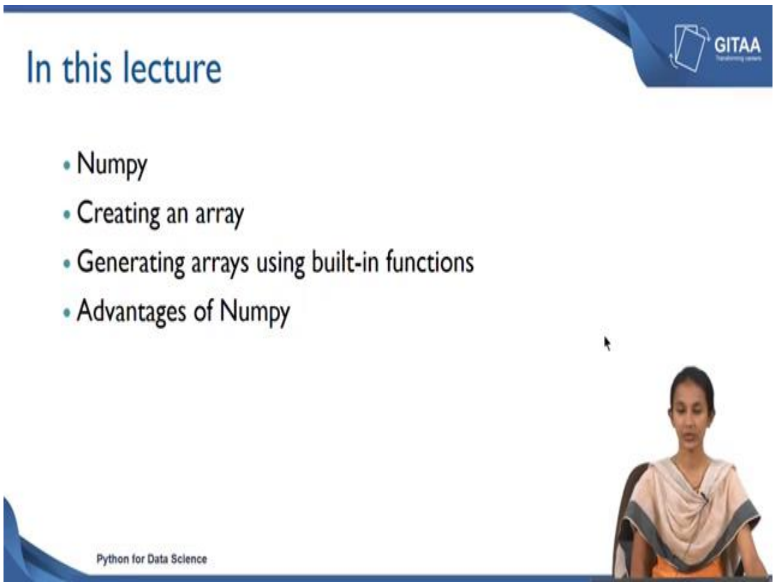
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 12
Numpy Part – 1

(Refer Slide Time: 00:18)



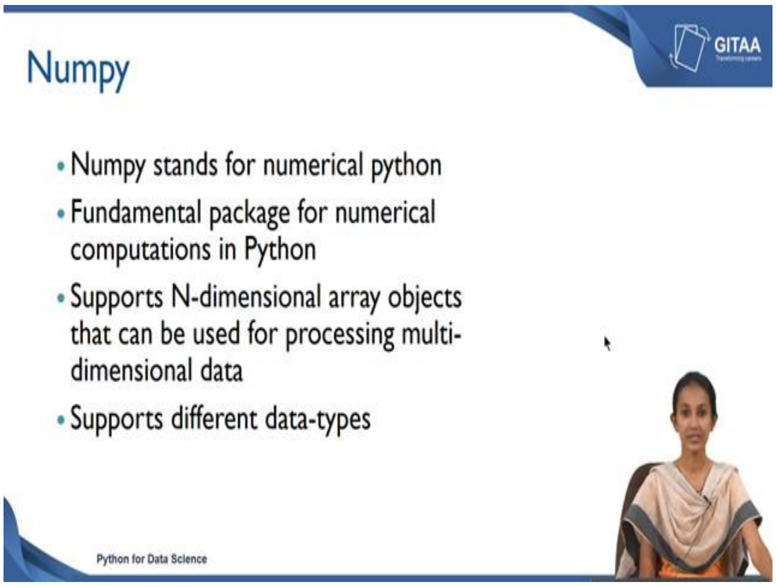
In this lecture

- Numpy
- Creating an array
- Generating arrays using built-in functions
- Advantages of Numpy

GITAA
Python for Data Science

Welcome to the lecture, in this lecture we will see Numpy. How to create an array using the Numpy packages which is available in Python; how to generate arrays using the built in functions, and also we will see some of the advantages of Numpy.

(Refer Slide Time: 00:29)



Numpy

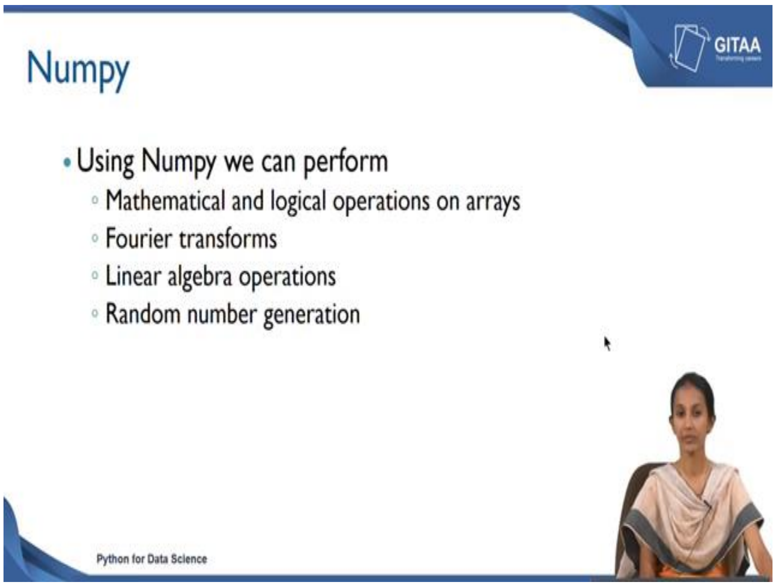
- Numpy stands for numerical python
- Fundamental package for numerical computations in Python
- Supports N-dimensional array objects that can be used for processing multi-dimensional data
- Supports different data-types

GITAA
Python for Data Science

So, first get started. So, Numpy stands for the numerical python. It is a basic fundamental packages which is available in python. It can be used for the numerical computations. So, if you wanted to calculate a sum or mean or median. You can use the Numpy package. There are different packages also available. You have Pandas, SciPy and all. Numpy is basically for the numerical computations. N-d array is a multidimensional container of the objects which of the same type and size.

It supports N dimensional array objects it can be used for processing the multidimensional data as well and also it supports different data types.

(Refer Slide Time: 01:14)



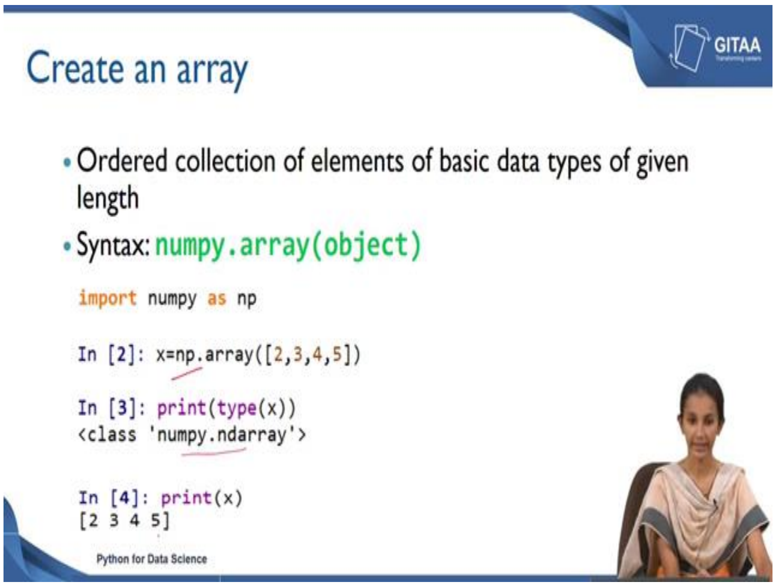
Numpy

- Using Numpy we can perform
  - Mathematical and logical operations on arrays
  - Fourier transforms
  - Linear algebra operations
  - Random number generation

Python for Data Science

Using Numpy we can perform some of the operations. The first operation is; mathematical and logical operations on arrays. So, if we wanted to compare 2 arrays we can do that. It can be used for the Fourier transforms as wells, linear algebra operations, if you wanted to find a rank or determinant to solve the system of equations. We can using Numpy package and Numpy it is also used for the random number generation.

(Refer Slide Time: 01:45)



Create an array

- Ordered collection of elements of basic data types of given length
- Syntax: `numpy.array(object)`

```
import numpy as np

In [2]: x=np.array([2,3,4,5])

In [3]: print(type(x))
<class 'numpy.ndarray'>

In [4]: print(x)
[2 3 4 5]
```

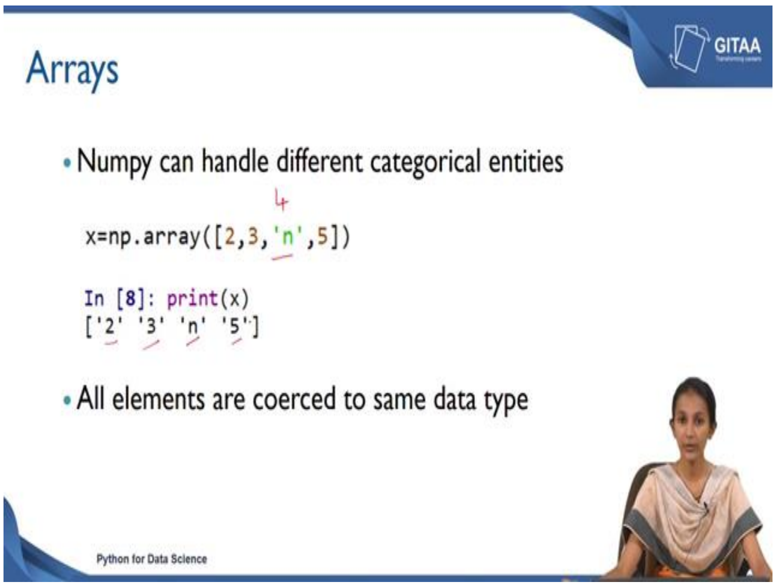
Python for Data Science

So, first we will see how to create an array. Array is basically of ordered collection of elements of the data types. So, the basic syntax is you have to specify the package name.

So, which is a numpy is a package.array and inside the parenthesis you have to specify the object. So, first we need to import the numpy package every time when you run the code.

So, I am going to import numpy as np. So, it basically analyze to the package, np.array inside the parenthesis I am giving the sets of values. So, I am giving 2, 3, 4, and 5 and I am storing it in a variable call x. When you give print(type(x)); type of x basically returns an output of the data type. Here we have declared it as an array it has returns an numpy nd array; so numpy n dimensional array. So, when you give print of x. So, you have 2 3 4 5 so, it is an array.

(Refer Slide Time: 02:47)



Arrays

- Numpy can handle different categorical entities

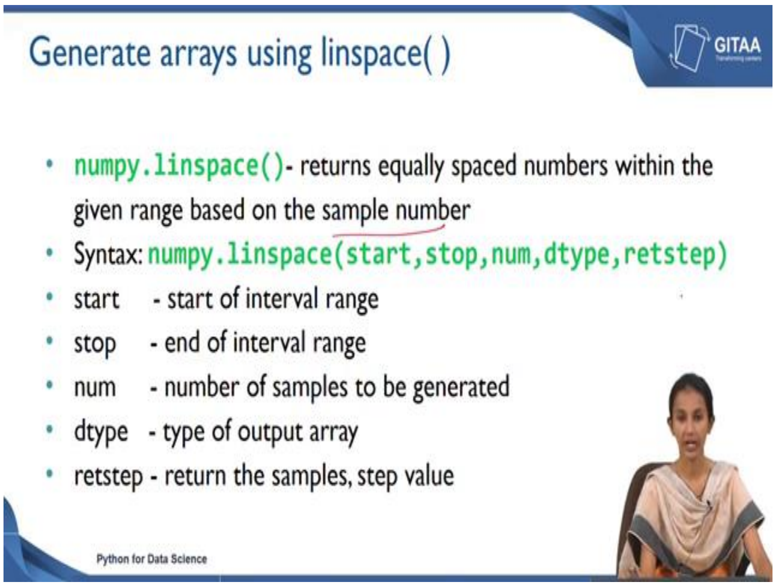
```
          4
x=np.array([2,3,'n',5])

In [8]: print(x)
['2' '3' 'n' '5']
```

- All elements are coerced to same data type

So, Numpy can also handle different categorical entities as well. So, we will take a small example earlier we had 2, 3, 4 and 5. Now, I am replacing 4 with n. So, I am giving inside the single quotes. So, it has been an array, now and I am storing again it in the variable call x. So, when you give print of x. So, each value will be converted to a same data type.

So, we had n with the inside the single quotes all the other values will also will be replace with the single quotes. So, their elements all the elements will be coerced to the same data type. Here we will created arrays using the built in function that is np.array we have given set of values.

(Refer Slide Time: 03:43)


Generate arrays using linspace( )

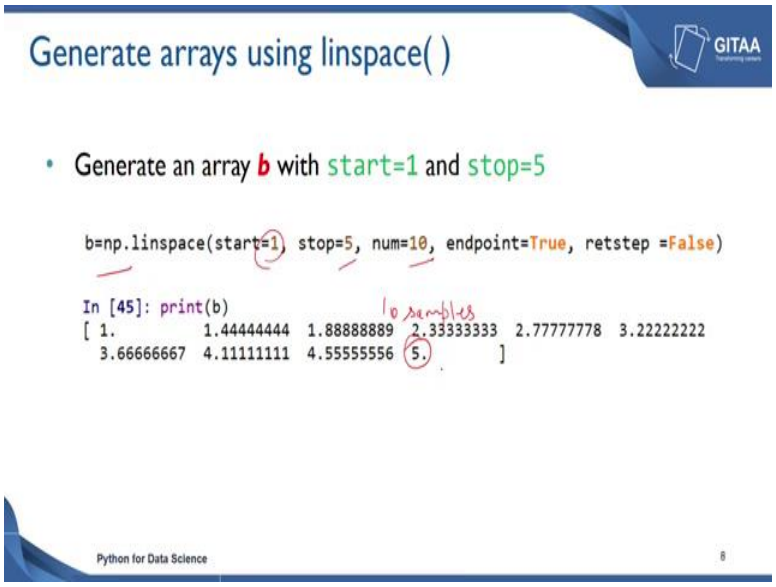
- numpy.linspace()- returns equally spaced numbers within the given range based on the sample number
- Syntax: numpy.linspace(start,stop,num,dtype,retstep)
- start - start of interval range
- stop - end of interval range
- num - number of samples to be generated
- dtype - type of output array
- retstep - return the samples, step value

GITAA
Python for Data Science

So, if you wanted to randomly generate arrays. So, there are also some of the built in functions which is available, we will see one by one. First is generate arrays using the linspace. So, numpy.linspace is an built function which is available. It returns an equally space numbers based on the sample number.

So, syntax is numpy.linspace(). So, first is start it is the start for the interval range, next is stop end of the an interval range, next is num. So, number of samples to be generated. So, if you wanted to generate 100 samples then you can specify the num is equal to 100. Next is dtype; type of an output array return step. So, the last one is what it does is it returns a sample and as well as a step value which has been incremented.

(Refer Slide Time: 04:37)



Generate arrays using linspace( )

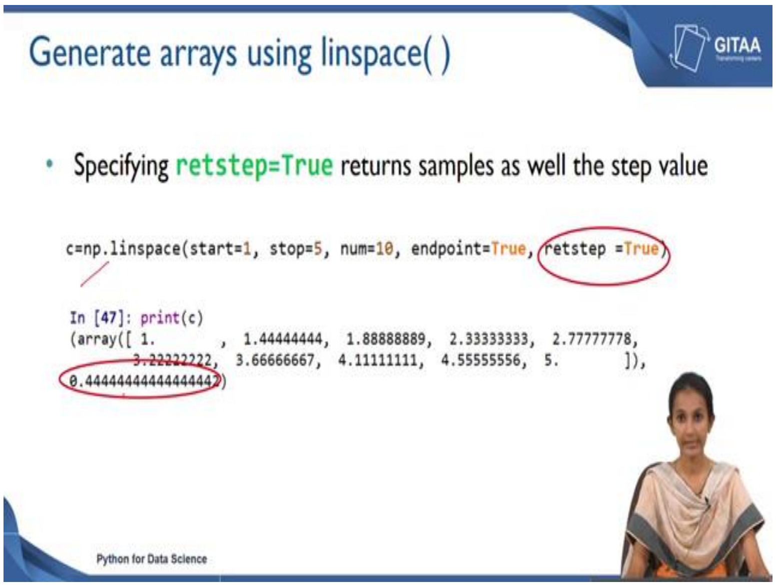
- Generate an array **b** with start=1 and stop=5

```
b=np.linspace(start=1, stop=5, num=10, endpoint=True, retstep =False)

In [45]: print(b)
[ 1.          1.44444444  1.88888889  2.33333333  2.77777778  3.22222222
  3.66666667  4.11111111  4.55555556  5.        ]
```

So, we will generate an array b with the start value of 1 and stop value of 5. So, here np.linspace start value 1, stop value 5, num is number of samples to be generated. So, here I wanted to generate 10 samples. So, next one is the end point. So, if you wanted to include the last value which is 5 or not. So, if you give end point equal to True; 5 will be included in the samples, restep equal to False.

So, if you give False it returns only the samples and not the increment value. So, when you print the output you will be getting 10 samples. So, first value is 1, second value is 1.444, 1.88, 2 and last value is 5. So, if you do not give end point equal to True; 5 will not be included in the samples. So, this is it basically returns 10 samples and the last value as been included in the samples.

(Refer Slide Time: 05:50)



Generate arrays using linspace( )

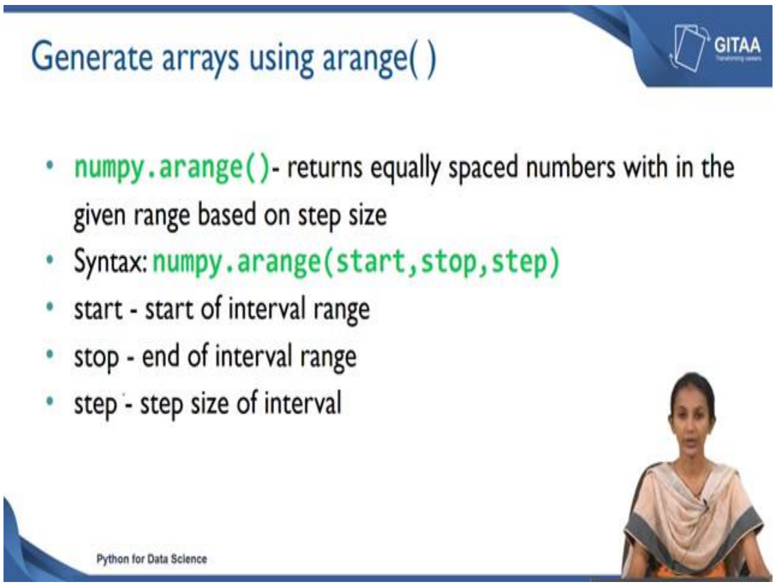
- Specifying retstep=True returns samples as well the step value

```
c=np.linspace(start=1, stop=5, num=10, endpoint=True, retstep =True)

In [47]: print(c)
(array([ 1.        ,  1.44444444,  1.88888889,  2.33333333,  2.77777778,
        3.22222222,  3.66666667,  4.11111111,  4.55555556,  5.        ]),
0.44444444444444442)
```

So, we will take an another case with restep is equal to True, np.linspace the same start value, stop value, number; which is the number of samples end point equal to True. So, till this it is a same command.

Here in the last one. So, I am given return step equal to True instead of False. So, the output will be first value 1, 1.44, 1.88 till 5 will be the same output. So, in the last case when you see, so we have got a 0.444 this is basically an increment value. So, if you give returns step equal to True; it basically returns the samples as well as a step value in which it has been incremented.

(Refer Slide Time: 06:32)



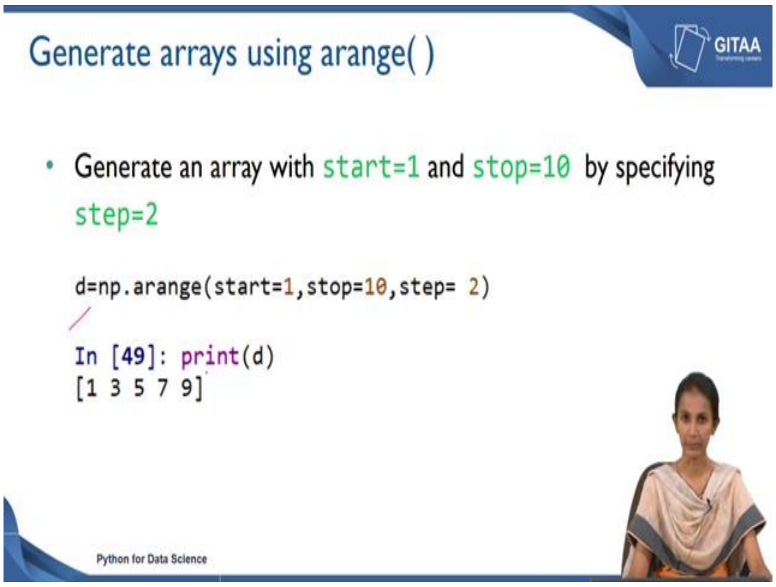
Generate arrays using arange( )

- numpy.arange()- returns equally spaced numbers with in the given range based on step size
- Syntax: numpy.arange(start,stop,step)
- start - start of interval range
- stop - end of interval range
- step - step size of interval

Python for Data Science

So, next is if you wanted to generate arrays based on the step value we can use the arange command. So, it numpy.arrange; it returns equally spaced numbers based on the step size. So, the syntax is numpy.arange(), start value have to specify which is in start of an interval range, stop value and which is and basically an end value and then step size in which you wanted to incremented.

(Refer Slide Time: 07:04)



Generate arrays using arange( )

- Generate an array with start=1 and stop=10 by specifying step=2

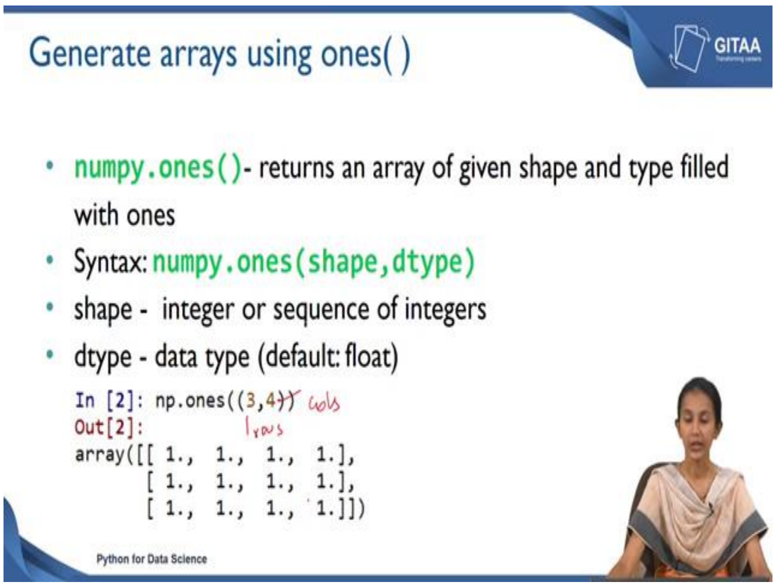
```
d=np.arange(start=1,stop=10,step= 2)

In [49]: print(d)
[1 3 5 7 9]
```

Python for Data Science

So, we will generate an array with the start value of 1 and stop the value of 10 and also the increment value is step size is 2. So, our command will be np.arange(start=1, stop=10, step=2). And, I am storing it in a variable call d. So, when you print d. So, you will be getting an output of 1 3 5 7 and 9. So, which means, so the first start value we have specified as 1. So, in steps of 2 it has to be incremental right.

So, 1+2 it will be 3; 3+2 5; 5+2 7; 7+2 9. So, the stop value is 10. So, after that it will not be able to incremented. So, it returns the 5 samples.

(Refer Slide Time: 07:50)



Generate arrays using ones( )

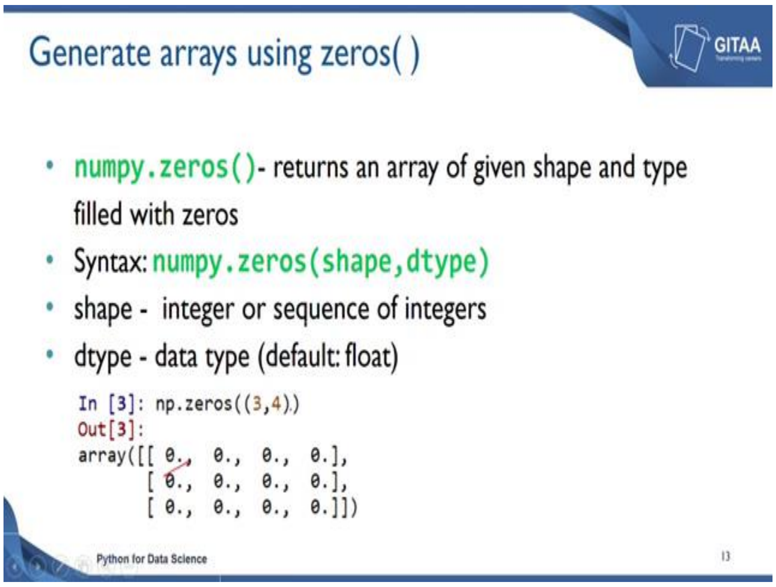
- numpy.ones()- returns an array of given shape and type filled with ones
- Syntax: numpy.ones(shape,dtype)
- shape - integer or sequence of integers
- dtype - data type (default: float)

```
In [2]: np.ones((3,4))
Out[2]:
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

Let say if you wanted to arrays filled with only with ones. So, you can use the command call numpy.ones and it returns an array of given shapes. So, if you specify the shape, number of rows and number of columns it returns an array fully filled with ones, but that the syntax is numpy.ones shape which is a integer or sequence of integer, dtype which is basically the data type.

This default one always is float data type. So, when you give np.ones(3,4), which means 3 stands for rows, 4 stands for number of columns. So, it has generated an array with 3 rows and 4 columns fully filled with ones. So, you can similarly you can also generate arrays using zeros as well, numpy.zeros is an function.

(Refer Slide Time: 08:39)



Generate arrays using zeros( )

- numpy.zeros()- returns an array of given shape and type filled with zeros
- Syntax: numpy.zeros(shape,dtype)
- shape - integer or sequence of integers
- dtype - data type (default: float)

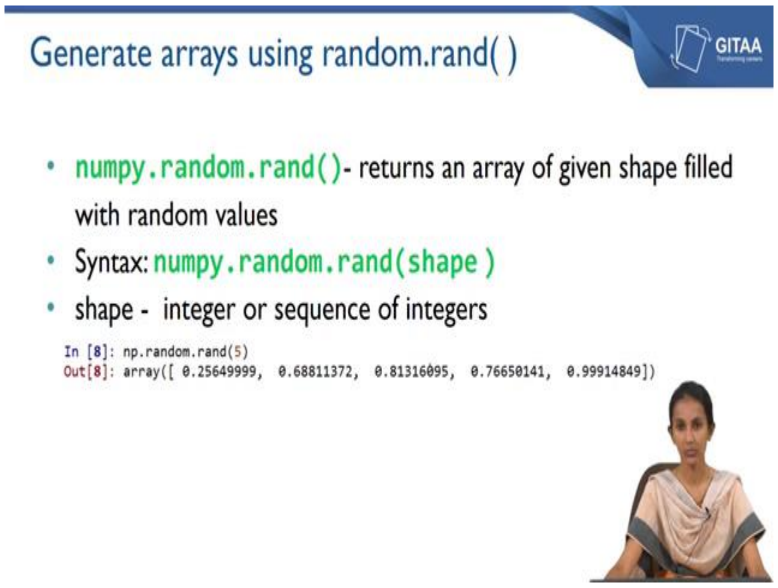
```
In [3]: np.zeros((3,4))
Out[3]:
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

Python for Data Science 13

So, it returns an array of the given shapes. So, if you specify the shape size that is dimensions. And, then so it returns and type filled with zeros. The syntax is numpy.zeros shape again which is an integer or sequence of integer dtype which is the basically a data type.

So, the default is always float when you give np.zeros. 3 stands for rows, 4 stands for columns. So, when you see the output will be 0 dot; which is an float value. So, you can also change the data type as well. So, you can specify in the np.zeros 3 comma 4 and then you can specify the data type it can be an integer as well.

(Refer Slide Time: 09:28)



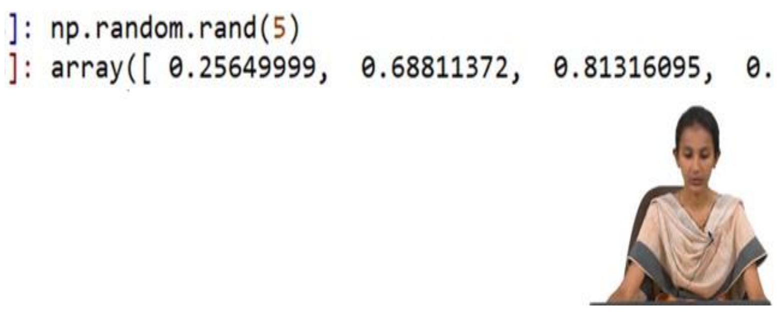
Generate arrays using random.rand( )

- numpy.random.rand()- returns an array of given shape filled with random values
- Syntax: numpy.random.rand(shape)
- shape - integer or sequence of integers

```
In [8]: np.random.rand(5)
Out[8]: array([ 0.25649999,  0.68811372,  0.81316095,  0.76650141,  0.99914849])
```

Let say if you wanted to generate random values. If you do not want to start a value from 1 to 10 or 1 to 100 you wanted a random values, then you can use the random.rand function which is available in python. So, numpy.random.rand() returns an array with the specified dimension with the random values. The syntax is numpy.random.rand(), with shape that can be an integer or sequence of integer.

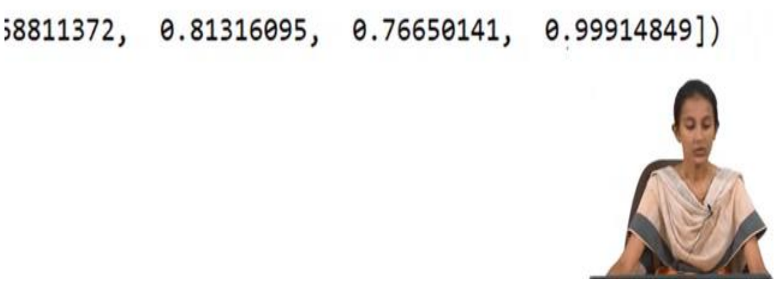
(Refer Slide Time: 09:56)



```
]: np.random.rand(5)
]: array([ 0.25649999,  0.68811372,  0.81316095,  0.
```

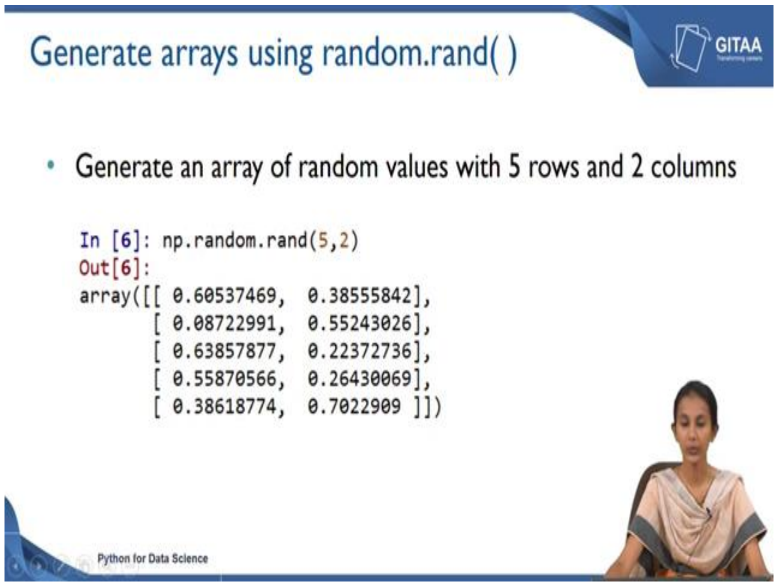
So, np.random.rand. So, I wanted to generate 5 values. So, it randomly generate 5 values so the array basically. So, the first value 0.2564, second one is 0.68, third one is 0.8131, fourth one is 0.7665, 0.999. So, it has been randomly generated 5 values. So, if you specify the number of rows and columns also you can get the random values.

(Refer Slide Time: 10:13)



```
58811372,  0.81316095,  0.76650141,  0.99914849])
```

(Refer Slide Time: 10:29)



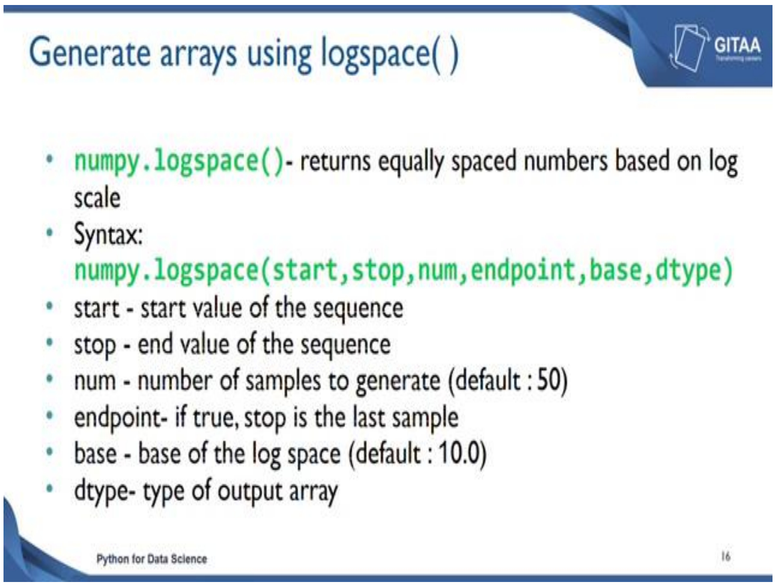
Generate arrays using random.rand( )

- Generate an array of random values with 5 rows and 2 columns

```
In [6]: np.random.rand(5,2)
Out[6]:
array([[ 0.60537469,  0.38555842],
       [ 0.08722991,  0.55243026],
       [ 0.63857877,  0.22372736],
       [ 0.55870566,  0.26430069],
       [ 0.38618774,  0.7022909 ]])
```

Let say if you wanted to generate an array of random values with 5 rows and 2 columns, you can specify that as well. So, np.random.rand(5,2) which basically stands for 5 rows and 2 columns; so, it has been created an array with 5 rows and 2 columns and it as random values. So, first value is 0.605, second one is 0.3855, 0.0 8 and let the last value 0.702290. So, it has been randomly generated values.

(Refer Slide Time: 11:06)



Generate arrays using logspace( )

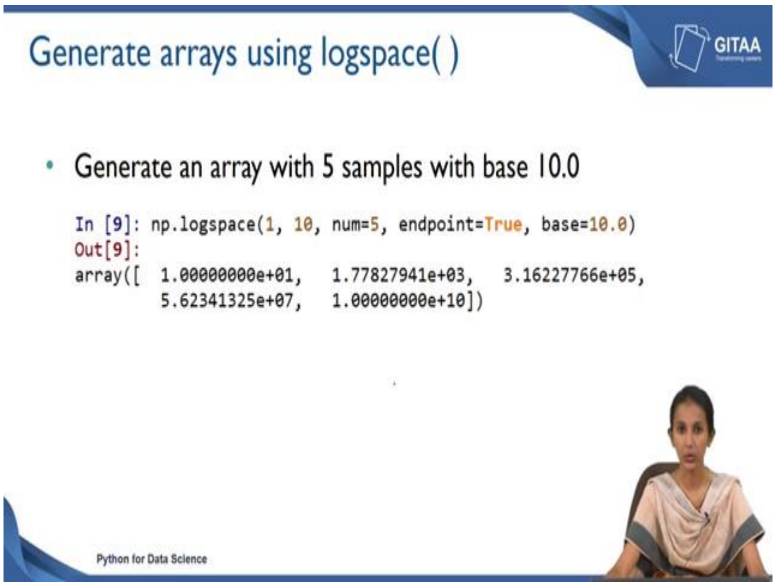
- numpy.logspace()- returns equally spaced numbers based on log scale
- Syntax:
  numpy.logspace(start,stop,num,endpoint,base,dtype)
- start - start value of the sequence
- stop - end value of the sequence
- num - number of samples to generate (default : 50)
- endpoint- if true, stop is the last sample
- base - base of the log space (default : 10.0)
- dtype- type of output array

Python for Data Science 16

Next if you wanted to generate arrays using the logspace as well using the logs then you can also do that. So, the numpy.logspace() it returns an again equally space numbers based on the log values which you specify. So, the command is the numpy.logspace start by you have to specify the start value of the sequence arange stop value number of samples to be generated.

So, the default in this case is 50 and then you have to specify the endpoint so, again its binary true or false. So, if you true the last sample will be included in the output, base the base values which you wanted to give the log base value, the default is 10.0, dtype type of the output array.

(Refer Slide Time: 12:00)



Generate arrays using logspace( )

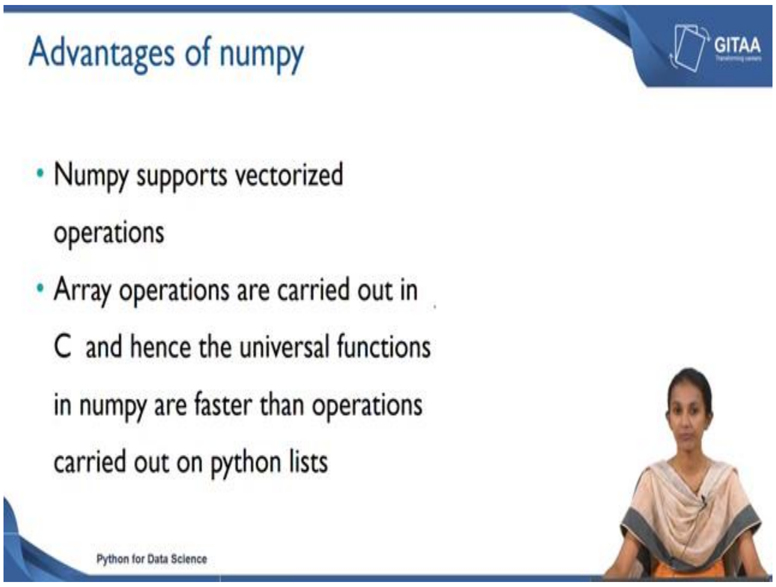
- Generate an array with 5 samples with base 10.0

```
In [9]: np.logspace(1, 10, num=5, endpoint=True, base=10.0)
Out[9]:
array([  1.00000000e+01,   1.77827941e+03,   3.16227766e+05,
         5.62341325e+07,   1.00000000e+10])
```

So, we will generate an array with 5 samples the base value is 10.0. So, d I am taking the default value. So, the np.logspace start value is 1 stop value is 10. So, if you wanted to generate 5 samples you have to specify num is equal to 5 and then endpoint equal to true which means a large sample has to be included in the output array.

And then base equal to 10.0. So, if you do not specify base is equal to 10.0 it is an automatically it takes 10.0, you can also specify the different base as well. So, it returns the 5 samples. So, the first value is 1.000e+01 ie, 10 and the last value is 1.000e+10 ie, 10 power 10. So, it returns the 5 samples in the array format.

(Refer Slide Time: 12:55)



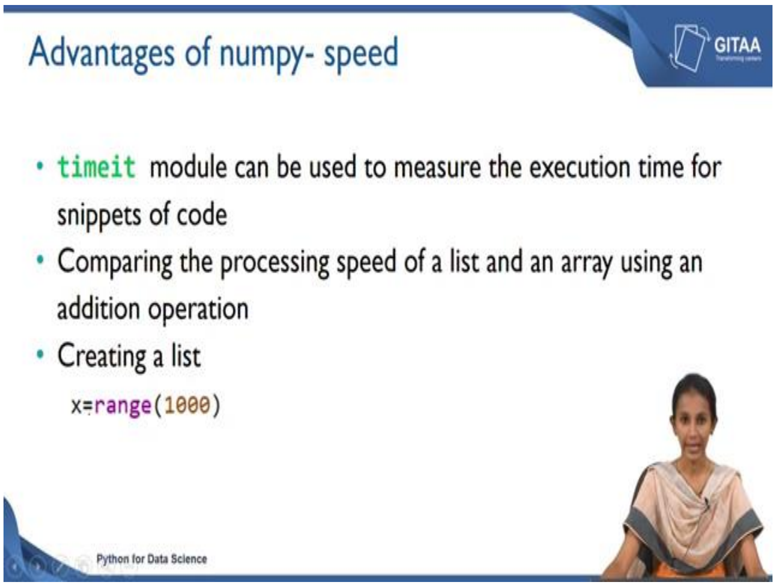
Advantages of numpy

- Numpy supports vectorized operations
- Array operations are carried out in C and hence the universal functions in numpy are faster than operations carried out on python lists

GITAA
Python for Data Science

Let us look at the some of the advantages of numpy. Numpy supports the vectorized operations. Array operations they are carried out in C in the backend and then they are executed in python. So, numpy performs faster than the list.

(Refer Slide Time: 13:10)



Advantages of numpy- speed

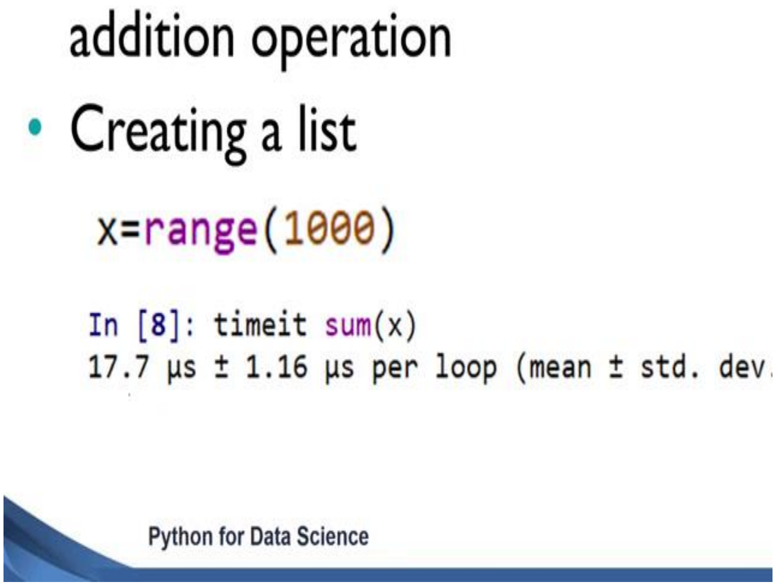
- timeit module can be used to measure the execution time for snippets of code
- Comparing the processing speed of a list and an array using an addition operation
- Creating a list

```
x=range(1000)
```

GITAA
Python for Data Science

So, first we will look at the speed. So, timeit is a module which is available in python. So, it can be used to measure the execution time which is basically a runtime for the snippets of codes. We will take an example, so we will compare the processing speed of a list and as well as an array using the addition operation. So, the first example is I will create a list using the range. So, range of 1000 and I am storing it in a variable call x. When you give timeit of sum of x it returns an output of 17.7 microseconds.

(Refer Slide Time: 13:42)



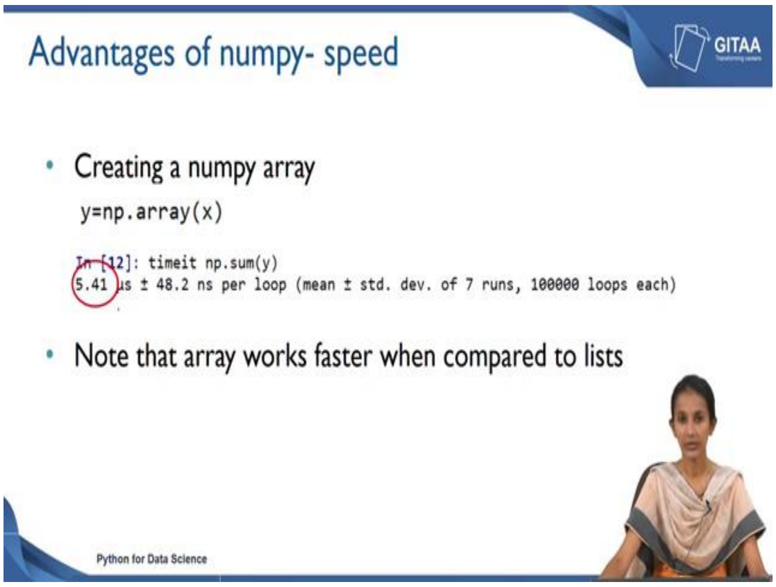
addition operation

- Creating a list

```
x=range(1000)

In [8]: timeit sum(x)
17.7 µs ± 1.16 µs per loop (mean ± std. dev.
```

Python for Data Science

(Refer Slide Time: 13:49)



Advantages of numpy- speed

- Creating a numpy array

```
y=np.array(x)

In [12]: timeit np.sum(y)
5.41 µs ± 48.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

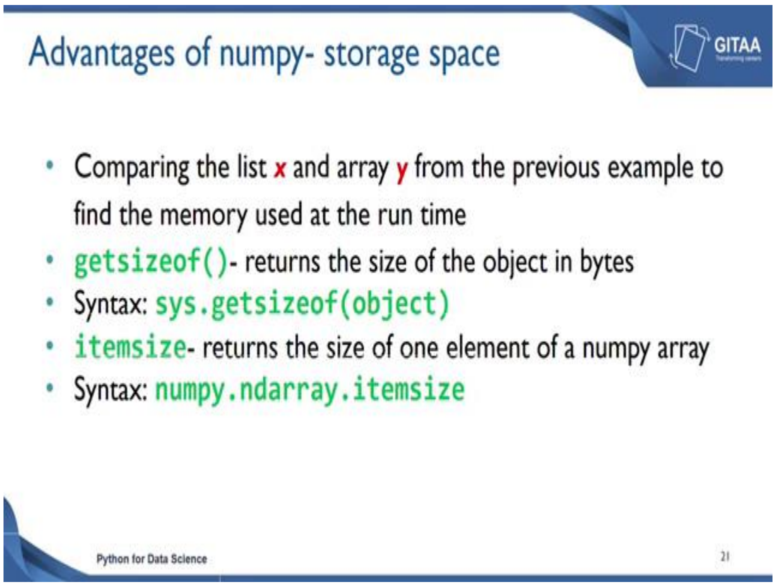
- Note that array works faster when compared to lists

Python for Data Science

So, next we will create a numpy array we will also calculate the runtime speed for this as well and then we will compare which one is faster. So, creating a numpy array; so, the x; x is equal to range of 1000 np.I am converting into an array and I am storing in a variable called y. So, when you give timeit np.sum(y), it returns an output of 5.41 microseconds.

So, the array works faster when compared with lists. So, this is a first advantage of numpy.

(Refer Slide Time: 14:25)



Advantages of numpy- storage space

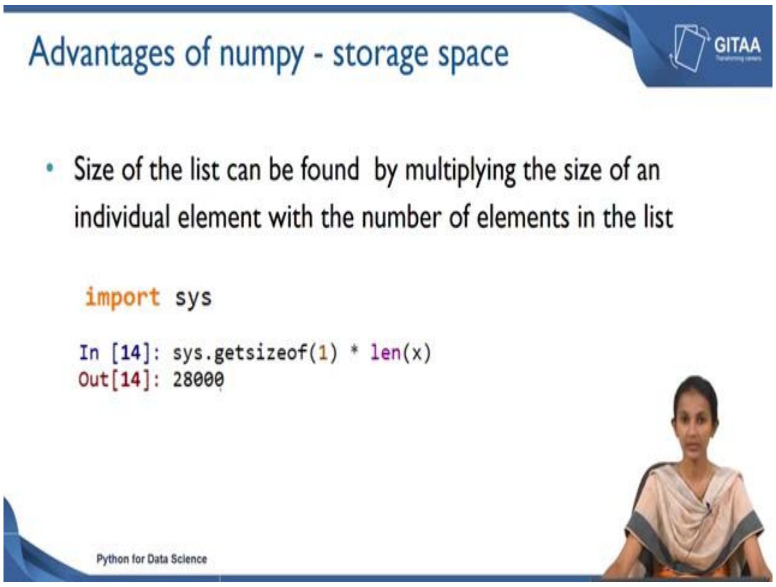
- Comparing the list **x** and array **y** from the previous example to find the memory used at the run time
- `getsizeof()`- returns the size of the object in bytes
- Syntax: `sys.getsizeof(object)`
- `itemsize`- returns the size of one element of a numpy array
- Syntax: `numpy.ndarray.itemsize`

Python for Data Science 21

We will also look at this storage space for both list as well as the numpy array. So, we will compare the list x and array y from the previous example and we will calculate the memory used at the runtime. So, getsizeof() which is an built in function, it returns the size of the object in bytes.

So, the syntax is sys it is an built in package it is used for system specific parameters. Getsizeof() inside the parentheses specify the object. So, it basically return the size of the objects in bytes. So, we have next one is itemsize it returns size of the 1 element of the numpy array. So, items size it can be used for the numpy array. So, if the syntax is numpy.nd array which is n dimensional array.itemsize.

(Refer Slide Time: 15:19)



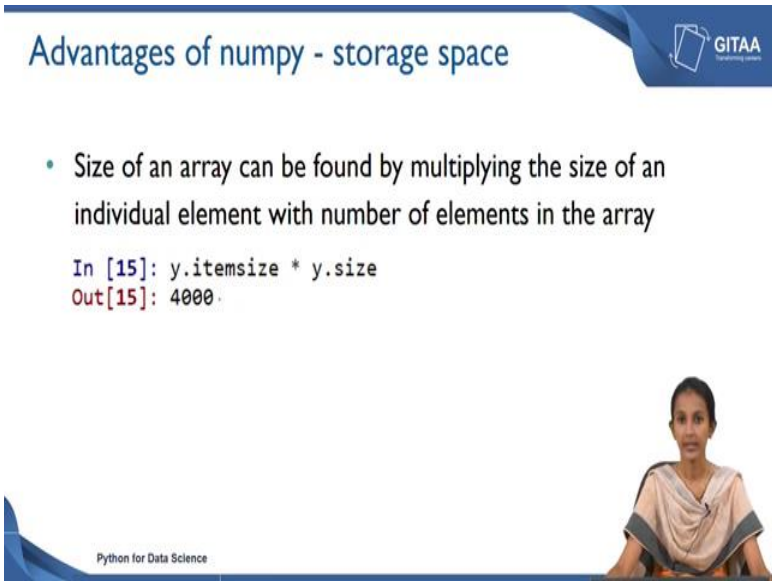
Advantages of numpy - storage space

- Size of the list can be found by multiplying the size of an individual element with the number of elements in the list

```
import sys

In [14]: sys.getsizeof(1) * len(x)
Out[14]: 28000
```

First we will calculate the storage space for the list and then we will do it for the array. So, the size of the list it can be found by multiplying the first we will take the size of an individual element and then with the number of elements in the list. So, for that we need to import a package called import sys. So, it is for system specific parameters. So, using this sys; sys.getsizeof(1), this multiplied with length of the elements in x. So, it returns an output of 28000 for the list it to 28000 bytes.

(Refer Slide Time: 16:02)



Advantages of numpy - storage space

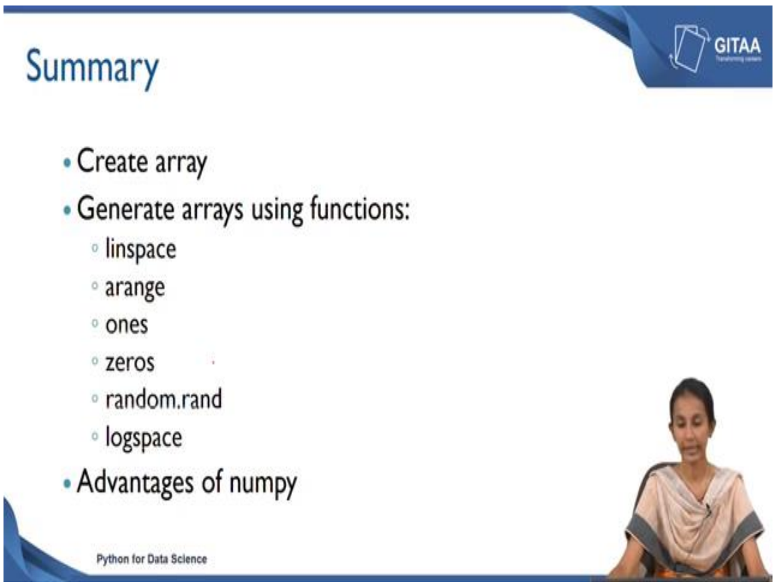
- Size of an array can be found by multiplying the size of an individual element with number of elements in the array

```
In [15]: y.itemsize * y.size
Out[15]: 4000
```

Similarly, we will calculate for the array as well. So, the size of an array it can be again found by multiplying the size of an individual element with the number of elements. So, if you have 10 number of elements. So, what it does it first it calculate size of an 1 individual elements with the 10 elements in the array. So, if you give y.so, the y is an array which we have stored already.

So, y.itemsize * y.the length of the size which is the number of elements which present in the array. So, for this it returns an output of 4000 which means 4000 bytes. For list it tooks 28000 bytes, for array it only tooks 4000 bytes. So, this is an another advantage of using the numpy compared to list.

(Refer Slide Time: 16:54)



Summary

- Create array
- Generate arrays using functions:
  - linspace
  - arange
  - ones
  - zeros
  - random.rand
  - logspace
- Advantages of numpy

GITAA
Python for Data Science

So, let us summarize. First we saw how to create an array and we also saw some of the built in functions to generate arrays. First one is linspace, it generate arrays based on the sample number we have to specify the start value, end value. Next we also saw arange; it generates array based on the step size. So, which is an increment value and we also saw ones. So, ones it basically generate arrays filled with fully filled with ones.

So, if we wanted to fully filled with zeros. So, numpy.zeros you can use, we also saw how to generate random values. So, random.rand it generate random values next we also saw logspace. We can specify that base value and you can generate arrays, we also saw some of the advantages of Numpy. So, the first one is we create an array, we also created a list we compared both and we also saw the storage spaces. We also saw some of the advantages of Numpy. So, we saw how to calculate the speed and we also saw how to calculate the storage space. So, Numpy uses the less bytes and also the less time to run the snippets of codes.

Thank you.