**Name - Sparsh Sharma Student ID – U3315385**

**Introduction to Information Technology (4478/8936)**

**Assignment II — Python Fundamentals**

Table of Contents

[Case Study 1 - Smart Classroom Monitor 2](#_Toc208913426)

[**Step 1: Explaining the problem** 2](#_Toc208913427)

[**Step 2: Inputs & Outputs** 2](#_Toc208913428)

[**Step 3: Algorithm Design** 3](#_Toc208913429)

[**Step 4: Flowchart** 6](#_Toc208913430)

[Step 5: Python Code 7](#_Toc208913431)

[Step 6: Testing 11](#_Toc208913432)

[Step 7: Refinement via GenAI 14](#_Toc208913433)

[Case Study 2 — Smart Classroom Monitor 15](#_Toc208913434)

[Step 1: Explaining the problem 15](#_Toc208913435)

[Step 2: Inputs & Outputs 15](#_Toc208913436)

[Step 3: Algorithm Design 16](#_Toc208913437)

[Step 3: Algorithm Design 17](#_Toc208913438)

[Step 4: Flowchart 18](#_Toc208913439)

[Step 5: Python Code 18](#_Toc208913440)

[Step 6: Testing 22](#_Toc208913441)

[Step 7: Refinement 23](#_Toc208913442)

[Case Study 3 — A Boolean Circuit Equivalence 23](#_Toc208913443)

[Step 1: Translate Boolean Circuits to Expressions 23](#_Toc208913444)

[Step 2: Python Code 24](#_Toc208913445)

[Step 3: Truth Table 25](#_Toc208913446)

[Step4 : Comparison 26](#_Toc208913447)

# Case Study 1 - Smart Classroom Monitor

## **Step 1: Explaining the problem**

For the cafe in the campus, a Point-of-Sale (POS) system need to be created. The menu has six or more than six items which belongs to certain category. The program shows the menu to the users and the users will select the items from the list. The users can add the items to the cart and they can print their receipt also. The cart shows many items which are added into it, finds the subtotal, adds tax and also checks whether the cart items has any discount.

The system requires a dictionary for storing the items in the menu, the price of the menu and the product's category, a set to monitoring the categories in the cart and a list to denote the cart.

The program uses the while loop to show the choices to the user like displaying the menu, including the products to the cart, viewing the cart, checkout (print the receipt with totals and discounts) and exiting the menu.

## **Step 2: Inputs & Outputs**

**Inputs which are collected from user:**

**1. Menu choice (int)**

User chooses the options in the main menu:

1 denotes Show menu, 2 denotes Add item, 3 denotes View cart, 4 denotes Checkout and 5 denotes Exit.

**2. Choosing the Item**

User chooses item to include the item to the cart.

**3. Quantity**

Number of quantities of the products which need to be included in cart.

**4. Eligibility of Student discount**

The user chooses either "y" / "n" to choose whether they are student or not.

**Outputs (to console)**

**1. Menu display (string)**

Shows all the available items details and the item's category details.

**2. Cart contents (list of strings)**

Shows the item which are available in the cart.

**3. Receipt (formatted string)**

Line items: Item's details like name of the item, price of the item, quantity of the item and total per item.

Discount is calculated by finding 5% off if student.

Subtotal is calculated by finding sum of item's costs.

Final Total is calculated by using the formula (subtotal + tax – discount)

Tax is calculated by finding 10% of subtotal.

## **Step 3: Algorithm Design**

**INPUT:**

**Menu option**

Source: Using console user will be providing the input

Validation: Must be an integer and the values should be between 1 to 5

Type: int

Units: N/A

**Item selection**

Source: Using console user will be providing the input

Validation: Must be available in the menu dictionary keys

Type: str or int

Units: N/A

**Quantity** (optional)

Source: Using console user will be providing the input

Validation: Integer value should be greater than 1

Type: int

Units: The total count of items

**Student discount eligibility**

Source: During checkout, collect the user input like y or n

Validation: Only n or y will be accepted

Type: boolean or string datatype

Units: N/A

**PROCESS**

**Task A: Display the menu**

Loop through menu dictionary and display item name, price (rounded to 2 decimals), and category.

**Task B: Add item to cart**

Collect then name or ID of the item to check whether they are valid.

Collect the quantity value.

Append the item and quantity) to the cart.

**Task C: View cart**

Loop through the cart.

Show each item with the quantity and price.

To display the unique types, maintain the set of categories.

**Task D: Checkout**

Loop through cart the to find the subtotal.

Find tax (tax = 0.10 × subtotal)

If the user is a student then the discount = 0.05 × subtotal otherwise discount = 0

Check whether the food is available in the categories and the drink is available in the categories then apply the meal deal reduction.

Find the final total = subtotal + tax – discount – deal

After checkout, then empty the cart.

**Loops**

**Select the items until the** valid input is entered

**Decisions**

If input is not valid then produce the error and prompt again.

If cart is empty and if the user chooses the checkout then generate warning and return to the menu.

**OUTPUT**

**Menu:**

--- Campus Café Menu ---

1. Coffee ($2.50) [Drink]

2. Muffin ($1.80) [Food]

...

**Cart:**

It has format: "2x Muffin – $3.60"

It displays the unique categories like "Categories in cart: {Food, Drink}"

**Receipt:**

Line items, subtotal, tax, discount and final total

Round floats to 2 decimal places

**Example:**

--- Receipt ---

2x Coffee $5.00

1x Muffin $1.80

--------------------

Subtotal: $6.80

Tax (10%): $0.68

Student Disc: -$0.34

--------------------

TOTAL: $7.14

## **Step 4: Flowchart**

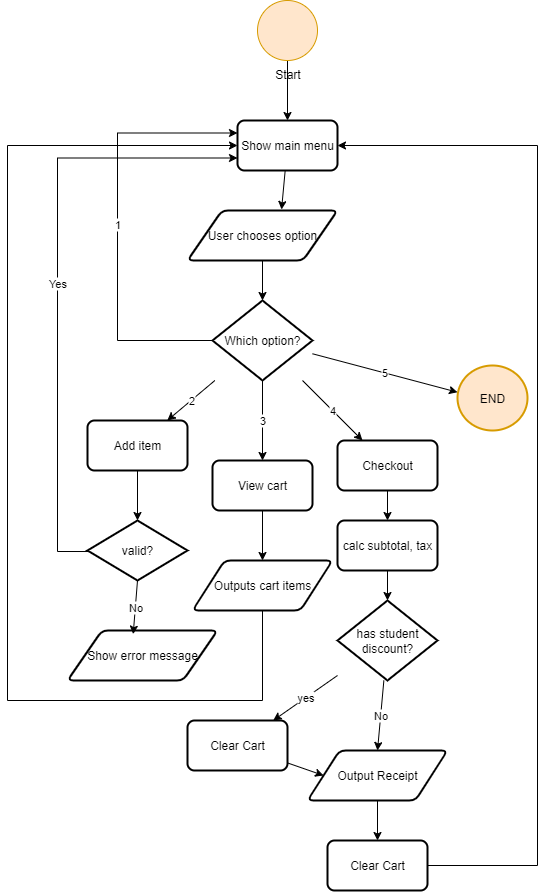


Figure: Case Study 1 Flow Chart

## Step 5: Python Code

# Campus Café Checkout System

# Learning goals: dict (menu), list (cart), set (categories), while/for loops, floats, Booleans, functions

# -----------------------

# Menu dictionary: item -> (price, category)

# -----------------------

menu = {

"Coffee": (2.50, "Drink"),

"Tea": (2.00, "Drink"),

"Juice": (3.00, "Drink"),

"Muffin": (1.80, "Food"),

"Sandwich": (4.50, "Food"),

"Salad": (3.80, "Food")

}

# Cart will store the values of item\_name and quantity

cart = []

# -----------------------

# Functions

# -----------------------

def show\_menu():

"""Display the café menu."""

print("\n--- Campus Café Menu ---")

for idx, (item, (price, category)) in enumerate(menu.items(), start=1):

print(f"{idx}. {item:10} ${price:.2f} [{category}]")

def add\_item(cart):

"""Include the item to the cart."""

show\_menu()

choice = input("Enter item name to add: ").strip().title()

if choice in menu:

try:

qty = int(input(f"Enter quantity of {choice}: "))

if qty > 0:

cart.append((choice, qty))

print(f"Added {qty}x {choice} to cart.")

else:

print("Quantity must be at least 1.")

except ValueError:

print("Invalid quantity. Please enter a number.")

else:

print("Item not found in the menu.")

def view\_cart(cart):

"""Show the items which are available in the cart."""

if not cart:

print("\n🛒 Cart is empty.")

return

print("\n--- Your Cart ---")

categories = set()

for item, qty in cart:

price, category = menu[item]

line\_total = price \* qty

print(f"{qty}x {item:10} ${line\_total:.2f}")

categories.add(category)

print("Unique categories in cart:", categories)

def checkout(cart):

"""Display the receipt and find the totals with tax, discounts, and meal deals."""

if not cart:

print("\n Cart is empty. Nothing to checkout.")

return

print("\n--- Receipt ---")

subtotal = 0

categories = set()

for item, qty in cart:

price, category = menu[item]

line\_total = price \* qty

subtotal += line\_total

categories.add(category)

print(f"{qty}x {item:10} ${line\_total:.2f}")

tax = 0.10 \* subtotal

# Ask for student discount

discount = 0

student = input("Are you a student? (y/n): ").strip().lower()

if student == "y":

discount = 0.05 \* subtotal

# Apply meal deal: $1 off if both food and drink are present

meal\_deal = 0

if "Food" in categories and "Drink" in categories:

meal\_deal = 1.00

print("Meal deal applied: $1.00 off")

total = subtotal + tax - discount - meal\_deal

print("---------------------------")

print(f"Subtotal: ${subtotal:.2f}")

print(f"Tax (10%): ${tax:.2f}")

print(f"Student Disc: -${discount:.2f}")

if meal\_deal > 0:

print(f"Meal Deal: -${meal\_deal:.2f}")

print("---------------------------")

print(f"TOTAL: ${total:.2f}")

print("---------------------------")

# Clear cart after checkout

cart.clear()

print("Checkout complete. Thank you!")

# -----------------------

# Loop for Main Program

# -----------------------

def main():

while True:

print("\n--- Campus Café POS ---")

print("1. Show Menu")

print("2. Add Item")

print("3. View Cart")

print("4. Checkout")

print("5. Exit")

try:

choice = int(input("Enter your choice: "))

except ValueError:

print("Invalid input. Please enter a number (1-5).")

continue

if choice == 1:

show\_menu()

elif choice == 2:

add\_item(cart)

elif choice == 3:

view\_cart(cart)

elif choice == 4:

checkout(cart)

elif choice == 5:

print("Exiting!")

break

else:

print("Invalid choice. Please select 1-5.")

# Run to get the output

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Step 6: Testing

Testing

Test Case 1: Show menu

Input: Choice = 1

Expected output (handwritten):

--- Campus Café Menu ---

1. Coffee $2.50 [Drink]

2. Tea $2.00 [Drink]

3. Juice $3.00 [Drink]

4. Muffin $1.80 [Food]

5. Sandwich $4.50 [Food]

6. Salad $3.80 [Food]

Notes: Checks whether the items, prices and categories are showed.

Test Case 2: Add item (valid)

Input: Choice = 2, Item = Coffee, Quantity = 2

Added 2x Coffee to cart.

Notes: Item is stored as ("Coffee", 2) in the cart.

Test Case 3: Add item (invalid item name)

Input: Choice = 2, Item = Pizza

Expected Output:

Item not found in the menu.

Notes: No change to cart.

Test Case 4: View cart (non-empty)

Precondition: Cart = [("Coffee", 2), ("Muffin", 1)]

Input: Choice = 3

Expected Output:

--- Your Cart ---

2x Coffee $5.00

1x Muffin $1.80

Unique categories in cart: {'Food', 'Drink'}

Notes: Makes sure that the categories are monitored correctly.

Test Case 5: Checkout

Precondition: Cart = [("Coffee", 2), ("Muffin", 1)]

Input: Choice = 4 → Student? y

Expected calculation:

Subtotal = (2 × 2.50) + (1 × 1.80) = $6.80

Tax = 10% of 6.80 = $0.68

Student Discount = 5% of 6.80 = $0.34

Meal Deal = $1.00 (food + drink present)

Total = 6.80 + 0.68 – 0.34 – 1.00 = $6.14

Expected Output:

--- Receipt ---

2x Coffee $5.00

1x Muffin $1.80

Meal deal applied: $1.00 off

---------------------------

Subtotal: $6.80

Tax (10%): $0.68

Student Disc: -$0.34

Meal Deal: -$1.00

---------------------------

TOTAL: $6.14

---------------------------

Checkout complete. Thank you!

Notes: Checks the logic for discount and deal.

Test Case 6: Checkout

Precondition: Cart = [("Tea", 1)]

Input: Choice = 4 → Student? n

Expected calculation:

Subtotal = 2.00

Tax = 0.20

Discount = 0

Meal deal = 0

Total = 2.20

Expected Output:

--- Receipt ---

1x Tea $2.00

---------------------------

Subtotal: $2.00

Tax (10%): $0.20

Student Disc: -$0.00

---------------------------

TOTAL: $2.20

---------------------------

Checkout complete. Thank you!

Test Case 7: Checkout with empty cart

Precondition: Cart = []

Input: Choice = 4

Expected Output:

Cart is empty so no need to checkout.

The Anaconda can be used by Python for making easy computations (Rolon-Mérette, Ross, Rolon-Mérette, & Church, 2016).

## Step 7: Refinement via GenAI

**Prompt provided to GenAI:**

“Using python code, how to generate the receipt which will be readable to enhance the checkout process of the Campus Café ?”

I have modified the receipt printing part to use the formats with f-strings to make sure that the price and totals are in columns. For example:

print(f"{qty}x {item:<12} ${line\_total:>6.2f}")

print(f"{qty}x {item:10} ${line\_total:.2f}")

**Justification:**

This refinement increases the readability of the receipt like a real café bill and has clear alignment of item names and costs. Understanding Python functions and libraries is very important to write reusable code (Ross, Church, & Rolon-Mérette, 2021).

# Case Study 2 — Smart Classroom Monitor

## Step 1: Explaining the problem

A program is created for the Smart Classroom Monitor problem. It will be monitoring the different aspects of the classroom. The system will be managing the state of the classroom. The program manages the student's attendance details by comparing the attendance and the capacity of the room. It checks whether the projector is in on state or off state. The system manages the capacity of the room and the manages the topic. The program makes use of the helper function for finding the minimum, maximum and the average values. Menu need to have loops with options which allows the users for setting the topic or including or removing the student, including the temperature or generating the report. The program needs to trigger alert if the room is full or the attendance is more than the capacity or if there is abnormal temperature. The program also remaining the user to on the projector if the topic is available. The aim is to create a smart classroom which has Python code with dictionaries, lists, sets, tuples, conditionals, functions and loops.

## Step 2: Inputs & Outputs

**Inputs**

Projector state: The user toggles off or on based on true or false values.

Room capacity: integer (int), number of seats in the classroom.

Topic: string (str) like data structure.

Student names: string (str), It will be included or removed from the attendance.

Temperature readings: float (float, in °C), it is added to the temperature log.

Menu choices: integer (int), it is chosen from the menu to choose an action.

Exit confirmation: string (str) which denotes whether the user is leaving the program.

**Outputs**

Updated classroom state: dictionary (dict) which shows the status of the projector, capacity and topic.

Attendance status: count of students presents and the set of student names (set).

Temperature statistics: tuple having temperature in °C.

Alerts are triggered during attendance is more than the capacity or the temperature is abnormal and to provide the remainder.

## Step 3: Algorithm Design

**1) INPUT**

Menu choice is collected from the from user (int, 1–7). Numbers should be within the range. Numerical computing in Python has become more efficient (Ascher, Dubois, Hinsen, Hugunin, & Oliphant, 2001).

Projector toggle (yes or no) is collected from the from user.

Topic is collected from the from user (str, e.g., "Math Lecture"). It will check whether it is not empty.

Student name is collected from the from user (str, added to or removed from set). It will check whether string is not empty.

Temperature reading is collected from the from user (float, in °C). It will check whether it is within the range.

Exit confirmation is collected from the user based on yes or no value.

**2) PROCESS**

Task A: Manage the state of the classroom.

Store projector status (bool), room capacity (int) and topic (str).

Update the state of the projector state using the toggle.

Update the topic using the input collected from the user. Multithreading in Python will execute many tasks concurrently (Matloff & Hsu, 2007).

Task B: Attendance management (set).

Include the name of the student to the set.

Eliminate the student name.

Compare the attendance count and the capacity of the room.

Task C: Temperature logging (list).

Add new temperature readings.

Use helper function for calculation purpose.

Task D: Alerts.

Decision 1: If attendance > capacity then display ROOM is FULL.

Decision 2: If temperature < 16 or > 28 then display the temperature warning.

Decision 3: While exiting if the topic is set but projector is off the display that the projector is off.

**Loop(s):**

While True loop need to be used to repeatedly show the menu.

**3) OUTPUT**

Room state report displays the projector status, capacity, topic.

Attendance status is displayed.

Temperature stats is displayed.

Alerts are displayed.

Final report on exit is showed.

## Step 3: Algorithm Design

BEGIN

SET room\_state = {projector\_on: False, capacity: 30, topic: ""}

SET temperature = []

SET attendance = {}

WHILE user is not available

SHOW menu

Collect the choice

IF choice is not valid THEN PRINT "Invalid"

ELSE

IF choice = toggle projector THEN UPDATE room\_state

IF choice = set the topic THEN UPDATE room\_state

IF choice = add the student or remove the student THEN UPDATE attendance

IF choice = add the temperature THEN APPEND to temperatures

IF choice = stats THEN CALCULATE min,max,avg

IF choice = report THEN PRINT room\_state, attendance, temps

ENDIF

END WHILE

IF topic set AND projector\_off THEN DISPLAY reminder

PRINT final summary

END

## Step 4: Flowchart

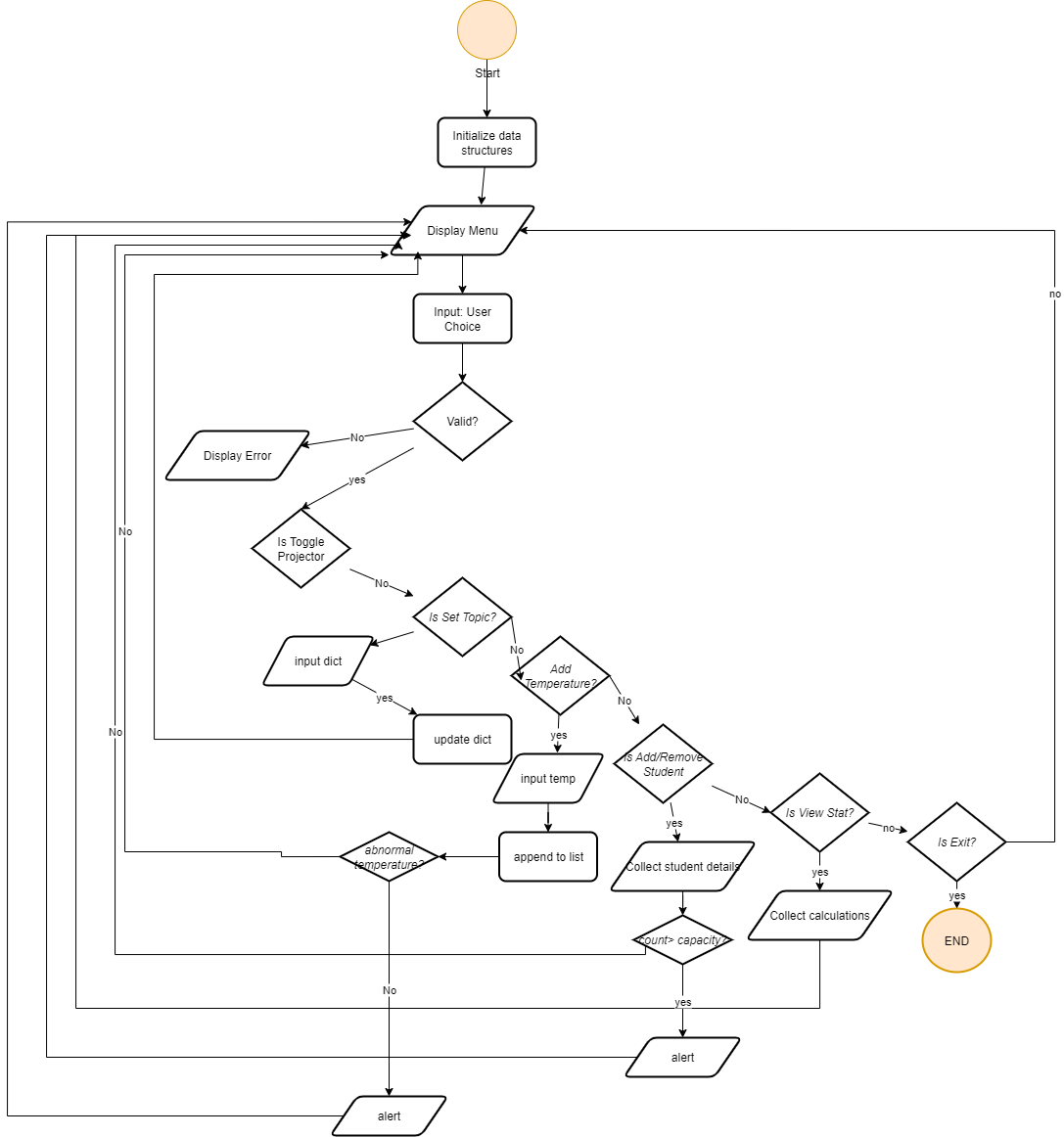


Figure: Case Study 2 FlowChart

Python allows to perform pipelines efficiently (Persson, Gonzalez, & Viren, 2010).

## Step 5: Python Code

# Smart Classroom Monitor

# -----------------------

# Initial Data

# -----------------------

room\_state = {"projector\_on": False, "capacity": 30, "topic": ""}

attendance = set()

temperatures = []

# -----------------------

# Functions

# -----------------------

def toggle\_projector(state):

state["projector\_on"] = not state["projector\_on"]

print(f"Projector is now {'ON' if state['projector\_on'] else 'OFF'}.")

def set\_topic(state):

topic = input("Enter topic for the class: ").strip()

state["topic"] = topic

print(f"Topic set to: {topic}")

def add\_student(attendance\_set, state):

name = input("Enter student name to add: ").strip()

if name:

attendance\_set.add(name)

print(f"{name} added.")

if len(attendance\_set) > state["capacity"]:

print("ROOM FULL!")

else:

print("Invalid name.")

def remove\_student(attendance\_set):

name = input("Enter student name to remove: ").strip()

if name in attendance\_set:

attendance\_set.remove(name)

print(f"{name} removed.")

else:

print("Student not found.")

def add\_temperature(temp\_list):

try:

temp = float(input("Enter temperature reading (°C): "))

temp\_list.append(temp)

if temp < 16 or temp > 28:

print("Temperature Warning!")

except ValueError:

print("Invalid input. Enter a number.")

def stats(temp\_list):

if not temp\_list:

print("No temperature data.")

return

min\_temp = min(temp\_list)

max\_temp = max(temp\_list)

avg\_temp = sum(temp\_list)/len(temp\_list)

print(f"Temperature Stats -> Min: {min\_temp:.1f}°C, Max: {max\_temp:.1f}°C, Avg: {avg\_temp:.1f}°C")

def report(state, attendance\_set, temp\_list):

print("\n--- Classroom Report ---")

print(f"Projector: {'ON' if state['projector\_on'] else 'OFF'}")

print(f"Topic: {state['topic'] or 'No topic set'}")

print(f"Attendance ({len(attendance\_set)}/{state['capacity']}): {', '.join(attendance\_set) if attendance\_set else 'None'}")

stats(temp\_list)

print("------------------------")

# -----------------------

# Main Loop

# -----------------------

def main():

while True:

print("\n--- Smart Classroom Monitor ---")

print("1. Toggle Projector")

print("2. Set Topic")

print("3. Add Student")

print("4. Remove Student")

print("5. Add Temperature")

print("6. View Stats")

print("7. Report / Exit")

choice = input("Enter your choice (1-7): ").strip()

if choice == "1":

toggle\_projector(room\_state)

elif choice == "2":

set\_topic(room\_state)

elif choice == "3":

add\_student(attendance, room\_state)

elif choice == "4":

remove\_student(attendance)

elif choice == "5":

add\_temperature(temperatures)

elif choice == "6":

stats(temperatures)

elif choice == "7":

if room\_state["topic"] and not room\_state["projector\_on"]:

print("Reminder: Projector is OFF while a topic is set!")

report(room\_state, attendance, temperatures)

break

else:

print("Invalid choice. Enter 1-7.")

# -----------------------

# Run Program

# -----------------------

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Step 6: Testing

Test Case 1: Toggle Projector

Input: Choice 1 → toggle projector

Expected Output: Projector is now ON (or OFF if toggled again)

Result: Works as expected

Test Case 2: Set Topic

Input: Choice 2 → topic "Math Lecture"

Expected Output: Topic set to: Math Lecture

Result: Works

Test Case 3: Add Students

Input: Choice 3 → add "Alice", "Bob"

Expected Output: Names added; if count > capacity → "ROOM FULL!"

Result: Attendance set modifies correctly and ROOM FULL alert works

Test Case 4: Add Temperature

Input: Choice 5 → 20, 15, 30

Expected Output: Temperatures appended; warnings for <16 or >28 → "Temperature Warning!"

Result: Alerts is shown.

Test Case 5: Stats

Input: Choice 6

Expected Output: Min: 15.0°C, Max: 30.0°C, Avg: 21.7°C

Result: Correct statistics

Test Case 6: Report / Exit

Input: Choice 7 with topic set & projector off

Expected Output: Show an alert "Projector is OFF while a topic is set!" and display the summary of state, attendance and temps

Result: Report is generated as expected

## Step 7: Refinement

Prompt: Enhance the Python code to provide output more clearly.

What changed: Included a neat format print statements which are formatted with comma and each action has functions with alerts.

Justification: Enhanced the readability, user experience and maintainability.

# Case Study 3 — A Boolean Circuit Equivalence

## Step 1: Translate Boolean Circuits to Expressions

**Circuit A**

**Given steps:**

(a) not\_a = not a

(b) not\_b = not b

(c) nand\_c = not (not\_a and not\_b) → NAND operation

(d) not\_b2 = not b (same as b, reused)

(e) and\_e = (not\_b2 and nand\_c and a) → AND of d, c, a

(f) and\_f = (b and nand\_c and a) → AND of b, c, a

(g) out\_A = and\_e or and\_f → OR of e and f

So, out\_A = ((not b and (not (not a and not b)) and a) or (b and (not (not a and not b)) and a))

**Circuit B**

Given steps:

(h) not\_b = not b

(i) or\_i = not\_b or c

(j) out\_B = or\_i and a

So, final Circuit B expression in Python terms:

out\_B = ((not b or c) and a)

## Step 2: Python Code

# Boolean Circuit Simulation

def circuit\_A(a, b):

# Circuit A steps

not\_a = not a # a)

not\_b = not b # b)

nand\_c = not (not\_a and not\_b) # c)

not\_b2 = not b # d)

and\_e = not\_b2 and nand\_c and a # e)

and\_f = b and nand\_c and a # f)

out\_A = and\_e or and\_f # g)

return out\_A

def circuit\_B(a, b, c):

# Circuit B steps

not\_b\_h = not b # h)

or\_i = not\_b\_h or c # i)

out\_B = or\_i and a # j)

return out\_B

# -----------------------

# Get Inputs from User

# -----------------------

def get\_boolean\_input(prompt):

while True:

val = input(prompt + " (True/False): ").strip().lower()

if val in ['true', 't', '1']:

return True

elif val in ['false', 'f', '0']:

return False

else:

print("Invalid input. Enter True or False.")

# User input

A = get\_boolean\_input("Enter value for A")

B = get\_boolean\_input("Enter value for B")

C = get\_boolean\_input("Enter value for C")

# Compute outputs

X = circuit\_A(A, B) # Output of Circuit A

Y = circuit\_B(A, B, C) # Output of Circuit B

# Display results

print("\n--- Circuit Outputs ---")

print(f"Input: A={A}, B={B}, C={C}")

print(f"Output X (Circuit A): {X}")

print(f"Output Y (Circuit B): {Y}")

## Step 3: Truth Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **X (Circuit A)** | **Y (Circuit B)** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## Step4 : Comparison

