1. What is an array? How do arrays differ from lists? Explain the types of an arrays. with a suitable example for each one.

A:

Array: An array consists of data elements of a same data type. We can say that an array saves a lot of memory and reduce the length of the code.

Array Index: In an array, elements are identified by their indexes. Array index starts with 0.

Array Element: Elements are items stored in an array and can be accessed by their index.

Array length: the length of an array is determined by the number of elements it can contain.

Arrays vs List

Both list and array are used to store the data in Python. these data structures allow us to indexing, slicing and iterating.

A list is a built-in, linear data structure of Python. It is used to store the data in a sequence manner.

example

```
1. list = [31, 60, 19, 12]
2. print (list)
3. print (type (list))
```

output:

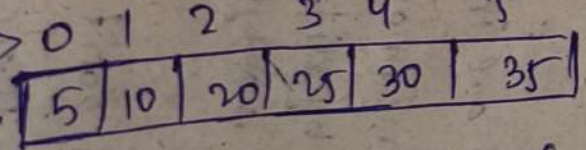
```
[31, 60, 19, 12]
<class 'list'>
```

Types of Arrays.

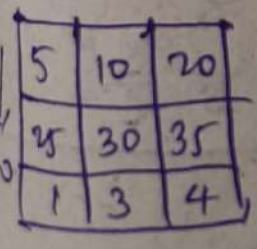
One-dimensional array (1-D): You can imagine a 1D array as a row, where elements are stored one after another.

Two-dimensional array (2-D): Multidimensional arrays can be considered as an array of arrays or as a matrix consisting of rows and columns.

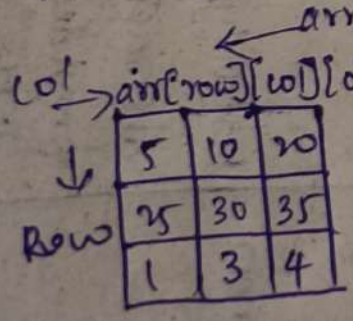
Three-dimensional array (3D): A 3-D multidimensional array contains three dimensions, so it can be considered an array of two-dimensional arrays.

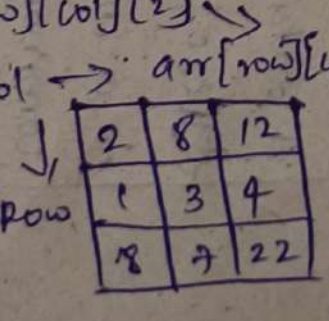


(a) One Dimensional array.



(b) Two-dimensional array

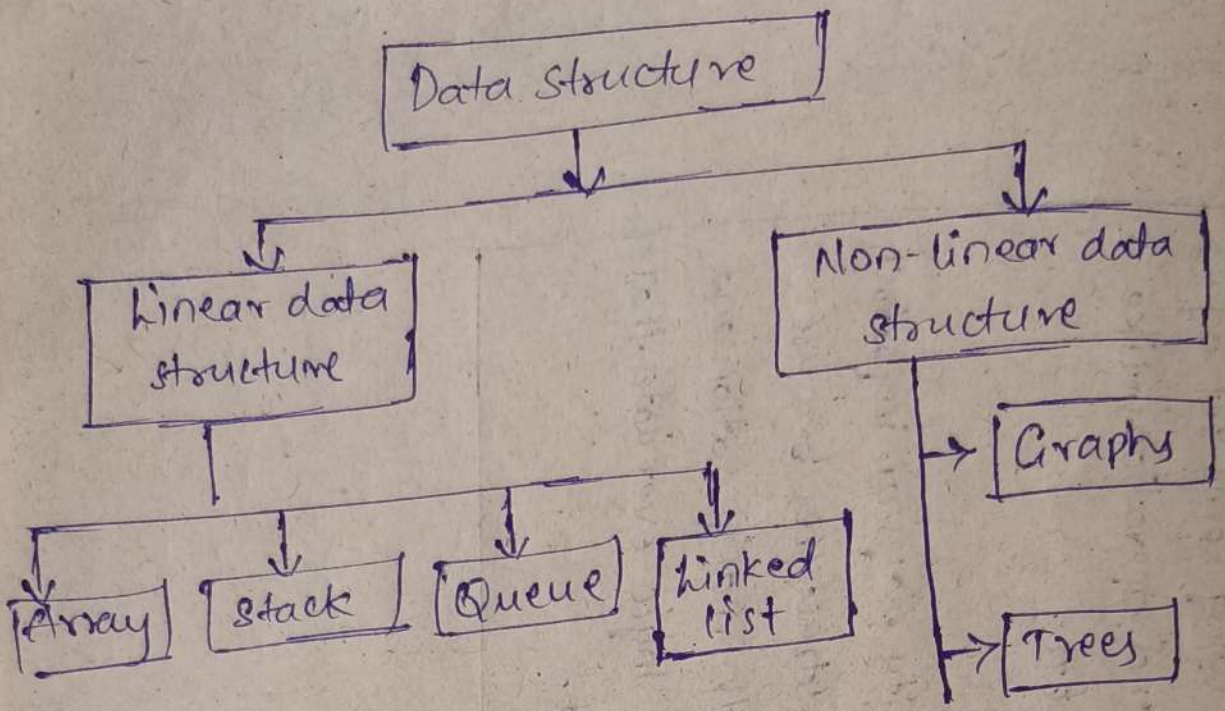




Three dimensional array

Q) What is the difference between a data type and a data structure? Can you classify data structures based on their characteristics.

Key	Data type	Data structure.
Def.	A data type represents the nature and type of data. All the data that belong to a common data type share some common properties.	(OR) data structure is the collection that holds data which can be manipulated and used in programming so that operations and algorithms can be more easily applied.
Implementation	Data types are implemented in an abstract manner. Their definitions are provided by different languages in different ways.	Data structures are implemented in a concrete manner. their definition includes what type of data they are going to store and deal with.
Storage	Data types don't ~~share~~ store the value of data; they represent only the type of data that is stored.	Data structures hold the data along with their values. they occupy space in the main memory of the computer. Also, data structures can hold different types of data within one single object.
Assignment	Data types represent the type of value that can be stored, so values can directly be assigned to the data type variables.	In case of data structure, the data is assigned using some set of algorithms and operations like push, pop, etc.
Performance	There is no issue of time complexity because data types deal only with the type and nature of data	Time complexity plays an important role in data structures because they deal with manipulation and execution of logic over data that is stored.



3) Explain linear search technique with an example program?

A: Linear search is a method of finding elements within a list. It is also called a sequential search. It is the simplest searching algorithm because it searches the desired element in a sequential manner. It compares each and every element with the value that we are searching for. If both are matched the element is found, and the algorithm returns the key's index position.

Implementation of Linear Search Algorithm:

```
# Python code to linearly search x in arr[].
# If x is present then return its location,
  otherwise return -1.

def search(arr, N, x):
    for i in range(0, N):
        if (arr[i] == x):
            return i

    return -1

# Driver code
if __name__ == "__main__":
    arr = [2, 3, 4, 10, 40]
    x = 10
    N = len(arr)

    # Function call
    result = search(arr, N, x)
    if (result == -1):
        print("Element is not present in array")
    else:
        print("Element is present at index", result)
```

4. Define Binary search? Explain with an example program?

Q: A binary search is an algorithm to find a particular element in the list. There are many searching algorithms but the binary search is most popular among them. The elements in the list must be sorted to apply the binary search algorithm. If elements are not sorted then sort them first.

Step by step implementation of binary search.

→ Sort the array in ascending order.

→ Set the low index to the first element of the array and the high index to the last element.

→ Set the middle index to the average of the low and high indices.

→ If the element at the middle index is the target element, return the middle index.

→ If the target element is less than the element at the middle index, set the high index to the middle index - 1.

→ If the target element is greater than the element at the middle index, set the low index to the middle index + 1.

Recursive implementation of Binary search:

```
# Python3 Program for recursive binary search.
# Returns index of x in arr if present, else -1
def binarysearch(arr, l, r, x):
    # check base case
    if r >= l:
        mid = l + (r-l) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binarysearch(arr, l, mid-1, x)
        else:
            return binarysearch(arr, mid+1, r, x)
    else:
        # Element is not present in the array
        return -1
# Driver code
arr = [2, 3, 4, 10, 40]
x = 10
# Function call
result = binarysearch(arr, 0, len(arr)-1, x)
if result != -1:
    print("Element is present at index %d" % result)
else:
    print("Element is not present in array")
```

Output:

Element is present at index 3

Time complexity : o(logn)

Auxilliary space : o(logn)

5) Explain the quick sort algorithm. Write a python program to implement a quick sort for an array using input method.

A: Quick sort:

-> The quick sort algorithm falls under the divide and conquer class of algorithms, where we break (divide) a problem into smaller chunks that are much simpler to solve (conquer).

-> The Quick sorting process is mainly considered in three phases.

* Partitioning the list/array
* Pivot selection
* Implementation.

-> Before we divide the list into smaller chunks, we have to partition it. This is the heart of the quick sort algorithm.

-> There are different variations of quicksort where the pivot element is selected from different positions. (start, median, random, end).

→ Randomly picking the middle or last element in the array as the pivot does not improve the situation any further.

→ There are two famous partition schemes for Quick sort

* Hoare Partition
* Lomuto Partition.

Write a python program to arrange the elements in ascending order using Quick sort (with functions)

```
def quick-sort(elements, start, end):
    if start < end:
        pi = partition(elements, start, end)
        quick-sort(elements, start, pi-1)
        quick-sort(elements, pi+1, end)

def partition(elements, start, end):
    pivot_index = start
    pivot = elements[pivot_index]
    while start < end:
        while start < len(elements) and elements[start]
                <= pivot:
            start += 1
        while elements[end] > pivot:
            end -= 1
```

```
if start < end:
    elements[start], elements[end] = elements[end],
                                      elements[start]
elements[pivot_index], elements[end] = elements[end],
                                       elements[pivot_index]
return end
elements = [11, 9, 29, 7, 2, 15, 28]
# elements = ["mona", "dhaval", "aamir", "tina", "chang"]
quick_sort(elements, 0, len(elements)-1)
print(elements)
```

6> Explain the Merge sort algorithm. Write a python program to implement a Merge sort for an array using input method.

A: Merge sort:

Generally, this merge sort works on the basis of divide and conquer algorithm. The three steps need to be followed is divide, conquer and combine.

Algorithm:

1. Split the unsorted list.
2. Compare each of the elements and group them.
3. Repeat step 2 until whole list is merged and sorted.

Write a python program to arrange the elements in ascending order using Merge sort.

```
def merge-sort(arr):
# Base case: if the input array is empty or has only one element, it is already sorted.
    if len(arr) <= 1:
        return arr
# Divide the array into two halves
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]
# Recursively sort each half
    left_sorted = merge_sort(left_half)
    right_sorted = merge_sort(right_half)
# Merge the sorted halves
    result = []
    i=j=0
    while i< len(left_sorted) and j<len(right_sorted):
        if left_sorted[i] < right_sorted[j]:
            result.append(left_sorted[i])
            i+=1
        else:
            result.append(right_sorted[j])
            j+=1
```

```
# Append the remaining elements of the left
  or right half
  result.extend(left_sorted[i:])
  result.extend(right_sorted[j:])
  return result.

arr = [3,1,4,1,5,9,2,6,5,3,5]
print(merge_sort(arr))
```

7) Explain the Bubble sort algorithm. Write a python program to implement a Bubble sort for an array using input method.

A: Bubble Sort:

It is a simple sorting algorithm which sorts 'n' number of elements in the list by comparing the each pair of adjacent items and swaps them if they are in wrong order.

Algorithm:

1. Starting with the first element (index=0), compare the current element with the next element of a list.
2. If the current element is greater than the next element of the list then swap them.

3. If the current element is less than the next element of the list move to the next element.

4. Repeat step 1 until it correct order is framed.

Write a python program to arrange the elements in ascending order using bubble sort:

```
list1 = [9,16,6,26,0]
print("unsorted list1 is", list1)
for j in range(len(list1)-1):
    if list1[i] > list1[i+1]:
        list1[i], list1[i+1] = list1[i+1], list1[i]
        print(list1)
    else:
        print(list1)
print()
print("sorted list is", list1)
```

Output:

```
unsorted list1 is [9,16,6,26,0]
[9,16,6,26,0]
[9,6,16,26,0]
[9,6,16,26,0]
[9,6,16,0,26]
[6,9,16,0,26]
[6,9,16,0,26]
```

[6, 9, 0, 16, 26]

[6, 9, 0, 16, 26]

[6, 9, 0, 16, 26]

[6, 0, 9, 16, 26]

[6, 0, 9, 16, 26]

[0, 6, 9, 16, 26]

[0, 6, 9, 16, 26]

[0, 6, 9, 16, 26]

[0, 6, 9, 16, 26]

Sorted list is [0, 6, 9, 16, 26]

1.a> What are stacks, and what are the main characteristics of a stack.

A: Stacks:

A stack is a data structure that stores a collection of elements and operates on them based on the principle of Last-In-First Out (LIFO). This means that the last element added to the stack is the first one to be removed. Also, the inbuilt functions in Python make the code short and simple.

```
# Python code to demonstrate implementing stack
  using list.
stack = ["Amar", "Akbar", "Anthony"]
stack.append("Ram")
stack.append("Iqbal")
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

Output:

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
Iqbal
```

['Amar', 'Akbar', 'Anthony', 'Ram']

Ram

['Amar', 'Akbar', 'Anthony']

characteristics of a stacks:

1. LIFO ordering: The last item pushed onto the stack is the first item popped off the stack.

2. Push and Pop Operations: The two primary operations that can be performed on a stack are "push", which adds an item to the top of the stack, and "pop", which removes the item from the top of the stack.

3. Top Element Access: A stack allows access only to the top element, which is the most recently added item.

4. Limited Accessibility: Stacks do not allow access to elements in the middle of the stack.

5. Dynamic size: stacks can grow or shrink dynamically as items are added or removed.

6. Contiguous Memory Allocation: Stacks typically are contiguous memory allocation.

7. Stacks overflow: A stack has a finite amount of memory allocated to it. When the stack exceeds this memory limit, a stack overflow occurs.

8. Stack Underflow: A stack underflow occurs when an attempt is made to remove an element from an empty stack.

b) What are some of the key operations that can be performed on stack? Explain with examples

Ans: Stack operations:

1) Push(): Insert the element into linked list nothing but which is the top node of stack.

2) pop(): Return top element from the stack and move the top pointer to the second node of linked list or stack.

3) Peek(): Return the top element.

4) display(): Print all element of stack.

The stack has two primary operations:

1. Push - Adds an element to the top of the stack.

2. Pop - Removes the element from the top of the stack.

Push:

When a new element is added to the stack, it is placed on top of the existing elements. This is called push operation.

Consider an empty stack that can store integers.

push(5)
push(8)
push(2)

The stack will look like this:

|2|
|8|
|5|

Pop: When an element is removed from the stack, it is removed from the top of the stack. This is called a pop operation.

pop()
pop()

The stack will look like this:

|5|

27. Discuss the principle of stack data structure. Implement stack data structure using python program.

A: Stack Data structure: Advantages.

-> Easy implementation: It is easy to implement using arrays or linked lists, and its operations are simple to understand and implement.

-> Fast access time: It provides fast access time for adding and removing elements as they are added and removed from the top of the stack.

-> Used in compiler Design: It is used in compiler design for parsing and syntax analysis of programming languages.

Disadvantages:

-> Limited capacity: It has a limited capacity as it can only hold a fixed number of elements. If the stack becomes full, adding new elements may result in stack overflow, leading to the loss of data.

-> Memory management: It uses a contiguous block of memory, which can result in memory fragmentation if elements are added and removed frequently.

→ Stack overflow and underflow: It can result in stack overflow if too many elements are pushed onto the stack, and it can result in stack underflow if too many elements are popped from the stack.