Data Structure With Python 4331601

Unit No.	Unit Title	Teaching Hours	Distribution of Theory Marks			
			R Level	U Level	A Level	Total Marks
I	Basic Concepts of Data Structures	04	04	02	00	06
II	Basics of Object Oriented Programming	08	04	04	04	12
III	Stack and Queues	08	02	06	06	14
IV	Linked List	08	04	08	02	14
V	Searching and Sorting	08	02	06	06	14
VI	Trees	06	02	04	04	10
Total		**42**	**18**	**30**	**22**	**70**

Basic Concepts of Data Structures

Prepared By:

Dhaval Gandhi

Information Technology Department,

Dr S & S S Ghandhy College of Engineering & Technology,

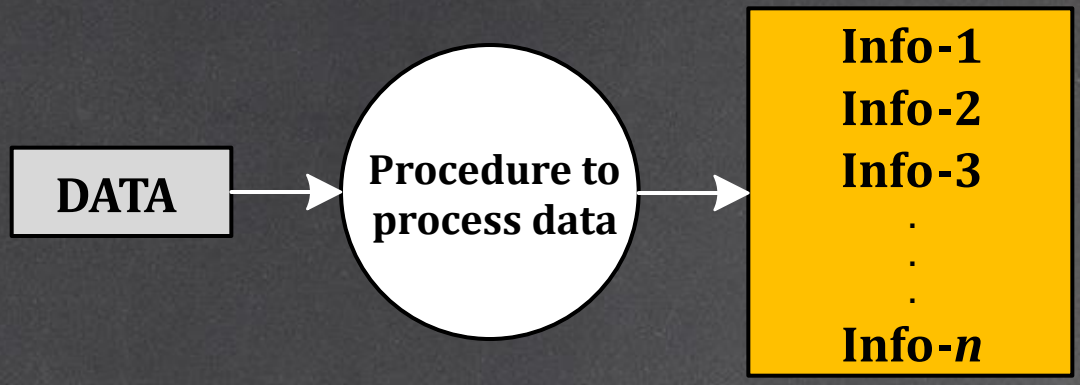
SURAT, GUJARAT

Learning Outcomes

- Types of data structures
- Linear and non-linear data structures
- Analysis Terms
  - Time Complexity and Space Complexity
  - Asymptotic Notations
- Python Specific Data Structures-
  - List, Tuple, Set, Dictionary
- Array in Python
  - import array
  - import numpy
- Operations on Arrays
- Arrays vs. List

Data & Information

- **Data** means value or a set of values.
  - 35
  - 21/12/2016,
  - "SURAT"
  - 12, 18, 24, 32



- **Information** means meaningful or processed data.
  - 35 — Age of a person
  - 21/12/2016 — Date of Birth
  - "SURAT" — Name of the City
  - 12, 18, 14, 30 — Marks of a subject

Data Type

- **Data type** is a term which refers to the kind of data. That may appear in computation.

35	Numeric (integer)
21/06/2019	Date
"SURAT"	String
12, 18, 14, 30	Array of integers

What is Data Structure?

- **Data:** Data is set of items
- **Structure:** How to organize data (A particular way of organizing data in computer)

- A **data structure** is a method for organizing and storing data, which would allow efficient data retrieval and usage.

What is Data Structure?

- It is mathematical/logical way of organizing data items in memory and represents the relationship among data that is stored in memory.
- It allows us to manipulate data by specifying set of values , set of operations that can be performed on set of rules that needs to be followed while performing operations.

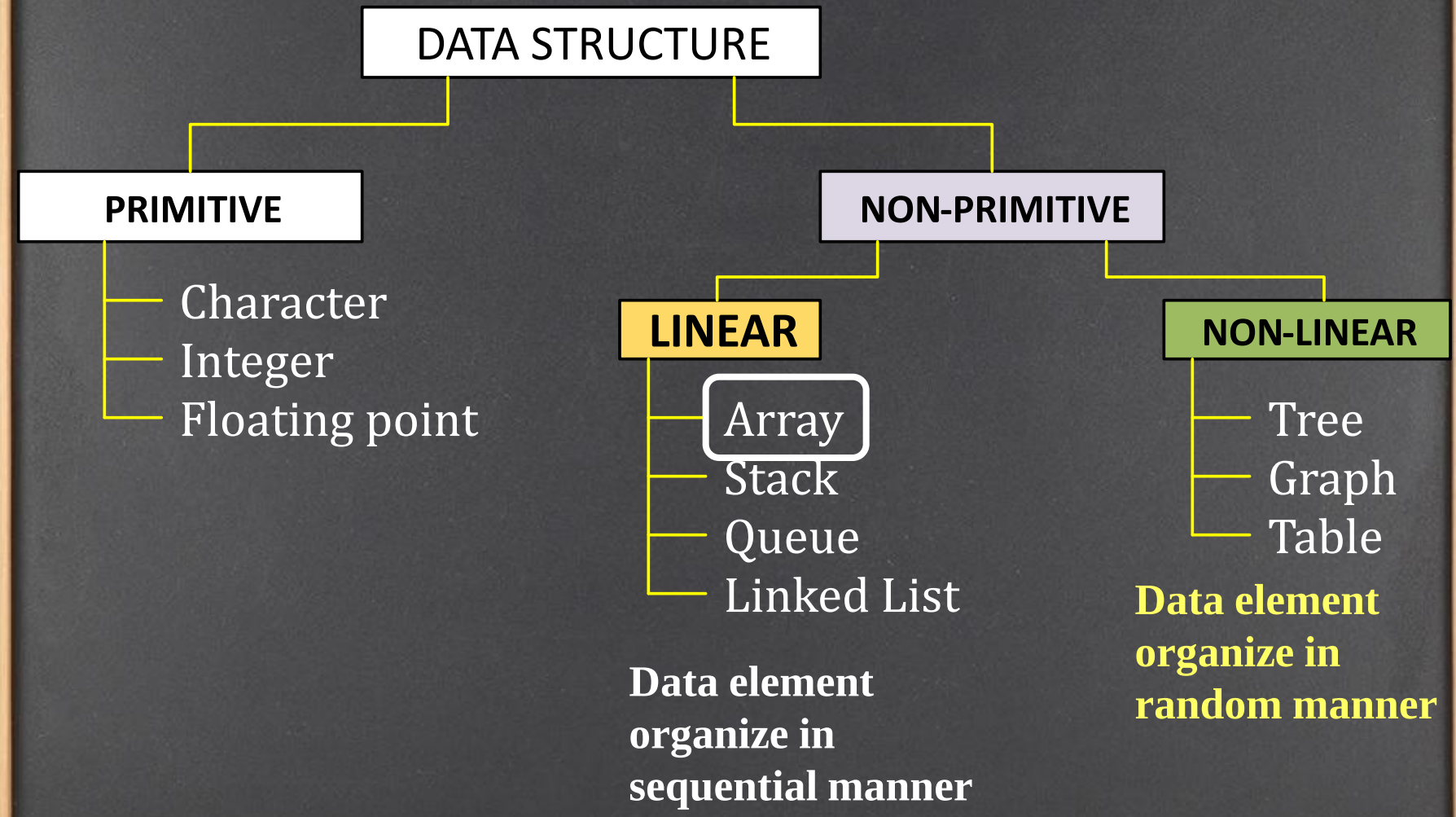
Why data structure?

- In computer, manipulation of primitive data does not require any extra effort on the part of user.
- In real-life applications, various kinds of data other than primitive are involved.
- Manipulation of real-life data requires following tasks:
  1. Storage representation of user data
  2. Retrieval of stored data
  3. Transformation of user data

Where is Data structure used?

- **Data Structure is used for**
  - How the data should be organized in the memory
  - How the flow of data should be controlled
  - How efficiently it can be retrieved and manipulated
  - How data should be designed and implemented to reduce the complexity and increase the efficiency of the algorithm

Types of Data Structure



Types of Data Structure

- Primitive Data Structure:-
- Data Structure which is directly operated by machine level instruction is known as Primitive Data Structure.
- All inbuilt data types are known as primitive DS.
- Ex. Integer, Float, Character, Pointer
- Non-Primitive Data Structure:-
- Data Structure which is not directly operated by machine level instruction is known as Non-Primitive Data Structure.
- It is derived from Primitive DS.
- They are divided into two types.
- Linear Data Structure
- Non Linear Data Structure.

Types of Data Structure

- Linear Data Structure:-
- Data Structure in which elements are arranged such that we can process them in linear manner(sequentially) is called linear DS.
- Ex. Array, List, Stack, Queue, Linked List.
- Non-Linear Data Structure:-
- Data Structure in which elements are arranged such that we can not process them in linear manner(sequentially) is called Non-Linear DS.
- Ex. Tree, Graph.

Array

- An **Array** is one of the linear non-primitive data structure.

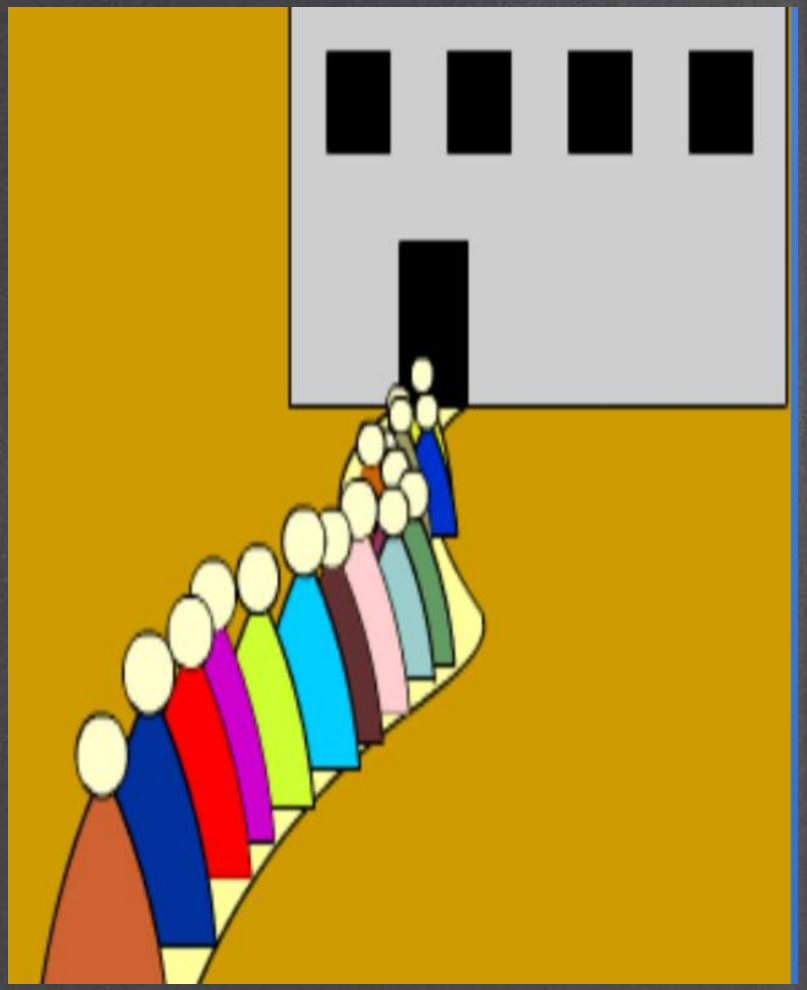
- **Array** is a collection of similar data type variables having contiguous(sequential) memory locations that share a common name.

Stack

- A stack is a linear data structure in which items can be inserted only from one end and get items back from the same end.
- There , the last item inserted into stack, is the first item to be taken out from the stack.
- It is known as LIFO (Last in First Out).

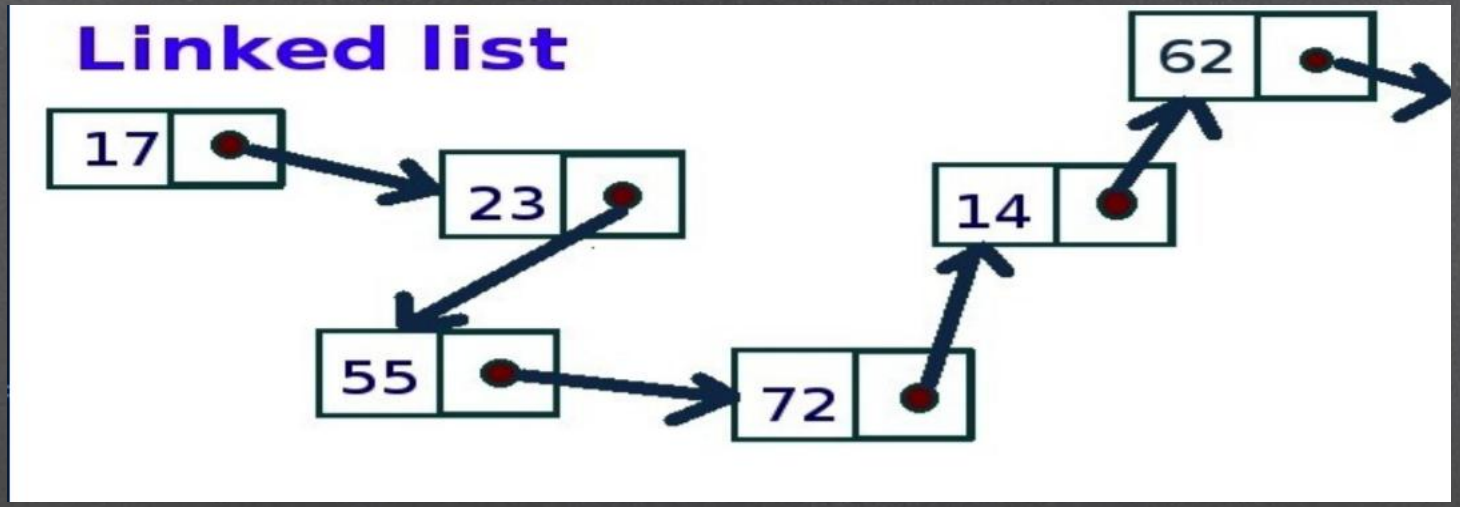
Queue

- A queue is two ended data structure in which items can be inserted from one end and taken out from the other end.
- Therefore , the first item inserted into queue is the first item to be taken out from the queue.
- This property is called First in First out [FIFO].



Linked List

- It is an ordered set which consist of variable number of elements.
- Here elements are logically adjacent to each other but physically not.
- It is a collection of nodes and each node consist of two parts.
- 1) Value of a node 2) Address of next node.



Linked List

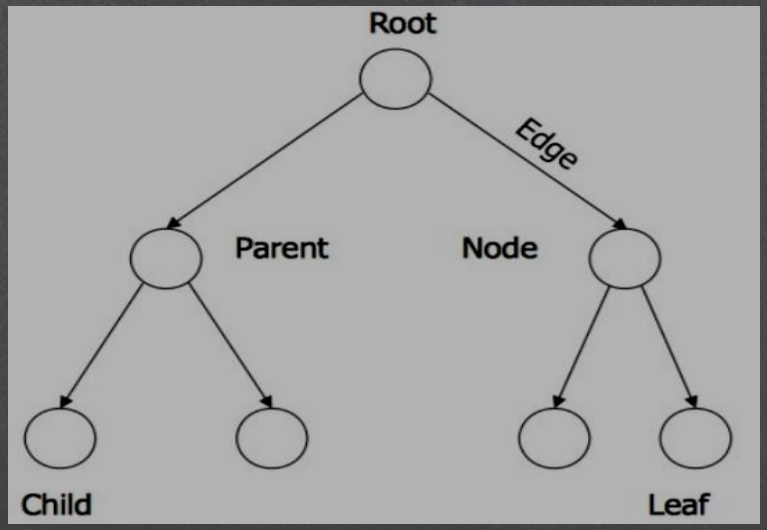
- In linked list space to store items is created as is needed and destroyed when space no longer required to store items.
- Hence linked list is a dynamic data structure space acquire only when need.
- It has four types:

1. Singly Linked List
2. Doubly Linked List
3. Circular Singly Linked List
4. Circular Doubly Linked List

Tree

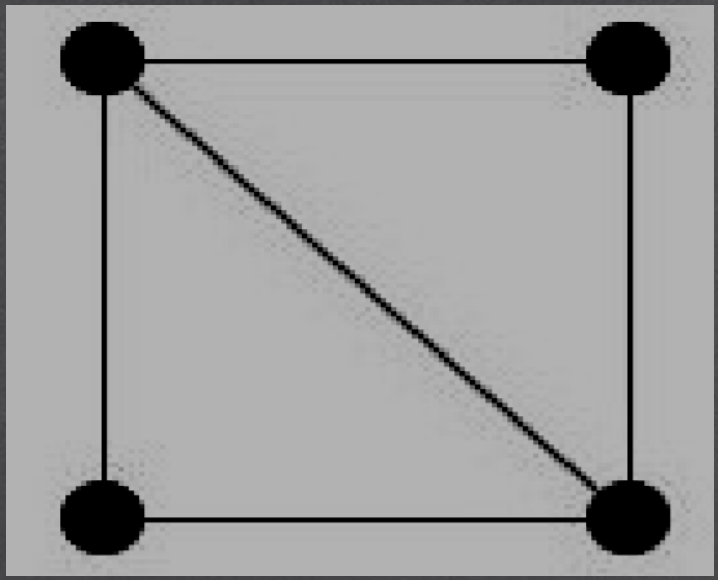
- It is used to represent data containing a hierarchical relationship between elements.
- It is defined as a finite set of one or more nodes such that

1. Special node root node having no predecessor.
2. All nodes in a tree except root node having only one predecessor.
3. All nodes in a tree having one or more successor.



Graph

- It is a set of items connected by edges.
- Each item is called a vertex or node. Trees are just like a special kinds of graphs. Graphs are usually represented by G = (V, E), where V is the set vertices and E is the set of Edges.



Different Types

Types	Description
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: Array
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: Tree, Graph
Homogeneous	In homogeneous data structures, all the elements are of same type. Example: Array
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: Structures
Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: Array
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: Linked List created using pointers

Algorithm

- An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain predefined task.
- It is a step by step solution of particular program.
- Every Algorithm must satisfy the following properties:
- Input- There should be 0 or more inputs supplied externally to the algorithm.
- Output- There should be at least 1 output obtained.
- Definiteness- Every step of the algorithm should be clear and well defined.
- Finiteness- The algorithm should have finite number of steps.
- Correctness- Every step of the algorithm must generate a correct output.

Algorithm

- Example:-
- Algorithm to find area of circle

Name: -AREA of CIRCLE A=¶r2

1. [initialize]
   START
2. [input]
   Input R
3. [calculate area of circle]
   Compute A= 3.14 * r * r
4. [output]
   Print A
5. [Exit]
   STOP

Algorithm

- Features:-
- Name:- identify by name which is written in capital letters.
- Steps:- It is made up of a sequence number of steps, each beginning with a square bracket which gives description of statement.
- Sequence:- All steps in algorithm are executed in sequence one by one, if it does not contain any looping structure.
- Ex., Area of a circle

Algorithm

- Selection [Decision Structure] :- If an algorithm contains any selection/ decision then it can be represented by selection.
- Statement that containing any decision is encountered, the condition first check and based on outcome of that condition statement is executed.

Name: -MAX(A,B)

1. [initialize]
   START
2. [input]
   Input A, B
3. [Selection/decision statement]
   if A > B then
   M←A else
   M←B
4. [output]
   Print M
5. [Exit]
   STOP

Algorithm

- Repetition [Looping Structure] :- When it is required to perform several steps again & again until certain condition is satisfied at that time repetition structure is used.

Name: -SUM OF FIRST TEN DIGIT(I, SUM)

1. [initialize]
   START
2. [Variable initialization]
   I← 1
   SUM ← 0
3. [Looping statement]
   Repeat step 4 while (I<=10)
4. SUM← SUM +1
   I← I+1
5. [output]
   Print SUM
6. [Exit]
   STOP

Analysis Term

- An algorithm is said to be efficient and fast, if it takes less time to execute and consumes less memory space.
- The performance of an algorithm is measured on the basis of following properties :
- Time Complexity
- Space Complexity
-

Analysis Term

- Time Complexity:-
- Amount of time needed by a program to complete its execution is known as time complexity.
- Measurement of time is done in terms of number of instructions executed by program during its execution.
- It depends on size of a programme and type of algorithm.

Analysis Term

- Space Complexity:-
- Amount of memory needed by a program during its execution is known as space complexity.
- There are two types of space complexity:-
- Static:- It contains space required for simple variables ,constant, instruction and fixed size structured variable such as an array.
- Dynamic:- It contains space required for structured variable to which memory allocated run time.
- It contains space required while function calling itself.

Asymptotic Notations

- **Asymptotic Notations** are the expressions that are used to represent the complexity of an algorithm.
- There are three **types of analysis** that we perform on a particular algorithm.
- **Best Case**: Measurement of minimum time required by an algorithm to complete its execution.
- It depends on different input values.
- Ex. Already sorted values
- **Worst Case**: Measurement of maximum time required by an algorithm to complete its execution.
- It depends on different input values.
- Ex., Reverse order.
- **Average Case**: Measurement of average time required by an algorithm to complete its execution.
- Ex., Different input values.

Asymptotic Notations

- **Types of Data Structure Asymptotic Notation**

  1. Big-O Notation (O) – Big O notation specifically describes worst case scenario.

  2. Omega Notation (Ω) – Omega(Ω) notation specifically describes best case scenario.

  3. Theta Notation (θ) – This notation represents the average complexity of an algorithm.

Asymptotic Notations

- **Big-O Notation (O)**
- Big O notation specifically describes worst case scenario.
- It represents the upper bound running time complexity of an algorithm.
- **O(1)**
- It represents the complexity of an algorithm that always execute in same time or space regardless of the input data.
- **Example:-**Accessing array index(int num = arr[5])
- **O(n)**
- It represents the complexity of an algorithm, whose performance will grow linearly (in direct proportion) to the size of the input data.
- **O(n) example:-** The execution time will depend on the size of array. When the size of the array increases, the execution time will also increase in the same proportion (linearly)
- Traversing an array

Asymptotic Notations

- O(n^2)
- It represents the complexity of an algorithm, whose performance is directly proportional to the square of the size of the input data.
- O(n^2) example
- Traversing a 2D array Other examples: Bubble sort, insertion sort and selection sort algorithms.
- O(logn)
- An algorithm in which during each iteration the input data set is partitioned into subparts.
- Example: Quick sort and Binary Sort.

Asymptotic Notations

- **Omega Notation (Ω)**
- Omega notation specifically describes best case scenario. It represents the lower bound running time complexity of an algorithm.
- So if we represent a complexity of an algorithm in Omega notation, it means that the algorithm cannot be completed in less time than this, it would at-least take the time represented by Omega notation or it can take more (when not in best case scenario).
- **Theta Notation (θ)**
- This notation describes both upper bound and lower bound of an algorithm so we can say that it defines exact asymptotic behavior.
- In the real case scenario the algorithm not always run on best and worst cases, the average running time lies between best and worst and can be represented by the theta notation.

Standard Data Types

- The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has Six standard data types –
  - Numbers
  - String
  - List
  - Tuple
  - Set
  - Dictionary

Standard Data Types

- Python Numbers:-
- Number data types store numeric values. Number objects are created when you assign a value to them. For example − var1 = 1 var2 = 10
- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is
- del var1[,var2[,var3[....,varN]]]]
- You can delete a single object or multiple objects by using the del statement.
- For example −
- del var

Standard Data Types

- Python Numbers:-
- Python supports three different numerical types –
  - int (signed
  - float (floating point real values)
  - complex (complex numbers)
- All integers in Python3 are represented as long integers. Hence, there is no separate number type as long.

Standard Data Types

- Python Strings:-
- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows either pair of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string .
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.
- For example –
- str= 'Hello
- print (str) # Prints complete string
- print (str * 2) # Prints string two times
- print (str + "TEST") # Prints concatenated string

Standard Data Types

- Python Lists:-
- Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

Standard Data Types

- Python Lists:-
- The elements in the list can be mutable, which means that we can add, remove and modify the value of existing elements. Lists can contain any type of data, i.e. they are heterogeneous.
- As the list is an ordered sequence of elements, we can access the elements of the list by their index value starting from 0 to count-1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example −

Standard Data Types

```
#!/usr /bin/python3
list = [ 'abcd ', 786 , 2.23, 'john', 70.2]
tinylist= [123, 'john']
print (list)  # Prints complete list
print (list[0])  # Prints first element of the list
print (list[1:3]) # Prints elements starting from 2nd till 3rd
print (list[2:])  # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist )  # Prints concatenated lists
```

Standard Data Types

This produces the following result −

```
[' abcd ', 786, 2.23, 'john',70.20000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.20000000000003]
[123, 'john', 123, 'john']
[' abcd ', 786, 2.23, 'john', 70.20000000000003,
123, 'john']
```

Standard Data Types

Python Tuples :-

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas.
- Unlike lists, however, tuples are enclosed within parenthesis. The elements in the tuple are immutable, which means that we cannot add, remove or manipulate the elements in the tuple.
- The main difference between lists and tuples are – Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists. For example –

Standard Data Types

```
#!/usr /bin/python3
tuple= ( abcd ', 786 , 2.23, 'john', 70.2)
tinytuple = (123, 'john')
print (tuple ) # Prints complete tuple
print (tuple [0]) # Prints first element of the tuple
print (tuple [1:3]) # Prints elements starting from 2nd till
    3rd
print (tuple [2:]) # Prints elements starting from 3rd
    element
print (tinytuple * 2) # Prints tuple two times
print (tupletuple+ tinytuple ) # Prints concatenated tuple
```

Standard Data Types

This produces the following result −

```
(' abcd ', 786, 2.23, 'john',70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
(' abcd ', 786, 2.23, 'john', 70.200000000000003,
123, 'john')
```

Standard Data Types

Python Dictionary :-

- Python's dictionaries are kind of hash-table type.
- They work like associative arrays or hashes found in Perl and consist of key key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- The keys and values comprises of any immutable data type and its data type remains ordered.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

Standard Data Types

```
#!/usr /bin/python3
dict={}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict= {'name': 'john','code':6734, 'dept': '
print (dict ['one']) # Prints value for 'one' key
print (dict [2]) # Prints value for 2 key
print (tinydict ) # Prints complete dictionary
print (tinydict.keys ()) # Prints all the keys
print (tinydict.values ()) # Prints all the values
```

Standard Data Types

This produces the following result −

This is one

This is two

{'name': 'john', 'dept': 'sales', 'code': 6734}

dict_keys(['name', 'dept', 'code'])

dict_values(['john', 'sales',6734])

- Dictionaries have no concept of order among the elements.
- It is incorrect to say that the elements are "out of order"; they are simply unordered.

Standard Data Types

Python Set :-

- Set is an unordered collection of unique items.
- Set is defined by values separated by comma inside braces{ }.
- Items in a set are not ordered.
- The elements in the sets are mutable. The main characteristic of a set is that it cannot have duplicate elements.
- A Set can also contain any type of data, i.e. they are also of heterogeneous type.

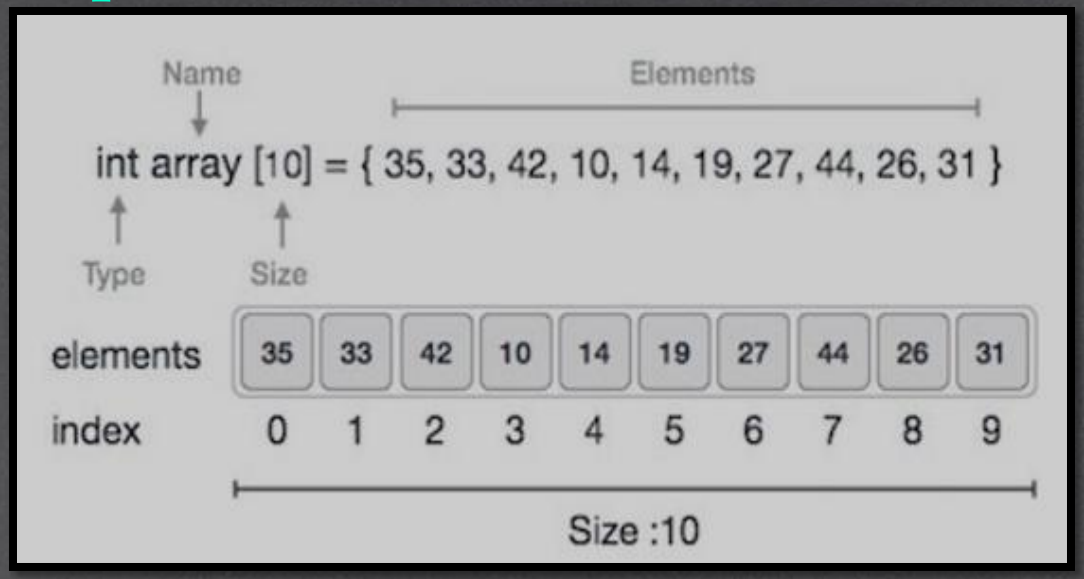
```
a = {5,2,3,1,4} # printing set variable
 print("a = ", a)
# data type of variable a
print(type(a))
```

Array

- Array:-
- An array is defined as a collection of items that are stored at contiguous memory locations.
- It is a container which can hold a fixed number of items, and these items should be of the same type.
- Most of the data structures make use of arrays to implement their algorithms.
- 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.

Array

- **Array Representation:-**



- **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.**

Array

- **Characteristics of an array:-**
- **Every element has assign a address.**
- **It always starts with 0 index number and always ends with one less than size of array.**
- **Arrays are mutable.**
- **It stored data sequentially.**
- **Account of memory occupied by array is depends on data type and number of elements.**

Array

- **Basic Operations of an array:-**
- 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.

Applications

1. To implement mathematical vector and matrices, many databases: small & large, consist of 1D arrays whose elements are records.
2. To implement other data structures, such as heaps, hash tables, queues, stacks
3. One or more large arrays are sometimes used to emulate in-program dynamic memory allocation, particularly memory pool allocation.

Array Operations

- 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])
- Typecode are the codes that are used to define the type of value the array will hold

Array Operations

Typecode	Value
b	Represents signed integer of size 1 byte/td>
B	Represents unsigned integer of size 1 byte
c	Represents character of size 1 byte
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
f	Represents floating point of size 4 bytes
d	Represents floating point of size 8 bytes

Array

- **Example 1:-**

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

Output:- array('d', [1.1, 3.5, 4.5])

- **Example 2:-**

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

```
10
20
30
40
50
```

Array

- **Accessing array elements :-**

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

Output:-

```
First element: 2
Second element: 4
Second last element: 8
```

Array

- **change or add elements:-**

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

Output:-

```
array('i', [0, 2, 3, 5, 7, 10])
array('i', [0, 2, 4, 6, 8, 10])
```

Array

- **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.**
- **We can add one item to the array using the append() method.**
- **If we want to add several items then using the extend() method.**

Array

- **Example:-**

```
import array as arr
numbers = arr.array('i', [1, 2, 3])
numbers.append(4)
print(numbers)
numbers.extend([5, 6, 7])
print(numbers)
numbers.insert(1,50)
print(numbers)
```

Output:

```
array('i', [1, 2, 3, 4])
array('i', [1, 2, 3, 4, 5, 6, 7])
array('i', [1, 50, 2, 3, 4, 5, 6, 7])
```

Array

- **Deletion Operation :-**
- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- We can use the remove() method to remove the given item, and pop() method to remove an item at the given index.
- If We can delete one or more items from an array using Python's del statement.

Array

- **Example:-**

```
import array as arr
number = arr.array('i', [1, 2, 5, 3, 4])
del number[2]  # removing third element
print(number)
number.remove(4) # removing element having value 4
print(number)
print(number.pop(2))
print(number)
del number  # deleting entire array
print(number)
```

Output:

```
array('i', [1, 2, 3, 4])
array('i', [1, 2, 3])
3
array('i', [1, 2])
# Error: array is not defined
```

Array

- **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.**

```
import array as arr
number = arr.array('i', [10, 20, 30, 40, 50])
print(number.index(20))
print(number.index(40))
```

Output:-

1

3

Array

- **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.**

```
import array as arr
number = arr.array('i', [10, 20, 30, 40, 50])
number[1]=5
print(number)
```

Output:-

array('i', [10, 5, 30, 40, 50])

Array

- **Slicing Python Arrays :-**
- **We can access a range of items in an array by using the slicing operator :.**

Array

```
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
```

Output:-

```
array('i', [62, 5, 42])
array('i', [2, 5, 62])
array('i', [52, 48, 5])
array('i', [2, 5, 62, 5, 42, 52, 48, 5])
```

Array

- **We can also concatenate two arrays using + operator.**

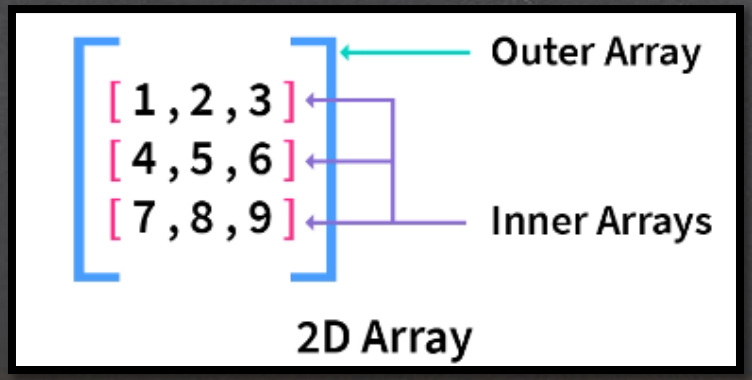
```
import array as arr
odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])
numbers = arr.array('i')    # creating empty array of integer
numbers = odd + even
print(numbers)
```

Output:-

array('i', [1, 3, 5, 2, 4, 6])

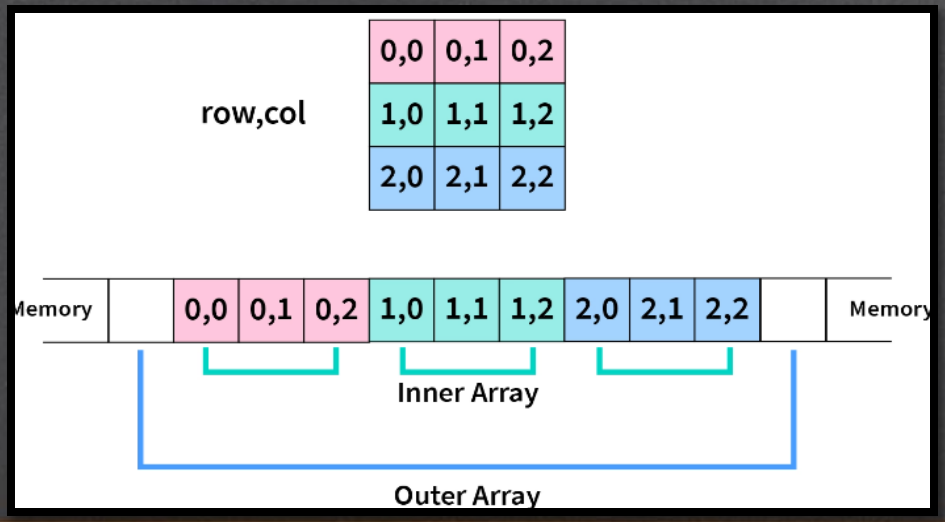
Array

- 2-D array:-
- 2D array in python is a two-dimensional data structure used for storing data generally in a tabular format.
- 2D array in python is a two-dimensional data structure, stored linearly in the memory. This means that it has two dimensions, the rows and the columns and thus it also represents a matrix.



Array

- **2-D array:-**
- In Python, we can access elements of a two-dimensional array using two indices. The first index refers to the indexing of the list and the second index refers to the position of the elements. If we define only one index with an array name, it returns all the elements of 2-dimensional stored in the array.



Array

- Syntax of creating 2D Array:-

array_name=[[r1c1,r1c2,r1c3,..],[r2c1,r2c2,r2c3,...],. . . .]

- where array_name is the name of the array, r1c1, r1c1 etc are elements of the array.
- Here r1c1 means that it is the element of the first column of the first row.
- A 2D array is an array of arrays.
- Accessing 2D Array Elements:-
- array_name[row_ind][col_ind]
- array_name[row_ind]
- where array_name is the name of the array, row_ind is the row index of the element and col_ind is the column index of the element.

Array

- Example of 2D Array:-

```
from array import *
Student_dt = [ [72, 85, 87, 90, 69], [80, 87, 65, 89, 85], [96, 91, 70, 78, 97], [90, 93, 91, 90, 94], [57, 89, 82, 69, 60] ]
print(Student_dt[1]) # print all elements of index 1
print(Student_dt[0]) # print all elements of index 0
print(Student_dt[2]) # print all elements of index 2
print(Student_dt[3][4]) # it defines the 3rd index and 4 position of the data element.
```

- Output:

[80, 87, 65, 89, 85]

[72, 85, 87, 90, 69]

[96, 91, 70, 78, 97]

94

Array

- **Traversing the element in 2D :-**

```
from array import *
Student_dt = [ [72, 85, 87, 90, 69], [80, 87, 65,
  89, 85], [96, 91, 70, 78, 97], [90, 93, 91, 90,
  94], [57, 89, 82, 69, 60] ]
for x in Student_dt:  # outer loop
    for i in x:  # inner loop
        print(i, end = " ") # print the elements
    print()
```

72 85 87 90 69

80 87 65 89 85

96 91 70 78 97

90 93 91 90 94

57 89 82 69 60

Array

- **Insert elements in a 2D Array:-**
- **We can insert elements into a 2 D array using the insert() function that specifies the element index number and location to be inserted.**

Array

```
from array import *
arr1 = [[1, 2, 3, 4], [8, 9, 10, 12]]
print("Before inserting the array elements: ")
print(arr1) # print the arr1 elements.
arr1.insert(1, [5, 6, 7, 8])  # first parameter
  defines the index no., and second parameter
  defines the elements
print("After inserting the array elements ")
print(arr1)
for i in arr1: # Outer lo
    for j in i: # inner l
        print(j, end = "
    print()
```

```
Before inserting the array elements:
[[1, 2, 3, 4], [8, 9, 10, 12]]
After inserting the array elements
[[1, 2, 3, 4], [5, 6, 7, 8], [8, 9, 10, 12]]
1 2 3 4
5 6 7 8
8 9 10 12
```

Array

- **Update elements in a 2D Array:-**
- **In a 2D array, the existing value of the array can be updated with a new value.**
- **In this method, we can change the particular value as well as the entire index of the array.**

Array

```
from array import *
arr1 = [[1, 2, 3, 4], [8, 9, 10, 12]]
print("Before inserting the array elements: ")
print(arr1) # print the arr1 elements.
arr1[0] = [2, 2, 3, 3] # update the value of
  the index 0
arr1[1][2] = 99 # define the index [1] and
  position [2] of the array element to update
  the value.
print("After updating the array elements ")
for i in arr1: # Outer loop
    for j in i: # inner loo
        print(j, end = " “)
    print()
```

```
Before inserting the array elements:
[[1, 2, 3, 4], [8, 9, 10, 12]]
After updating the array elements
2 2 3 3
8 9 99 12
```

Array

- **Delete elements in a 2D Array:-**
- In a 2- D array, we can remove the particular element or entire index of the array using del() function in Python.

Array

```
from array import *
arr1 = [[1, 2, 3, 4], [8, 9, 10, 12]]
print("Before inserting the array elements: ")
print(arr1) # print the arr1 elements.
del(arr1[0][2]) # delete the particular element
  of the array.
del(arr1[1]) # delete the index 1 of the 2-D
  array.
print("After Deleting the array elements ")
for i in arr1: # Outer loop
    for j in i: # inner loo
        print(j, end = " “)
    print()
```

```
Before Deleting the array elements:
[[1, 2, 3, 4], [8, 9, 10, 12]]
After Deleting the array elements
1 2 4
```

Array

- **Size of 2D Array:-**
- A len() function is used to get the length of a two-dimensional array.
- In other words, we can say that a len() function determines the total index available in 2-dimensional arrays.
- Example:-

```
array_size = [[1, 3, 2],[2,5,7,9], [2,4,5,6]]
print("The size of two dimensional array is : ")
print(len(array_size)) # it returns 3
```

List v/s Array

List	Array
The list can store the value of different types.	It can only consist of value of same type.
The list cannot handle the direct arithmetic operations.	It can directly handle arithmetic operations.
We need to import the array before work with the array.	The lists are the build-in data structure so we don't need to import it.
The lists are less compatible than the array to store the data.	An array are much compatible than the list.
It consumes a large memory.	It is a more compact in memory size comparatively list.
It is suitable for storing the longer sequence of the data item.	It is suitable for storing shorter sequence of data items.
We can print the entire list using explicit looping.	We can print the entire list without using explicit looping.
It can be nested to contain different types of elements.	It must contain either all nested elements of same size.

Array

- **Numpy:-**
- NumPy is short for "Numerical Python".
- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

Array

- **Operations using NumPy:-**
- Using NumPy, a developer can perform the following operations –
- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra.
- NumPy has in-built functions for linear algebra and random number generation.

Array

- **Use of NumPy:-**
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- This behavior is called locality of reference in computer science.
- This is the main reason why NumPy is faster than lists.

Array

- **Installation of NumPy:-**
- Standard Python distribution doesn't come bundled with NumPy module. A lightweight alternative is to install NumPy using popular Python package installer, pip.
- **pip install numpy**
- Install it using this command:
- **C:\Users\Your Name>pip install numpy**

Array

- **NumPy:-**
- The most important object defined in NumPy is an N-dimensional array type called ndarray.
- It describes the collection of items of the same type.
- Items in the collection can be accessed using a zero-based index.
- Every item in an ndarray takes the same size of block in the memory.
- Each element in ndarray is an object of data-type object (called dtype).

numpy.array(object, dtype = None, ndmin = 0)

- object-Any object exposing the array interface method returns an array, or any (nested) sequence.
- dtype-Desired data type of array, optional
- ndmin-Specifies minimum dimensions of resultant array

Array

```
import numpy as np
a = np.array([1,2,3])
print (a)
print(type(a))
# more than one dimensions
a = np.array([[1, 2], [3, 4]])
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]], d
# Array of complex numbers
print(A)
# minimum dimensions
a = np.array([1, 2, 3,4,5], ndmin = 2)
print (a)
a = np.array([1, 2, 3,4,5], ndmin = 3)
print (a)
```

```
[1 2 3]
<class 'numpy.ndarray'>
[[1 2]
 [3 4]]
[[1.1 2.  3. ]
 [3.  4.  5. ]]
[[1.+0.j 2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j 5.+0.j]]
[[1 2 3 4 5]]
[[[1 2 3 4 5]]]
```

Array

```
Access Array Elements:-
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
print(arr[2] + arr[3])
#2-D array
arr=np.array([[1,2,3,4,5],[6,7,8,9
print('2nd element on 1st row: ',
print('5th element on 2nd row: ',
print("A[0] =", arr[0]) # First Ro
print("A[2] =", arr[2]) # Third Ro
print("A[-1] =", arr[-1]) # Last
   this case)
print("A[:,0] =",arr[:,0]) # First
print("A[:,3] =", arr[:,3]) # Four
print("A[:,-1] =", arr[:,-1]) #
   column in this case)
```

```
1
7
2nd element on 1st row:
2
5th element on 2nd row:
10
A[0] = [1 2 3 4 5]
A[2] = [4 6 7 8 9]
A[-1] = [4 6 7 8 9]
A[:,0] = [1 6 4]
A[:,3] = [4 9 8]
A[:,-1] = [ 5 10  9]
```

Array

Access Array Elements:-

```
import numpy as np
#3-D array
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
print(arr[1, 0, 2])
#negetive index
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

```
6
9
Last element from 2nd dim:  10
```

Array

- **Shape of an Array :-**
- The shape of an array is the number of elements in each dimension.
- **Reshaping array:-**
- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

Array

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7
print(arr.shape)
a = np.array([[1,2,3],[4,5,6]])
a.shape = (3,2)
print (a)
arr = np.array([1, 2, 3, 4, 5, 6, 7,
  11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
newarr = arr.reshape(2, 3, 2)
print(newarr)
newarr = arr.reshape(2, 2, 3)
print(newarr)
```

```
(2, 4)
[[1 2]
 [3 4]
 [5 6]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

Array

```
import numpy as np
A = np.arange(4)
print('A =', A)
B = np.arange(12).reshape(2, 6)
print('B =', B)
array=np.arange(10,20,2)
print(array)
```

```
A = [0 1 2 3]
 B = [[ 0 1 2 3 4 5]
      [ 6 7 8 9 10 11]]
[10 12 14 16 18]
```

Array

```
import numpy as np
array=np.zeros(10)
print("An array of 10 zer
print(array)
array = np.zeros( (2, 3)
print(array)
array=np.ones(10)
print("An array of 10 ones:")
print(array)
array=np.ones(10)*5
print("An array of 10 fives:")
print(array)
```

```
An array of 10 zeros:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[0. 0. 0.]
 [0. 0. 0.]]
An array of 10 ones:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
An array of 10 fives:
[5. 5. 5. 5. 5. 5. 5. 5. 5. 5.
```

Array

- **Slicing of a Matrix:-**

```
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 4th 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]
```

Array

- **Slicing of a Matrix:-**

```
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
print(A[:1,])  # first row, all columns
print(A[:,2])  # all rows, second column
print(A[:, 2:5])  # all rows to the
  fifth column
```

```
[[ 1  4  5 12]
 [-5  8  9  0]]
[[ 1  4  5 12 14]]
[ 5  9 11]
[[ 5 12 14]
 [ 9  0 17]
 [11 19 21]]
```

Array

- **Addition of Two Matrices:-**

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

```
[[ 2  4]
 [ 5 -6]]
[[ 9 -3]
 [ 3  6]]
[[11  1]
 [ 8  0]]
```

Array

- **Multiplication of Two Matrices:-**

p = [[1, 0], [0, 1]] q = [[1, 2], [3, 4]]

Matrix multiplication, and the detailed calculation

$$\begin{bmatrix} 1*1+0*3 & 1*2+0*4 \\ 0*1+1*3 & 0*2+1*4 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Array

- **Multiplication of Two Matrices:-**

```
import numpy as np
# Multiplication of Two Matrices
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(A)
print(B)
print(C)
```

```
[[ 3  6  7]
 [ 5 -3  0]]
[[ 1  1]
 [ 2  1]
 [ 3 -3]]
[[ 36 -12]
 [ -1   2]]
```

Array

- **Transpose of Two Matrices:-**

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

```
[[ 1  2  3]
 [ 1  1 -3]]
```

THANK YOU