Scientific Computing Using Python

Professor. Vivek Aggarwal and Professor. Mani Mehra

Department of Mathematics

Indian Institute of Technology, Delhi

Lecture No. 04

Hello and welcome to Scientific Computing using Python. So in the last three lectures, we have discussed very basic commands of Python. So let's start further today.

First, I will open a new file, notebook. I will name it Lec4. In our last class, we discussed for loops, so today we will continue that.

Now today we will discuss how to define a function in Python. If we have to define a function in Python, then how can we do it?

To define a function, we use the keyword def. So to define it, we start with the word def. We have to see how to use it.

Now, if we have to define a function, then we need arguments in the function, and we will also have to give input to define that function. Because like in mathematics, when we define a function, we input a value from a domain in it and we get the output. Similarly, we will have to do this in Python as well.

So what I do is, first I write def. After that, I define the function. I write any function — I take the function name as my_function, okay. I named it and I have shown that we will give input in it. So like if we write f(x), then f and after that we take brackets and write x inside it. So we have defined my_function and put brackets in it. It means that we are going to define a function.

Okay, now what are we going to do in this function?

Now, if we are given the function x^2, $f(x) = x^2$, then we will put any input in it, and its square will come out. So now, what am I going to do inside the function? Suppose I am going to write print inside it so that I get to know that if I give any input, is it entering the function or not. So, for this, I write inside: Hello from inside the function. I am getting it printed.

```
def my_function():
    print( “ Hello from inside the function”)
```

Okay, so I wrote it and I defined it like this. Now I entered it, so there is no error — okay, no error came.

Now I call this. Suppose my_function (), so what do we have to do with this function? We have to call it because only when we call it, the values inside the function will run. So as I called it, the statement written inside it came: Hello from inside the function.

Okay, so what does it mean? We will call the function, input the argument value in it, and the output that we have inside the function — we will get it through print. It got printed here. So we can define the function like this.

Okay, so this thing that is in brackets, we call it an argument, okay?

So how do we get an argument? What do we do with an argument? The information that we have can be passed into the function using the argument. So this argument is the information that will be passed to us through this.

Okay, so like I had kept it empty earlier inside it, now I define — let's take, define a function. I will write this my_function. I am defining it.

Okay, so inside this, I have written a function name. I gave the name of an argument, and inside I wrote print. In print, I wrote fname + — okay, so after that I wrote: "Kumar". I wrote it like this.

Okay, so what did I do? I defined this function. Here I gave the argument fname. It means that this function is related to the name. So, what do we have to input from this? What is its domain?

We have to give in the domain — we have to send string characters inside it, and after that, we will get this in the output.

Okay, so I defined it. Now I will call it. Okay, suppose I called it — let's call, I wrote print or call my_function.

Now inside this, I defined — because we have to send the character, so I sent the name. Suppose Ram, I defined this. Okay.

So I sent this argument, and I will call the function once more. I wrote Rakesh inside it.

Okay, I am defining the function like this:

```
def my_function(fname):
   print(fname+ “Kumar”)
my_function( “ Ram”)
my_function( ‘ Rakesh’)
```

So what did I do? I entered it. So we got it written? Now what did I do? From here, we input Ram, and from here the function got called and it will go here. So what does it mean? That this input Ram went to fname, and what is happening inside the function? So, here I wrote fname. The value here will come here. Plus, it will get added to it — what I added is Kumar. Okay, so it came written: Ram Kumar. Then I sent someone's name Rakesh. After that, I inputted it here and the output was written: Rakesh Kumar.

Now what we have to do is — we sent the argument from here, and we got what was written in the argument. Now we have to see whether we can send two arguments or not.

So I defined:

```
def my_function():
```

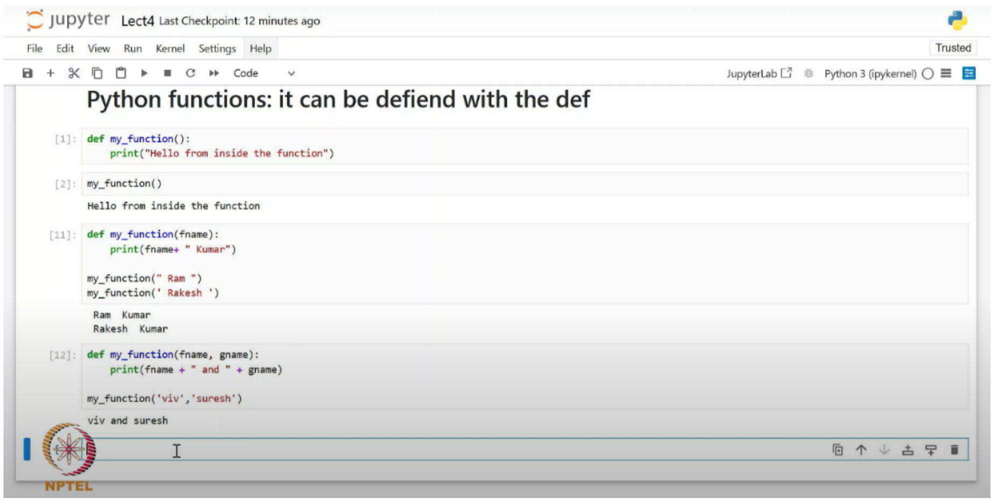
I defined my function, and what I did in it now was that I sent two arguments — fname and I sent the second argument, gname. Okay, I defined two arguments. And after that, I wrote:

print(fname + " and " + gname)

Like this, I defined the function and gave the print command inside it.

Okay, now I have to call the function. So I called my_function, and I wrote inside it — now what we have to do is give two names. So I gave two names just like this —'viv' and gave the second one like this —'suresh'. And I pressed enter. So from here, one argument went to fname, and the other one, which is Suresh, went to gname. So we sent two inputs in both the arguments, and whatever solution or output we had defined, that will be displayed here.

(Refer slide time: 10:01)


jupyter Lect4 Last Checkpoint: 12 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

Python functions: it can be defiend with the def

```
[1]: def my_function():
         print("Hello from inside the function")

[2]: my_function()

     Hello from inside the function

[11]: def my_function(fname):
          print(fname+ " Kumar")

      my_function(" Ram ")
      my_function(' Rakesh ')

       Ram  Kumar
       Rakesh  Kumar

[12]: def my_function(fname, gname):
          print(fname + " and " + gname)

      my_function('viv','suresh')

      viv and suresh
```

Okay, and if I write this function here , my_function , And I send only one argument, okay, I sent one argument, suppose anyone's name, any name I sent it, I don't remember — so what will happen here is that an error will immediately come:

"my_function is missing one required positional argument."

Okay, because we had two. So what are we getting? How many arguments did we send in it? We sent only one. So due to that, we may get an error. So we have to keep the number of arguments equal to the number of inputs. Okay, so if we forget how many arguments we have to give, then — if I write # No. of arguments — okay, I write number of arguments, that means I can give infinite number of arguments, , not infinite, but large number of arguments.

So what do we have to do for this?

I write def, I define the function first. Okay, so before defining, the name of my function came. So, like this we have to define —def my_function. Okay, now what did I do inside it?

I put a star — okay — and under the star, I wrote guest. What does it mean? That now I am giving the input of the guest's name. Okay, so now we don't know how many guests we have — how many guests are going to come — so we don't know this.

Let me print it. Okay, I write in print — Guests is/are: or whatever it is. I wrote it like this, okay, and I put a + and here I wrote guest.

Okay, so now we are taking the complete guest list. So let's see what comes now.

What did I do? I defined here. I wrote my_function like this, and in this I gave the input. In the input, I gave some VKA, I gave this, I gave one argument, SE, I gave the short form of some name, okay, and I gave MT. So we defined all these names. After that, I pressed enter.

Now here comes an error is written: “Can only concatenate str (not tuple) to str.”

Okay, so then I define it again. I write this — I write one — let's see what comes. Here we got the output. So, what do we have to do? We have to tell the guest which guest we are typing here — which guest we want to print.

```
# No. of arguments

def my_function( *guest):

   print( ‘Guest is/are ‘ + guest[1]


my_function(‘VKA’ , ‘SE’ , ‘MT’ )
```

So suppose I want to print the second guest, then the second guest will be written. Okay, I want that the first guest, whoever comes, should be told first of all — that in our party, who is the first guest in it? So, if the first guest was VKA, then his name will be displayed.

Okay, if we want to see the last guest — that who is coming last — then MT is coming. In this case, there are three. I can also increase it to four. Okay, right?

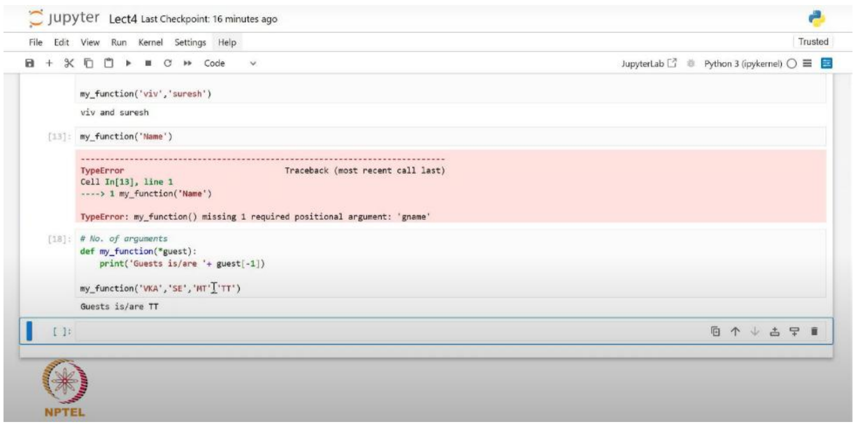
I gave some such names and I saw who was our last guest — then it will be written here.

Okay, so what can we do with this?

That we don't know how many guests will come to the party, so we will make guest an argument by putting a star, and according to that, we will keep giving input for whatever number of guests come.

So we can define it like this.

(Refer slide time: 14:40)



jupyter Lect4 Last Checkpoint: 16 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
my_function('viv','suresh')
viv and suresh

[13]: my_function('Name')

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 my_function('Name')

TypeError: my_function() missing 1 required positional argument: 'gname'

[18]: # No. of arguments
def my_function(*guest):
    print('Guests is/are '+ guest[-1])

my_function('VKA','SE','MT''TT')
Guests is/are TT

[ ]:
```

Now let’s see what else we can do in this.

Now I want to define a mathematical function, so I wrote my_function, and I defined here. I define a variable like we do in maths. So, I have defined x, and what is x doing? It is returning — okay, returns. So what is this return command? It returns our output. Now we don't want to print it. Now what do I have to do? Suppose I have to return, and what is happening in the return is that suppose I take 7 and define its power x. For this, I have defined this function:

```
def my_function(x):
   return 7**x
```

Okay, so you will give the value of x, and the output will come here. Now I have defined such a function. I called this function here as my_function.

Now suppose I wrote 2 here (my_function(2)) . So what did I do? The value of 2 will go here, the function will be called, the input will go into it, and what it is returning here — in this return — it will raise 7 to the power and return its value, meaning it will show it to us in the form of output. So, as I called it output 49 came. Okay, so we have this.

So now what are we doing in this — whatever elements I will input here, I input 3 , this came 343. Okay, I input 1, this came 7. So we can input like this, and the values will keep on returning.

Okay, so either we can tell through print, or we can return any value.

Like this I define. I define something else — define my_function.

Okay, so what did I define here? I defined x with a backslash. Okay, in this, I print(x). I printed this. Now I call my function. Now I called any value let 5 — it means the 5 value which went here — and the value of x got printed here. Okay, so we printed its value like this.

```
def my_function(x,/):
   print(x)
my_function(5)
```

So what did we do here? Here x was a variable name. We defined it. Here also we defined a variable with this symbol. So here, the value that we have ,that is 5 came here. Okay, so like this we can define functions. We can send an argument. We can send a number. We can send a character. We can send a string. So, we will use all these things in the future. So, we can use functions in this way.

Now we have done a lot of commands. So what do we do now? We will use the inbuilt libraries that will be useful for us in the future.

So, the most important library among those libraries that we will use is NumPy — Numerical Python.

So what will we do? We will use NumPy. So it is a short form of Numerical Python. Now look — we have to deal with numerical values. We have to do numerical work. So you know that our numerical values are either in vector form or in matrix form. So now we are going to deal with that. So, what should we do for that? We have a module — a mathematical module — which we call module. In this mathematical module, we can do all the mathematical operations and calculations very easily.

So, we will see how to use our NumPy module. If it is not installed in your Python, then you will have to install it. Otherwise, as we are doing in Anaconda — the module is already installed in it.

So, what we have to do in this module is — this module deals with arrays. Its specialty is that it deals with arrays. We know that arrays are collections of elements. Like, in the form of vectors, we have defined a vector or matrices. So, if we have to deal with all these arrays and deal with them in a fast way, then we use NumPy.

So what we will do in NumPy is that the first thing we have to do is — in NumPy, we will have to write:
import numpy

So what will happen with this is that it will get imported into our Python. After that, we said that this is used to define arrays — which means vectors which we define — can be vectors or matrices also.

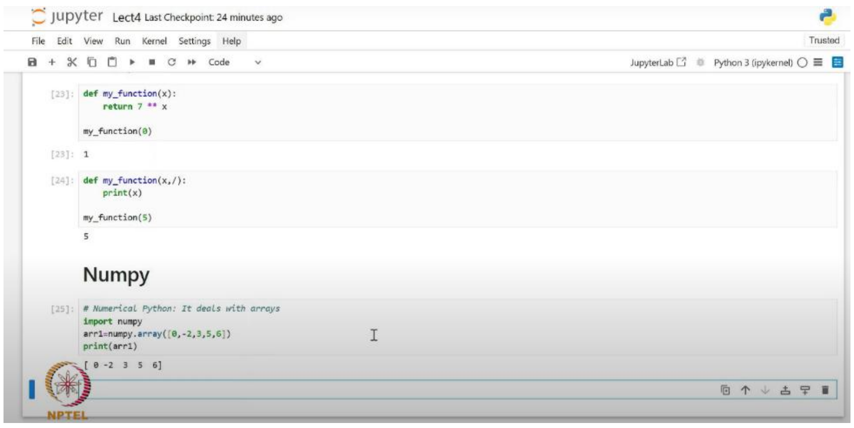
So we defined an array. Now how to define an array? So I did:
arr1 = numpy.array(...)

Okay, so what we did is that NumPy is a very big module. In it, there is a command array. We called that command with .array. And after that, I defined an array in it — I defined something like: 0, -2 , 3 , 5 , 6 , 7. So I wrote an array. Okay, after that, what did I do? I printed this array. Now we see this array is written — but what does it mean?

The first thing we did was import NumPy. And after importing NumPy, we have to call it. Now there are many functions defined inside it. Okay, so we have to generate an array. So what did we do? We wrote:
numpy.array(...)

And after that, we defined the value of the array and named it array1 . I printed it. So, we got this value: [0 -2 3 5 6]

(Refer slide time: 22:01)


jupyter Lect4 Last Checkpoint: 24 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[23]: def my_function(x):
          return 7 ** x

      my_function(0)

[23]: 1

[24]: def my_function(x,/):
          print(x)

      my_function(5)

      5
```

Numpy

```
[25]: # Numerical Python: It deals with arrays
      import numpy
      arr1=numpy.array([0,-2,3,5,6])
      print(arr1)

      [ 0 -2  3  5  6]
```

Okay, now what to do with this? We can write it in short form also. So I can write here that :NumPy can be written as np? So, we can write its short form as well — like in life also, if someone's name is very big, then we can call him by a short name.So, what will we do here? We import:
import numpy as np

I have defined it here also. Now I will call NumPy as np.

I defined an array. After that I defined another array. Now what will I do? I will write:
np.array(...)

Why? Because I have defined here that now whenever I write NumPy, it means I will write np. So all the commands are in np. So, I have defined the array — something like: [1, 2, 3, 5, 10, 12]. I have defined it like this. And after that, I called it, and it got displayed.

So, it means np — which is NumPy is a word of five characters. There is no need to write it again and again. We will directly write np. Okay, so we can write np.

Now I write the command print, so numpy is okay, and here I have written version.
"AttributeError: module 'numpy' has no attribute..." — this one, okay.

Now, what we have done is that we have defined the array. Like this, I have defined an array. I can also do an array of zero dimension, I can also do an array of one dimension, I can also do an array of two dimensions. So, what do I do now? I see arrays of different dimensions, okay? Now, like I have written this array, what type is this array? So what do I do? I print it, okay? And in print, I write the type (print(type(arr2)), and what will be the type of the array that I have defined? So the class that is written is <class 'numpy.ndarray'> , okay? So this is a numerical array from the class of numpy, meaning it is used from the module. So this is its class.

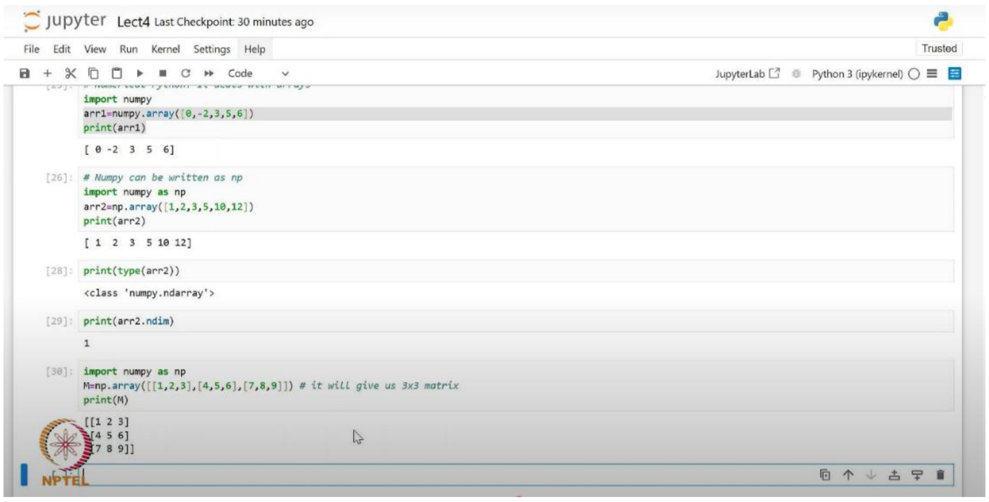
Like this, I can also write its dimension, okay? So, I have defined it, print, and what do I do now? I try to see the dimension of this arr2. So I wrote .ndim, so its dimension is 1, okay? So one dimension — because it is a simple one vector, an array whose values are one to a single value — so its dimension becomes one dimension, okay? So, like this, now instead of taking one dimension, I want to define two dimensions. So, what do I do? I write a new one. So, I write import numpy as np, okay? Now I define, suppose, matrix = np.array, okay? So what did I do? Now we have to go to two dimensions, and so what do we have to do? I am defining a matrix.

So in the matrix, I wrote:

M = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

I defined it in the form of a matrix. Now let's see what happens — print(M) and I printed M. So I wrote that matrix. So what will this give us? It will give us a 3 by 3 matrix, okay? So the matrix comes whose first row is [1, 2, 3], second [4, 5, 6], and [7, 8, 9]. So here we have a 3 by 3 matrix.

(Refer slide time: 28:23)


jupyter Lect4 Last Checkpoint: 30 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
import numpy
arr1=numpy.array([0,-2,3,5,6])
print(arr1)

[ 0 -2  3  5  6]

[26]: # Numpy can be written as np
import numpy as np
arr2=np.array([1,2,3,5,10,12])
print(arr2)

[ 1  2  3  5 10 12]

[28]: print(type(arr2))

<class 'numpy.ndarray'>

[29]: print(arr2.ndim)

1

[30]: import numpy as np
M=np.array([[1,2,3],[4,5,6],[7,8,9]]) # it will give us 3x3 matrix
print(M)

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Now I have to see its type — if I want to check what is its type — okay, so I define M here, I wrote this: print(type(M)) , and it came to know from here that it is a numpy array. Now I got a numpy array, okay? So what do I do? Now I see its dimension. So what do I do here? I will try to find out its dimension — how to do it: M.ndim. Okay, so this will print the dimension of M. So 2 is here, which means the array that we have is two-dimensional. Okay, so how will we know it is two-dimensional? Because every member of it in array. Like, there is one element — its dimension is one, then the second element — its dimension is one, then another element — its dimension is one, okay? So when we arranged one-one dimension, then we have an array of two dimensions, okay? So now we have an array whose dimension is 2.

In this way, we can define any dimension — even 3 or 4 dimensions. So, it is not necessary for us to have only 3 by 3 matrix. Now I will take it and I have defined it M1 . It is not that I have to do only this. I have done this, okay?

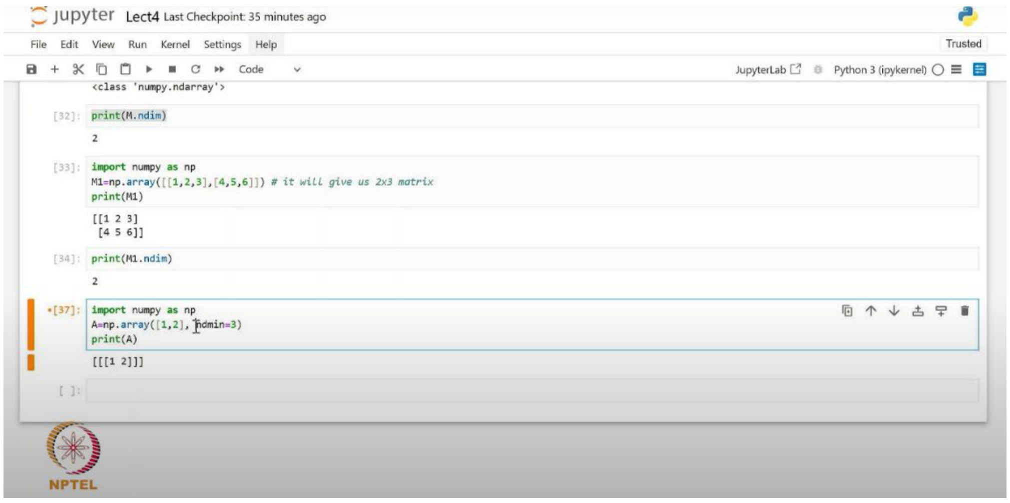
M1= np.aray([[1,2,3],[4,5,6]])

So this is the matrix that we have. Now in this, we have two vectors, two rows basically — first row and second row. Let's see what will happen in this. I did this, so we have a matrix, okay? Now this matrix has one row and two rows — what will be its dimension? If we want to see its dimension, then what do I do? I press control V here and try to check its dimension — so this is two dimensions, okay? So we have this two-dimensional array.

So, like this, we can define anything. Suppose I defined something, now I write import numpy as np, I wrote this, okay? So now what did I do? I write an array — I want to define an array. I wrote a, okay? So I wrote np.array, and I defined it like this: [1, 2]

I wrote a vector, okay? So what did I do with this? Used these elements, and here I want that its dimension should be 3. I defined it, okay? And I wrote A=np.array([1,2],ndmin=3), which means that I am defining an array whose values are given and its dimension is also given. Okay, I wrote this, and after that, I also printed it — I printed this.

(Refer slide time: 33:12)


jupyter Lect4 Last Checkpoint: 35 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
<class 'numpy.ndarray'>
[32]: print(M.ndim)
2
[33]: import numpy as np
      M1=np.array([[1,2,3],[4,5,6]]) # it will give us 2x3 matrix
      print(M1)
[[1 2 3]
 [4 5 6]]
[34]: print(M1.ndim)
2
[37]: import numpy as np
      A=np.array([1,2], ndmin=3)
      print(A)
[[[1 2]]]
[ ]:
```

So, I defined it, okay? Now we have an array A, whose elements are [1,2] and whose dimension is 3. So in this way, we can define different types of arrays with the help of n dimensions. So, from here we have defined it, how many dimensions it should have — so on that basis, we have defined it here.

Now let us see how we can do indexing. So let's say we have to define # Numpy array indexing. We know what is indexing — giving it numbering, doing indexing.

So I write print. Now, as I had defined it above, now I write M1 .So, it is an array as we have defined, now I write print. Above, I had mentioned an array of one dimension — array2. So suppose I write print(array2[0]), I am calling first element of array 2.I got it written.

Now look, the array2 which we have — I write it here. I print it again — array2. So what is array2? It is array([1 , 2 , 3 , 5 , 10 , 12]). Now what I have to do is indexing. Now this is the first element of this array — this is the zero value. We start it from zero. Its indexing is:

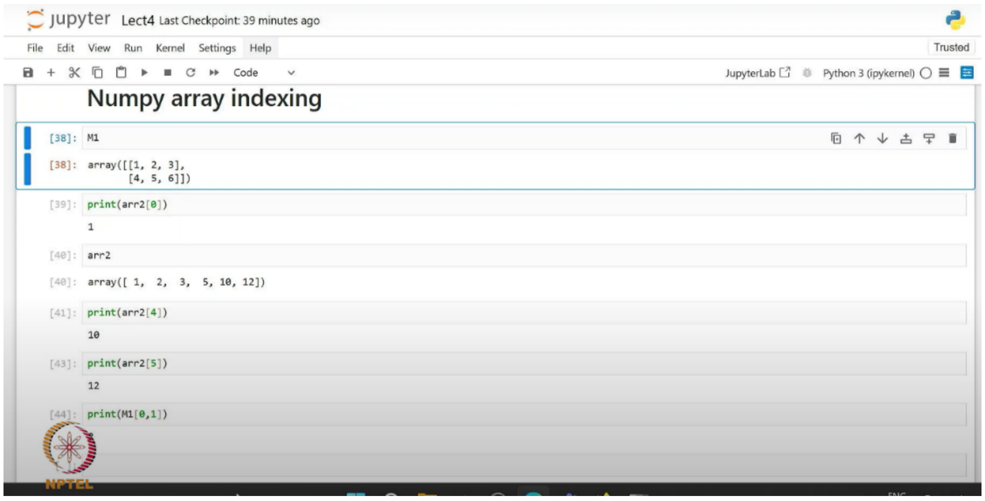
1→ first element , 2 → second element , 3 → third element , 5 → fourth element , 10 → fifth element , 12→ sixth element.

Okay? So inside this, we have a total of six elements, but the indexing which starts in Python starts from zero

So, if I called to print the first element of this array which is at zero position, then it got printed here and it got written as 2. Okay. Now, if I print it, then this came to us. Now I made it four, so the value of 10, which was at the indexing of four, got printed.

So, what did we do here? Array2 , and five, the indexing element—the sixth element—we printed that. It is written there. We can print it like this. Now just like we printed the one dimension, now what I want to do is I want to print the two dimension. So, I wrote print(M1[0,1]) here. Now let's see. I want to print the element of this matrix. I want to print this. I want to see what the element would be. So, we have 2. Now look, zero and one. The M1 that we had is this. So, in M1, the first element is zero-zero element. The second element is zero-one element. This is the zero- two element.

(Refer slide time: 37:30)



Numpy array indexing

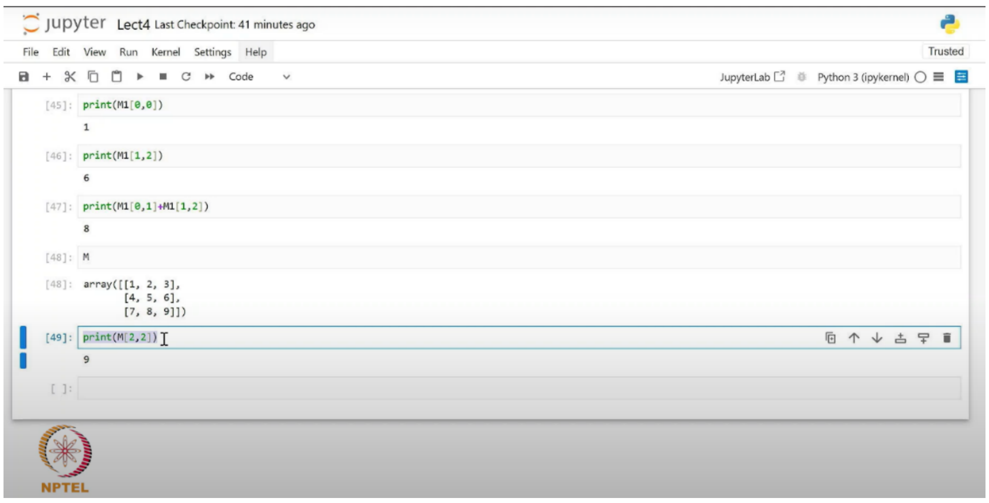
```
[38]: M1
[38]: array([[1, 2, 3],
             [4, 5, 6]])
[39]: print(arr2[0])
      1
[40]: arr2
[40]: array([ 1,  2,  3,  5, 10, 12])
[41]: print(arr2[4])
      10
[43]: print(arr2[5])
      12
[44]: print(M1[0,1])
```

So, whenever indexing happens in Python, always keep in mind that it starts from zero. Now I have to print this. But what do I do? I try to print it. So one came. Zero-zero element, which is the first element—I print it like this.

And if I write 1, 2 here, then the element written as '6' is displayed. So we can call any element like this. I can also do this here, that print—now I wrote—M1, which was our array and the element of our array I took is [0,1], Suppose I added it with [1,2] element of M1 . We can also do this. So 8 is displayed.

So, what was on M1[0, 1]? Our values were ‘2', and at M1[1,2] was six. So I added both of them and got 8. So, we can also define the mathematical operation here. So if we are told to print any element like this—now the M we had was a 3 by 3 matrix—so someone tells me that, "Bhaiya, call it, please call it in this," any element which we want to see, what is the value on [2,2], which element is this of this matrix? So, when we called, it came 9. So, the last element which is there will be typed here.

(Refer slide time: 39:41)


jupyter Lect4 Last Checkpoint: 41 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[45]: print(M1[0,0])
      1
[46]: print(M1[1,2])
      6
[47]: print(M1[0,1]+M1[1,2])
      8
[48]: M
[48]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
[49]: print(M[2,2])
      9
[ ]:
```

Like this, we can take any other. And now if someone tells me that, "Bhaiya, its element [1,1] show me what it is," so '5' has come. So this is the diagonal element. So we can take out all its elements. Zero-zero. So it comes like this, okay.So with its help, we can take out any element. We can call any element of any array with the help of np. We have defined an array in numpy and after that, called it. Okay. So now we can take any values.

Now I have defined an array. Okay. So instead of a two-dimension array if I take a three-dimension array, then I write here matrix M3 as array.. So let's first write: import numpy as np.

Now I have defined an array. So, in that, we have written np.array. Now I have to define in it. So now I have to define the values. So I took : M3= np.array([[[1,2,3],[4,5,6]],[[7,8,9],[2,4,6]]]) .Okay. So this is done. Now I printed this out.

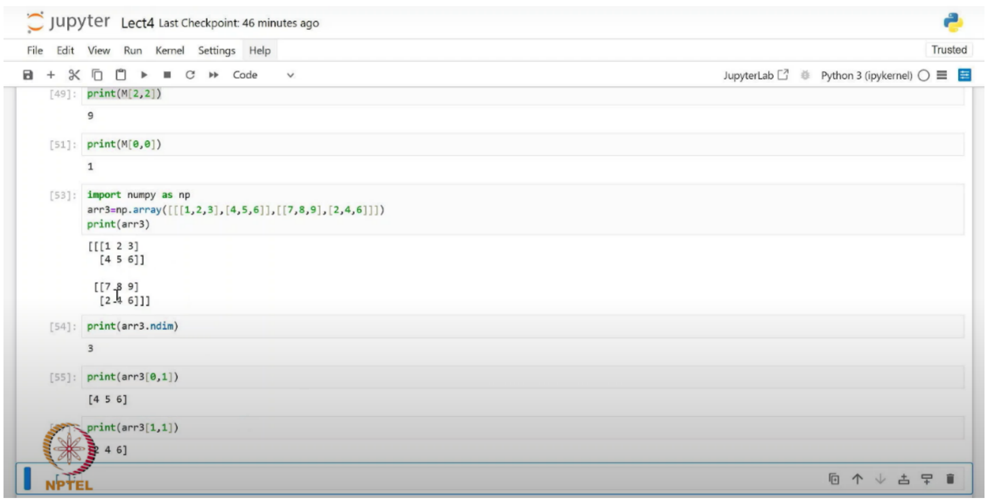
So here we have now defined it. Now see, what is there in this? What is the first element of this? This is a two-dimension array, a matrix. What is the second element? That is also a two-dimension array. Okay. So, these values are a two-dimension array.

Now, as we had defined here above—here we had a one-dimension array. See, a one-dimension array, a one-dimension array—from that, we have a two-dimension array. Now using this, I write it here and define it here. I have the array and its dimension is here. So now this dimension of this has become three-dimension.

Okay, in this, one element is this and another element is this. So, like this, we have a three-dimension array. We can define four dimensions. So we can call the values like this. Now suppose I call here print. Now I write arr3 and write its elements zero and one. Now let's see what happens. [4 5 6] has come.

Zero means—zero means this is its first row in which there are two elements , one this and other this. Like this I can write print and I defined one, one. Here I get [2 4 6]. The last element .

(Refer slide time: 44:23)


jupyter Lect4 Last Checkpoint: 46 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[49]: print(M[2,2])

      9

[51]: print(M[0,0])

      1

[53]: import numpy as np
      arr3=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[2,4,6]]])
      print(arr3)

      [[[1 2 3]
        [4 5 6]]

       [[7 8 9]
        [2 4 6]]]

[54]: print(arr3.ndim)

      3

[55]: print(arr3[0,1])

      [4 5 6]

      print(arr3[1,1])

      [2 4 6]
```

Okay. So now what we have is that array has been generated,that is a three-dimension array and we can call it. Let's see what will happen. Now I call in it. Print(arr3[0,0,0]). One is written. So, I called the zero element. Zero of all is zero row, zero column it is . So in three-dimension, I called the first element and get value 1. So if we transfer three arguments, we will get each elements. If we input two arguments, we will get rows here. Like this, we can call any element of our array that is defined. Now we have called it and we also get to know its dimension.

So now our next task is slicing the numpy array. What does slicing mean? Cutting it into slices. So what do we do for that? Now we have an array—array that is two-dimensional. For example, if we have a two-dimensional array, can we cut it and make it one-dimensional? Or can we cut and take out some of its elements? So, this is what we have to see. In this case, slicing means taking some elements from that array. So for this, what do we have to do for slicing? We will have to define it.

So first, suppose I defined import numpy as np. I wrote this. After that, I defined an array. So let me take one dimensional array and I will take it, okay. And I wrote np.array, and what did I do here? I defined it and I wrote [0, 3, 5, 8, 10]. I wrote it like this. So I have defined an array. And now I want to print it. Suppose I will print all the elements of the array that I have defined, from zero to 2. Do this once, okay. Now I write y, print 0:2 like this. I simply wrote this, so it came out 0 and 3. So its value came out: 0 and 3. Here I wrote 4 instead of 2, so look, what is this? Zero element, one element, two element, three element. Now what will happen in this? It is going from zero to four, so the last element will always be excluded. So, when we write this, I can write a comment here for myself:

What to do in slicing: always start, then write a colon. In the middle, you can define the end, colon, step—like this. Okay. And keep in mind that the end is excluded, and I am excluded. So, the value that we have is this. And now I say lets take from one to 4, so we get [3, 5, 8]. Why? Because the first indexing was zero. It is a zero value. It will go till the three, it will not take the fourth.

Now what I have to do is that if I check it in this and suppose I have defined an array here, named it A2, and I have given the value of the array a very large value: [0, 3, 5, 8, 10, 11, -3, -4, 20]. Okay, I wrote it like this. And now what did I do? I put a colon. I want to go from one to six. Okay. And take steps of two. So let's see what happens.

Now see, which value was at the first position? It was 3, so 3 came. And we have to go till six. So after 3 we wanted the gap of two. So after one, I wrote the third one—that came 8. It will go after the third one—the fifth one. So it basically came to the sixth position, so it will be excluded. And we had to take the gap of two.

So, for example, if I do not write six, I write it as 7 : or let us do this in steps of two. Or if I do this—the step of 3 comes. Okay. So we printed it.

Import numpy as np

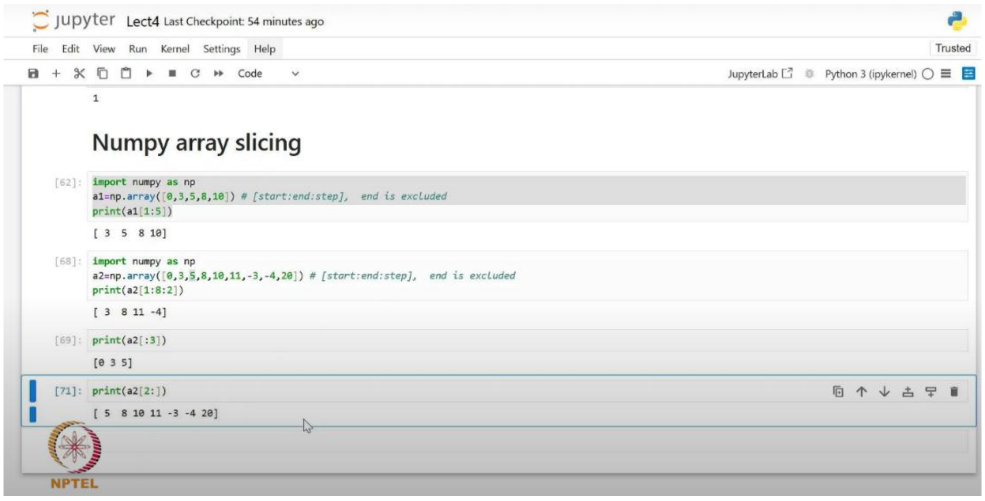
a2= np.array([0, 3, 5, 8, 10, 11, -3, -4, 20])

print(a2[1:7:2])

So now we had to go from one to seven. That is we had to go to -4 here and we had to go with a gap of two-two. So what did we write? First 3, then 8, then 11. Here the last one is excluded. The value of six will not come. I write it—let me write eight—so this value comes here. Okay.

So like this, we can do slicing with the command. We can print the numpy array. Now, like I did this—now if someone tells me that I want to print all of them—then I can write it like this: print(a2[0:3]). I want to print all the elements of a2. I want to print all the elements from the starting and I want to go till three. Okay, go till here. So the value that I got is [0, 3, 5]. Zero position, first, second, and third. Third is excluded. So it became [0, 3, 5]. So what I did was I wrote it like this. I want to print our array and start from the second value and go till the last. For this, I wrote here: print(a2[2:]). So what happened here is that the element that was at the second position was 5. From that till the last, all will be printed.

(Refer slide time: 52:39)


jupyter Lect4 Last Checkpoint: 54 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

1

Numpy array slicing

```
[62]: import numpy as np
      a1=np.array([0,3,5,8,10]) # [start:end:step],  end is excluded
      print(a1[1:5])

      [ 3  5  8 10]

[68]: import numpy as np
      a2=np.array([0,3,5,8,10,11,-3,-4,20]) # [start:end:step],  end is excluded
      print(a2[1:8:2])

      [ 3  8 11 -4]

[69]: print(a2[:3])

      [0 3 5]

[71]: print(a2[2:])

      [ 5  8 10 11 -3 -4 20]
```

So, like this, we can print any of the numpy arrays with the help of this one. Numpy module discussed related to it, we told why we are using numpy—because we are generating arrays. We are generating vectors and matrices. So, we used a few commands of that today, and we will discuss it further in the next lecture.

So, thank you for watching this lecture.