Siliguri Institute of Technology

Data Structure Lab with Python

MCAN-291

Dr. Tumpa Banerjee

Assistant Professor, Department of MCA

Code: MCAN-291	**Paper: Data Structure Lab with Python**	
Contacts Hours / Week: 4	**Total Contact Hours: 40**	**Credit: 2**

Course Outcome:

After successful completion of this course, students will be able to:

- To understand linear and non-linear data structures.
- To understand different types of sorting and searching techniques.
- To know how to create an application specific data structure.
- To solve the faults / errors that may appear due to wrong choice of data structure.
- To analyze reliability of different data structures in solving different problems.

UNITS	COURSE CONTENT
1	Implementation of data structure operations (Insertion, deletion, traversing, searching) on array. Linear search, Binary search.
2	Implementation of stack, queue operation using array. Pop, Push, Insertion, deletion, Implementation of circular queue. Infix to postfix conversion, postfix expression evaluation
3	Implementation of linked lists: Single linked list, circular linked list, double linked list, doubly circular linked list. Implementation of stack and queue using linked list. Merging two linked list, Linked list representation of a polynomial, polynomial addition, polynomial multiplication.
4	Tree: creating Binary Search tree, recursive and non-recursive traversal of BST, deletion in BST, calculating height of a BST, building AVL tree.
5	Implementation of sorting techniques: selection, bubble, quick sort, insertion sort, merge sort, heap sot, implementation of priority queue. Hash table implementation.
6	Implementation of Graph: representation, searching, BFS, DFS

Module	Objective	Programs
1	Recapitulating of the Python Programming concept	Revise basic concepts of Python Programming A. Write a program to print first 5 character of your name using for loop. B. Write down the difference between 'python' command and 'import' command. C. Write a program that create a dictionary with the frequency of the vowels from an inputted string. For example: input:'institute'. Output:{'i':2,'u':1,'e':1} D. Write a program to calculate sum of the following series: 1+2+3+...+n E. Write a function that takes a string as a parameter and returns a string with every successive repetitive character replaced with a star(*). For example, 'balloon' is returned as 'bal*o*n'. F. Write a function thant takes a list of integers as a parameters andd returns third smallest number from the list. For example, input:[34,89,54,20,50,76,10,45,90] output: 34
1	Revise the concept of OPPs/class in Python Programming	Write a program to create $student$ class with the following members: Data members: $name, roll, marks$ Member functions: __init__(), initialize the object with name, roll $showdata()$ display all the details of the student $showmarks()$ display marks of the student
1	Implementing the concept of ADT using OOPs	Create an ADT of array using OOPs concept. Define a class with the name $array1$ and with the following members Data member List l Data member Size of the array max Define a member function(constructor) $\_init\_$ () which define an empty list l and define size of the list. Define member function $CreateArray()$, take input for the list l with size

		max Define member function $ShowArray()$ display all the values of the array Define member function $LinearSearch()$, search one item in the array, return index of the item Define member function $Sorting()$, arrange all the elements in sorted order Define member function $BinarySearch()$, return index of the item in the sorted array Write a main function, create an object of the class $array1$, call all the member functions of the class $array1$ and implements data structure operations on array.
2	Implementing Stack	Create an ADT for stack using OOPs concept. Define a class with the name $Stack$ and with the following members Data member List l Data member Size of the array max Data membr top to identify top of the stack Define a member function(constructor) $\_init\_$ () which define an empty list l and define size of the list and initialize top with the value -1. Define member function $Push()$, to insert new element at the top of the stack. Modify top value accordingly. Define member function $Traverse()$ to display all the values of the stack Define member function $Pop()$ to remove an element from top of the stack. . Modify top value accordingly.
2	Implementing Queue	Create an ADT for Queue using OOPs concept(Imagine Python list do not support append and pop). Define a class with the name $Queue$ and with the following members Data member List l Data member Size of the array max Data member $front$ and $rear$ Define a member function(constructor) $\_init\_$ () which define a list l with the entries zeros of size max and initialize $front$ and $rear$ with the value -1. Define member function $Insertion()$, to insert new element at the top of the rear. Define member function $Traverse()$ to display all the values of the queue Define member function $Deletion()$ to remove an element from front of the queue.
2		The queue implementation defined in the last question does not utilize the storage space effectively. This issues leads to circular queue, implement circular queue with the required operations.
3	Implement Single Linked List	Implement Single Linked List in python using the following informations: Create class $node$ with the members $data$ and $next$. Develop a class $Linkedlist$ with the following members: Data member $start$ Define function $\_init\_()$ to initialize the object of class $LinkedList$ Define a function $InsertBegining()$ to insert a new node in the beginning of the linked list Define a function $InsertEnd()$ to insert at the end of the linked list Define a function $InsertSpecified()$ to insert at specified position Define a function $DeleteStart()$ to delete the start node

		Define a function $DeleteEnd()$ to delete from the end. Define a function $DeleteSpecified()$ to delete specified node Define a function $traverse()$ to display all the data of the linked list. Define a function $Reverse()$ to reverse the order of the nodes in the linked list Define a function $Search()$ to search an element of the linked list
3	Implement circular Linked List	Implement Circular Linked List in python using the following information: Create class $node$ with the members $data$ and $next$. Develop a class $CirLinkedlist$ with the following members: Data member $start$ Define function $\_\_init\_\_()$ to initialize the object of class $CirLinkedList$ Define a function $InsertBegining()$ to insert a new node in the beginning of the linked list Define a function $InsertEnd()$ to insert at the end of the linked list Define a function $InsertSpecified()$ to insert at specified position Define a function $DeleteStart()$ to delete the start node Define a function $DeleteEnd()$ to delete from the end. Define a function $DeleteSpecified()$ to delete specified node Define a function $traverse()$ to display all the data of the linked list. Define a function $Reverse()$ to reverse the order of the nodes in the linked list Define a function $Search()$ to search an element of the linked list
3	Implement Doubly Linked List	Implement Doubly Linked List in python using the following information: Create class $node$ with the members $data$ and $next$. Develop a class $DblLinkedlist$ with the following members: Data member $start$ Define function $\_\_init\_\_()$ to initialize the object of class $DblLinkedList$ Define a function $InsertBegining()$ to insert a new node in the beginning of the linked list Define a function $InsertEnd()$ to insert at the end of the linked list Define a function $InsertSpecified()$ to insert at specified position Define a function $DeleteStart()$ to delete the start node Define a function $DeleteEnd()$ to delete from the end. Define a function $DeleteSpecified()$ to delete specified node Define a function $traverse()$ to display all the data of the linked list. Define a function $Reverse()$ to reverse the order of the nodes in the linked list Define a function $Search()$ to search an element of the linked list
3	Implement Doubly Circular Linked List	Implement Doubly Circular Linked List in python using the following information: Create class $node$ with the members $data$ and $next$. Develop a class $DblCirLinkedlist$ with the following members: Data member $start$ Define function $\_\_init\_\_()$ to initialize the object of class $DblCirLinkedList$ Define a function $InsertBegining()$ to insert a new node in the beginning of the linked list Define a function $InsertEnd()$ to insert at the end of the linked list Define a function $InsertSpecified()$ to insert at specified position Define a function $DeleteStart()$ to delete the start node Define a function $DeleteEnd()$ to delete from the end. Define a function $DeleteSpecified()$ to delete specified node Define a function $traverse()$ to display all the data of the linked list. Define a function $Reverse()$ to reverse the order of the nodes in the linked list Define a function $Search()$ to search an element of the linked list

4	Implement Binary Search Tree	Implement Binary Search tree with the following details: Create a class node with the data members $left, right$, and key, and constructor $\_init\_()$, to initialize the value of key Create a class $BnrySeacrchTree$ with the following members: Data members $root$ Define constructor $\_init\_()$ to initialize $root$ with null value Define a member function $insert(self, value)$ to insert a new node in the binary search tree Define a member function $delete(self)$ to delete a node from the binary search tree. Define a member function $preorder()$ for preorder traversal Define a member function $postorder()$ for postorder traversal Define a member function $inorder()$ for inorder traversal Define a member function $hight$ to calculate height of the tree
4	Implement balance tree	Implement AVL tree in Python programming
5	Implementing the concept of graph To calculate shortest path between any two nodes	Implement graph using python programming with the followings: Create a class $vertex$ with the properties $nodename$ and $nextNode$ Create a class $neighbor$ with the properties $Node$ and $Next$ Create a class graph with the data member $start$ Define a member function $Add\ vertex$ to add vertices in the graph Define a member function $AddEdges$ to add neighboring nodes. Define a member function BFS for traversal Define a member function DFS for traversal Define a member function $Primes$ for finding shortest path

Miscellaneous:

Q1

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

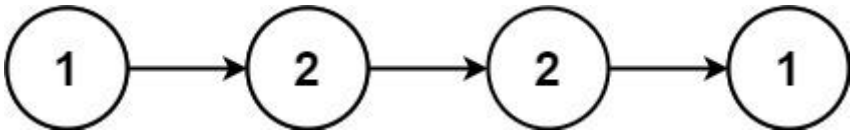
Constraints:

- 1 <= s.length <= 10^4
- s consists of parentheses only '()[]{}'.

Q2

Given the head of a singly linked list, return true if it is a palindrome.

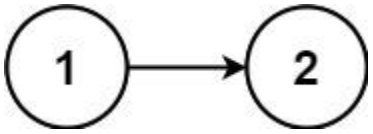
Example 1:



Input: head = [1,2,2,1]

Output: true

Example 2:



Input: head = [1,2]

Output: false

Constraints:

- The number of nodes in the list is in the range [1, 10^5].
- 0 <= Node.val <= 9

Q3

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two **distinct 0-indexed** integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the **next greater element** of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return *an array* ans *of length* nums1.length *such that* ans[i] *is the* ***next greater element*** *as described above.*

Example 1:

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.

- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

Example 2:

Input: nums1 = [2,4], nums2 = [1,2,3,4]

Output: [3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 2 is underlined in nums2 = [1,2,3,4]. The next greater element is 3.
- 4 is underlined in nums2 = [1,2,3,4]. There is no next greater element, so the answer is -1.

Constraints:

- 1 <= nums1.length <= nums2.length <= 1000
- 0 <= nums1[i], nums2[i] <= 10^4
- All integers in nums1 and nums2 are **unique**.
- All the integers of nums1 also appear in nums2.

Q4

Given two strings s and t, return true if they are equal when both are typed into empty text editors. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

Example 1:

Input: s = "ab#c", t = "ad#c"

Output: true

Explanation: Both s and t become "ac".

Example 2:

Input: s = "ab##", t = "c#d#"

Output: true

Explanation: Both s and t become "".

Example 3:

Input: s = "a#c", t = "b"

Output: false

Explanation: s becomes "c" while t becomes "b".

Constraints:

- 1 <= s.length, t.length <= 200
- s and t only contain lowercase letters and '#' characters.

Q5

You are given a string `s` consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

Example 1:

Input: s = "abbaca"

Output: "ca"

Explanation:

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Example 2:

Input: s = "azxxzy"

Output: "ay"

Q6

Given the array prices where prices[i] is the price of the ith item in a shop. There is a special discount for items in the shop, if you buy the ith item, then you will receive a discount equivalent to prices[j] where j is the minimum index such that j > i and prices[j] <= prices[i], otherwise, you will not receive any discount at all.

Return an array where the ith element is the final price you will pay for the ith item of the shop considering the special discount.

Example 1:

Input: prices = [8,4,6,2,3]

Output: [4,2,4,2,3]

Explanation:

- For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.
- For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.
- For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.
- For items 3 and 4 you will not receive any discount at all.

Example 2:

Input: prices = [1,2,3,4,5]

Output: [1,2,3,4,5]

Explanation: In this case, for all items, you will not receive any discount at all.

Example 3:

Input: prices = [10,1,1,6]

Output: [9,0,1,6]

Constraints:

- 1 <= prices.length <= 500
- 1 <= prices[i] <= 10^3

Q7

Given a string s of lower and upper case English letters.

A good string is a string which doesn't have **two adjacent characters** s[i] and s[i + 1] where:

- 0 <= i <= s.length - 2
- s[i] is a lower-case letter and s[i + 1] is the same letter but in upper-case or **vice-versa**.

To make the string good, you can choose **two adjacent** characters that make the string bad and remove them. You can keep doing this until the string becomes good.

Return *the string* after making it good. The answer is guaranteed to be unique under the given constraints.

Notice that an empty string is also good.

Example 1:

Input: s = "leEeetcode"

Output: "leetcode"

Explanation: In the first step, either you choose i = 1 or i = 2, both will result "leEeetcode" to be reduced to "leetcode".

Example 2:

Input: s = "abBAcC"

Output: ""

Explanation: We have many possible scenarios, and all lead to the same answer. For example:

"abBAcC" --> "aAcC" --> "cC" --> ""

"abBAcC" --> "abBA" --> "aA" --> ""

Example 3:

Input: s = "s"

Output: "s"

Constraints:

- 1 <= s.length <= 100
- s contains only lower and upper case English letters.

Q8

Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

Output: 3

Example 2:

Input: nums = [2,2,1,1,1,2,2]

Output: 2

Constraints:

- n == nums.length
- 1 <= n <= 5 * 10^4
- -10^9 <= nums[i] <= 10^9

Q9

Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

Example 1:

Input: arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2

Output: 2

Explanation:

For arr1[0]=4 we have:
|4-10|=6 > d=2
|4-9|=5 > d=2
|4-1|=3 > d=2
|4-8|=4 > d=2
For arr1[1]=5 we have:
|5-10|=5 > d=2
|5-9|=4 > d=2
|5-1|=4 > d=2
|5-8|=3 > d=2
For arr1[2]=8 we have:
|8-10|=2 <= d=2
|8-9|=1 <= d=2
|8-1|=7 > d=2
|8-8|=0 <= d=2

Example 2:
Input: arr1 = [1,4,2,3], arr2 = [-4,-3,6,10,20,30], d = 3
Output: 2

Example 3:

Input: arr1 = [2,1,100,3], arr2 = [-5,-2,10,-3,7], d = 6
Output: 1

Constraints:

- 1 <= arr1.length, arr2.length <= 500
- -1000 <= arr1[i], arr2[j] <= 1000
- 0 <= d <= 100

Q10

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must be unique and you may return the result in any order.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Explanation: [4,9] is also accepted.
```

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

Q11

Alice and Bob have a different total number of candies. You are given two integer arrays aliceSizes and bobSizes where aliceSizes[i] is the number of candies of the i^{th} box of candy that Alice has and bobSizes[j] is the number of candies of the j^{th} box of candy that Bob has.

Since they are friends, they would like to exchange one candy box each so that after the exchange, they both have the same total amount of candy. The total amount of candy a person has is the sum of the number of candies in each box they have.

Return a*n integer array* answer *where* answer[0] *is the number of candies in the box that Alice must exchange, and* answer[1] *is the number of candies in the box that Bob must exchange*. If there are multiple answers, you may **return any** one of them. It is guaranteed that at least one answer exists.

Example 1:

Input: aliceSizes = [1,1], bobSizes = [2,2]

Output: [1,2]

Example 2:

Input: aliceSizes = [1,2], bobSizes = [2,3]

Output: [1,2]

Example 3:

Input: aliceSizes = [2], bobSizes = [1,3]

Output: [2,3]

Constraints:

- 1 <= aliceSizes.length, bobSizes.length <= 10^4
- 1 <= aliceSizes[i], bobSizes[j] <= 10^5
- Alice and Bob have a different total number of candies.
- There will be at least one valid answer for the given input.

Q12

You are given an m x n binary matrix mat of 1's (representing soldiers) and 0's (representing civilians). The soldiers are positioned in front of the civilians. That is, all the 1's will appear to the left of all the 0's in each row.

A row i is **weaker** than a row j if one of the following is true:

- The number of soldiers in row i is less than the number of soldiers in row j.
- Both rows have the same number of soldiers and i < j.

Return *the indices of the* k ***weakest*** *rows in the matrix ordered from weakest to strongest*.

Example 1:
Input: mat =
[[1,1,0,0,0],
[1,1,1,1,0],
[1,0,0,0,0],
[1,1,0,0,0],
[1,1,1,1,1]],
k = 3
Output: [2,0,3]

Explanation:
The number of soldiers in each row is:
- Row 0: 2
- Row 1: 4
- Row 2: 1
- Row 3: 2
- Row 4: 5
The rows ordered from weakest to strongest are [2,0,3,1,4].

Example 2:
Input: mat =
[[1,0,0,0],
 [1,1,1,1],
 [1,0,0,0],
 [1,0,0,0]],
k = 2
Output: [0,2]
Explanation:
The number of soldiers in each row is:
- Row 0: 1
- Row 1: 4
- Row 2: 1
- Row 3: 1
The rows ordered from weakest to strongest are [0,2,3,1].

Constraints:

- m == mat.length
- n == mat[i].length
- 2 <= n, m <= 100
- 1 <= k <= m
- matrix[i][j] is either 0 or 1.