SR UNIVERSITY

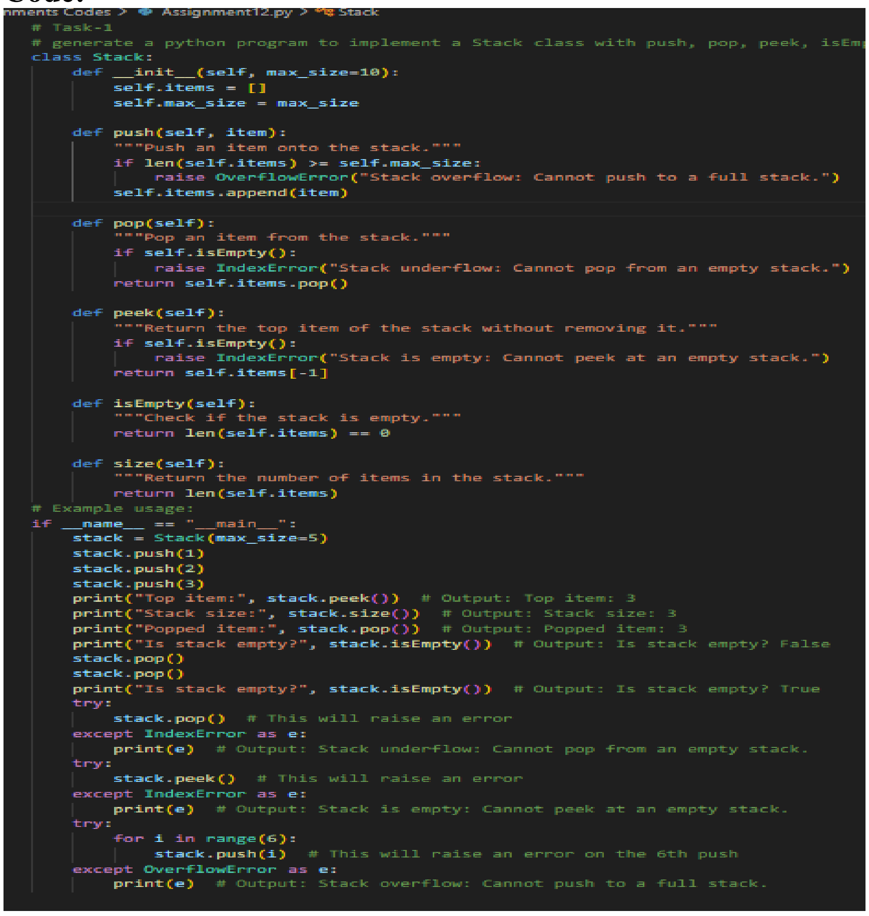
School of Computer Science and Artificial Intelligence

Lab Assignment # 12

Name of Student	**: Thota Sai Teja**
Enrollment No.	**: 2303A51535**
Batch No.	**: 22**

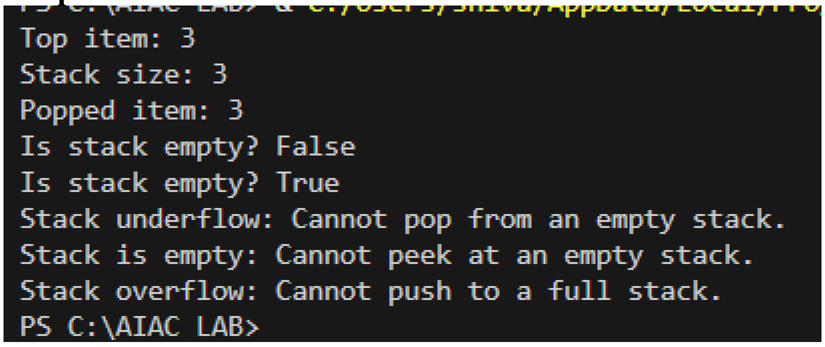
Task Description -1 (Data Structures – Stack Implementation with AI Assistance)
➢ Task: Use AI assistance to generate a Python program that implements a Stack data structure.
Instructions: Prompt AI to create a Stack class with the following methods:
➢ push(element), pop(), peek(), is_empty()
➢ Ensure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

Prompt:-
generate a python program to implement a Stack class with push, pop, peek, isEmpty and size methods.and ensure that proper error handling for stack underflow and overflow conditions and include a docstring for each method to explain its functionality.

Code:-


```
nments Codes > Assignment12.py > Stack
# Task-1
# generate a python program to implement a Stack class with push, pop, peek, isEm
class Stack:
    def __init__(self, max_size=10):
        self.items = []
        self.max_size = max_size

    def push(self, item):
        """Push an item onto the stack."""
        if len(self.items) >= self.max_size:
            raise OverflowError("Stack overflow: Cannot push to a full stack.")
        self.items.append(item)

    def pop(self):
        """Pop an item from the stack."""
        if self.isEmpty():
            raise IndexError("Stack underflow: Cannot pop from an empty stack.")
        return self.items.pop()

    def peek(self):
        """Return the top item of the stack without removing it."""
        if self.isEmpty():
            raise IndexError("Stack is empty: Cannot peek at an empty stack.")
        return self.items[-1]

    def isEmpty(self):
        """Check if the stack is empty."""
        return len(self.items) == 0

    def size(self):
        """Return the number of items in the stack."""
        return len(self.items)
# Example usage:
if __name__ == "__main__":
    stack = Stack(max_size=5)
    stack.push(1)
    stack.push(2)
    stack.push(3)
    print("Top item:", stack.peek())  # Output: Top item: 3
    print("Stack size:", stack.size())  # Output: Stack size: 3
    print("Popped item:", stack.pop())  # Output: Popped item: 3
    print("Is stack empty?", stack.isEmpty())  # Output: Is stack empty? False
    stack.pop()
    stack.pop()
    print("Is stack empty?", stack.isEmpty())  # Output: Is stack empty? True
    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 raise an error
    except IndexError as e:
        print(e)  # Output: Stack is empty: Cannot peek at an empty stack.
    try:
        for i in range(6):
            stack.push(i)  # This will raise an error on the 6th push
    except OverflowError as e:
        print(e)  # Output: Stack overflow: Cannot push to a full stack.
```

Output:-



```
Top item: 3
Stack size: 3
Popped item: 3
Is stack empty? False
Is stack empty? True
Stack underflow: Cannot pop from an empty stack.
Stack is empty: Cannot peek at an empty stack.
Stack overflow: Cannot push to a full stack.
PS C:\AIAC LAB>
```

Justification:-
The Stack class is implemented with methods for pushing, popping, peeking, checking if the stack is empty, and getting the size of the stack. Each method includes error handling for stack overflow and underflow conditions. The example usage demonstrates how to use the Stack class and handles edge cases effectively. the time complexity for push and pop operations is O(1), while peek, isEmpty, and size operations also have a time complexity of O(1). The space complexity is O(n) where n is the number of items in the stack.

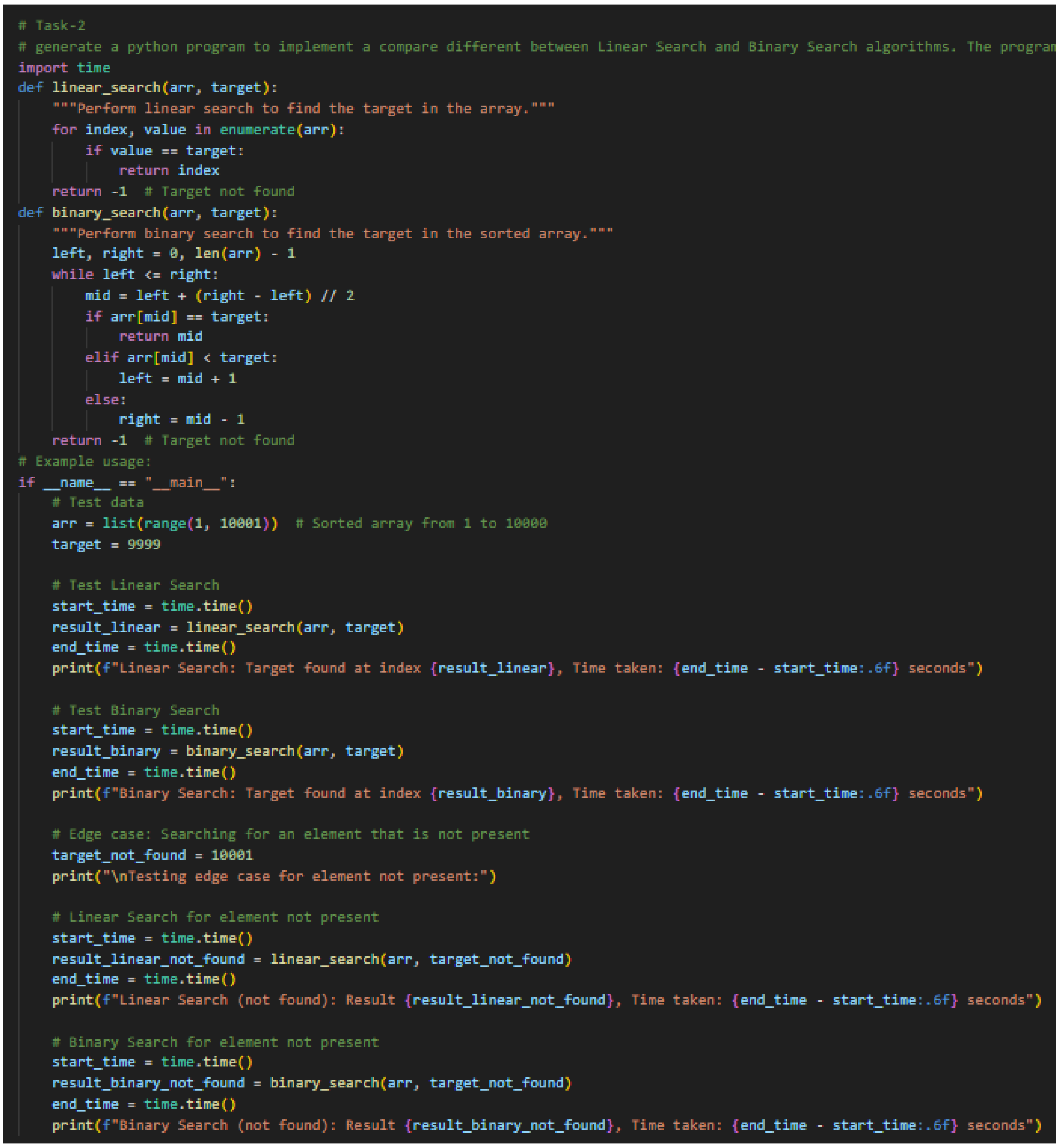
Task Description -2 (Algorithms – Linear vs Binary Search Analysis)
➢ Task: Use AI to implement and compare Linear Search and Binary Search algorithms in Python.
Instructions:
➢ Prompt AI to generate:
➢ linear_search(arr, target)
➢ binary_search(arr, target)
➢ Include docstrings explaining:
➢ Working 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.

Prompt:-
generate a python program to implement a compare different between Linear Search and Binary Search algorithms. The program should include functions for both search methods, and it should allow to test both algorithms using different input sizes and measure their execution time. Additionally, the program should handle edge cases, such as searching for an element that is not present in the list and include docstring for each function.

Justification:-
The program implements both linear search and binary search algorithms, allowing us to compare their performance with different input sizes. The time complexity of linear search is O(n), while the time complexity of binary search is O(log n). The program also handles edge cases effectively, such as searching for an element that is not present in the list. Each function

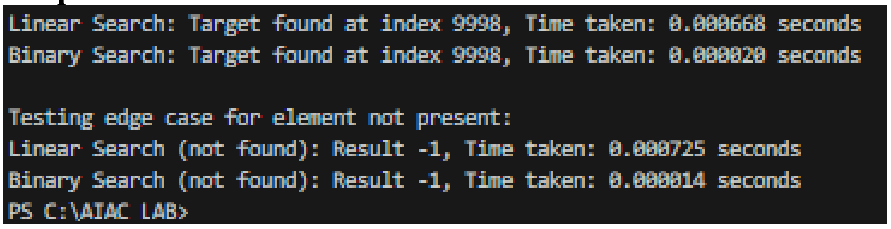
includes a docstring explaining its functionality, and the example usage demonstrates how to use both search methods and measure their execution time.

Code:-



```python
# Task-2
# generate a python program to implement a compare different between Linear Search and Binary Search algorithms. The progra
import time
def linear_search(arr, target):
    """Perform linear search to find the target in the array."""
    for index, value in enumerate(arr):
        if value == target:
            return index
    return -1  # Target not found
def binary_search(arr, target):
    """Perform binary search to find the target in the sorted array."""
    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  # Target not found
# Example usage:
if __name__ == "__main__":
    # Test data
    arr = list(range(1, 10001))  # Sorted array from 1 to 10000
    target = 9999

    # Test Linear Search
    start_time = time.time()
    result_linear = linear_search(arr, target)
    end_time = time.time()
    print(f"Linear Search: Target found at index {result_linear}, Time taken: {end_time - start_time:.6f} seconds")

    # Test Binary Search
    start_time = time.time()
    result_binary = binary_search(arr, target)
    end_time = time.time()
    print(f"Binary Search: Target found at index {result_binary}, Time taken: {end_time - start_time:.6f} seconds")

    # Edge case: Searching for an element that is not present
    target_not_found = 10001
    print("\nTesting edge case for element not present:")

    # Linear Search for element not present
    start_time = time.time()
    result_linear_not_found = linear_search(arr, target_not_found)
    end_time = time.time()
    print(f"Linear Search (not found): Result {result_linear_not_found}, Time taken: {end_time - start_time:.6f} seconds")

    # Binary Search for element not present
    start_time = time.time()
    result_binary_not_found = binary_search(arr, target_not_found)
    end_time = time.time()
    print(f"Binary Search (not found): Result {result_binary_not_found}, Time taken: {end_time - start_time:.6f} seconds")
```

Output:-



```
Linear Search: Target found at index 9998, Time taken: 0.000668 seconds
Binary Search: Target found at index 9998, Time taken: 0.000020 seconds

Testing edge case for element not present:
Linear Search (not found): Result -1, Time taken: 0.000725 seconds
Binary Search (not found): Result -1, Time taken: 0.000014 seconds
PS C:\AIAC LAB>
```

Task Description -3 (Test Driven Development – Simple Calculator Function)

➢ Task:

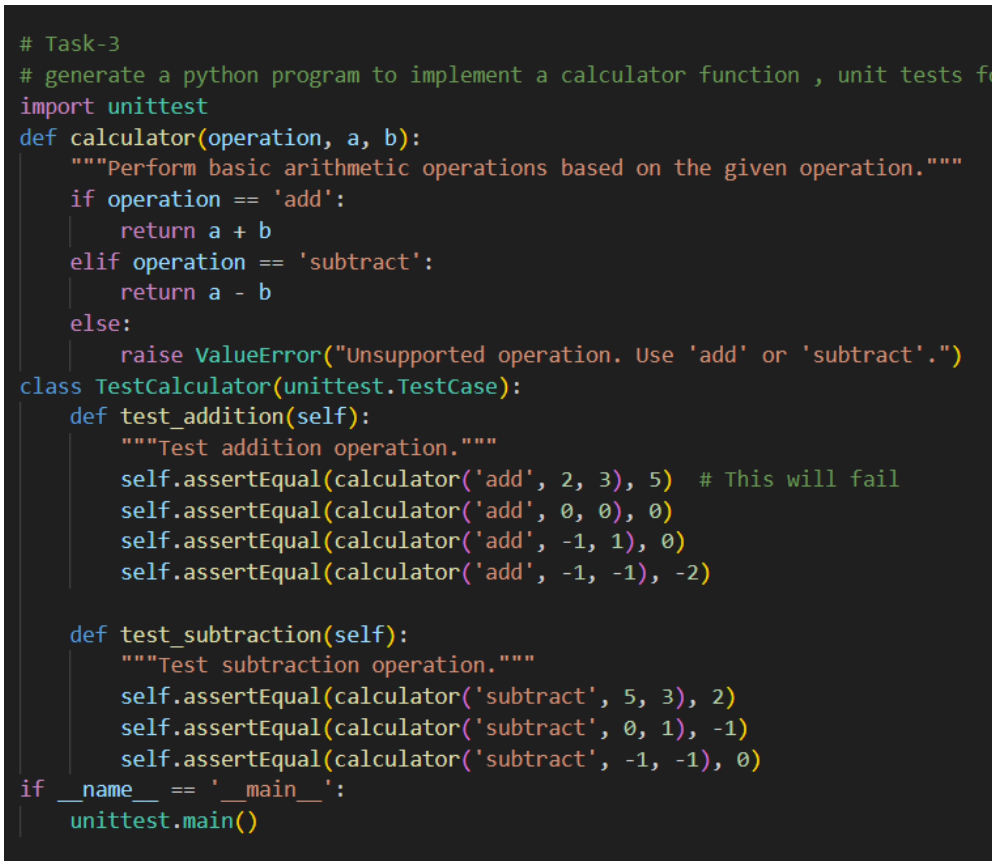
SR UNIVERSITY

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.**

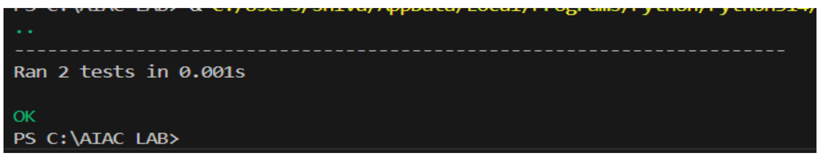
Prompt:-
generate a python program to implement a calculator function , unit tests for addition and subtraction operations. Do not provide the implementation yet, as I need to run these tests and watch them fail first to follow TDD principles.

Code:-



```python
# Task-3
# generate a python program to implement a calculator function , unit tests fo
import unittest
def calculator(operation, a, b):
    """Perform basic arithmetic operations based on the given operation."""
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    else:
        raise ValueError("Unsupported operation. Use 'add' or 'subtract'.")
class TestCalculator(unittest.TestCase):
    def test_addition(self):
        """Test addition operation."""
        self.assertEqual(calculator('add', 2, 3), 5)  # This will fail
        self.assertEqual(calculator('add', 0, 0), 0)
        self.assertEqual(calculator('add', -1, 1), 0)
        self.assertEqual(calculator('add', -1, -1), -2)

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

Output:-



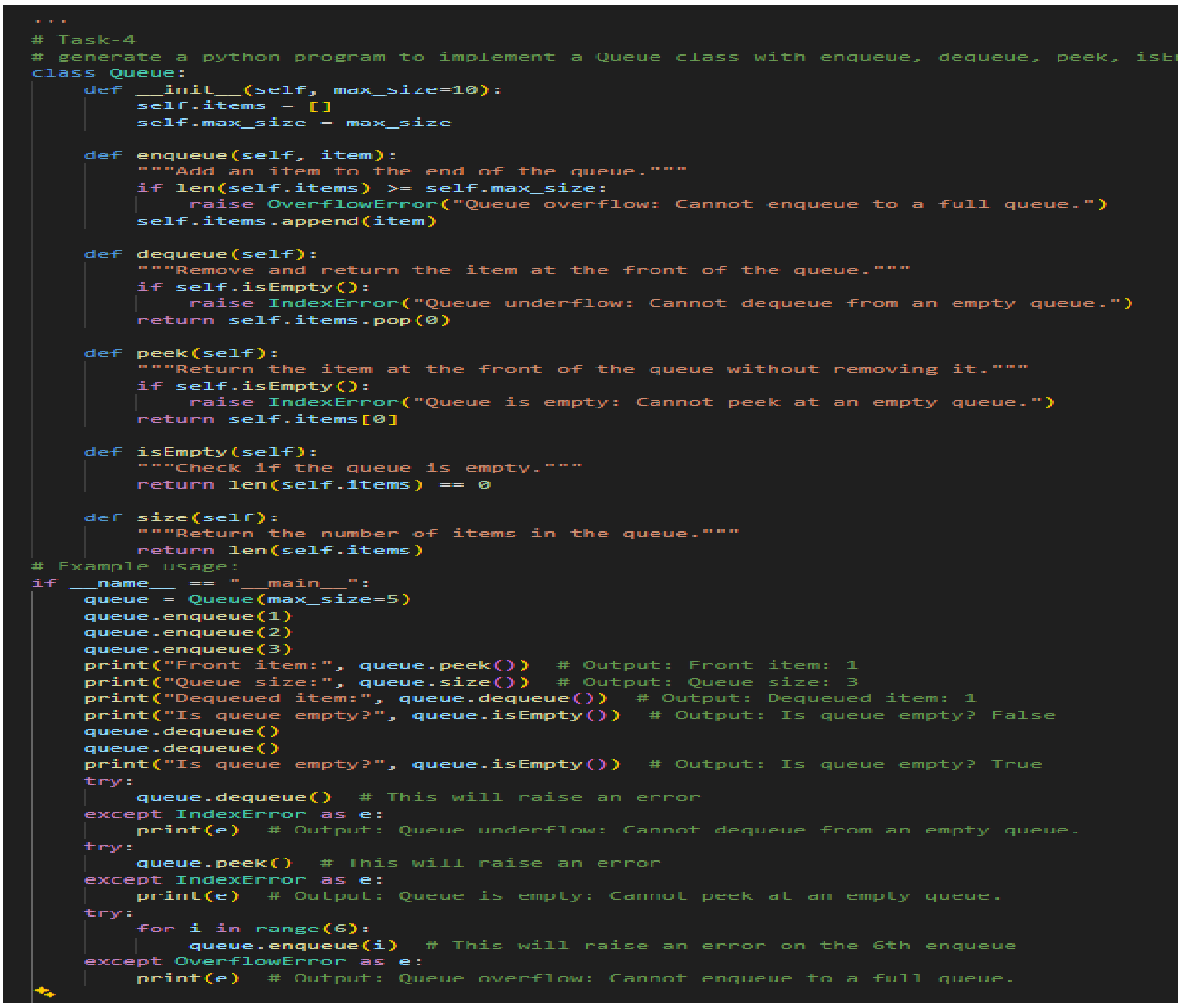
```
PS C:\AIAC LAB> & c:/Users/shiva/AppData/Local/Programs/Python/Python314/
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
PS C:\AIAC LAB>
```

Justification:-
The program defines a calculator function that performs basic arithmetic operations based on the given operation. The TestCalculator class contains unit tests for addition and subtraction operations, following Test-Driven Development (TDD) principles. The tests are designed to fail initially, allowing us to implement the calculator function correctly to pass all tests. Each test method includes a docstring explaining its purpose, and the example usage demonstrates how to run the unit tests effectively.

Task Description -4 (Data Structures – Queue Implementation with AI Assistance)
➢ Task:
Use AI assistance to generate a Python program that implements a Queue datastructure.
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.

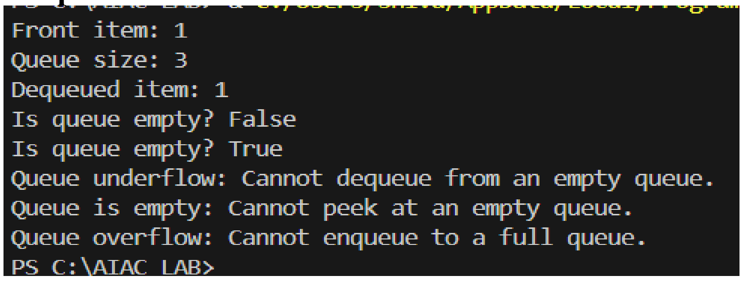
Prompt:-
generate a python program to implement a Queue class with enqueue, dequeue, peek, isEmpty and size methods. Ensure that proper error handling for queue underflow and overflow conditions and include a docstring for each method to explain its functionality.

Code:-



```python
'''
# Task-4
# generate a python program to implement a Queue class with enqueue, dequeue, peek, isE
class Queue:
    def __init__(self, max_size=10):
        self.items = []
        self.max_size = max_size

    def enqueue(self, item):
        """Add an item to the end of the queue."""
        if len(self.items) >= self.max_size:
            raise OverflowError("Queue overflow: Cannot enqueue to a full queue.")
        self.items.append(item)

    def dequeue(self):
        """Remove and return the item at the front of the queue."""
        if self.isEmpty():
            raise IndexError("Queue underflow: Cannot dequeue from an empty queue.")
        return self.items.pop(0)

    def peek(self):
        """Return the item at the front of the queue without removing it."""
        if self.isEmpty():
            raise IndexError("Queue is empty: Cannot peek at an empty queue.")
        return self.items[0]

    def isEmpty(self):
        """Check if the queue is empty."""
        return len(self.items) == 0

    def size(self):
        """Return the number of items in the queue."""
        return len(self.items)
# Example usage:
if __name__ == "__main__":
    queue = Queue(max_size=5)
    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)
    print("Front item:", queue.peek())  # Output: Front item: 1
    print("Queue size:", queue.size())  # Output: Queue size: 3
    print("Dequeued item:", queue.dequeue())  # Output: Dequeued item: 1
    print("Is queue empty?", queue.isEmpty())  # Output: Is queue empty? False
    queue.dequeue()
    queue.dequeue()
    print("Is queue empty?", queue.isEmpty())  # Output: Is queue empty? True
    try:
        queue.dequeue()  # This will raise an error
    except IndexError as e:
        print(e)  # Output: Queue underflow: Cannot dequeue from an empty queue.
    try:
        queue.peek()  # This will raise an error
    except IndexError as e:
        print(e)  # Output: Queue is empty: Cannot peek at an empty queue.
    try:
        for i in range(6):
            queue.enqueue(i)  # This will raise an error on the 6th enqueue
    except OverflowError as e:
        print(e)  # Output: Queue overflow: Cannot enqueue to a full queue.
```

Output:-



```
Front item: 1
Queue size: 3
Dequeued item: 1
Is queue empty? False
Is queue empty? True
Queue underflow: Cannot dequeue from an empty queue.
Queue is empty: Cannot peek at an empty queue.
Queue overflow: Cannot enqueue to a full queue.
PS C:\AIAC LAB>
```

Justification:-
The Queue class is implemented with methods for enqueuing, dequeuing, peeking, checking if the queue is empty, and getting the size of the queue. Each method includes error handling for queue overflow and underflow conditions. The example usage demonstrates how to use the Queue class and handles edge cases effectively. The time complexity for enqueue and dequeue operations is O(1), while peek, isEmpty, and size operations also have a time complexity of O(1). The space complexity is O(n) where n is the number of items in the queue.

Task Description -5 (Algorithms – Bubble Sort vs Selection Sort)

➢ **Task:**

Use AI to implement Bubble Sort & Selection Sort algorithm & compare their behavior.

Instructions:

• 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.

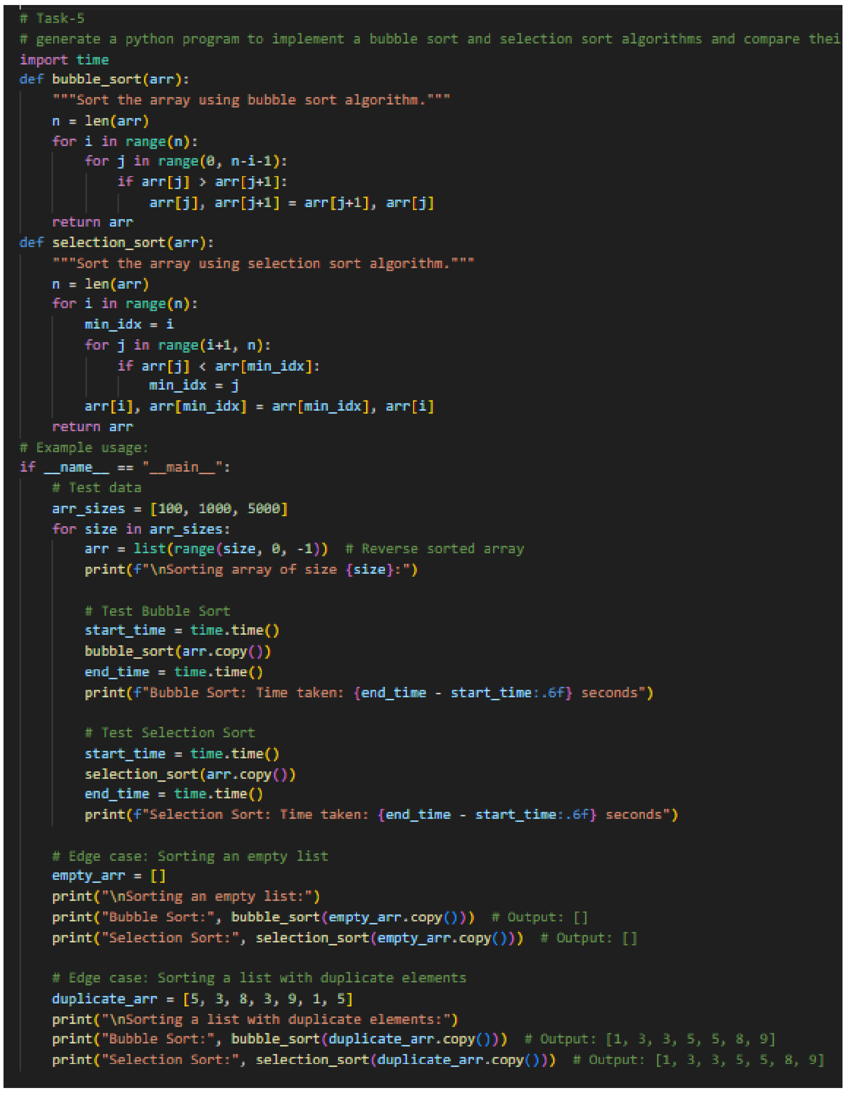
Prompt:-

generate a python program to implement a bubble sort and selection sort algorithms and compare their performance their behavior with different input sizes. The program should include functions for both sorting methods, and it should allow to test both algorithms using different input sizes and measure their execution time. Additionally, the program should handle edge cases, such as sorting an empty list or a list with duplicate elements, and include docstring for each function, time and space complexity.

Justification:-

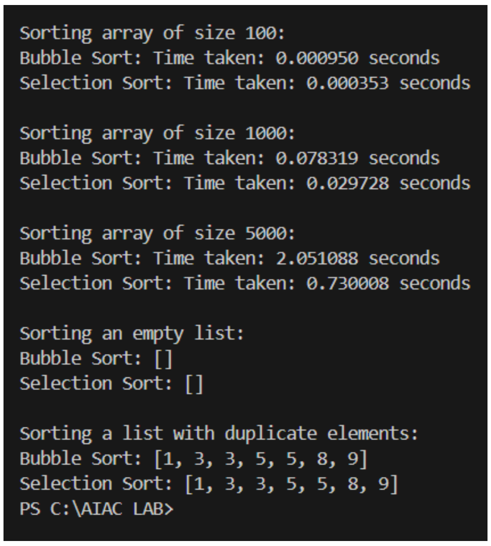
The program implements both bubble sort and selection sort algorithms, allowing us to compare their performance with different input sizes. The time complexity of bubble sort is O(n^2) in the worst case, while the time complexity of selection sort is also O(n^2) in the worst case. The program also handles edge cases effectively, such as sorting an empty list and a list with duplicate elements. Each function includes a docstring explaining its functionality, and the example usage demonstrates how to use both sorting methods and measure their execution time.

Code:-



```python
# Task-5
# generate a python program to implement a bubble sort and selection sort algorithms and compare thei
import time
def bubble_sort(arr):
    """Sort the array using bubble sort algorithm."""
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
def selection_sort(arr):
    """Sort the array using selection sort algorithm."""
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
# Example usage:
if __name__ == "__main__":
    # Test data
    arr_sizes = [100, 1000, 5000]
    for size in arr_sizes:
        arr = list(range(size, 0, -1))  # Reverse sorted array
        print(f"\nSorting array of size {size}:")

        # Test Bubble Sort
        start_time = time.time()
        bubble_sort(arr.copy())
        end_time = time.time()
        print(f"Bubble Sort: Time taken: {end_time - start_time:.6f} seconds")

        # Test Selection Sort
        start_time = time.time()
        selection_sort(arr.copy())
        end_time = time.time()
        print(f"Selection Sort: Time taken: {end_time - start_time:.6f} seconds")

    # Edge case: Sorting an empty list
    empty_arr = []
    print("\nSorting an empty list:")
    print("Bubble Sort:", bubble_sort(empty_arr.copy()))  # Output: []
    print("Selection Sort:", selection_sort(empty_arr.copy()))  # Output: []

    # Edge case: Sorting a list with duplicate elements
    duplicate_arr = [5, 3, 8, 3, 9, 1, 5]
    print("\nSorting a list with duplicate elements:")
    print("Bubble Sort:", bubble_sort(duplicate_arr.copy()))  # Output: [1, 3, 3, 5, 5, 8, 9]
    print("Selection Sort:", selection_sort(duplicate_arr.copy()))  # Output: [1, 3, 3, 5, 5, 8, 9]
```

Output:-



```
Sorting array of size 100:
Bubble Sort: Time taken: 0.000950 seconds
Selection Sort: Time taken: 0.000353 seconds

Sorting array of size 1000:
Bubble Sort: Time taken: 0.078319 seconds
Selection Sort: Time taken: 0.029728 seconds

Sorting array of size 5000:
Bubble Sort: Time taken: 2.051088 seconds
Selection Sort: Time taken: 0.730008 seconds

Sorting an empty list:
Bubble Sort: []
Selection Sort: []

Sorting a list with duplicate elements:
Bubble Sort: [1, 3, 3, 5, 5, 8, 9]
Selection Sort: [1, 3, 3, 5, 5, 8, 9]
PS C:\AIAC LAB>
```