AI ASSISTANT CODING

ASSIGNMENT-12

Name: Kommu Madhupriya

Hallticket:2303A51583

Batch:22

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.

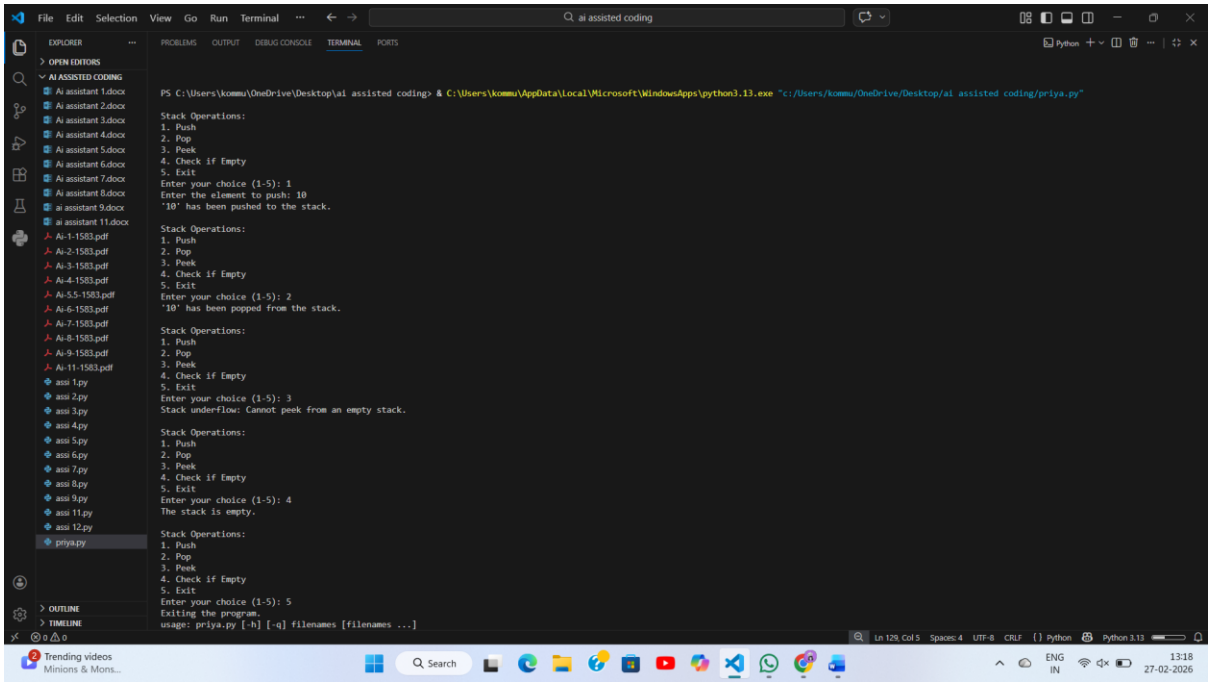
Code:

```python
#Generate a Python program that implements a Stack data structure using a
#class. Include the methods push(element), pop(), peek(), and is_empty(). Add
#proper error handling for stack underflow and include clear docstrings for each
#method explaining their functionality. Take input from user.
from py_compile import main
class Stack:
    """
    A class to represent a Stack data structure.
    Methods
    -------
    push(element):
        Adds an element to the top of the stack.
    pop():
        Removes and returns the top element of the stack. Raises an error if the stack is empty.
    peek():
        Returns the top element of the stack without removing it. Raises an error if the stack is empty.
    is_empty():
        Returns True if the stack is empty, False otherwise.
    """
    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
        -------
        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 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
# Example usage
if __name__ == "__main__":
    stack = Stack()
    while True:
        print("\nStack Operations:")
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Check if Empty")
        print("5. Exit")
        choice = input("Enter your choice (1-5): ")
        if choice == '1':
            element = input("Enter the element to push: ")
            stack.push(element)
            print(f"'{element}' has been pushed to the stack.")
        elif choice == '2':
            try:
                popped_element = stack.pop()
                print(f"'{popped_element}' has been popped from the stack.")
            except IndexError as e:
                print(e)
        elif choice == '3':
            try:
                top_element = stack.peek()
                print(f"The top element is: '{top_element}'")
            except IndexError as e:
                print(e)
        elif choice == '4':
            if stack.is_empty():
                print("The stack is empty.")
            else:
                print("The stack is not empty.")
        elif choice == '5':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 5.")
if __name__ == "__main__":
    main()
```

Expected Output:

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

➢ Properly documented methods with docstrings.



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.

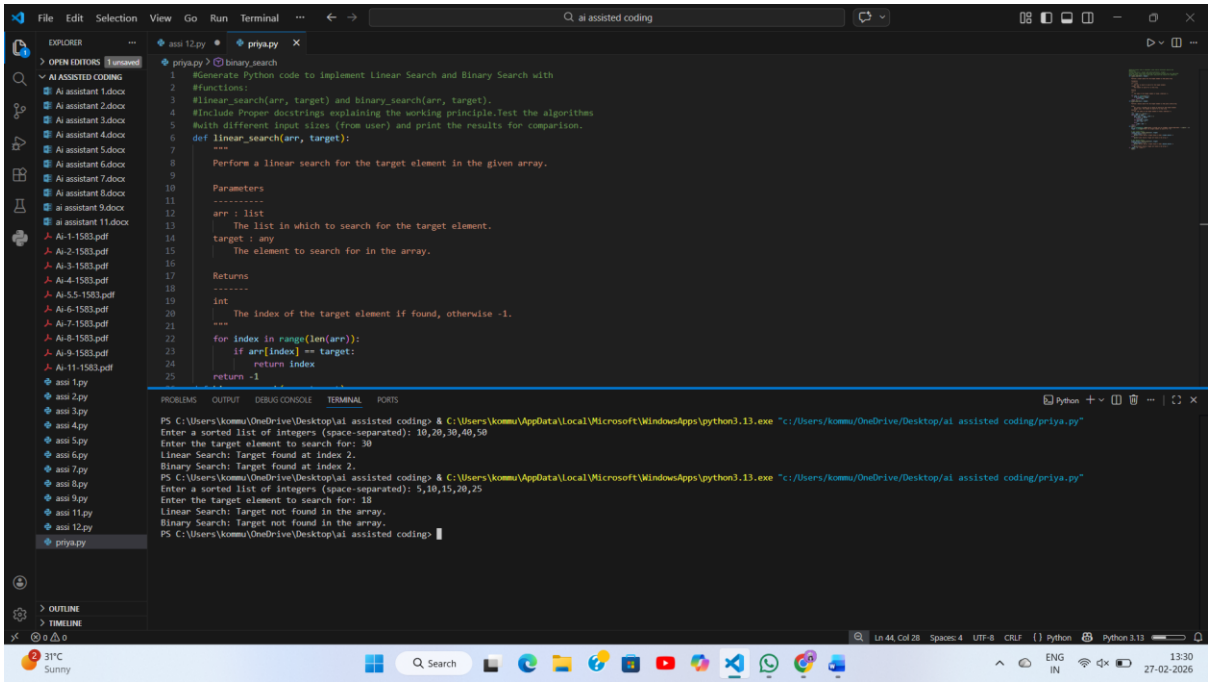
Code:

assi 12.py ● priya.py ×

priya.py > binary_search

```python
#Generate Python code to implement Linear Search and Binary Search with
#functions:
#linear_search(arr, target) and binary_search(arr, target).
#Include Proper docstrings explaining the working principle.Test the algorithms
#with different input sizes (from user) and print the results for comparison.
def linear_search(arr, target):
    """
    Perform a linear search for the target element in the given array.

    Parameters
    ----------
    arr : list
        The list in which to search for the target element.
    target : any
        The element to search for in the array.

    Returns
    -------
    int
        The index of the target element if found, otherwise -1.
    """
    for index in range(len(arr)):
        if arr[index] == target:
            return index
    return -1
def binary_search(arr, target):
    """
    Perform a binary search for the target element in the given sorted array.

    Args:
        arr (list): A sorted list in which to search for the target element.
        target (any): The element to search for in the array.
    Returns:
        int: The index of the target element 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
def main():
    arr = list(map(int, input("Enter a sorted list of integers (space-separated): ").split(',')))
    target = int(input("Enter the target element to search for: "))

    # Test Linear Search
    linear_result = linear_search(arr, target)
    if linear_result != -1:
        print(f"Linear Search: Target found at index {linear_result}.")
    else:
        print("Linear Search: Target not found in the array.")

    # Test Binary Search
    binary_result = binary_search(arr, target)
    if binary_result != -1:
        print(f"Binary Search: Target found at index {binary_result}.")
    else:
        print("Binary Search: Target not found in the array.")
if __name__ == "__main__":
    main()
```

Expected Output:



• Python implementations of both search algorithms.

• AI-generated comments and complexity analysis.

• Test results showing correctness and 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.

Code:

```python
import unittest
from calculator import add, subtract
class TestCalculator(unittest.TestCase):
    """Unit test cases for the simple calculator functions."""

    def test_add(self):
        """Test the addition function."""
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

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

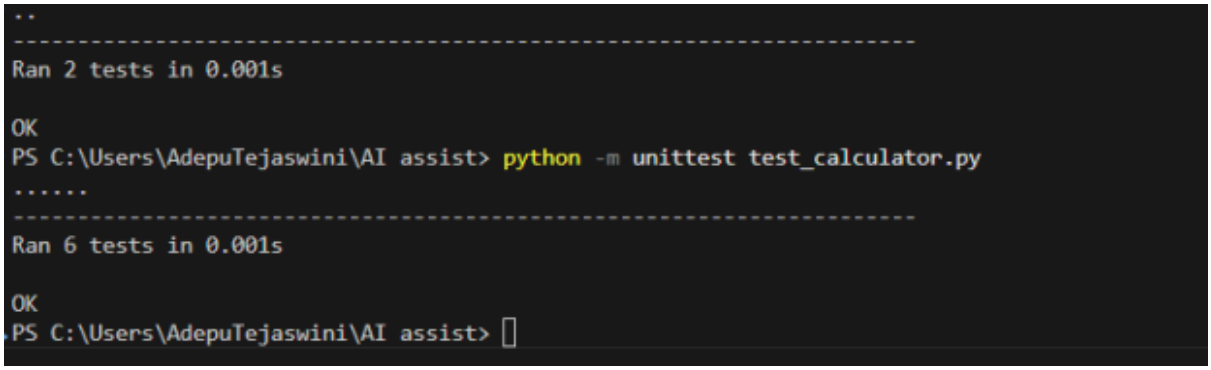
```python
import unittest
from calculator import add, subtract

class TestCalculator(unittest.TestCase):

    def test_add_positive(self):
        self.assertEqual(add(5, 3), 8)

    def test_add_negative(self):
        self.assertEqual(add(-2, -3), -5)

    def test_add_zero(self):
        self.assertEqual(add(0, 5), 5)

    def test_subtract_positive(self):
        self.assertEqual(subtract(10, 4), 6)

    def test_subtract_negative(self):
        self.assertEqual(subtract(-5, -2), -3)

    def test_subtract_zero(self):
        self.assertEqual(subtract(5, 0), 5)

if __name__ == "__main__":
    unittest.main()
```

Expected Output:

➢ Separate test file and implementation file.

➢ Test cases executed before implementation.

➢ Final implementation passing all test cases.



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()

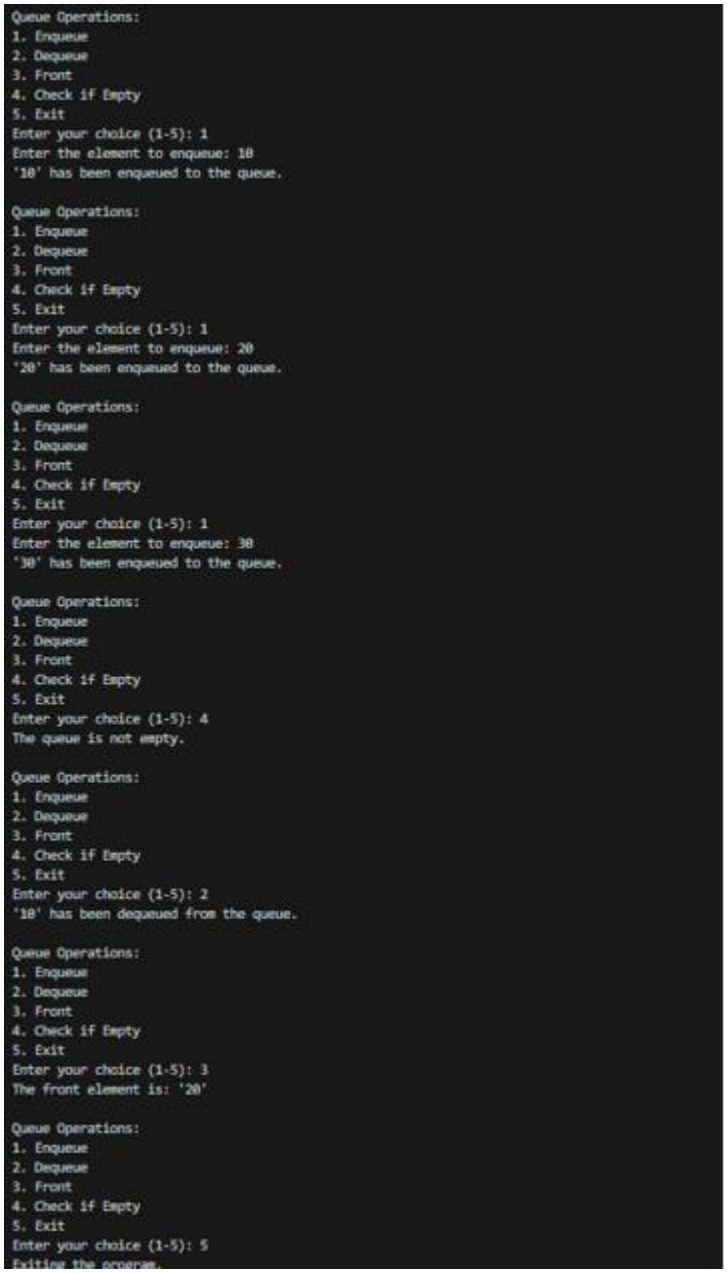
• is_empty()

➢ Handle queue overflow and underflow conditions.

➢ Include appropriate docstrings for all methods.

```python
class Queue:
    """A class representing a Queue data structure."""

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

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

        Args:
            element: The element to be added to the queue.
        """
        self.queue.append(element)

    def dequeue(self):
        """Remove and return the front element of the queue.

        Returns:
            The front element of the queue.

        Raises:
            IndexError: If the queue is empty (queue underflow).
        """
        if self.is_empty():
            raise IndexError("Queue underflow: Cannot dequeue from an empty queue.")
        return self.queue.pop(0)

    def front(self):
        """Return the front element of the queue without removing it.

        Returns:
            The front element of the queue.

        Raises:
            IndexError: If the queue is empty (queue underflow).
        """
        if self.is_empty():
            raise IndexError("Queue underflow: Cannot access front of an empty queue.")
        return self.queue[0]

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

        Returns:
            True if the queue is empty, False otherwise.
        """
        return len(self.queue) == 0
def main():
    queue = Queue()
    while True:
        print("\nQueue Operations:")
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. Front")
        print("4. Check if Empty")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            element = input("Enter the element to enqueue: ")
            queue.enqueue(element)
            print(f"'{element}' has been enqueued to the queue.")
        elif choice == '2':
            try:
                dequeued_element = queue.dequeue()
                print(f"'{dequeued_element}' has been dequeued from the queue.")
            except IndexError as e:
                print(e)
        elif choice == '3':
            try:
                front_element = queue.front()
                print(f"The front element is: '{front_element}'")
            except IndexError as e:
                print(e)
        elif choice == '4':
            if queue.is_empty():
                print("The queue is empty.")
            else:
                print("The queue is not empty.")
        elif choice == '5':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 5.")
if __name__ == "__main__":
    main()
```

Expected Output:



➢ A fully functional Queue implementation in Python.

➢ Proper error handling and documentation.

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.

```python
def bubble_sort(arr):
    """Sort an array using the Bubble Sort algorithm.

    Args:
        arr: A list of elements to be sorted.

    Returns:
        A sorted list of elements.

    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) (in-place sorting).
    """
    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):
    """Sort an array using the Selection Sort algorithm.

    Args:
        arr: A list of elements to be sorted.

    Returns:
        A sorted list of elements.

    Time Complexity: O(n^2) in all cases (best, average, and worst).
    Space Complexity: O(1) (in-place sorting).
    """
    n = len(arr)
    # Traverse through all elements in the array
    for i in range(n):
        # Find the minimum element in the remaining unsorted array
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap the found minimum element with the first element of the unsorted array
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr
def main():
    arr = input("Enter a list of elements to sort (comma separated): ").split(',')
    print("Original array:", arr)

    sorted_arr_bubble = bubble_sort(arr.copy())
    print("Sorted array using Bubble Sort:", sorted_arr_bubble)

    sorted_arr_selection = selection_sort(arr.copy())
    print("Sorted array using Selection Sort:", sorted_arr_selection)
if __name__ == "__main__":
    main()
```

Expected Output:

• Correct Python implementations of both sorting algorithms.

• Complexity analysis in docstrings.

```
Enter a list of elements to sort (comma separated): 5,2,9,1,5,6
Original array: ['5', '2', '9', '1', '5', '6']
Sorted array using Bubble Sort: ['1', '2', '5', '5', '6', '9']
Sorted array using Selection Sort: ['1', '2', '5', '5', '6', '9']
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code
-12.py"
Enter a list of elements to sort (comma separated): 1,2,3,4,5
Original array: ['1', '2', '3', '4', '5']
Sorted array using Bubble Sort: ['1', '2', '3', '4', '5']
Sorted array using Selection Sort: ['1', '2', '3', '4', '5']
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code
-12.py"
Enter a list of elements to sort (comma separated): [10]
Original array: ['[10]']
Sorted array using Bubble Sort: ['[10]']
Sorted array using Selection Sort: ['[10]']
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code
-12.py"
Enter a list of elements to sort (comma separated): 4,2,2,8,3,3,1
Original array: ['4', '2', '2', '8', '3', '3', '1']
Sorted array using Bubble Sort: ['1', '2', '2', '3', '3', '4', '8']
Sorted array using Selection Sort: ['1', '2', '2', '3', '3', '4', '8']
PS C:\Users\AdepuTejaswini\AI assist>
```