UNIT-III

Arrays

Topics covered

Arrays

- Advantages of Arrays
- Creating an Array
- Importing the Array Module
- Indexing and Slicing on Arrays
- Types of arrays
- working with arrays using numpy.

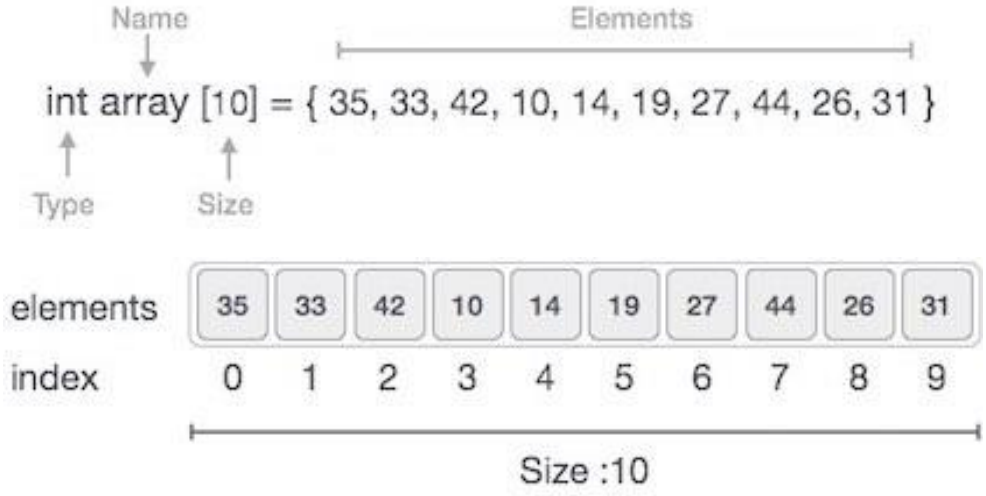
Arrays

- An array is a collection of items stored at contiguous memory locations
- The idea is to store multiple items of the same type together
- Arrays are used to store multiple values in one single variable
- Arrays can be multidimensional, and all elements in an array need to be of the **same type**, all integers or all floats
- A user can treat lists as arrays.
- Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.
- **Element**– Each item stored in an array is called an element.
- **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

Arrays

Array Representation

- Arrays can be declared in various ways in different languages. Below is an illustration.



- As per the above illustration, following are the important points to be considered.
- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Arrays

Basic Operations

Following are the basic operations supported by an array.

- **Traverse** – print all the array elements one by one.
- **Insertion** – Adds an element at the given index.
- **Deletion** – Deletes an element at the given index.
- **Search** – Searches an element using the given index or by the value.
- **Update** – Updates an element at the given index.

Arrays

Advantages

- In an array, accessing an element is very easy by using the index number.
- The search process can be applied to an array easily
- 2D Array is used to represent matrices
- For any reason a user wishes to store multiple values of similar type then the Array can be used and utilized efficiently

Disadvantages

- Array size is fixed
- Array is homogeneous - only one type of value can be store in the array (eg..int)
- Array is Contiguous blocks of memory
- Insertion and deletion are not easy in Array

Arrays

- **Python Lists Vs array Module as Arrays**
- We can treat lists as arrays. However, we cannot constrain the type of elements stored in a list.
- For example:

```
a = [1, 3.5, "Hello"]
```

- If you create arrays using **array** module, all elements of the array must be of the same numeric type.
- import array as arr
- a = arr.array('d', [1, 3.5, "Hello"]) // Error

Arrays

Creating an Array

- Array is created in Python by importing **array** module to the python program.
- Then the array is declared as shown below.

```
from array import *
arrayName = array(typecode, [Initializers])
```

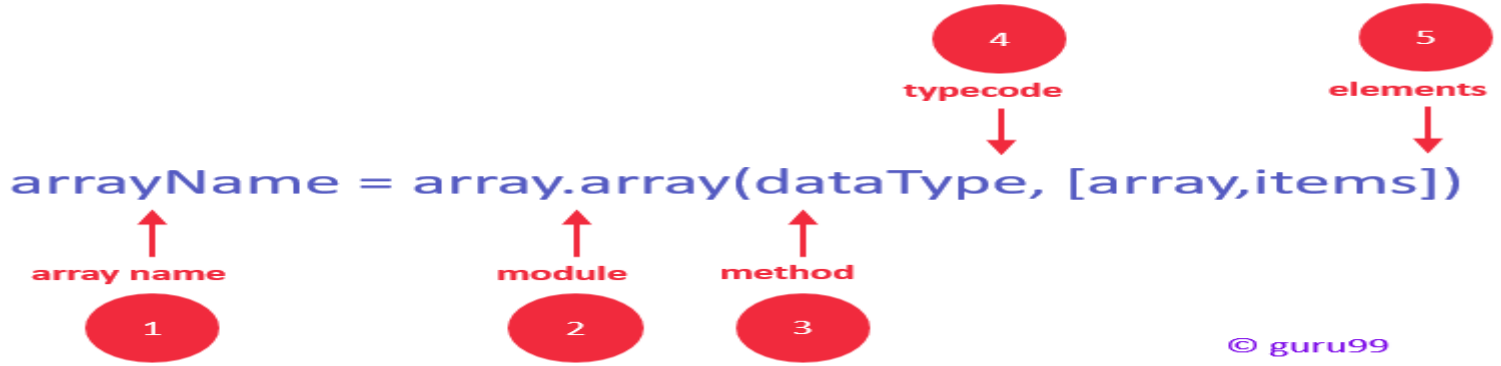
or

```
import array
arrayName = array.array(type code, [array,items])
```

Arrays

- **Array Syntax**

1. **Identifier**: specify a name like usually, you do for variables
2. **Module**: Python has a special module for creating array in Python, called "array" – you must import it before using it
3. **Method**: the array module has a method for initializing the array. It takes two arguments, type code, and elements.
4. **Type Code**: specify the data type using the type codes available (see list below)
5. **Elements**: specify the array elements within the square brackets, for example [130,450,103]



Arrays

- **Typecode** are the codes that are used to define the type of value the array will hold. Some common typecodes used are

Type code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

Arrays

Creating Array

lets create and print an array using python.

```
from array import *
array1 = array('i', [10,20,30,40,50])   #array method
for x in array1:
        print(x)
```

Accessing Array Element

- We can access each element of an array using the index of the element.
- The below code shows how

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

Arrays

Insertion Operation

- Insert operation is to insert one or more data elements into an array.
- Based on the requirement, a new element can be added at the beginning, end, or any given index of array.
- Here, we add a data element at the middle of the array using the python in-built **insert() method**.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.insert(1,60)                    # arrayName.insert(index, value)
for x in array1:
  print(x)
```

append() is also used to add the value mentioned in its arguments at the end of the array.

```
array1.append(70)
```

Arrays

- We can add one item to a list using append() method or add several items using extend()method.

```
import array as arr
 numbers = arr.array('i', [1, 2, 3])
 numbers.append(4)
print(numbers)     # Output: array('i', [1, 2, 3, 4])

# extend() appends iterable to the end of the array
numbers.extend([5, 6, 7])
print(numbers)     # Output: array('i', [1, 2, 3, 4, 5, 6, 7])
```

Arrays

- We can concatenate two arrays using + operator.

```
import array as arr

odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])

numbers = arr.array('i')            # creating empty array of integer
numbers = odd + even

print(numbers)
```

Arrays

Deletion Operation

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- Here, we remove a data element at the middle of the array using the python in-built **remove()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.remove(40)          #You can also use the 'del' statement of Python. Del array1[index]
for x in array1:
  print(x)
```

Error arises if element doesn't exist in the set

pop() function can also be used to **remove and return an element from the array**, but by default it removes only the **last element of the array**, to remove element from a **specific position** of the array, index of the element is passed as an argument to the pop() method.

Arrays

Deletion Operation

- We can delete one or more items from an array using Python's del statement.

```
import array as arr

number = arr.array('i', [1, 2, 3, 3, 4])

del number[2] # removing third element
print(number) # Output: array('i', [1, 2, 3, 4])

del number # deleting entire array
print(number) # Error: array is not defined
```

Arrays

Search Operation

- You can perform a search for an array element based on its value or its index.
- Here, we search a data element using the python in-built **index()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1.index(40))
```

it produces the following result which shows the index of the element.

If the value is not present in the array then the program returns an error.

Arrays

Update Operation

- Update operation refers to updating an existing element from the array at a given index.
- Here, we simply reassign a new value to the desired index we want to update.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1[2] = 80
for x in array1:
    print(x)
```

Arrays

- **How to change or add elements?**
- Arrays are mutable; their elements can be changed in a similar way like lists.

```
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
# changing first element
numbers[0] = 0
print(numbers)    # Output: array('i', [0, 2, 3, 5, 7, 10])
# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)    # Output: array('i', [0, 2, 4, 6, 8, 10])
```

Arrays

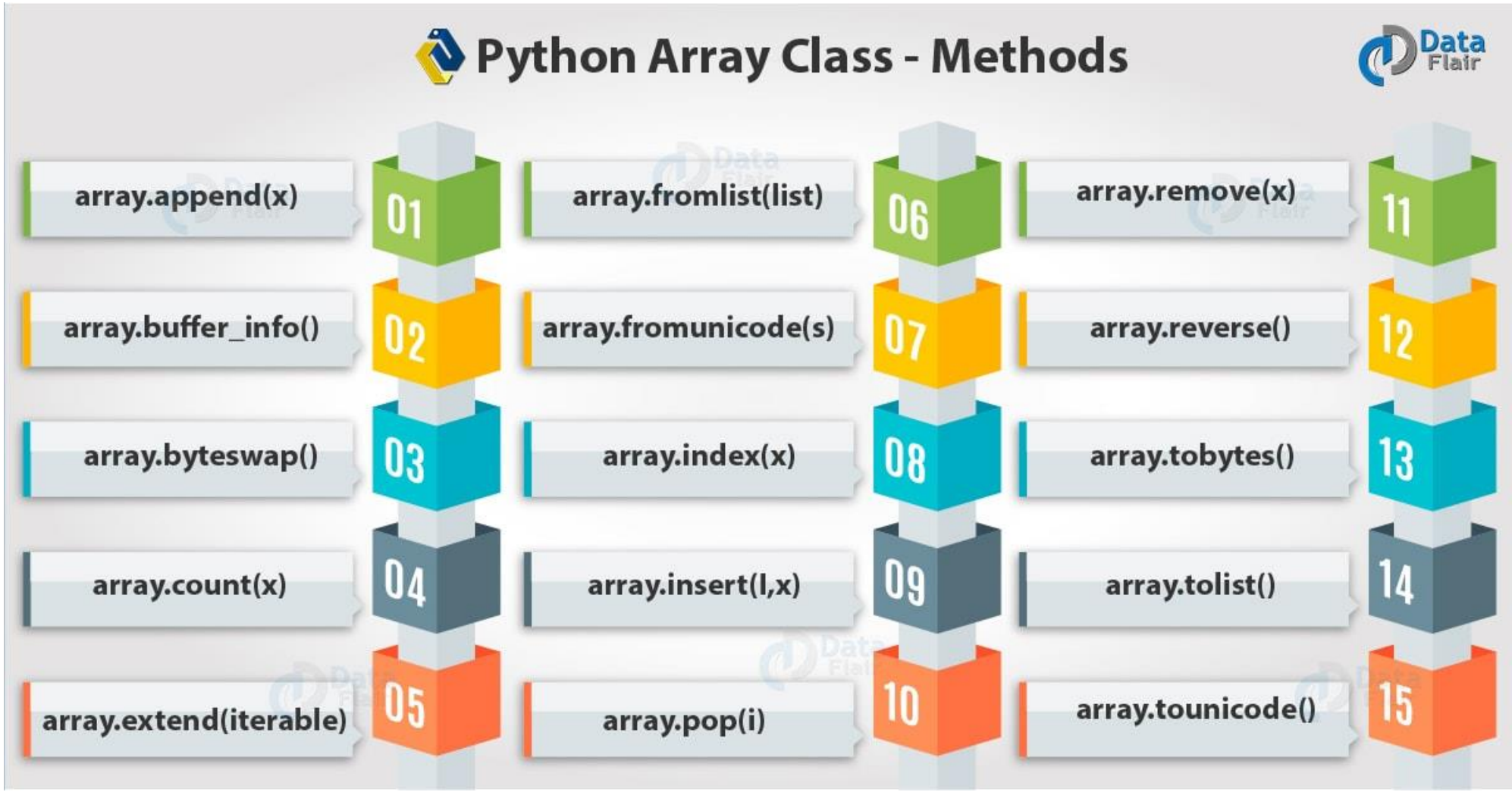
- **Slicing of a Array**
- There are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation.

- Slice operation is performed on array with the use of colon(:).
  - To print elements from beginning to a range use [:Index]
  - to print elements from end use [:-Index]
  - to print elements from specific Index till the end use [Index:]
  - to print elements within a range, use [**Start Index:End Index**] and
  - to print whole List with the use of slicing operation, use [:].
  - Further, to print whole array in reverse order, use **[::-1**].

Arrays

```
import array as arr

numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)

print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

Arrays


Python Array Class - Methods
Data Flair
array.append(x)
01
array.buffer_info()
02
array.byteswap()
03
array.count(x)
04
array.extend(iterable)
05
array.fromlist(list)
06
array.fromunicode(s)
07
array.index(x)
08
array.insert(I,x)
09
array.pop(i)
10
array.remove(x)
11
array.reverse()
12
array.tobytes()
13
array.tolist()
14
array.tounicode()
15

Arrays

Types of arrays

- One dimensional
- Two Dimensional
- Multi Dimensional

Arrays

Two dimensional array is an array within an array.

It is an array of arrays.

In this type of array the position of an data element is referred by two indices instead of one.

Consider the example of recording temperatures 4 times a day, every day.

Some times the recording instrument may be faulty and we fail to record data.

Day 1 - 11 12 5 2

Day 2 - 15 6 10

Day 3 - 10 8 12 5

Day 4 - 12 15 8 6

```
from array import *
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]
print(T[0])
print(T[1][2])
```

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

Arrays

- To print out the entire two dimensional array we can use python for loop as shown below.
- We use end of line to print out the values in different rows.

```
from array import *
T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]
for r in T:
    for c in r:
        print(c,end = " ")
    print()
```

Inserting Values in Two Dimensional Array

```
T.insert(2, [0,5,11,13,6])
```

Updating Values in Two Dimensional Array

```
T[2] = [11,9] T[0][3] = 7
```

Arrays

Here are few more examples related to Python matrices using nested lists.

Add two matrix

```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

Arrays

- # Program to transpose a matrix using nested loop

```
X = [[12,7],    [4 ,5],    [3 ,8]]
 result = [[0,0,0],
           [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[j][i] = X[i][j]
for r in result:
   print(r)
```

Arrays

- # Program to multiply two matrices using nested loops

```
# 3x3 matrix
X = [[12,7,3], [4 ,5,6], [7 ,8,9]]

# 3x4 matrix
Y = [[5,8,1,2], [6,7,3,0], [4,5,9,1]]

# result is 3x4
result = [[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

```
# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
      # iterate through rows of Y
      for k in range(len(Y)):
         result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

NumPy Array

- NumPy stands for Numerical Python.
- NumPy is a package for scientific computing which has support for a powerful N-dimensional array object.
- NumPy is the best when it comes to delivering the best high-performance multidimensional array objects and tools to work on them.

Advantages of using NumPy Arrays:

The most important benefits of using it are :

- It consumes **less memory**.
- It is **fast** as compared to the python List.
- It is **convenient to use**.

NumPy Array

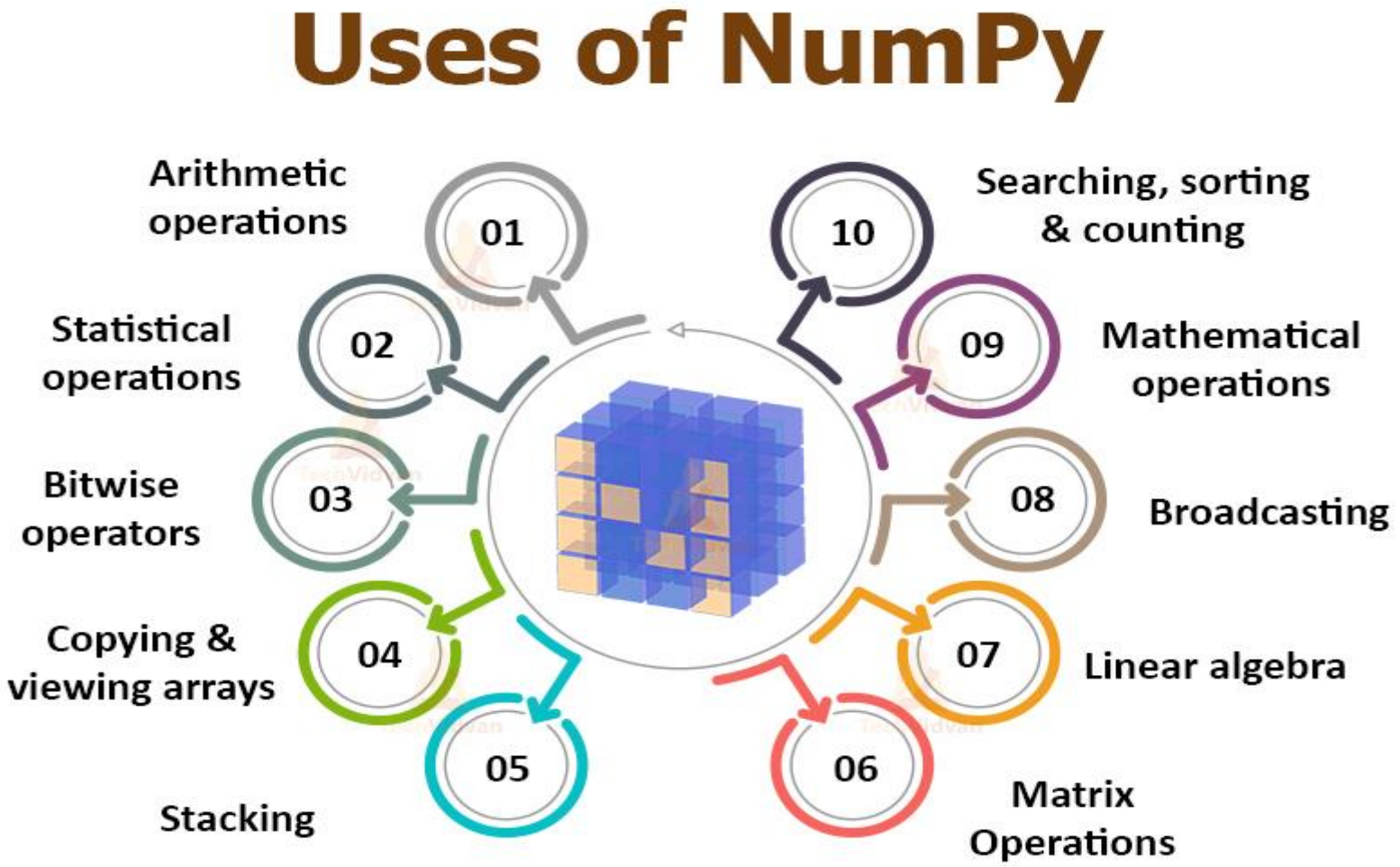
- Install **NumPy Package**

  >pip install numpy

```
import numpy as np
a = np.array([1, 2, 3])
print(a)          # Output: [1, 2, 3]
print(type(a))    # Output: <class 'numpy.ndarray'>
```

- As you can see, NumPy's array class is called ndarray.

NumPy Array


Uses of NumPy
Arithmetic operations
01
Statistical operations
02
Bitwise operators
03
Copying & viewing arrays
04
Stacking
05
Matrix Operations
06
Linear algebra
07
Broadcasting
08
Mathematical operations
09
Searching, sorting & counting
10

NumPy Array

- **Comparing Memory use**

```
import numpy as np
import time
import sys
# Creating a NumPy array with 10 elements
array = np.arange(10)
# array.itemsize : Size of one element
# array.size : length of array
print("Size of NumPy array: ", array.size * array.itemsize)

# Creating a list with 10 elements
# Now I'll print the size of list
list = range(0, 10)
# Multiplying size of 1 element with length of the list
print("Size of list: ", sys.getsizeof(1)*len(list))
```

Output:
Size of NumPy array: 40
Size of list: 280

NumPy Array

- **Fast Computation of NumPy array**

```
import numpy as np
import time
import sys
# let's declare the size
Size = 100000

# Creating two lists
list1 = range(Size)
list2 = range(Size)

# Creating two NumPy arrays
arr1 = np.arange(Size)
arr2 = np.arange(Size)

# Calculating time for Python list
start = time.time()
result = [(x+y) for x,y in zip(list1, list2)]

print("Time for Python List in msec: ", (time.time() - start) * 1000)

# Calculating time for NumPy array
start = time.time()
result = arr1+arr2
print("Time for NumPy array in msec: ", (time.time()- start) * 1000)

print("\nThis means NumPy array is faster than Python List")
```

Output:

```
Time for Python List in msec: 24.025440216064453
Time for NumPy array in msec: 2.995252609252929 7
This means NumPy array is faster than Python List
```

NumPy Array

- **NumPy is Convenient to use**

```
import numpy as np
import time

a1 = np.array([1, 2, 3])
a2 = np.array([4, 5, 6])

# To add two array you can simply do it by
print("ADD a1 and a2 elements : ", a1 +  a2)

# To sub two array you can simply do it by
print("SUB a1 and a2 elements : ", a1 - a2)

# To mul two array you can simply do it by
print("MUL a1 and a2 elements : ", a1 * a2)
```

```
# Calculating time for NumPy array
start = time.time()
result = arr1 +  arr2
print("Time for NumPy array in msec: ", (time.time()- start) * 1000)
```

Output:

```
ADD a1 and a2 elements : [5 7 9]
 SUB a1 and a2 elements : [-3 -3 -3]
MUL a1 and a2 elements : [ 4 10 18]
Time for NumPy array in msec: 0.0
```

NumPy Array

- An array is a grid of values and it contains information about the raw data, how to locate an element, and how to interpret an element.
- It has a grid of elements that can be indexed in various ways.
- The elements are all of the same type, referred to as the array **dtype.**
- The **rank** of the array is the number of dimensions.
- The **shape** of the array is a tuple of integers giving the size of the array along each dimension.

- The NumPy **ndarray** class is used to represent both **matrices and vectors**.
- A **vector** is an array with a single dimension (there's no difference between row and column vectors),
- while a **matrix** refers to an array with two dimensions.
- For **3-D** or higher dimensional arrays, the term **tensor** is also commonly used.

NumPy Array

- **attributes of an array**
- An array is usually a fixed-size container of items of the same type and size.
- The number of dimensions and items in an array is defined by its shape.
- dimensions are called **axes**
- ndarray.ndim Number of array dimensions.
- ndarray.size Number of elements in the array.
- ndarray.itemsize Length of one array element in bytes.
- ndarray.shape Tuple of array dimensions.

NumPy Array

How to create a NumPy array?

- There are several ways to create NumPy arrays.

- **1. Array of integers, floats and complex Numbers**

```
import numpy as np

A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)

A = np.array([[1.1, 2, 3], [3, 4, 5]])                    # Array of floats
print(A)

A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex)     # Array of complex numbers
print(A)
```

NumPy Array

- **2. Array of zeros and ones**

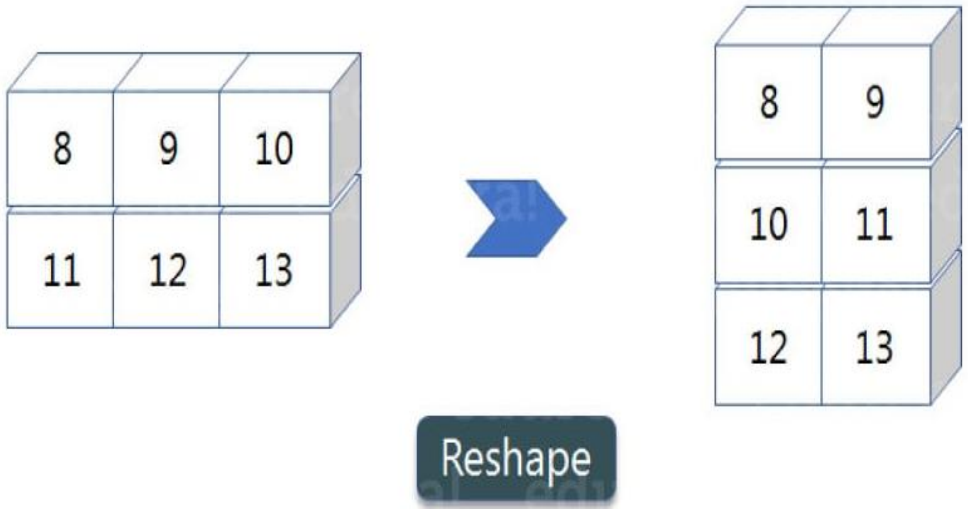
```
import numpy as np
zeros_array = np.zeros( (2, 3) )
print(zeros_array)

'''
 Output:
 [[0. 0. 0.]
  [0. 0. 0.]]
'''
```

- ones_array = np.ones((1, 5), dtype=np.int32) #specifying dtype
- print(ones_array) # Output: [[1 1 1 1 1]]
- Here, we have specified dtype to 32 bits (4 bytes). Hence, this array can take values from -2^{-31} to $2^{-31}-1$.

NumPy Array

- **3. Using arange() and reshape()**

```
import numpy as np
A = np.arange(4)
print('A =', A)

B = np.arange(12).reshape(2, 6)
print('B =', B)

'''
Output:
A = [0 1 2 3]
B = [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
'''
```



NumPy Array

- Examples

```
# Create an array of ones
np.ones((3,4))
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
# Create an array of zeros
np.zeros((2,3,4),dtype= int)
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
```

```
# Create an array with random values
from numpy import random
a=random.randint(100, size=(4,2))
[[79 87] [28 70] [67 29] [90 0]]

# Create an empty array
np.empty((3,2))
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]

# Create a full array
np.full((2,2),7)
[[ 7.  7.]
 [ 7.  7.]]
```

```
# Create an array of evenly-spaced values
np.arange(10,25,5)
[10 15 20]

# Create an array of evenly-spaced values
np.linspace(0,2,9)
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,
1.25,  1.5 ,  1.75,  2.  ])
```

np.identity().
An identity matrix is a square matrix of which all elements in the principal diagonal are ones and all other elements are zeros.

NumPy Array

- **How To Inspect Your NumPy Arrays**

```
my_array= np.array([[1, 2, 3], [3, 4, 5]])
# Print the number of `my_array`'s dimensions
print(my_array.ndim)
2
# Print the number of `my_array`'s elements
print(my_array.size)
6
# Print information about `my_array`'s memory layout
print(my_array.flags)
C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
# Print the length of one array element in bytes
print(my_array.itemsize)
8
# Print the total consumed bytes by `my_array`'s
elements
print(my_array.nbytes)
48
# Print the length of `my_array`
print(len(my_array))
2
# Change the data type of `my_array`
my_array.astype(float)
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.]])

a= np.array([1,2,3])
print(a.min())
print(a.max())
print(a.sum())
```

NumPy Array

- **Copying from One Array to Another**

  We can copy content from one array to another using the copyto function.

```
import numpy as np
#creating an array a zeros square array of dimensions 2X2
zeros_array = np.zeros([2,2], dtype = int)
print ("Array zeros is:", zeros_array)

ones_array = np.ones([2,2], dtype = int)
print ("Array ones is :", ones_array)

#copying content from ones_array to zeros
np.copyto(zeros_array,ones_array)
print ("New zeros array :", zeros_array)
```

NumPy Array

- **Matrix Operations**
- addition of two matrices, multiplication of two matrices and transpose of a matrix.
- We used nested lists before to write those programs.
- Let's see how we can do the same task using NumPy array.

- **Addition of Two Matrices**
- We use + operator to add corresponding elements of two NumPy matrices.

```
import numpy as np
 A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B      # element wise addition
print(C)
```

```
'''
Output:
[[11  1]
 [ 8  0]]
'''
```

NumPy Array

- **Multiplication of Two Matrices**
- To multiply two matrices, we use dot() method. Learn more about how numpy.dot works.
- **Note:** * is used for array multiplication (multiplication of corresponding elements of two arrays) not matrix multiplication.

```
import numpy as np

A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(C)
```

```
'''
Output:
[[ 36 -12]
 [ -1   2]]
'''
```

NumPy Array

- **Transpose of a Matrix**
- We use numpy.transpose to compute transpose of a matrix.

```
import numpy as np
A = np.array([[1, 1], [2, 1], [3, -3]])
print(A.transpose())    #A.T

'''
Output:
[[ 1  2  3]
 [ 1  1 -3]]
'''
```

As you can see, NumPy made our task much easier.

NumPy Array

- **Access matrix elements, rows and columns**
- **Access matrix elements**
- Similar like lists, we can access matrix elements using index. Let's start with a one-dimensional NumPy array.
-

```
import numpy as np
A = np.array([2, 4, 6, 8, 10])

print("A[0] =", A[0])     # First element
print("A[2] =", A[2])     # Third element
print("A[-1] =", A[-1])   # Last element
```

Output:
A[0] = 2
A[2] = 6
A[-1] = 10

NumPy Array

- Now, let's see how we can access elements of a two-dimensional array (which is basically a matrix).

```
import numpy as np
 A = np.array([[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

#  First element of first row
print("A[0][0] =", A[0][0])
 # Third element of second row
print("A[1][2] =", A[1][2])
 # Last element of last row
print("A[-1][-1] =", A[-1][-1])
```

When we run the program, the output will be:

```
A[0][0] = 1
A[1][2] = 9
A[-1][-1] = 19
```

NumPy Array

- **Access rows of a Matrix**

```
import numpy as np

A = np.array([[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

print("A[0] =", A[0]) # First Row
print("A[2] =", A[2]) # Third Row
print("A[-1] =", A[-1]) # Last Row (3rd row in this case)
```

When we run the program, the output will be:

```
A[0] = [1, 4, 5, 12]
A[2] = [-6, 7, 11, 19]
A[-1] = [-6, 7, 11, 19]
```

NumPy Array

- **Access columns of a Matrix**

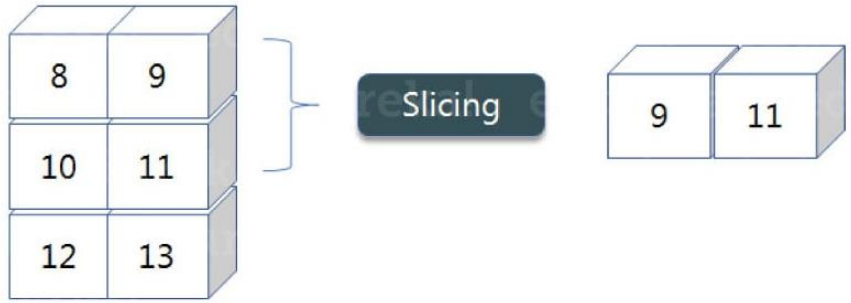
```
import numpy as np

A = np.array([[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

print("A[:,0] =",A[:,0]) # First Column
print("A[:,3] =", A[:,3]) # Fourth Column
print("A[:,-1] =", A[:,-1]) # Last Column (4th column in this case)
```

When we run the program, the output will be:

```
A[:,0] = [ 1 -5 -6]
A[:,3] = [12  0 19]
A[:,-1] = [12  0 19]
```

NumPy Array

- **Slicing of a Matrix**
- Slicing of a one-dimensional NumPy array is similar to a list.
- Let's take an example:




```
import numpy as np
letters = np.array([1, 3, 5, 7, 9, 7, 5])

# 3rd to 5th elements
print(letters[2:5])       # Output: [5, 7, 9]

# 1st to  _ elements
print(letters[:-5])       # Output: [1, 3]

# 6th to last elements
print(letters[5:])        # Output:[7, 5]

# 1st to last elements
print(letters[:])         # Output:[1, 3, 5, 7, 9, 7, 5]

# reversing a list
print(letters[::-1])        # Output:[5, 7, 9, 7, 5, 3, 1]
```

NumPy Array

- Now, let's see how we can slice a matrix.

```
import numpy as np

A = np.array([[1, 4, 5, 12, 14],
   [-5, 8, 9, 0, 17],
   [-6, 7, 11, 19, 21]])

print(A[:2, :4])  # two rows, four columns

''' Output:
[[ 1  4  5 12]
 [-5  8  9  0]]
'''

print(A[:1,])  # first row, all columns

''' Output:
[[ 1  4  5 12 14]]
'''

print(A[:,2])  # all rows, second column

''' Output:
[ 5  9 11]
'''

print(A[:, 2:5])  # all rows, third to fifth column

'''Output:
[[ 5 12 14]
 [ 9  0 17]
 [11 19 21]]
'''
```

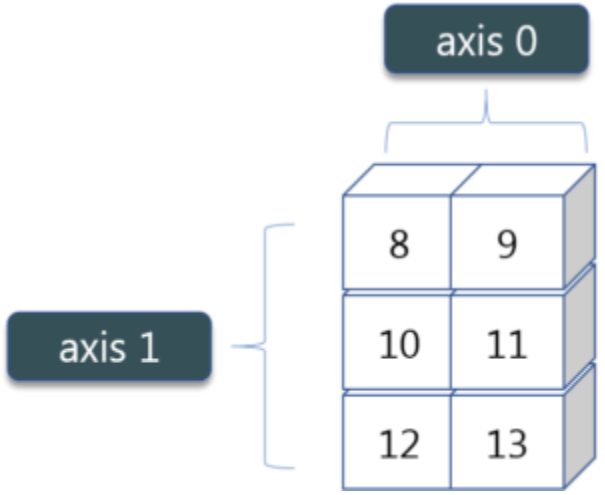
NumPy Array

- **concept of axis in python numpy**

  As you can see in the figure, we have a numpy array 2*3. Here the **rows are called as axis 1** and the **columns are called as axis 0**.

- Now you must be wondering what is the use of these axis?
- Suppose you want to calculate the sum of all the columns, then you can make use of axis.
- a= np.array([(1,2,3),(3,4,5)])
- print(a.sum(axis=0))
- Output - [4 6 8]

- Therefore, the sum of all the columns are added where 1+3=4, 2+4=6 and 3+5=8.
- Similarly, if you replace the axis by 1, then it will print [6 12] where all the rows get added.



NumPy Array

```
import numpy as np
a=np.array([(1,2,3),(3,4,5,)])
print(np.sqrt(a))
print(np.std(a))
```

Output –

```
 [[ 1.     1.41421356   1.73205081] [ 1.73205081      2.       2.23606798]]
1.29099444874
```

NumPy Array

- **How To Load NumPy Arrays From Text**
- when you want to get started with data analysis, you'll need to load data from text files.
- Make use of some specific functions to load data from your files, such as **loadtxt() or genfromtxt()**
- # This is your data in the text file ("data.txt")

  # Value1 Value2 Value3

  # 0.2536 0.1008 0.3857

  # 0.4839 0.4536 0.3561

  # 0.1292 0.6875 0.5929

  # 0.1781 0.3049 0.8928

  # 0.6253 0.3486 0.8791

- # Import your data
- x, y, z = np.loadtxt('data.txt', skiprows=1, unpack=True)

NumPy Array

- # Your data in the text file

  # Value1 Value2 Value3

  # 0.4839 0.4536 0.3561

  # 0.1292 0.6875 MISSING

  # 0.1781 0.3049 0.8928

  # MISSING 0.5801 0.2038

  # 0.5993 0.4357 0.7410

- my_array2 = np.genfromtxt('data2.txt', skip_header=1, filling_values=-999)

NumPy Array

- The examples indicated this maybe implicitly, but, in general, genfromtxt() gives you a little bit more flexibility; It's more robust than loadtxt().
- Let's make this difference a little bit more practical: the latter, loadtxt(), only works when each row in the text file has the same number of values;
- So when you want to handle missing values easily, you'll typically find it easier to use genfromtxt().

NumPy Array

- **How To Save NumPy Arrays**
- Once you have done everything that you need to do with your arrays, you can also save them to a file.
- If you want to save the array to a text file, you can use the **savetxt()function** to do this:

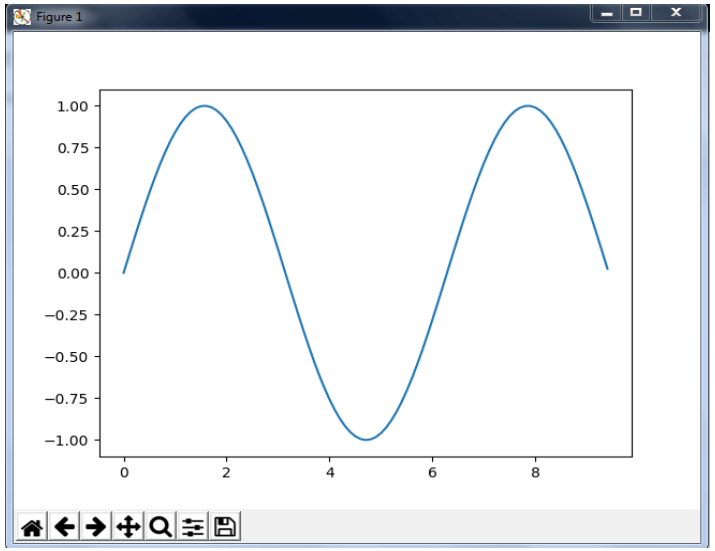
```
import numpy as np
x = np.array([1,2,3])
np.savetxt('test.out', x, delimiter=',')
```

NumPy Array

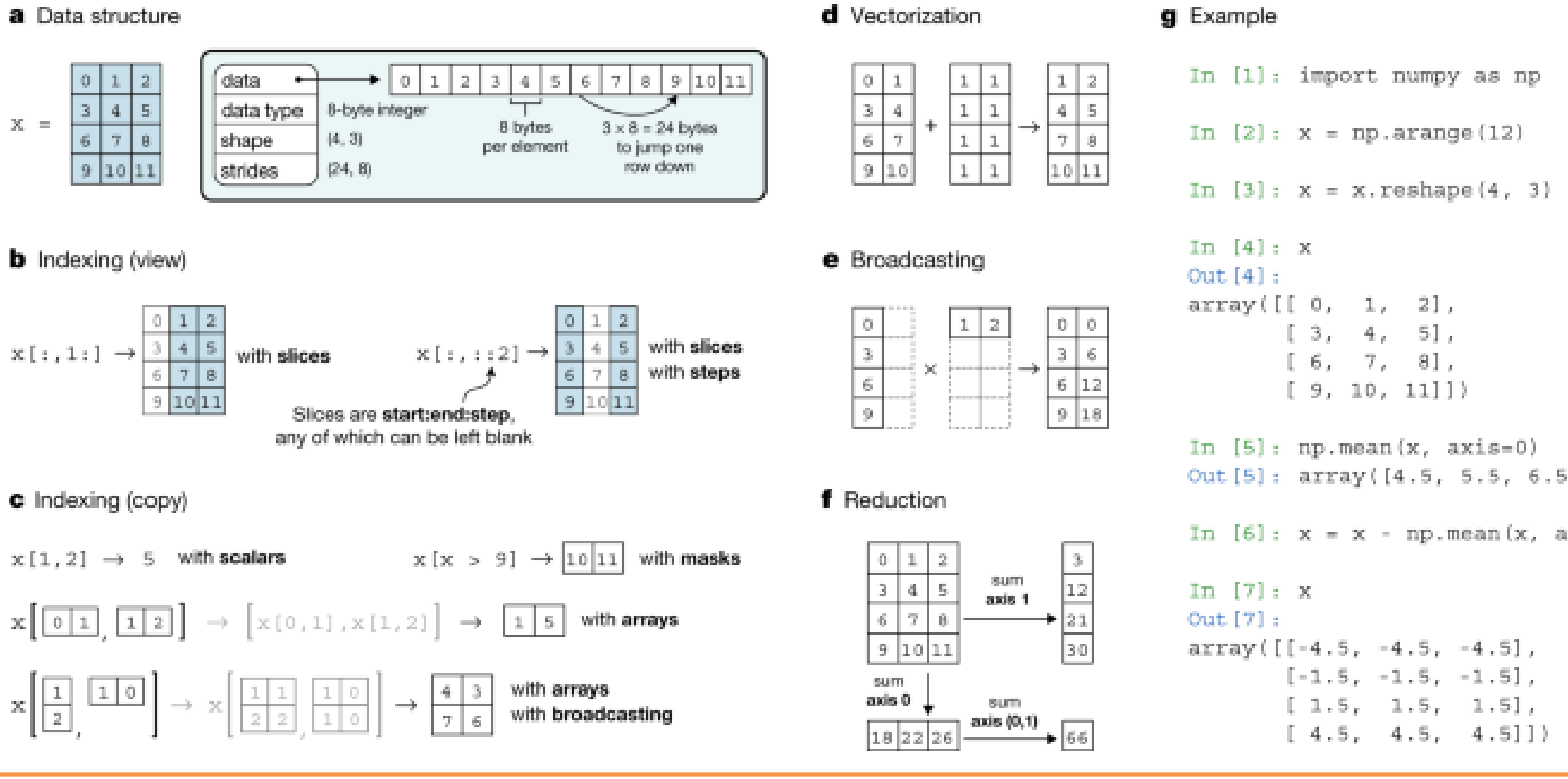
- **Python Numpy Special Functions**
- There are various special functions available in numpy such as sine, cosine, tan, log etc.

- First, let's begin with sine function where we will learn to plot its graph.
- For that, we need to import a module called *matplotlib*.

```
import numpy as np
import matplotlib.pyplot as plt

x= np.arange(0,3*np.pi,0.1)
y=np.sin(x)
plt.plot(x,y)
plt.show()
```



NumPy Array




```
In [1]: import numpy as np

In [2]: x = np.arange(12)

In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```

NumPy Array

- **Pass a matrix in a function**

```
import numpy as np
def function( x ):
 return 0.5 * x + 2

x = np.arange(0,10,0.1)
y = function(x)
print(y)
```

MALLA REDDY
UNIVERSITY

www.mallareddyuniversity.ac.in