

**SR UNIVERSITY**  
**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

Course- B.Tech

Type- Programming Elective

Course Code-

Course Name- Problem Solving using Python

Year- 2024-2025

Semester- Odd

Date – 1<sup>st</sup> October 2024

Batch – all batches

**Topics: String, list, tuple, set, dictionary operations**

**A. Course Registration System:**

You are developing a course registration system for a university. The system will prompt students to input their full name (first and last name) along with their desired course. The following rules must be enforced:

Course Validation:

- The system will validate if the chosen course is one of the available options listed in `subject_list = ["AI", "ML", "DS", "IoT", "Blockchain"]`.
- If the inputted course is not in the list, the system will prompt for re-entry, allowing up to a maximum of 3 attempts.
- After 3 invalid attempts, the system will display the message: "Registration Failed. Invalid course selection."

Once a valid course is selected, the system will display the message:

"Registration Successful for <Full Name in Title Case> in <Course>."

Input:

enter full name

enter desired course

Output:

Display appropriate message

**Example 1:****Input**

arjun desai

AI

**Output**

Registration Successful for Arjun Desai in AI.

**Example 2:****Input**

ananya reddy

history

DS

**Output**

Chosen wrong course, select course from available subject list ["AI", "ML", "DS", "IoT"]

Registration Successful for Ananya Reddy in ML.

**Example 3:****Input**

ravi sharma

dance

music

painting

**Output**

Registration Failed for Ravi Sharma. Invalid course selection.

**B. Contact management application:** Create a contact management application in Python using a dictionary. Each contact should have a name and a phone number. Implement the following functionalities:

- a. Add a new contact.
- b. Remove an existing contact.
- c. Update a contact's phone number.

Implement a contact management system using a dictionary in Python. Perform the following tasks:

1. Display an empty dictionary.
2. Add a contact:
  - o Name: Rahul
  - o Phone Number: 1876543210
3. Try to display a contact named Priya, who does not exist in the dictionary.
4. Add the following contacts:
  - o Name: Aisha, Phone Number: 2276543210
  - o Name: Raghava, Phone Number: 3333543210
5. Display all contacts in the dictionary.
6. Display only the phone numbers (values) of all contacts.
7. Display the names (keys) of all contacts.
8. Display items (contacts) whose names begin with the letter 'A'.

**Output:**

Initial contacts: { }

Contact added: Rahul - 1876543210

Contact not found: Priya

All contacts:

Rahul - 1876543210

Aisha - 2276543210

Raghava - 3333543210

Phone numbers (values):

['1876543210', '2276543210', '3333543210']

Contact names (keys):

['Rahul', 'Aisha', 'Raghava']

Contacts whose names begin with 'A':

Aisha – 2276543210

### **C. Fruit Price Management System:**

You are tasked with creating a Fruit Price Management System using a dictionary in Python.

#### **Follow these steps:**

- Define an Empty Dictionary: Create a dictionary named fruit\_prices to store fruit names as keys and their corresponding prices as values.
- Add Items One by One: Add the following six fruits along with their prices to the dictionary:
  - Apple: 150
  - Banana: 80
  - Cherry: 100
  - Mango: 180
  - Orange: 60
  - Grapes: 150
- Display All Items: Write code to display all fruit-price pairs stored in the dictionary.
- Display Keys: Extract and display the names of the fruits (keys) from the dictionary.
- Display Values: Extract and display the prices (values) from the dictionary.
- Calculate Total cost and Surcharge: Calculate the total cost by summing the prices of all fruits and then add a 10% surcharge on the total bill.
- Merge and Order Keys: Sort the fruit names in alphabetical order and display them.
- Display Total Bill: Finally, display the total bill after listing the ordered fruits.

#### **Output:**

All items (Fruit - Price):

Apple - 150

Banana - 80

Cherry - 100

Mango - 180

Orange - 60

Grapes - 150

Fruit names (keys):

['Apple', 'Banana', 'Cherry', 'Mango', 'Orange', 'Grapes']

Prices (values):

[150, 80, 100, 180, 60, 150]

Total bill: 720

Surcharge (10%): 72.0

Total bill with surcharge: 792.0

Ordered fruit names:

Apple, Banana, Cherry, Grapes, Mango, Orange

Total bill for the ordered fruits with surcharge: 792.0

#### **D. Set Operations for CSE Sections**

You are managing student records for two events at a university. Each event has a set of students identified by their unique IDs.

- Create a set called Hackathon representing students participating in the Hackathon:

Hackathon = {101, 102, 103, 104, 105, 106}

- Create another set called CodeFest representing students participating in CodeFest:

CodeFest = {104, 105, 106, 107, 108, 109}

- Display both sets Hackathon and CodeFest

- Perform Set Operations:

- **Union:** Calculate and display the union of Hackathon and CodeFest to find all unique students across both competitions.

- **Intersection:** Calculate and display the intersection of Hackathon and CodeFest to

find students participating in both competitions.

- **Difference:**

- Calculate and display the difference between Hackathon and CodeFest (students in Hackathon not in CodeFest).
  - Calculate and display the difference between CodeFest and Hackathon (students in CodeFest not in Hackathon).
- Check if Hackathon is a subset of CodeFest and display the result.
  - Check if CodeFest is a superset of Hackathon and display the result.
  - Add a new student ID to Hackathon and display the updated set.
  - Remove a student ID from CodeFest and display the updated set.