**Walmart Shopping Budget Simulation - Documentation**

**Objective**

This Python program simulates a shopping experience at Walmart with a fixed budget of $100. It ensures users do not exceed their budget, accounts for tax, and provides a receipt upon checkout.

**Group Member Contributions**

* **Chilawo Munene (G00450721):** Implemented user input handling and validation to ensure proper data entry.
* **Grace Ngoma (G00450740):** Developed budget constraint logic and the calculate\_tax() function to apply tax correctly.
* **Shalom Donga (G00450733):** Created the cart display and final checkout process, including the receipt format.
* **Dingani Freddie Kandiwo (G00450731):** Tested the program for edge cases, fixed bugs, and documented the program logic.

**Features and Functionalities**

1. **User Input Handling**
   * Users can enter item names and prices interactively.
   * Input validation ensures prices are valid and greater than zero.
   * Users can remove items from their cart before completing checkout.
2. **Budget Constraint**
   * Users can add items until the total (including tax) reaches or exceeds $100.
   * If an item exceeds the remaining budget, a warning is displayed.
   * Users can either checkout or continue shopping within their remaining balance.
3. **Tax Calculation**
   * A function calculates tax based on a 10.44% tax rate.
   * Tax is applied to each item before verifying the total budget constraint.
4. **Cart Management**
   * Items and their prices are stored in lists.
   * The program displays the current cart, including:
     + Item names and prices
     + Subtotal before tax
     + Tax amount
     + Total cost after tax
     + Remaining balance
5. **Checkout Process**
   * Users can choose to complete checkout, remove items, or cancel the transaction.
   * A final receipt is displayed upon successful checkout.
6. **Error Handling**
   * Ensures numeric input for prices.
   * Prevents negative values or empty item names.
   * Handles invalid user choices gracefully.

**Functions**

1. **calculate\_tax(price)**
   * Computes tax at a rate of 10.44% for a given item price.
2. **checkout\_process(items, prices, max\_total)**
   * Displays the cart summary.
   * Allows users to proceed to checkout, remove items, or cancel the transaction.
3. **main()**
   * Handles user input for adding items.
   * Manages the budget constraint.
   * Calls checkout\_process() when the user chooses to checkout.

**User Interaction**

* Users enter item names and prices.
* They can review their cart at any time.
* If the budget is exceeded, the program prevents further additions.
* Users can finalize the purchase, remove items, or cancel the transaction.

***Group member 3 enlisted AI to help in the formatting of invoice-like structure. Documentation of features learned and used:***

1. "{:^40}".format("SHOPPING CART")

^ means center-align the text

40 means use 40 characters of width total

This centers "SHOPPING CART" in a 40-character wide space

1. "{:<25} {:>12}".format("Item", "Price ($)")

:<25 means left-align and use 25 characters for the first value ("Item")

:>12 means right-align and use 12 characters for the second value ("Price ($)")

Creates column headers with "Item" on the left and "Price ($)" on the right

1. for i, (item, price) in enumerate(zip(items, prices), 1):

zip() pairs up elements from the items and prices lists together -> ("apple", 1.99)

, 1 tells it to start counting from 1 instead of 0 -> 1: ("apple", 1.99), 2: ("banana", 0.99)

(item, price) unpacks each zip pair into separate variables

1. f"{i}. {item:<22} ${price:>9.2f}"

{i}. prints the item number

{item:<22} left-aligns the item name in 22 spaces

${price:>9.2f} right-aligns the price, using 9 spaces, with 2 decimal places

.2f means format as float with 2 decimal places

1. "{:<25} ${:>9.2f}".format("Subtotal:", total\_before\_tax)

Left-aligns "Subtotal:" in 25 spaces

Right-aligns the price with dollar sign in 9 spaces, 2 decimal places

1. For the remove item section:

Gets user input for item number

Subtracts 1 to convert from user-friendly 1-based indexing to 0-based list indexing

Checks if index is valid (between 0 and list length)

Uses pop() to remove and return the item at that index