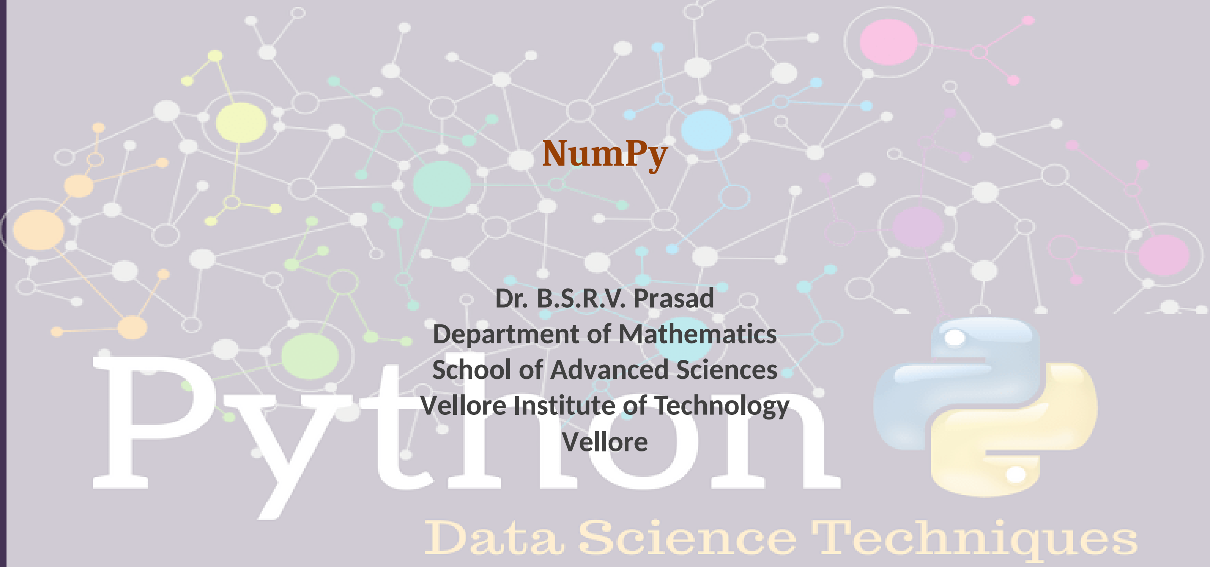
NumPy

Dr. B.S.R.V. Prasad
Department of Mathematics
School of Advanced Sciences
Vellore Institute of Technology
Vellore



srvprasad.bh@gmail.com (Personal)
srvprasad.bh@vit.ac.in (Official)

NumPy: Creating and manipulating numeric data

NumPy

- is a general-purpose array-processing package.
- provides a high-performance multidimensional array object, and tools for working with these arrays.
- fundamental package for scientific computing with Python.
- contains various features including these important ones:
  - A powerful N-dimensional array object
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities

Import Conventions

The recommended convention to import **NumPy** is:

```python
import numpy as np
```

Arrays in NumPy

NumPy’s main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called axes. The number of axes is rank.
- NumPy’s array class is called ndarray. It is also known by the alias **array**.

1-D array construction

```python
import numpy as np
a = np.array([0, 1, 2, 3])
print('Array a =',a)
print('Array a dimension = ',a.ndim)
print('Array a shape = ',a.shape)
print('Array a size (total no.of elements) = ',a.size)
print('Array a data type = ',a.dtype)
print('Array a length = ',len(a))
```

2-D array construction

```python
import numpy as np
a = np.array([[0, 1, 2, 3.1],[4, 5, 6, 7]])
print('Array a =',a)
print('Array a dimension = ',a.ndim)
print('Array a shape = ',a.shape)
print('Array a size (total no.of elements) = ',a.size)
print('Array a data type = ',a.dtype)
print('Array a length = ',len(a)) #Returns the size of the first dimension
```

3-D array construction

```python
import numpy as np
a = np.array([[[0, 1, 2, 3],[4, 5, 6, 7]],[[8, 9, 10, 11],[12, 13, 14, 15]]])
print('Array a =',a)
print('Array a dimension = ',a.ndim)
print('Array a shape = ',a.shape)
print('Array a size (total no.of elements) = ',a.size)
print('Array a data type = ',a.dtype)
print('Array a length = ',len(a)) #Returns the size of the first dimension
```

NumPy Array Creation

There are other various ways to create arrays in NumPy.

- We can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.
- NumPy offers several functions to create arrays with initial placeholder content. For example: np.zeros, np.ones, np.full, np.empty, etc.
- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
  - **arange:** returns evenly spaced values within a given interval. step size is specified.
  - **linspace:** returns evenly spaced values within a given interval. num no. of elements are returned.

NumPy Array Creation

- **Reshaping array:** We can use reshape method to reshape an array.
  - Consider an array with shape $(a1, a2, a3, \ldots, aN)$. We can reshape and convert it into another array with shape $(b1, b2, b3, \ldots, bM)$.
  - The only required condition is: $a1 \times a2 \times a3 \ldots \times aN = b1 \times b2 \times b3 \ldots \times bM$. (i.e original size of array remains unchanged.)
- **Flatten array:** We can use flatten method to get a copy of array collapsed into one dimension. It accepts order argument. Default value is ‘C’ (for row-major order). Use ‘F’ for column major order.

NumPy Array Creation: Demonstration

```python
# Python program to demonstrate
# array creation techniques
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n”, a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n”, b)
```

NumPy Array Creation: Demonstration

```python
# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n”, c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s. The Array type is complex:\n”, d)

# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n”, e)
```

NumPy Array Creation: Demonstration

```python
# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n”, f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between 0 and 5:\n”, g)
```

NumPy Array Creation: Demonstration

```
# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n”, arr)
print ("Reshaped array:\n”, newarr)
```

NumPy Array Creation: Demonstration

```python
# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n”, arr)
print ("Fattened array:\n”, flarr)
```

NumPy Arrays

In NumPy the arrays data type will be determined automatically

```python
a = np.array([1, 2, 3])
print(a.dtype)

a = np.array([1.0, 2, 3])
print(a.dtype)

a = np.array([1+1j, 2.0+2j, 3])
print(a.dtype)

a = np.array([True, False, False, True, False])
print(a.dtype)

a = np.array(['Vellore', 'Chennai', 'Vijayawada', 'Delhi', 'Bengaluru'])
print(a.dtype)
```

Indexing and Slicing

```
>>> a = np.arange(10)
>>> print(a)
>>> print(a[0]+a[2])
```

Please note that the index in Python starts from 0. The usual python idiom for reversing a sequence is supported.

```
>>> arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
>>> print(arr[1,2])
```

```
>>> arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
>>> print(arr[0, 1, 2]) #Prints the value 6
```

Indexing and Slicing
Slicing Arrays

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don’t pass start its considered 0
- If we don’t pass end its considered length of array in that dimension
- If we don’t pass step its considered 1

Indexing and Slicing

```
a = np.random.rand(5,4)

a[1,1] #List the second row second element

a[:,2] #List all elements in the third column

a[1,:] #List all element in the second row.
a[1] #Similar to above

a[1:5] #List the elements from the second to fifth element.

a[(2,3):(1,2)] #List the [a[2,1], a[3,2]] elements
```

Indexing and Slicing

Negative Indexing

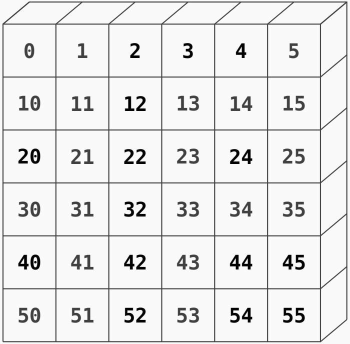
```
>>> a[-3:-1]
>>> array([7, 8])

>>> a[-3:]
>>> array([7, 8, 9])

>>> arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
>>> print('Last element from 2nd dim: ', arr[1, -1])
```

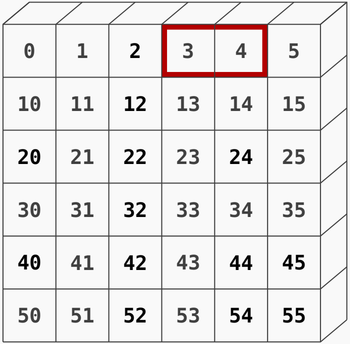
Indexing and Slicing

```
>>> a = np.array(
     [[0,1,2,3,4,5],
    [10,11,12,13,14,15],
    [20,21,22,23,24,25],
    [30,31,32,33,34,35],
    [40,41,42,43,44,45],
    [50,51,52,53,54,55]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

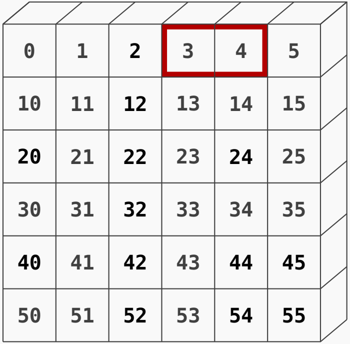
Indexing and Slicing



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

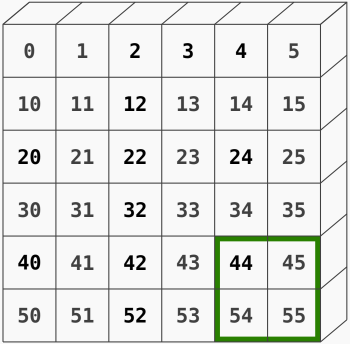
Indexing and Slicing

```
>>> a[0, 3:5]
>>> array([3, 4])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

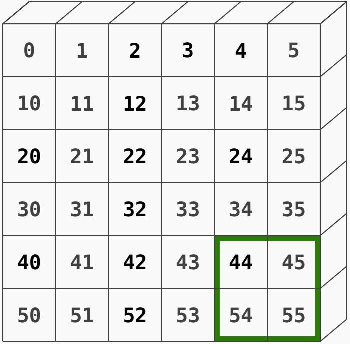
Indexing and Slicing



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

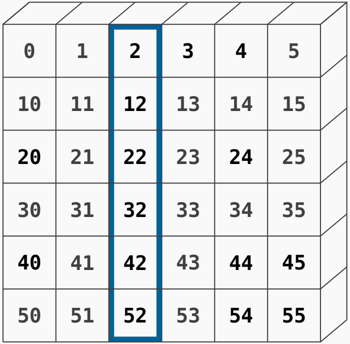
Indexing and Slicing

```
>>> a[4:, 4:]
>>> array([[44, 45],
       [54, 55]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

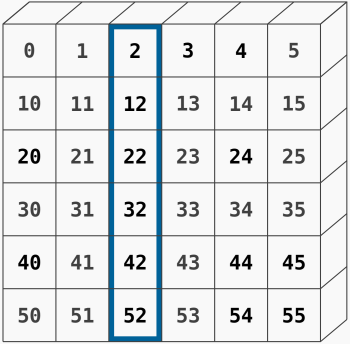
Indexing and Slicing



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

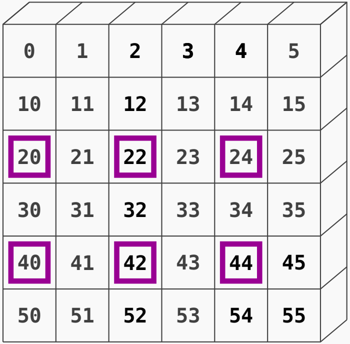
Indexing and Slicing

```
>>> a[:, 2]
>>> array([2, 12, 32, 42,
   52])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

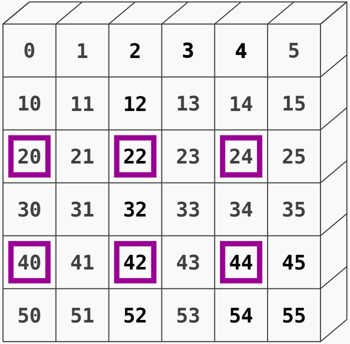
Indexing and Slicing



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Indexing and Slicing

```
>>> a[2::2, ::2]
>>> array([[20, 22, 24],
      [40, 42, 44]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Indexing and Slicing

Please note that Slicing of an array does not make a copy of the array. It just creates a view which refer to the original array. So, changing any elements in the slicing changes the elements in the original array

```
a = np.array([[0,1,2,3,4,5], [10,11,12,13,14,15],
    [20,21,22,23,24,25], [30,31,32,33,34,35],
    [40,41,42,43,44,45], [50,51,52,53,54,55]])
a[:,2] = [1, 1, 1, 1, 1, 1]
print(a)
```

Indexing and Slicing

To avoid this we need to use copy() command

```
a = np.array([[0,1,2,3,4,5], [10,11,12,13,14,15],
    [20,21,22,23,24,25], [30,31,32,33,34,35],
    [40,41,42,43,44,45], [50,51,52,53,54,55]])
b = a[:,2].copy()
b = np.array([1, 1, 1, 1, 1, 1])
print(a)
print(b)
```

Array Indexing and Boolean Indexing

```python
# Integer array indexing example
temp = a[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),”
                                   "(3, 0):\n”, temp)
# boolean array indexing example
cond = a < 0.5 # cond is a boolean array
temp = a[cond]
print ("\nElements less than 0.5:\n”, temp)
```

Basic Operations on Arrays

```
# Python program to demonstrate
# basic operations on single array
import numpy as np

a = np.array([1, 2, 5, 3])

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)
```

Basic Operations on Arrays

```
# square each element
print ("Squaring each element:”, a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:”, a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n”, a)
print ("Transpose of array:\n”, a.T)
```

Unary Operations

```python
# Python program to demonstrate
# unary operators in numpy
import numpy as np

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

# maximum element of array
print (”Largest element is:”, arr.max())
print (”Row-wise maximum elements:”,
                    arr.max(axis = 1))
```

Unary Operations

```python
# minimum element of array
print ("Column-wise minimum elements:",
                        arr.min(axis = 0))

# sum of array elements
print ("Sum of all array elements:",
                            arr.sum())

# cumulative sum along each row
print ("Cumulative sum along each row:\n",
                        arr.cumsum(axis = 1))
```