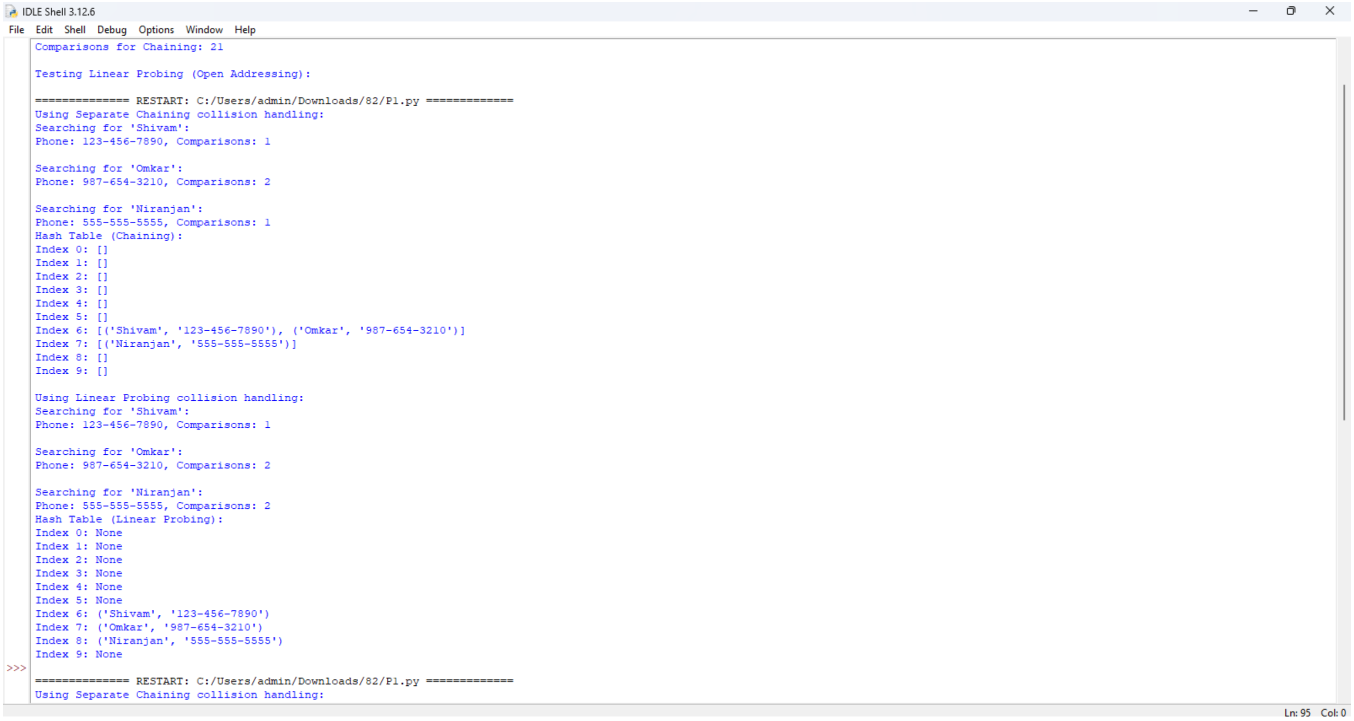
Assignment No.1

```
class HashTableChaining:
  def __init__(self, size):
    self.size = size
    self.table = [[] for _ in range(size)]  # List of lists for separate chaining

  def _hash(self, key):
    return sum(ord(c) for c in key) % self.size  # Simple hash function

  def insert(self, key, value):
    index = self._hash(key)
    for i, (k, v) in enumerate(self.table[index]):
      if k == key:
        self.table[index][i] = (key, value)
        return
    self.table[index].append((key, value))

  def search(self, key):
    index = self._hash(key)
    comparisons = 0
    for k, v in self.table[index]:
      comparisons += 1
      if k == key:
        return v, comparisons
    return None, comparisons

  def print_table(self):
    print("Hash Table (Chaining):")
    for index, bucket in enumerate(self.table):
            print(f"Index {index}: {bucket}")



class HashTableLinearProbing:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size  # Initialize with None (empty slots)

    def _hash(self, key):
        return sum(ord(c) for c in key) % self.size  # Simple hash function

    def insert(self, key, value):
        index = self._hash(key)
        original_index = index
        while self.table[index] is not None:
            if self.table[index][0] == key:
                self.table[index] = (key, value)
                return
            index = (index + 1) % self.size
            if index == original_index:
                raise Exception("Hash table is full")
        self.table[index] = (key, value)

    def search(self, key):
        index = self._hash(key)
        original_index = index
        comparisons = 0
        while self.table[index] is not None:
            comparisons += 1
            if self.table[index][0] == key:
                return self.table[index][1], comparisons
            index = (index + 1) % self.size
            if index == original_index:
                break
        return None, comparisons

    def print_table(self):
        print("Hash Table (Linear Probing):")
        for index, item in enumerate(self.table):
            print(f"Index {index}: {item}")



# Main program
def run_program(collision_type='chaining'):
    if collision_type == 'chaining':
        hash_table = HashTableChaining(10)
    else:
        hash_table = HashTableLinearProbing(10)

    # Insert data into the hash table
    hash_table.insert("Shivam", "123-456-7890")
    hash_table.insert("Omkar", "987-654-3210")
    hash_table.insert("Niranjan", "555-555-5555")

    # Search for keys and print results
    print("Searching for 'Shivam':")
    phone, comparisons = hash_table.search("Shivam")
    print(f"Phone: {phone}, Comparisons: {comparisons}")

    print("\nSearching for 'Omkar':")
    phone, comparisons = hash_table.search("Omkar")
    print(f"Phone: {phone}, Comparisons: {comparisons}")
    print("\nSearching for 'Niranjan':")
    phone, comparisons = hash_table.search("Niranjan")
    print(f"Phone: {phone}, Comparisons: {comparisons}")

    # Print the table for inspection
    hash_table.print_table()


# Example usage
print("Using Separate Chaining collision handling:")
run_program('chaining')

print("\nUsing Linear Probing collision handling:")
run_program('linear')
```

Output :



```
Comparisons for Chaining: 21

Testing Linear Probing (Open Addressing):

============== RESTART: C:/Users/admin/Downloads/82/P1.py =============
Using Separate Chaining collision handling:
Searching for 'Shivam':
Phone: 123-456-7890, Comparisons: 1

Searching for 'Omkar':
Phone: 987-654-3210, Comparisons: 2

Searching for 'Niranjan':
Phone: 555-555-5555, Comparisons: 1
Hash Table (Chaining):
Index 0: []
Index 1: []
Index 2: []
Index 3: []
Index 4: []
Index 5: []
Index 6: [('Shivam', '123-456-7890'), ('Omkar', '987-654-3210')]
Index 7: [('Niranjan', '555-555-5555')]
Index 8: []
Index 9: []

Using Linear Probing collision handling:
Searching for 'Shivam':
Phone: 123-456-7890, Comparisons: 1

Searching for 'Omkar':
Phone: 987-654-3210, Comparisons: 2

Searching for 'Niranjan':
Phone: 555-555-5555, Comparisons: 2
Hash Table (Linear Probing):
Index 0: None
Index 1: None
Index 2: None
Index 3: None
Index 4: None
Index 5: None
Index 6: ('Shivam', '123-456-7890')
Index 7: ('Omkar', '987-654-3210')
Index 8: ('Niranjan', '555-555-5555')
Index 9: None
>>>
============== RESTART: C:/Users/admin/Downloads/82/P1.py =============
Using Separate Chaining collision handling:
```