CODELAB I

ASSESSMENT 2: Utility App

Tutor : Ms. Lavanya Mohan

Programming Fundamentals

|  |  |
| --- | --- |
| **Student’s Name:** | Frem Fredric Baring Patalinghug |
| **Student ID** | 04-24-184 |
| **Github Repository Name:** | <https://github.com/Eldeston/Creative-Computing> |
| **Github Repository Link:** | <https://github.com/Eldeston/Creative-Computing/tree/main/Year%201/03%20-%20Intro%20to%20Programming/Assessments/Assessment%2002%20-%20Intro%20to%20Python> |
| **YouTube Link** | <https://youtu.be/KlgqMd71-FI> |

**Brief**

The task is to create a vending machine program using Python. This program will demonstrate my skills and knowledge in programming and Python itself. In this documentation, I will also explain each part of the code and why I chose this pipeline for my program.

**Specifications**

The following is the basic structure of the the vending machine:

- Booting animation

- User sign in

- Main system

- User interface

- Deposit cash

- Purchase items

- View receipt

- Admin access (Secret trigger)

- Enter password

- Admin interface

- Withdraw profits

- Add stock

- Remove stock

- Change password (Secret trigger)

- Check logs

- Exit admin interface

- Exit user interface

- Exit main system

The code also contains a basic logging system to list down all recent actions within the machine and each user interaction contains exception handling. The code also contains aliases for each available options for faster user input. The code also includes display animations that utilize time delays and console clearing. Finally, there are accessible debugging features that disables or adjusts these display animations and console clearing but can only be accessed within the code.

The vending machine has gone over 4 major revisions and follows strict code formatting rules. This is important to make code readable and easy to manage. The following are the rules for the code format:

- Use camelCase for all variables, and functions.

This is valid: malevolentKitchen

This is invalid: DOMAIN\_EXPANSION\_INFINITE\_VOID

- All operations must always have a space before and after.

This is valid: numeroUno + despacito

This is invalid: numeroUno-=despacito

- Parentheses, quotations, square brackets, and curly brackets should have no space next to it inside its contents.

This is valid: (neverGonna + giveYouUp) + (neverGonna + letYouDown)

This is invalid: [ neverGonna+runAround ] + [ andDesert+you ]

- Functions should have no space between its parentheses when called.

This is valid: turnThisIntoEven(Even)

This is invalid: turnThisIntoEven (Even)

- When defining a function or using control flow, colons should be separated with a space.

This is valid: while True : […]

This is invalid: if booleanCondition:[…]

- When making a single line comment, always start with “#” and a space to separate it from its contents

This is valid: # Nah, I’d win

This is invalid: #GOJO NOOOOOO

- When creating a single line list, tuple, or dictionary, comments should have one space after.

This is valid: [Yuji, Megumi, Nobara]

This is invalid: [Gojo ,Sukuna ,Yuta ]

- When creating a multi line list, tuple, or dictionary, indent its contents.

This is valid:

{

“Dialogue0” : “When is this gonna end?”,

“Dialogue1” : “Jesse, what the heck are you talking about?”

}

This is invalid:

[

“EMOTIONAL DAMAGE”,

“I\’VE PLAYED THESE GAMES BEFORRRREEE”

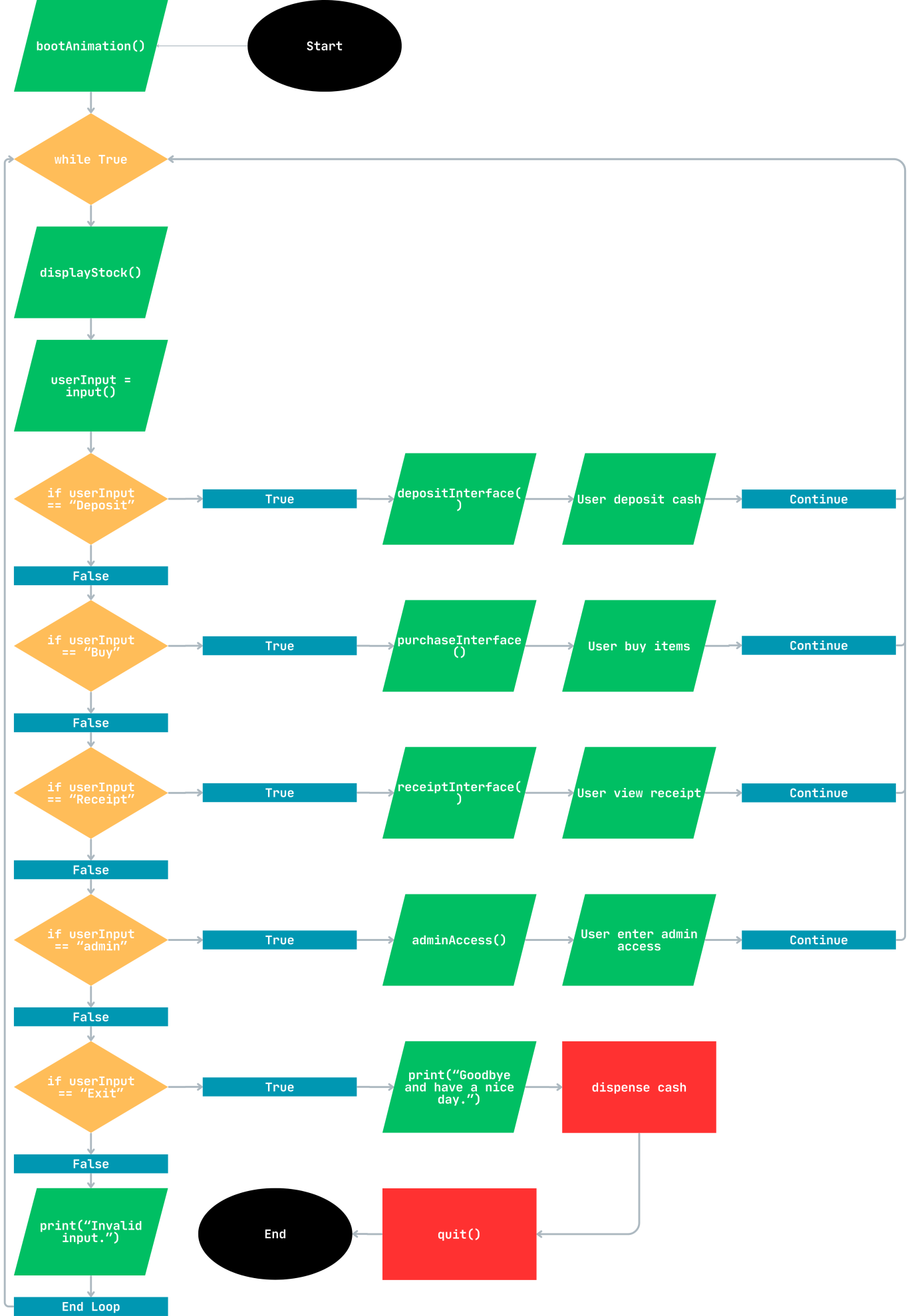
]

**System Flowchart**

The following is the flowchart of the core of the vending machine pipeline. The flowchart strictly follows a step-by-step format mimicing Python’s interpeter behavior. The flowchart is made with Canva’s whiteboard with color coded shapes and JetBrains font.

Canva Whiteboard Link:

<https://www.canva.com/design/DAGaSvH3r18/FCWgXUt6WXjcpKzmcD-Q0A/edit?utm_content=DAGaSvH3r18&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>



**Technical Description**

To explain the technicalities of my long vending machine, we need to look at it in it's basic form.

When you play an RPG game, what is the first thing you see when you launch the game? Or the first thing you see when you open a shop menu in-game? Or the first thing you see after booting up your BIOS menu? The first thing you will always see among these 3 things is a menu or a user interface. The following is the basic structure of my vending machine.

User Interface (Menu) :

Purchase (A separate interface for buying is found here)

Exit (Exit the program)

The structure contains 3 features: user interface, purchase, and exit. The user interface displays the options purchase and exit to the user and asks for user input to select between those options: "Purchase" for buying an item and "Exit" for exiting the program. So in code it would look like this:

# Run forever until break or exit

while True :

   # Ask for user input each loop

   userInput = input("\"Purchase\" to purchase coffee or \"Exit\" to exit program.")

   # If user purchases an item...

   if userInput == "Purchase" :

      purchaseInterface()

      print("Bought 1 can of coffee")

continue

   # If user exits program...

   if userInput == "Exit"

      print("Goodbye and have a nice day.")

      exit()

   # If all checks are passed, announce invalid input

   print("Invalid input.")

The pipeline is comes in line with the flowchart too. This allows for the developer to add as many options as they want with great flexibility and convenience just by adding another check statement and a function and using only one user input. The line at the end of the loop is then dedicated for exception handling, and when exiting the interface, a return or exit() can be used since it also acts as a break but can exit the function or exit the program altogether respectively.

This basic code is only scratching the surface of the actual code but this captures how the vending machine pipeline looks like in a bigger picture. The pipeline takes advantage of Python’s interpreter behavior by using a loop. Then for each loop the program asks for a user input. This input then determines how the code will flow from that point by using if statements to branch off the main loop. Then after the instructions are executed in that if statement, reset the loop to ask the user again for an input. This incredibly flexible design is used throughout the code in functions which is why there are 2 main interfaces: the user interface and the admin interface.

Both the user interface and admin interface uses a while loop to create the interface. Both also have basic display functions to display the items in stock and other stats. Both have a selection of options that can be accessed by typing any of the selections or by using its aliases. All these options are made visible to the user except admin access to prevent machine tampering. Additionally, admin access requires a password to proceed to the admin interface.

Lastly, I implemented some fancy text animations by utilizing time delays and string manipulation. The time delay formula to calculate when an animation ends are as follows:

The variable “timeStep” is needed to loop through the string with a specified time delay, “duration” determines the amount of time the animation takes, and “iterations” is the length of the string or prompt which also determines the amount of iterations the loop needs to go through. The function looks like this:

# Text animation function

def textAnimation(prompt = "Loading...", duration = shortDuration) :

    # Find the number of iterations based on the length of the string

    iterations = len(prompt)

    # This makes sure the loading bar will stop at a specified duration

    # timeStep / timeLength = 1 / iterations

    # timeStep = timeLength / iterations

timeStep = duration / iterations

    # Run loop through set iterations

    for iterations in range(iterations) :

        # Loop through the prompt and iterate

        print(prompt[iterations], end = "")

        # Wait until specified duration

        time.sleep(timeStep)

**Output**

\_\_ \_\_\_ \_ \_\_\_\_ \_\_ \_\_\_\_

/ |/ /\_\_\_ \_(\_)\_\_\_ / \_/\_\_\_ / /\_\_\_\_ \_\_\_\_\_/ \_\_/\_\_\_ \_\_\_\_\_\_\_\_\_

/ /|\_/ / \_\_ `/ / \_\_ \ / // \_\_ \/ \_\_/ \_ \/ \_\_\_/ /\_/ \_\_ `/ \_\_\_/ \_ \

/ / / / /\_/ / / / / / \_/ // / / / /\_/ \_\_/ / / \_\_/ /\_/ / /\_\_/ \_\_/

/\_/ /\_/\\_\_,\_/\_/\_/ /\_/ /\_\_\_/\_/ /\_/\\_\_/\\_\_\_/\_/ /\_/ \\_\_,\_/\\_\_\_/\\_\_\_/

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

Current cash: AED 100.0

Receipt history: 1

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

ID: 0, Name: Coca Cola, Stock: 16, Price: 3.0

ID: 1, Name: Pepsi, Stock: 16, Price: 3.0

ID: 2, Name: Miranda, Stock: 16, Price: 3.0

ID: 3, Name: Fanta, Stock: 16, Price: 3.0

ID: 4, Name: Mountain Dew, Stock: 16, Price: 3.0

ID: 5, Name: Water, Stock: 16, Price: 1.0

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

Type "Deposit/D" to deposit cash, "Purchase/P" to purchase items, "Receipt/R" to open receipt, or "Exit/E" to exit machine.

P

Purchasing items...

Enter item ID: 0

Enter item amount: 10

Dispensed 10 Coca Cola for AED 30.0 with AED 70.0 left in deposit.

Type "Continue" to continue making purchases or press enter key to exit interface...

**Critical Reflection**

Making this vending machine helped me learn more about Python programming. Prior to this assessment, I actually already have experience with programming with other languages which made programming with Python a lot easier. This gave me a huge advantage to completing this assessment.

However, there are things I have learned from this assessment such as the importance of functions and code formatting. Functions make code readable as well as making code easier to manage and reusing common functions especially on a larger scale. I believe code formatting is important in all areas of writing code to make code readable and easy to follow for the developer and other developers. Commenting on each line of code helps with code formatting as it helps further with readability by grouping related code and dividing them into easily readable chunks.

I also learned how to use Python’s inbuilt libraries by importing. These libraries can be accessed anywhere so long as the library is available in that Python version. I also learned how to import my own libraries. The difference is that it needs to be created in a local directory.

Overall, I believe this assessment helped me re-enforce a new understanding of the principals of coding that can be applicable to other programming languages regardless of the program’s syntax. I also had great fun programming the vending machine and tested my problem solving skills and added so many features and micro features. I also developed a strong discipline when following my own code formatting rules.

**Appendix**

# Import os, used to clear console

import os

# Import time, used to time animations and utilize delays

import time

# -------------------------------- # Debugging Settings # -------------------------------- #

# Display console

displayConsole = False

# Define common durations

# This determines the length of animations

longDuration = 8

normalDuration = 4

shortDuration = 2

# Declare user cash

userCash = 0.0

# Declare store cash

machineProfit = 0.0

# Declare authors for logging

user = "User"

admin = "Admin"

system = "System"

# Declare password (case sensitive)

password = "nahIdWin"

# Declare user receipt

userReceipt = []

# Declare receipt history

systemLog = []

# Declare item stock as a list

# Each list contains a dictionary defining the properties of the item's name, stock, and price

itemStock = [

    {

        "Name" : "Coca Cola",

        "Stock" : 16,

        "Price" : 3.0

    },

    {

        "Name" : "Pepsi",

        "Stock" : 16,

        "Price" : 3.0

    },

    {

        "Name" : "Miranda",

        "Stock" : 16,

        "Price" : 3.0

    },

    {

        "Name" : "Fanta",

        "Stock" : 16,

        "Price" : 3.0

    },

    {

        "Name" : "Mountain Dew",

        "Stock" : 16,

        "Price" : 3.0

    },

    {

        "Name" : "Water",

        "Stock" : 16,

        "Price" : 1.0

    }

]

# -------------------------------- # Display Functions # -------------------------------- #

# Clear console function

def clearConsole() :

    # If display console is enabled

if displayConsole : return

    # Check if the OS is Linux and use "clear" instead

os.system("cls" if os.name == "nt" else "clear")

# Text animation function

def textAnimation(prompt = "Loading...", duration = shortDuration) :

    # Find the number of iterations based on the length of the string

iterations = len(prompt)

    # This makes sure the loading bar will stop at a specified duration

    # timeStep / timeLength = 1 / iterations

    # timeStep = timeLength / iterations

timeStep = duration / iterations

    # Run loop through set iterations

    for iterations in range(iterations) :

        # Loop through the prompt and iterate

        print(prompt[iterations], end = "")

        # Wait until specified duration

        time.sleep(timeStep)

# Loading animation function

def loadingAnimation(prompt = "Loading...", duration = shortDuration, iterations = 64) :

    # Announce prompt

    print(prompt)

    # Reuse text animation as a loading bar because I'm lazy lol

textAnimation("████████████████████████████████████████████████████████████████", duration)

# -------------------------------- # Common Functions # -------------------------------- #

# Input to integer function, default input argument set to ""

def inputToInteger(prompt = "") :

    # Run under an infnite loop until return to exit function

    while True :

        # Input prompt

        userInput = input(prompt)

        # Checks if the input is a digit

        if userInput.isdigit() :

            # Convert string to integer and return output

            return int(userInput)

        # Announce incorrect input

        print("\nInvalid input.")

# Input to float function, default input argument set to ""

def inputToFloat(prompt = "") :

    # Run under an infnite loop until return to exit function

    while True :

        # Input prompt

        userInput = input(prompt)

        # Removes decimals once to check if the string is a digit

        if userInput.replace('.', '', 1).isdigit() :

            # Convert string to float and return output

            return float(userInput)

        # Announce incorrect input

        print("\nInvalid input.")

# Converts input to lower case

def inputLower(prompt = "") :

return input(prompt).lower()

# -------------------------------- # Debug Functions # -------------------------------- #

# Log system action function

def logAction(author = system, log = "Logged action") :

    # Access systemLog as a global variable

global systemLog

    # Log action as list to system log

systemLog += [author + ": " + log]

# Display stock function

def displayStock() :

    # Display top header

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

    # Display current stock using a loop

    for index, properties in enumerate(itemStock) :

        # Display item properties

        print(f"ID: {index}, Name: {properties["Name"]}, Stock: {properties["Stock"]}, Price: {properties["Price"]}")

        # Wait for a set amount of time

        time.sleep(0.0625)

    # Display bottom header

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

# -------------------------------- # Admin Functions # -------------------------------- #

# Withdraw profit function

def withdrawInterface() :

    # Access machineProfit as a global variable

global machineProfit

    # Log action as Admin

logAction(admin, "Withdrawing profits.")

    # Announce withdrawal

loadingAnimation(f"\nWithdrawing AED {machineProfit}...")

    # Set profit to 0.0 (because obviously this is just a program and we can't really emulate ejecting cash)

machineProfit = 0.0

    # Announce stock successfuly added

print("\nProfits successfully withdrawn.")

# Add stock interface

def addInterface() :

    # Access itemStock as a global variable

global itemStock

    # Log action as Admin

logAction(admin, "Adding stock.")

    # Enter item name

userItemName = input("\nEnter item name: ")

    # Enter item stock

userItemStock = inputToInteger("\nEnter item stock: ")

    # Enter item price

userItemPrice = inputToFloat("\nEnter item price: ")

    # Add new stock

    itemStock += [

        {

            "Name" : userItemName,

            "Stock" : userItemStock,

            "Price" : userItemPrice

        }

]

    # Announce stock successfuly added

print("\nStock successfuly added.")

# Remove stock

def removeInterface() :

    # Access itemStock as a global variable

global itemStock

    # Log action as Admin

logAction(admin, "Removing stock.")

    # Enter item index

itemIndex = inputToInteger("\nEnter item stock: ")

    # Remove item

del itemStock[itemIndex]

    # Announce stock successfuly added

print("\nStock successfuly removed.")

# Password interface function

def passwordInterface() :

    # Access password as a global variable

global password

    # Log action as Admin

logAction(admin, "Changing password.")

    # Enter old password

userInput = input("\nEnter old password (case sensitive): ")

    # If password is incorrect

    if not userInput == password :

        # Announce incorrect password

        print("\nIncorrect password.")

        # Exit function

        return

    # Enter new password

password = input("\nEnter new password (case sensitive): ")

    # Announce password successfuly changes

print("\nPassword successfuly changed.")

# Logs interface function

def logsInterface() :

    # Access systemLog as a global variable

global systemLog

    # Log action as Admin

logAction(admin, "Viewing history.")

    # Get systemLog length

logLength = len(systemLog)

    # Check if systemLog is empty

    if logLength == 0 :

        # Announce systemLog is empty

        print("\nLog is empty.")

        # Exit function

        return

    # Print new line for interface formatting sake,

    # print() by default prints a newline if the parameters are empty

print()

    # This makes sure the loading bar will stop at a specified duration

    # timeStep / timeLength = 1 / iterations

    # timeStep = timeLength / iterations

timeStep = shortDuration / logLength

    # Show systemLog in a loop

    for index in range(logLength) :

        print(systemLog[index])

        # Wait until specified duration

        time.sleep(timeStep)

    # Ask user to enter to continue

    textAnimation("\nPress enter key to continue...")

    # This is for responding to prompt

input()

# -------------------------------- # Admin Interface # -------------------------------- #

# Admin interface function

def adminInterface() :

    # Access machineProfit as a global variable

global machineProfit

    # Access systemLog as a global variable

global systemLog

    # Log action as Admin

logAction(admin, "Interface accessed.")

    # Run under an infnite loop until interface exit

    while True :

        # Clear current console

        clearConsole()

        # Display admin interface

        print(

            f"""

    \_\_\_       \_\_          \_          \_\_\_\_      \_\_            \_\_\_\_

   /   | \_\_\_\_/ /\_\_\_ \_\_\_  (\_)\_\_\_     /  \_/\_\_\_  / /\_\_\_\_  \_\_\_\_\_/ \_\_/\_\_\_ \_\_\_\_\_\_\_\_\_

  / /| |/ \_\_  / \_\_ `\_\_ \/ / \_\_ \    / // \_\_ \/ \_\_/ \_ \/ \_\_\_/ /\_/ \_\_ `/ \_\_\_/ \_ \\

 / \_\_\_ / /\_/ / / / / / / / / / /  \_/ // / / / /\_/  \_\_/ /  / \_\_/ /\_/ / /\_\_/  \_\_/

/\_/  |\_\\_\_,\_/\_/ /\_/ /\_/\_/\_/ /\_/  /\_\_\_/\_/ /\_/\\_\_/\\_\_\_/\_/  /\_/  \\_\_,\_/\\_\_\_/\\_\_\_/

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

Machine profits: AED {machineProfit}

Machine history: {len(systemLog)}

            """

            )

        # Display stock

        displayStock()

        # Display available user options and ask user for input

        textAnimation("\nType \"Withdraw/W\" to withdraw profits, \"Add/A\" to add new stock, \"Remove/R\" to remove stock, \"Logs/L\" to view logs, or \"Exit/E\" to exit interface.\n")

        # Set output string to all lowercase allowing for all selections will be case insensitive

        userInput = inputLower()

        # Check if user types "withdraw"

        if userInput in ("withdraw", "w") :

            # Announce withdrawing profits

            print("\nWithdrawing profits...")

            # Enter withdraw interface

            withdrawInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            continue

        # Check if user types "add"

        if userInput in ("add", "a") :

            # Announce adding stock

            print("\nAdding stock...")

            # Enter add interface

            addInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            continue

        # Check if user types "remove"

        if userInput in ("remove", "r") :

            # Announce removing stock

            print("\nRemoving stock...")

            # Enter remove interface

            removeInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            continue

        # Check if user types "password"

        if userInput in ("password", "p") :

            # Announce changing password

            print("\nChanging password...")

            # Enter password interface

            passwordInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            continue

        # Check if user types "history"

        if userInput in ("logs", "l") :

            # Announce viewing history

            print("\nViewing logs...")

            # Enter history interface

            logsInterface()

            # Reset loop

            continue

        # Check if user types "exit"

        if userInput in ("exit", "e") :

            # Announce exiting interface

            print("\nExiting interface...")

            # Exit function

            return

# Admin access function

def adminAccess() :

    # Case sensitive input

userInput = input("\nEnter password (case sensitive): ")

    # Check if password is correct

    if userInput == password :

        # Announce access grants

        print("\nAccess granted.")

        # Enter admin interface

        adminInterface()

        # Log action as Admin

        logAction(admin, "Exiting interface.")

        # Exit function

        return

    # Announce invalid password and exit

print("\nAccess denied.")

# -------------------------------- # Vending Machine Functions # -------------------------------- #

# Deposit interface function

def depositInterface() :

    # Access user as a global variable

global user

    # Access userCash as a global variable

global userCash

    # Access userReceipt as a global variable

global userReceipt

    # Log action as User

logAction(user, "Depositing cash.")

    # Ask user cash input

userInput = inputToFloat("\nEnter cash: ")

    # Insert cash

userCash += userInput

    # Make a recent receipt

    recentReceipt = f"Deposited AED {userInput}."

    # Record recent receipt as list

userReceipt += [recentReceipt]

    # Log action as User

logAction(user, recentReceipt)

    # Announce amount deposited

print("\n" + recentReceipt)

# Buy interface function

def buyInterface() :

    # Access user as a global variable

global user

    # Access userCash as a global variable

    global userCash

    # Access machineProfit as a global variable

global machineProfit

    # Access userReceipt as a global variable

global userReceipt

    # Log action as User

logAction(user, "Buying items.")

    # Run under an infnite loop until interface exit

    while True :

        # Ask user item ID input

        itemID = inputToInteger("\nEnter item ID: ")

        # Check if item ID does not exists

        if itemID not in range(len(itemStock)) :

            # Announce item ID does not exist

            print("\nItem ID does not exist.")

            # Reset loop

            continue

        # Ask user stock amount

        stock = inputToInteger("\nEnter item amount: ")

        # Check if item stock is insufficient

        if itemStock[itemID]["Stock"] < stock :

            # Announce insufficient item stock

            print("\nInsufficient item stock.")

            # Reset loop

            continue

        # Calculate total price by multiplying item price and stock

        totalPrice = itemStock[itemID]["Price"] \* stock

        # Check if the user has insufficient funds.

        if totalPrice > userCash :

            # Announce insufficient cash

            print("\nInsufficient cash.")

            # Reset loop

            continue

        # If all checks are valid, subtract current cash by total price

        userCash -= totalPrice

        # Add total price to storePrice

        machineProfit += totalPrice

        # Subtract current stock by stock requested

        itemStock[itemID]["Stock"] -= stock

        # Make a recent receipt

        recentReceipt = f"Dispensed {stock} {itemStock[itemID]["Name"]} for AED {totalPrice} with AED {userCash} left in deposit."

        # Record recent receipt as list

        userReceipt += [recentReceipt]

        # Log action as User

        logAction(user, recentReceipt)

        # Announce recent receipt

        print("\n" + recentReceipt)

        # Ask if user wants to continue or press enter key to exit interface

        userInput = inputLower("\nType \"Continue/C\" to continue making purchases or press enter key to exit interface...")

        # Check if user types anything but continue

        if not userInput in ("continue", "c") :

            # Exit function

            return

# Receipt interface function

def receiptInterface() :

    # Access user as a global variable

global user

    # Access userReceipt as a global variable

global userReceipt

    # Log action as User

logAction(user, "Viewing receipt.")

    # Get userReceipt length

receiptLength = len(userReceipt)

    # Check if recentReceipt is empty

    if receiptLength == 0 :

        # Announce recentReceipt is empty

        print("\nReceipt is empty.")

# Ask user to enter to continue

        textAnimation("\nPress enter key to continue...")

        # This is for responding to prompt

        input()

        # Exit function

        return

    # Print new line for interface formatting sake,

    # print() by default prints a newline if the parameters are empty

print()

    # This makes sure the loading bar will stop at a specified duration

    # timeStep / timeLength = 1 / iterations

    # timeStep = timeLength / iterations

    timeStep = shortDuration / receiptLength

    # Show userReceipt in a loop

    for index in range(receiptLength) :

        print(userReceipt[index])

        # Wait until specified duration

        time.sleep(timeStep)

    # Ask user to enter to continue

    textAnimation("\nPress enter key to continue...")

    # This is for responding to prompt

input()

# -------------------------------- # Vending Machine Interface # -------------------------------- #

# User interface function (otherwise known as the core of the program)

def userInterface() :

    # Access user as a global variable

global user

    # Access userCash as a global variable

global userCash

    # Access userReceipt as a global variable

global userReceipt

    # Log action as User

logAction(user, "Interface accessed.")

    # Run under an infnite loop until program exit

    while True :

        # Clear current console

        clearConsole()

        # Display user interface

        print(

            f"""

    \_\_  \_\_\_      \_          \_\_\_\_      \_\_            \_\_\_\_

   /  |/  /\_\_\_ \_(\_)\_\_\_     /  \_/\_\_\_  / /\_\_\_\_  \_\_\_\_\_/ \_\_/\_\_\_ \_\_\_\_\_\_\_\_\_

  / /|\_/ / \_\_ `/ / \_\_ \    / // \_\_ \/ \_\_/ \_ \/ \_\_\_/ /\_/ \_\_ `/ \_\_\_/ \_ \\

 / /  / / /\_/ / / / / /  \_/ // / / / /\_/  \_\_/ /  / \_\_/ /\_/ / /\_\_/  \_\_/

/\_/  /\_/\\_\_,\_/\_/\_/ /\_/  /\_\_\_/\_/ /\_/\\_\_/\\_\_\_/\_/  /\_/  \\_\_,\_/\\_\_\_/\\_\_\_/

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

Current cash: AED {userCash}

Receipt history: {len(userReceipt)}

            """

        )

        # Display stock

        displayStock()

        # Display available user options and ask user for input

        textAnimation("\nType \"Deposit/D\" to deposit cash, \"Buy/B\" to buy items, \"Receipt/R\" to open receipt, or \"Exit/E\" to exit machine.\n")

        # Set output string to all lowercase allowing for all selections will be case insensitive

        userInput = inputLower()

        # Check if user types "Deposit"

        if userInput in ("deposit", "d") :

            # Announce depositing cash

            print("\nDepositing cash...")

            # Enter deposit interface

            depositInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            Continue

        # Check if user types "Buy"

        if userInput in ("buy", "b") :

            # Announce buying items

            print("\nBuying items...")

            # Enter buy interface

            buyInterface()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            Continue

        # Check if user types "Receipt"

        if userInput in ("receipt", "r") :

            # Announce opening reciept

            print("\nOpening receipt...")

            # Enter reciept interface

            receiptInterface()

            # Reset loop

            Continue

        # Check if user types admin

        if userInput in ("admin", "a") :

            # Announce accessing admin

            print("\nAccessing admin...")

            # Enter admin access

            adminAccess()

            # Wait for a small duration

            loadingAnimation("\nLoading interface...", shortDuration)

            # Reset loop

            Continue

        # Check if user types "Exit"

        if userInput in ("exit", "e") :

            # Announce program exit

            print("\nGoodbye and have a nice day!")

            # Announce ejecting cash

            textAnimation(f"\nDispensing AED {userCash} for {user}...", shortDuration)

            # Make a new line in console

            print()

            # Wait for a small duration

            loadingAnimation("\nExiting interface...", shortDuration)

            # Exit function, usually I'd use quit() to exit program immediately but this is for the sake logging user actions

            return

        # If for some reason all inputs are not one of the above, announce invalid input

        print("\nInvalid input.")

        # Wait for a small duration

        loadingAnimation("\nLoading interface...", shortDuration)

# -------------------------------- # Program Main # -------------------------------- #

# Core function

def mainSystem() :

    # Access user as a global variable

global user

    # Access user as a global variable

    global userCash

    # Access user as a global variable

    global userReceipt

    # Run under an infnite loop until program exit

    while True :

        # Clear current console

        clearConsole()

        # Announce program via text animation

        textAnimation(

        """

 \_    \_\_               \_\_\_                \_\_  \_\_\_           \_\_    \_

| |  / /\_\_  \_\_\_\_  \_\_\_\_/ (\_)\_\_\_  \_\_\_\_ \_   /  |/  /\_\_\_ \_\_\_\_\_\_/ /\_  (\_)\_\_\_  \_\_\_

| | / / \_ \/ \_\_ \/ \_\_  / / \_\_ \/ \_\_ `/  / /|\_/ / \_\_ `/ \_\_\_/ \_\_ \/ / \_\_ \/ \_ \\

| |/ /  \_\_/ / / / /\_/ / / / / / /\_/ /  / /  / / /\_/ / /\_\_/ / / / / / / /  \_\_/

|\_\_\_/\\_\_\_/\_/ /\_/\\_\_,\_/\_/\_/ /\_/\\_\_, /  /\_/  /\_/\\_\_,\_/\\_\_\_/\_/ /\_/\_/\_/ /\_/\\_\_\_/

                             /\_\_\_\_/

Version 4.0, by Eldeston

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

        """

        )

        # Ask user for their name

        userInput = input("\nEnter username or enter a digit to exit.\n")

        # If user input is 0, exit funciton

        if userInput.isdigit() :

            # Exit function

            Return

        # Set user as userInput

        user = userInput

        # Wait for a small duration

        loadingAnimation(f"\nLogging in as {user}...", shortDuration)

        # Log action as User

        logAction(user, f"Logged in as {user}.")

        # Load user interface

        userInterface()

        # Set user cash to 0

        userCash = 0.0

        # Empty receipt

        userReceipt = []

        # Wait for a small duration

        loadingAnimation(f"\n{user} is logging out...", shortDuration)

        # Log action as System

        logAction(user, f"{user} logged out.")

# -------------------------------- # Program Main # -------------------------------- #

# Log action as System

logAction(system, "Booting system.")

# Core of the program

mainSystem()

# Log action as User

logAction(user, "Exiting main interface.")

# Log action as System

logAction(system, "Shutting down.")

# BASIC STRUCTURE

# - [x] Booting animation

# - [x] User sign in

# - [x] Main system

#     - [x] User interface

#         - [x] Deposit cash

#         - [x] Buy items

#         - [x] View receipt

#         - [x] Admin access (Secret trigger)

#             - [x] Enter password

#             - [x] Admin interface

#                 - [x] Withdraw profits

#                 - [x] Add stock

#                 - [x] Remove stock

#                 - [x] Change password

#                 - [x] Check logs

#                 - [x] Exit admin interface

#         - [x] Exit user interface

#     - [x] Exit main system

# OTHER IMPLEMENTED CHANGES:

# - [x] Code formatting

# - [x] Exception handling

# - [x] Implement a basic user interface

# - [x] Implement optional aliases for options

# - [x] Ask user if they want to make a second buy or implement an entire interface dedicated to buying additional items

# - [x] Implement a secret trigger to access admin interface:

#   - [x] Admin command to add new items

#   - [x] Admin command to remove items

#   - [x] Admin command to withdraw profits

#   - [x] Admin command to view system log

#   - [x] Admin command to change password

# DISCONTINUED CHANGES

# - [ ] Implement recommendation system using existing dictionary

# - [ ] Implement multiple categories of items using lists or dictionary

# - [ ] Implement accessibility features

#   - [ ] Keyboard navigation

#   - [ ] Text to speech

#   - [ ] Sound queues

**Reference List**

<https://stackoverflow.com/>

<https://wiki.python.org/moin/>

<https://www.pythoncheatsheet.org/>