Scientific Computing Using Python

Professor. Vivek Aggarwal and Professor. Mani Mehra

Department of Mathematics

Indian Institute of Technology, Delhi

Lecture No. 05

Welcome to Scientific Computing Python. In the last lecture, we introduced the NumPy module. So today, we will look at other basic commands related to it. Let's get started.

In the last lecture, we did the same thing—we used the NumPy module and learned many of its basic commands. And here, we did the slicing of NumPy arrays. So, we defined an array, and after defining the array, we tried to print it—whether we want to print all its elements or print some elements—we saw that. So, we printed it.

Now I can also print it with a minus sign. Like this: I have defined array2 here. And what I am doing here is, I go from two to minus 1 and see if we get its print. So, it has come till the last. So, its value will go till the last.

Now my value is minus 1, and let's see what it means to print. I defined it: a2, and in this, I defined minus one. So, the value that came is 20. The value 20 went to the last. So, what does it mean?

The indexing of the array—if we go from left to right, it will start from zero. Zero to further values—such indexing will happen. If we go from right to left, in reverse, then what will happen there is that the last element will be considered -1, and the element next to it will be -2.

If I write here print(a2[-2]), I want to print its value. So -4 came after printing. Okay. So, the values that are coming in the last will be -1, the next one from it will be -2, -3, -4—such values will keep coming. So, this indexing is from behind. If we start from the end, then we will have to take it as minus 1.

So, in this way, we can define the indexing of any values—that is, the array. Now, if we want to do slicing of the array, then in slicing, suppose we take 2D array and I write:

import numpy as np

After that, I took an array, took any array a2, I defined it, and I have already taken any a2 in it. I take any x2 I wrote it:

x2= np.array([...])

Okay. So after this, I defined it. In that, I defined one element first[2,4,6]. Okay. Secondly, I defined:

[0,-2,4] .I defined this. Okay. So, this two-dimensional array I defined, I printed it:

print(x2)

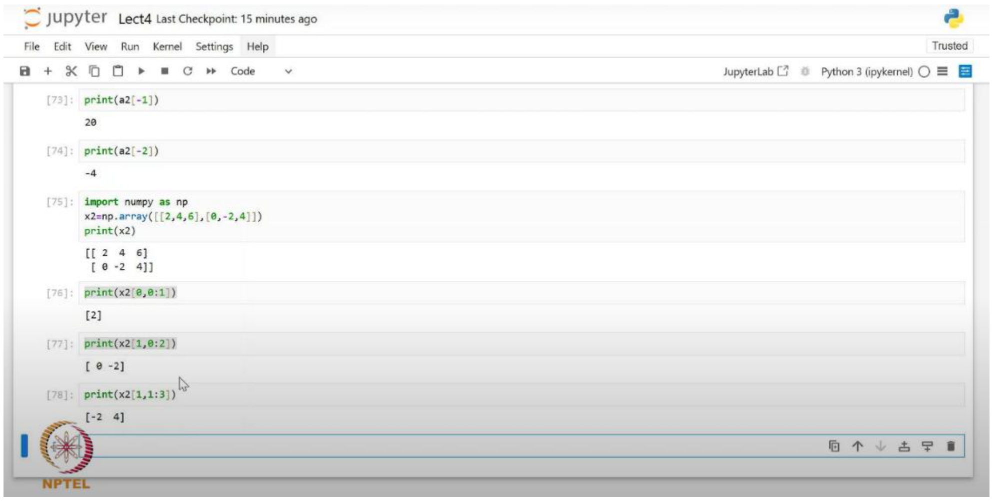
So, this is x2 defined. Okay. So now, when I see this, it gets printed. Now, what have I done? I want to print it. Okay. But I want that the printed value of which array—of the x2 array—should be printed from where to where?

So, I want that these values should be printed from row zero, and go from column zero to one. If I do this, then which value will come? This value will come [2] because the last is not excluded.

Now, I want to do the same thing. I have printed it. I copied it, and I put '1' in place of '0' here. Okay. And I wrote 1 here. So, what is the value? The value is [0, -2].

Like this, I did Ctrl+V, and I wrote this. I wrote : print(x2[1,1:3]).

(Refer slide time: 5:49)


Jupyter Lect4 Last Checkpoint: 15 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[73]: print(a2[-1])
      20
[74]: print(a2[-2])
      -4
[75]: import numpy as np
      x2=np.array([[2,4,6],[0,-2,4]])
      print(x2)
      [[ 2  4  6]
       [ 0 -2  4]]
[76]: print(x2[0,0:1])
      [2]
[77]: print(x2[1,0:2])
      [ 0 -2]
[78]: print(x2[1,1:3])
      [-2  4]
```

So, this value came: [-2, 4]. So, we can also do its slicing like this, and we can print it with the command. So, I printed it like this. I can also do this for it. I wrote Ctrl+V, and I defined the x2 from 1 to—or I do this—I write 0, I wrote 0, and after that, I did a colon and did 2. See, what did we do? A sub-matrix will be defined here whose elements are:

[[4 6], [-2 4]]

So now, what are we doing in this? I first defined the index from 0 to 2. So, 0 and 1 will come here. So, both 0 and 1 rows came. I defined the columns from 1 to 3. So, columns 1 and 2 are this and this. So, their values will be displayed. Okay?

What did we do in this? By slicing it, we got the 2×2 matrix, and we printed it here. So, in this way, we can define the matrices . Now, I had defined a matrix above M. Let's see. It was a 3×3 matrix that we had defined earlier. Now in that case, I take it and print it. Okay. So, I printed it, and I took the given array M. So, now in this array, we have three dimensions, It is of two dimensions, with three rows.

So, what do I do in it? I write this command:

print(M[0:2, 1:3])

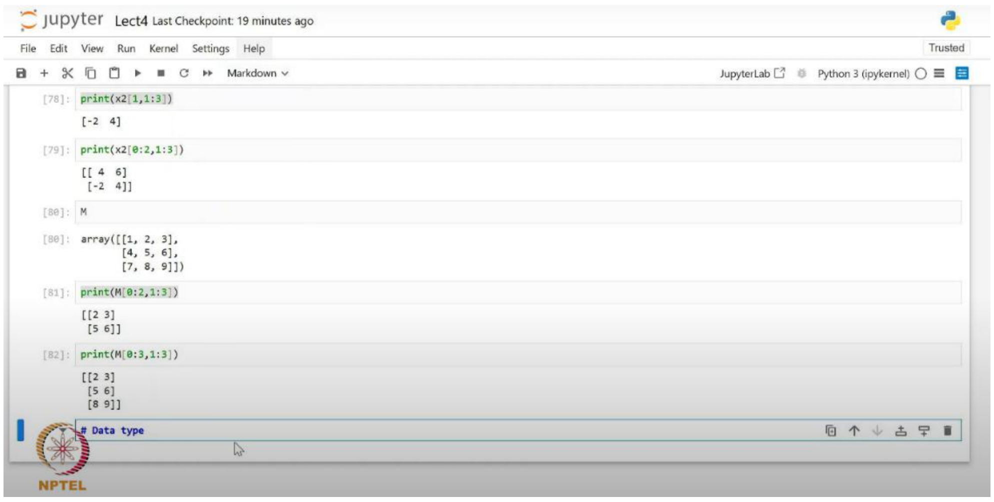
So, we took 0 and 1 row, so the 0th row was this, and the 1st row we had this, whose indexing is known. And the first column was this, and the second column was this.

So, what did we do? We got the [[2 3],[5 6]] sub-matrix printed here. Okay. I can change this, and I can control V and here, I can also write: print(M[0:3 , 1:3])

I printed it. So, look—this is the whole matrix that we had. If we leave out the first column, then all the values will come in it. So, we have sliced it like this. Right?

So we can do slicing in this way. Now I can print any row, use any column, or use any elements. I can use any sub-matrix. So, in this way, we can call the values of any array or any sub-matrix.

(Refer slide time: 9:34)


jupyter Lect4 Last Checkpoint: 19 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[78]: print(x2[1,1:3])
      [-2  4]

[79]: print(x2[0:2,1:3])
      [[ 4  6]
       [-2  4]]

[80]: M

[80]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

[81]: print(M[0:2,1:3])
      [[2 3]
       [5 6]]

[82]: print(M[0:3,1:3])
      [[2 3]
       [5 6]
       [8 9]]

      # Data type
```

Now, in the data that we have, we can also find out what type of data is there. What type of data have we printed in it? Now, like if I write, # Data type, we define the data type. This is how we have defined it. Like, we have used the first command to check the data type. So, in this case also, we can check by writing the data type. I write print here. Like I did, I had defined it above, and maybe I had defined it as an array a1. So I will print it, and I write: print(a1.dtype) to check the data type. I defined it and wrote it here. The data type is integer, and its value is written in 32 bits. So here we have defined the data type. 10:26

Similarly, if I take values and apply a string, then we can do it with strings also. Like till now, I have taken only numerical data. So I wrote:

import numpy as np

Now let's define an array which is in string form. So I wrote:

s = np.array(['Apple', 'Mango', 'Orange', 'Banana'])

We have defined an array and I printed it:

print(s)

So we can do it. If I check its data type, then I write:

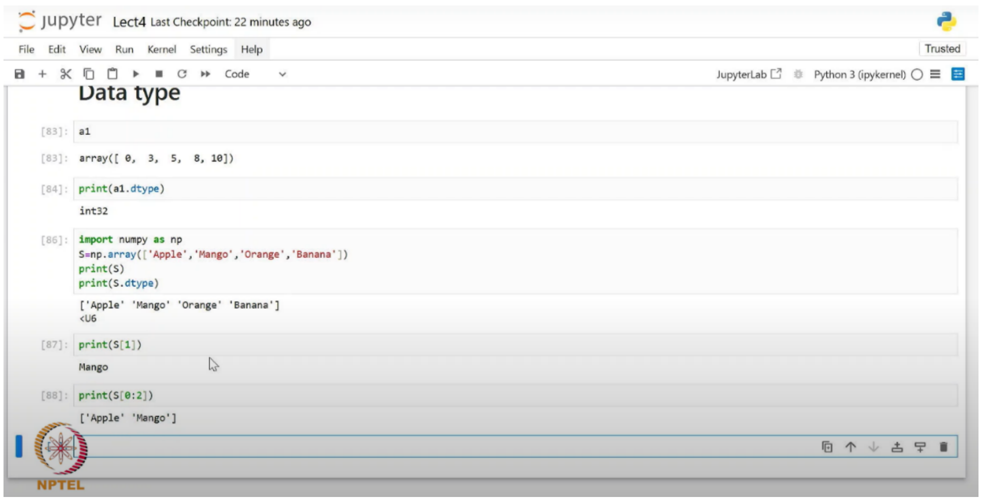
print(s.dtype)

We get this: a string of type '<U6'. I can also print it from here. So this string came, and we have its type.

So in this way, we can also define numerical values or string values in an array. These things are already defined by us. Now, in NumPy, we can define it like this. If I want to print a specific value, suppose I write s[1], then let's see what comes: "mango". So now, in this string array, the values are: "Apple" at 0 position, "Mango" at 1, "Orange" at 2, and "Banana" at 3. So I called the value at the first index, and it returned "mango".

Similarly, I printed s[0:2], and it printed [‘Apple’, "Mango".

(Refer slide time: 13:18)


jupyter Lect4 Last Checkpoint: 22 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

Data type

```
[83]: a1
[83]: array([ 0,  3,  5,  8, 10])
[84]: print(a1.dtype)
      int32
[86]: import numpy as np
      S=np.array(['Apple','Mango','Orange','Banana'])
      print(S)
      print(S.dtype)
      ['Apple' 'Mango' 'Orange' 'Banana']
      <U6
[87]: print(S[1])
      Mango
[88]: print(S[0:2])
      ['Apple' 'Mango']
```

If I want to access the last value using a negative index, I write s[-1], which gives us "banana". So we have to specify the array from which we want to see the value. Using such commands, we can easily call values from arrays, whether they are numerical or string type.

Now, what do we do next? Let's see some more commands. This is about NumPy Array Copy and View. Let's define how we can copy an array and how we can view it. 14:19

We have some data:

import numpy as np

a1 = np.array([9, 3, 5, 7, 10])

x = a1.copy()

x[-1] = 20

Now what happened? The a1 array got copied into x, and then I changed the last value of x to 20.

print(a1)

print(x)

First, we defined an array a1. Then we created a new array x using the copy command. The last value of x was changed to 20. Now we have to see whether this change affects a1 or not. The output will be:

a1: [9 3 5 7 10]

x: [9 3 5 7 20]

So, a1 remains unchanged. In this case, x is a copy, and changes to x do not affect a1. In this case we copy an array in another array and the other array we can change it will not affect the original array.

Now let’s try the view method:

a2 = np.array([0, 3, 5, -7, 12])

x1 = a2.view()

x1[-1] = 100

print(a2)

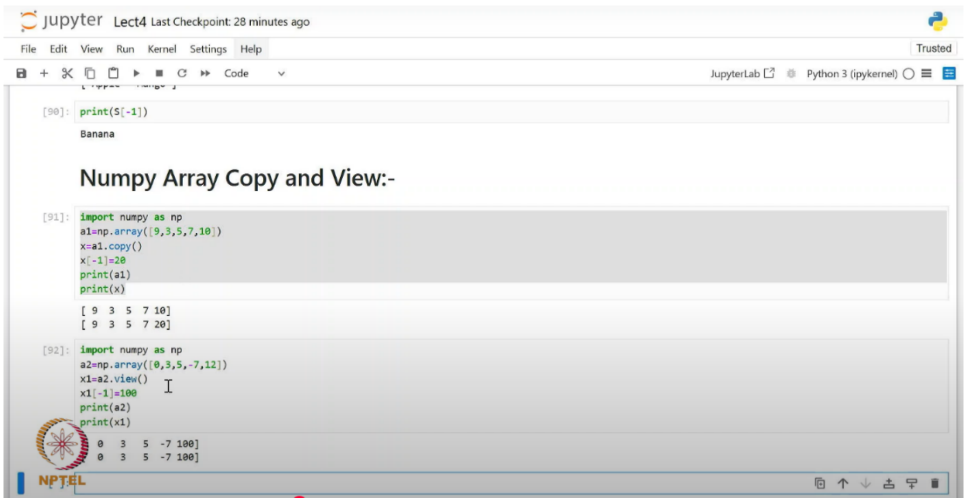
print(x1)

We defined an array a2 and created a view x1. Then we changed the last value of x1 to 100. In this case, both a2 and x1 will reflect the same change:

a2: [0 3 5 -7 100]

x1: [0 3 5 -7 100]

(Refer slide time: 18:49)


jupyter Lect4 Last Checkpoint: 28 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[90]: print(S[-1])

Banana
```

Numpy Array Copy and View:-

```
[91]: import numpy as np
      a1=np.array([9,3,5,7,10])
      x=a1.copy()
      x[-1]=20
      print(a1)
      print(x)

      [ 9  3  5  7 10]
      [ 9  3  5  7 20]

[92]: import numpy as np
      a2=np.array([0,3,5,-7,12])
      x1=a2.view()
      x1[-1]=100
      print(a2)
      print(x1)

      [ 0   3   5  -7 100]
      [ 0   3   5  -7 100]
```

So here, using view, we view x1 in a2 and the changes that we made here with view, both arrays share the same data. We copied it in x1 and make its last value as 100. Now a2 and x1 both value become 100. In copy it was just coping but in view both become same. Changes made in the view reflect in the original array.

Now let’s explore further and define x2 in place of x1 and see what happen:

x2 = x1

x2[-1] = 100

print(a2)

print(x1)

print(x2)

I copied a2 as view in x1 , then I kept equal to x2 and define its last value 100. This shows that all three – a2, x1, and x2, share the same data. All changes made to one reflect in the others. This shows the difference between copy and view in NumPy.

In copy, the data is duplicated and changes do not affect the original. In view, both the original and the new variable share the same data. So, this is how copy and view work in NumPy.

Now the next thing we have is whether we can change the shape of the array. So what we have to do now is # NumPy array shape. We will try to change the shape of the array. First, we will define it, then we can also do reshaping.

So let's change the shape. Now let's see—we want to see the shape of the array. So I write:

import numpy as np

Now what have we done? Let's define a vector. I will define an array:

arr1 = np.array([9, 3, 5, 6, 7])

We have an array. Now we do:

print(arr1)

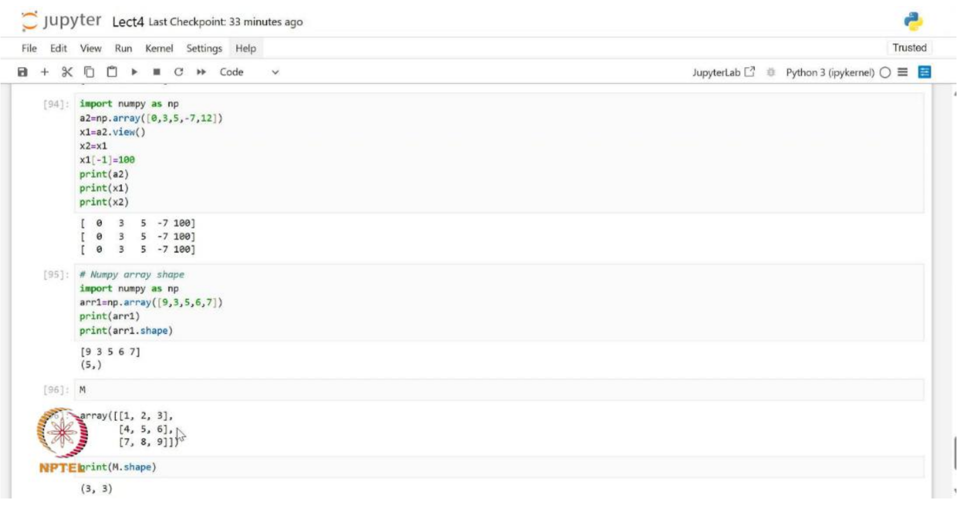
print(arr1.shape)

So I just defined an array, printed it, and wrote its shape. It tells us the array is [9 3 5 6 7] , and number of elements are 5. So, it tells us that it is a one-dimensional array.

We had defined M earlier. Suppose we define it like:

print(M.shape)

(Refer slide time: 22:36)


jupyter Lect4 Last Checkpoint: 33 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[94]: import numpy as np
      a2=np.array([0,3,5,-7,12])
      x1=a2.view()
      x2=x1
      x1[-1]=100
      print(a2)
      print(x1)
      print(x2)

      [  0   3   5  -7 100]
      [  0   3   5  -7 100]
      [  0   3   5  -7 100]

[95]: # Numpy array shape
      import numpy as np
      arr1=np.array([9,3,5,6,7])
      print(arr1)
      print(arr1.shape)

      [9 3 5 6 7]
      (5,)

[96]: M

      array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

      print(M.shape)

      (3, 3)
```

Here, it's written (3, 3) . It means 3 array and each array has three elements. So it's a one-dimensional array of shape (3, 3). Like this it define the shape, It tells us what shape our array is, what is its shape?

So now, like I wrote, it is a one-dimension array. So we have shown that inside it, there are five elements, and it is one dimension. Okay, so similarly, we have this array, and inside it, there are three elements, and in all three, there are three elements. Like this, we have also defined a two-dimension array above. We have an array. Let's see what this is.

The arr3 is still in our Python memory, which is defined. So what do we do now? I wrote print(arr3) and then defined the shape. So it appears as (2, 2, 3). Now see, how many two-dimension arrays are there in this? One is here. Okay, and what are the elements in it?

Now see, if we look at this, if we leave out the first two, then the third will be the same. So what is there in this? That is a two-dimension array. Meaning, a two-dimension array which is a three-dimension array. Okay, what is there in it? Two is this array and one is this array. What is inside it? The values that we have are three values inside it. So the array which is showing like this had 3 by 3 in it that 2 by 3. So, in this way, our shape is defined.

Now we will reshape. So, what is reshape? Check this. We have to reshape an array. Now let's see what this is. Shift and enter # Reshape of an array.

So now we have to see whether we can reshape the array as well. Can we reshape the array? Okay, so what will we do now? I will create a new array.

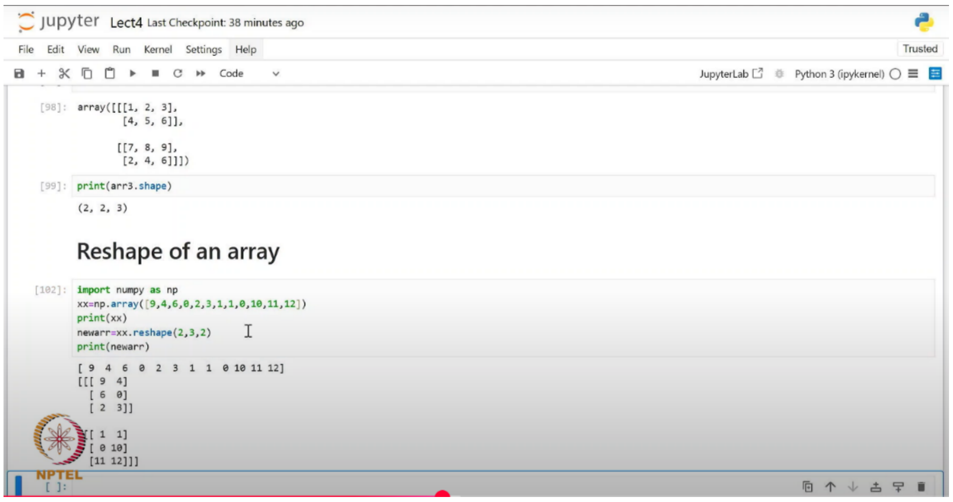
import numpy as np. I wrote this. Okay, after that, I defined an array. I named it

xx = np.array([9,4,6,0,2,3,1,1,0]). So,, I defined this array and printed it. It came out.

Now what do I do? I try to reshape it. So, I write: newarr = xx.reshape(2,3,2), okay?

In which form — 2,3,and 2 should be done in this form. Now let's see what will happen. So its dimensions are not complete. Okay. So we have to increase the dimension of this. Let's see how much dimension it has. So, we have nine. Let's increase it. Let's do this 10, 11, 12. Now it is written as is. Okay, so what have we done? We have to reshape it. Now we get it printed as well — print(newarr). So, it is written.

(Refer slide time: 28:13)


jupyter Lect4 Last Checkpoint: 38 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[98]: array([[[1, 2, 3],
               [4, 5, 6]],

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

[99]: print(arr3.shape)

      (2, 2, 3)
```

Reshape of an array

```
[102]: import numpy as np
       xx=np.array([9,4,6,0,2,3,1,1,0,10,11,12])
       print(xx)
       newarr=xx.reshape(2,3,2)
       print(newarr)

       [ 9  4  6  0  2  3  1  1  0 10 11 12]
       [[[ 9  4]
         [ 6  0]
         [ 2  3]]

        [[ 1  1]
         [ 0 10]
         [11 12]]]

[ ]:
```

Now what do we have? The output dimension of this that we have is an array. So what will we do in this — that three arrays will be defined in it, and two elements will be defined in each. So this is what we have. Okay, so two-dimension array. Each one will have three elements. There will be three arrays, and in that, there will be two elements.

So here we have one this element and one this element. So it becomes two. Okay, so there will be two. Now what do we do? Inside it, there are three arrays defined. These three are three rows, and inside it, there are two columns — meaning two values defined in each.

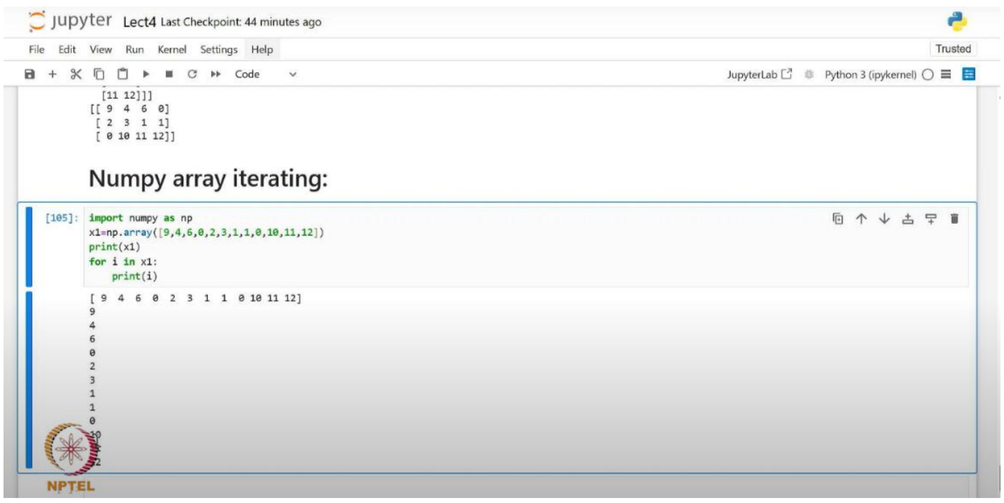
So if we define this, then it means that we have reshaped it. And reshape has to be done only then. First, we have to check how many elements we should have. So if we reshape, then its elements will be defined, like the last time we did it was error defined in that. So we use reshape very often. We have to reshape our vector and take it to some other form. So we bring it in this form.

I change it separately and write newarr1 = xx.reshape(3,4).

So, it is written. And I print it: print(newarr1). I wrote this. It has come.

So, what happens now? I have defined in newarr1 a matrix which has three rows and four columns. So, this matrix has come to us. Look, we have got three rows and four elements. Means what is happening in this is that we have got three one-dimension arrays, and each array has four elements. We can also say this. So, what happened to us is that we reshaped all the elements of the array and brought them to this form. And how the elements are coming — see 9, 4, 6, 0 came here first. Then 2, 3, 1, 1 came in the second row, and 0, 10, 11, 12 came in the third row. So, in this way, we distribute it in what we wanted to reshape the dimension of our array. Okay, so in this way, we can use reshape.

(Refer slide time: 33:40)


jupyter Lect4 Last Checkpoint: 44 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
  [11 12]]]
[[ 9  4  6  0]
 [ 2  3  1  1]
 [ 0 10 11 12]]
```

Numpy array iterating:

```
[105]: import numpy as np
       x1=np.array([9,4,6,0,2,3,1,1,0,10,11,12])
       print(x1)
       for i in x1:
           print(i)
```

```
[ 9  4  6  0  2  3  1  1  0 10 11 12]
9
4
6
0
2
3
1
1
0
10
11
12
```

Now we see one more thing — that is NumPy array iterating. Okay. Iterating means the iteration that we do — going from one to another, going from second to third — one by one element. We call it iteration. Okay. So, we want to iterate in this. So how can we do it? If we have an array, and we want to iterate something inside it, then how can we use it? So we see that. So, I marked it down and pressed Shift + Enter. So here I wrote import numpy as np. Now what do I do? I take an array. Let's take xx. So in place of xx, I write x1. I have defined it. This is an array. Okay, so we printed it, right?

Now I want to print it in a column instead of printing it here. See, if we print this, and after this, if I write it like this — I apply a for loop. Then I write for i in x1: print(i). Now let's see what happens.

See, first I printed, then a row got printed, and the whole element of the array will get printed. But after that, I applied iteration, then I applied a for loop in it. So, I wrote for i in x1: — meaning there is some indexing. It will go to x1 and will print it. So, what did it do? First it printed 9, then 4, then 6. So, it printed a column vector. This is what we call iteration — that it printed it in iterating form.

So, in the same way, what we have — vectors — even one-dimension vectors — we can print them like this. We can even do two dimensions. What do I have to do for two dimensions? Now I write import numpy as np. I wrote this. Okay. After that, I define a vector np.array(...). And what do I do in it? I define a two-dimension array. Okay. So, in this I wrote [2, 3, 5]. I took this. I took one element. And the second element — I took [3, 6, 8], the second element.

So, from here, I defined this. I created a two-dimensional array — a two-dimensional array is such that every element of which is one-dimensional. So, a one-dimensional array becomes this — [2, 3, 5]. The second element — [3, 6, 8]. So, we have defined a two-dimensional array.

Now what do I do? I define for i in x2:. I get it printed. The answer came. So, this came [1, 2, 3] and [3, 6, 8]. So, what does it mean? This first went into that and then went into x2. So, it got the first element, and it printed it. Then it iterated again and went to the second element and printed that.

So, what we did in this was that it will get printed twice. Its iteration will be twice. So, we iterated it twice, and its values got printed twice and came to us. So, in the same way, we did this. We can define it.

So in this way, we can print it. Now, like we have this two-dimension array. So, I want to print its element one-by-one in such a way that first one is written, then two is written, then 3 comes. We want to print like this. So, what will I do? In this, I define a for loop. Then copy this — Ctrl + C, Ctrl + V. Okay. I defined x2. Now I wrote for i in x2: . Now I write for j in i: . Okay. And print(j) written like this. Now see, i will go to x2, and j will go to i, and after that it will be printed. So, this value is: 1, 2, 3, 3, 6, 8. So, we have printed the element one by one.

Okay, so what does it mean? The value of i will go till 2, and after that, it will print the elements inside it. So, what will it do? First, it goes to the first index of i. It goes inside it and sees that there are 1 to 3 elements, so it prints those three elements. After that, the value gets changed in the value of x2. It goes to the second values and then prints them. So here, each element gets printed.

So, with the help of for, we can print all of it. Okay. So, like this, we can use some more commands. We can join the two arrays.

So, we can also do # Joining two arrays. How to do this? Now we have two arrays. Let me write two arrays.

import numpy as np

I wrote this. So I defined arr1:

x1 = np.array([2, 4, 6])

Like this, I defined arr2:

x2 = np.array([-2, 4, -3])

So, we have these two arrays defined. Now what we have to do is to join them. In English, we call it “concatenation.” That is where the first one ends, the second one is joined with it — joined along the same axis. So, we have to check this.

I defined a new array:

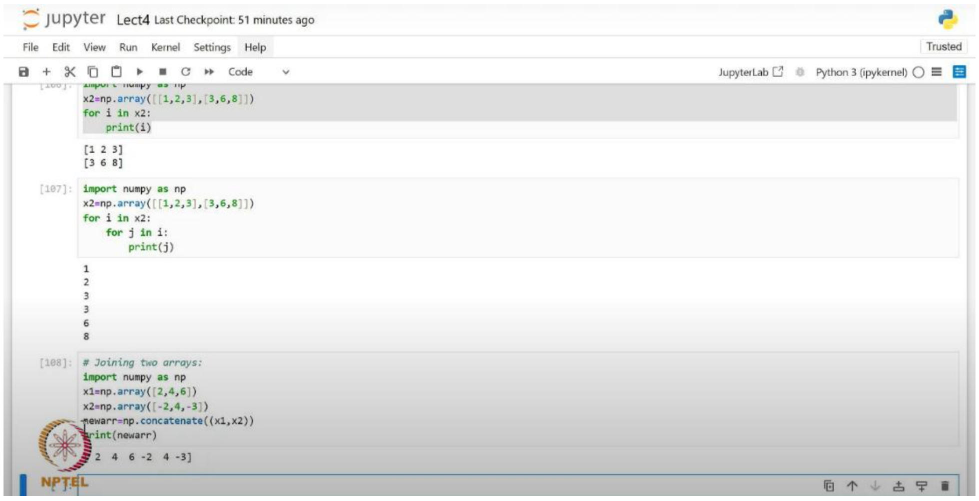
newarr = np.concatenate((x1, x2))

print(newarr)

Now lets see what happen with the array I defined just now? The new array that has becomes:

[2 4 6 -2 4 -3]

(Refer slide time: 40:55)



```
import numpy as np
x2=np.array([[1,2,3],[3,6,8]])
for i in x2:
    print(i)
```

```
[1 2 3]
[3 6 8]
```

[107]:
```
import numpy as np
x2=np.array([[1,2,3],[3,6,8]])
for i in x2:
    for j in i:
        print(j)
```

```
1
2
3
3
6
8
```

[108]:
```
# Joining two arrays:
import numpy as np
x1=np.array([2,4,6])
x2=np.array([-2,4,-3])
newarr=np.concatenate((x1,x2))
print(newarr)
```

```
[ 2  4  6 -2  4 -3]
```

Like this. Now, if I write x2 before x1, then what happens? This time, x2 comes first, and x1 comes after. Why? Because this is along the axis. So, this is a one-dimensional array. So, the first one mentioned will be printed first, then the second one. Now both of them are connected — they are joined. So, we can join them like this.

Now let’s take a two-dimensional array. If we have two arrays like:

x1 = np.array([[2, 4],[1,3]])

x2 = np.array([[-2, 4],[-6,-4]])

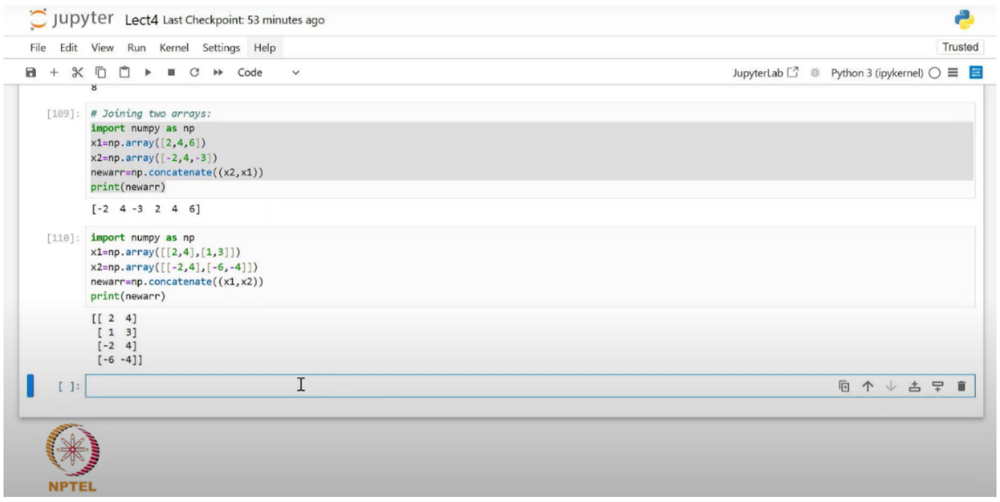
Now, if I join both of them:

newarr = np.concatenate((x1,x2))

print(newarr)

Then x1 is the first array (a matrix), and x2 is the second array (also a matrix). After joining along axis (rows), we get both arrays joined together.

(Refer slide time: 43:10)


jupyter Lect4 Last Checkpoint: 53 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[109]: # Joining two arrays:
       import numpy as np
       x1=np.array([2,4,6])
       x2=np.array([-2,4,-3])
       newarr=np.concatenate((x2,x1))
       print(newarr)

       [-2  4 -3  2  4  6]

[110]: import numpy as np
       x1=np.array([[2,4],[1,3]])
       x2=np.array([[-2,4],[-6,-4]])
       newarr=np.concatenate((x1,x2))
       print(newarr)

       [[ 2  4]
        [ 1  3]
        [-2  4]
        [-6 -4]]

[ ]:
```

There is another command called stacking. Stacking can be done in two ways.

Let's say we want to do along rows. Now let's define two arrays like as above but in place of concatenate, we will write hstack like this:

import numpy as np

x1 = np.array([2, 4, 6])

x2 = np.array([-2, 4, -3])

newarr = np.hstack((x1, x2))

print(newarr)

This gives:

[2 4 6 -2 4 -3]

So, what it has done is that it has defined the rows horizontally and stack it along the — horizontally. So, staking is done in this way.

Now let's try if we can do vertical stacking. So, define vstack like that.

Neaaa1 = np.vstack(x1))

print(newarr)

print(neaaa1)

This gives:

[[2 4 6 -2 4 -3]

 [[2]

  [4]

[6]]

So now the array is two-dimensional . vstack is vertical. So, what happen in vertical, its dimension changes. So, what we did: x1 we had a vector which was one dimensional we have stacked it vertically, so what happened in vertical, the first element became two, then the second element became four and then the element became six, like this we have It becomes like a matrix.

Now let’s explore one more command that is # Numpy splitting. So, in this we have to see whether the numpy can be split or not, so we will try to split this thing with the help of the array underscore. Suppose we have:

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

We want to split this into 3 equal parts:

newarr = np.array_split(arr, 3)

It will split the array that we have in three equal parts and after that I will print this.

print(newarr)

This returns:

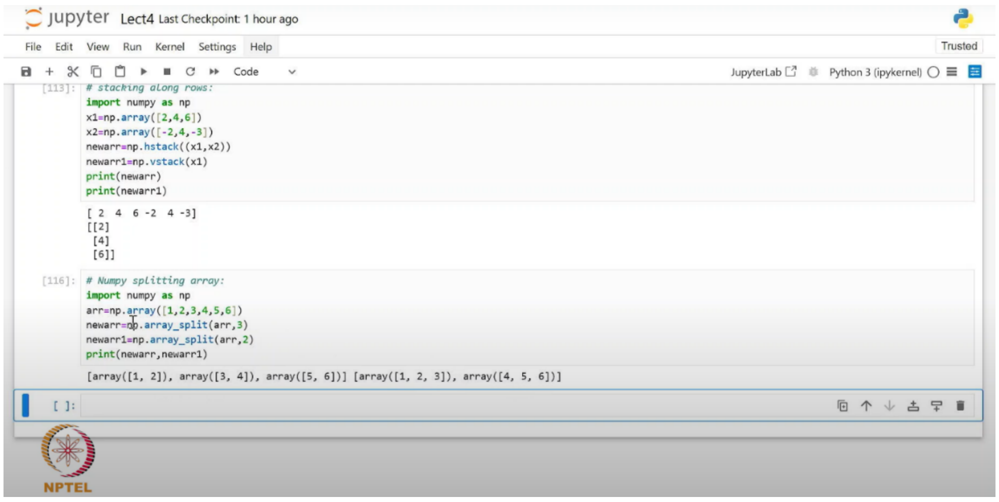
[array([1, 2]), array([3, 4]), array([5, 6])]

Similarly, to split into 2 parts:

newarr1 = np.array_split(arr, 2)

print(newarr,newarr1)

(Refer slide time: 50:20)


jupyter Lect4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[113]: # stacking along rows:
import numpy as np
x1=np.array([2,4,6])
x2=np.array([-2,4,-3])
newarr=np.hstack((x1,x2))
newarr1=np.vstack(x1)
print(newarr)
print(newarr1)

[ 2  4  6 -2  4 -3]
[[2]
 [4]
 [6]]

[116]: # Numpy splitting array:
import numpy as np
arr=np.array([1,2,3,4,5,6])
newarr=np.array_split(arr,3)
newarr1=np.array_split(arr,2)
print(newarr,newarr1)

[array([1, 2]), array([3, 4]), array([5, 6])] [array([1, 2, 3]), array([4, 5, 6])]

[ ]:
```

So, what happened here, I divided it in three parts, here it split it into two parts. We can also print the arrays together like this. So, numpy is quite useful. So, we can use np.array_split() to split arrays into equal or nearly equal parts.

Now let’s see how to search for an element in an array. So, the command for this is where. Let’s see how we can do . So, we have defined earlier we can copy that and have:

```
import numpy as np
```

```
arr = np.array([1, 2, 3,  4, 5, 6])
```

In search what do I have to do in search is that, we get a where of array that where is the element. So, let’s write:

```
x = np.where(arr == 3)
```

```
print(x)
```

This gives:

```
(array([2]), dtype=int64),)
```

It means position of x is 2 and the data type is integer is of 64bit. So, using where we get its value, at which position it is and what is its data type. Like this we can do for any array we have . Let change the previous one and take:

```
import numpy as np
```

```
arr = np.array([1, 2, 3,  4, 5, 6, 7, -2, -4, -5])
```

```
x = np.where(arr%2 == 0)
```

What does this mean, I divided the entire array by two and I get those element whose remainder is zero, what does it mean, I get only the even values which are in this.

This gives:

```
(array([1, 3, 5, 7, 8], dtype=int64),)
```

So, the positions are 1, we have 2 at one position then at third position we had 4, after that at fifth position 6 likewise we get. These are the indices of even numbers (2, 4, 6, -2, -4).

Similarly,I can define for odd numbers:

```
x = np.where(arr % 2 == 1)
```

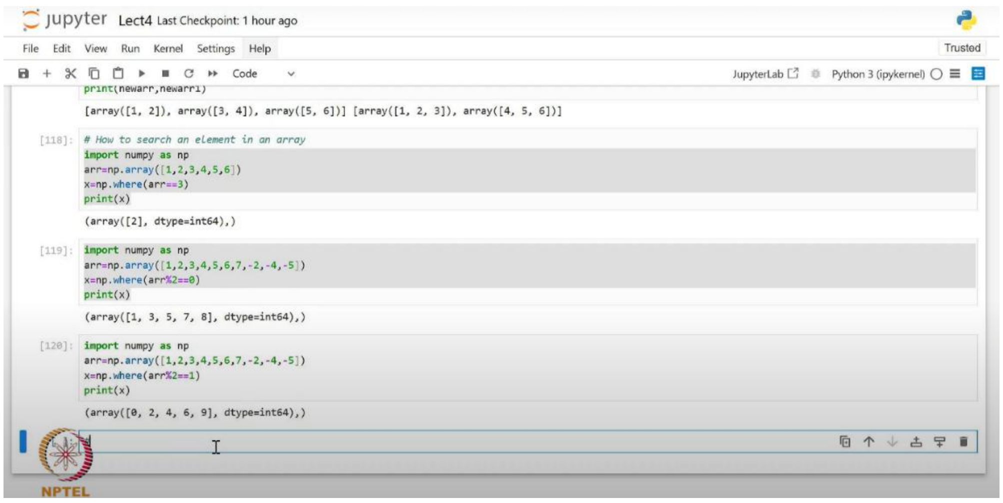
```
print(x)
```

This gives:

```
(array([0, 2, 4, 6, 9], dtype=int64),)
```

Which are the indices of odd numbers (1, 3, 5, 7, -5). Like this we can do for odd and for even number, we can do sorting.

(Refer slide time: 55:48)


jupyter Lect4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
print(newarr,newarr1)
[array([1, 2]), array([3, 4]), array([5, 6])] [array([1, 2, 3]), array([4, 5, 6])]

[118]: # How to search an element in an array
import numpy as np
arr=np.array([1,2,3,4,5,6])
x=np.where(arr==3)
print(x)
(array([2], dtype=int64),)

[119]: import numpy as np
arr=np.array([1,2,3,4,5,6,7,-2,-4,-5])
x=np.where(arr%2==0)
print(x)
(array([1, 3, 5, 7, 8], dtype=int64),)

[120]: import numpy as np
arr=np.array([1,2,3,4,5,6,7,-2,-4,-5])
x=np.where(arr%2==1)
print(x)
(array([0, 2, 4, 6, 9], dtype=int64),)
```

Sorting means to sort. We call it that we have to search in the array and it will return us the index of where that specific value should be inserted to maintain the sorted order. Let’s take a example of sorting also to understand it. Let us define an array.

arr = np.array([1, 2, 3, 4, 7])

x = np.searchsorted(arr, 5)

print(x)

In this, I have defined an array and I said to tell me where the number 7 is.

This gives: 4

This means that in 4th position we have the value 7. So, it gave us the answer 7 that is x=4. So, it gave us its position. Now, like in this we have four now I write it like this:

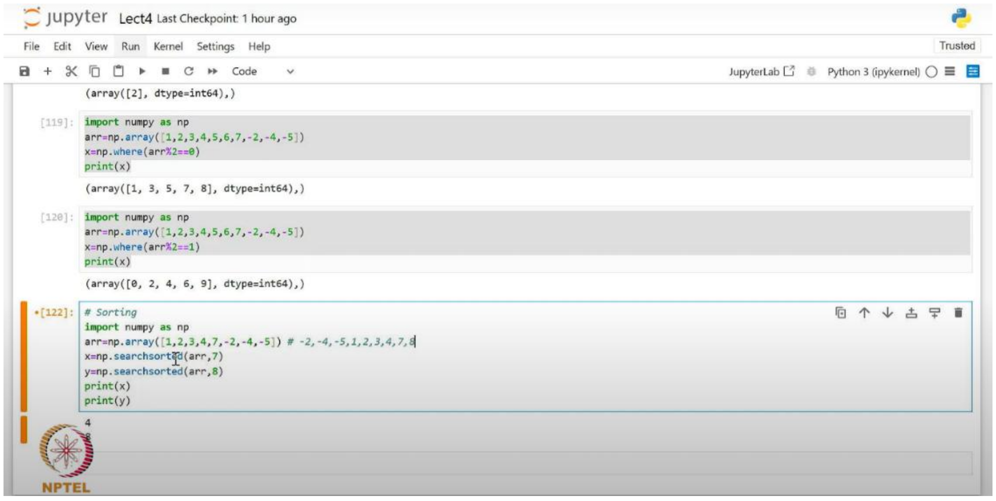
arr = np.array([1, 2, 3, 4, 7, -2, -4, -5])

y = np.searchsorted(arr,8)

print(y)

This gives: 8

(Refer slide time: 58:27)


jupyter Lect4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
(array([2], dtype=int64),)

[119]: import numpy as np
arr=np.array([1,2,3,4,5,6,7,-2,-4,-5])
x=np.where(arr%2==0)
print(x)
(array([1, 3, 5, 7, 8], dtype=int64),)

[120]: import numpy as np
arr=np.array([1,2,3,4,5,6,7,-2,-4,-5])
x=np.where(arr%2==1)
print(x)
(array([0, 2, 4, 6, 9], dtype=int64),)

•[122]: # Sorting
import numpy as np
arr=np.array([1,2,3,4,7,-2,-4,-5]) # -2,-4,-5,1,2,3,4,7,8
x=np.searchsorted(arr,7)
y=np.searchsorted(arr,8)
print(x)
print(y)
4
8
```

This gives us the position of 8 as 8. How many elements are there in it? There are eight elements. Okay? So, it a see its position, it will go from zero to seven. So, how is this value showing eight. It is showing 8 because if we take this array and sort it, sort means increasing, or decreasing or put in a sequence. So, it will be like [-5, -4, -2, 1, 2, 3, 4, 7], so if we insert 8 it will be at last only that means its position is 8. In earlier case it was 7 means it was at 4th position so its answer came to be four. Sorted means we have first sort it and then it was searched. Here we will call this command like this that searchsorted. So, with the help of this we have checked the position of 7 as well as 8.

So today we learned many basic and useful NumPy commands.

Thank you very much for watching this lecture.

Namashkar