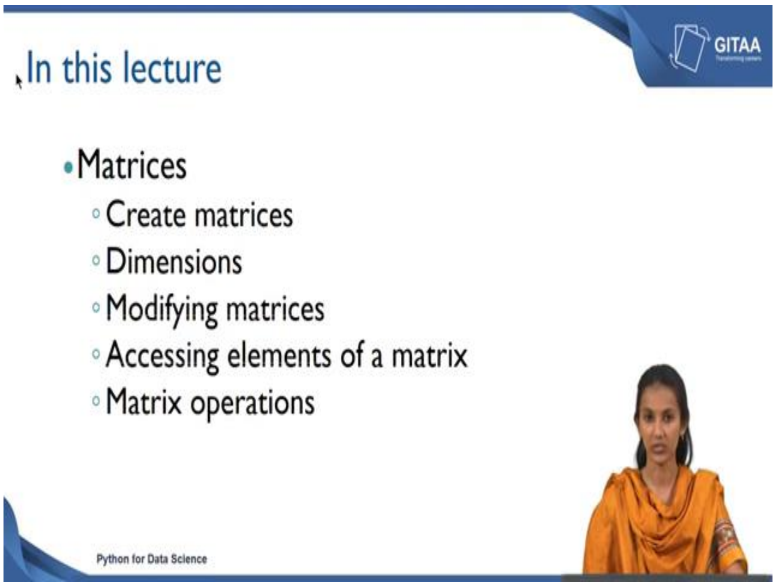
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 14
Matrix

(Refer Slide Time: 00:16)



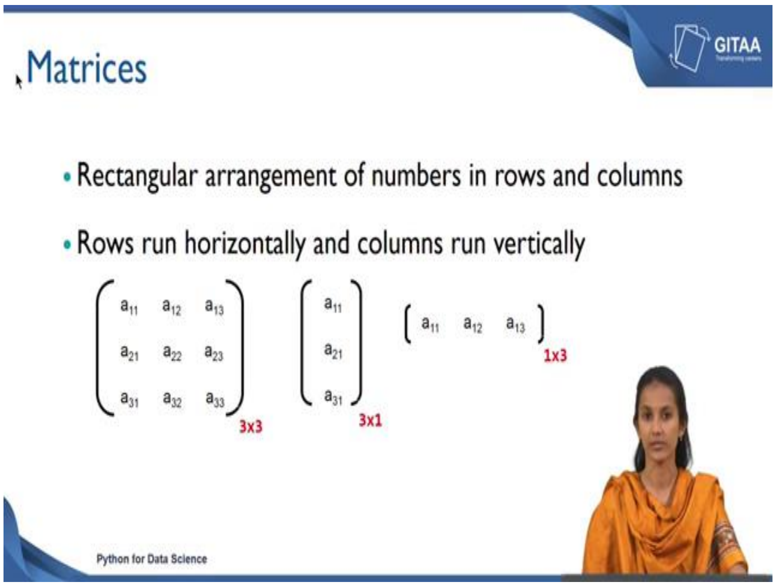
In this lecture

- Matrices
  - Create matrices
  - Dimensions
  - Modifying matrices
  - Accessing elements of a matrix
  - Matrix operations

GITAA
Python for Data Science

Welcome to the lecture in this lecture we will see what is mean by Matrices. How to create matrices in python and also how to check dimensions of the matrix, we will also see how to modify elements in the matrix and how to access the elements and also the some of the matrix operations which is available in python.

(Refer Slide Time: 00:41)



Matrices

- Rectangular arrangement of numbers in rows and columns
- Rows run horizontally and columns run vertically

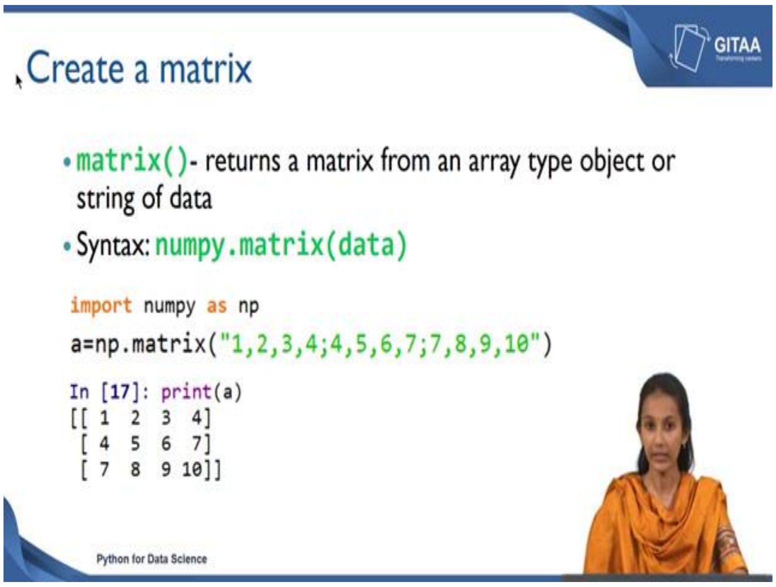
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}_{3x3} \quad \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix}_{3x1} \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \end{pmatrix}_{1x3}$$

Python for Data Science

Let us get started. So, first we will see what is mean by a matrix. Matrices are two dimensional data structure consisting of numbers in rows and columns, rows runs horizontally and columns runs vertically. First we will take a matrix, so this consists of 9 elements. So, it from a_{11} to a_{12}, a_{13}.

So, this is a first row and second row it consist of elements of a_{21}, a_{22}, a_{23}, similarly for third row. We will take an another example so this one is a 3 cross 1 matrix which means 3 rows and 1 columns this one will be a 1 cross 3 matrix which means 1 row and 3 columns.

(Refer Slide Time: 01:34)


Create a matrix

- `matrix()`- returns a matrix from an array type object or string of data
- Syntax: `numpy.matrix(data)`

```
import numpy as np
a=np.matrix("1,2,3,4;4,5,6,7;7,8,9,10")
```

```
In [17]: print(a)
[[ 1  2  3  4]
 [ 4  5  6  7]
 [ 7  8  9 10]]
```

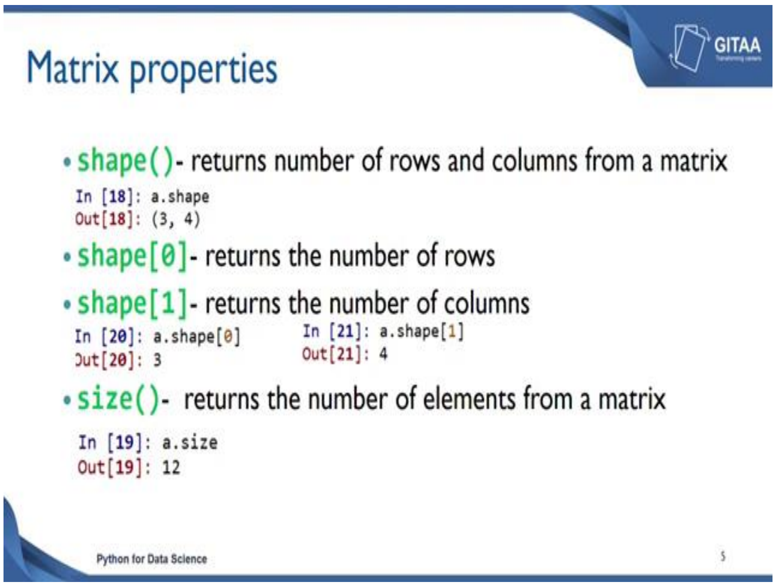
Let us see how to create a matrix in python matrix is command to create a matrix. So, it basically returns a matrix from the array type object. So, the syntax is you have to import the numpy package.

So, from the numpy package we will call the function call matrix inside the parenthesis you have to specify the values for the matrix. Before creating a matrix we will first import the numpy package as np. Now we will create a 3 cross 4 matrix which means 3 rows and 4 columns. So, since we have already imported numpy as np.

So, I will call here after np from np.matrix. So, inside the parenthesis we need to specify the values right I am giving values 1, 2, 3, 4 after that 4 you can see there is a semicolon which means so, we have fill the values for the first row 4 5 6 still 7 and after 7 again you can see a semicolon.

So, which means we have fill the values for the second row similarly you can give values for the third row any values can be given to create a matrix. We are storing a matrix in the variable call a. When we print a so it has a created a matrix with 3 cross 4.

(Refer Slide Time: 02:55)



Matrix properties

- shape()- returns number of rows and columns from a matrix

```
In [18]: a.shape
Out[18]: (3, 4)
```

- shape[0]- returns the number of rows
- shape[1]- returns the number of columns

```
In [20]: a.shape[0]        In [21]: a.shape[1]
Out[20]: 3                 Out[21]: 4
```

- size()- returns the number of elements from a matrix

```
In [19]: a.size
Out[19]: 12
```

Python for Data Science

5

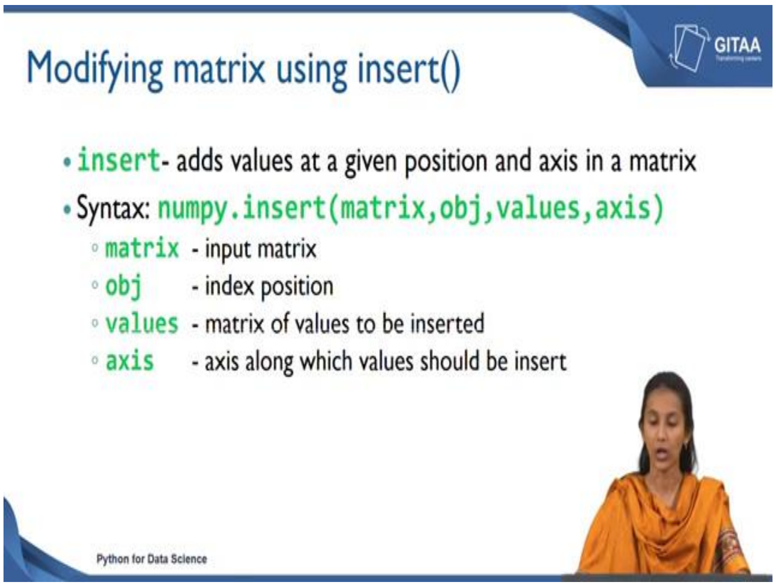
Let us see some of the matrix properties. So, if you wanted to know the number of rows and columns then you can use the shape command. So, you have to specify the matrix name followed by.shape. So, we will basically returns the number of rows and number of columns let us say if you wanted to get the number of rows alone then you can use the shape inside the square brackets you can specify 0.

So, 0 is basically for rows and 1 is basically for columns. So, shape of 1 it basically returns the number of columns in the matrix.

So, a.shape(0) 0 it basically returns the number of rows. So, in our case we have 3 rows similarly if you wanted to extract the number of columns then you can use a.shape(1).

So, in our case it returns a value 4 which means 4 columns are there in a matrix. Let us say if you wanted to extract the number of elements in the matrix then you can use the size command. So, we again use a matrix a a.size, so since it is a 3 cross 4 matrix. So, we will be getting a 12 elements in the matrix.

(Refer Slide Time: 04:15)



Modifying matrix using insert()

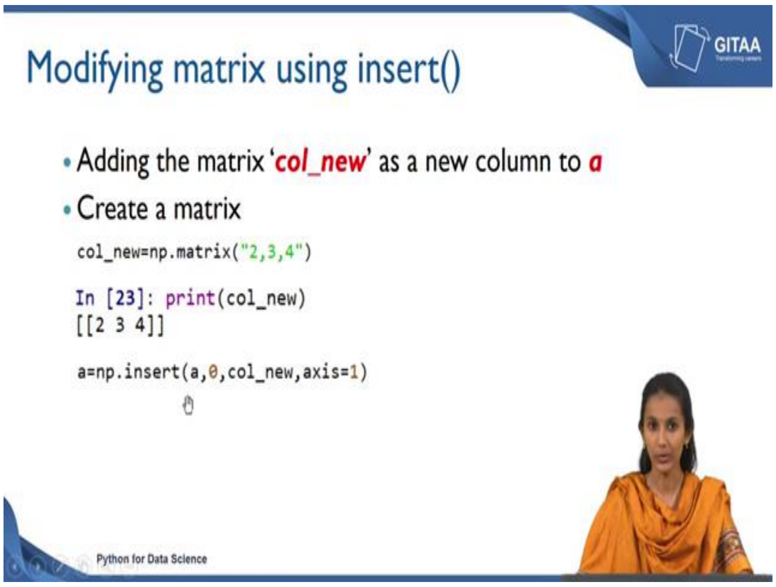
- insert- adds values at a given position and axis in a matrix
- Syntax: numpy.insert(matrix,obj,values,axis)
  - matrix - input matrix
  - obj - index position
  - values - matrix of values to be inserted
  - axis - axis along which values should be insert

Python for Data Science

Let us see how to modify the matrix using some of the built in functions which is available. So, first we will use the insert command. So, insert what it does is it adds a values at a given position and also you can specify the axis as well.

So, the syntax is numpy.insert you have to specify the matrix which is a input matrix and then the object which means a index position at which index you need to insert the matrix and the corresponding values and then axis. So, axis is equal to 0 is for rows, and axis is equal 1 is for columns along which axis you need to insert the matrix values.

(Refer Slide Time: 05:05)



Modifying matrix using insert()

- Adding the matrix '*col_new*' as a new column to *a*
- Create a matrix

```
col_new=np.matrix("2,3,4")

In [23]: print(col_new)
[[2 3 4]]

a=np.insert(a,0,col_new,axis=1)
```

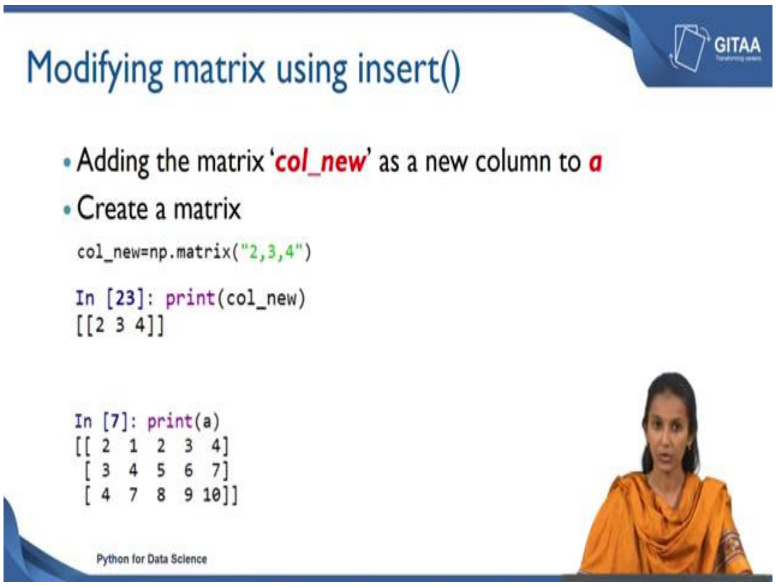
Python for Data Science

So, first we will add that matrix col_new as a new column to the existing matrix which is a. So, first we will create a matrix col_new and we will supply some values col_new. So, in that it has values 2, 3 and 4. So, when you print it so it returns a values which you are declared earlier let us see how to insert this matrix to the existing matrix a.

So, np.insert so the a which is the existing matrix you have to specify the old matrix and if you wanted to index you have to specify index position at which index position you will need to insert the matrix.

So, here I am specifying 0, which means it will be inserted at the zeroth index position and then the new matrix which we have created now col_new and if you wanted to insert along the columns you have to specify axis is equal to 1 and I am storing it back to the same matrix a.

(Refer Slide Time: 06:19)



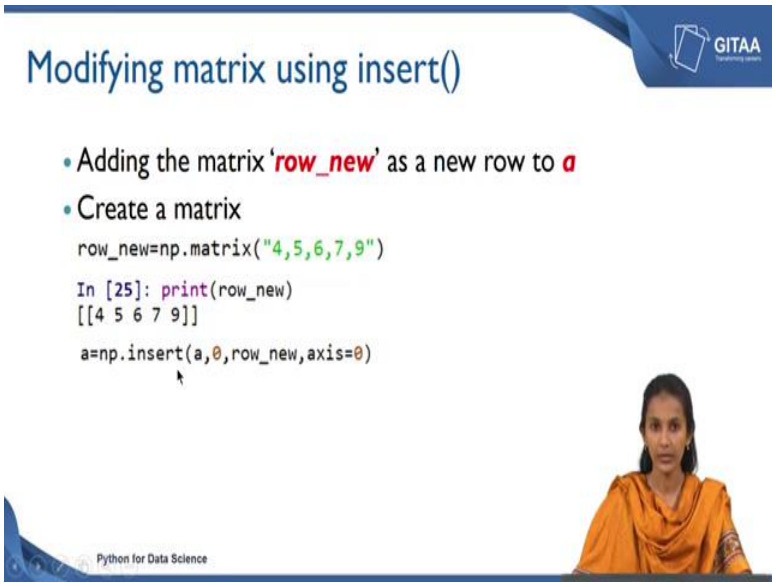
Modifying matrix using insert()

- Adding the matrix '*col_new*' as a new column to *a*
- Create a matrix

```
col_new=np.matrix("2,3,4")

In [23]: print(col_new)
[[2 3 4]]


In [7]: print(a)
[[ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

Now let us see what is the output when you print a. So, now it will be inserted at the zeroth position. So, you can also insert at the first position first index position or the second index position as well.

(Refer Slide Time: 06:29)



Modifying matrix using insert()

- Adding the matrix '**row_new**' as a new row to **a**
- Create a matrix

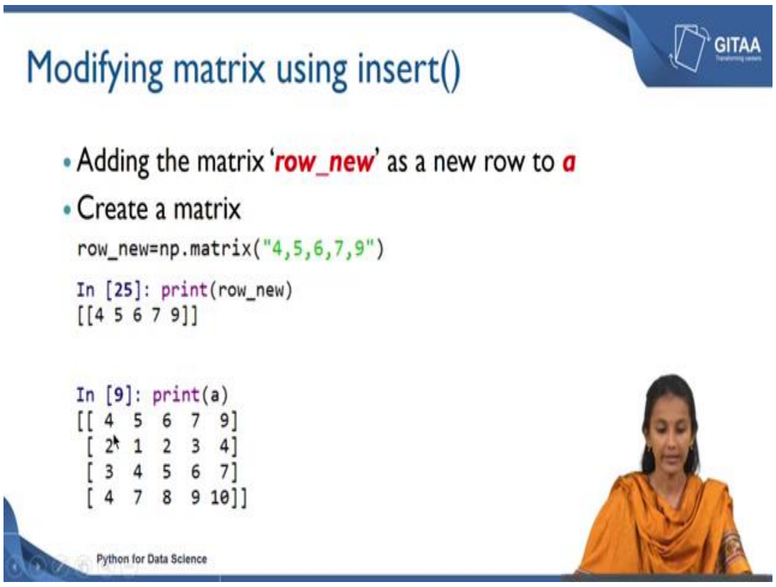
```
row_new=np.matrix("4,5,6,7,9")

In [25]: print(row_new)
[[4 5 6 7 9]]

a=np.insert(a,0,row_new,axis=0)
```

Let us see how to add a matrix along the row wise. So, first we will create a matrix row_new and we will add it as a new row to the existing matrix a. So, in the a matrix so now, the dimensions will be 3 cross 5 right. So, we need to supply 5 values for the row matrix.

So, np.matrix so inside the parenthesis I am specifying the values for the row matrix and storing it in a matrix row_new when you print 4, 5, 6, 7, 9 will be printed.

Let us see how to add this matrix to the existing matrix a. So, you can use the np.insert command. So, the existing matrix a and here also I am adding at the zeroth index position and then the matrix which we have created now which is a row_new. Since we wanted to add along row wise we have to specify axis is equal to 0, I am storing it back to the same matrix a now.

(Refer Slide Time: 07:40)



Modifying matrix using insert()

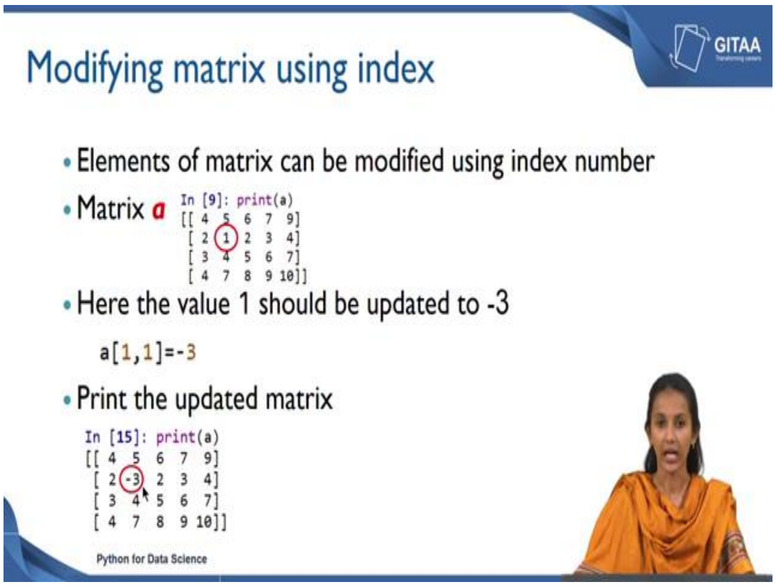
- Adding the matrix '*row_new*' as a new row to *a*
- Create a matrix

```
row_new=np.matrix("4,5,6,7,9")

In [25]: print(row_new)
[[4 5 6 7 9]]


In [9]: print(a)
[[ 4  5  6  7  9]
 [ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

Python for Data Science

So, let us see what is a output. So, when we print a. So, now, the dimensions will be changed. So, we have already add 3 cross 5; now the dimensions of the matrix will be a 4 cross 5 matrix which means 4 rows and 5 columns and the matrix has been added at the zeroth index position. So, 4, 5, 6, 7, 9 has been added along the row wise at the zeroth index position.

(Refer Slide Time: 08:04)



Modifying matrix using index

- Elements of matrix can be modified using index number
- Matrix *a*

```
In [9]: print(a)
[[ 4  5  6  7  9]
 [ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

- Here the value 1 should be updated to -3

```
a[1,1]=-3
```

- Print the updated matrix

```
In [15]: print(a)
[[ 4  5  6  7  9]
 [ 2 -3  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

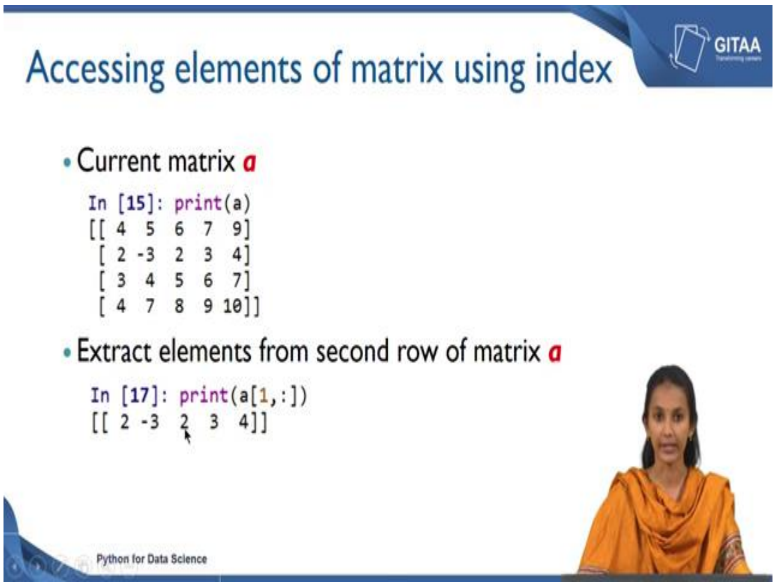
Python for Data Science

Let us see how to modify the matrix using the index number. So, I am going to print the matrix a. So, this is our matrix now the dimensions is 4 cross 5. So, let us say if you wanted to change the value of 1, 2 -3.

So, we can change the elements of matrix using the index number. So, you have to specify the matrix name a. So, the row index is 1, and the column index is 1.

So, you have to specify the index numbers. So, 1 corresponds to the row index and this 1 corresponds to the column index since we wanted to change to the -3; I am specifying here. So, let us see the modified matrix we will print the updated matrix. So, this will be our updated matrix. Now the value of 1 has been change to -3.

(Refer Slide Time: 08:58)



Accessing elements of matrix using index

- Current matrix *a*

```
In [15]: print(a)
[[ 4  5  6  7  9]
 [ 2 -3  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

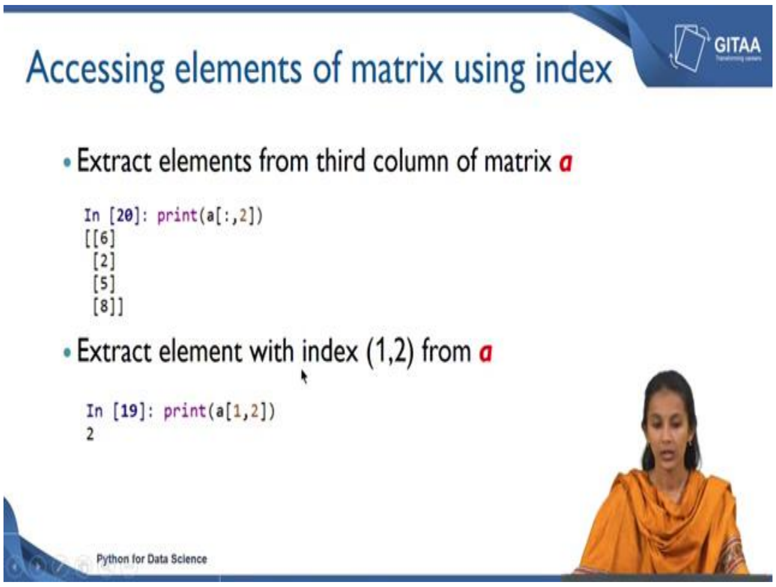
- Extract elements from second row of matrix *a*

```
In [17]: print(a[1,:])
[[ 2 -3  2  3  4]]
```

Next we will see how to access a elements of the matrix using the index number. So, first we will print the current matrix which is our a. So, after modified so this is our current matrix which is of dimensions 4 cross 5. Let us say if you wanted to extract elements from the second row you can access using the index number.

So, if you give print of a followed by the row index. So, the second row corresponds to the index number of 1. So, 1 comma colon which means takes all the columns corresponding to that index. So, once you have done this so we will get an output of 2, -3, 2, 3, 4.

(Refer Slide Time: 09:44)



Accessing elements of matrix using index

- Extract elements from third column of matrix *a*

```
In [20]: print(a[:,2])
[[6]
 [2]
 [5]
 [8]]
```

- Extract element with index (1,2) from *a*

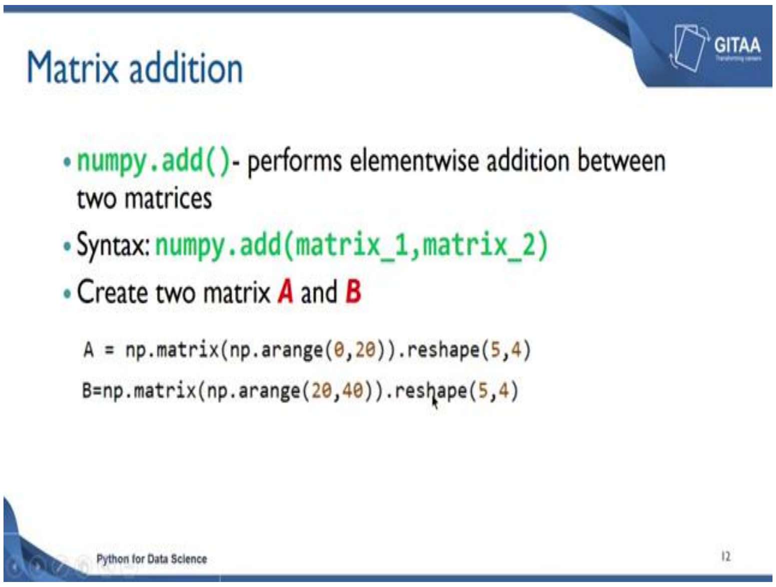
```
In [19]: print(a[1,2])
2
```

Let us say if you wanted to extract from the third column then you can do that as well. So, you have to specify the index number for the row index and for the column index here we wanted to extract from the third column. So, the third column corresponds to the index position 2 colon comma 2.

So, if you wanted to extract elements from the third column. So, the corresponding index number is 2. So, we wanted to extract those rows of a corresponding rows as well so that is why we are specifying colon. So, we will be getting an output of 6, 2, 5, 8, so these are the values for the corresponding third column.

If you wanted to extract a particular element using the index number you can also do that as well. So let us say you wanted to extract the element corresponding to the index of 1 comma 2. So, when you specify print of a. So, first is row index 1 comma 2 it returns an element 2.

(Refer Slide Time: 10:48)



Matrix addition

- numpy.add()- performs elementwise addition between two matrices
- Syntax: numpy.add(matrix_1,matrix_2)
- Create two matrix **A** and **B**

```
A = np.matrix(np.arange(0,20)).reshape(5,4)
B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Python for Data Science 12

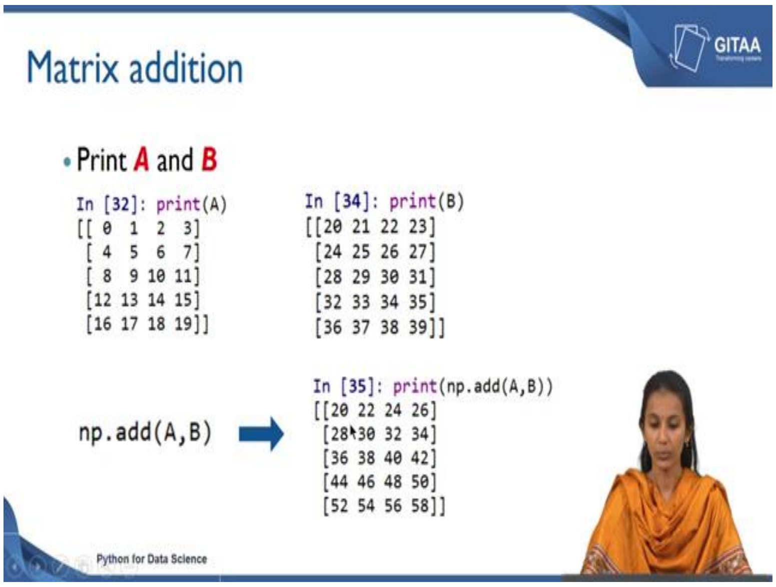
So, till now we saw how to create a matrix and some of the modifying operations let us get started with matrix operations. So, there we can also do addition, subtraction, multiplication, division as well so let us see one by one. So, first is matrix addition the command is numpy.add.

So, what it does is? It performs the element wise addition between the two matrices the syntax is numpy.add. So, inside the parenthesis you have to specify the matrix 1 name and followed by the matrix 2.

So, we will create a two matrix A and B. So, a so before calling the np.matrix you have to input the numpy package; import numpy as np and then you can use the command np.matrix here I am giving the values 0 to 19, so using the arange command. So, we have already saw what arange does and reshape, I am specifying the row dimensions and column dimensions.

Similarly you can create a matrix B here I am giving the values 20 to 40, so the same dimensions so which is 5 rows and 4 columns.

(Refer Slide Time: 12:09)



Matrix addition

- Print **A** and **B**

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

np.add(A,B) →

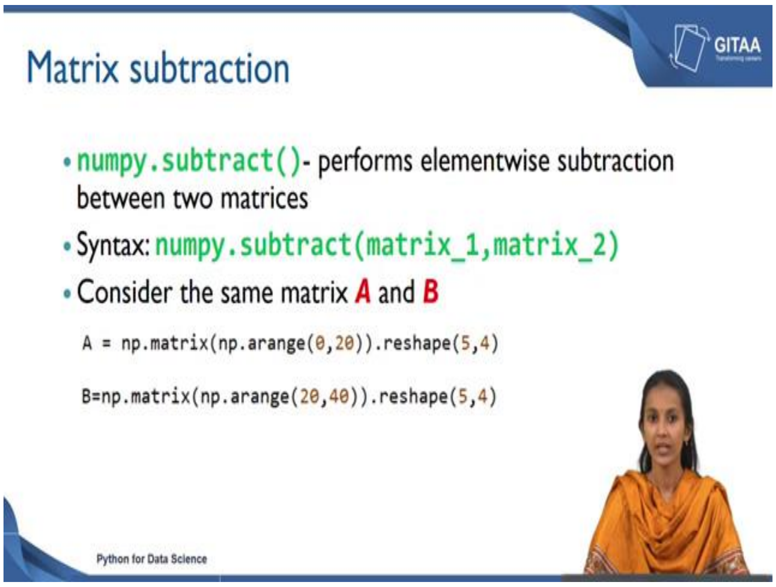
```
In [35]: print(np.add(A,B))
[[20 22 24 26]
 [28 30 32 34]
 [36 38 40 42]
 [44 46 48 50]
 [52 54 56 58]]
```

Python for Data Science

So, let us print A and B both the matrix. So, when we print A it has created a matrix with 5 rows and 4 columns start value is 0 and till the end value 19. Similarly when we print B so it has created A matrix with 5 rows and 4 columns the start value is 20, 21, 22 till 39. So, let us do this matrix addition for this both the matrix A and B so np.add A comma B. So, we will be getting an output.

So, it adds element wise. So, it adds 0 plus 20 and next one is will be 1 and 21, so we will be getting an output like this so 20, 22, 24 and 26 etcetera.

(Refer Slide Time: 13:01)



Matrix subtraction

- `numpy.subtract()`- performs elementwise subtraction between two matrices
- Syntax: `numpy.subtract(matrix_1,matrix_2)`
- Consider the same matrix **A** and **B**

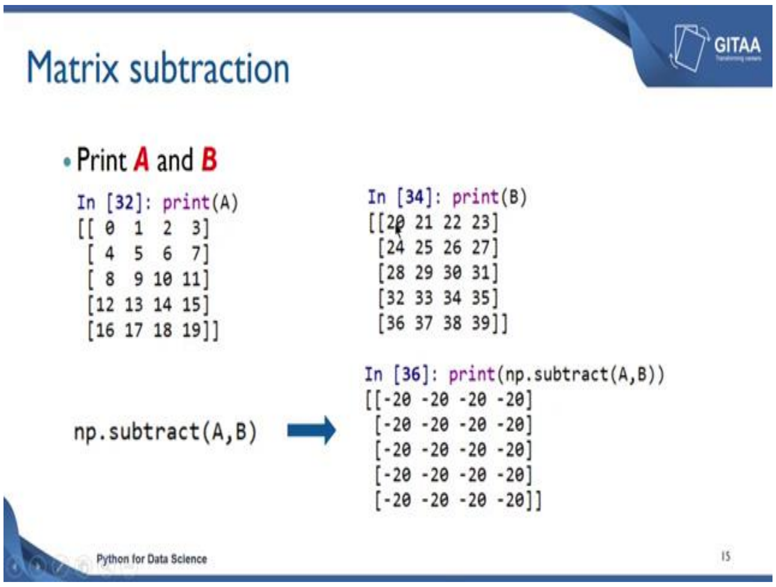
```
A = np.matrix(np.arange(0,20)).reshape(5,4)

B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Python for Data Science

Next we will do the matrix subtraction. So, numpy.subtracts it performs the element wise subtraction between the two matrices.

So, the syntax is numpy.subtract inside the parenthesis you have to specify the matrix 1 name followed by the matrix 2 name. So, we will consider the same matrix A and B which we have created earlier, so A this one is A. So, we use the values from 0 to 20 and B these are the values.

(Refer Slide Time: 13:31)



Matrix subtraction

- Print A and B

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

np.subtract(A,B)

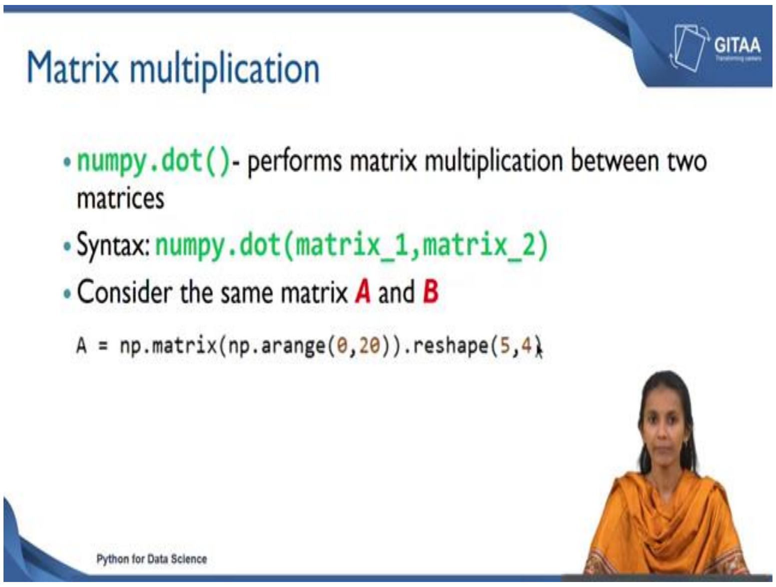
```
In [36]: print(np.subtract(A,B))
[[-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]]
```

Python for Data Science 15

So, let us print A and B. So, this is our matrix. So, for this matrix we will use the matrix subtraction. So, the command is np.subtract inside the parenthesis you have to specify the matrix A name and the followed by the matrix 2 which is B. So, when we print the matrix output will be -20 for the first one.

So, it is 0 -20. So, we will be getting an output of -20 for the second element. So, it is 1 - 21 so it is again -20, so for all the elements we are getting -20.

(Refer Slide Time: 14:18)



Matrix multiplication

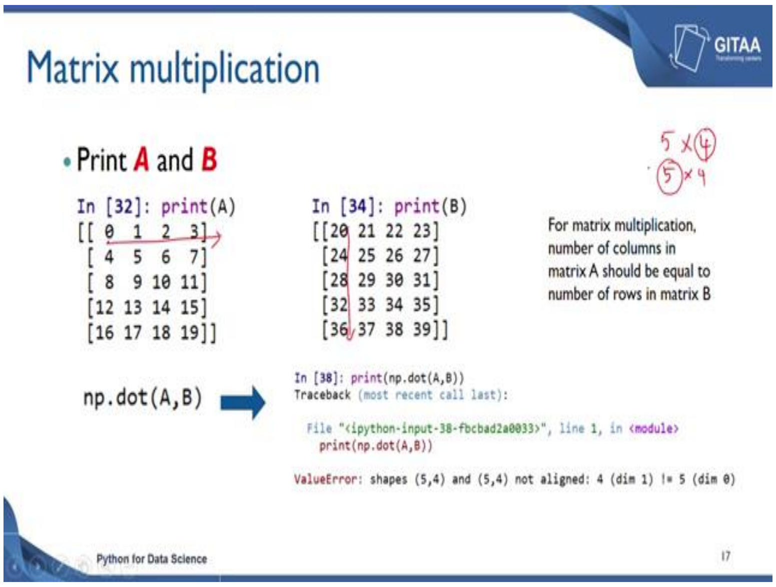
- numpy.dot()- performs matrix multiplication between two matrices
- Syntax: numpy.dot(matrix_1,matrix_2)
- Consider the same matrix *A* and *B*

```
A = np.matrix(np.arange(0,20)).reshape(5,4)
```

Python for Data Science

Next we will do matrix multiplication. So, numpy.dot it performs the matrix multiplication between the two matrices. So, the syntax is numpy.dot inside the parenthesis is again you have to specify the matrix 1 name followed by the matrix 2 name.

(Refer Slide Time: 14:40)



Matrix multiplication

- Print *A* and *B*

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

$5 \times \boxed{4}$

$\boxed{5} \times 4$

For matrix multiplication, number of columns in matrix A should be equal to number of rows in matrix B

np.dot(A,B)

```
In [38]: print(np.dot(A,B))
Traceback (most recent call last):

  File "<ipython-input-38-fbcbad2a0033>", line 1, in <module>
    print(np.dot(A,B))

ValueError: shapes (5,4) and (5,4) not aligned: 4 (dim 1) != 5 (dim 0)
```

Python for Data Science

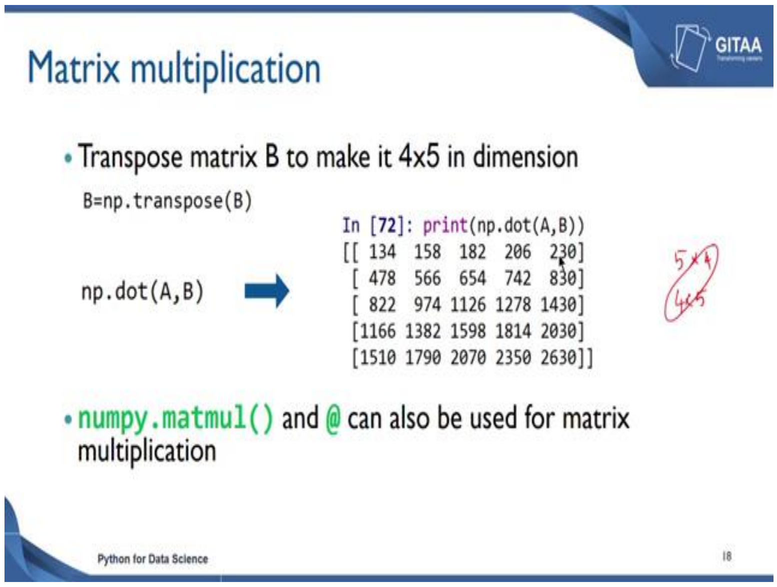
17

So, we will consider the same matrix A and B we will print again A and B. So, this is our A matrix and this is our B matrix. So, we will use the numpy.command for matrix multiplication. So, np.dot(A,B) so it does the matrix multiplication matrix multiplication in sense. So, it takes first row wise and next for the B matrix it takes along the column wise.

So it multiplies 0 into 20 plus 1 into 24 plus 2 into 28 plus 3 into 32. Let us see what is A output which are going to get. So, it has throws upon error value error shapes 5 comma 4 and 5 comma 4 not aligned yes for the A matrix we have the dimensions 5 cross 4, for the B matrix we have again 5 cross 4 for the matrix multiplication.

So, the one rule which we have to follow is number of columns in the matrix A it is basically should be equal to the number of rows in matrix B. So, since 5 cross 4 for matrix A and 5 cross 4 for matrix B when you compare the number of columns in matrix A is not equal to the number of rows in matrix B. So, that is why it has throwed up an error.

(Refer Slide Time: 16:13)



Matrix multiplication

- Transpose matrix B to make it 4x5 in dimension

```
B=np.transpose(B)
```

```
np.dot(A,B)
```

```
In [72]: print(np.dot(A,B))
[[ 134  158  182  206  230]
 [ 478  566  654  742  830]
 [ 822  974 1126 1278 1430]
 [1166 1382 1598 1814 2030]
 [1510 1790 2070 2350 2630]]
```

5×4

4×5

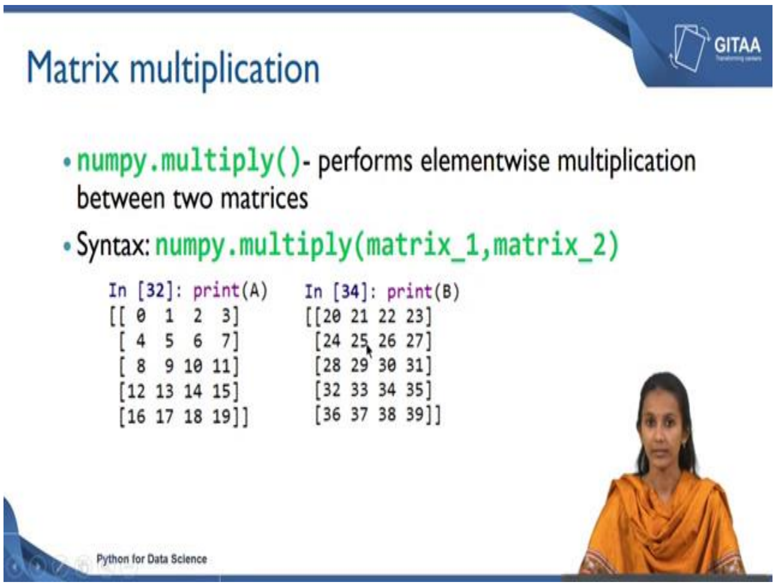
- numpy.matmul() and @ can also be used for matrix multiplication

Python for Data Science 18

So, now what we will do is we will transpose the B matrix to make it A 4 cross 5 in dimension. So, for that we will use the command call np.transpose for the B matrix and we will store it back in the same matrix B.

So, when we perform the matrix multiplication for this matrix now we will be getting an output yes. Now our A matrix is 5 cross 4 and now we have changed the B matrix to 4 cross 5. So, the dimensions are equal, so now, it has gives an output. So, it is 134, 158, 182, 206, 230 etcetera. So, you can also use numpy.matmul and also the at operator it can be used for the matrix multiplication as well.

(Refer Slide Time: 17:07)



Matrix multiplication

- numpy.multiply()- performs elementwise multiplication between two matrices
- Syntax: numpy.multiply(matrix_1,matrix_2)

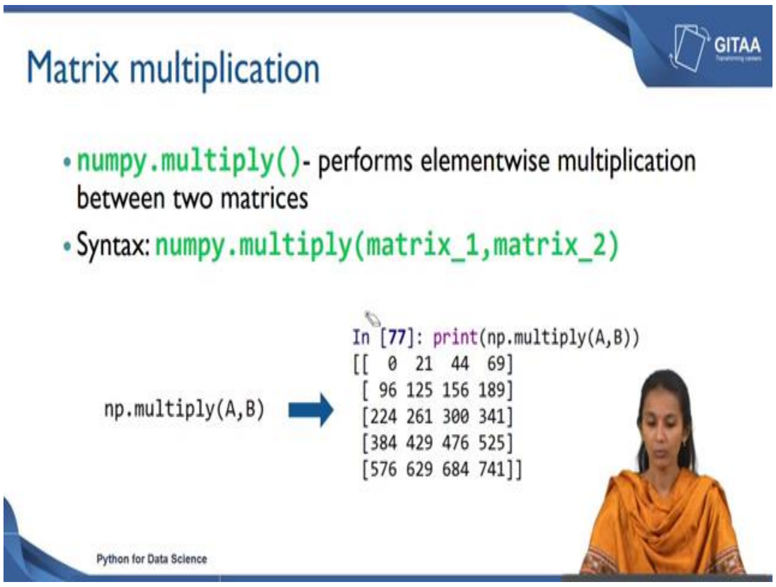
```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

Let us say if you wanted to perform element wise multiplication then there is a command call numpy.multiply it performs the element wise. Syntax is numpy.multiply so inside the parenthesis you have to specify the matrix 1 name followed by the matrix 2 name.

So, this is a matrix which we have created earlier before transposing. So, this is our A matrix and this is our B matrix. Now we will perform the element wise multiplication for this both the matrix.

(Refer Slide Time: 17:41)



Matrix multiplication

- `numpy.multiply()`- performs elementwise multiplication between two matrices
- Syntax: `numpy.multiply(matrix_1,matrix_2)`

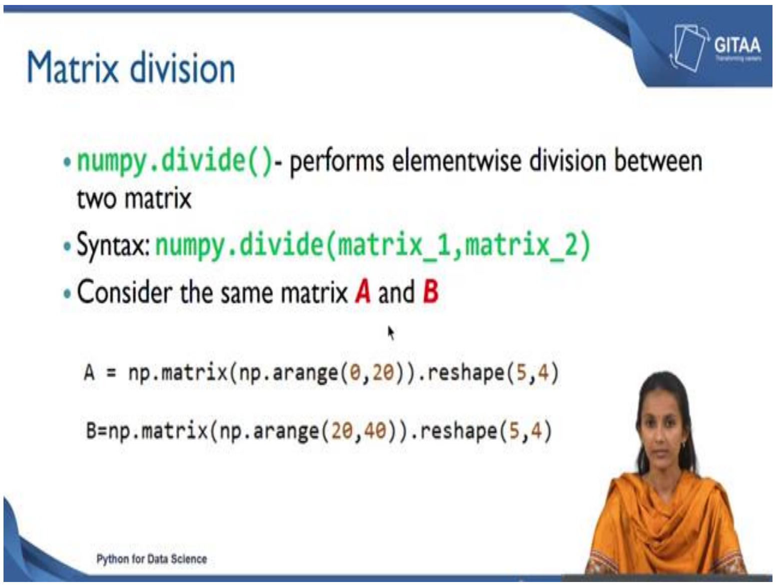
np.multiply(A,B)

```
In [77]: print(np.multiply(A,B))
[[  0  21  44  69]
 [ 96 125 156 189]
 [224 261 300 341]
 [384 429 476 525]
 [576 629 684 741]]
```

Python for Data Science

So, the command is np.multiply(A,B). So, it has been multiplied so 0 into 20 then it will be 1 into 21. So, you will be getting an output of like this 0, 21, 44, 69 and etcetera.

(Refer Slide Time: 18:00)



Matrix division

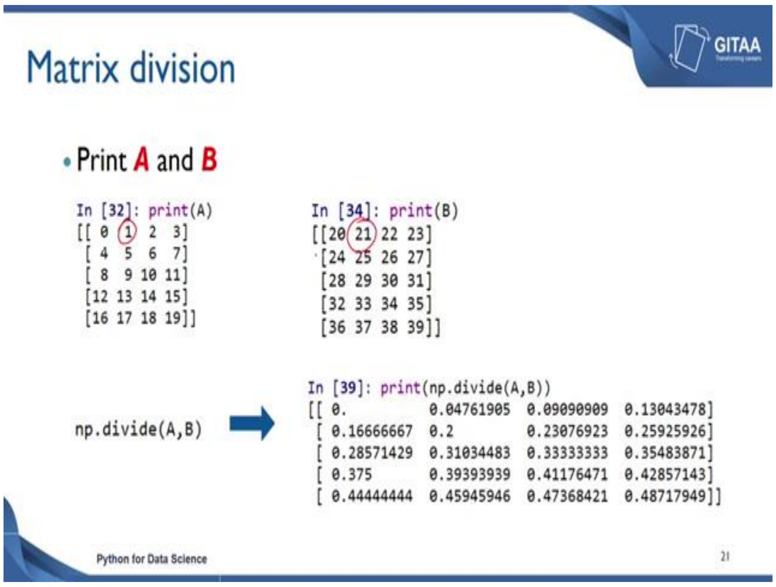
- `numpy.divide()`- performs elementwise division between two matrix
- Syntax: `numpy.divide(matrix_1,matrix_2)`
- Consider the same matrix **A** and **B**

```
A = np.matrix(np.arange(0,20)).reshape(5,4)

B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Python for Data Science

Let us see how to do the matrix division numpy.divide it performs basically the element wise division between the two matrices. So, the syntax is numpy.divide again you have to specify the matrix 1 name followed by the matrix 2 name. So, we will consider the same matrix A and B. So, this our matrix which we have created earlier and this is our B matrix.

(Refer Slide Time: 18:28)



Matrix division

- Print ***A*** and ***B***

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

np.divide(A,B)

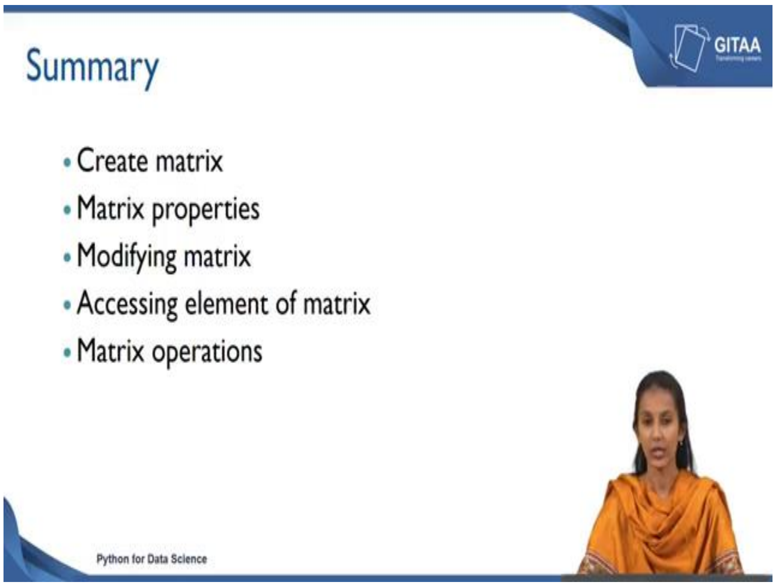
```
In [39]: print(np.divide(A,B))
[[ 0.          0.04761905  0.09090909  0.13043478]
 [ 0.16666667  0.2         0.23076923  0.25925926]
 [ 0.28571429  0.31034483  0.33333333  0.35483871]
 [ 0.375       0.39393939  0.41176471  0.42857143]
 [ 0.44444444  0.45945946  0.47368421  0.48717949]]
```

Python for Data Science 21

So let us print A and B. So, A has values 0, 1, 2, 3 till 19 and B has values 20, 21, 23 till 39. So both are 5 cross 4 matrix. So, we will perform an element wise division. So, the command is np.divide(A,B). So, it has to return values when we print. So, we will be getting a value like this.

So, first it takes 0, first element of the first matrix and first element of the second matrix it is 0 divided by 20 and next one is it will be 1 divided by 21. So, it performs element wise.

(Refer Slide Time: 19:15)



Summary

- Create matrix
- Matrix properties
- Modifying matrix
- Accessing element of matrix
- Matrix operations

Python for Data Science

So, let us summarize. So, first we saw how to create a matrix so you can use a command call matrix. So, before creating a matrix you have to import the numpy package and then we also saw some of the matrix properties. So, the shape it basically gives the number of rows and number of columns, size it basically returns a number of elements in the matrix we also saw how to modified matrix using the index number.

So, if you wanted to access a particular element or row or a column we saw how to access A elements using the index number and we also saw some of the matrix operation which is addition for addition you have to use the numpy.add for subtraction it is numpy.subtract for matrix multiplication we saw numpy.dot.

So, if you wanted to perform element wise you have to use numpy.multiply and we also saw division which is numpy.divide. So, these are the matrix operation which we saw.