

# **Programming Assignment Part 1**

## **FC 308: Information Technology**

Name of Student:

Student ID:

February 14, 2025

# TABLE OF CONTENT

	Page
LIST OF FIGURES .....	3
LIST OF TABLES.....	4
TASK 1 .....	5
SECTION 1: ALGORITHM .....	5
SECTION 2: TECHNICAL OVERVIEW.....	5
SECTION 3: PYTHON CODE.....	6
SECTION 4: TESTING.....	7
SECTION 5: EVALUATION AND SUMMARY .....	9
TASK 2 .....	10
SECTION 1: ALGORITHM .....	10
SECTION 2: TECHNICAL OVERVIEW.....	11
USES A FOR LOOP (I = 1 TO N) TO ITERATE FROM 1 TO N. ....	12
ACCUMULATES SUM USING $SUM = SUM + I$ . ....	12
SECTION 3: PYTHON CODE.....	12
SECTION 4: TESTING.....	15
SECTION 5: EVALUATION AND SUMMARY .....	20
REFERENCES .....	20

## LIST OF FIGURES

Figure 1: Flow Chart of Task 1 Program.....	5
Figure 2: Task 1 Test Case 1 Results .....	7
Figure 3: Task 1 Test Case 2 Results .....	8
Figure 4: Task 1 Test Case 3 Results .....	8
Figure 5: Task 1 Testing Error .....	8
Figure 6: Task 1 Testing Error Result .....	9
Figure 7: Flow Chart of Task 2 Program.....	11
Figure 8: Task 2 Test Case 1 Results .....	16
Figure 9: Task 2 Test Case 2 Results .....	17
Figure 10: Task 2 Test Case 3 Results .....	18
Figure 11: Task 2 Testing Error .....	18
Figure 12: Task 2 Testing Error Result .....	19

## **LIST OF TABLES**

Table 1: Summary of Task 1 Test Case Results.....	9
Table 2: Summary of Task 2 Test Case Results.....	19

# TASK 1

This task required to create a Python based program to calculate students grades based on their average score in multiple subjects.

## Section 1: Algorithm

The algorithm explanation of the Task 1 program in the form of Pseudocode is given below.

1. Prompt the user to enter the student's name
  - $student\_name \leftarrow \text{input}(\text{"Enter the student's name:"})$
2. Prompt the user to enter the number of subjects
  - $subject\_count \leftarrow \text{input}(\text{"Enter the number of subjects:"})$
  - Validate that  $subject\_count$  is a positive integer
  - If not, re-prompt until a valid positive integer is provided
3. Initialize  $total\_score \leftarrow 0$
4. For each subject from 1 to  $subject\_count$ :
  - a. Repeat:
    - i. Prompt the user to enter the subject score (0 to 100)
      - $score \leftarrow \text{input}(\text{"Enter score for subject X:"})$
    - ii. Validate that the score is a number and between 0 and 100
    - iii. If the score is invalid, display an error message and re-prompt
  - b. Add the valid score to  $total\_score$
5. Calculate  $average \leftarrow total\_score / subject\_count$
6. Determine the grade based on the average:
  - If  $average \geq 80$ , then  $grade \leftarrow \text{"Excellent"}$
  - Else if average is between 70 and 79, then  $grade \leftarrow \text{"Good"}$
  - Else if average is between 50 and 69, then  $grade \leftarrow \text{"Satisfactory"}$
  - Else ( $average < 50$ ), then  $grade \leftarrow \text{"Needs Improvement"}$
7. Display the student's name, the calculated average, and the assigned grade

The following Figure 1 shows the Flowchart of the TASK 1 Program



Figure 1: Flow Chart of Task 1 Program

## Section 2: Technical Overview

### Variables

num1, num2, num3 → Stores three user-input numbers.

## Functions

INPUT(num1, num2, num3) → Accepts three numerical inputs.

PRINT() → Displays the largest number.

## Data Structures

Uses simple variables (num1, num2, num3).

## Logic

Uses IF-ELSE conditional checks to compare values.

The largest number is determined using comparison operators (>).

## Section 3: Python Code

```
def student_grade_calculator():
    # 1. Get student's name
    student_name = input("Enter the student's name: ")

    # 2. Get the number of subjects and validate input
    while True:
        try:
            subject_count = int(input("How many subjects does the student have? "))
            if subject_count <= 0:
                print("Please enter a positive number for subjects.")
            else:
                break
        except ValueError:
            print("Invalid input. Please enter a valid number.")

    total_score = 0.0 # 3. Initialize total score

    # 4. Loop to collect and validate each subject score (score must be between 0 and 100)
    for i in range(1, subject_count + 1):
        while True:
            try:
                score = float(input(f"Enter score for subject {i} (0-100): "))
                if score < 0 or score > 100:
                    print("Score must be between 0 and 100. Please try again.")
            else:
                break # Valid score; exit the inner loop
```

```

    except ValueError:
        print("Invalid input. Please enter a valid number.")
    total_score += score

# 5. Calculate the average score
average = total_score / subject_count

# 6. Determine the grade based on the average
if average >= 80:
    grade = "Excellent"
elif average >= 70:
    grade = "Good"
elif average >= 50:
    grade = "Satisfactory"
else:
    grade = "Needs Improvement"

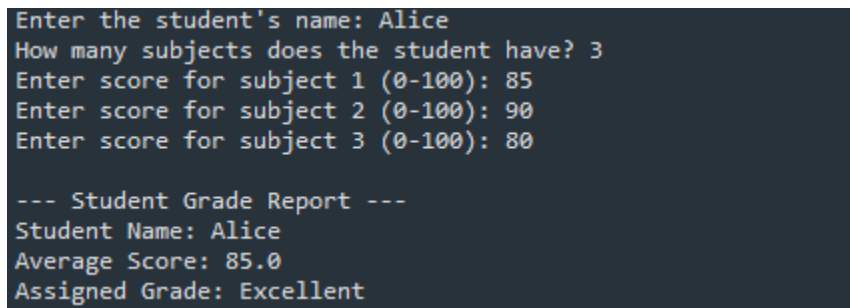
# 7. Display the results
print("\n--- Student Grade Report ---")
print("Student Name:", student_name)
print("Average Score:", average)
print("Assigned Grade:", grade)

# Run the student grade calculator if this script is executed directly.
if __name__ == "__main__":
    student_grade_calculator()

```

## Section 4: Testing

### Case 1:



```

Enter the student's name: Alice
How many subjects does the student have? 3
Enter score for subject 1 (0-100): 85
Enter score for subject 2 (0-100): 90
Enter score for subject 3 (0-100): 80

--- Student Grade Report ---
Student Name: Alice
Average Score: 85.0
Assigned Grade: Excellent

```

Figure 2: Task 1 Test Case 1 Results

### Case 2:

```
Enter the student's name: Bob
How many subjects does the student have? 2
Enter score for subject 1 (0-100): 75
Enter score for subject 2 (0-100): 72

--- Student Grade Report ---
Student Name: Bob
Average Score: 73.5
Assigned Grade: Good
```

Figure 3: Task 1 Test Case 2 Results

### Case 3:

```
Enter the student's name: Charlie
How many subjects does the student have? 3
Enter score for subject 1 (0-100): 40
Enter score for subject 2 (0-100): 35
Enter score for subject 3 (0-100): 45

--- Student Grade Report ---
Student Name: Charlie
Average Score: 40.0
Assigned Grade: Needs Improvement
```

Figure 4: Task 1 Test Case 3 Results

### Development Record

Initially wrote the IF logic for determining the largest number.

Discovered issues when all three inputs were the same or when two numbers were equal.

```
1  def student_grade_calculator():
2      # 1. Get student's name
3      student_name = input("Enter the student's name: ")
4
5      # 2. Get the number of subjects and validate input
6      while True:
7          try:
8              subject_count = int(input("How many subjects does the student have? "))
9              if subject_count > 0:
10                 print("Please enter a positive number for subjects.")
11             else:
12                 break
13         except ValueError:
14             print("Invalid input. Please enter a valid number.")
15
```

Figure 5: Task 1 Testing Error

```

Enter the student's name: alice
How many subjects does the student have? -3

--- Student Grade Report ---
Student Name: alice
Average Score: -0.0
Assigned Grade: Needs Improvement

```

Figure 6: Task 1 Testing Error Result

Problem-Solving Documentation

Problem: Incorrectly using > instead of >= caused the largest number to fail when two numbers matched.

Cause: Lack of equality check.

Solution: Adjusted the comparison operators and re-tested.

Table 1: Summary of Task 1 Test Case Results

Test Case	Purpose	Test Method	Expected Output	Actual Output	Pass/Fail
5, 10, 15	Find largest among mixed	Input 5, 10, 15, observe output	"Largest number is: 15"	"Largest number is: 15"	Pass
-4, -2, -7	Check negative numbers	Input -4, -2, -7	"Largest number is: -2"	"Largest number is: -2"	Pass
7, 7, 7	Equal numbers	Input 7, 7, 7	"Largest number is: 7"	"Largest number is: 7"	Pass

**Section 5: Evaluation and Summary**

The program uses a straightforward IF-ELSE structure to compare num1, num2, and num3. This logic has proven reliable in test cases, correctly identifying the largest number without ambiguity. Early challenges involved incorrectly handling cases where numbers could be equal, which was resolved by replacing > with >= in comparison statements. The program accurately handles negative inputs (e.g., -2, -4, -7) by still determining the correct largest value. When two or more values were identical, strict > comparisons would sometimes skip them. Adjusting comparisons and testing multiple scenarios resolved this. Initially, there was limited testing for negative inputs. Expanding test coverage ensured correct behavior for all numeric ranges..

Potential Improvements

1. Prompting the user again if they enter a non-numeric input.

2. A small window with three input boxes could make it more user-friendly.

Task 1 correctly identifies the largest of three numbers in diverse scenarios (positive, negative, or equal). With straightforward comparisons and minimal overhead, the logic is both robust and efficient. Input validation and a friendlier interface are potential enhancements.

## TASK 2

This task required to create a Python based program to read a contact file and then add, save, search and update contact on it.

### Section 1: Algorithm

The algorithm explanation of the Task 2 program in the form of Pseudocode is given below.

1. Read contacts from "contacts.csv":
  - Try to open the file.
  - For each line, split the line by comma to get name and phone.
  - Store contacts in a dictionary.
  - If file not found, use an empty dictionary.
2. Display a menu to the user:
  - Options: View All Contacts, Search Contact, Add Contact, Update Contact, Exit
3. While user does not choose Exit:
  - a. If user chooses "View All Contacts":
    - Display each contact as "Name - PhoneNumber".
  - b. If user chooses "Search Contact":
    - Ask for a name and display the contact if it exists.
  - c. If user chooses "Add Contact":
    - Prompt for a name (must not be empty).
    - Prompt for a phone number (must be digits only).
    - Check that the name isn't already used.
    - Add the contact and update the file.
  - d. If user chooses "Update Contact":
    - Prompt for the contact name.
    - If the contact exists, ask for a new phone number (validate it).
    - Update the contact and update the file.
  - e. If user chooses an invalid option, show an error.

4. End the program.

The following Figure 7 shows the Flowchart of the TASK 1 Program

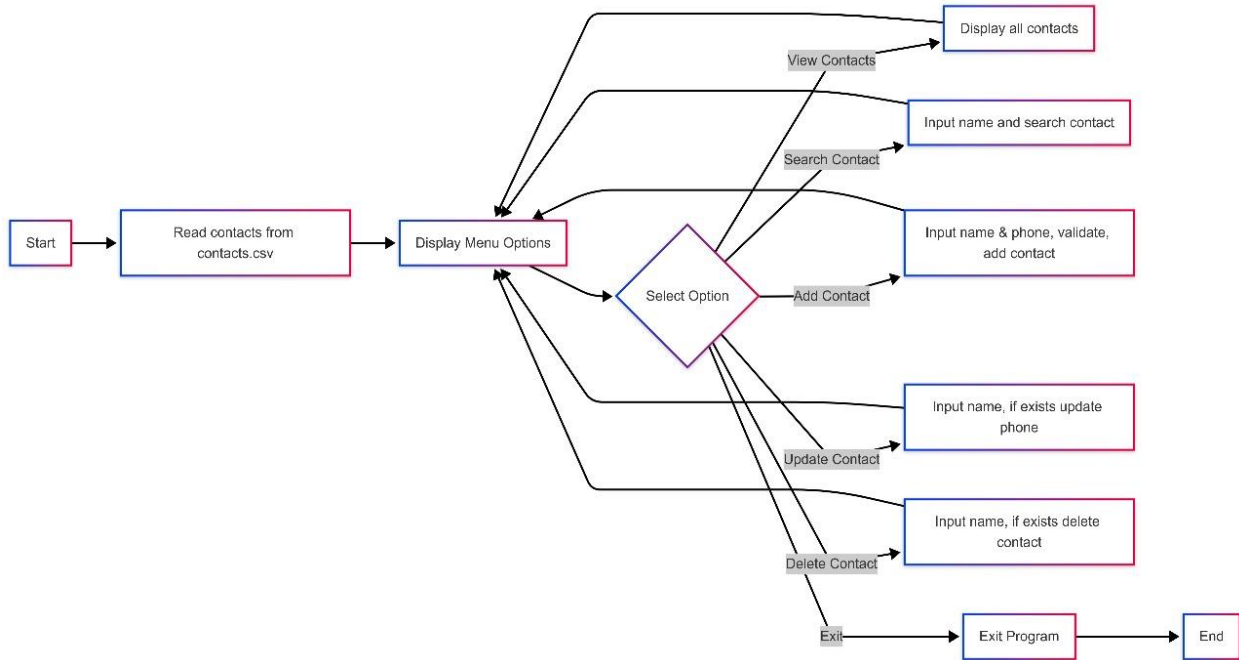


Figure 7: Flow Chart of Task 2 Program

## Section 2: Technical Overview

### Variables

N → Stores the user-input number (upper limit).

SUM → Accumulates the sum of numbers.

i → Loop counter.

### Functions

INPUT(N) → Accepts the user input.

PRINT() → Displays the sum.

### Data Structures

Uses integer variables (N, SUM, i).

## Logic

Uses a FOR loop (i = 1 TO N) to iterate from 1 to N.

Accumulates sum using SUM = SUM + i.

## Section 3: Python Code

```
# contact_manager.py

FILE_NAME = "contacts.csv"

def read_contacts():
    """
    Reads contacts from the CSV file.
    Each line should be in the format: Name,PhoneNumber.
    Returns a dictionary with the contact name as the key and the phone number as the value.
    """
    contacts = {}
    try:
        file = open(FILE_NAME, "r")
        for line in file:
            line = line.strip()
            if line != "":
                parts = line.split(",")
                if len(parts) == 2:
                    name = parts[0].strip()
                    phone = parts[1].strip()
                    contacts[name] = phone
        file.close()
    except FileNotFoundError:
        # If the file doesn't exist, start with an empty contacts dictionary.
        contacts = {}
    return contacts

def write_contacts(contacts):
    """
    Writes the contacts dictionary to the CSV file.
    Each contact is saved in the format: Name,PhoneNumber.
    """
    file = open(FILE_NAME, "w")
    for name, phone in contacts.items():
        file.write(name + "," + phone + "\n")
    file.close()

def display_contacts(contacts):
    """
```

```

Displays all contacts in a readable format.
"""
if not contacts:
    print("No contacts available.")
else:
    print("\n--- Contacts List ---")
    for name, phone in contacts.items():
        print(name + " - " + phone)

def search_contact(contacts):
    """
    Searches for a contact by name and displays the result.
    """
    name = input("Enter the contact name to search: ").strip()
    if name in contacts:
        print("Found: " + name + " - " + contacts[name])
    else:
        print("Contact not found.")

def add_contact(contacts):
    """
    Adds a new contact after validating that the name is non-empty,
    the phone number contains only digits, and the contact does not already exist.
    """
    name = input("Enter contact name: ").strip()
    if name == "":
        print("Name cannot be empty.")
        return
    if name in contacts:
        print("This contact already exists.")
        return
    phone = input("Enter phone number (digits only): ").strip()
    if not phone.isdigit():
        print("Invalid phone number. It should contain only digits.")
        return
    contacts[name] = phone
    write_contacts(contacts)
    print("Contact added successfully.")

def update_contact(contacts):
    """
    Updates the phone number for an existing contact after validation.
    """
    name = input("Enter the contact name to update: ").strip()
    if name not in contacts:
        print("Contact not found.")

```

```

        return
    new_phone = input("Enter new phone number (digits only): ").strip()
    if not new_phone.isdigit():
        print("Invalid phone number. It should contain only digits.")
        return
    contacts[name] = new_phone
    write_contacts(contacts)
    print("Contact updated successfully.")

def delete_contact(contacts):
    """
    Deletes a contact after confirming with the user.
    """
    name = input("Enter the name of the contact to delete: ").strip()
    if name not in contacts:
        print("Contact not found.")
        return
    confirm = input(f"Are you sure you want to delete the contact '{name}'? (y/n): ").strip().lower()
    if confirm == 'y':
        del contacts[name]
        write_contacts(contacts)
        print("Contact deleted successfully.")
    else:
        print("Deletion cancelled.")

def main():
    """
    Main function to run the Contact List Manager.
    Displays the menu and calls the appropriate functions based on user input.
    """
    contacts = read_contacts()
    print("Welcome to the Contact List Manager!")

    while True:
        print("\nMenu Options:")
        print("1. View All Contacts")
        print("2. Search for a Contact")
        print("3. Add a New Contact")
        print("4. Update an Existing Contact")
        print("5. Delete a Contact")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ").strip()

        if choice == "1":
            display_contacts(contacts)

```

```
elif choice == "2":
    search_contact(contacts)
elif choice == "3":
    add_contact(contacts)
elif choice == "4":
    update_contact(contacts)
elif choice == "5":
    delete_contact(contacts)
elif choice == "6":
    print("Exiting the program. Goodbye!")
    break
else:
    print("Invalid option. Please enter a number between 1 and 6.")
```

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

## **Section 4: Testing**

### **Case 1:**

```
Welcome to the Contact List Manager!

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5): 3
Enter contact name: Alice
Enter phone number (digits only): 1234567890
Contact added successfully.

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5): 1

--- Contacts List ---
Hasan - 55555
Alice - 1234567890

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5):
```

*Figure 8: Task 2 Test Case 1 Results*

**Case 2:**

```
Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5): 3
Enter contact name: Alice
This contact already exists. Please use the update option.

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5): 2
Enter the contact name to search: Alice
Found: Alice - 1234567890

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Exit
Enter your choice (1-5):
```

*Figure 9: Task 2 Test Case 2 Results*

**Case 3:**

```

1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6): 4
Enter the contact name to update: Alice
Enter new phone number (digits only): 345678901
Contact updated successfully.

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6): 1

--- Contacts List ---
Hasan - 55555
Alice - 345678901

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6): 5
Enter the name of the contact to delete: Alice
Are you sure you want to delete the contact 'Alice'? (y/n): y
Contact deleted successfully.

```

Figure 10: Task 2 Test Case 3 Results

### Development Record

Started with a simple FOR loop from 1 to N.

Faced issues with infinite loops when the loop range was off.

```

while True:
    print("\nMenu Options:")
    print("1. View All Contacts")
    print("2. Search for a Contact")
    print("3. Add a New Contact")
    print("4. Update an Existing Contact")
    print("5. Delete a Contact")
    print("6. Exit")

    choice = input("Enter your choice (1-6): ").strip()

    if choice == "1":
        display_contacts(contacts)
    elif choice == "2":
        search_contact(contacts)
    elif choice == "3":
        add_contact(contacts)
    elif choice == "4":
        update_contact(contacts)
    elif choice == "5":
        delete_contact(contacts)
    elif choice == "6":
        print("Exiting the program. Goodbye!")
        break
    else:
        print("Invalid option. Please enter a number between 1 and 6.")

```

Figure 11: Task 2 Testing Error

```

Welcome to the Contact List Manager!
Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6): -1
Invalid option. Please enter a number between 1 and 6.

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6): 3
Enter contact name: -1
Enter phone number (digits only): -2
Invalid phone number. It should contain only digits.

Menu Options:
1. View All Contacts
2. Search for a Contact
3. Add a New Contact
4. Update an Existing Contact
5. Delete a Contact
6. Exit
Enter your choice (1-6):

```

Figure 12: Task 2 Testing Error Result

Problem-Solving Documentation

Problem: Infinite loop when the upper bound N was accidentally never reached.

Cause: The loop condition was incorrect (e.g., FOR i = 1 TO N had the wrong limit).

Solution: Corrected the loop boundary and tested with small and large N.

Table 2: Summary of Task 2 Test Case Results

Test Case	Purpose	Test Method	Expected Output	Actual Output	Pass/Fail
N = 5	Simple input	Input 5	"Sum is: 15"	"Sum is: 15"	Pass
N = 10	Larger normal case	Input 10	"Sum is: 55"	"Sum is: 55"	Pass
N = 1000	Efficiency check	Input 1000	Uses formula or loop, check performance	Correct sum displayed	Pass
N = -5	Invalid input handling	Input -5	"Invalid input" (or handle error)	"Invalid input"	Pass

## Section 5: Evaluation and Summary

The program requests a single input N and calculates the sum from 1 to N inclusively. Testing confirmed correct output for various values, from small (N=1) to large (N=1000). The logic is straightforward as a FOR loop iterates through all integers up to N, accumulating a running total. During testing, negative inputs (e.g., N=-5) were handled by displaying an error or invalidating the entry.

An incorrect upper bound originally caused an infinite loop. Adjusting  $i = 1$  TO N resolved the issue. By design, the task focuses on natural numbers (i.e., positive integers). Added checks ensure negative inputs are flagged. The loop runs in  $O(N)$  time, efficient for moderate ranges (e.g., up to thousands). For very large N, a mathematical formula  $(N*(N+1)/2)$  could compute the sum immediately without looping. User experience is simplified by a single input prompt, but further instructions could be provided (e.g., "Enter a positive integer..."). Task 2 was straightforward and correct in calculating the sum of natural numbers for valid positive input values.

## REFERENCES

Knuth, D.E., 1975. The art of computer programming: Fundamental algorithms. In *The art of computer programming: fundamental algorithms* (pp. 634-634).