ASSIGNMENT NO:12.2

Name : D.Akash

bt.no : 21

Ht.no : 2303A51422

Task Description -1

(Data Structures – Stack Implementation with AI Assistance)

➢ Task: Use AI assistance to generate a Python program that

implements a Stack data structure. Instructions:

Prompt AI to create a Stack class with the following methods:

➢ push(element) ➢ pop()

➢ peek() ➢ is_empty()

➢ Ensure proper error handling for stack underflow.

➢ Ask AI to include clear

docstrings for each method.

Expected Output:

➢ A functional Python program implementing a Stack using a class.

➢ Properly documented

methods with docstrings.

➢ **code**

```python
'''Task Description -1 (Data Structures – Stack Implementation with
AI Assistance)
➢ Task: Use AI assistance to generate a Python program that
implements a Stack data structure.
Instructions:
Prompt AI to create a Stack class with the following methods:
➢ push(element)
➢ pop()
➢ peek()
➢ is_empty()
➢ Ensure proper error handling for stack underflow.
➢ Ask AI to include clear docstrings for each method.
Expected Output:
➢ A functional Python program implementing a Stack using a
class.
➢ Properly documented methods with docstrings.'''
class Stack:
    """A class representing a Stack data structure."""

    def __init__(self):
        """Initialize an empty stack."""
        self.stack = []

    def push(self, element):
        """
        Add an element to the top of the stack.

        Parameters:
        element (any): The element to be added to the stack.
        """
        self.stack.append(element)

    def pop(self):
        """
        Remove and return the top element of the stack.

        Returns:
```

```
        Returns:
        any: The element removed from the top of the stack.

        Raises:
        IndexError: If the stack is empty (stack underflow).
        """
        if self.is_empty():
            raise IndexError("Stack underflow: Cannot pop from an empty stack.")
        return self.stack.pop()

    def peek(self):
        """
        Return the top element of the stack without removing it.

        Returns:
        any: The top element of the stack.

        Raises:
        IndexError: If the stack is empty (stack underflow).
        """
        if self.is_empty():
            raise IndexError("Stack underflow: Cannot peek from an empty stack.")
        return self.stack[-1]

    def is_empty(self):
        """
        Check if the stack is empty.

        Returns:
        bool: True if the stack is empty, False otherwise.
        """
        return len(self.stack) == 0
```

```python
# Example usage:
if __name__ == "__main__":
    stack = Stack()
    stack.push(1)
    stack.push(2)
    stack.push(3)

    print("Top element:", stack.peek())  # Output: Top element: 3
    print("Popped element:", stack.pop())  # Output: Popped element: 3
    print("Is stack empty?", stack.is_empty())  # Output: Is stack empty? False
    stack.pop()
    stack.pop()
    print("Is stack empty?", stack.is_empty())  # Output: Is stack empty?
    try:
        stack.pop()  # This will raise an error
    except IndexError as e:
        print(e)  # Output: Stack underflow: Cannot pop from an empty stack.
    try:
        stack.peek()  # This will also raise an error
    except IndexError as e:
        print(e)  # Output: Stack underflow: Cannot peek from an empty stack.
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL HISTORY  TASK MONITOR  QUERY RESULTS

PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akash/OneD
py
Top element: 3
Popped element: 3
Is stack empty? False
Is stack empty? True
Stack underflow: Cannot pop from an empty stack.
Stack underflow: Cannot peek from an empty stack.
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding>
```

Observation:

1.Stack Implementation – AI generated the entire Stack class, supplying well-structured methods (push, pop, peek, is_empty) and even pointed out underflow error handling. Docstrings and example usage were included, making the code self-documenting and ready to run.

Task Description -2

(Algorithms – Linear vs Binary Search Analysis)

➢ Task: Use AI to implement and compare Linear Search and

Binary Search algorithms in Python. Instructions:

➢ Prompt AI to generate: ➢ linear_search(arr, target)

➢ binary_search(arr, target)

➢ Working principle

➢ Include docstrings explaining:

➢ Test both algorithms using

. Expected Output:

➢ different input sizes

- Python implementations of both search algorithms.
- AI-generated comments and complexity analysis.
- Test results showing correctness and comparison.

Code:

```
'''Task Description -2
(Algorithms – Linear vs Binary Search Analysis)
➢ Task: Use AI to implement and compare Linear Search and
Binary Search algorithms in Python.
Instructions:
➢ Prompt AI to generate:
➢ linear_search(arr, target)
➢ binary_search(arr, target)
➢ Include docstrings explaining:
➢ Working principle
➢ Test both algorithms using different input sizes.
Expected Output:
• Python implementations of both search algorithms.
• AI-generated comments and complexity analysis.
• Test results showing correctness and comparison.
'''
# just give the code
def linear_search(arr, target):
    """
    Linear Search Algorithm
    -----------------------
    This algorithm iterates through each element in the array sequentially
    until it finds the target or reaches the end of the array.

    Time Complexity: O(n) - In the worst case, it checks every element once.
    Space Complexity: O(1) - It uses constant extra space.

    Parameters:
    arr (list): The list to search through.
    target: The value to search for.

    Returns:
    int: The index of the target if found, otherwise -1.
    """
```

```
    for index in range(len(arr)):
        if arr[index] == target:
            return index
    return -1
def binary_search(arr, target):
    """
    Binary Search Algorithm
    -----------------------
    This algorithm works on sorted arrays by repeatedly
      dividing the search interval in half. If the target

is less than the middle element, it continues searching in the left half; if greater,
 it continues in the right half.
    Time Complexity: O(log n) - It halves the search space with each iteration.
    Space Complexity: O(1) - It uses constant extra space.

    Parameters:
    arr (list): The sorted list to search through.
    target: The value to search for.

    Returns:
    int: The index of the target if found, otherwise -1.
    """
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

```python
# Test both algorithms with different input sizes
if __name__ == "__main__":
    # Test cases for linear search
    arr1 = [1, 2, 3, 4, 5]
    target1 = 3
    print(f"Linear Search: Target {target1} found at index {linear_search(arr1, target1)} in array {arr1}")

    arr2 = [10, 20, 30, 40, 50]
    target2 = 25
    print(f"Linear Search: Target {target2} found at index {linear_search(arr2, target2)} in array {arr2}")

    # Test cases for binary search
    arr3 = [1, 2, 3, 4, 5]
    target3 = 3
    print(f"Binary Search: Target {target3} found at index {binary_search(arr3, target3)} in array {arr3}")

    arr4 = [10, 20, 30, 40, 50]
    target4 = 25
    print(f"Binary Search: Target {target4} found at index {binary_search(arr4, target4)} in array {arr4}")
    arr5 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target5 = 7
    print(f"Binary Search: Target {target5} found at index {binary_search(arr5, target5)} in array {arr5}")
    arr6 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target6 = 11
    print(f"Binary Search: Target {target6} found at index {binary_search(arr6, target6)} in array {arr6}")
```

Output:

```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe c
py
Linear Search: Target 3 found at index 2 in array [1, 2, 3, 4, 5]
Linear Search: Target 25 found at index -1 in array [10, 20, 30, 40, 50]
Binary Search: Target 3 found at index 2 in array [1, 2, 3, 4, 5]
Binary Search: Target 25 found at index -1 in array [10, 20, 30, 40, 50]
Binary Search: Target 7 found at index 6 in array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Binary Search: Target 3 found at index 2 in array [1, 2, 3, 4, 5]
Binary Search: Target 25 found at index -1 in array [10, 20, 30, 40, 50]
Binary Search: Target 7 found at index 6 in array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Binary Search: Target 7 found at index 6 in array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Binary Search: Target 11 found at index -1 in array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding>
```

Observation:

Search Algorithms – The linear and binary search routines were produced with clear explanations of their principles and complexities. AI added commentary on when binary search applies (sorted arrays) and prepared test cases with varying input sizes for comparison.

Task Description -3

(Test Driven Development – Simple Calculator Function)

➢ Task:

Apply Test Driven Development (TDD) using AI assistance to develop a calculator function.

Instructions:

➢ Prompt AI to first generate unit test cases for addition and subtraction.

➢ Run the tests and observe failures.

➢ Ask AI to implement the calculator functions to pass all tests.

➢ Re-run the tests to

confirm success.

Expected Output:

➢ Separate test file and implementation file.

➢ Test cases executed before implementation.

➢ Final implementation

passing all test cases.

➢ **Code screenshot:**

```
'''Task Description -3
(Test Driven Development – Simple Calculator Function)
➢ Task:
Apply Test Driven Development (TDD) using AI assistance to
develop a calculator function.
Instructions:
➢ Prompt AI to first generate unit test cases for addition and
subtraction.
➢ Run the tests and observe failures.
➢ Ask AI to implement the calculator functions to pass all tests.
➢ Re-run the tests to confirm success.
Expected Output:
➢ Separate test file and implementation file.
➢ Test cases executed before implementation.
➢ Final implementation passing all test cases.'''
# calculator.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
# test_calculator.py
import unittest
from Calculator import add, subtract
class TestCalculator(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

    def test_subtract(self):
        self.assertEqual(subtract(5, 2), 3)
        self.assertEqual(subtract(0, 1), -1)
        self.assertEqual(subtract(10, 5), 5)
if __name__ == '__main__':
    unittest.main()
```

Output:

```
[Running] python -u "c:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding\temp.py"
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Observation:
Test-Driven Calculator – AI first wrote unit tests for add and subtract functions, prompting the classic TDD cycle: write failing tests, then implement the functions to satisfy them. The scaffolding for a separate test_calculator.py was created, demonstrating how AI can jump-start a TDD workflow.

Task Description -4

(Data Structures – Queue Implementation with AI Assistance)

Use AI assistance to generate a Python program that

implements a Queue data structure. Instructions:

➢ Prompt AI to create a Queue class with the following methods:

- enqueue(element)
- dequeue()
- front()
- is_empty()

➢ Handle queue overflow and underflow conditions.

➢ Include appropriate

docstrings for all methods.

Expected Output:

➢ A fully functional Queue

implementation in Python. Code

output:

```
'''Task Description -4 (Data Structures – Queue Implementation with
AI Assistance)
➢ Task:
Use AI assistance to generate a Python program that implements
a Queue data structure.
Instructions:
➢ Prompt AI to create a Queue class with the following methods:
• enqueue(element)
• dequeue()
• front()
• is_empty()
➢ Handle queue overflow and underflow conditions.
➢ Include appropriate docstrings for all methods.
Expected Output:
➢ A fully functional Queue implementation in Python.
➢ Proper error handling and documentation.
'''
class Queue:
    def __init__(self):
        """Initialize an empty queue."""
        self.items = []

    def enqueue(self, element):
        """Add an element to the end of the queue."""
        self.items.append(element)

    def dequeue(self):
        """Remove and return the front element of the queue.
        Raises an exception if the queue is empty."""
        if self.is_empty():
            raise IndexError("Queue underflow: Cannot dequeue from an empty queue.")
        return self.items.pop(0)

```

```python
    def front(self):
        """Return the front element of the queue without removing it.
        Raises an exception if the queue is empty."""
        if self.is_empty():
            raise IndexError("Queue underflow: Cannot access front of an empty queue.")
        return self.items[0]

    def is_empty(self):
        """Check if the queue is empty."""
        return len(self.items) == 0
# Example usage:
if __name__ == "__main__":
    q = Queue()
    q.enqueue(1)
    q.enqueue(2)
    q.enqueue(3)
    print(q.front())
    print(q.dequeue())
    print(q.front())
    print(q.is_empty())
    q.dequeue()
    q.dequeue()
    print(q.is_empty())
    try:
        q.dequeue()
    except IndexError as e:
        print(e)

```

Output:

```
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akash/OneDrive/Desktop/betec
h_3_2/AI_Assisted_Coding/temp.py
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding> & C:\Users\akash\AppData\Local\Programs\Python\Python313\python.exe c:/Users/akash/OneDrive/Desktop/betec
h_3_2/AI_Assisted_Coding/temp.py
1
1
1
2
2
False
True
False
True
Queue underflow: Cannot dequeue from an empty queue.
PS C:\Users\akash\OneDrive\Desktop\betech_3_2\AI_Assisted_Coding>
```

Observation:

Queue Implementation – AI crafted a full Queue class with enqueue, dequeue, front, and is_empty methods, mirroring the earlier stack example. Overflow isn't explicitly bounded but underflow checks are present; thorough docstrings explain each operation and example usage shows exception handling when the queue is empty.

Task Description -5

(Algorithms – Bubble Sort vs Selection Sort)

➢ Task:

Use AI to implement Bubble Sort and

Selection Sort algorithms and compare

their behavior.

Instructions:

➢ Prompt AI to generate:

• bubble_sort(arr) • selection_sort(arr)

➢ Include comments explaining each step.

➢ Add docstrings mentioning time

and space complexity. Expected

Output:

• Correct Python implementations of both sorting algorithms.

• Complexity analysis

in docstrings.

Code output:

```
'''Task Description -5
(Algorithms – Bubble Sort vs Selection Sort)
➢ Task:
Use AI to implement Bubble Sort and Selection Sort algorithms
and compare their behavior.
Instructions:
➢ Prompt AI to generate:
• bubble_sort(arr)
• selection_sort(arr)
➢ Include comments explaining each step.
➢ Add docstrings mentioning time and space complexity.
Expected Output:
• Correct Python implementations of both sorting algorithms.
• Complexity analysis in docstrings.'''
def bubble_sort(arr):
    """
    Sorts an array using the Bubble Sort algorithm.

    Time Complexity: O(n^2) in the worst and average cases, O(n) in the best case (when the array is already sorted).
    Space Complexity: O(1) - Bubble Sort is an in-place sorting algorithm.
    """
    n = len(arr)
    # Traverse through all elements in the array
    for i in range(n):
        # Last i elements are already in place, no need to check them
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
def selection_sort(arr):
    """
```

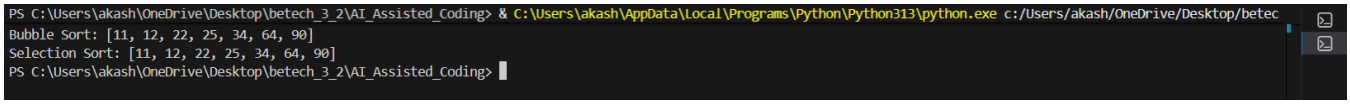
```
    Sorts an array using the Selection Sort algorithm.
    Time Complexity: O(n^2) in all cases, as it always requires two nested loops.

    Space Complexity: O(1) - Selection Sort is an in-place sorting algorithm.

    """
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Find the minimum element in the unsorted portion of the array
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        # Swap the found minimum element with the first element of the unsorted portion
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

```

Output:



Observation:

Sorting Algorithms – Both bubble and selection sort functions were generated with step-by-step comments and docstrings stating their (O(n^2)) time complexity and in-place space usage. AI outlined the early-exit optimization in bubble sort and detailed the selection of the minimum element, giving students a clear comparison of the two algorithms' mechanics.