Assignment 2

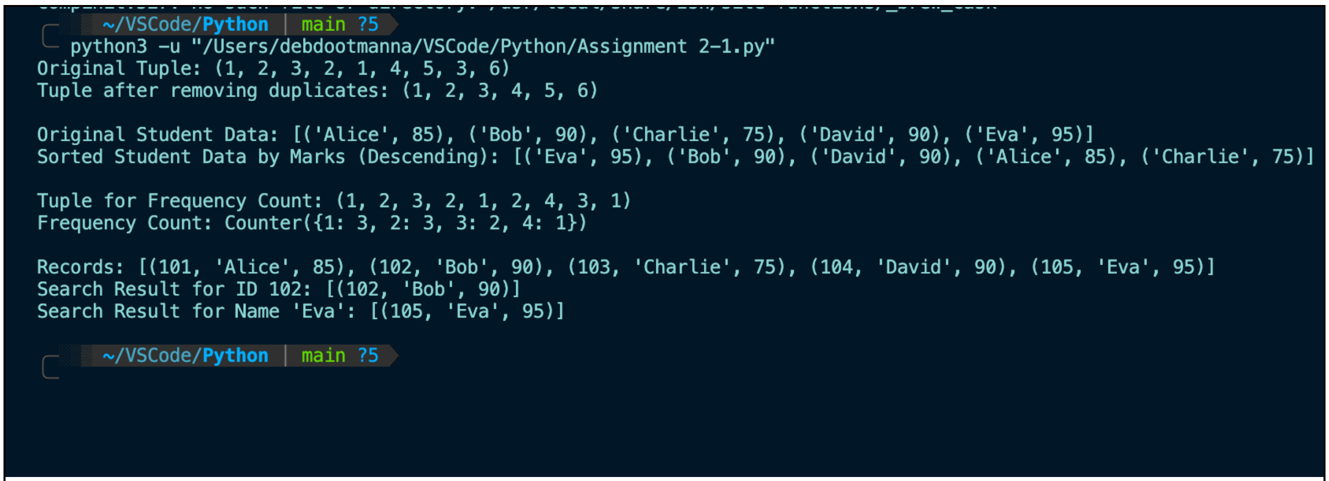
Use tuples in practical data handling scenarios.

Aim:
Write a function that takes a tuple as an argument and returns the tuple with all duplicates removed.
Create a list of tuples representing student names and marks, and sort the list by marks.
Write a program to count the frequency of elements in a tuple using Counter from the collections module.
Implement a tuple-based record system where each tuple represents a record (ID, Name, and Marks) and perform search operations

Code:

```python
from collections import Counter

# Function to remove duplicates from a tuple
def remove_duplicates(input_tuple):
    return tuple(set(input_tuple))

# Function to create a list of tuples representing students and sort by marks
def sort_students_by_marks(student_data):
    return sorted(student_data, key=lambda x: x[1], reverse=True)

# Function to count the frequency of elements in a tuple using Counter
def count_frequency(input_tuple):
    return Counter(input_tuple)

# Function to implement a tuple-based record system and search operations
def search_record(records, search_id=None, search_name=None):
    if search_id:
        result = [record for record in records if record[0] == search_id]
    elif search_name:
        result = [record for record in records if record[1].lower() == search_name.lower()]
    else:
        result = []
    return result

# Main program to demonstrate the functionality
if __name__ == "__main__":
    # 1. Removing duplicates from a tuple
    input_tuple = (1, 2, 3, 2, 1, 4, 5, 3, 6)
    print("Original Tuple:", input_tuple)
    print("Tuple after removing duplicates:",
remove_duplicates(input_tuple))

    # 2. List of tuples representing student names and marks
    students = [("Alice", 85), ("Bob", 90), ("Charlie", 75), ("David",
90), ("Eva", 95)]
    print("\nOriginal Student Data:", students)
    sorted_students = sort_students_by_marks(students)
    print("Sorted Student Data by Marks (Descending):",
sorted_students)

    # 3. Counting the frequency of elements in a tuple
    frequency_tuple = (1, 2, 3, 2, 1, 2, 4, 3, 1)
    print("\nTuple for Frequency Count:", frequency_tuple)
    print("Frequency Count:", count_frequency(frequency_tuple))

    # 4. Tuple-based record system
    records = [
        (101, "Alice", 85),
        (102, "Bob", 90),
        (103, "Charlie", 75),
        (104, "David", 90),
        (105, "Eva", 95)
    ]
    print("\nRecords:", records)
    # Search by ID
    search_id = 102
    print(f"Search Result for ID {search_id}:", search_record(records,
search_id=search_id))
    # Search by Name
    search_name = "Eva"
    print(f"Search Result for Name '{search_name}':",
search_record(records, search_name=search_name))
```

Output Screenshot:

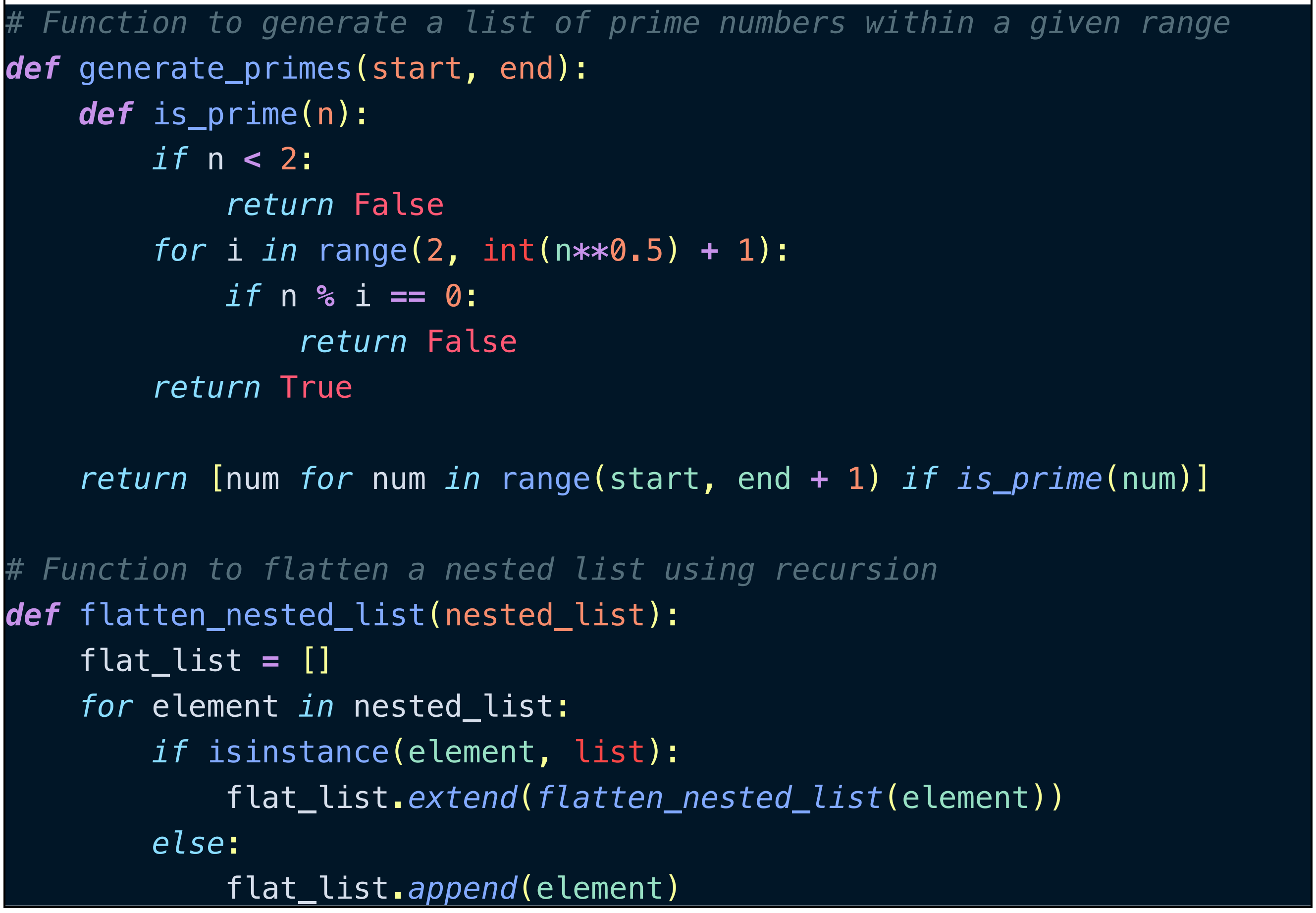


```
 ~/VSCode/Python | main ?5
 python3 -u "/Users/debdootmanna/VSCode/Python/Assignment 2-1.py"
Original Tuple: (1, 2, 3, 2, 1, 4, 5, 3, 6)
Tuple after removing duplicates: (1, 2, 3, 4, 5, 6)

Original Student Data: [('Alice', 85), ('Bob', 90), ('Charlie', 75), ('David', 90), ('Eva', 95)]
Sorted Student Data by Marks (Descending): [('Eva', 95), ('Bob', 90), ('David', 90), ('Alice', 85), ('Charlie', 75)]

Tuple for Frequency Count: (1, 2, 3, 2, 1, 2, 4, 3, 1)
Frequency Count: Counter({1: 3, 2: 3, 3: 2, 4: 1})

Records: [(101, 'Alice', 85), (102, 'Bob', 90), (103, 'Charlie', 75), (104, 'David', 90), (105, 'Eva', 95)]
Search Result for ID 102: [(102, 'Bob', 90)]
Search Result for Name 'Eva': [(105, 'Eva', 95)]

 ~/VSCode/Python | main ?5
```

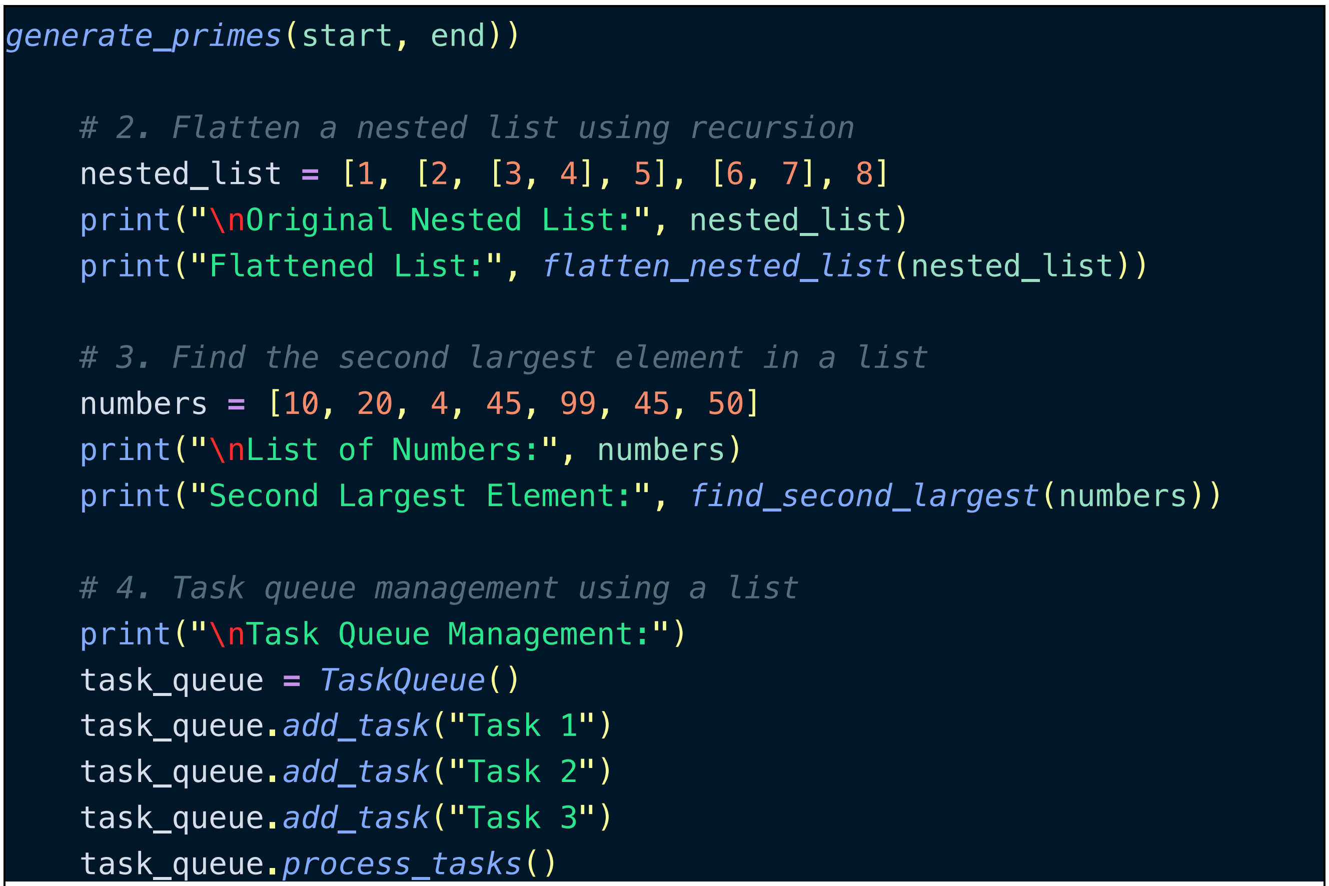
Apply list concepts to real-world scenarios.

Aim:
Implement a program to generate a list of prime numbers within a given range.
Flatten a nested list using recursion.
Write a program to find the second largest element from a list without using built-in functions.
Use a list to manage a task queue, where tasks are added, removed, and processed sequentially.

Code:

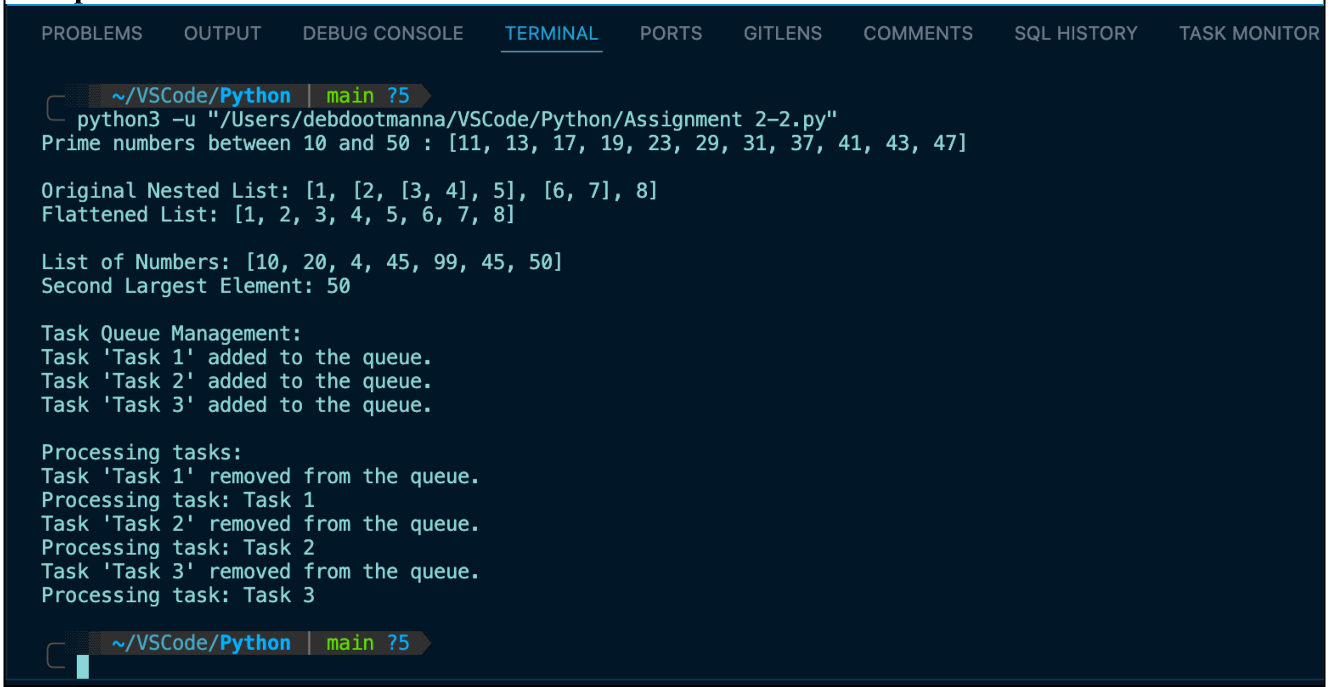


```python
# Function to generate a list of prime numbers within a given range
def generate_primes(start, end):
    def is_prime(n):
        if n < 2:
            return False
        for i in range(2, int(n**0.5) + 1):
            if n % i == 0:
                return False
        return True

    return [num for num in range(start, end + 1) if is_prime(num)]

# Function to flatten a nested list using recursion
def flatten_nested_list(nested_list):
    flat_list = []
    for element in nested_list:
        if isinstance(element, list):
            flat_list.extend(flatten_nested_list(element))
        else:
            flat_list.append(element)
    return flat_list

# Function to find the second largest element from a list without using
built-in functions
def find_second_largest(numbers):
    if len(numbers) < 2:
        return None  # Not enough elements for second largest
    largest = second_largest = float('-inf')
    for num in numbers:
        if num > largest:
            second_largest, largest = largest, num
        elif num > second_largest and num != largest:
            second_largest = num
    return second_largest if second_largest != float('-inf') else None

# Task queue management using a list
class TaskQueue:
    def __init__(self):
        self.queue = []

    def add_task(self, task):
        self.queue.append(task)
        print(f"Task '{task}' added to the queue.")

    def remove_task(self):
        if self.queue:
            task = self.queue.pop(0)
            print(f"Task '{task}' removed from the queue.")
            return task
        else:
            print("No tasks in the queue to remove.")
            return None

    def process_tasks(self):
        print("\nProcessing tasks:")
        while self.queue:
            task = self.remove_task()
            print(f"Processing task: {task}")

# Main program to demonstrate the functionality
if __name__ == "__main__":
    # 1. Generate a list of prime numbers within a range
    start, end = 10, 50
    print("Prime numbers between", start, "and", end, ":",
```



```python
generate_primes(start, end))

    # 2. Flatten a nested list using recursion
    nested_list = [1, [2, [3, 4], 5], [6, 7], 8]
    print("\nOriginal Nested List:", nested_list)
    print("Flattened List:", flatten_nested_list(nested_list))

    # 3. Find the second largest element in a list
    numbers = [10, 20, 4, 45, 99, 45, 50]
    print("\nList of Numbers:", numbers)
    print("Second Largest Element:", find_second_largest(numbers))

    # 4. Task queue management using a list
    print("\nTask Queue Management:")
    task_queue = TaskQueue()
    task_queue.add_task("Task 1")
    task_queue.add_task("Task 2")
    task_queue.add_task("Task 3")
    task_queue.process_tasks()
```

Output Screenshot:



```
  ~/VSCode/Python | main ?5
  python3 -u "/Users/debdootmanna/VSCode/Python/Assignment 2-2.py"
Prime numbers between 10 and 50 : [11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

Original Nested List: [1, [2, [3, 4], 5], [6, 7], 8]
Flattened List: [1, 2, 3, 4, 5, 6, 7, 8]

List of Numbers: [10, 20, 4, 45, 99, 45, 50]
Second Largest Element: 50

Task Queue Management:
Task 'Task 1' added to the queue.
Task 'Task 2' added to the queue.
Task 'Task 3' added to the queue.

Processing tasks:
Task 'Task 1' removed from the queue.
Processing task: Task 1
Task 'Task 2' removed from the queue.
Processing task: Task 2
Task 'Task 3' removed from the queue.
Processing task: Task 3

  ~/VSCode/Python | main ?5
```

Apply dictionary concepts to real-world scenarios.

Aim:

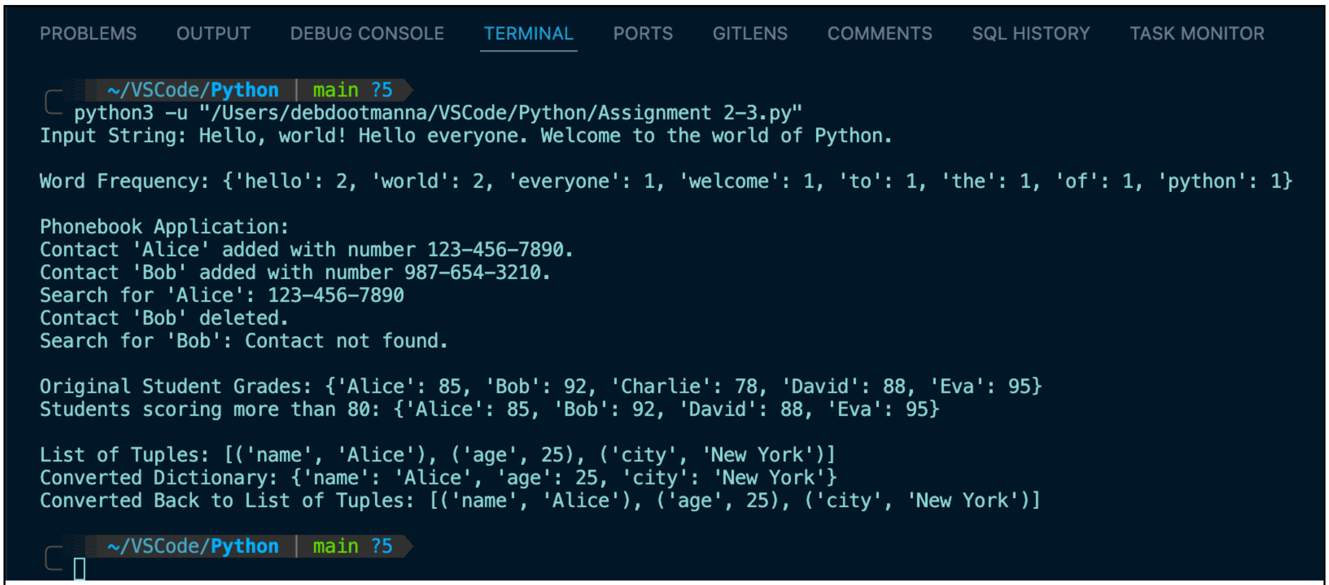
Write a program to count the frequency of each word in a string and store it in a dictionary.
Implement a simple phonebook application using a dictionary where users can add, delete, and search for contacts.
Create a dictionary of students and their grades. Write a program to filter students who scored more than a specific mark.
Write a program to convert a list of tuples (key-value pairs) into a dictionary and vice versa.

Code:

```python
# 1. Count the frequency of each word in a string and store it in a
dictionary
def count_word_frequency(input_string):
    words = input_string.split()
    frequency = {}
    for word in words:
        word = word.lower().strip(",.!?")  # Normalize words (lowercase
and remove punctuation)
        frequency[word] = frequency.get(word, 0) + 1
    return frequency

# 2. Simple phonebook application
class Phonebook:
    def __init__(self):
        self.contacts = {}

    def add_contact(self, name, number):
        self.contacts[name] = number
        print(f"Contact '{name}' added with number {number}.")

    def delete_contact(self, name):
        if name in self.contacts:
            del self.contacts[name]
            print(f"Contact '{name}' deleted.")
        else:
            print(f"Contact '{name}' not found.")

    def search_contact(self, name):
        return self.contacts.get(name, "Contact not found.")

# 3. Filter students who scored more than a specific mark
def filter_students_by_grade(students, min_marks):
    return {name: grade for name, grade in students.items() if grade >
min_marks}

# 4. Convert list of tuples (key-value pairs) into a dictionary and
vice versa
def list_to_dict(tuple_list):
    return dict(tuple_list)

def dict_to_list(dictionary):
    return list(dictionary.items())
```

```python
# Main program to demonstrate functionality
if __name__ == "__main__":
    # 1. Word frequency count
    input_string = "Hello, world! Hello everyone. Welcome to the world of Python."
    print("Input String:", input_string)
    word_freq = count_word_frequency(input_string)
    print("\nWord Frequency:", word_freq)

    # 2. Phonebook application
    print("\nPhonebook Application:")
    phonebook = Phonebook()
    phonebook.add_contact("Alice", "123-456-7890")
    phonebook.add_contact("Bob", "987-654-3210")
    print("Search for 'Alice':", phonebook.search_contact("Alice"))
    phonebook.delete_contact("Bob")
    print("Search for 'Bob':", phonebook.search_contact("Bob"))

    # 3. Filter students by grades
    students = {"Alice": 85, "Bob": 92, "Charlie": 78, "David": 88, "Eva": 95}
    min_marks = 80
    print("\nOriginal Student Grades:", students)
    filtered_students = filter_students_by_grade(students, min_marks)
    print(f"Students scoring more than {min_marks}:", filtered_students)

    # 4. Convert between dictionary and list of tuples
    tuple_list = [("name", "Alice"), ("age", 25), ("city", "New York")]
    print("\nList of Tuples:", tuple_list)
    converted_dict = list_to_dict(tuple_list)
    print("Converted Dictionary:", converted_dict)
    converted_list = dict_to_list(converted_dict)
    print("Converted Back to List of Tuples:", converted_list)
```

Output Screenshot:



```
~/VSCode/Python | main ?5
python3 -u "/Users/debdootmanna/VSCode/Python/Assignment 2-3.py"
Input String: Hello, world! Hello everyone. Welcome to the world of Python.

Word Frequency: {'hello': 2, 'world': 2, 'everyone': 1, 'welcome': 1, 'to': 1, 'the': 1, 'of': 1, 'python': 1}

Phonebook Application:
Contact 'Alice' added with number 123-456-7890.
Contact 'Bob' added with number 987-654-3210.
Search for 'Alice': 123-456-7890
Contact 'Bob' deleted.
Search for 'Bob': Contact not found.

Original Student Grades: {'Alice': 85, 'Bob': 92, 'Charlie': 78, 'David': 88, 'Eva': 95}
Students scoring more than 80: {'Alice': 85, 'Bob': 92, 'David': 88, 'Eva': 95}

List of Tuples: [('name', 'Alice'), ('age', 25), ('city', 'New York')]
Converted Dictionary: {'name': 'Alice', 'age': 25, 'city': 'New York'}
Converted Back to List of Tuples: [('name', 'Alice'), ('age', 25), ('city', 'New York')]

~/VSCode/Python | main ?5
```

Conclusion/Summary:

In this assignment, we explored the versatile applications of Python dictionaries in solving real-world problems. By using dictionaries, we efficiently counted word frequencies in a string, implemented a simple and interactive phonebook application, filtered student records based on criteria, and converted data between dictionaries and list of tuples. These tasks demonstrated the power of dictionaries in organizing, retrieving, and manipulating data efficiently. This exercise highlights the importance of choosing the right data structure to simplify complex operations in Python.

Student Signature & Date	**Marks:**	**Evaluator Signature & Date**