**Case Study 1: Campus Café Checkout**

**Step 1:- Understanding the problem**

The aim of this case study is to design and implement a point-of-sale system specifically for a campus café environment. Such a system should replicate the workflow of a real checkout counter, where a cashier first presents available items to the customer, then records their order into a system, and finally produces a receipt summarizing the transaction.

The program must therefore support multiple essential functions: displaying menu items with prices and categories, allowing users to add their selections to a digital cart, reviewing the cart before purchase, and generating a final receipt that includes details such as discounts, taxes, and the overall total. This ensures the program not only fulfills functional requirements but also simulates an authentic real-world scenario.

**Step 2:- Inputs and Outputs**

**Inputs:**

* Menu choice (string/integer) – The user selects options such as showing the menu, adding an item, viewing the cart, checking out, or exiting.
* Item number (string/integer) – When adding an item, the user specifies the corresponding item number from the menu.
* Student discount (y/n) – At checkout, the system asks whether the customer is a student. If yes, a discount is applied.

**Outputs:**

* Displayed menu – The café’s items shown with names, categories, and prices.
* Cart summary – A list of items added by the customer along with a subtotal.
* Receipt – A formatted output showing each line item with price, subtotal, discount (if any), tax, and final total.

**Step 3: Algorithm**

The algorithm is a step-by-step description of how the program will process data and guide the user through interactions.

**INPUT:**

* User choice (menu option: show menu, add item, checkout, exit)
* Item number and quantity (integers)
* Discount code/student discount eligibility (y/n)

**PROCESS:**

* Initialize cart and prices
* Display menu items
* Ask user to select an item → validate item exists
* If valid: ask for quantity → add item and quantity to cart
* Ask if more items to add
* Compute subtotal → display subtotal
* Add tax to subtotal
* Check meal deal eligibility → apply if valid
* Ask for student discount → validate discount code (if any) → apply or skip
* Compute final total (subtotal + tax – discounts)
* Display final total
* Print receipt
* Save transaction to sales log
* Clear cart for next customer
* Repeat loop until user exits checkout

**OUTPUT:**

* Menu list with items and prices
* Cart summary with line items and subtotal
* Receipt including: subtotal, applied discounts, tax, and final total
* Confirmation message after checkout and exit message

**Step 4:-Pseudocode and Flowchart**

**Pseudocode:**

# Constants

TAX\_RATE = 0.10

STUDENT\_DISCOUNT = 0.05

# Initialize

cart = []

prices = {"Coffee": 2.50, "Sandwich": 4.00, "Tea": 1.50, "Juice": 3.00}

meal\_deals = [("Coffee", "Sandwich")] # example combo

# Start Checkout Loop

while True:

print("1. Show Menu")

print("2. Add Item")

print("3. View Cart")

print("4. Checkout")

print("5. Exit")

choice = input("Enter your choice: ")

# ---- SHOW MENU ----

if choice == "1":

for item, price in prices.items():

print(item, ":", price)

# ---- ADD ITEM ----

elif choice == "2":

item = input("Select item: ")

if item not in prices:

print("Invalid item!")

else:

qty = int(input("Enter quantity: "))

cart.append((item, qty))

print(qty, item, "added to cart")

# ---- VIEW CART ----

elif choice == "3":

if not cart:

print("Cart is empty")

else:

subtotal = sum(prices[i] \* q for i, q in cart)

print("Cart:", cart)

print("Subtotal =", subtotal)

# ---- CHECKOUT ----

elif choice == "4":

if not cart:

print("Nothing to checkout")

else:

subtotal = sum(prices[i] \* q for i, q in cart)

print("Subtotal =", subtotal)

# Apply tax

tax = subtotal \* TAX\_RATE

total = subtotal + tax

# Check meal deal eligibility

for deal in meal\_deals:

if all(any(i == d for i, \_ in cart) for d in deal):

print("Meal deal applied!")

total \*= 0.9 # example 10% meal deal discount

break

# Student discount

student = input("Apply student discount? (y/n): ")

if student.lower() == "y":

total -= total \* STUDENT\_DISCOUNT

# Final receipt

print("Final Total =", round(total, 2))

print("Receipt printed.")

print("Transaction saved.")

cart.clear()

# ---- EXIT ----

elif choice == "5":

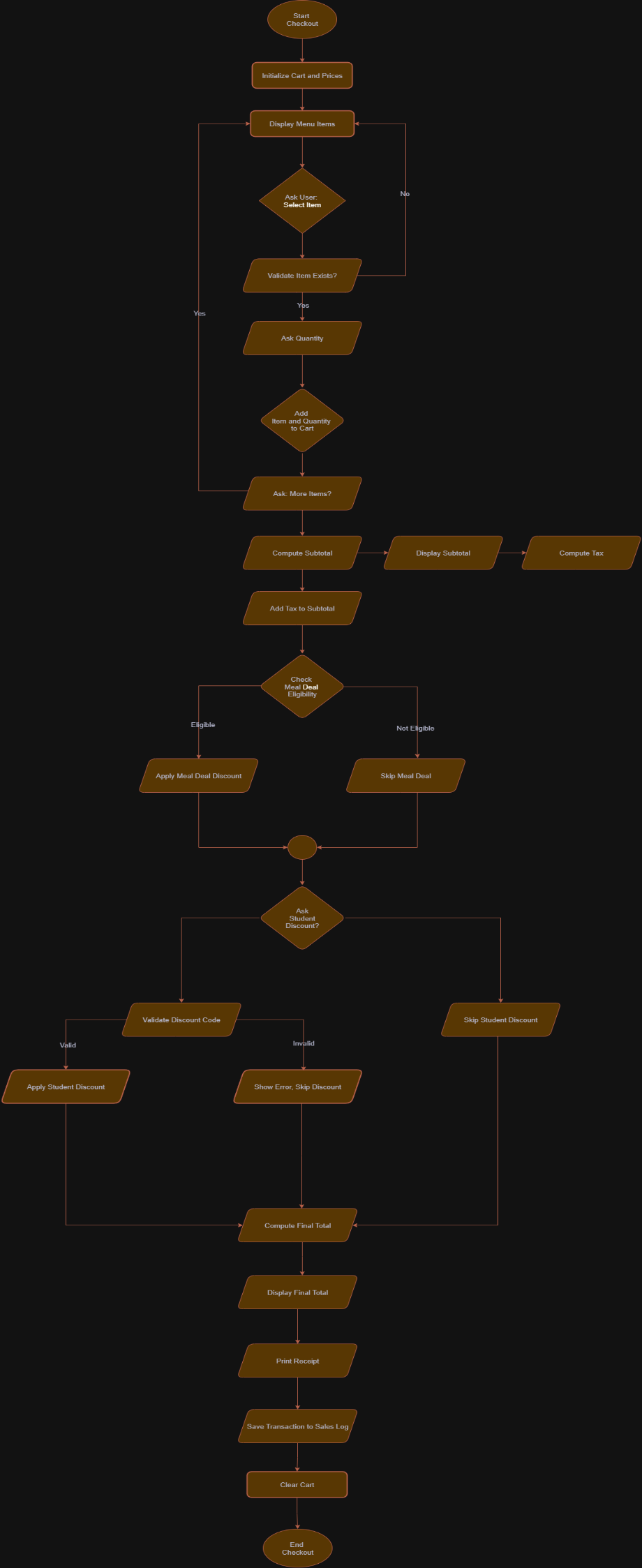
print("Exiting checkout. Thank you!")

break

else:

print("Invalid choice")

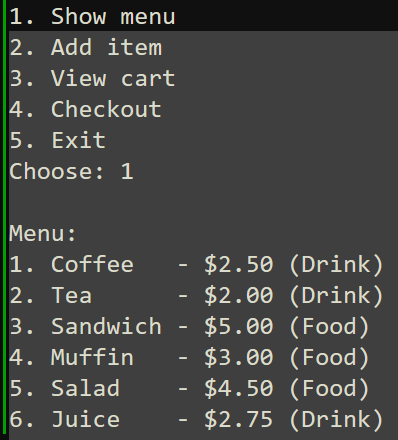
**Flowchart:**



**Step 6:- Testing Code**

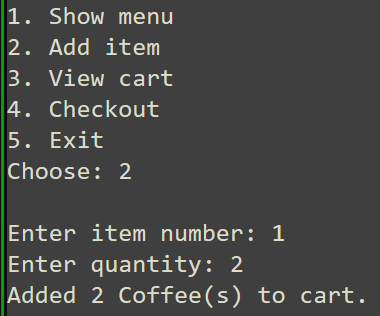
1. **Test Case 1: Show Menu**

* Input: User selects option 1 (Show Menu).
* Processing: The system should iterate through the menu dictionary and print all items with their prices and categories.
* Expected Output: A formatted list of available items (e.g., “Coffee – $2.50, Sandwich – $4.00, Tea – $1.50, Juice – $3.00”).
* Purpose: Verifies that menu data is displayed correctly.



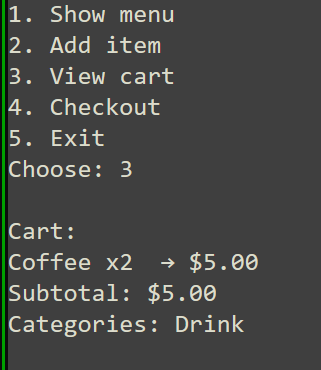
1. **Test Case 2: Add Item to Cart**

* Input: User selects option 2, enters Coffee as the item, and chooses quantity 1.
* Processing: The program validates that “Coffee” exists, then appends (Coffee, 1) to the cart.
* Expected Output: Message → “1 Coffee added to cart.”
* Purpose: Confirms that adding items updates the cart correctly.



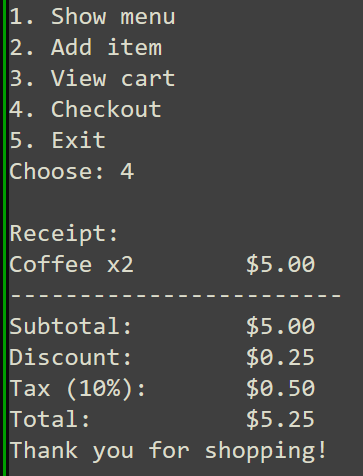
1. **Test Case 3: View Cart**

* Input: User selects option 3.
* Processing: The program iterates through the cart list and computes the subtotal.
* Expected Output: List of all items in cart with subtotal (e.g., “Cart: Coffee ×1, Subtotal = $2.50”).
* Purpose: Ensures cart contents and subtotal calculation are correct.



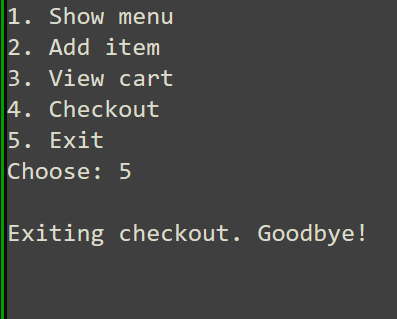
1. **Test Case 4: Checkout with Student Discount**

* Input: Add Coffee ($2.50), then select Checkout (4), and answer “yes” for student discount.
* Processing:
* Subtotal = $2.50
* Tax = $2.50 × 10% = $0.25
* Student Discount = 5% of ($2.50 + $0.25) = $0.14
* Final Total = $2.61



1. **Test Case 5: Invalid Input Handling**

* Input: User selects option 9.
* Processing: The system checks against valid menu options and rejects invalid entry.
* Expected Output: “Invalid choice. Please try again.”
* Purpose: Verifies robust error handling for invalid inputs.



**Step 7:- Refinement via GenAI:**

After initial implementation, the program was refined with the help of Generative AI (GenAI) to improve readability, user experience, and code quality.

**Prompts I've Used with GenAI:**

* “Check if my pseudocode is correct or can be simplified for clarity.”
* “Suggest a better way to structure test cases in a template.”
* “How can I make the receipt format clearer and professional?”
* “How do I avoid rounding issues in tax and discount calculations?”

**What I've Changed After Refinement:**

* Pseudocode Improvements – Long steps were broken down into concise structured blocks, making it easier to follow. Loops and conditions were rewritten in a cleaner format.
* Testing Framework – A consistent test-case template was introduced (Input → Processing → Expected Output → Purpose). This made testing structured and repeatable.
* Receipt Formatting – Instead of plain text, the receipt output was reformatted with aligned columns (item name, price, quantity, subtotal), which looks more like a professional bill.
* Discount & Rounding Handling – Suggested using round(value, 2) for monetary calculations, ensuring totals are always shown correctly to two decimal places.
* User Experience Enhancements – Clearer prompts (“Enter quantity for Coffee:”) and confirmation messages (“Your cart has been updated”) were added to improve usability.

**Justification for Changes:**

* The refinements made the program easier to understand (clean pseudocode).
* The testing became standardized and rigorous.
* The receipt output looked professional and user-friendly.
* The program became more robust against errors and rounding inconsistencies.