Week 8 Assignment

Name: Drishti Durgesh Telgu
Student ID: SM20240093
Unit: ICT_102
Professor: Dr. Duy Nguyen

Tutorial:

Part 1: Understanding Recursive Functions

Create a Python script named recursive_functions.py that includes the following recursive functions:

factorial: Computes the factorial of a number.

fibonacci: Computes the nth Fibonacci number.

Each function should include detailed docstrings

```
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


def fibonacci(n):
    if n < 0:
            raise ValueError("Fibonacci number is not defined for negative
numbers.")
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

```
if __name__ == "__main__":
    print("Factorial Tests:")
    print(f"Factorial of 15: {factorial(15)}")
    print(f"Factorial of 0: {factorial(0)}")
    print("\nFibonacci Tests:")
    print(f"Fibonacci of 5: {fibonacci(5)}")
    print(f"Fibonacci of 10: {fibonacci(10)}")
```

Output:

C:\Users\61411\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\61411\PycharmProjects\pythonProject\.venv\recursive_function.py

Factorial Tests:

Factorial of 15: 1307674368000

Factorial of 0: 1

Fibonacci Tests:

Fibonacci of 5: 5

Fibonacci of 10: 55

Process finished with exit code 0

Part 2: Understanding Stack and Queue

Create a Python script named data_structures.py that includes the implementation of a stack and a queue using lists. Include methods for typical stack and queue operations (push, pop, enqueue, dequeue, peek, is_empty).

```
class Stack:
    def __init__(self):

        self.items = []
```

```
    def push(self, item):

        self.items.append(item)

    def pop(self):

        if self.is_empty():
            raise IndexError("Pop from an empty stack.")
        return self.items.pop()

    def peek(self):

        if self.is_empty():
            raise IndexError("Peek from an empty stack.")
        return self.items[-1]

    def is_empty(self):

        return len(self.items) == 0


class Queue:
    def __init__(self):

        self.items = []

    def enqueue(self, item):

        self.items.append(item)

    def dequeue(self):

        if self.is_empty():
            raise IndexError("Dequeue from an empty queue.")
        return self.items.pop(0)

    def peek(self):

        if self.is_empty():
```

```
            raise IndexError("Peek from an empty queue.")
        return self.items[0]

    def is_empty(self):

        return len(self.items) == 0

if __name__ == "__main__":

    print("Stack Operations:")
    stack = Stack()
    print("Is stack empty?", stack.is_empty())
    stack.push(10)
    stack.push(20)
    stack.push(30)
    print("Peek top item:", stack.peek())
    print("Pop item:", stack.pop())
    print("Peek top item after pop:", stack.peek())
    print("Is stack empty after operations?", stack.is_empty())

    print("\nQueue Operations:")
    queue = Queue()
    print("Is queue empty?", queue.is_empty())
    queue.enqueue(100)
    queue.enqueue(200)
    queue.enqueue(300)
    print("Peek front item:", queue.peek())
    print("Dequeue item:", queue.dequeue())
    print("Peek front item after dequeue:", queue.peek())
    print("Is queue empty after operations?", queue.is_empty())
```

Output :

C:\Users\61411\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\61411\PycharmProjects\pythonProject\.venv\data_structures.py

Stack Operations:

Is stack empty? True

Peek top item: 30

Pop item: 30

Peek top item after pop: 20

Is stack empty after operations? False

Queue Operations:

Is queue empty? True

Peek front item: 100

Dequeue item: 100

Peek front item after dequeue: 200

Is queue empty after operations? False

Process finished with exit code 0

Part 3: Understanding Binary Search Using Recursive Function

Create a Python script named binary_search.py that includes a recursive implementation of the binary search algorithm.

```
def binary_search(arr, target, low, high):
    if low > high:
        return -1

    mid = (low + high) // 2

    if arr[mid] == target:
        return mid
    elif arr[mid] < target:
        return binary_search(arr, target, mid + 1, high)
    else:
        return binary_search(arr, target, low, mid - 1)
if __name__ == "__main__":
    test_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
    targets = [3, 7, 1, 10, 11]

    for target in targets:
        result = binary_search(test_array, target, 0, len(test_array) - 1)
        if result != -1:
            print(f"Target {target} found at index {result}.")
        else:
            print(f"Target {target} not found in the array.")
```

Output:

C:\Users\61411\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\61411\PycharmProjects\pythonProject\.venv\binary_search.py

Target 3 found at index 2.

Target 7 found at index 6.

Target 1 found at index 0.

Target 10 found at index 9.

Target 11 not found in the array.

Process finished with exit code 0

Part 4: Writing Recursive Functions to Solve Real-World Problems

Create a Python script named real_world_recursion.py that includes the following recursive functions to solve real-world problems:

sum_nested_list: Computes the sum of all integers in a nested list.

flatten_list: Flattens a nested list.

```
def sum_nested_list(nested_list):

    total = 0
    for item in nested_list:
        if isinstance(item, list):
```

```
            total += sum_nested_list(item)
        else:
            total += item
    return total


def flatten_list(nested_list):
    flat_list = []
    for item in nested_list:
        if isinstance(item, list):
            flat_list.extend(flatten_list(item))
        else:
            flat_list.append(item)
    return flat_list



if __name__ == "__main__":
    nested_list_1 = [1, [2, [3, 4]], 5]
            print("Sum    of    nested    list    [1,    [2,    [3,    4]],    5]:",
sum_nested_list(nested_list_1))
    nested_list_2 = [1, [2, [3, 4]], 5]
    print("Flattened list [1, [2, [3, 4]], 5]:", flatten_list(nested_list_2))
    nested_list_3 = [[1, 2], [3, [4, [5]]], 6]
          print("Sum   of   nested   list   [[1,   2],   [3,   [4,   [5]]],   6]:",
sum_nested_list(nested_list_3))
    print("Flattened list [[1, 2], [3, [4, [5]]], 6]:",
          flatten_list(nested_list_3))
```

Output:

C:\Users\61411\PycharmProjects\pythonProject\.venv\Scripts\python.exe

C:\Users\61411\PycharmProjects\pythonProject\.venv\real_world_recursion.py

Sum of nested list [1, [2, [3, 4]], 5]: 15

Flattened list [1, [2, [3, 4]], 5]: [1, 2, 3, 4, 5]

Sum of nested list [[1, 2], [3, [4, [5]]], 6]: 21

Flattened list [[1, 2], [3, [4, [5]]], 6]: [1, 2, 3, 4, 5, 6]

Process finished with exit code 0