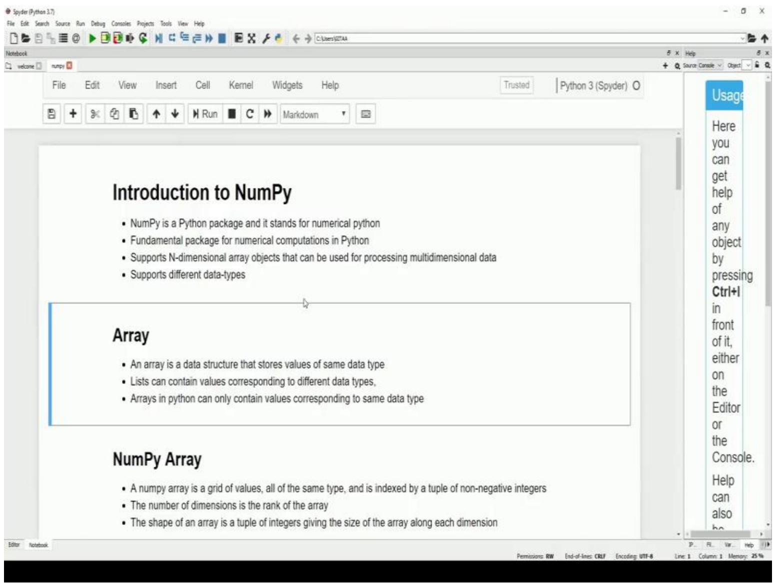
Python for Data Science

Prof. Ragunathan Rengasamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 12
NumPy

(Refer Slide Time: 00:15)



Introduction to NumPy

- NumPy is a Python package and it stands for numerical python
- Fundamental package for numerical computations in Python
- Supports N-dimensional array objects that can be used for processing multidimensional data
- Supports different data-types

Array

- An array is a data structure that stores values of same data type
- Lists can contain values corresponding to different data types,
- Arrays in python can only contain values corresponding to same data type

NumPy Array

- A numpy array is a grid of values, all of the same type, and is indexed by a tuple of non-negative integers
- The number of dimensions is the rank of the array
- The shape of an array is a tuple of integers giving the size of the array along each dimension

Welcome to the lecture on the NumPy. In this lecture we are going to look at what number libraries about and we will be seeing how we can use the number library to do some numerical operations on it. So, that it will come in handy in our upcoming lectures. Let me just introduce you to the numpy library basically numpy is a Python package and it stands for numerical Python it is a fundamental package for numerical computations in Python.

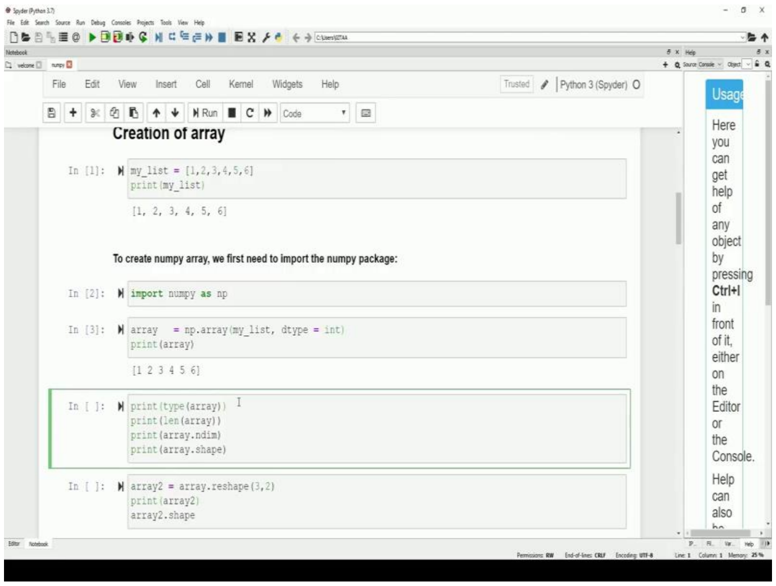
So, if you want to perform any numerical operations then numpy library will be used and it supports n dimensional array objects that can be used for processing multi-dimensional data and it supports different data types. We have already seen Python arrays in our previous lectures to recall what array is? An array is a data structure that stores values of same data type. So, if you are creating an array then all the elements in the array should be of same data-type.

We have also seen another container called list which can contain values corresponding to different data types. But arrays in Python can only contain values corresponding to the same data type. So, we have already been introduced to what Python arrays are. In this lecture we can see what are the advantages of using numpy array? A numpy arrays a grid of values all of the same data type and is indexed by a tuple of non-negative integers.

And the numpy is one of the core libraries for scientific computing in Python and it also provides a high-performance multi-dimensional array object and using numpy mathematical and logical operations on arrays can be performed. So, that is why we prefer using the numpy arrays over built-in arrays in Python. So, in Python the number of dimensions of the array is called as the rank of the array.

And a tuple of integers giving the size of the array along each dimension is known as shape of the array. So, once you have an array with you if you want to look at the shape of it then it is going to give you a pupil of integers giving the size of the array along each dimension. As we know that numpy provides a high-performance multi-dimensional array object. And elements in numpy arrays are accessed using square brackets like the other erase and it can be initialized by using the nested Python list.

(Refer Slide Time: 02:43)



Creation of array

```
In [1]:  my_list = [1,2,3,4,5,6]
         print(my_list)

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

To create numpy array, we first need to import the numpy package:

```
In [2]:  import numpy as np

In [3]:  array   = np.array(my_list, dtype = int)
         print(array)

         [1 2 3 4 5 6]

In [ ]:  print(type(array))
         print(len(array))
         print(array.ndim)
         print(array.shape)

In [ ]:  array2 = array.reshape(3,2)
         print(array2)
         array2.shape
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also

But there are several ways to create arrays for example you can create an array from a regular Python list or tuple using the array function. I am creating a list which contains the values ranging from 1 to 6 and the name of the list is my list and you can see the output that the values are enclosed inside the square brackets. So, to create a numpy array we first need to import the numpy package. Before even in putting the numpy package you should have installed the numpy library in your machine in order to import it in your spider or in any of the IDE’s. So, let us see how to install the numpy package.

(Video Start Time: 03:28)

So, you must install numpy in order to use it. So, you can install numpy by going to the terminal or command prompt and typing the pip install numpy. So, this is how we installed the numpy library or any of the libraries that are required but I am sure you would have already installed the numpy library in order to use it. This is just a demo on how to install the library since I already have numpy in the machine it says requirement already satisfied. So, let us just go back to the IDE and start working on it.

(Video End Time: 04:09)

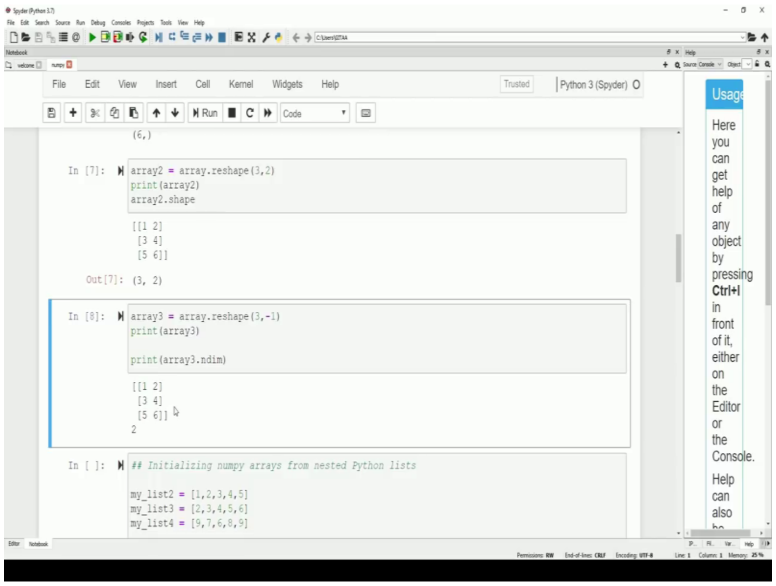
So, I am importing the numpy library as np. So, np is just analyzed to the numpy library. So, whenever I am going to use any function or access any function from the numpy library using a dot operator then I am just going to refer it as np not as numpy. So, once importing you can call the numpy dot array function. Basically array is one of the function that is used to create arrays and numpy and I am using a dot operator to access that function from the numpy library.

Since I have imported it as np I am just using np dot array and the basic n dimensional array is created using an array function in numpy as follows. I am using the array function from numpy library and every numpy array is a grid of elements of the same type but numpy provides a large set of numeric data types that you can use to construct arrays. So, numpy tries to get a data type when you create an array but the function construct erased usually also include an optional argument to explicitly specify the data type.

So, if you see here the first argument to the array function is my list which has the list of values from 1 to 6 and the next argument is dtype is equal to integers which specify that I am going to create an array with the data type integer. If you are not providing anything it will automatically guess a data type when you create an array itself but if you want to explicitly specify the data type then this is how you should be doing it.

So, I am creating an array using the array function from numpy and I am just printing it. So, the array is created now.

(Refer Slide Time: 05:54)



```
(6,)

In [7]:  array2 = array.reshape(3,2)
         print(array2)
         array2.shape

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

Out[7]:  (3, 2)

In [8]:  array3 = array.reshape(3,-1)
         print(array3)

         print(array3.ndim)

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

In [ ]:  ## Initializing numpy arrays from nested Python lists

         my_list2 = [1,2,3,4,5]
         my_list3 = [2,3,4,5,6]
         my_list4 = [9,7,6,8,9]
```

Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also

So, now I am printing type of array. So, this basically gives you though the data type of the object if you printed the output is class numpy dot nd array because numpy's main object is the homogeneous multi-dimensional array it will give you the type of array as numpy dot nd array irrespective of the dimensions of your array. The next thing is len of array which is going to give you the total number of elements in the array.

So, the total number of elements in the array is 6 and you can also get the number of axis of the array by giving ndim which stands for number of dimensions of the array that it says one because you have only one dimension to it that it means that you have only one access to your array. So, now let me just check dimension of the array. So, dimension of the array is going to give you an output of tuple of integers indicating the size of the array in each dimension. We know the dimension of the array is 1 and let us just check the size along each dimension.

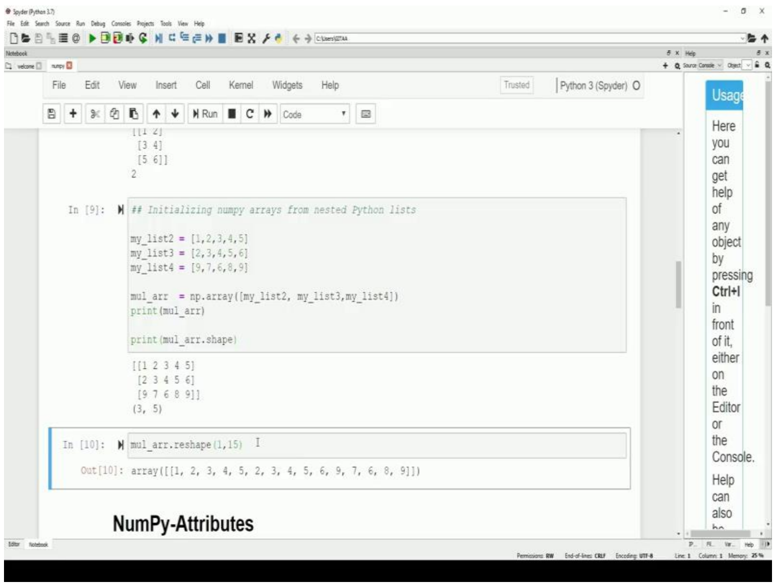
So, automatically you will have a vector when you create an array. So, you will have 6 rows with 1 column that is where you have 6, and nothing. But the shape of an array can be changed with various commands. I am going to use a command called reshape that is going to be used to reshape your array but you know that the following command will return only your modified array but it is not going to change the original array at all.

So, let us see how we can use the reshape function. So, 3, 2 represents though it is going to reshape the column with 3 rows and with 2 columns. So, I have reshaped the array which has 3 rows and 2 columns. Now and after reshaping it if you check the shape of the array it is going to give you the output as 3, 2 we have reshaped the array as for the dimension 3, 2. But if a dimension is given as minus 1 in a reshaping operation like the example that I have given then the other dimensions are automatically calculated.

Because the 3 the first argument 3 represents how many number of it should have and the minus one represents the other dimension is going to be calculated automatically by the numpy library because this is just a toy example to illustrate you how the arrays are created and how we can perform some operations on arrays. But if you are dealing with a large set of array then in that case you are not sure about the dimensions of your array then if you just want to reshape it according to your desired dimension then you can just use minus 1 as any one of the dimension.

Because in order to use reshape at least you should be specifying one dimension exactly and the other dimension can be calculated automatically by the numpy array once you fix minus 1. So, here I have fixed minus 1 along the column dimension. So, you will have 3 rows and 2 columns this has been calculated according to the size of your array then. So, basically it depends on the number of elements that the array consists of since I have 6 values I have just given it as 3.

(Refer Slide Time: 09:16)



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

In [9]:  ## Initializing numpy arrays from nested Python lists

         my_list2 = [1,2,3,4,5]
         my_list3 = [2,3,4,5,6]
         my_list4 = [9,7,6,8,9]

         mul_arr  = np.array([my_list2, my_list3,my_list4])
         print(mul_arr)

         print(mul_arr.shape)

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

In [10]: mul_arr.reshape(1,15)

Out[10]: array([[1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 9, 7, 6, 8, 9]])
```

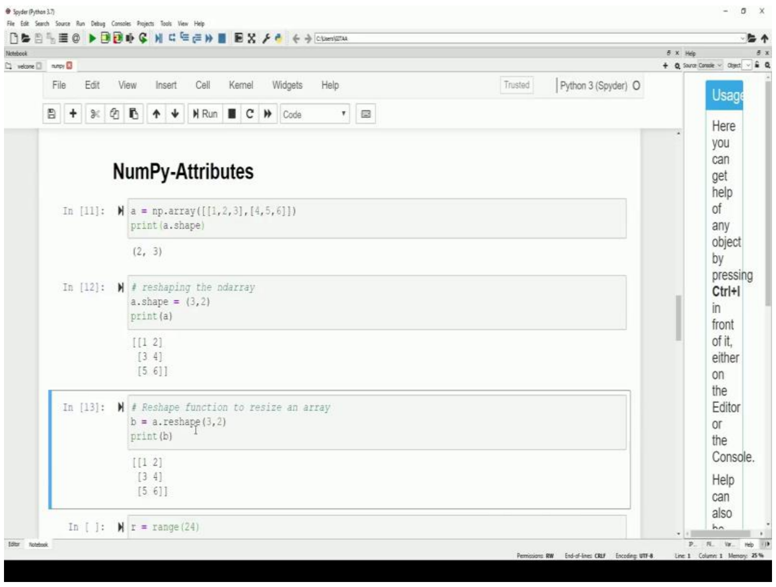
NumPy-Attributes

So, it is going to reshape it as 3, 2. So, that is what it has but now if you check the dimension of your array using ndim it just says us to because you have to access to the array now one knows row and oneness column. So, the dimension of the array is 2. So, we can also initialize numpy arrays from nested Python lists as well. So, initially we had only one list as an argument to the number array but here I am going to create an umpire array from nested Python list for that I have created three sets of lists having few numbers to it.

So, I have three lists my list 2 my list 3 my list 4 I am using the array function from numpy library and inside that I am going to give lists of lists as the elements that needs to be passed in as an argument. So, if you can see here my list itself is a list and I have given three lists and then given square brackets outside of it and inside the list also I have three lists to it. So, if you just print it, it is going to give you the output like this. So, you have created an array called mul underscore arr which has some values to it.

Now if you check the shape of it to check the dimension it gives us 3, 5 since it has 3 rows with 5 columns. And you should be able to reshape your multi-dimensional array as well using the reshape function and I am just reshaping the multi-dimensional array as 1, 15. So, that I can have the array values in single row with 15 separate columns

(Refer Slide Time: 10:58)

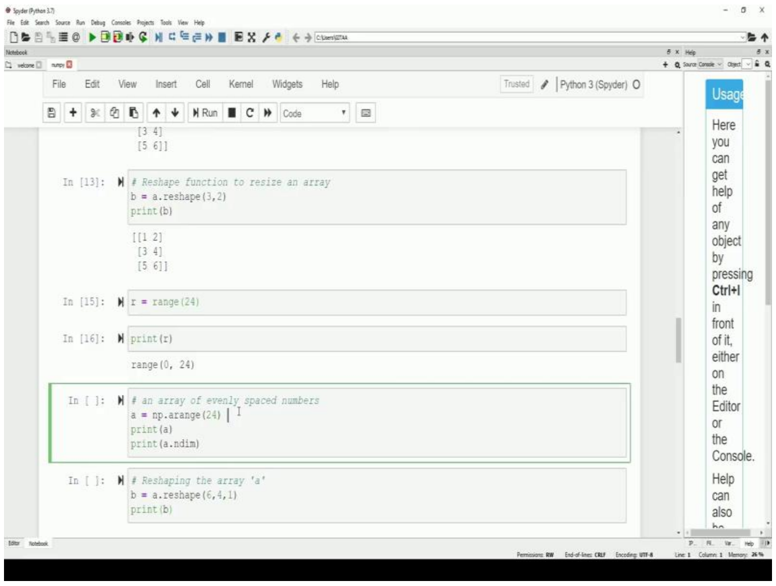

NumPy-Attributes

```
In [11]:  a = np.array([[1,2,3],[4,5,6]])
          print(a.shape)

          (2, 3)

In [12]:  # reshaping the ndarray
          a.shape = (3,2)
          print(a)

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

In [13]:  # Reshape function to resize an array
          b = a.reshape(3,2)
          print(b)

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

In [ ]:   r = range(24)
```

And then we are going to look at some of the attributes that you can fetch from your numpy. So, I have just created an array called a which has two lists to it. So, it has two rows and three columns that is the shape of the array the shape is one of the attribute. As I have told you there are several ways or there are several commands that is used to reshape the array. But I have given you an example on, on how to reshape the array using the reshape function.

Here there is another method which you can use to reshape the n dimensional array because a dot shape basically gives you the dimension of the array and I am just passing a tuple which contains some integer and which also describes the dimension of the array as the input. I have reshaped my array with the dimension 3, 2. So, now the reshaped array is given here and I am also mentioned how to reshape the array using the reshape function.

(Refer Slide Time: 11:52)



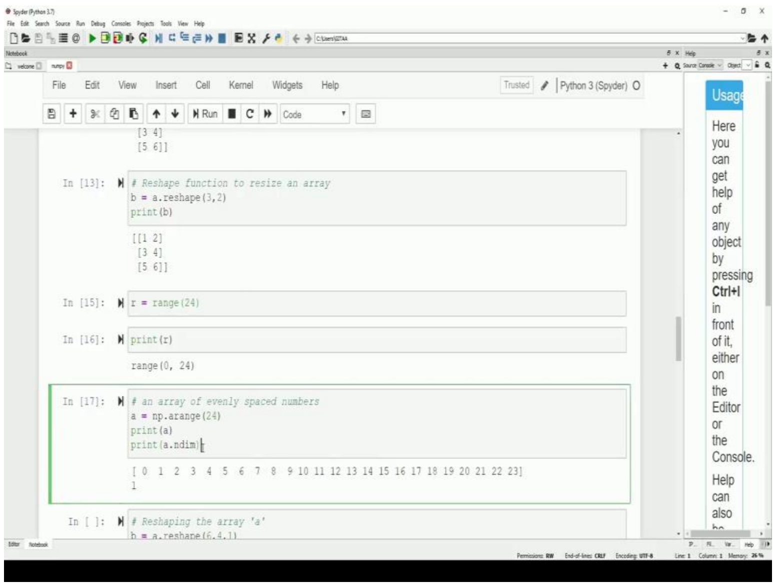
```
 [3 4]
 [5 6]]

In [13]:  # Reshape function to resize an array
          b = a.reshape(3,2)
          print(b)

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

In [15]:  r = range(24)

In [16]:  print(r)

          range(0, 24)

In [ ]:   # an array of evenly spaced numbers
          a = np.arange(24)
          print(a)
          print(a.ndim)

In [ ]:   # Reshaping the array 'a'
          b = a.reshape(6,4,1)
          print(b)
```

Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also

You will also be able to create a sequence of values and then created as an array using the range function. So, I have given a range of 24. So, it is going to give me a sequence of values starting from 0 to 24 as a range. But the downside to use the range function is you would not be able to see what are the values that are there under the created range it is just going to return the range of 0 comma 24 as an output when you print the r.

So, the range function is often used when you are dealing with control structures and if you want to iterate a value with the given sequence of values using range in that case range will be very helpful. But to create a sequence of values we can use a function called arranged from the numpy library.

(Refer Slide Time: 12:42)

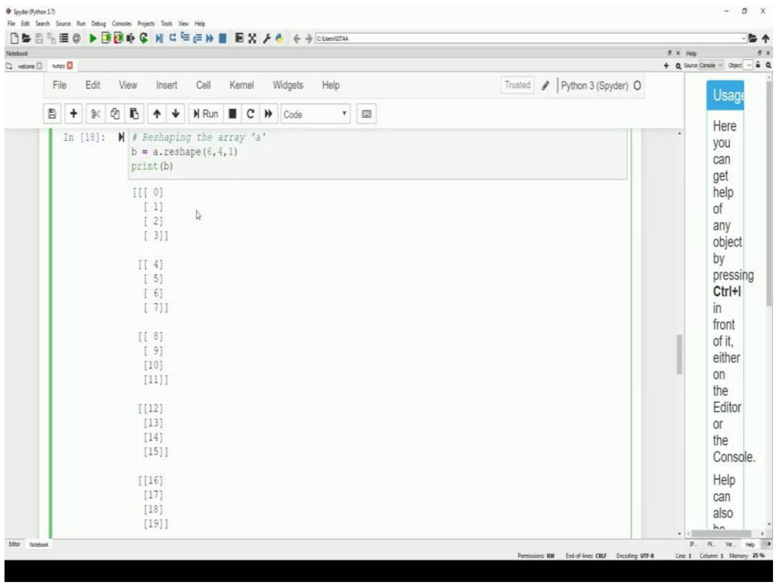


```
 [3 4]
 [5 6]]

In [13]:  # Reshape function to resize an array
          b = a.reshape(3,2)
          print(b)

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

In [15]:  r = range(24)

In [16]:  print(r)

          range(0, 24)

In [17]:  # an array of evenly spaced numbers
          a = np.arange(24)
          print(a)
          print(a.ndim)

          [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
          1

In [ ]:   # Reshaping the array 'a'
          b = a.reshape(6,4,1)
```

It is called as a rage because from numpy it is going to create an array based on the sequence of values that you have given at as an input but it is going to return an array that is why the function name itself is a range. So, the arrange function syntax the start value stop value and then incremental value. So, it basically takes in three arguments I have just given only one argument. So, by default it is going to take that value as a stop value and if you are not given any start value it is going to start from zero.

And by default it is going to increment the value but number one I have given 24 here but you would have got till 23 that describes that the stock value whatever I have given here it will return one little n minus one. So, if you have given 24 it will return to 23. If you check the dimension of your the array created now that is a it is only 1.

(Refer Slide Time: 13:40)



```
In [18]:  # Reshaping the array 'a'
           b = a.reshape(6,4,1)
           print(b)

[[[ 0]
  [ 1]
  [ 2]
  [ 3]]

 [[ 4]
  [ 5]
  [ 6]
  [ 7]]

 [[ 8]
  [ 9]
  [10]
  [11]]

 [[12]
  [13]
  [14]
  [15]]

 [[16]
  [17]
  [18]
  [19]]
```

Usage

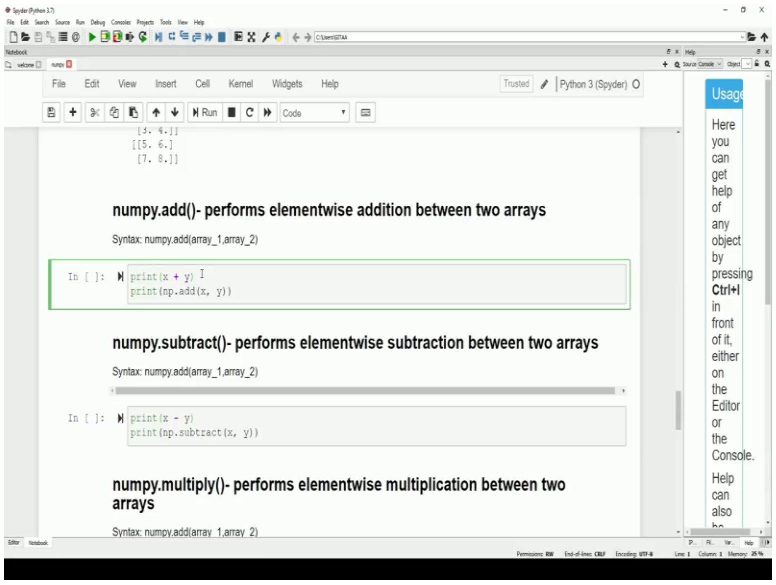
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also

So, if you can see here this is the original array I am just wrapping the array a given 6 as the first argument it is going to keep all the values into 6 vectors and each vector will have the size 4, 1 under each vector you will have values in four rows and with single column from 0 to 3 is a vector and it has and it has a dimension 4 cross 1 and the second vector also has 4 values to it in a single column similarly in the other vectors.

So, I have given the first dimension as the number of vectors it should have or under each vector what is the size that is 4, 1 that represents the number of rows and number of columns that you are going to use to reshape the array. So, this is how we can reshape the array using the reshape function.

(Refer Slide Time: 14:26)



```
 [3. 4.]]
[[5. 6.]
 [7. 8.]]
```

numpy.add()- performs elementwise addition between two arrays

Syntax: numpy.add(array_1,array_2)

```
In [ ]: print(x + y)
        print(np.add(x, y))
```

numpy.subtract()- performs elementwise subtraction between two arrays

Syntax: numpy.add(array_1,array_2)

```
In [ ]: print(x - y)
        print(np.subtract(x, y))
```

numpy.multiply()- performs elementwise multiplication between two arrays

Syntax: numpy.add(array_1,array_2)

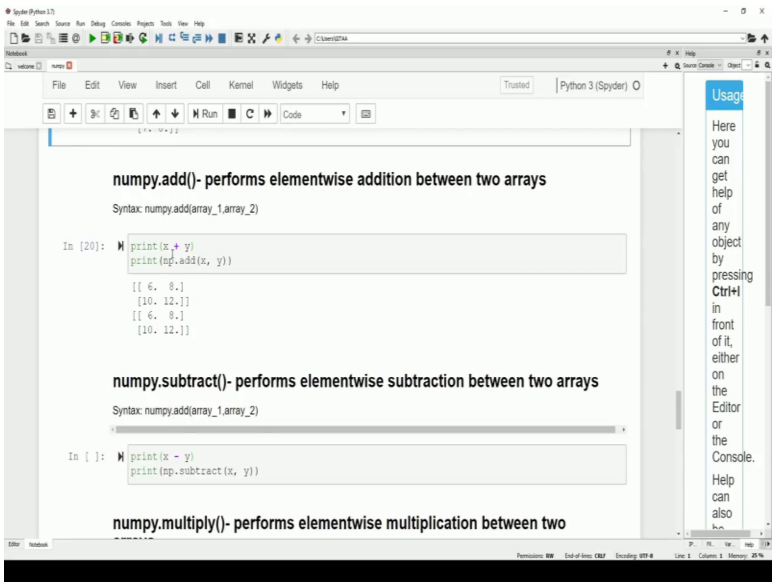
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also

And you can as well perform some of the arithmetic operations on the numpy arrays. So, I have created two arrays. So, the elements inside the x and y arrays are of float 64 and you can perform some of the arithmetic operations like addition, subtraction, multiplication, division and so, on and so, forth on the arrays that you created. So, let us let me just take you through one example on each. So, first we are going to see add. So, the numpy dot at performs the element wise addition between two arrays.

And the syntax is just you can use the add function and inside the function you can specify the to erase that is going to be used to do the addition and you can do that or you can do the element wise addition using the plus operator as well. So, I am illustrating the addition using both plus operator and using the add function. So, as I have mentioned earlier it is going to do the element wise addition.

(Refer Slide Time: 15:22)



numpy.add()- performs elementwise addition between two arrays

Syntax: numpy.add(array_1,array_2)

```
In [20]:  print(x + y)
          print(np.add(x, y))

          [[ 6.  8.]
           [10. 12.]]
          [[ 6.  8.]
           [10. 12.]]
```

numpy.subtract()- performs elementwise subtraction between two arrays

Syntax: numpy.add(array_1,array_2)

```
In [ ]:  print(x - y)
         print(np.subtract(x, y))
```

numpy.multiply()- performs elementwise multiplication between two

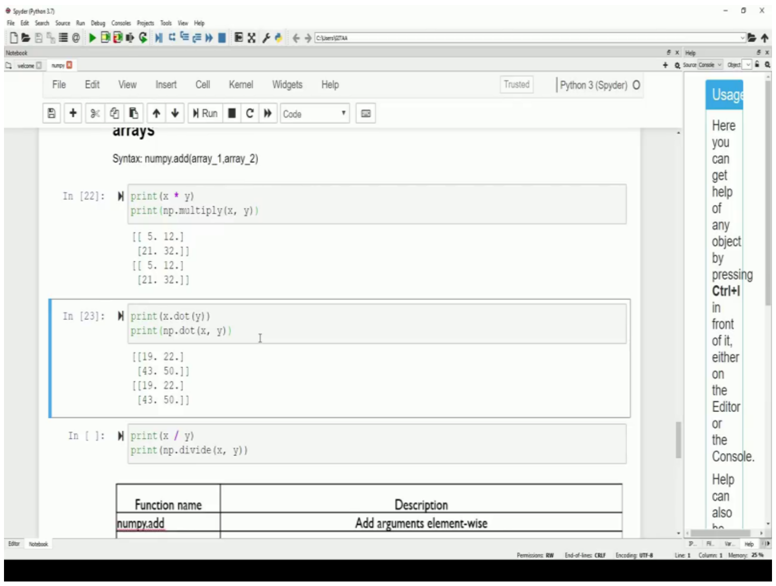
Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also

So, the first element of the array a that is one. So, the first element of the array x that is one and the first element of the array y that is prime will get added one plus five would be 6. So, that is the output that you are seeing here. So similarly it does the element wise addition and on all the cases and you have to output for one plus two operator and for one using the np dot add function.

(Refer Slide Time: 15:44)



arrays

Syntax: numpy.add(array_1,array_2)

```
In [22]:  print(x * y)
          print(np.multiply(x, y))

          [[ 5. 12.]
           [21. 32.]]
          [[ 5. 12.]
           [21. 32.]]

In [23]:  print(x.dot(y))
          print(np.dot(x, y))

          [[19. 22.]
           [43. 50.]]
          [[19. 22.]
           [43. 50.]]

In [ ]:  print(x / y)
         print(np.divide(x, y))
```

Function name	Description
numpy.add	Add arguments element-wise

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

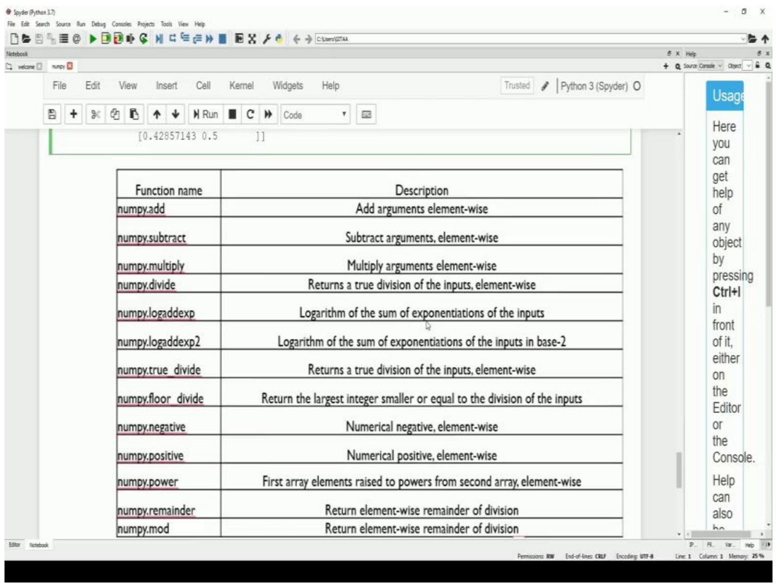
Help can also

Similarly you can perform the element wise subtraction between two arrays using the subtract function from numpy and the syntax remains the same and you just need to replace the same command with the minus and the subtract function and it does the element wise subtraction and you can perform the element wise multiplication between two arrays and using astrup you will be able to do element wise multiplication between two arrays.

But if you want to get the dot product of two arrays then you can use the function called dot x dot of y is going to give you the dot product of the two arrays x and y and you can use the dot function either by applying it on one array and then using it on the other array or you can just use np dot dot and inside the function you can just give the two desired erase that needs to be multiplied.

And similarly you can also perform the division between the two arrays that is x and y by using the slash operator or by using the divide function from numpy library. So, all of the addition subtraction multiplication and division using the required commands is going to do the element wise operation between two arrays.

(Refer Slide Time: 16:55)



```
 [0.42857143 0.5        ]]
```

Function name	Description
numpy.add	Add arguments element-wise
numpy.subtract	Subtract arguments, element-wise
numpy.multiply	Multiply arguments element-wise
numpy.divide	Returns a true division of the inputs, element-wise
numpy.logaddexp	Logarithm of the sum of exponentiations of the inputs
numpy.logaddexp2	Logarithm of the sum of exponentiations of the inputs in base-2
numpy.true_divide	Returns a true division of the inputs, element-wise
numpy.floor_divide	Return the largest integer smaller or equal to the division of the inputs
numpy.negative	Numerical negative, element-wise
numpy.positive	Numerical positive, element-wise
numpy.power	First array elements raised to powers from second array, element-wise
numpy.remainder	Return element-wise remainder of division
numpy.mod	Return element-wise remainder of division

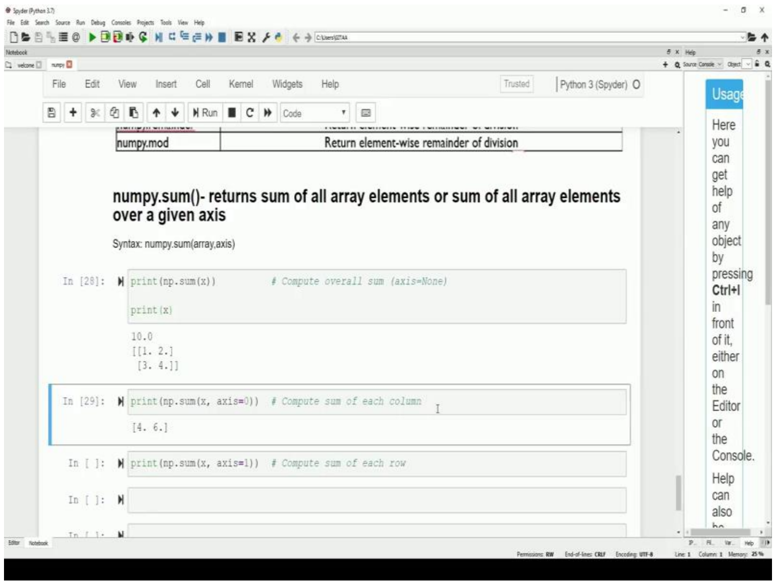
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also

I have given a snippet on how you can use the other functions on the arrays. I have given the function name in the first column and I will just also added the discription along with that and you can refer to it basically in order to explore more functions that we can perform on numpy arrays.

(Refer Slide Time: 17:15)



numpy.mod	Return element-wise remainder of division

numpy.sum()- returns sum of all array elements or sum of all array elements over a given axis

Syntax: numpy.sum(array,axis)

```
In [28]:  print(np.sum(x))         # Compute overall sum (axis=None)

          print(x)

          10.0
          [[1. 2.]
           [3. 4.]]

In [29]:  print(np.sum(x, axis=0))  # Compute sum of each column

          [4. 6.]

In [ ]:   print(np.sum(x, axis=1))  # Compute sum of each row

In [ ]:
```

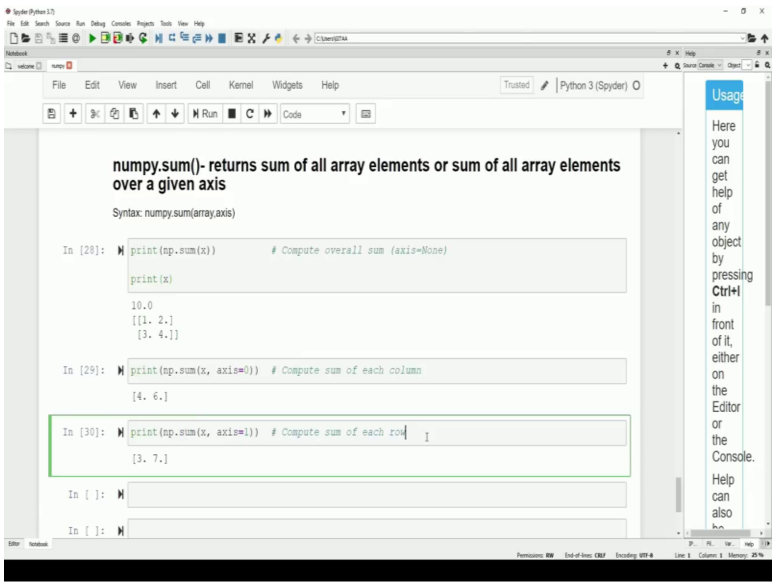
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also

Then we are going to perform some operations on numpy erase the first thing there is sum. So, using the dot sum from the numpy library returns the sum of all array elements or sum of all array elements over a given axis. So, the syntax is numpy dot sum and the first argument is the array desired array that is going to be used. The next one is axis. So, let us see how we can use this function. So, I am just printing np dot some of x if you can look at here I am accessing the function sum from the numpy library and inside the function I have just given what only one argument that is x which is the array that we have already created and I have not given any access value.

So, by default the axis is none. So, it is just going to compute the overall sum of the array x. So, the printer the output of x. So, this is what the array x contains in it and by using np dot some on x it computes the overall sum of the array x 1 plus 2 Plus 3 plus 4 but if you are providing a value in the access argument then it is going to perform accordingly. So, the axis values can either be 0 or 1. So, the zero represents performing the operation on rows and the 1 represents performing operations on columns.

(Refer Slide Time: 18:33)



numpy.sum()- returns sum of all array elements or sum of all array elements over a given axis

Syntax: numpy.sum(array,axis)

```
In [28]: print(np.sum(x))          # Compute overall sum (axis=None)

         print(x)

         10.0
         [[1. 2.]
          [3. 4.]]

In [29]: print(np.sum(x, axis=0))  # Compute sum of each column

         [4. 6.]

In [30]: print(np.sum(x, axis=1))  # Compute sum of each row

         [3. 7.]

In [ ]:

In [ ]:
```

For example it computes the sum of each column on the array x for example I have given access is equal to 0 then it is going to take the value from the first row and then value from the second row though it is going to hop across the rows it is just going to give you the sum of each column. So, if you see 1 plus 3 gives you 4 and 2 plus 4 plus 6. So, it basically if you give axis is equal to 0 it computes the sum of each column and by giving access is equal to 1 it computes the sum of each row. So, that each row sum will be later by hopping across the columns that is we have given one that is 1 plus two will be three and three plus 4 will be 7.

So, similarly you will be able to use any pattern built in functions like mean median instead of sum you can explore more on that and you will also be able to perform the other functions that you have already created on the number array as well. In this lecture we have seen what is the advantage of using the numpy arrays over the built-in Python arrays? And in the upcoming lectures we will be seeing how these number arrays can be used to perform the mathematical and logical operations on multi-dimensional data, thank you.