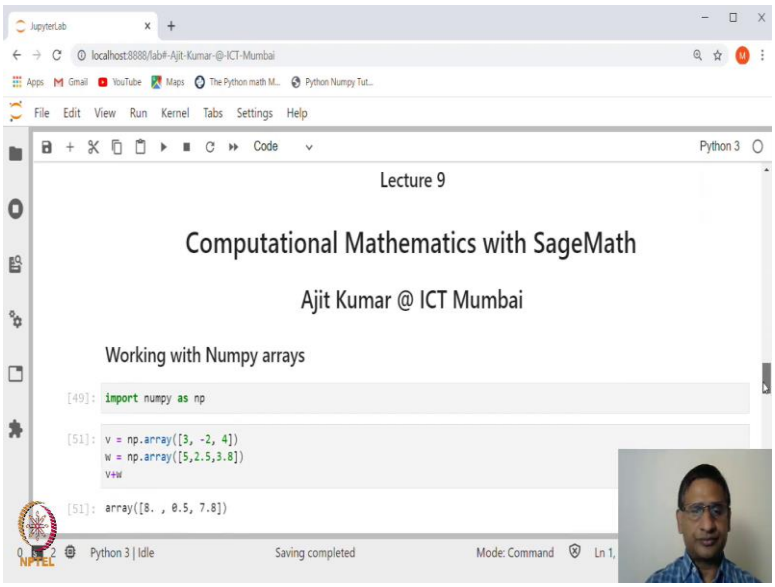
Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 10
Use of NumPy module

So, welcome to the 9th lecture on Computational Mathematics with SageMath. In the previous lecture, we introduced a very important module called NumPy and we saw what, some of the functions inside NumPy. Two important things that we saw, was 'arange' inside NumPy and 'np.array' function, that is, NumPy dot array function inside NumPy library.

And we also saw that some of the functions that, actually the math whatever functions you have in math module, all these functions are available inside NumPy module also. However, these mathematical functions can operate on any list, NumPy array ok, that is what we saw.

(Refer Slide Time: 01:11)



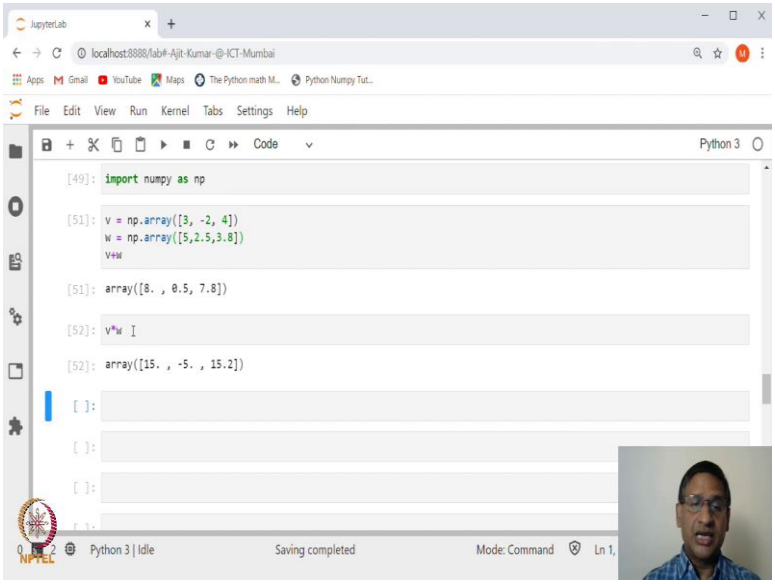
So, now, let us look at some more operations on Numpy array. So, as I said, if you create one-dimensional array, it can be thought of as vector. So, let us look at some of the

operations that we know, that can be applied on vectors. So, let us define a vector v. So, first of all, let me import. So, import, import numpy as np, import.

So, spelling mistake, import numpy as np, and let us define a vector v as np dot array, array, and inside that you give the list. Suppose we want to define a vector in, in three dimensions. So I will say, 3, -2 and then 4, and another vector let us say w, this is np dot array inside that let us this, this the coordinates are 5, 2.1 or 2.5, 3 point, let us say 8. So, these are the two vectors.

Now, if these are the two vectors, you can apply any operations on this vector. So, for example, you can add v plus w, you will get this.

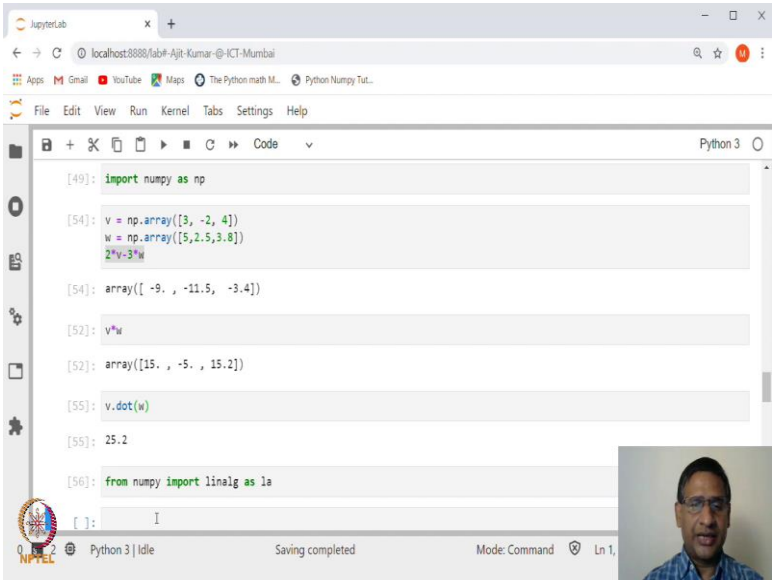
(Refer Slide Time: 02:43)



You can, you can even multiply v and w. So, if I say v star w, you will get the term by term multiplication. So, 3 * 5 is 15, -2 * 2.5 is minus point -5 and then 4 * 3.8 is 15.2.

So, this * gives you term by term multiplication. And this you could not have done in simple Python list. You have to write a small code for multiplying term by term two arrays, two lists ok, but here it is available. But in case you have two vectors, you also have notion, notion of dot product.

(Refer Slide Time: 03:30)

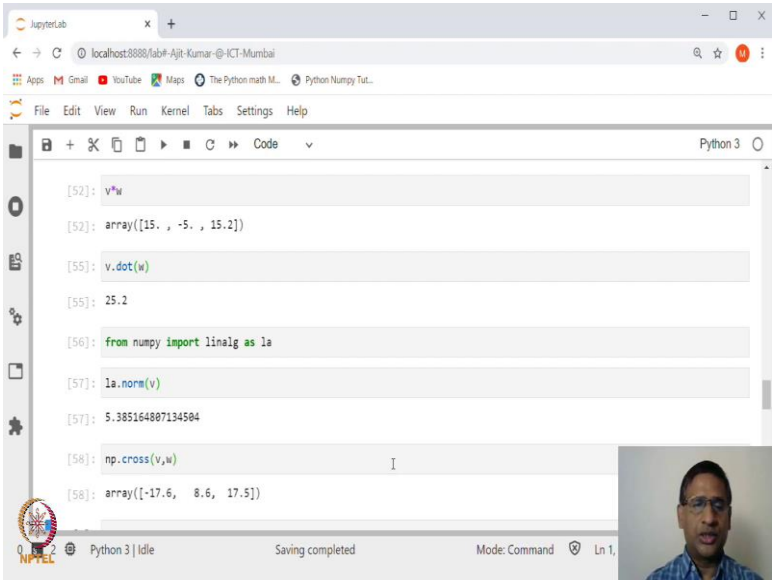


Of course, any scalar linear combination you can write; you could have written 2 times v plus, plus 3 times w or minus 3 times w, it will, 3 times w, it will, it will work.

So, any scalar linear combination of vectors you can find out. Suppose we want to find out dot product.

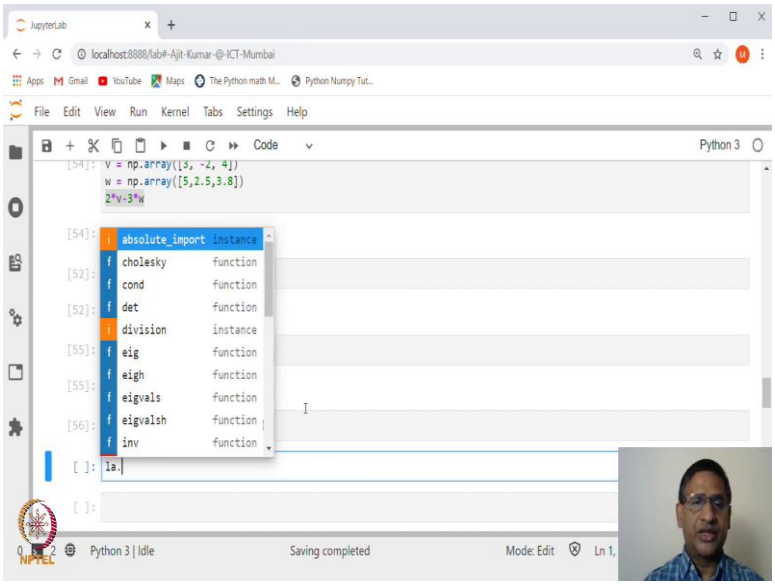
So, in that case you can simply say, v dot (.) and then dot and then w; it will give you dot product. If you want to find out, let us say, length of a vector, then you may need to use linalg package. So, let me import that, import. So, let me say from numpy, numpy, import linalg package and let us say this I will import as a la, ok?

(Refer Slide Time: 04:32)



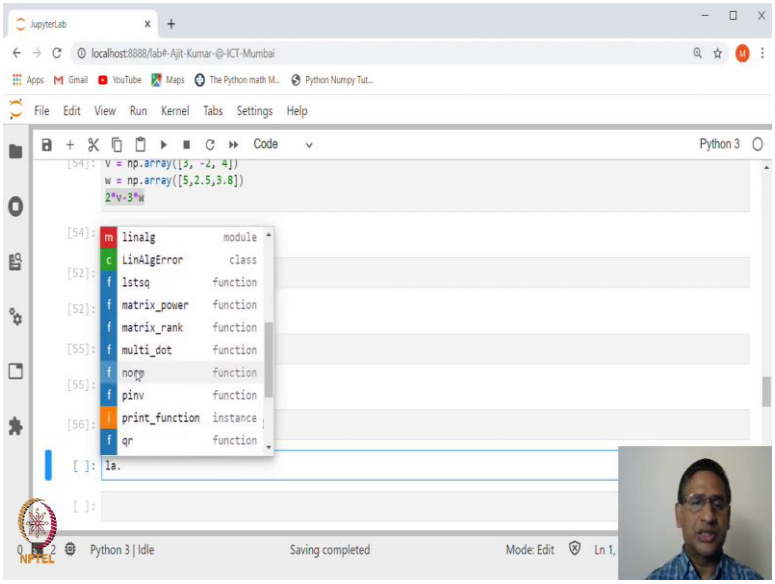
And inside this linalg package, again you have several methods. So, if I say la dot and, la dot and press tab.

(Refer Slide Time: 04:36)



Then you will see, for example, you have determinant, eigenvalues, eigenvectors, inverse, and things like that ok? Norm is the length of the vector.

(Refer Slide Time: 04:45)



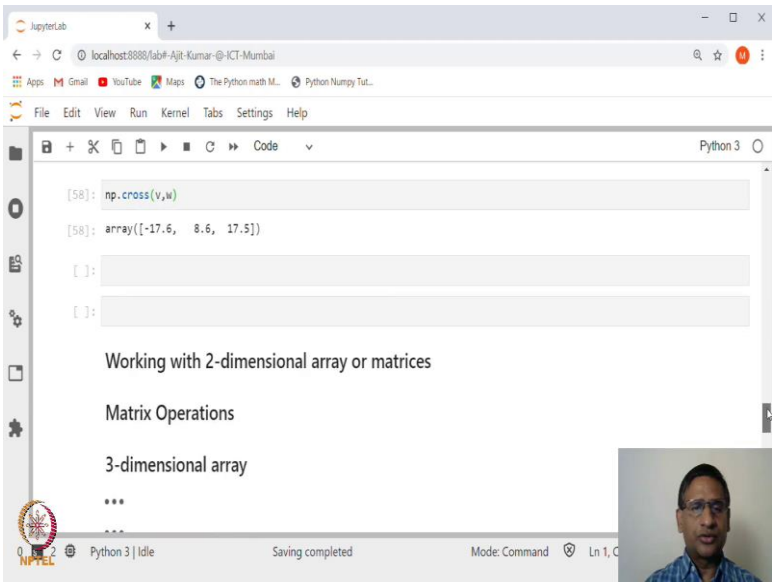
So, if I say this norm and if I say v, it will give me length of the vector, which is same as the square root of the first coordinate square, plus second coordinate square, plus third coordinate square; that is what is called the l-2 norm or Euclidean norm. You have also other norms you can find out, ok?

Similarly, you can find out even cross product. So, if you say np.cross and then if I say v comma w, you will get cross product.

So, all the standard methods of vector operations are available when you create one-dimensional array using NumPy array, ok?

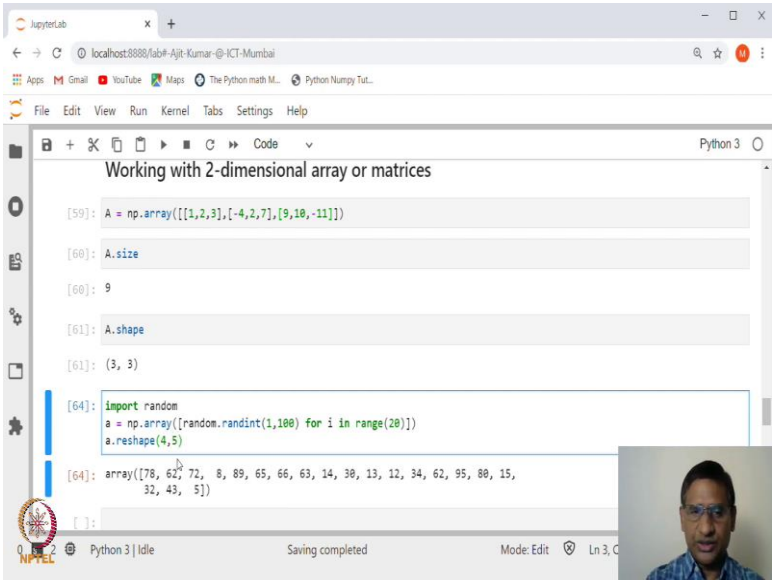
So, that is a very nice feature and other, you should try to explore other methods that you know or other operations that you know on vectors. Not only vectors in two-dimensional, three-dimensional; you can have vectors in any dimension and all these operations can be applied, ok?

(Refer Slide Time: 05:53)



So, now let us look at how to define 2-dimensional array or matrices and look, let us see some of the standard methods on matrices, ok? So, let us add some input cells here. So, how do we create 2-dimensional array? So, again the method is very simple, you can use same function np dot array, and these entries you have to put, it is a list of list; so that means you need to put the list inside a list ok, that is A.

(Refer Slide Time: 06:47)



So, let us create that. So, if I say, for example, let us say A is equal to np dot array and this is let us say, this is the list and inside that, each row you have to write in a square

bracket. So, the first row is let us say 1, 2, 3; second row is -4, 2, 7; third row is 9, 10, -11, this is a three by three matrix. You can find out what is A dot let us say size, size of A.

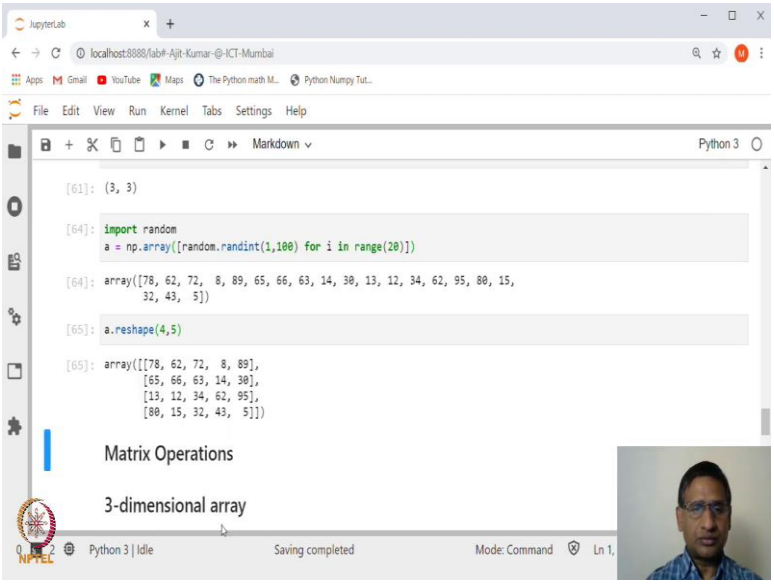
So, this is a size 9; that means it gives you the number of entries inside this. If you want to find out dimension of this, you can say A dot shape, A.shape; this is 3 by 3, so this is a three by three matrix. You can even reshape this matrix, you can apply, so, let us see, what do you mean by reshaping. So, let us generate, suppose we want to generate a matrix of order 4 cross 5 ok, matrix of shape 4 cross 5 and the entries are random integers from, let us say, 1 to 100.

So, first, we need to generate a list of random integers between, of 20 integers, between 1 and 100. So, how do I generate that? So, we will first import random. So, import, import random, and then create a list. So, list, let me create as A, A, which is a list of random numbers. So, this is in random.randint, and with, from 1 to 100, and this we want to generate 20 times.

So, for i in range 20, so there are, these are 20 random numbers and we will create an array of this. So, np.array, this is a list of 20 numbers. Now this, so let us see what is A? A is 20 random numbers as in one-dimensional array. Now, let us say A.reshape; suppose we want to reshape as a 4 cross 5 matrix.

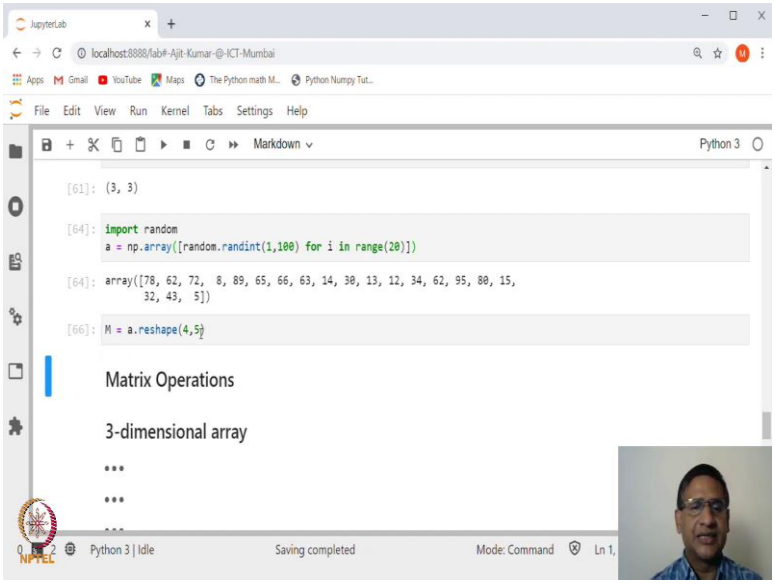
So, let me put it here and put it in the next input cell so that you will also see what is happening.

(Refer Slide Time: 09:37)



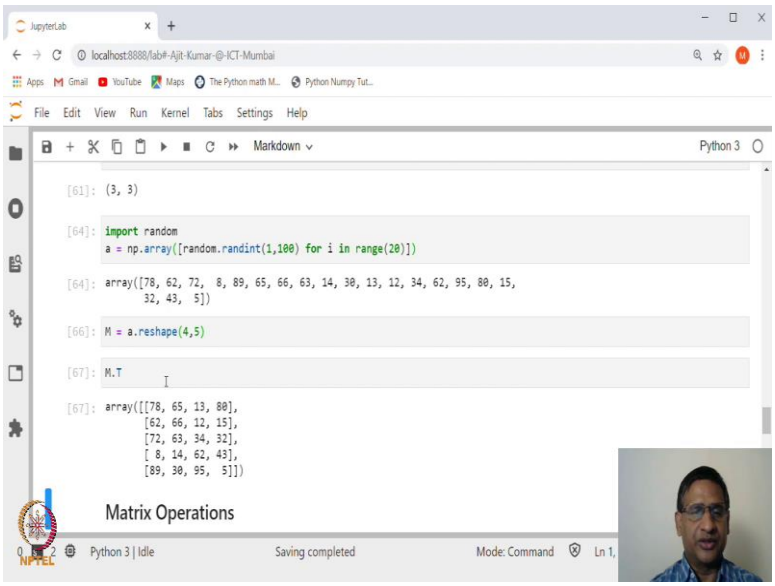
So, A.reshape and we want this to be 4 cross 5, ok? So, this is a 2-dimensional array. And how it has created? It has created A, first row is first five numbers, next row is next five numbers and so on. Of course, there is facilities, you can look at reshape function; you can reshape column-wise also, it should be possible.

(Refer Slide Time: 10:09)



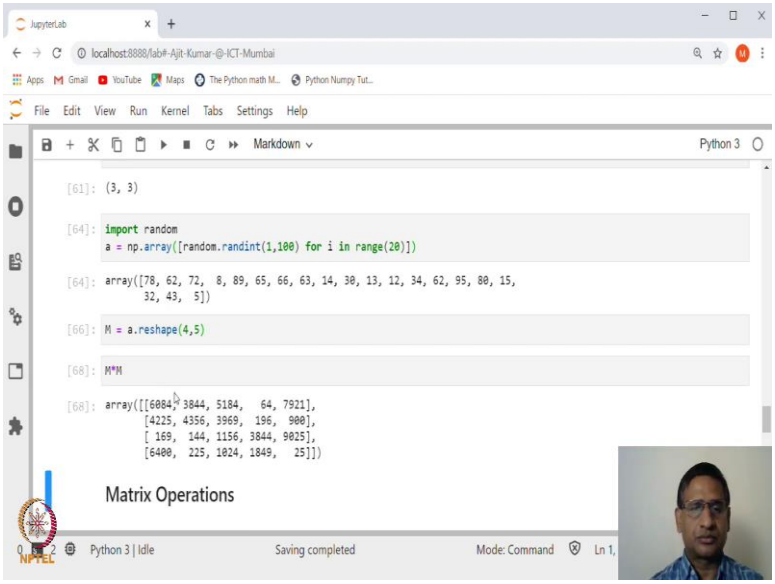
But however, so let us put this in M, M as a matrix, ok?

(Refer Slide Time: 10:16)



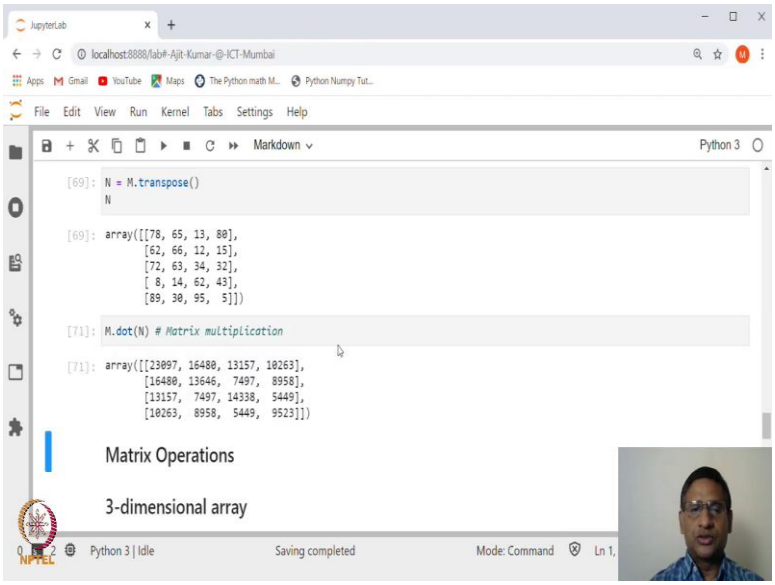
Now, if you want to find the transpose of this matrix, you can simply say M dot capital T, this is a transpose of this matrix.

(Refer Slide Time: 10:25)



If you want to write M star M, this will give you, this is a term by term multiplication as we saw in case of array.

(Refer Slide Time: 10:42)



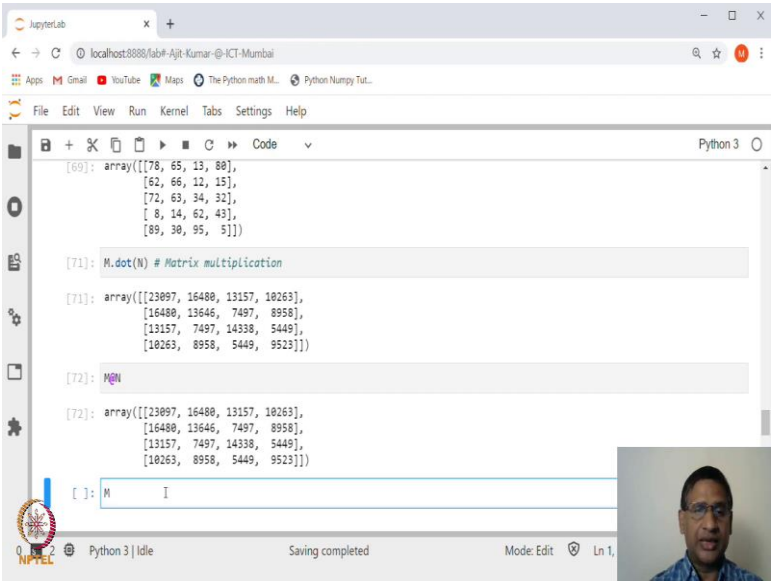
Suppose you want to have another matrix. So, let me,let me say my matrix N is actually transpose of M, so M dot T. You could also use a transpose function.

So, M dot transpose will also give you the transpose. So, I have two matrices M and N; one M is 4 cross 5 and N is 5 cross 4.

So, of course, M into N makes sense, you can multiply M and N. So, for example, if I say M star N, this will not work; because M star, star means term by term multiplication. If you want to use matrix multiplication; actually matrix multiplication of two matrices can be thought of as extension of dot product.

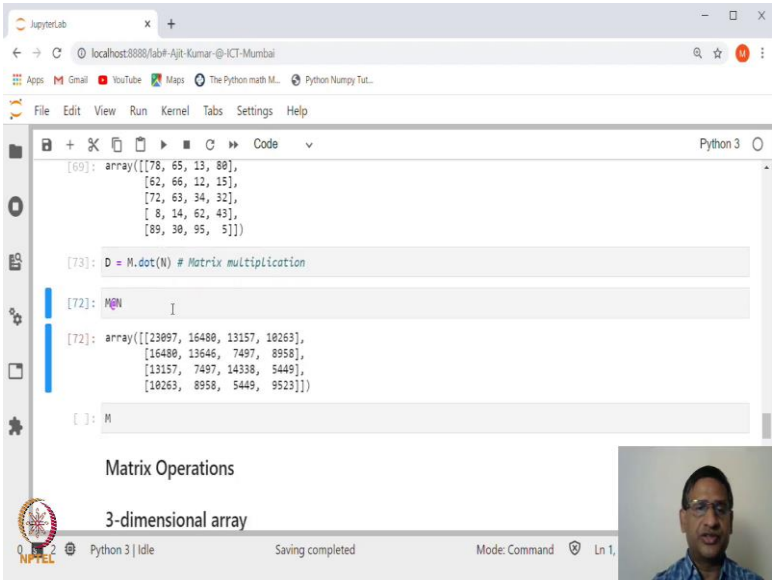
So, the same function M, we looked at how to find dot product of two vectors using dot, dot function. So, here also it will work, M dot and then dot and inside that bracket you write N. So, this is actually matrix multiplication; it is multiplying a matrix M with N. So, this is matrix multiplication, right?

(Refer Slide Time: 12:05)



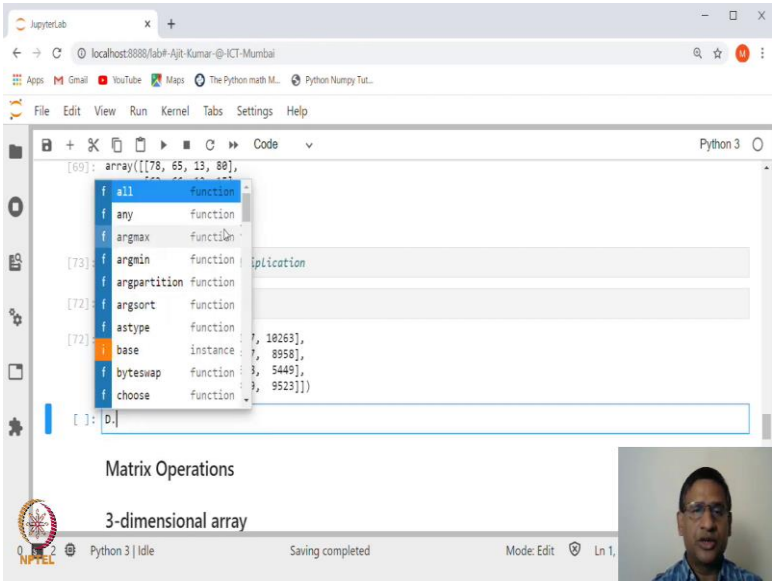
So, there is another way of multiplying two matrices; you can also say M and at the rate N, M @ N, this is also matrix multiplication. Now, some of the other operations on matrices you can find out. For example, if I want to find determinant.

(Refer Slide Time: 12:34)



So, if I say M, now M is not a square matrix; you cannot find determinant. So, let us store M into N, which is actually a 4 by 4 matrix. Let us store this into let us say matrix D, this is D, and let us we see, we want to find determinant of this matrix. So, how do I do that?

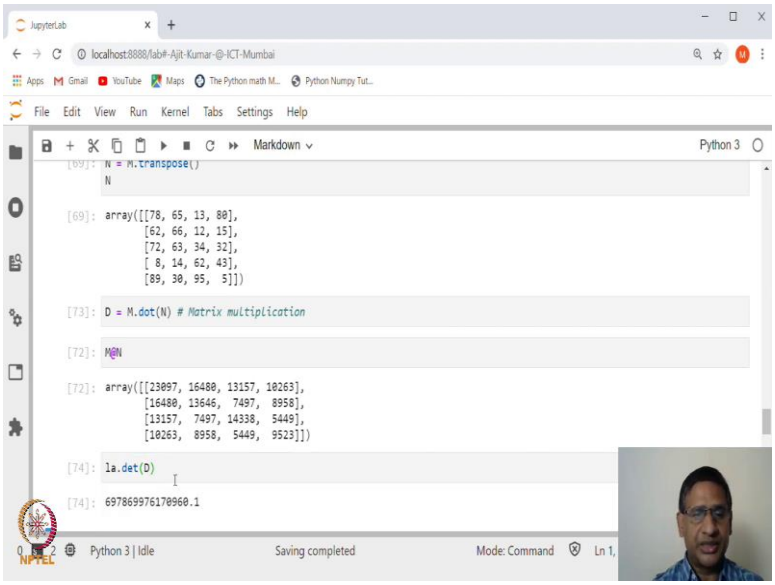
(Refer Slide Time: 12:50)



So, if I say M dot, M dot, sorry this is D dot and then press tab; you may not see the determinant here, there is no determinant function, but where it is? It is in linear algebra

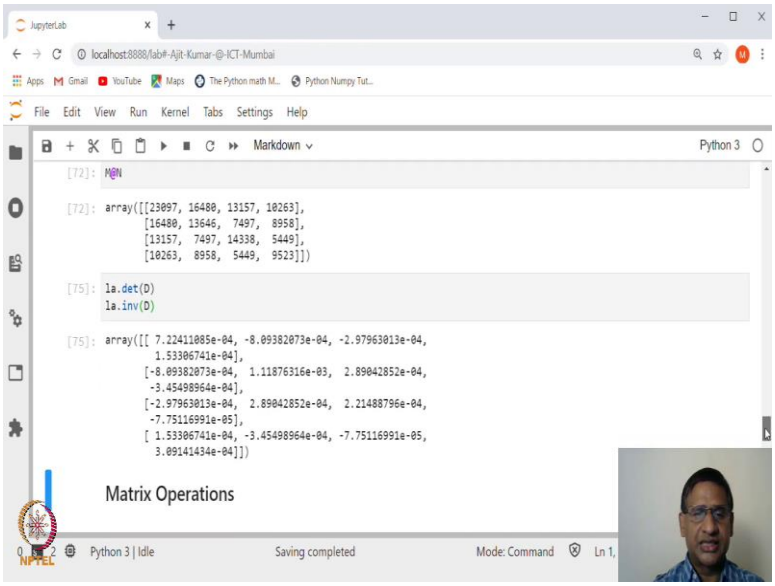
package. So, we have not imported linear algebra package. So, no, we have already done it.

(Refer Slide Time: 13:13)



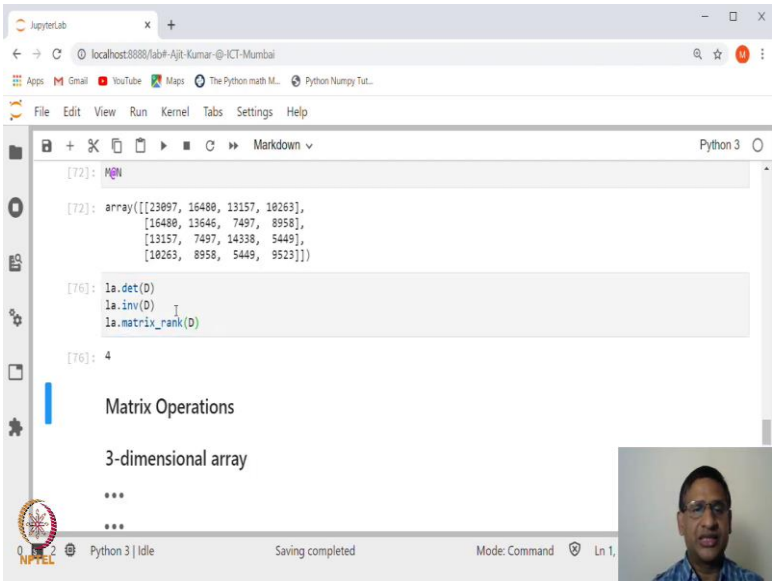
So, let us say la dot determinant, det of D. So, this is a determinant, this is quite big, right?

(Refer Slide Time: 13:28)



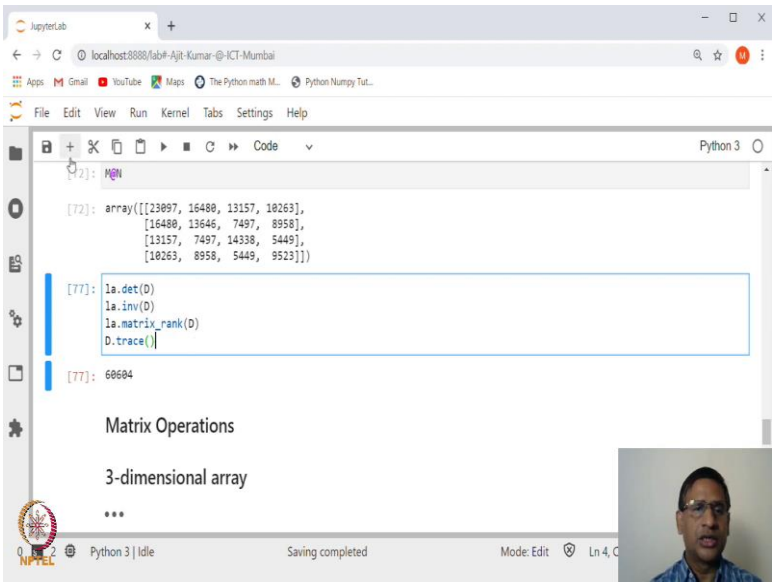
So, if I want to find out inverse of this; I can simply say la dot inverse of D, this is the inverse of this matrix.

(Refer Slide Time: 13:39)



There are other functions like, if I want to find out rank; I can say la dot I think matrix rank and then D, this will give me matrix. This is a matrix rank of D is 4, so it is, that is why it is an invertible matrix; the other operations you can try to explore this linalg package and inside that you can look at other functions.

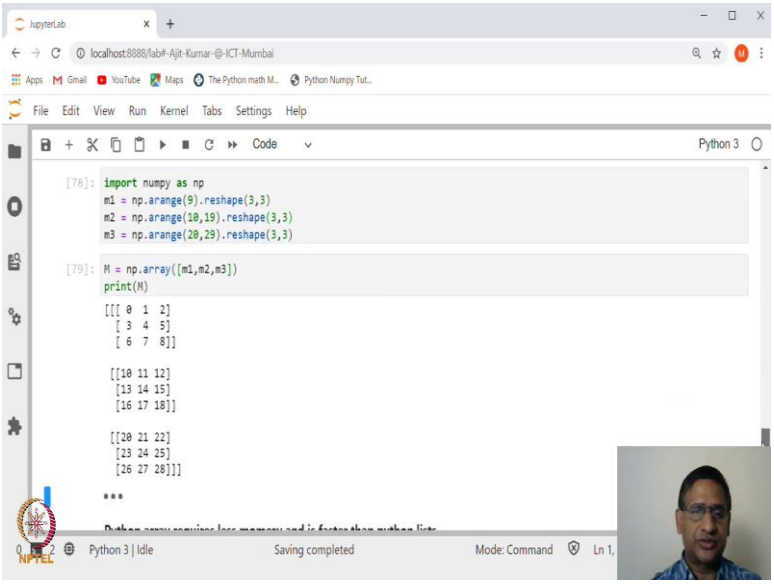
(Refer Slide Time: 14:03)



For example, you can find out trace; so you can say D dot trace, this trace is sum of the diagonal entries, right?

So, we know how to multiply matrices, how to find transpose, how to find determinant, how to find inverse, and other operations you can look at. Similarly, you can also find out various row operations etcetera. You can also define higher dimension array. So, for example, let us say we want to define a 3-dimensional array.

(Refer Slide Time: 14:41)

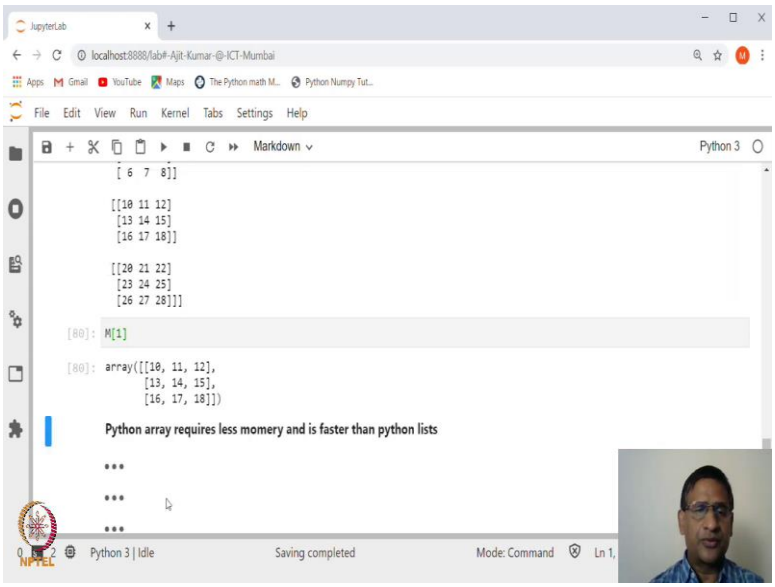


So, how do we do that? So, all you need to do is, so, 3-dimensional array will be actually a list of matrices, list of matrices. So, suppose we want a list of three matrices. So, let us say, first matrix is 3 by 3 matrix 1, 2, 3, 4, 5, 6, 7, 8, 9; second matrix is from 10 to 19 again 3 cross 3 and third matrix is 20 to 29.

So, this is a 3-dimensional array. Now, how do I create? So, I will simply say np dot array and inside a list m1, m2, m3. And if you look at, what is this list? So, this is a list of matrices 3, list of 3, 3 cross 3 matrices; instead of 3 cross 3 matrices, you can have other dimension also. So, this is a 3-dimensional array and you can create four-dimensional array, five-dimensional array etcetera.

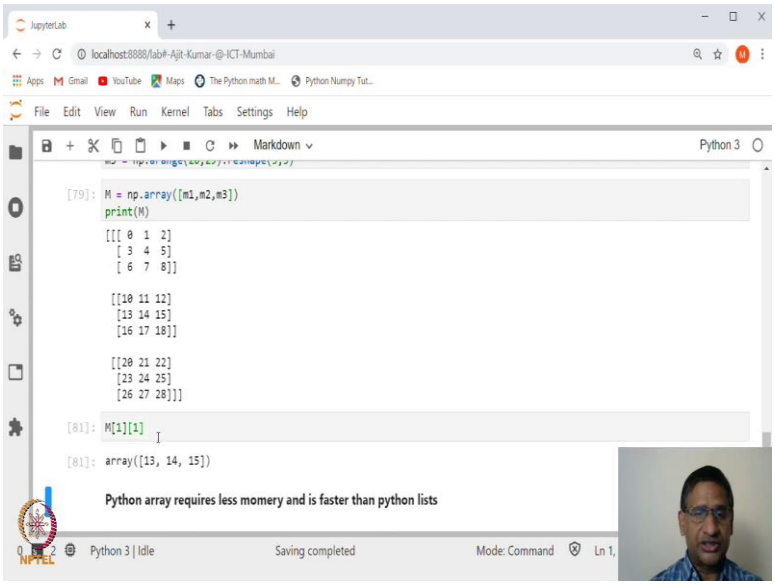
Suppose we want to find out how to get particular entries inside this. So, if you have a matrix, you can take any slice of it; similarly you can take any slice of any 3-dimensional matrix array also. So, how do we do that?

(Refer Slide Time: 16:04)



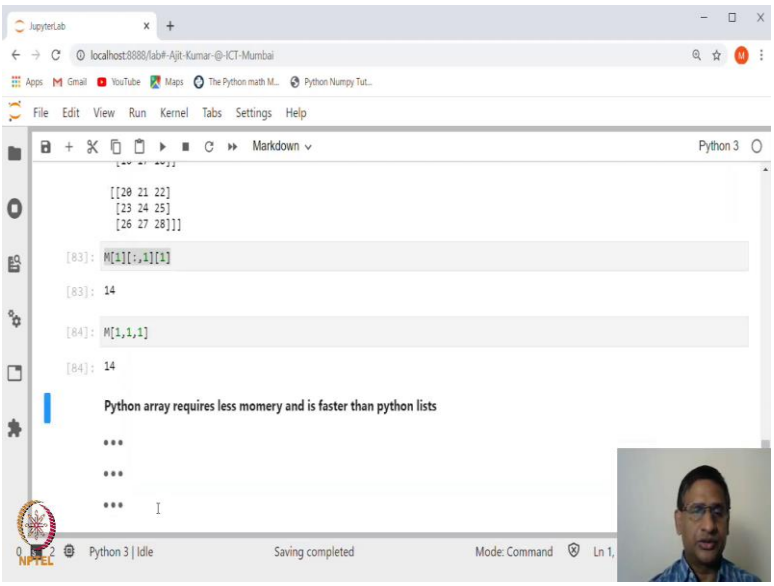
So, for example, if I, if I say, if I say, what is M of 1? So, M of 1, what was M? M was list of three matrices, m1 will be second matrix in that list. So, M[1] should be this small m2 which is a 3 cross 3 matrix starting from 10 and going up to 19. So, that is what you see here, M[1] is 10, 11, 12, 13, 14, 15, 16, 17, 18. So, this is the.

(Refer Slide Time: 16:31)



Now, suppose I want to find out first column of this. So, this is a matrix, I want to find, let us say, second column of this. So, second column of this how do I get? If I say 1 here it will give me second row right; that you can check, this is the second row of this. I want second column.

(Refer Slide Time: 16:54)



So, second column means, you look at all the rows and then second column, all the rows this means you can have a colon b; that means starting with a-th row to b, b minus 1-th row that is what it will give and this is 1. So, this will give you the second column of M[1] that is what you see 11, 14, 17. And if I want to find out, let us say, this of 1, first entry this is 14.

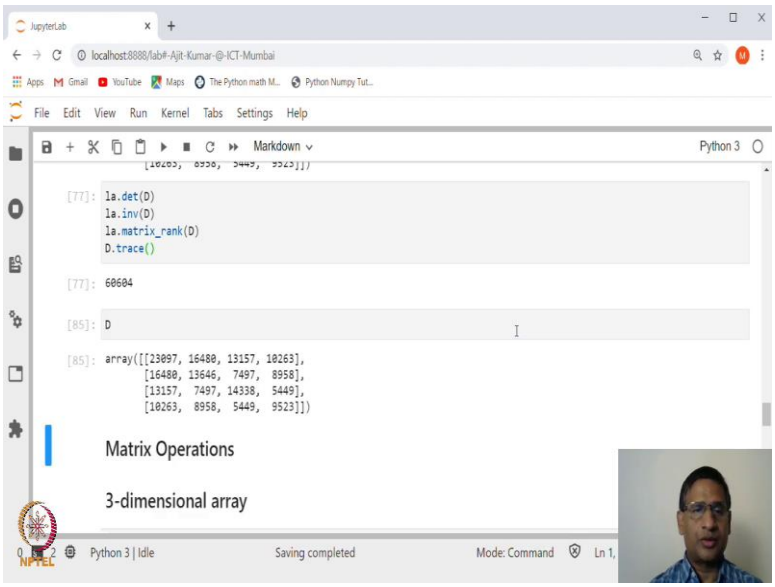
So, this we could have also found using just writing M of 1 comma 1 comma 1, right? So, this is 14. So, that is how you can take any slice of any array, slice of any array, ok? So, NumPy has very nice features to deal with arrays of any dimension.

Usually in mathematics, we make use of 1-dimensional array that is vectors, 2-dimensional array matrices; sometimes 3-dimensional array you can think of 3-dimensional array as a possibly colour image, any colour image can be thought of as 3-dimensional array, ok?

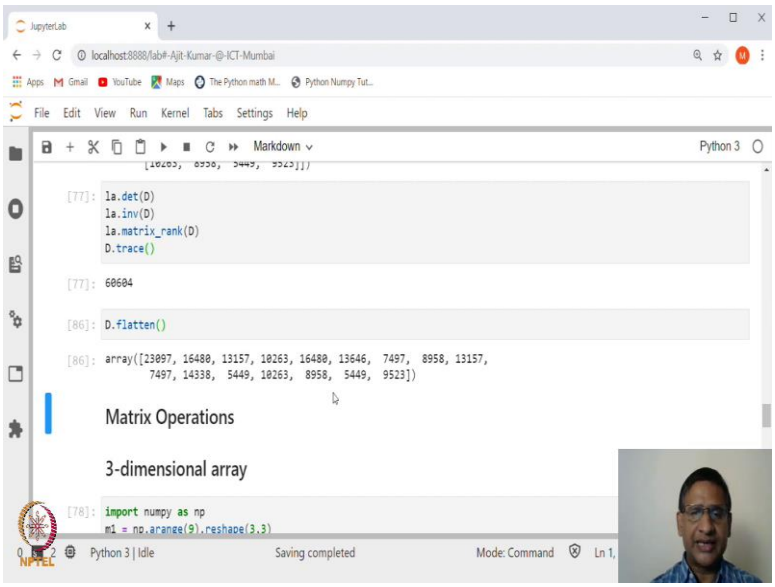
So, I expect all of you to explore other methods or operations on matrices. So, create some random matrices and you can deal with other operations that you know. Similarly,

you can, you can also look at another, another thing; for example, if you have a matrix, let us say for example, we created a matrix, we created a matrix let me say D, we have created a matrix D, ok?

(Refer Slide Time: 18:51)

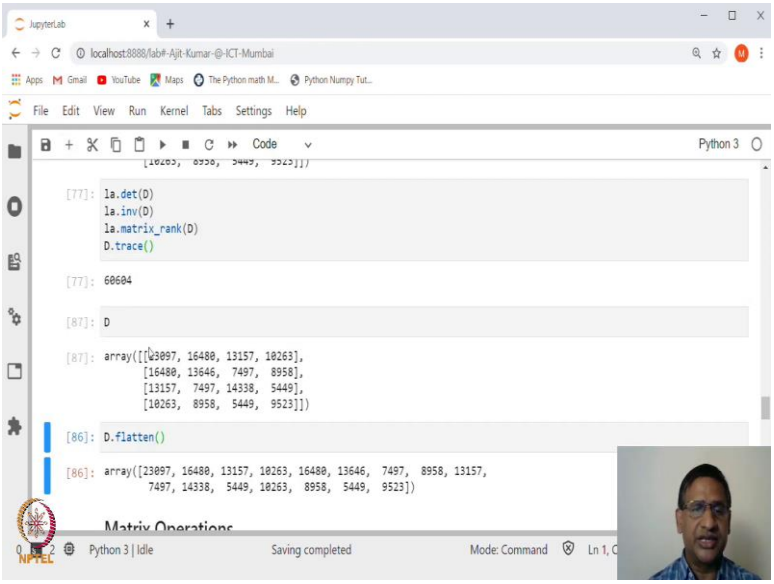


(Refer Slide Time: 18:59)



This is the matrix D, 4 cross 4 matrix. And suppose we apply D, look at D dot flatten, D dot flatten; what will it do? It will flatten this list, 2-dimensional list, and make a 1-dimensional list. And how it flattens? Actually if you, if you simply look at, let us see how it has flattened.

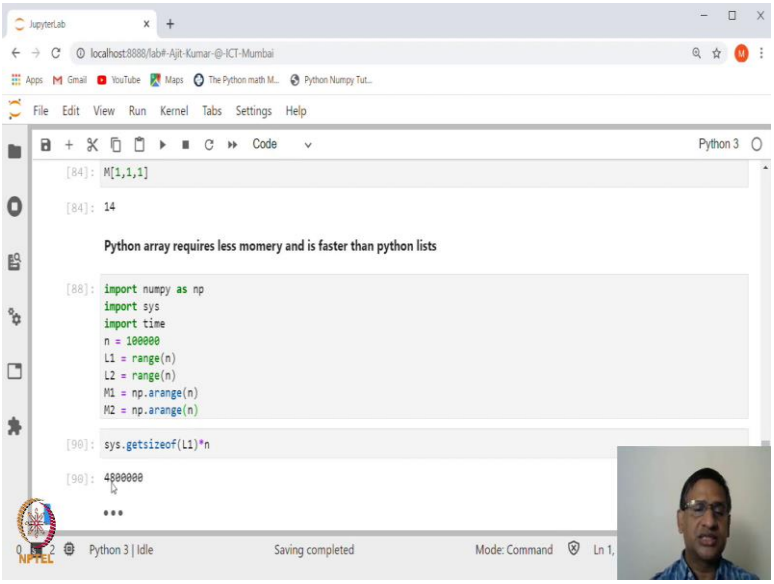
(Refer Slide Time: 19:21)



So, let us look at what is D and then compare this D with D flatten. So, D is this way it has flattened row-wise, right and that is how you actually, we created this matrix using reshape command.

So, once you have a matrix, you can flatten it; if you have a one-dimensional list, you can make a matrix using reshape command. So, all these functionality are available in this, ok? Now, let us look at, as I said in the beginning of yesterday's lecture; that Python array requires less memory and it is faster. So, let me just give you one more example why do I say this.

(Refer Slide Time: 20:07)



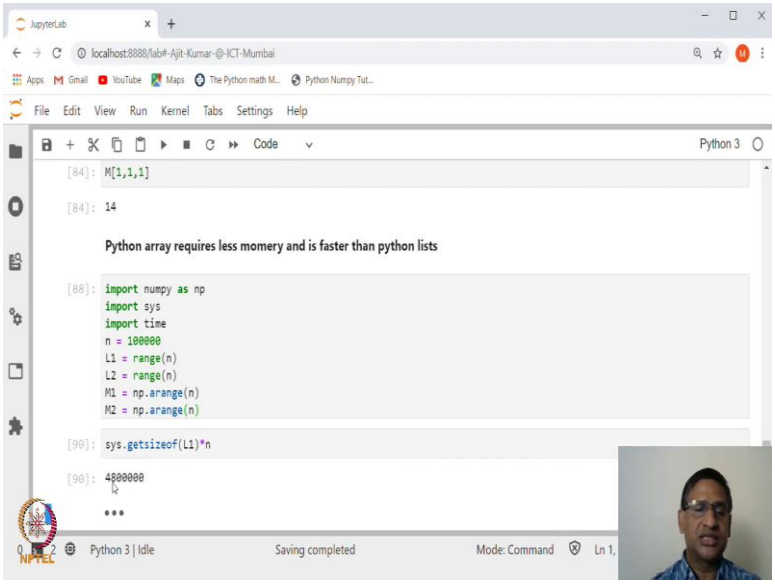
So, let us create a list using range function and also a list using arange function from numpy, ok?

So, we will import another module called sys, which will which will provide some functions to find out the size of any item inside a list. Similarly, let us import time module to find out time it takes to do various computation.

So, let us create a ‘n’ of this 10 to the power let us say 5. So, this is the for example, list of one million entries, and L1 is range n and L2 is also range n. So, the L1 and L2 are two lists.

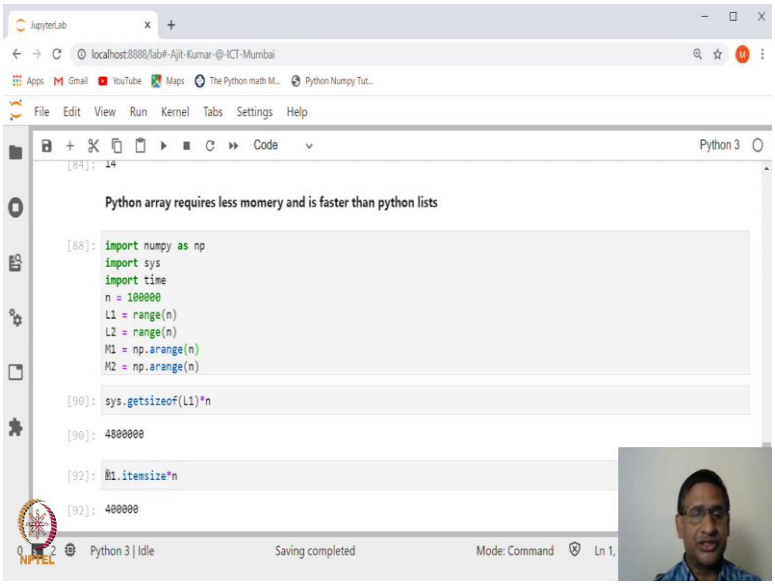
And let us say M1 and M2 are again same list, but created using arange function from NumPy library.Now, let us see, suppose we want to find out size of each entry inside L1. So, what is size of each entry inside the L1? That is, can be obtained by sys dot get size of.

(Refer Slide Time: 21:28)



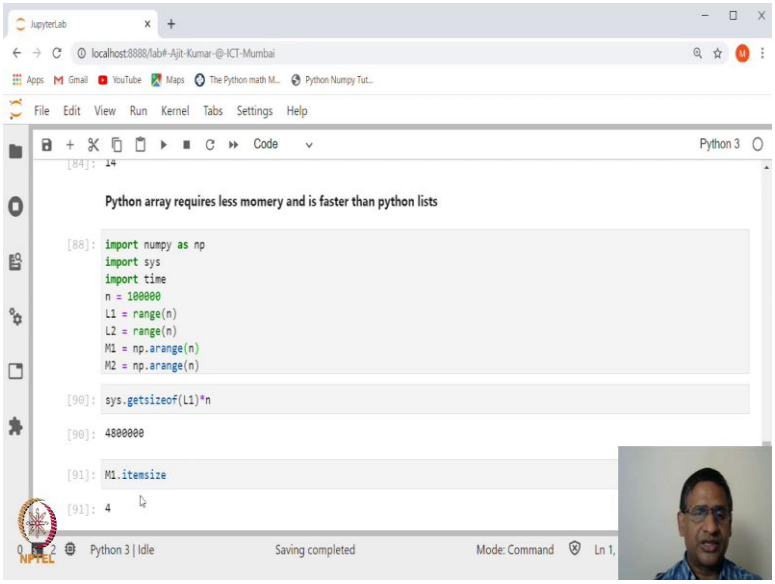
So this is the size of each entry is 48 byte actually. And so, what is the total size? So, this we will multiply by n, this is, this is the size of entire list L1 and similarly L2.

(Refer Slide Time: 21:47)



Whereas, if you want to find out size of each entry inside M1, which is created as an array, one-dimensional array of same length and so this can be obtained using M1 dot item size, item size.

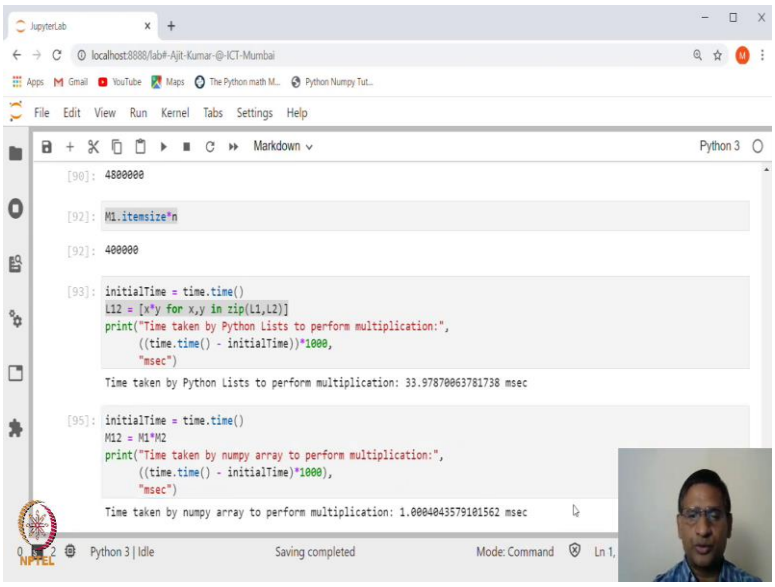
(Refer Slide Time: 22:02)



And so, this is if I say , this is 4, each entry has size 4 only; whereas for list it was, simple list it was 48.

So, that is a huge difference. So, if I multiply this by star n, this is the size of entire list. So, as compared to this, the memory required for storing array using NumPy is quite small. So, that is why you see, the python array requires less memory in order to store any list.

(Refer Slide Time: 22:41)



Now, suppose we want to find out, let us say we want to multiply two arrays component-wise term by term; we saw that in case of if you have a two lists, you can simply say multiplication in between, star in between.

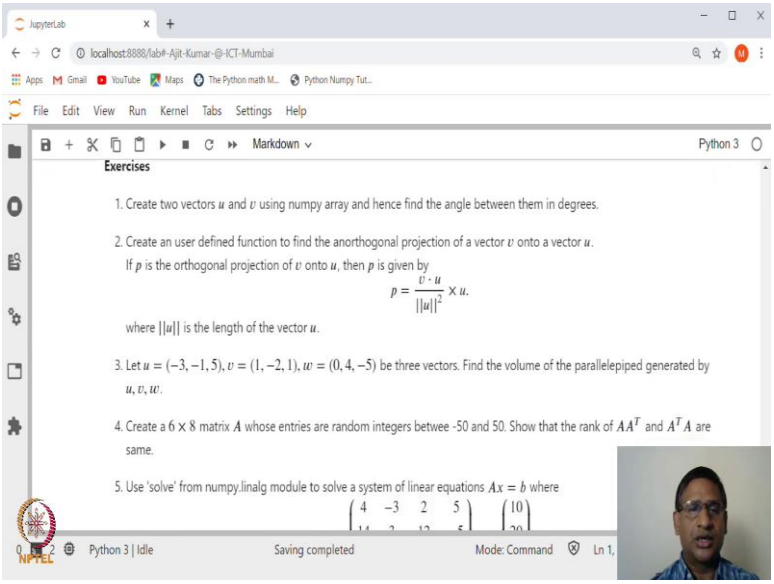
Whereas, star in between for a simple python list will not work; so what we will do is, we will create a simple comprehensive for loop for this. So, x star y for x and y in zip L1, L2; zip L1, L2 will create a list of tuples of L1 and L2. So, each coordinate of, suppose L1 has x i's and L2 has y i's; then it will create a list of tuples of (xi,yi), i going from 1 to 10 to the power 5, so this is multiplication.

And in the beginning, let us say, store the beginning time, the initial time as time dot time and then you find out this and then find out again what is the time, end time and subtract from the, subtract initial time from that.

And then since it is going to be very small, let us multiply this by 10000. So, it will give you in millisecond. So, it says that the time required for completing this term by term multiplication is 33.97 millisecond. Whereas same thing, if we do it for multiplying two arrays term by term; let us say M1 star M2, M1 and M2 were one-dimensional arrays created using NumPy array. And the same thing initial time you find out M1, M2, I have called for the list or array M1 into M2 and then again compute the time it takes and then, no I have to run once more.

So, you can see here this, this time taken is almost one millisecond. So, if compared to this, this is quite fast. So, that is what I meant by saying this NumPy array requires less memory, and also it is fast. So, that is one reason why the computation with array created inside NumPy is preferred, ok?

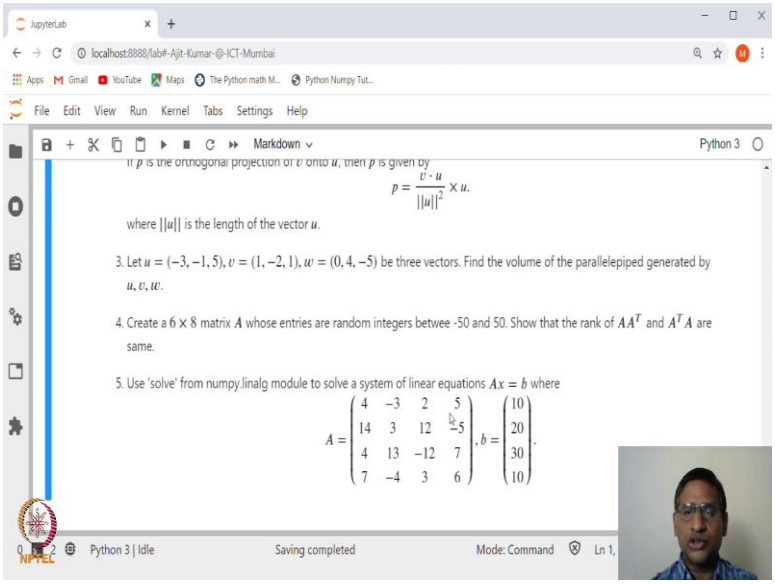
(Refer Slide Time: 25:05)



So, at the end let me give you simple exercises, which you should do; of course, we will be posting the solution of these problems, but you should not wait for us to post the solution, these are very simple exercises.

So, first exercise is to create two vectors u and v using NumPy array and hence find the angle between them in degree. So, angle can be obtained using dot product. And create a user-defined function to find the orthogonal projection of a vector u, v onto u and suppose p is the orthogonal projection; just for your convenience, I have given you the formula for orthogonal projection of v onto u. So, it is v, u dot v divided by length of u, square times u. We have already seen how to find the length, ok?

(Refer Slide Time: 25:58)



And the next one is, create three vectors u, v, w in three dimension and then find out the volume of the parallelepiped generated by these u, v, w. So, you must have seen as an application to cross product and dot product, but how to find the volume of this parallelepiped? Another exercise is to create a 6 cross 8 random matrix of integers, integers are varying from minus 50 to 50.

We already created one 4 cross 5 matrix, and then show that the rank of A A transpose is same as rank of A transpose A, which is also rank of A. In fact, it should show that, rank of A A transpose and it is same as rank of A is same as rank of A transpose A, ok? And another problem is, inside linalg package, there is a function called solve, solve, and which can solve system of linear equations.

So, suppose you take a system of linear equations say Ax equal to b, where A is this 4 cross 4 matrix, b is this, try to solve this system of linear equations using NumPy, inside NumPy use linalg. So, inside linalg, there is a function called solve, ok? So, these are the five simple exercises, you should be able to solve them.

Thank you very much; I will see you in the next lecture.