Data Structure

What is a Data Structure?

A data structure is a specialized format for organizing, processing, retrieving, and storing data. It is a way of arranging data on a computer so that it can be used efficiently.

Think of it like this: You can store books in a pile, on a shelf in order, or in separate boxes by genre. Each method of organization is a "data structure" for your books, and each has its own advantages for finding, adding, or removing a book.

Key Points:

- **Organization:** Data structures define the relationship between the data items.
- **Operations:** They support various operations like traversal, insertion, deletion, searching, and sorting.
- **Efficiency:** The choice of data structure affects the performance (speed and memory usage) of a program.

Classification of Data Structures:

1) **Primitive:** Basic data types directly supported by the programming language (e.g., `int`, `float`, `char`).
2) **Non-Primitive:** Data structures derived from primitive types are called as non-primitive data structrures. They can be further categorized as following:
    i) **Linear:** In linear data structures data elements are arranged in a sequential order. (e.g., Array, Stack, Queue, Linked List).
    ii) **Non-Linear:** In non linear data structures data elements are not arranged in a sequence. (e.g., Trees, Graphs).

Stack

A stack is a **Linear Data Structure** that follows a particular order in which operations are performed. The order is **LIFO (Last In, First Out)** or **FILO (First In, Last Out)**.

Real-Life example: Stack of Plates

Imagine a stack of plates where you can only add a new plate to the **top** of the stack. You can only remove a plate from the **top** of the stack. The plate that was placed last is the first one to be removed. This is the core principle of a stack.

Basic Terminology of a stack:

- **Top:** The pointer that points to the topmost element in the stack.
- **Base/Bottom:** The pointer that points to the bottommost element. It remains fixed.
- **Push:** The operation of adding an element to the stack.
- **Pop:** The operation of removing an element from the stack.

Operations on Stack

The two fundamental operations on a stack are **Push** and **Pop**.

1) PUSH Operation: The `Push` operation adds an element to the top of the stack.

Algorithm to push:

- Check if the stack is full (`isFull`), it results in an **Overflow** condition.
- If the stack is not full, increment the `top` pointer.
- Add the new element to the location pointed by `top`.

Example:
Let's push `10`, then `20`, then `30` onto an empty stack.

Step 1: Push 10
Stack: [`10`] Top -> 0
Step 2: Push 20
Stack: [`10, 20`] Top -> 1
Step 3: Push 30
Stack: [`10, 20, 30`] Top -> 2

2) POP Operation : The `Pop` operation removes the element from the top of the stack.

Algorithm:

- Check if the stack is empty (`isEmpty`), it results in an **Underflow(stack empty)** condition.
- If the stack is not empty, access the element pointed by `top`.
- Decrement the `top` pointer (Pop the topmost element).

Example (Continuing from the previous stack):
Let's pop elements from the stack `[10, 20, 30]`.

```
Initial Stack: [10, 20, 30]                Top -> 2

Step 1: Pop (returns 30)
Stack: [10, 20]                            Top -> 1

Step 2: Pop (returns 20)
Stack: [10]                                Top -> 0

Step 3: Pop (returns 10)
Stack: []                                  Top -> -1 (Empty)
```

Implementation of Stack using List in Python

In Python, we can easily implement a stack using a list. The `append()` method is used for `Push` and the `pop()` method is used for `Pop`.

```
stack = [ ]                # Creating a stack

# PUSH Operation

def push(element):
    stack.append(element)    # append() adds to the end of the list (top of the stack)
    print(f"Pushed {element}. Stack: {stack}")

# POP Operation

def pop():
    if len(stack) == 0:                    # Check if stack is empty
        print("Stack is empty! Underflow!")
        return None
    else:
        popped_element = stack.pop() # pop() removes and returns the last element
        print(f"Popped {popped_element}. Stack: {stack}")
        return popped_element
```

```
# Function to check if stack is empty

def is_empty():
    return len(stack) == 0

# Function to check the top element (Peek)
def peek():
    if not is_empty():
        return stack[-1]
    else:
        print("Stack is empty!")
        return None

print("Stack Implementation using List")

push(10)
push(20)
push(30)
print("\nTop element is:", peek())
pop()
pop()
pop()
pop() # This will cause an underflow
```

Output of the Code:

```
Pushed 10. Stack: [10]
Pushed 20. Stack: [10, 20]
Pushed 30. Stack: [10, 20, 30]
Top element is: 30
Popped 30. Stack: [10, 20]
Popped 20. Stack: [10]
Popped 10. Stack: []
Stack is empty! Underflow!
```

Important Questions on Stack

Q1: In a Stack, if you try to remove an element from an empty stack, it is called as _____.

Ans: Underflow

Q2: What will be the top element of the stack after these operations?
PUSH(10), PUSH(20), POP(), PUSH(30), PUSH(40), POP()
Ans:

30 *(Sequence: Push 10, Push 20, Pop removes 20, Push 30, Push 40, Pop removes 40. Top is now 30)*.

Q3: State True or False:
"A stack is a linear data structure that follows the First-In-First-Out (FIFO) principle."

Ans: False

Q4: Write the output of the following operations performed on a stack implemented using a list in Python. Assume the stack is initially empty.

```
stack = []
stack.append('A')
stack.append('B')
print(stack.pop())
stack.append('C')
print(stack.pop())
print(stack.pop())
```

Ans:

```
B
C
A
```

Q5: Write a function PUSH() in Python to add a book in a stack of books considering the following:

- The stack is implemented as a list.
- The book name is taken as user input.

Ans:

```
def PUSH(books):
    book_name = input("Enter the book name: ")
    books.append(book_name)
```

Q6: A stack STK has the following entries: [45, 78, 23, 11, 90] (where 45 is at the bottom and 90 is at the top). What will be the status of the stack after performing the following operations? Show the stack after each step.

POP(STK)
POP(STK)
PUSH(STK, 50)
POP(STK)

Ans:

Stack Status:

Initial stack is: [45, 78, 23, 11, 90] (Top=90)

After POP(STK): [45, 78, 23, 11] (Top=11)

After POP(STK): [45, 78, 23] (Top=23)

After PUSH(STK, 50): [45, 78, 23, 50] (Top=50)

After POP(STK): [45, 78, 23] (Top=23)

Q7: Write an algorithm for PUSH operation in a stack that is implemented using a list. Assume the list has a fixed size MAX.

Ans:

1. IF TOP == MAX - 1 THEN
    2. PRINT "Stack Overflow"
3. ELSE
    4. TOP = TOP + 1
    5. STK[TOP] = ELEMENT
6. END IF

Q8: Evaluate the following postfix expression using a stack. Show the status of the stack after every operation. 5, 3, +, 2, *, 4, -

Ans:

```
Steps:

Push 5 -> Stack: [5]
Push 3 -> Stack: [5, 3]
+: Pop 3, Pop 5 -> 5+3=8 -> Push 8 -> Stack: [8]
Push 2 -> Stack: [8, 2]
*: Pop 2, Pop 8 -> 8*2=16 -> Push 16 -> Stack: [16]
Push 4 -> Stack: [16, 4]
-: Pop 4, Pop 16 -> 16-4=12 -> Push 12 -> Stack: [12]


Final result is 12.
```

V. Imp Questions:

Ques[compartmet exam 2025]:

A stack named KeyStack contains records of some computer keyboards. Each record is represented as a list containing Make, Keys, Connectivity. The Make and Connectivity are strings, and Keys is an integer. For example, a record in the stack may be ['Hitech', 105, 'USB'].

Write the following user-defined functions in Python to perform the specified operations on KeyStack :

(I)push_key(KeyStack, new_key): This function takes the stack KeyStack and a new record new_key as arguments and pushes this new record onto the stack.

(II)pop_key(KeyStack): This function pops the topmost record from the stack and returns it. If the stack is already empty, the function should display the message “Underflow”.

(III) isEmpty(KeyStack): This function checks whether the stack is empty. If the stack is empty, the function should return True, otherwise the function should return False.

Ans:

```
# Function to push a new keyboard record onto the stack

def push_key(KeyStack, new_key):

    KeyStack.append(new_key)

    print(f"Added keyboard: {new_key}")


# Function to pop the topmost record from the stack

def pop_key(KeyStack):

    if isEmpty(KeyStack):

        print("Underflow")

        return None
```

```
    else:
        popped_record = KeyStack.pop()
        print(f"Removed keyboard: {popped_record}")
        return popped_record

# Function to check if the stack is empty
def isEmpty(KeyStack):
    return len(KeyStack) == 0
```

Ques: Write the following user-defined functions in Python :

(I) the push_vowels(S,St): Here S is a string and St is a list representing a stack. The function should push all the vowels of the string S onto the stack St. For example, if string S is "Easy Concepts", then the function push_vowels() should push the elements 'E' ,'a ','o' , 'e' onto the stack.

(II) pop_one(St) : The function should pop an element from the stack St, and return this element. If the stack is empty, then the function should display the message 'Stack Underflow', and return None.

(III) display_all(St): The function should display all the elements of the stack St, without deleting them. If the stack is empty, the function should display the message 'Empty Stack'.

Ans:

```
def push_vowels(S, St):
    vowels = "AEIOUaeiou"
    for char in S:
        if char in vowels:
            St.append(char)
    print(f"Pushed vowels from '{S}' onto stack")

def pop_one(St):
    if len(St) == 0:
        print("Stack Underflow")
        return None
    else:
        popped_element = St.pop()
        return popped_element
```

```
def display_all(St):
    if len(St) == 0:
        print("Empty Stack")
    else:
        print("Stack elements (top to bottom):")
        for i in range(len(St)-1, -1, -1):
            print(St[i])
# ------------------------ function calls-------------------------------
stack = [ ]
S= "Easy Concepts"
push_vowels(S, stack)
print("Element", pop_one(stack), "deleted")
display_all(stack)
```