Intro to Programming

**Assessment 2: Utility App**

|  |  |
| --- | --- |
| **Student’s Name** | Felix Ian Jasper Saupan |
| **Roll No** | 04-24-173 |
| **Github Repository Name** | Assessment-2 |
| **Github Repository Link** | <https://github.com/flxnjspr/Assessment-2>  Github username: flxnjspr |
| **Repository Screen Shot** |  |

**The Development Document**

Specification

We were tasked to make a vending machine program that has a menu of drinks and snacks presented via the console, a set of numbers or codes that the user can input to select a particular drink or snack, a way of managing money so the user can input any amount of money and have the correct change returned, a message that tells the user that a particular drink or snack has been dispensed, a message that tells the user how much change they have received, a way of allowing users to buy additional items, and appropriate error checking to validate inputs and ensure the user has enough money.

System Flowchart

DISPLAY MENU

START

ENTER PRODUCT CODE

IS PRODUCT CODE VALID??

END

GIVE CHANGE

NO

YES

PAYMENT ENOUGH?

INSERT MONEY

SHOW CART

NO

YES

MORE ITEMS?

ADD ITEM & DEDUCT

YES

UNAVAILABLE MESSAGE

NO

IS ITEM IN STOCK?

ERROR MESSAGE

NO

Technical Description & Walkthrough

**Vending Machine Program Walkthrough**

**Overview**

This Python program simulates a vending machine where users can select products, pay, and receive change. It also includes a free bottled water promotion for purchases of three or more items. The system is designed to handle inventory tracking, payment processing, and change calculation efficiently.

**Key Components**

1. **Product List (menu)**

-A dictionary stores product details, including the item code, name, price, and stock quantity.

-The stock reduces when an item is purchased.

1. **Display Menu (showMenu())**

-Presents a structured table showing available products, prices, and stock levels.

-Ensures users can make informed purchasing decisions.

1. **Payment (payment(total\_price))**

-Ensures users insert enough money to complete their purchase.

-Provides continuous feedback on the remaining amount needed.

-Validates input to prevent incorrect values (e.g., negative amounts or non-numeric inputs).

1. **Change Calculation (calculateChange(change))**

-Uses common currency denominations (1.00, 0.25, 0.10, 0.05, 0.01 AED).

-Ensures change is broken down into the smallest number of coins/bills.

-Handles rounding errors to prevent incorrect calculations.

1. **Free Drink Promotion (addFreeDrink())**

-If a customer purchases three or more items, they receive a free bottled water.

-If water is out of stock, the program notifies the customer.

**How It Works (main())**

1. **User Selects Products**

-The menu is displayed, showing available items.

-The user enters a product code corresponding to the desired item.

-The program checks if the item exists and is in stock.

-If valid, the item is added to the cart, stock is updated, and the total price increases.

-If the item is out of stock or the code is invalid, an error message appears, and the user is prompted to try again.

1. **Checkout Process**

-Once the user finishes selecting items, the total cost is displayed.

-The payment function prompts the user to insert money.

-The system ensures enough money is provided before proceeding.

-If excess money is inserted, change is calculated and displayed.

1. **Item Dispensing**

-After successful payment, the purchased items are dispensed.

-If eligible, the user receives a free bottled water.

-A message is displayed confirming the transaction and listing the dispensed items.

**Error Handling and User Guidance**

* **Invalid Inputs:** If a user enters an incorrect product code or an invalid amount of money, the system prompts them to try again.
* **Out of Stock Items:** Users are informed if their selected item is unavailable.
* **Incorrect Payment Inputs:** If a non-numeric or negative value is entered, the program requests a valid input.

**Example Scenario**

1. The user starts the vending machine, and the menu is displayed.
2. They enter C1 to buy Chips Ahoy, which costs AED 1.50.
3. They choose K1 (Kit Kat) and R1 (Reese’s), bringing the total to AED 3.75.
4. Since they purchased three items, they receive a free bottled water.
5. They insert AED 5.00.
6. The machine calculates the change (AED 1.25) and dispenses it.
7. The program lists the purchased items, change breakdown, and thanks the user.

**Conclusion**

This vending machine program efficiently manages product selection, payment handling, and change calculation. It ensures a smooth user experience with clear instructions and error handling. The free drink promotion adds an incentive for larger purchases. The modular design allows easy modification and expansion, making it adaptable for real-world applications.

Critical Reflection

One thing I really like about my vending machine program is how organized and easy to use it is. The menu layout is clear, listing all the items, their prices, and stock levels in a way that makes it simple to navigate. I also think the payment system works well because it ensures the user inserts enough money before finalizing their purchase. The way the program calculates and breaks down change into different denominations is another feature I’m proud of, as it makes the transaction feel more realistic. Plus, I really like the free drink reward system—it adds a fun incentive for people to buy more items, making the vending machine experience more engaging.

That said, there are things I want to improve. One issue is how user input is handled. Right now, if someone enters something unexpected—like typing something other than "yes" or "no" when asked if they want another item—the program doesn’t handle it properly. I also think the payment system could be more flexible. Instead of requiring users to insert money step by step, it would be more realistic to allow them to enter the full amount at once and receive the correct change. Another improvement would be transitioning from a text-based interface to a graphical user interface (GUI), which would make the program more visually appealing and user-friendly. Additionally, adding a feature to restock items and track total earnings over time would make the vending machine more dynamic and practical.

Appendix

#VENDING MACHINE V5  
  
# Creating a dictionary for the items  
menu = {  
 "C1": {"name": "Chips Ahoy", "price": 1.50, "stock": 5},  
 "R1": {"name": "Reese's", "price": 1.25, "stock": 5},  
 "N1": {"name": "Nutella Biscuits", "price": 0.75, "stock": 5},  
 "K1": {"name": "Kit Kat", "price": 1.00, "stock": 5},  
 "S1": {"name": "Snickers", "price": 1.50, "stock": 5},  
 "M1": {"name": "M&M's", "price": 2.00, "stock": 5},  
 "W1": {"name": "Water", "price": 1.00, "stock": 5},  
 "L1": {"name": "Lay's", "price": 9.65, "stock": 5},  
 "H1": {"name": "Hershey", "price": 1.50, "stock": 5},  
 "P1": {"name": "Pringles", "price": 7.25, "stock": 5},  
}  
  
def showMenu():  
 print("\n||=======================================================||")  
 print("|| Welcome to Felix's Vending Machine! ||")  
 print("||=======================================================||")  
 print("|| Code | Item | Price (AED) | Stock ||")  
 print("||======|======================|=============|===========||")  
 for code, item in menu.items():  
 print(f"|| {code:<4} | {item['name']:<20} | {item['price']:<11.2f} | {item['stock']:<9} ||")  
 print("||=======================================================||")  
  
def payment(total\_price):  
 total\_inserted = 0  
 while total\_inserted < total\_price:  
 try:  
 remaining = total\_price - total\_inserted  
 print(f"\nThe total cost is AED {remaining:.2f}.")  
 amount = float(input("Insert money (AED): "))  
 if amount <= 0:  
 print("Please insert a valid amount.")  
 else:  
 total\_inserted += amount  
 except ValueError:  
 print("Invalid input. Please enter a numeric value.")  
 return total\_inserted  
  
def calculateChange(change):  
 denominations = [1.00, 0.25, 0.10, 0.05, 0.01]  
 change\_breakdown = {}  
 for denom in denominations:  
 count = int(change // denom)  
 if count > 0:  
 change\_breakdown[denom] = count  
 change -= count \* denom  
 change = round(change, 2)  
 return change\_breakdown  
  
def addFreeDrink():  
 drink\_code = "W1"  
 if menu[drink\_code]["stock"] > 0:  
 menu[drink\_code]["stock"] -= 1  
 print("\nCongratulations! You have earned a free drink (Bottled Water) because you purchased 3 items. It has been dispensed.")  
 else:  
 print("\nWe're sorry. You would have earned a free drink, but bottle water is out of stock.")  
  
def main():  
 total\_spent = 0.0  
 purchase\_count = 0  
 cart = []  
 total\_price = 0.0  
  
 while True:  
 showMenu()  
 choice = input("\nEnter the code of the product you want to purchase: ").upper()  
  
 if choice in menu:  
 item = menu[choice]  
 if item['stock'] > 0:  
 print(f"\nYou have selected {item['name']} for AED {item['price']:.2f}.")  
 cart.append((item['name'], item['price']))  
 total\_price += item['price']  
 item['stock'] -= 1  
 purchase\_count += 1  
  
 another = input("\nDo you want another product? (yes/no): ").strip().lower()  
 if another != 'yes':  
 print("\nYour selected items:")  
 for item\_name, price in cart:  
 print(f" - {item\_name}: AED {price:.2f}")  
  
 print(f"\nThe total amount to pay is AED {total\_price:.2f}.")  
 total\_inserted = payment(total\_price)  
 change = total\_inserted - total\_price  
  
 print(f"\nThank you for your purchases! Your change is AED {change:.2f}.")  
 change\_breakdown = calculateChange(change)  
 if change\_breakdown:  
 print("\nYour change breakdown:")  
 for denom, count in change\_breakdown.items():  
 print(f"AED {denom:.2f}: {count} coins/bills")  
  
 print("\nItems dispensed:")  
 for item\_name, \_ in cart:  
 print(f" - {item\_name}")  
  
  
 if purchase\_count >= 3:  
 addFreeDrink()  
  
 print(f"\nThank you for using Felix's vending machine! You spent a total of AED {total\_price:.2f}.")  
 break  
 else:  
 print("Sorry, those are not in the options. Please enter yes or no")  
 else:  
 print(f"\nSorry, {item['name']} is out of stock.")  
 else:  
 print("\nInvalid product code. Please try again.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()