(Refer Slide Time: 25:57)



```
   :toctree: generated/

   jv       -- Bessel function of the first kind of real order and complex argument.
   jve      -- Exponentially scaled Bessel function of order `v`.
   yn       -- Bessel function of the second kind of integer order and real argument.
   yv       -- Bessel function of the second kind of real order and complex argument.
   yve      -- Exponentially scaled Bessel function of the second kind of real order.
   kn       -- Modified Bessel function of the second kind of integer order `n`
   kv       -- Modified Bessel function of the second kind of real order `v`
   kve      -- Exponentially scaled modified Bessel function of the second kind.
   iv       -- Modified Bessel function of the first kind of real order.
   ive      -- Exponentially scaled modified Bessel function of the first kind.
   hankel1  -- Hankel function of the first kind.
   hankel1e -- Exponentially scaled Hankel function of the first kind.
   hankel2  -- Hankel function of the second kind.
   hankel2e -- Exponentially scaled Hankel function of the second kind.

The following is not an universal function:

.. autosummary::
   :toctree: generated/

   lmbda -- Jahnke-Emden Lambda function, Lambdav(x).

Zeros of Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^
```

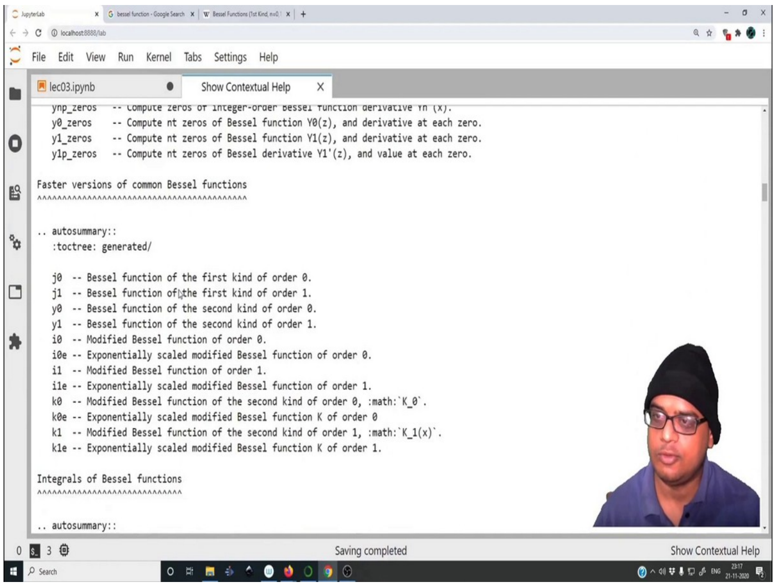
So, in particular we will be interested in jv. So, jv is the Bessel function of the first kind of real order and complex argument. This hankel functions.

(Refer Slide Time: 26:08)



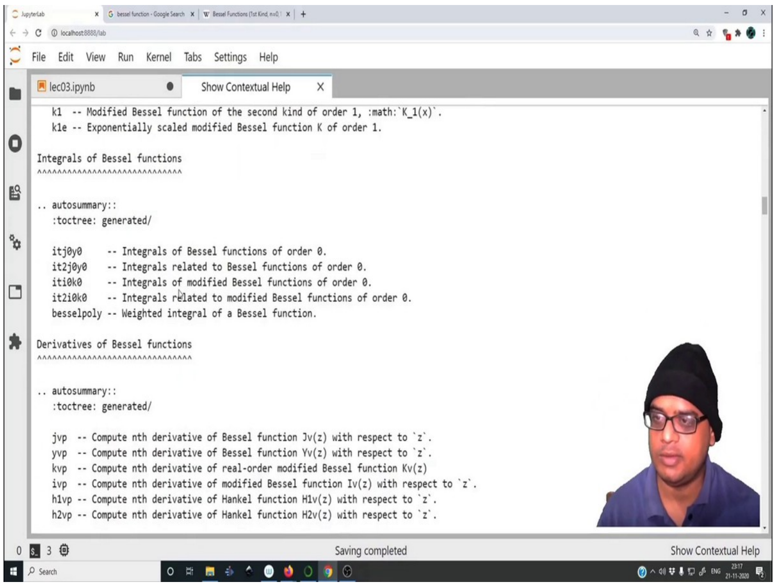
```
The following is not an universal function:

.. autosummary::
   :toctree: generated/

   lmbda -- Jahnke-Emden Lambda function, Lambdav(x).

Zeros of Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^

These are not universal functions:

.. autosummary::
   :toctree: generated/

   jnjnp_zeros -- Compute zeros of integer-order Bessel functions Jn and Jn'.
   jnyn_zeros  -- Compute nt zeros of Bessel functions Jn(x), Jn'(x), Yn(x), and Yn'(x).
   jn_zeros    -- Compute zeros of integer-order Bessel function Jn(x).
   jnp_zeros   -- Compute zeros of integer-order Bessel function derivative Jn'(x).
   yn_zeros    -- Compute zeros of integer-order Bessel function Yn(x).
   ynp_zeros   -- Compute zeros of integer-order Bessel function derivative Yn'(x).
   y0_zeros    -- Compute nt zeros of Bessel function Y0(z), and derivative at each zero.
   y1_zeros    -- Compute nt zeros of Bessel function Y1(z), and derivative at each zero.
   y1p_zeros   -- Compute nt zeros of Bessel derivative Y1'(z), and value at each zero.

Faster versions of common Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

(Refer Slide Time: 26:10)



```
   ynp_zeros   -- Compute zeros of integer-order Bessel function derivative Yn'(x).
   y0_zeros    -- Compute nt zeros of Bessel function Y0(z), and derivative at each zero.
   y1_zeros    -- Compute nt zeros of Bessel function Y1(z), and derivative at each zero.
   y1p_zeros   -- Compute nt zeros of Bessel derivative Y1'(z), and value at each zero.

Faster versions of common Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

.. autosummary::
   :toctree: generated/

   j0  -- Bessel function of the first kind of order 0.
   j1  -- Bessel function of the first kind of order 1.
   y0  -- Bessel function of the second kind of order 0.
   y1  -- Bessel function of the second kind of order 1.
   i0  -- Modified Bessel function of order 0.
   i0e -- Exponentially scaled modified Bessel function of order 0.
   i1  -- Modified Bessel function of order 1.
   i1e -- Exponentially scaled modified Bessel function of order 1.
   k0  -- Modified Bessel function of the second kind of order 0, :math:`K_0`.
   k0e -- Exponentially scaled modified Bessel function K of order 0
   k1  -- Modified Bessel function of the second kind of order 1, :math:`K_1(x)`.
   k1e -- Exponentially scaled modified Bessel function K of order 1.

Integrals of Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

.. autosummary::
```

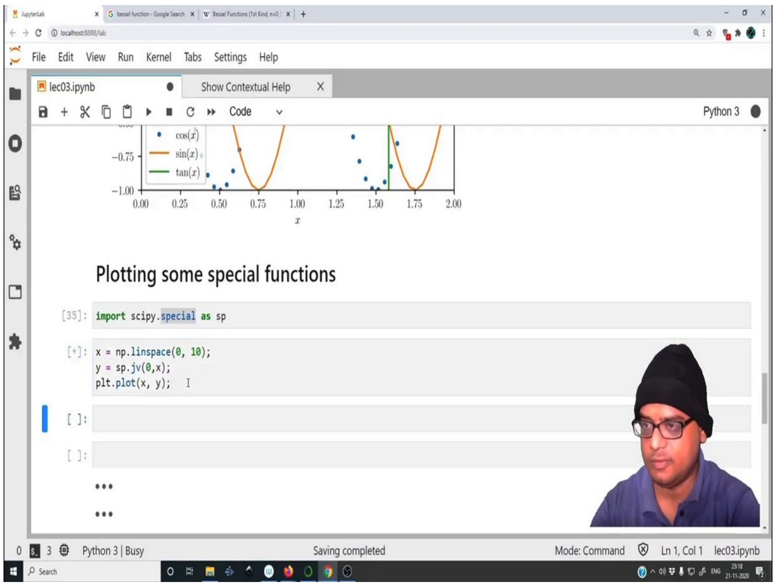
(Refer Slide Time: 26:12)



```
 k1  -- Modified Bessel function of the second kind of order 1, :math:`K_1(x)`.
 k1e -- Exponentially scaled modified Bessel function K of order 1.

Integrals of Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

.. autosummary::
   :toctree: generated/

   itj0y0     -- Integrals of Bessel functions of order 0.
   it2j0y0    -- Integrals related to Bessel functions of order 0.
   iti0k0     -- Integrals of modified Bessel functions of order 0.
   it2i0k0    -- Integrals related to modified Bessel functions of order 0.
   besselpoly -- Weighted integral of a Bessel function.

Derivatives of Bessel functions
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

.. autosummary::
   :toctree: generated/

   jvp  -- Compute nth derivative of Bessel function Jv(z) with respect to `z`.
   yvp  -- Compute nth derivative of Bessel function Yv(z) with respect to `z`.
   kvp  -- Compute nth derivative of real-order modified Bessel function Kv(z)
   ivp  -- Compute nth derivative of modified Bessel function Iv(z) with respect to `z`.
   h1vp -- Compute nth derivative of Hankel function H1v(z) with respect to `z`.
   h2vp -- Compute nth derivative of Hankel function H2v(z) with respect to `z`.
```

The Zeros of Bessel functions. So, there is a whole bunch of functions that are available and you can see all those in the contextual help.

(Refer Slide Time: 26:20)

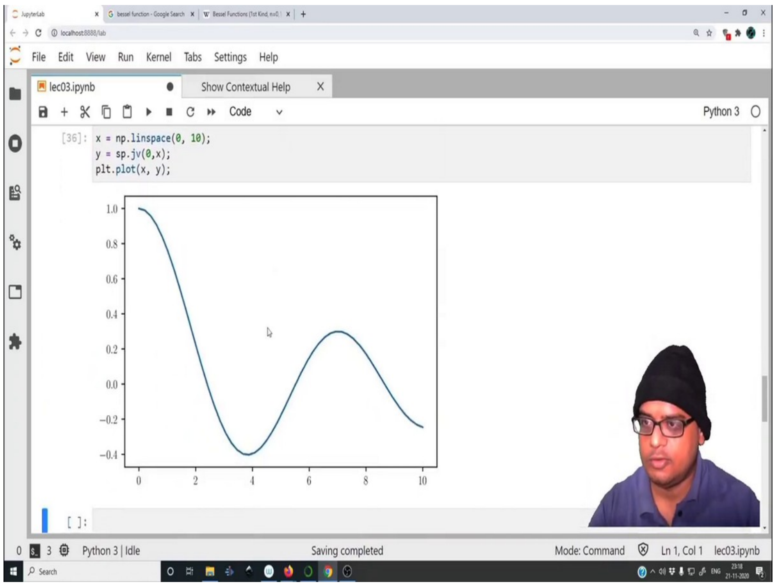


Plotting some special functions

```
[35]: import scipy.special as sp

[*]: x = np.linspace(0, 10);
     y = sp.jv(0,x);
     plt.plot(x, y);

[ ]:

[ ]:
```

So, let us proceed let us define first the domain over which we want to evaluate the Bessel function. So, x = np.linspace(0, 10), let y = sp.jv(0,x). So, this is the order of this Bessel function of the first kind and x is the domain over which you want to evaluate. So, let us go ahead and do a simple plot. So, plt.plot(x, y). Let us see what this gives.

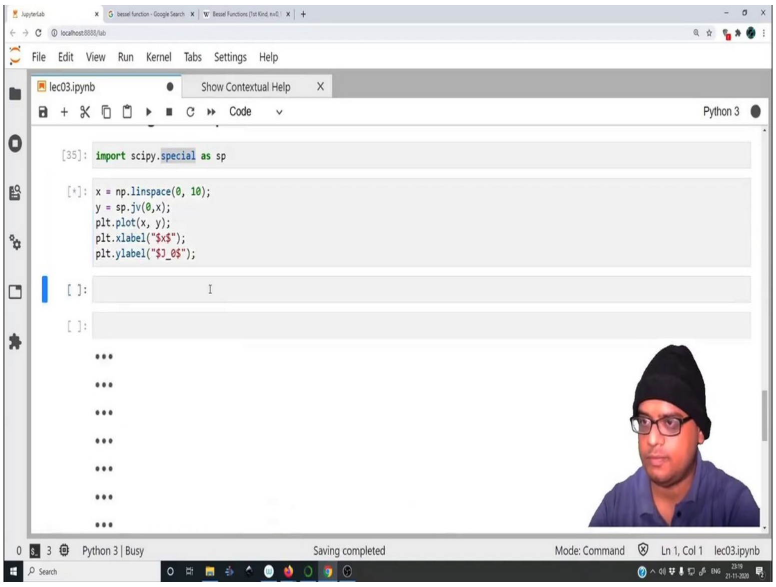
(Refer Slide Time: 27:00)



```
[36]: x = np.linspace(0, 10);
      y = sp.jv(0,x);
      plt.plot(x, y);
```

So, there you go this is the zeroth order Bessel function of the first kind.

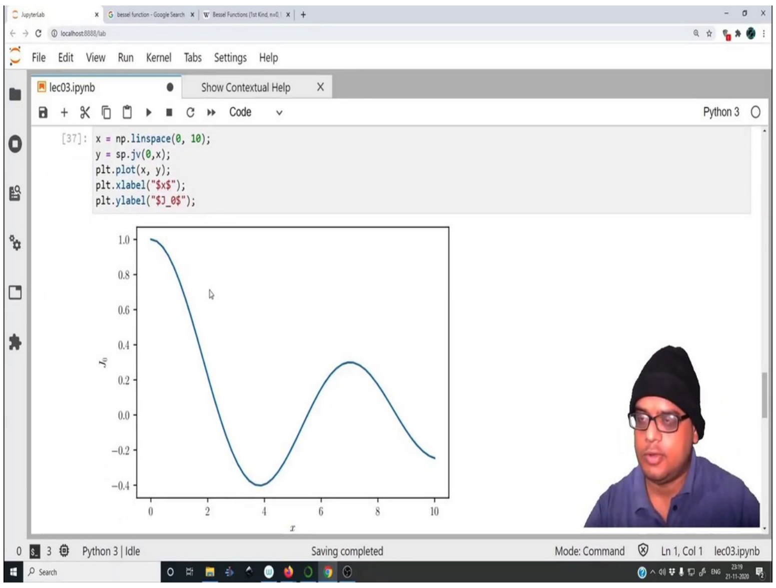
(Refer Slide Time: 27:09)



```
[35]: import scipy.special as sp

[*]: x = np.linspace(0, 10);
     y = sp.jv(0,x);
     plt.plot(x, y);
     plt.xlabel("$x$");
     plt.ylabel("$J_0$");
```

So, let us put some label. So, plt.xlabel("x") and plt.ylabel("J_0") in this particular case.

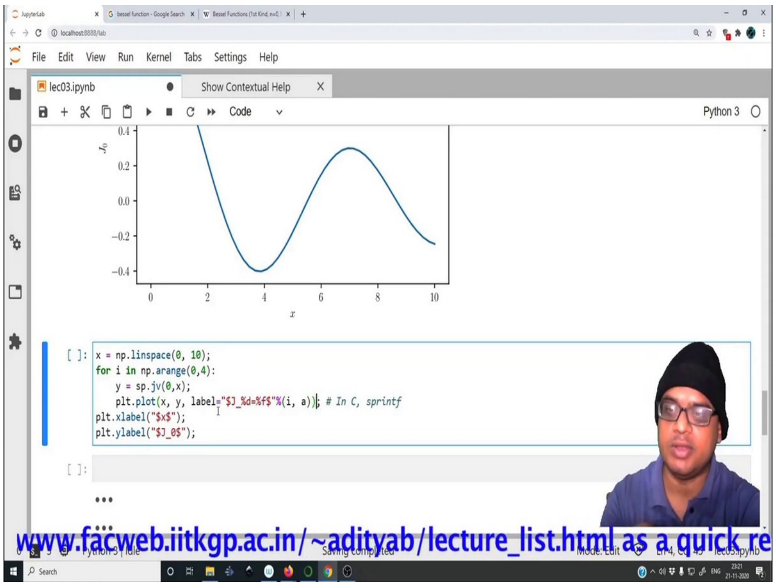
(Refer Slide Time: 27:30)



```
[37]: x = np.linspace(0, 10);
      y = sp.jv(0,x);
      plt.plot(x, y);
      plt.xlabel("$x$");
      plt.ylabel("$J_0$");
```

So, now let us try to plot various orders through a loop ok. So, once x is defined we do not really want to do anything with that.

(Refer Slide Time: 27:50)

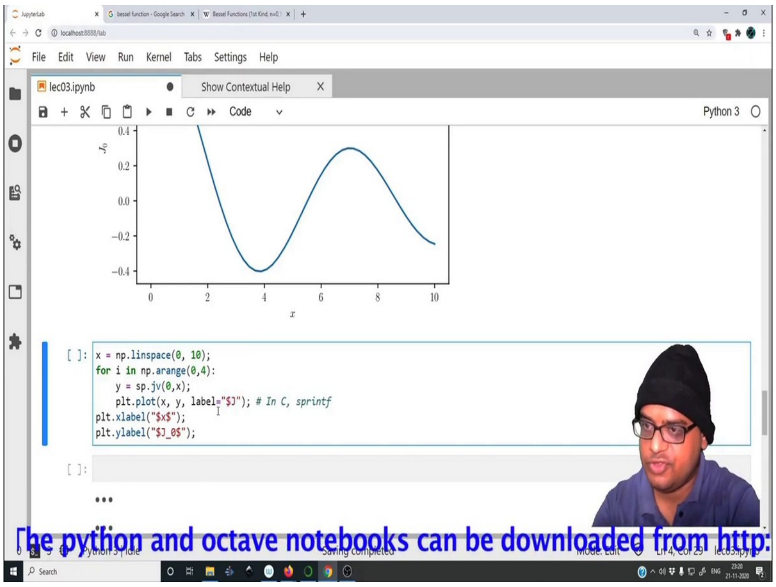


```
[ ]: x = np.linspace(0, 10);
     for i in np.arange(0,4):
         y = sp.jv(0,x);
         plt.plot(x, y, label="$J_%d=%f$"%(i, a)); # In C, sprintf
     plt.xlabel("$x$");
     plt.ylabel("$J_0$");
```

www.facweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick re

So, let me just copy this piece of code. So, that how to plot a single function is available to you later on as well. So, let me define a for loop. So, for i in np.arange(0,4). So, that we go from 0, 1, 2, 3 ok. So, y will be defined in the loop itself. So, I need to indent it with respect to the for. Remember that anything that is indented is inside the for loop then we will plot this. So, the ylabels can be outside the loop because they need to be executed only once.

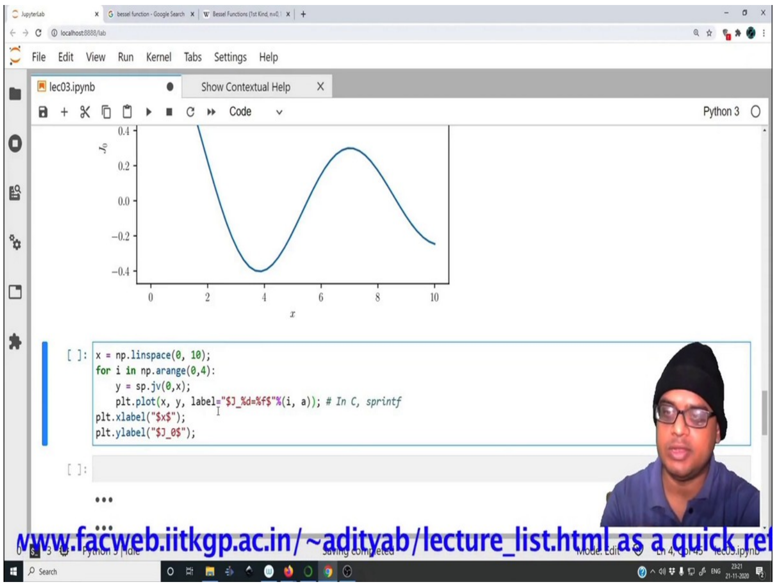
(Refer Slide Time: 28:39)



```
x = np.linspace(0, 10);
for i in np.arange(0,4):
    y = sp.jv(0,x);
    plt.plot(x, y, label="$J"); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("$J_0$");
```

So, we do not need to indent those lines we need to plot it, but we will give them a label. So, label equal to. So, for each order that we plot we need to upgrade the label. So, in programming languages like C you would ideally use something called as a sprintf. So, in C I would use sprintf, but in Python you do not need to use that we can simply make use of a drop in.

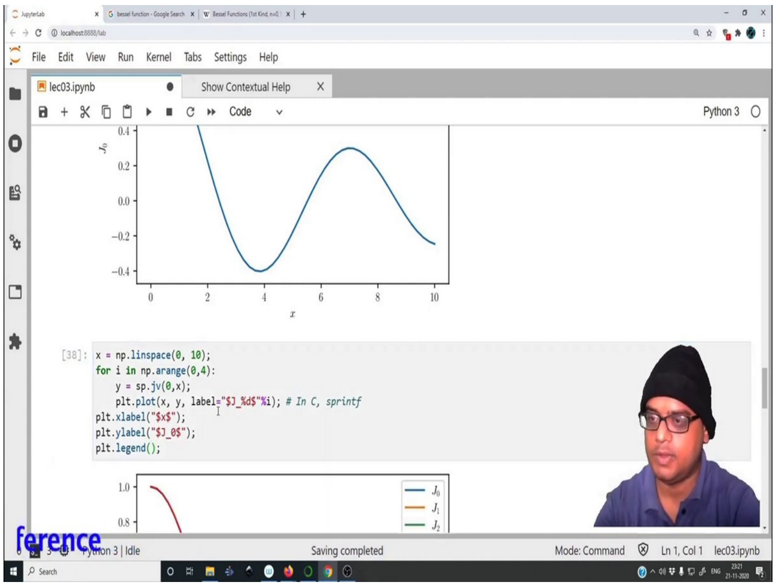
(Refer Slide Time: 29:10)



```
x = np.linspace(0, 10);
for i in np.arange(0,4):
    y = sp.jv(0,x);
    plt.plot(x, y, label="$J_%d=%f$"%(i, a)); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("$J_0$");
```

So, $J_%d$ and after this we want to replace the % d with the integer i. So, another % sign and i if there are multiple drop-ins that you want to do. So, suppose there was something like equal to % f for example, then you would simply put a bracket and put the name of the variable that you want to plot that you want to drop-in in place of the % f.

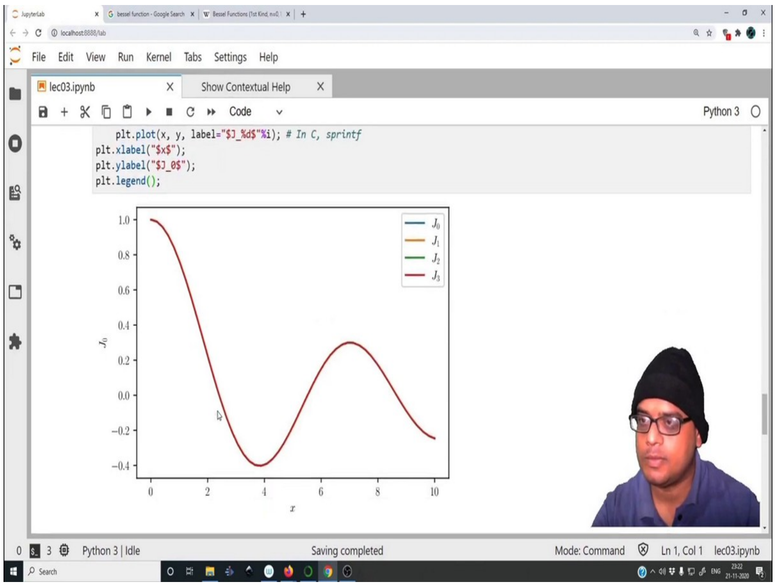
(Refer Slide Time: 29:47)



```
[38]: x = np.linspace(0, 10);
      for i in np.arange(0,4):
          y = sp.jv(0,x);
          plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
      plt.xlabel("$x$");
      plt.ylabel("$J_0$");
      plt.legend();
```

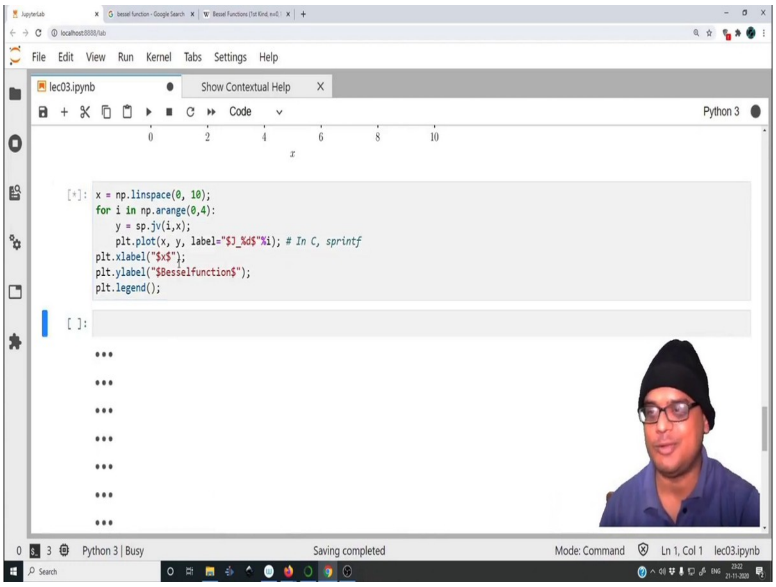
In this case we do not want anything so, I just do this ok. So, let me close the bracket. So, this will go in a loop. For each i it will plot the corresponding Bessel function and then in the end it will create the label and in the end we want to also make the legend. So, plt.legend.

(Refer Slide Time: 30:15)



```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("$J_0$");
plt.legend();
```

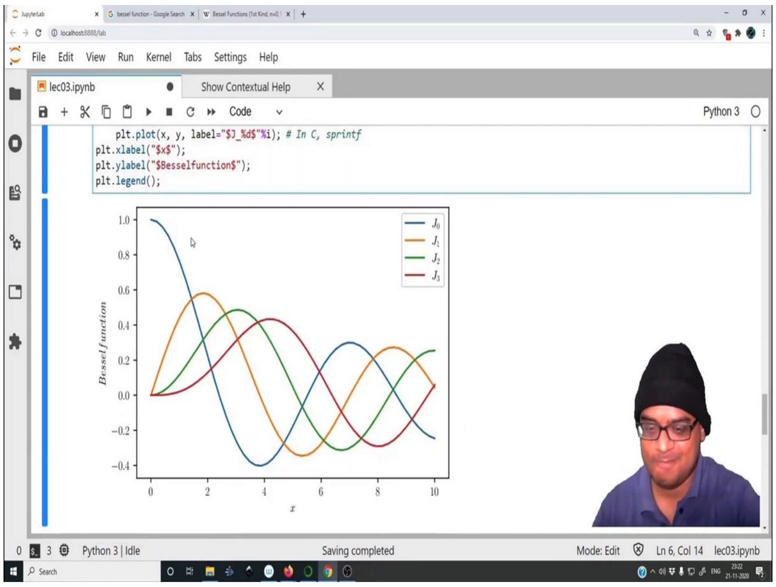
(Refer Slide Time: 30:35)



```
[*]: x = np.linspace(0, 10);
     for i in np.arange(0,4):
         y = sp.jv(i,x);
         plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
     plt.xlabel("$x$");
     plt.ylabel("$Besselfunction$");
     plt.legend();
```

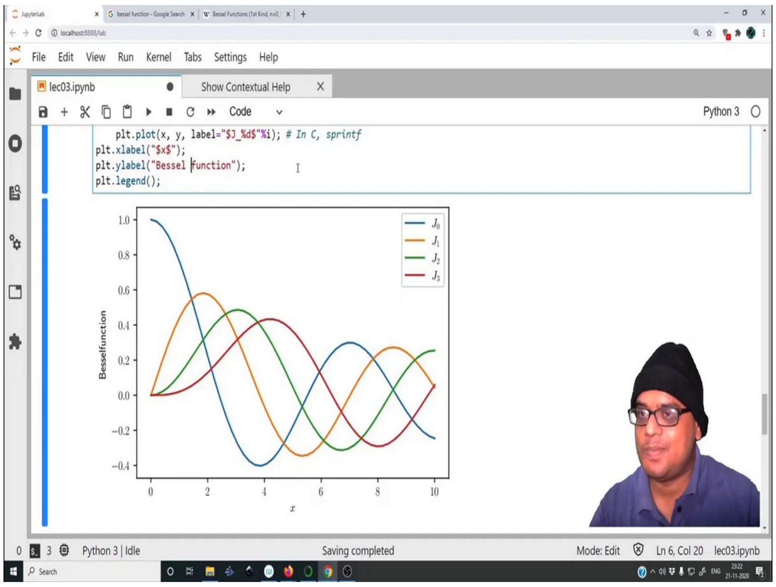
So, when we run this it just shows ok, this should be only. So, we have only we forgot to substitute this 0 by i because it went into the loop each time it just put 0. So, it is a classic mistake because we are using we are trying to convert we are trying to reuse code that we have already written. So, do not make that mistake. So, we need to substitute that 0 by an i. So, that every time we go in the loop we have a new i and we make a new plot great.

(Refer Slide Time: 31:01)



```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("$Besselfunction$");
plt.legend();
```

(Refer Slide Time: 31:06)

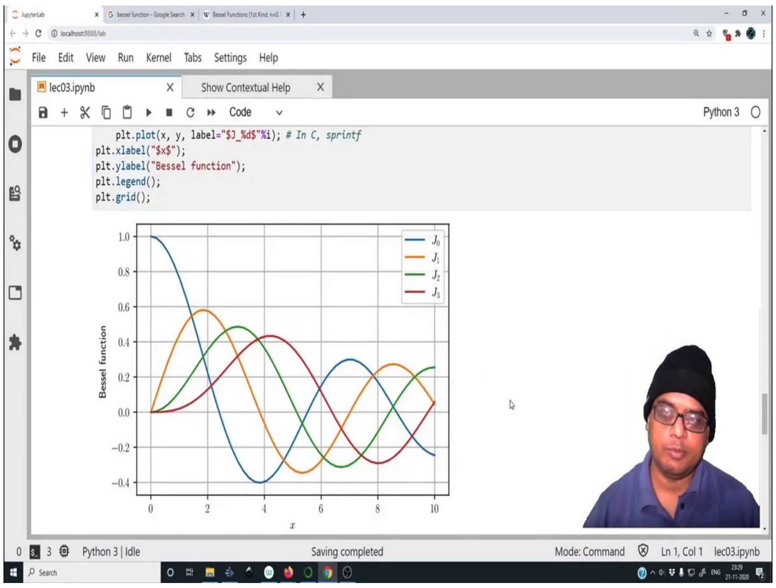


```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
```

So, this is the this is how the function looks like. We do not need $ over here great. So, let us go to what the plot looks like. So, solid line, dashed line, dash I mean forget about it. So, this is how you can generate the plot of Bessel function as shown in Wikipedia and we have done it for J 0, J 1, J 2 and J 3 and you can practically plot any function you want we have seen in the contextual help a bunch of functions that are available to us.

But, apart from plotting these Bessel functions are very important for problems which involve cylindrical polar coordinate systems. These appears these appear as solutions to many of those differential equations. So, while I am just showing you some of the functions available in the scipy dot special sub module they do have a very deep physical meaning and that is why they are included as special functions.

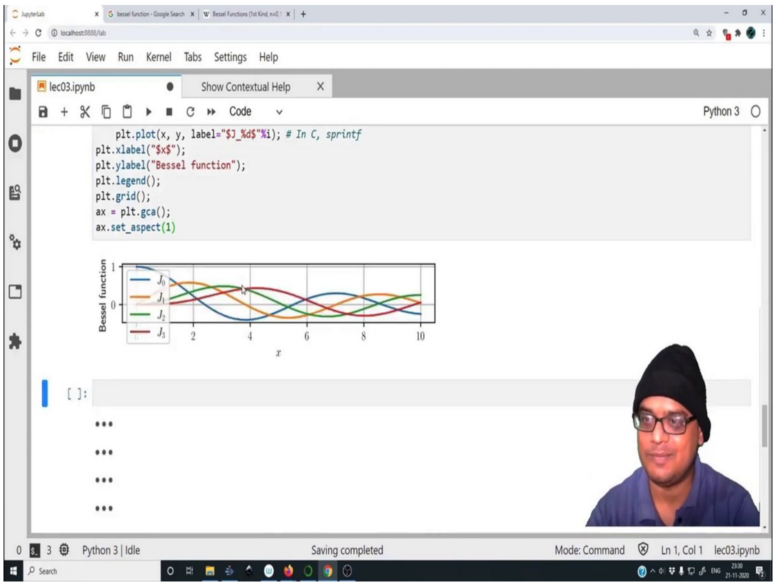
(Refer Slide Time: 32:08)



```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
plt.grid();
```

Now, in this particular plot we see that the extent of the y axis is from - 0.421 while the extent of the x axis is from 0 to 10, but the scale of the x axis. So, it goes from 0 to 10 it looks this big while the y axis goes from - 0.421 and it looks it is much more magnified than the x axis meaning 1 unit on the x-axis ≠ 1 unit on the y-axis. So, in order to show the true aspect ratio of the plot we can do the following.

(Refer Slide Time: 32:51)

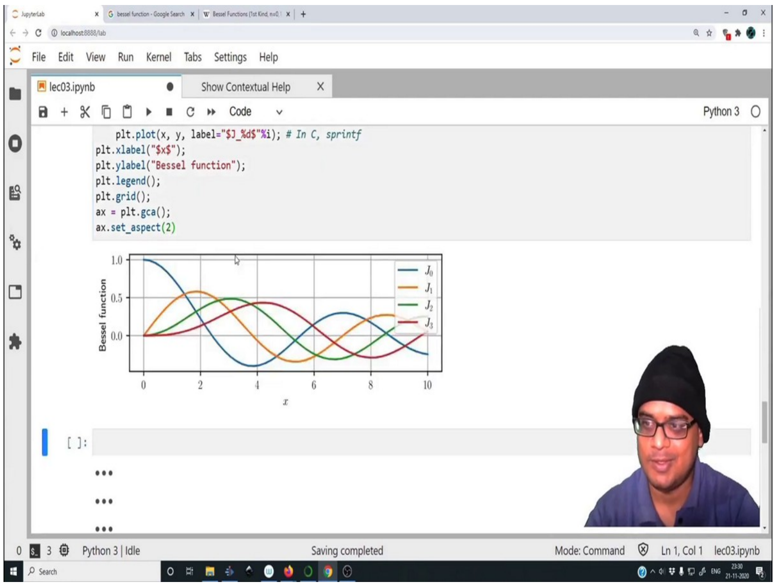


```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
plt.grid();
ax = plt.gca();
ax.set_aspect(1)
```

We can first of all query the axis. So, we can say ax = plt.gca(). So, gca is an abbreviation for get current access. So, once we have the object ax we can set the various properties of ax. So, ax.set_aspect()and we have to put a number over here.

So, if you put 1 it means the scale of the y-axis divided by the scale of the x-axis is equal to 1. So, let us see what happens. So, once we do that we see that now one unit on the y axis is equal to exactly one unit on the x axis, but now the plot is too compressed for our liking let me make it 2.

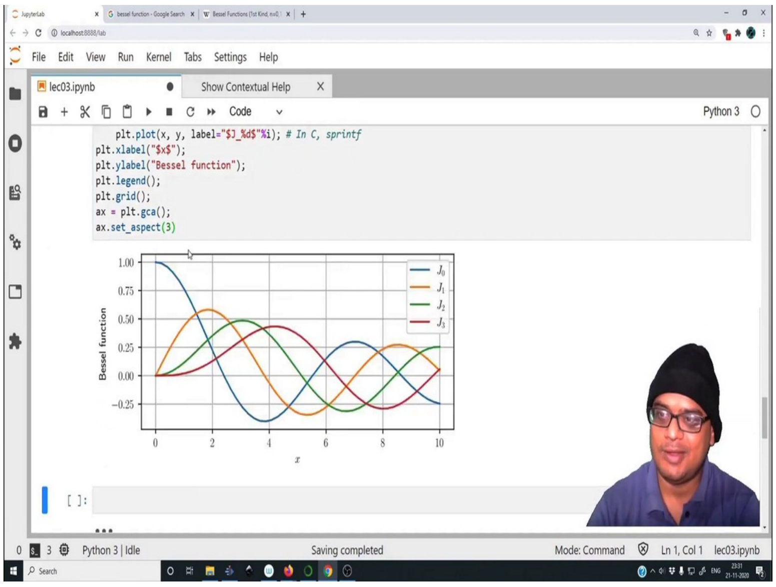
(Refer Slide Time: 33:31)



```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
plt.grid();
ax = plt.gca();
ax.set_aspect(2)
```

So, the y-axis is scaled twice as compared to the x-axis. So, by doing this we can draw attention to some of the features that may otherwise be visibly absent in the plot ok.

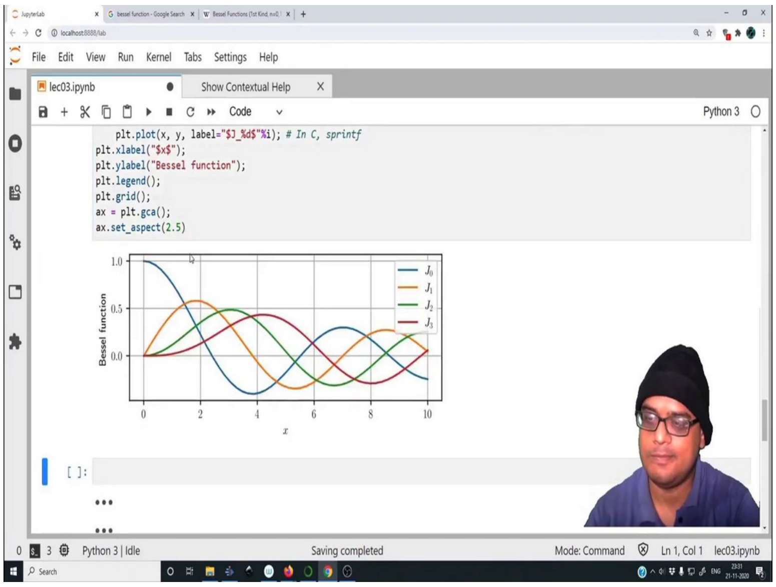
(Refer Slide Time: 33:48)



```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
plt.grid();
ax = plt.gca();
ax.set_aspect(3)
```

And, this is quite useful to show various things like boundary layers and all that and people use this in their research all the time ok. It is important to show the true or the appropriate scale of the plot ok. So, this is how you can set the aspect ratio. It can be a fraction as well. It can be 2.5 as well.

(Refer Slide Time: 34:07)

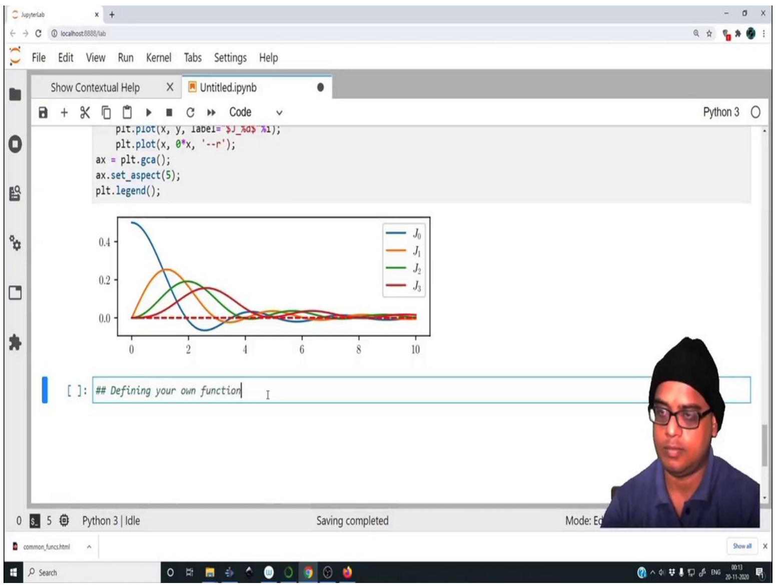


```
    plt.plot(x, y, label="$J_%d$"%i); # In C, sprintf
plt.xlabel("$x$");
plt.ylabel("Bessel function");
plt.legend();
plt.grid();
ax = plt.gca();
ax.set_aspect(2.5)
```

```
[ ]:
```

So, this is how you can set the aspect ratio.

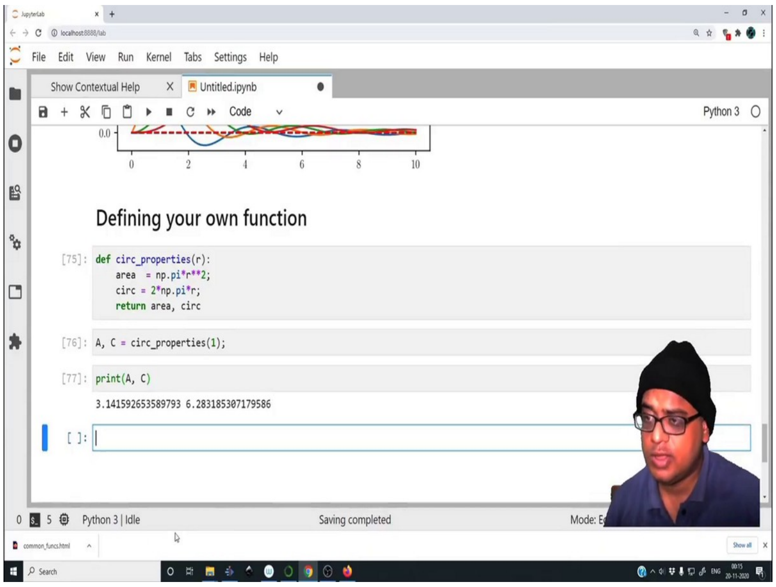
(Refer Slide Time: 34:20)



```
    plt.plot(x, y, label="$J_%d$"%i);
    plt.plot(x, 0*x, '--r');
ax = plt.gca();
ax.set_aspect(5);
plt.legend();
```

```
[ ]: ## Defining your own function
```

(Refer Slide Time: 34:27)



Defining your own function

```
[75]: def circ_properties(r):
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[76]: A, C = circ_properties(1);

[77]: print(A, C)

      3.141592653589793 6.283185307179586
```

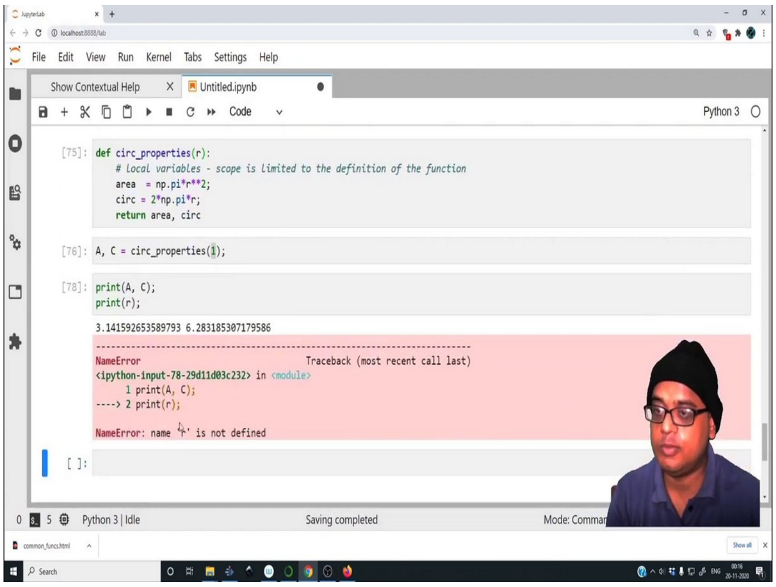
So, what is a function? It takes some input parameters, it does some operations on the input parameters and it gives us an output some meaningful results. You ideally want to abstract whatever is going on inside the function you want to keep it to yourself because suppose you give it to someone else. He just wants to use your function to get some output. He does not really have to rewrite or re look into the program every time ok.

So, a way of encapsulating this idea is to make your own function. So, function is defined in the following manner. So, def, so, then the name of the function. So, suppose the name of the function is circ properties it is a very cliche thing. So, the input to circ properties would be r, then we click on colon and then we press enter. So, now, automatically we are indented.

So, whatever we write inside this block this indented block will be a part of the function. So, let us define area = np.pi*r^2 and circ = 2*np.pi*r and we want to return area and circle we evaluate this cell. So, it does nothing it just stores that definition of the function in the memory.

So, now, suppose I say A, C equal to circ properties. So, let r = 1, when we evaluate the cell let us print A and C. So, A is the area. So, πr^2; so, π times 1^2 and the circumference will be $2\pi r$. So, it checks out. So, we have input one parameter into the function and we have obtained two outputs. So, suppose instead of giving these two as so, we are assigning the return values of the circ properties into two variables ok.

(Refer Slide Time: 37:07)



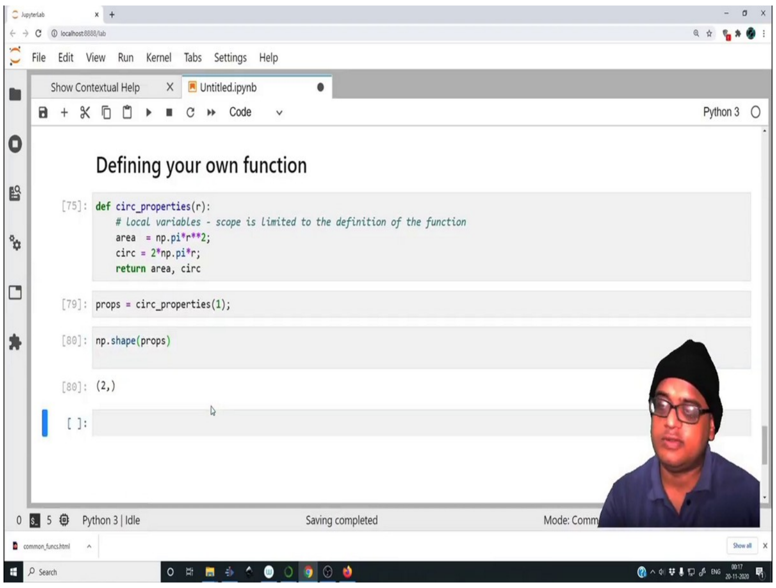
```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[76]: A, C = circ_properties(1);

[78]: print(A, C);
      print(r);

      3.141592653589793 6.283185307179586
      ---------------------------------------------------------------------------
      NameError                                 Traceback (most recent call last)
      <ipython-input-78-29d11d03c232> in <module>
            1 print(A, C);
      ----> 2 print(r);

      NameError: name 'r' is not defined

[ ]:
```

Here whatever is defined inside this block so, these are known as local variables and their scope. So, the variable scope is only limited to the definition of the function; meaning that the moment you make this function call it will take this value 1 it will temporarily store it in a variable called as r. So, there will be no variable called as r otherwise.

So, if we try to print r so, there is no; there is no variable called r. So, all this r is its a local set of variables which is temporarily created to do whatever computation you want inside. Similarly, area circ so, all these are also temporary variables. They do not really exist after the function has been called. So, this is very important. So, the scope of variables inside a function they remain local.

So, you can write a big chunk of logic whatever you want to define inside you define it, whatever computation you want to do you compute it and then you return only the things you want to return to the main file whatever has been created inside is destroyed once the parameters have been passed back to the main file ok.

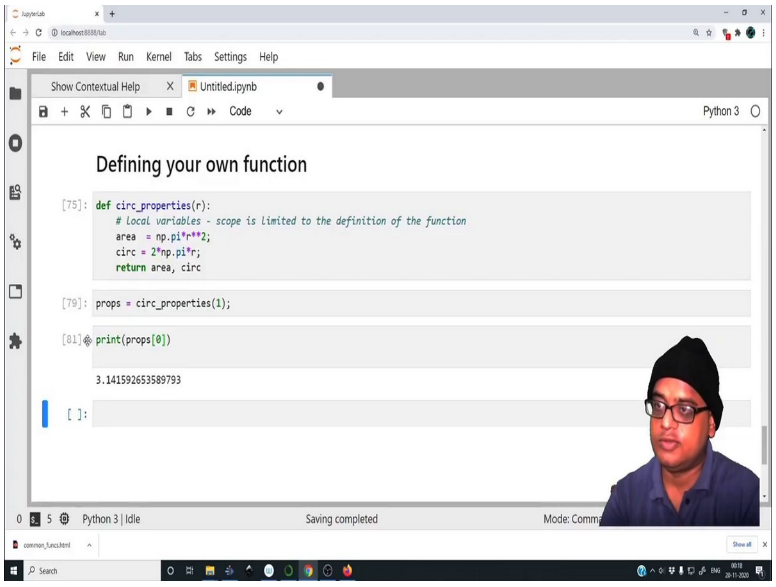
(Refer Slide Time: 38:39)


Defining your own function

```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[79]: props = circ_properties(1);

[80]: np.shape(props)

[80]: (2,)

[ ]:
```

So, A and C get this value similarly we could have assigned everything only to these props suppose. So, now, props is basically so, np.shape (props), so, it is something which contains two elements as shown by the shape.

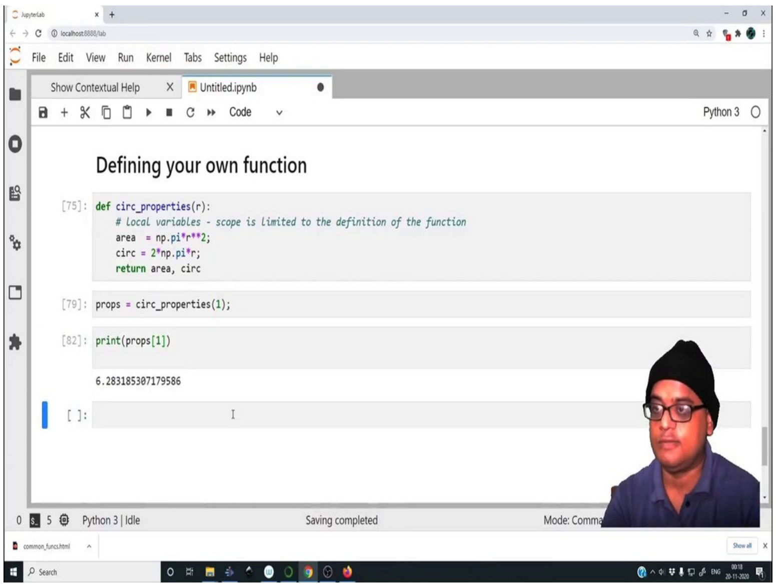
(Refer Slide Time: 38:53)


Defining your own function

```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[79]: props = circ_properties(1);

[81]: print(props[0])

      3.141592653589793

[ ]:
```

So, print props[0]. So, this gives us the first return value that is area.

(Refer Slide Time: 39:05)

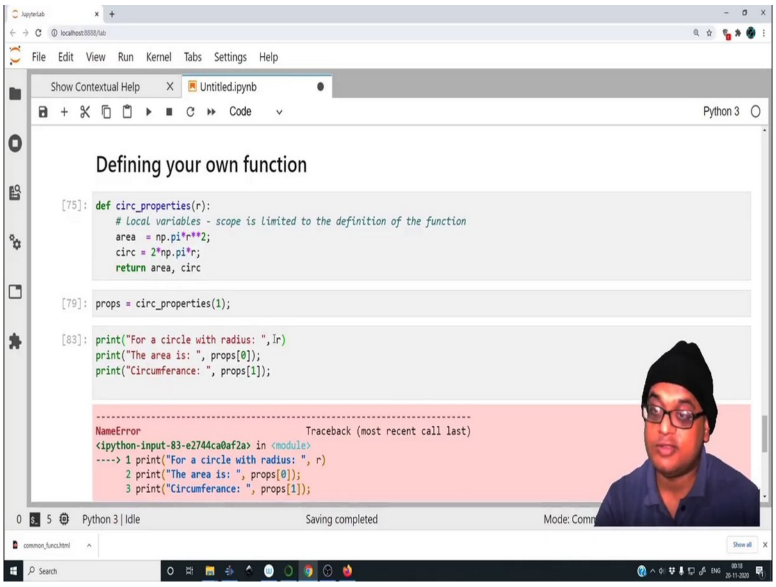


Defining your own function

```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[79]: props = circ_properties(1);

[82]: print(props[1])

      6.283185307179586

[ ]:
```

And, props[1] it gives us the circumference.

(Refer Slide Time: 39:11)

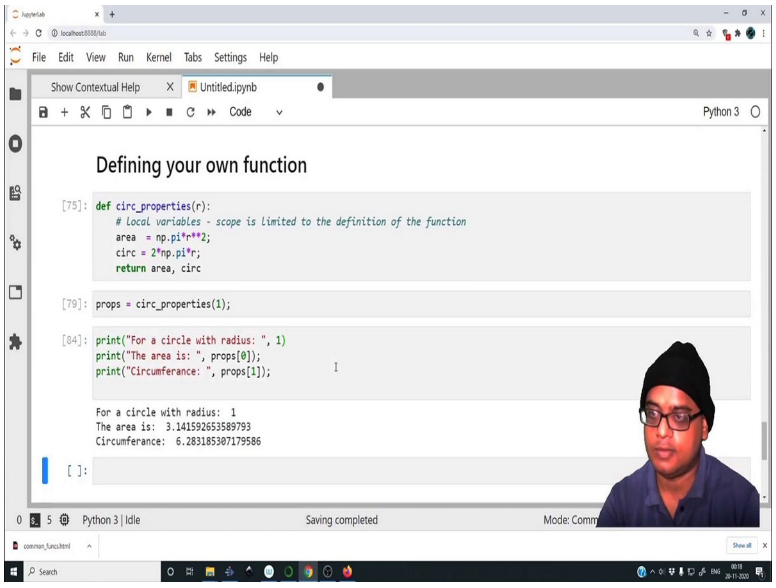


Defining your own function

```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[79]: props = circ_properties(1);

[83]: print("For a circle with radius: ", r)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);

      ---------------------------------------------------------------------------
      NameError                                 Traceback (most recent call last)
      <ipython-input-83-e2744ca0af2a> in <module>
      ----> 1 print("For a circle with radius: ", r)
            2 print("The area is: ", props[0]);
            3 print("Circumferance: ", props[1]);
```

So, I can prettify this. So, I can say the area is.

(Refer Slide Time: 39:54)

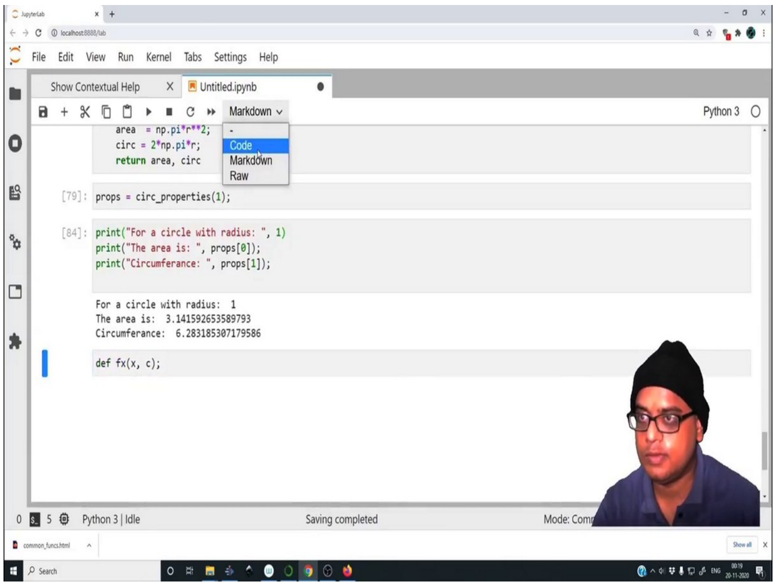


Defining your own function

```
[75]: def circ_properties(r):
          # local variables - scope is limited to the definition of the function
          area  = np.pi*r**2;
          circ = 2*np.pi*r;
          return area, circ

[79]: props = circ_properties(1);

[84]: print("For a circle with radius: ", 1)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);

      For a circle with radius:  1
      The area is:  3.141592653589793
      Circumferance:  6.283185307179586
```

So, r is undefined, so obviously, we have to put 1 over here ok. So, this is how you can prettify the output. So, this is how we can create a function. So, let us create a function which will make a polynomial for us.

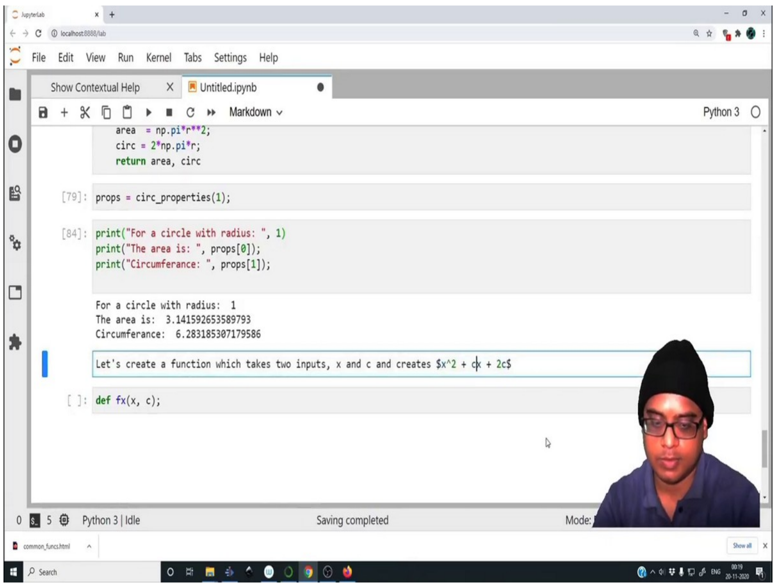
(Refer Slide Time: 40:09)



```
      area  = np.pi*r**2;
      circ = 2*np.pi*r;
      return area, circ

[79]: props = circ_properties(1);

[84]: print("For a circle with radius: ", 1)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);

      For a circle with radius:  1
      The area is:  3.141592653589793
      Circumferance:  6.283185307179586

      def fx(x, c);
```

So, def fx(x, c), right it takes input as x and c.

(Refer Slide Time: 40:26)



```
    area  = np.pi*r**2;
    circ = 2*np.pi*r;
    return area, circ
```

```
[79]: props = circ_properties(1);
```

```
[84]: print("For a circle with radius: ", 1)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);
```

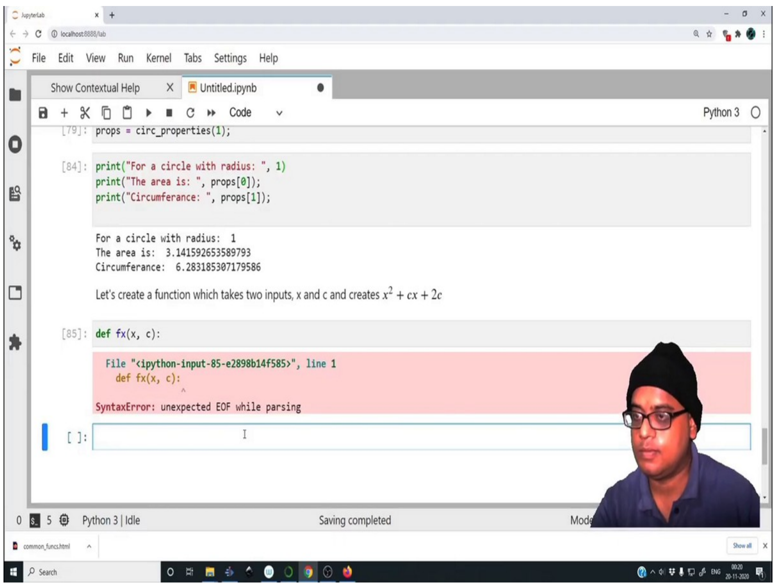
```
For a circle with radius:  1
The area is:  3.141592653589793
Circumferance:  6.283185307179586
```

```
Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$
```

```
[ ]: def fx(x, c);
```

So, let me create; let me create a cell.

(Refer Slide Time: 40:51)



```
[79]: props = circ_properties(1);
```

```
[84]: print("For a circle with radius: ", 1)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);
```

```
For a circle with radius:  1
The area is:  3.141592653589793
Circumferance:  6.283185307179586
```

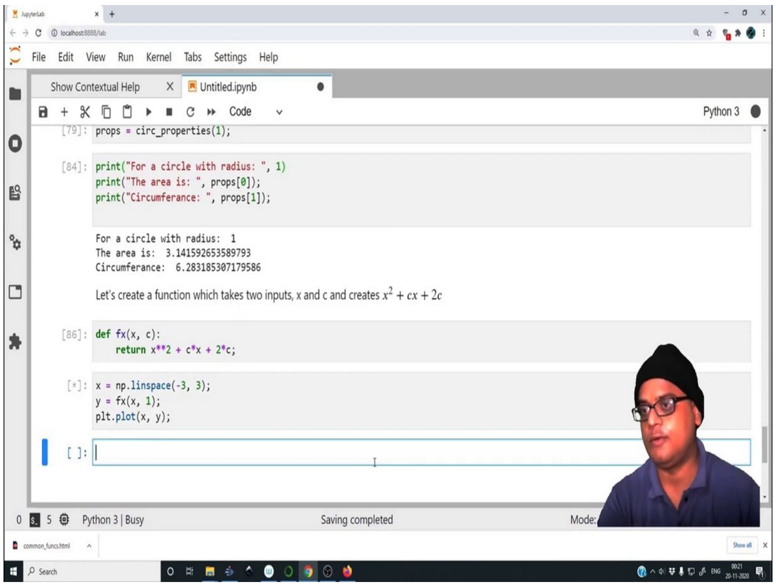
Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[85]: def fx(x, c):
```

```
  File "<ipython-input-85-e2898b14f585>", line 1
    def fx(x, c):
                 ^
SyntaxError: unexpected EOF while parsing
```

So, let us make this particular function. So, def fx(x, c). So, colon.

(Refer Slide Time: 41:07)



```
[79]: props = circ_properties(1);

[84]: print("For a circle with radius: ", 1)
      print("The area is: ", props[0]);
      print("Circumferance: ", props[1]);

      For a circle with radius:  1
      The area is:  3.141592653589793
      Circumferance:  6.283185307179586
```

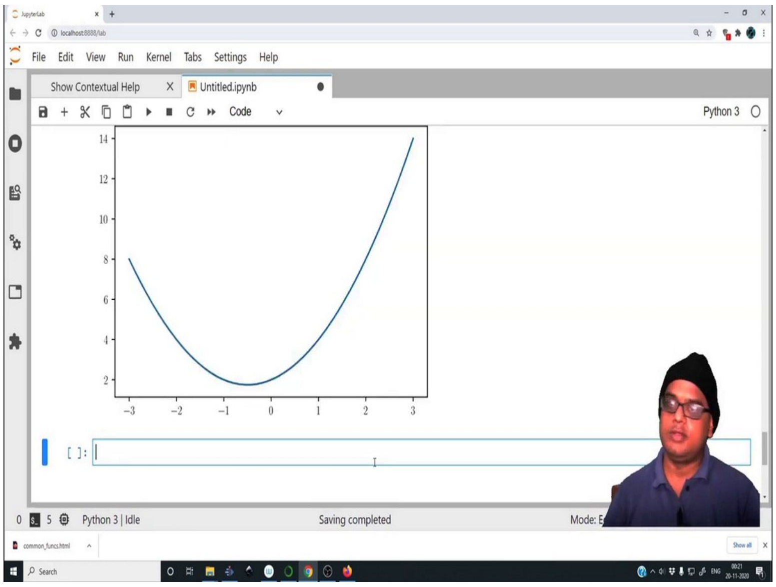
Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[86]: def fx(x, c):
          return x**2 + c*x + 2*c;

[*]: x = np.linspace(-3, 3);
     y = fx(x, 1);
     plt.plot(x, y);

[ ]:
```

We will return x^2 + c*x + 2*c. So, we evaluate the cell. So, now, let us. So, in the previous example we have seen that once we can pass a single scalar to the function and we can obtain two scalars. But, again the function is broadcastable to all the elements of the array x. So, once when we define x = np.linspace(-3, 3), we can define y = fx (x,1).

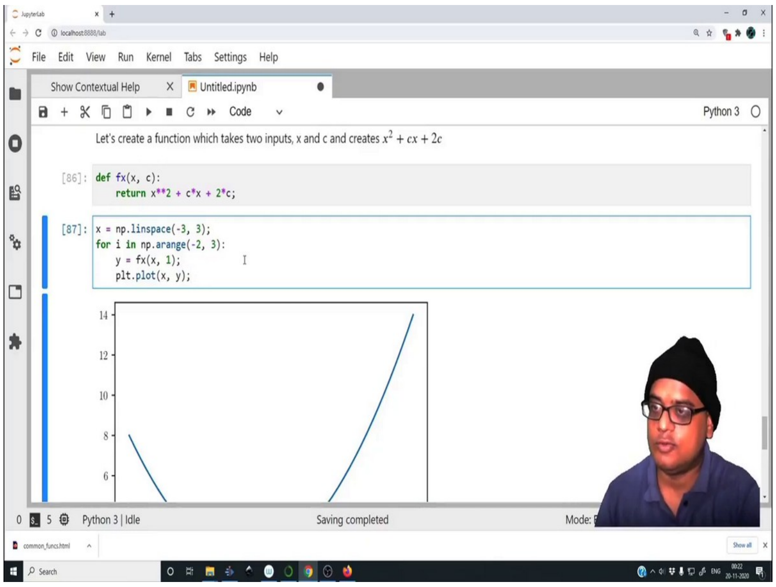
So, it will take each element of x and evaluate the function f x for all those elements of x essentially y will take each values of x and evaluate x^2 + c*x + c ok. So, in order to see whether that is true or not we will do plt.plot (x,y). So, let us see what we obtain and there you have it.

(Refer Slide Time: 42:19)



So, this is the curve of $x^2 + x + 2$ so. In fact, let us plot a family of curves parameterized by different c values.

(Refer Slide Time: 42:44)

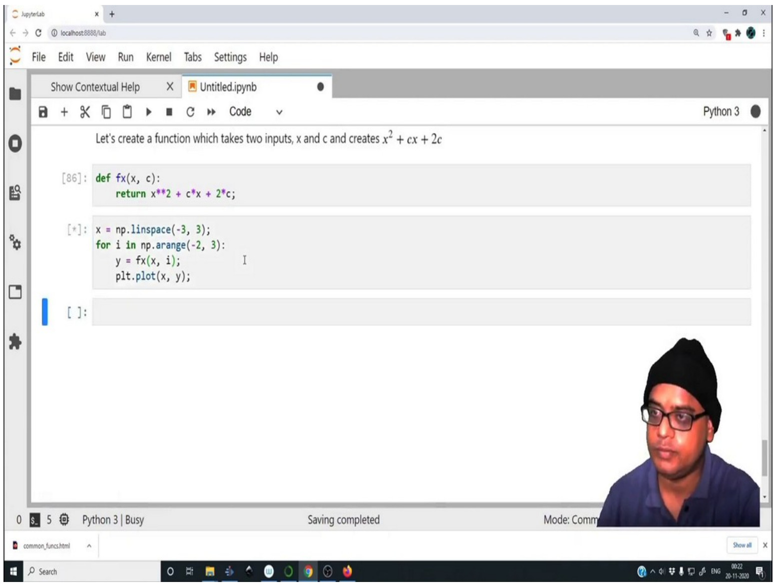


Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[86]: def fx(x, c):
          return x**2 + c*x + 2*c;

[87]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y = fx(x, 1);
          plt.plot(x, y);
```

So, how to make a loop out of this. So, we go here for i in np.arange(-2, 3) we indent this and we indent this by indenting these two lines we are essentially pushing those two lines inside the for loop. If they are not indented respect to the for loop they will not be executed in the for loop. It is unlike C programming where things inside the curly braces are a part of the loop ok, in this the syntax is not like that.

(Refer Slide Time: 43:21)

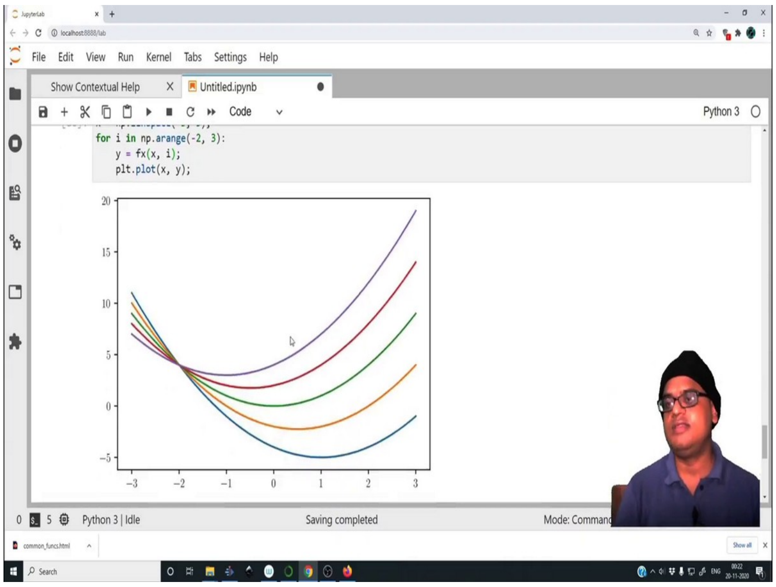


Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[86]: def fx(x, c):
          return x**2 + c*x + 2*c;

[*]:  x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y = fx(x, i);
          plt.plot(x, y);

[ ]:
```

So, instead of x, 1, we will pass x, i.

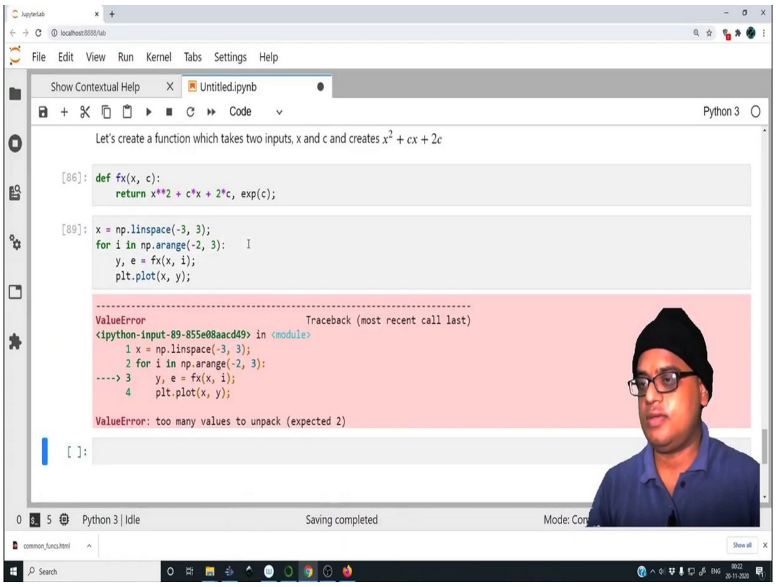
(Refer Slide Time: 43:24)



```
for i in np.arange(-2, 3):
    y = fx(x, i);
    plt.plot(x, y);
```

So, let us evaluate the cell and these are the family of curves ok. So, these are all the family of curves obtained for different values of C. So, this is how you can make your own functions; feel free to explore this. You can pass arrays, you can get return as arrays you can in fact, you can return an array and the scalar as well.

(Refer Slide Time: 43:48)

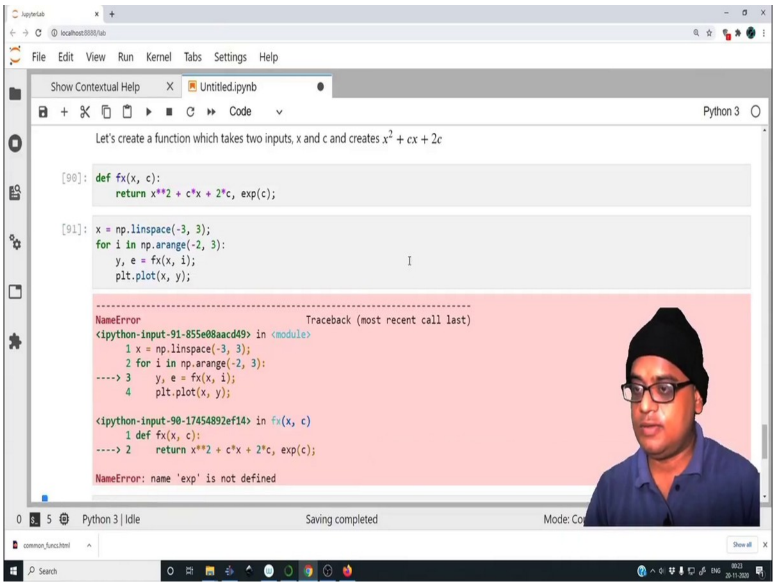


Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[86]: def fx(x, c):
          return x**2 + c*x + 2*c, exp(c);

[89]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y, e = fx(x, i);
          plt.plot(x, y);

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-89-855e08aacd49> in <module>
      1 x = np.linspace(-3, 3);
      2 for i in np.arange(-2, 3):
----> 3     y, e = fx(x, i);
      4     plt.plot(x, y);

ValueError: too many values to unpack (expected 2)

[ ]:
```

So, if I return exp(c) so, then y, e.

(Refer Slide Time: 43:59)



Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

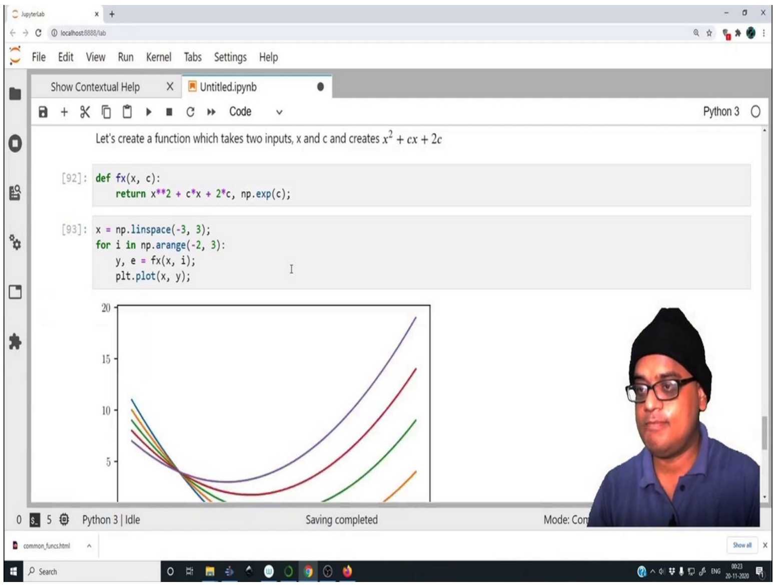
```
[90]: def fx(x, c):
          return x**2 + c*x + 2*c, exp(c);
```

```
[91]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y, e = fx(x, i);
          plt.plot(x, y);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-91-855e08aacd49> in <module>
      1 x = np.linspace(-3, 3);
      2 for i in np.arange(-2, 3):
----> 3     y, e = fx(x, i);
      4     plt.plot(x, y);

<ipython-input-90-17454892ef14> in fx(x, c)
      1 def fx(x, c):
----> 2     return x**2 + c*x + 2*c, exp(c);

NameError: name 'exp' is not defined
```

So, I have to evaluate this evaluate this exponential is not defined because it is np.exp.

(Refer Slide Time: 44:03)

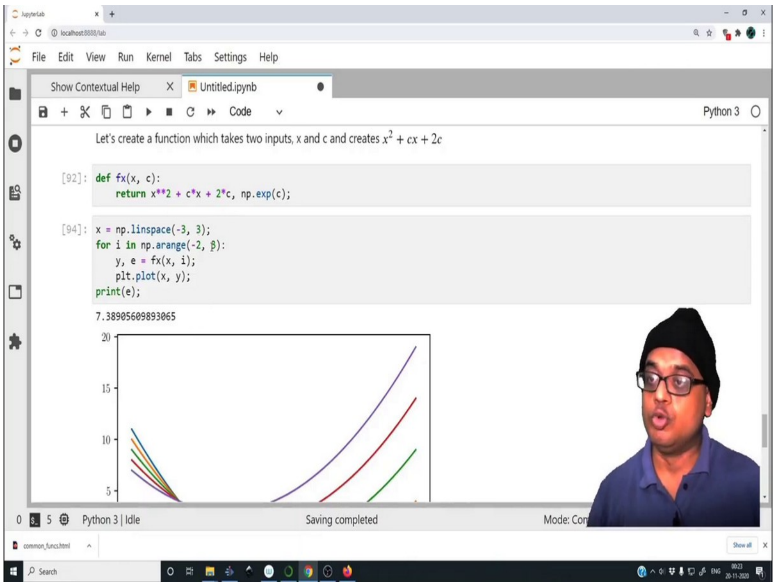


Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

```
[92]: def fx(x, c):
          return x**2 + c*x + 2*c, np.exp(c);
```

```
[93]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y, e = fx(x, i);
          plt.plot(x, y);
```

(Refer Slide Time: 44:09)



Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

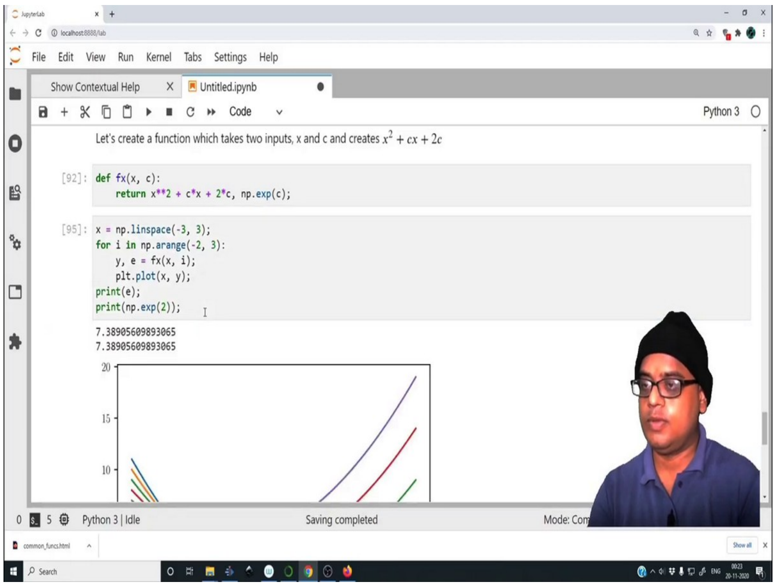
```
[92]: def fx(x, c):
          return x**2 + c*x + 2*c, np.exp(c);
```

```
[94]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y, e = fx(x, i);
          plt.plot(x, y);
      print(e);
```

```
7.38905609893065
```

So, now, if I print the value of e it will give me exponential of what will be the value? exp (2) because the last value of I would have been 2. So, it would have printed exponential of 2.

(Refer Slide Time: 44:26)



Let's create a function which takes two inputs, x and c and creates $x^2 + cx + 2c$

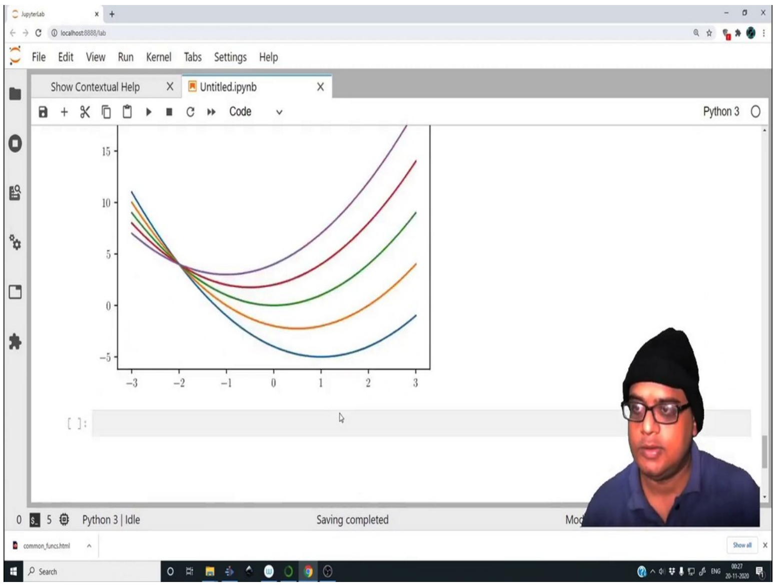
```
[92]: def fx(x, c):
          return x**2 + c*x + 2*c, np.exp(c);
```

```
[95]: x = np.linspace(-3, 3);
      for i in np.arange(-2, 3):
          y, e = fx(x, i);
          plt.plot(x, y);
      print(e);
      print(np.exp(2));
```

```
7.38905609893065
7.38905609893065
```

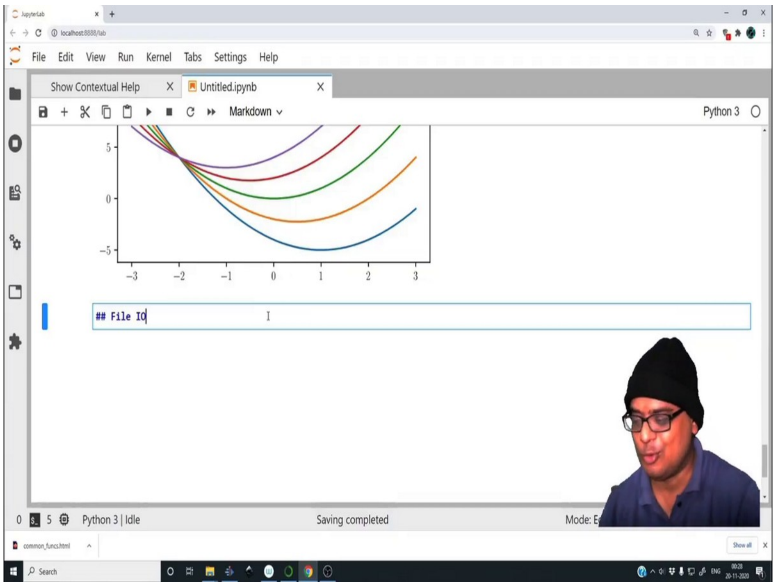
Just to verify that there you go. So, we can. So, essentially we have passed in we have passed into the function an array and a scalar and we are obtaining as an output an array and a scalar. So, you can mix the type of variables that you can given to the function get out of the function. There is no boundaries to this you can apply as much logic as you want ok. But, remember try to keep things simple. Do not try to overcomplicate functions. In the end you will yourself have troubles reading your own code.

(Refer Slide Time: 45:10)



So, now we come to the very last of this particular lecture of course. The most important thing that you will be using in your research work or whatever is to perform file input output ok.

(Refer Slide Time: 45:28)

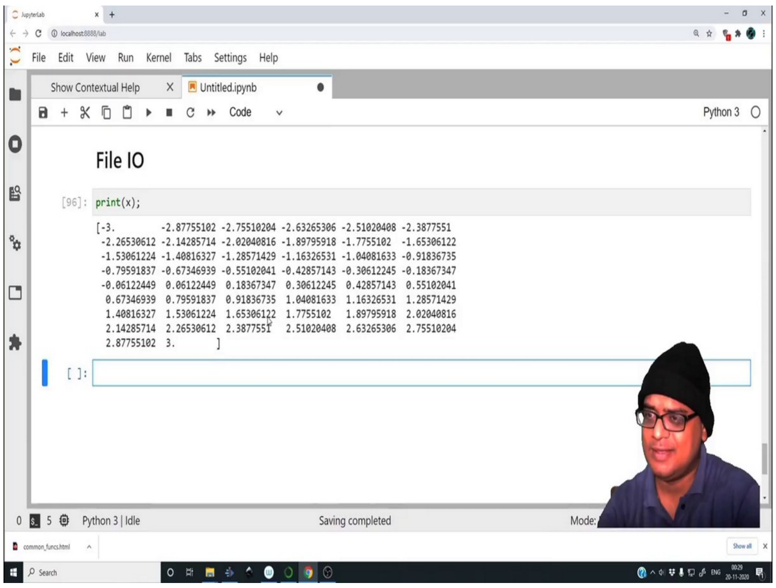


```
## File IO
```

So, file IO so, file IO is means outputting some data to a file or reading some data from a file and this is incredibly useful if you are doing experiments where you have a bunch of data from some probe or some apparatus and then you want to plot that data or post process that data.

Or this is also useful when you are using some other programming language like C. So, using C to generate a bunch of textual data and then you want to quickly plot it so, Python provides you the plotting libraries to load. So, not only to load the file, but to plot it as well. It is not so trivial to plot something in C. So, C is more low label you see.

(Refer Slide Time: 46:16)



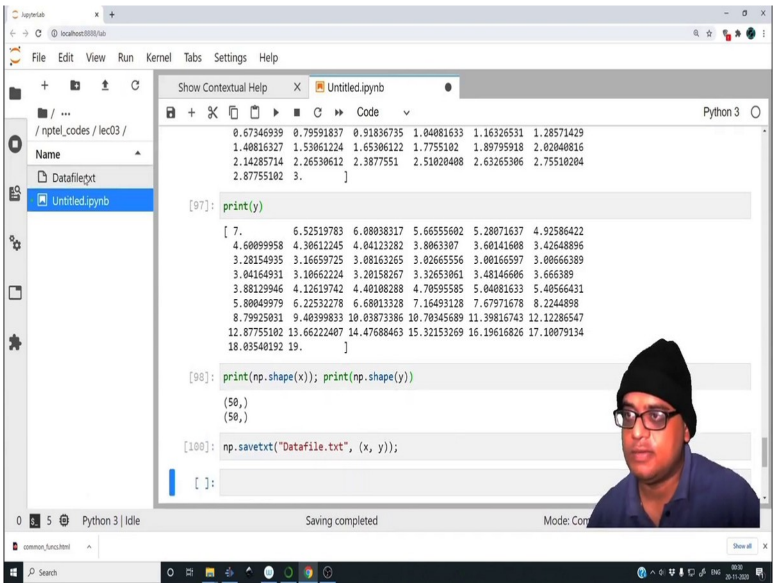
File IO

```
[96]: print(x);
```

```
[-3.         -2.87755102 -2.75510204 -2.63265306 -2.51020408 -2.3877551
 -2.26530612 -2.14285714 -2.02040816 -1.89795918 -1.7755102  -1.65306122
 -1.53061224 -1.40816327 -1.28571429 -1.16326531 -1.04081633 -0.91836735
 -0.79591837 -0.67346939 -0.55102041 -0.42857143 -0.30612245 -0.18367347
 -0.06122449  0.06122449  0.18367347  0.30612245  0.42857143  0.55102041
  0.67346939  0.79591837  0.91836735  1.04081633  1.16326531  1.28571429
  1.40816327  1.53061224  1.65306122  1.7755102   1.89795918  2.02040816
  2.14285714  2.26530612  2.3877551   2.51020408  2.63265306  2.75510204
  2.87755102  3.        ]
```

So, file IO: so, let us quickly see what arrays we have. So, print x so, this is the x array and let us print y.

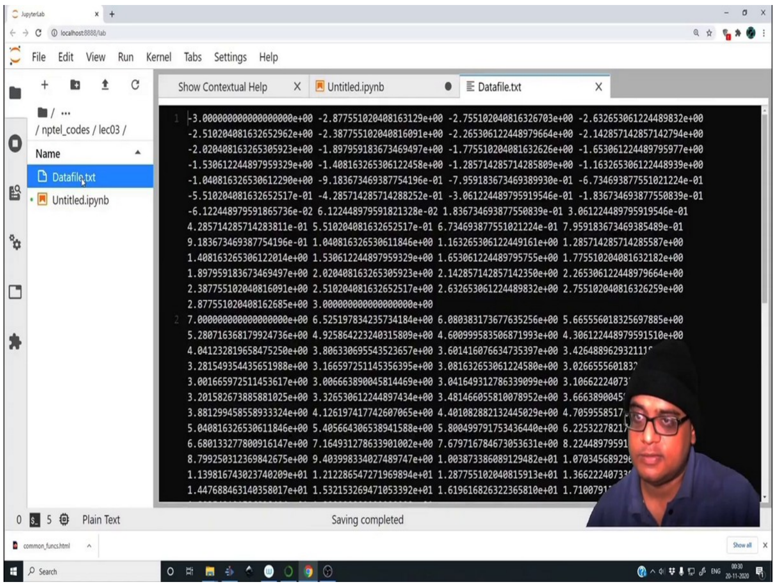
(Refer Slide Time: 46:24)



```
 0.67346939  0.79591837  0.91836735  1.04081633  1.16326531  1.28571429
 1.40816327  1.53061224  1.65306122  1.7755102   1.89795918  2.02040816
 2.14285714  2.26530612  2.3877551   2.51020408  2.63265306  2.75510204
 2.87755102  3.        ]

[97]: print(y)

[ 7.          6.52519783  6.08038317  5.66555602  5.28071637  4.92586422
  4.60099958  4.30612245  4.04123282  3.8063307   3.60141608  3.42648896
  3.28154935  3.16659725  3.08163265  3.02665556  3.00166597  3.00666389
  3.04164931  3.10662224  3.20158267  3.32653061  3.48146606  3.666389
  3.88129946  4.12619742  4.40108288  4.70595585  5.04081633  5.40566431
  5.80049979  6.22532278  6.68013328  7.16493128  7.67971678  8.2244898
  8.79925031  9.40399833 10.03873386 10.70345689 11.39816743 12.12286547
 12.87755102 13.66222407 14.47688463 15.32153269 16.19616826 17.10079134
 18.03540192 19.        ]

[98]: print(np.shape(x)); print(np.shape(y))

(50,)
(50,)

[100]: np.savetxt("Datafile.txt", (x, y));

[ ]:
```

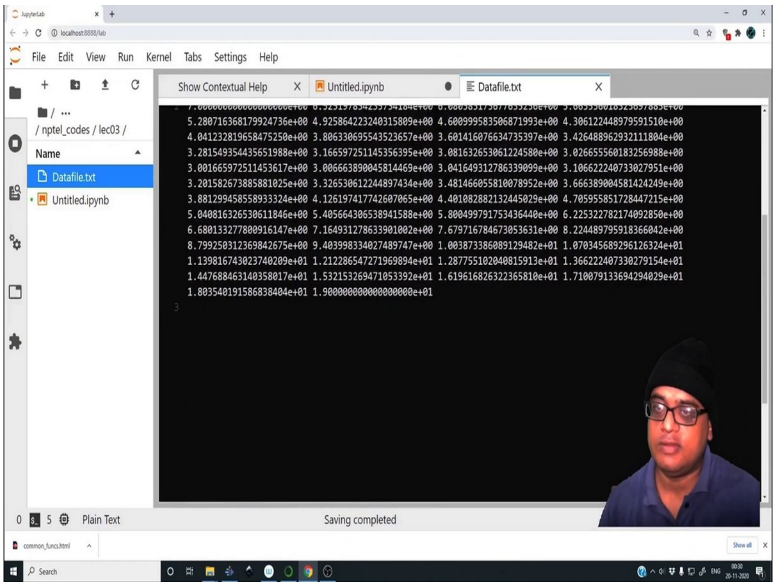
So, this is the y array. So, let us confirm whether the shapes of x and y are the same. So, print(np.shape(x)), print(np.shape(y)). So, they are the same. So, now suppose I want to output to a file these two arrays. So, what I will do is np.savetxt. So, the name of the file will be Datafile.txt. Inside this I will output x and y. So, after running this let us go. So, its created this data file inside the folder that we are working in.

(Refer Slide Time: 47:32)



```
-3.000000000000000000e+00 -2.877551020408163129e+00 -2.755102040816326703e+00 -2.632653061224489832e+00
-2.510204081632652962e+00 -2.387755102040816091e+00 -2.265306122448979664e+00 -2.142857142857142794e+00
-2.020408163265305923e+00 -1.897959183673469497e+00 -1.775510204081632626e+00 -1.653061224489795977e+00
-1.530612244897959329e+00 -1.408163265306122458e+00 -1.285714285714285809e+00 -1.163265306122448939e+00
-1.040816326530612290e+00 -9.183673469387754196e-01 -7.959183673469389930e-01 -6.734693877551021224e-01
-5.510204081632652517e-01 -4.285714285714288252e-01 -3.061224489795919546e-01 -1.836734693877550839e-01
-6.122448979591865736e-02 6.122448979591821328e-02 1.836734693877550839e-01 3.061224489795919546e-01
4.285714285714283811e-01 5.510204081632652517e-01 6.734693877551021224e-01 7.959183673469385489e-01
9.183673469387754196e-01 1.040816326530611846e+00 1.163265306122449161e+00 1.285714285714285587e+00
1.408163265306122014e+00 1.530612244897959329e+00 1.653061224489795755e+00 1.775510204081632182e+00
1.897959183673469497e+00 2.020408163265305923e+00 2.142857142857142350e+00 2.265306122448979664e+00
2.387755102040816091e+00 2.510204081632652517e+00 2.632653061224489832e+00 2.755102040816326259e+00
2.877551020408162685e+00 3.000000000000000000e+00
7.000000000000000000e+00 6.525197834235734184e+00 6.080383173677635256e+00 5.665556018325697885e+00
5.280716368179924736e+00 4.925864223240315809e+00 4.600999583506871993e+00 4.306122448979591510e+00
4.041232819658475250e+00 3.806330695543523657e+00 3.601416076634735397e+00 3.426488962932111
3.281549354435651988e+00 3.166597251145356395e+00 3.081632653061224580e+00 3.02665556018327
3.001665972511453617e+00 3.006663890045814469e+00 3.041649312786339099e+00 3.10662224073
3.201582673885881025e+00 3.326530612244897434e+00 3.481466055810078952e+00 3.66638900045
3.881299458558933324e+00 4.126197417742607065e+00 4.401082882132445029e+00 4.7059558517
5.040816326530611846e+00 5.405664306538941588e+00 5.800499791753436440e+00 6.22532278217
6.680133277800916147e+00 7.164931278633901002e+00 7.679716784673053631e+00 8.22448979591
8.799250312369842675e+00 9.403998334027489747e+00 1.003873386089129482e+01 1.07034568929
1.139816743023740209e+01 1.212286547271969894e+01 1.287755102040815913e+01 1.36622240733
1.447688463140358017e+01 1.532153269471053392e+01 1.619616826322365810e+01 1.7100791
```

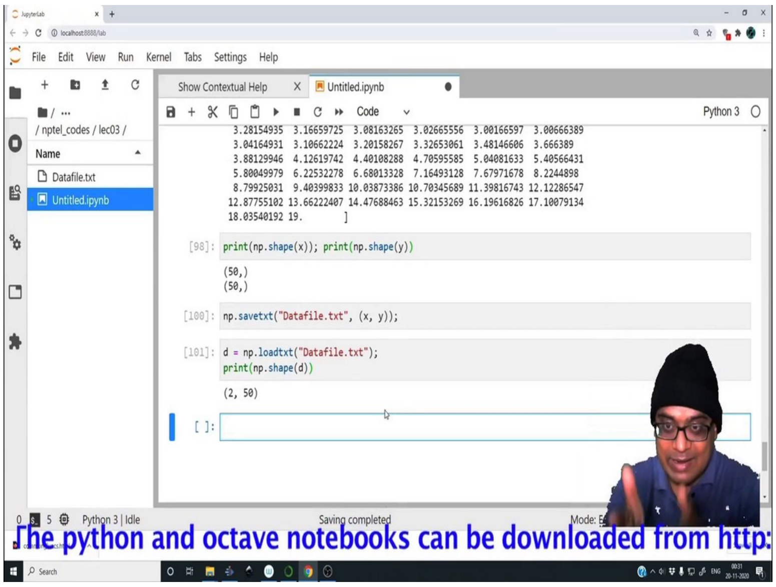
(Refer Slide Time: 47:40)



```
5.280716368179924736e+00 4.925864223240315809e+00 4.600999583506871993e+00 4.306122448979591510e+00
4.041232819658475250e+00 3.806330695543523657e+00 3.601416076634735397e+00 3.426488962932111804e+00
3.281549354435651988e+00 3.166597251145356395e+00 3.081632653061224580e+00 3.026655560183256988e+00
3.001665972511453617e+00 3.006663890045814469e+00 3.041649312786339099e+00 3.106622240733027951e+00
3.201582673885881025e+00 3.326530612244897434e+00 3.481466055810078952e+00 3.666389004581424249e+00
3.881299458558933324e+00 4.126197417742607065e+00 4.401082882132445029e+00 4.705955851728447215e+00
5.040816326530611846e+00 5.405664306538941588e+00 5.800499791753436440e+00 6.225322782174092850e+00
6.680133277800916147e+00 7.164931278633901002e+00 7.679716784673053631e+00 8.224489795918366042e+00
8.799250312369842675e+00 9.403998334027489747e+00 1.003873386089129482e+01 1.070345689296126324e+01
1.139816743023740209e+01 1.212286547271969894e+01 1.287755102040815913e+01 1.366222407330279154e+01
1.447688463140358017e+01 1.532153269471053392e+01 1.619616826322365810e+01 1.710079133694294029e+01
1.803540191586838404e+01 1.900000000000000000e+01
```

So, let us open the data file. So, its the first row contains all the x data while the second row contains all the y data. So, its essentially. So, let us load it.

(Refer Slide Time: 47:47)



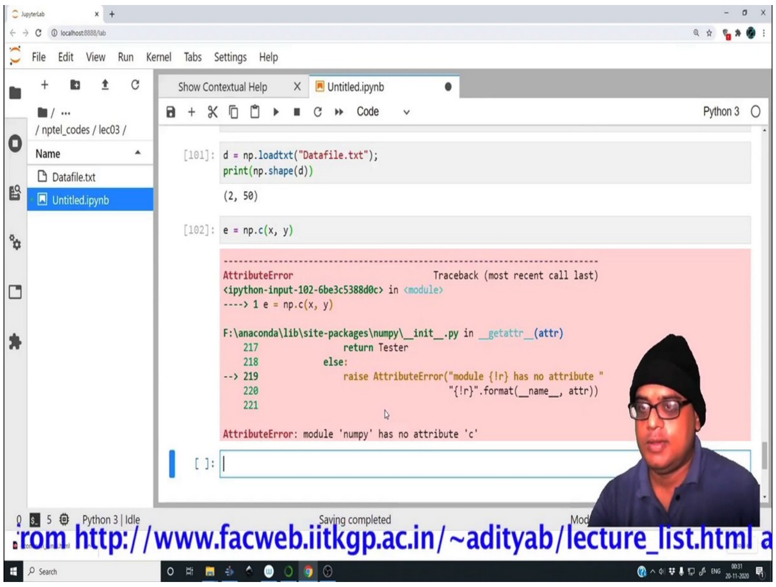
```
 3.28154935  3.16659725  3.08163265  3.02665556  3.00166597  3.00666389
 3.04164931  3.10662224  3.20158267  3.32653061  3.48146606  3.666389
 3.88129946  4.12619742  4.40108288  4.70595585  5.04081633  5.40566431
 5.80049979  6.22532278  6.68013328  7.16493128  7.67971678  8.2244898
 8.79925031  9.40399833 10.03873386 10.70345689 11.39816743 12.12286547
12.87755102 13.66222407 14.47688463 15.32153269 16.19616826 17.10079134
18.03540192 19.        ]

[98]: print(np.shape(x)); print(np.shape(y))

(50,)
(50,)

[100]: np.savetxt("Datafile.txt", (x, y));

[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

(2, 50)
```

The python and octave notebooks can be downloaded from http:

So, let us do d = np.loadtxt("Datafile.txt"). So, once we have loaded this let us find out the shape what d contains. So, basically we had one row containing x, one row containing y. So, now essentially d should contain 2 rows and 50 elements in each row. So, let us see. So, let us print(np.shape(d)). So, again it is 2 rows and 50 elements in each row, but that is not how we are accustomed to printing data to a file or reading.

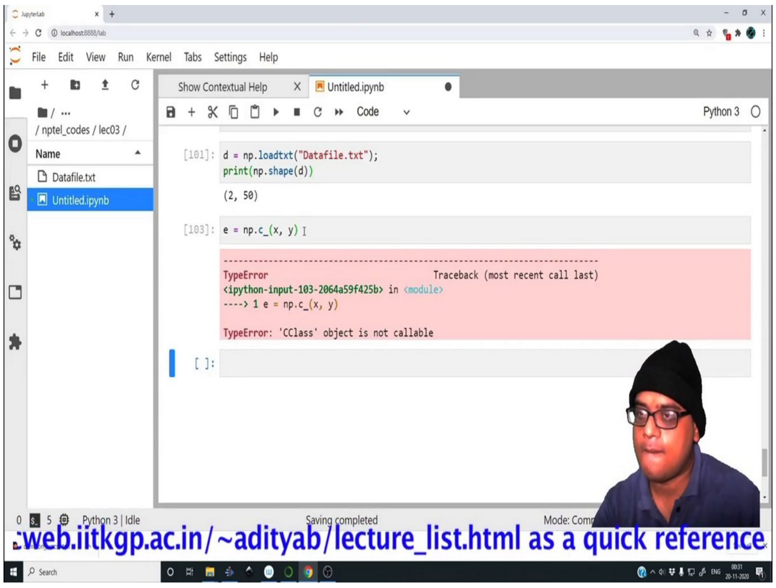
We are more accustomed to x data like this and y data like this rather than having x data like this y data I mean it does not make a difference, but still if you want to print it out and show it to someone you would rather have it in a column format right. So, the way to do it is to concatenate the two arrays before dumping them to the text file. So, let us quickly see what concatenating means. So, np.c_[x, y].

(Refer Slide Time: 48:33)



```
[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

       (2, 50)

[102]: e = np.c(x, y)

       ---------------------------------------------------------------------------
       AttributeError                            Traceback (most recent call last)
       <ipython-input-102-6be3c5388d0c> in <module>
       ----> 1 e = np.c(x, y)

       F:\anaconda\lib\site-packages\numpy\__init__.py in __getattr__(attr)
           217                 return Tester
           218             else:
       --> 219                 raise AttributeError("module {!r} has no attribute "
           220                                      "{!r}".format(__name__, attr))
           221

       AttributeError: module 'numpy' has no attribute 'c'

[ ]:
```

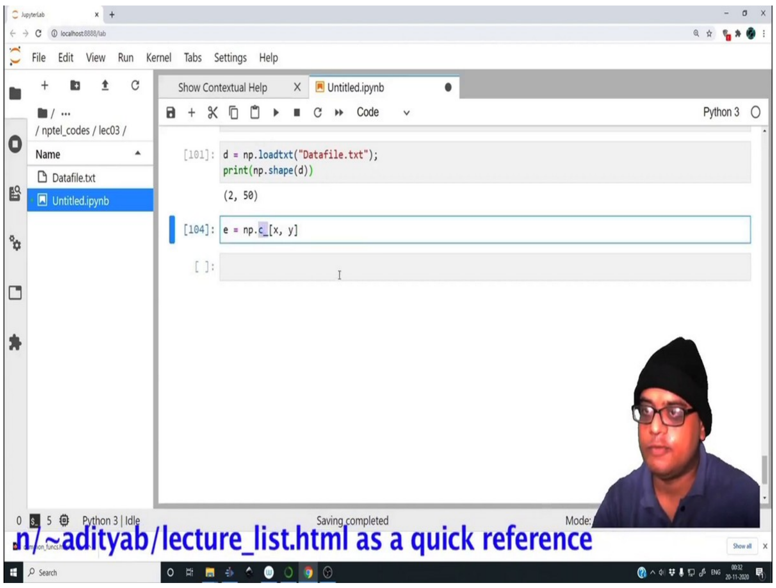
(Refer Slide Time: 49:02)



```
[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

       (2, 50)

[103]: e = np.c_(x, y)

       ---------------------------------------------------------------------------
       TypeError                                 Traceback (most recent call last)
       <ipython-input-103-2064a59f425b> in <module>
       ----> 1 e = np.c_(x, y)

       TypeError: 'CClass' object is not callable

[ ]:
```

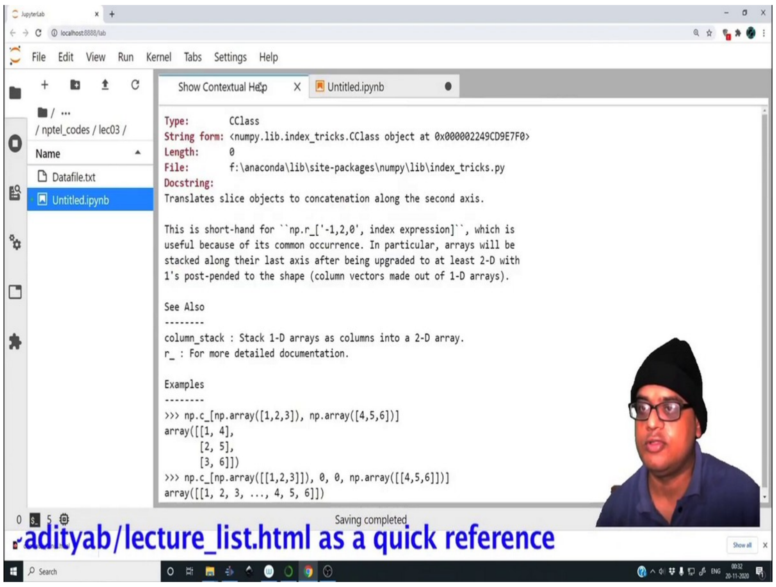
So, the function is c underscore. This should be within brackets.

(Refer Slide Time: 49:11)



```
[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

       (2, 50)

[104]: e = np.c_[x, y]

[ ]:
```

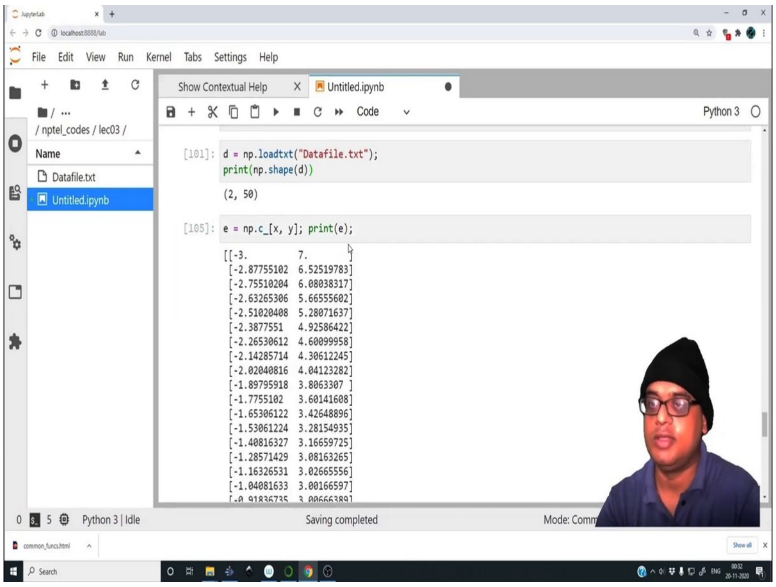
(Refer Slide Time: 49:16)



```
Type:        CClass
String form: <numpy.lib.index_tricks.CClass object at 0x000002249CD9E7F0>
Length:      0
File:        f:\anaconda\lib\site-packages\numpy\lib\index_tricks.py
Docstring:
Translates slice objects to concatenation along the second axis.

This is short-hand for ``np.r_['-1,2,0', index expression]``, which is
useful because of its common occurrence. In particular, arrays will be
stacked along their last axis after being upgraded to at least 2-D with
1's post-pended to the shape (column vectors made out of 1-D arrays).

See Also
--------
column_stack : Stack 1-D arrays as columns into a 2-D array.
r_ : For more detailed documentation.

Examples
--------
>>> np.c_[np.array([1,2,3]), np.array([4,5,6])]
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])]
array([[1, 2, 3, ..., 4, 5, 6]])
```

So, np.c; so, if I double click on this and go to contextual help. So, it translates to slicing objects along a given axis ok. So, r and c are the two so, c is column stacking and r is row stacking. So, we are using column stacking over here and we are passing into the square bracket the two variables that you want to stack.

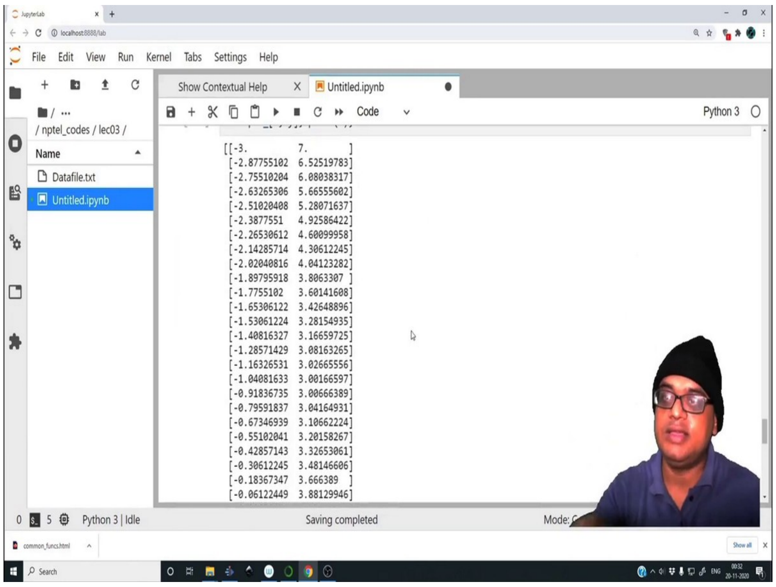
(Refer Slide Time: 49:41)



```
[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

       (2, 50)

[105]: e = np.c_[x, y]; print(e);

       [[-3.          7.        ]
        [-2.87755102  6.52519783]
        [-2.75510204  6.08038317]
        [-2.63265306  5.66555602]
        [-2.51020408  5.28071637]
        [-2.3877551   4.92586422]
        [-2.26530612  4.60099958]
        [-2.14285714  4.30612245]
        [-2.02040816  4.04123282]
        [-1.89795918  3.8063307 ]
        [-1.7755102   3.60141608]
        [-1.65306122  3.42648896]
        [-1.53061224  3.28154935]
        [-1.40816327  3.16659725]
        [-1.28571429  3.08163265]
        [-1.16326531  3.02665556]
        [-1.04081633  3.00166597]
        [-0.91836735  3.00666389]
```

So, in fact, let us print e and look at what has happened.

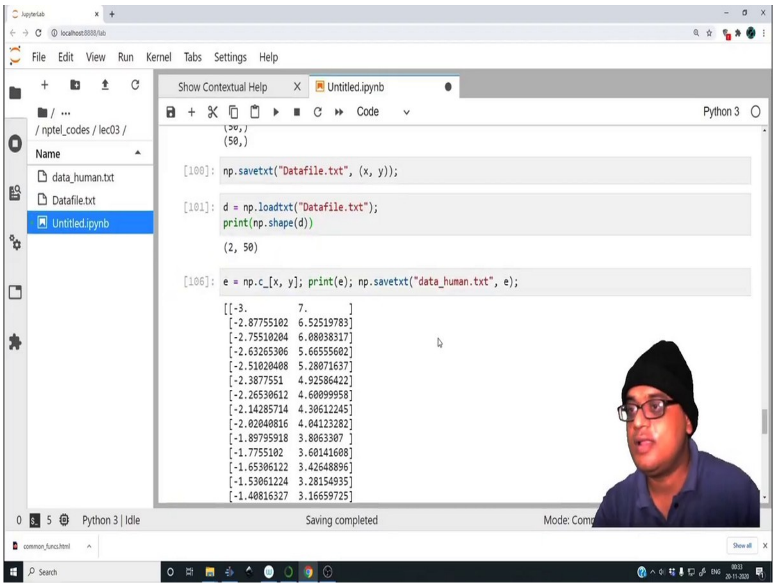
(Refer Slide Time: 49:47)



```
[[-3.          7.        ]
 [-2.87755102  6.52519783]
 [-2.75510204  6.08038317]
 [-2.63265306  5.66555602]
 [-2.51020408  5.28071637]
 [-2.3877551   4.92586422]
 [-2.26530612  4.60099958]
 [-2.14285714  4.30612245]
 [-2.02040816  4.04123282]
 [-1.89795918  3.8063307 ]
 [-1.7755102   3.60141608]
 [-1.65306122  3.42648896]
 [-1.53061224  3.28154935]
 [-1.40816327  3.16659725]
 [-1.28571429  3.08163265]
 [-1.16326531  3.02665556]
 [-1.04081633  3.00166597]
 [-0.91836735  3.00666389]
 [-0.79591837  3.04164931]
 [-0.67346939  3.10662224]
 [-0.55102041  3.20158267]
 [-0.42857143  3.32653061]
 [-0.30612245  3.48146606]
 [-0.18367347  3.666389  ]
 [-0.06122449  3.88129946]
```

So, obviously, e contains all the x elements and all the y elements and in a given row we have x and y ok. So, now, e makes sense. So, now, what we can do in this instead of dumping x and y into the text file we will dump e into the text file.

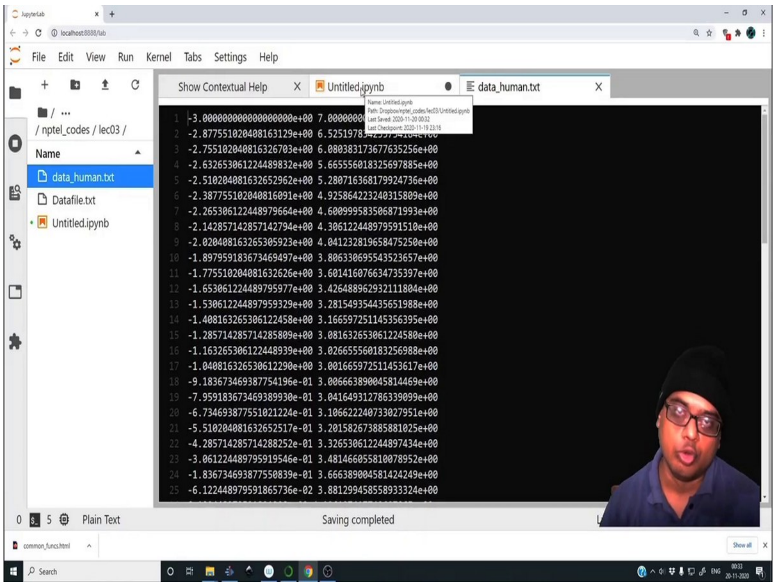
(Refer Slide Time: 50:06)



```
(50,)
(50,)

[100]: np.savetxt("Datafile.txt", (x, y));

[101]: d = np.loadtxt("Datafile.txt");
       print(np.shape(d))

       (2, 50)

[106]: e = np.c_[x, y]; print(e); np.savetxt("data_human.txt", e);

       [[-3.          7.        ]
        [-2.87755102  6.52519783]
        [-2.75510204  6.08038317]
        [-2.63265306  5.66555602]
        [-2.51020408  5.28071637]
        [-2.3877551   4.92586422]
        [-2.26530612  4.60099958]
        [-2.14285714  4.30612245]
        [-2.02040816  4.04123282]
        [-1.89795918  3.8063307 ]
        [-1.7755102   3.60141608]
        [-1.65306122  3.42648896]
        [-1.53061224  3.28154935]
        [-1.40816327  3.16659725]
```

So, we will go over here and we will say np.savetxt. So, data human readable too bad if you are a computer or a robot. So, let me run the cell.

(Refer Slide Time: 50:24)



```
-3.000000000000000000e+00 7.0000000
-2.877551020408163129e+00 6.5251978
-2.755102040816326703e+00 6.080383173677635256e+00
-2.632653061224489832e+00 5.665556018325697885e+00
-2.510204081632652962e+00 5.280716368179924736e+00
-2.387755102040816091e+00 4.925864223240315809e+00
-2.265306122448979664e+00 4.600999583506871993e+00
-2.142857142857142794e+00 4.306122448979591510e+00
-2.020408163265305923e+00 4.041232819658475250e+00
-1.897959183673469497e+00 3.806330695543523657e+00
-1.775510204081632626e+00 3.601416076634735397e+00
-1.653061224489795977e+00 3.426488962932111804e+00
-1.530612244897959329e+00 3.281549354435651988e+00
-1.408163265306122458e+00 3.166597251145356395e+00
-1.285714285714285809e+00 3.081632653061224580e+00
-1.163265306122448939e+00 3.026655560183256988e+00
-1.040816326530612290e+00 3.001665972511453617e+00
-9.183673469387754196e-01 3.006663890045814469e+00
-7.959183673469389930e-01 3.041649312786339099e+00
-6.734693877551021224e-01 3.106622240733027951e+00
-5.510204081632652517e-01 3.201582673885881025e+00
-4.285714285714288252e-01 3.326530612244897434e+00
-3.061224489795919546e-01 3.481466055810078952e+00
-1.836734693877550839e-01 3.666389004581424249e+00
-6.122448979591865736e-02 3.881299458558933324e+00
```

Let us open the file data underscore human and there you go. We have all the x values and all the y values. We do not need to worry about all the rows and all that. So, its just a matter of concatenating everything and of course, I mean you could run a loop over each element and bring them out, but come on that is not the way you get job done in Python or in octave.

In octave or Python you want to make as much as work possible with direct vectors or matrices in use that is how you achieve speed in such kinds of languages. So, with this we end this session on commonly used functions and of course, this is just a very small subset of what is actually existing but I just wanted to showcase some of the things that we will be using in this course.

And, of course, in the due course of this particular set of lectures we will be encountering many more functions and we will discuss about them as and when we require them. So, keep an keep an eye out on all these things. So, with this I end this particular lecture it is goodbye from me.

Have a good day bye.