

VENDING MACHINE PROGRAM - ASSESSMENT 2

A Comprehensive Documentation of the Development process of the
Vending Machine Program

By student: 0325109

Project Details:

This program aims to simulate the basic functions of a common vending machine using python. The basic functions simulated should include an interface for inserting funds, an interface for selecting the chosen product and a function that returns accurate change. Additional functions may be included such as inventory control, multiple purchases and receipts.

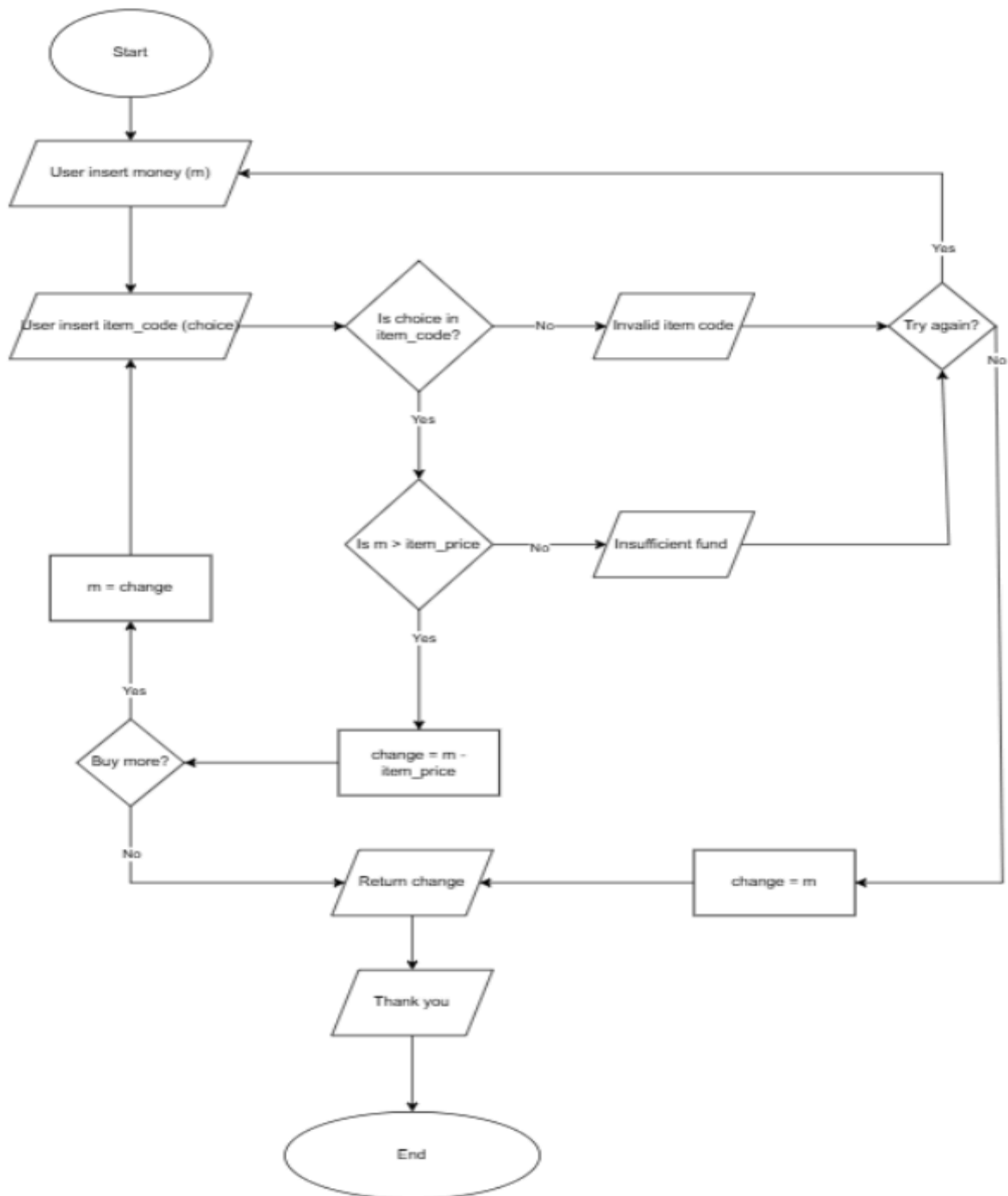
Specifications

The following is a list of features contained within the vending machine program:

- Allow users to view available vending machine items
- Accept and validate user funds
- Enable item selection via item codes
- Validate user selected item code
- Prevent purchases with insufficient funds or stock
- Update funds and stock as items are purchased
- Track purchased items
- Generate a receipt at the end of the transaction
- Calculate change to be returned with 25cents as a minimum denomination
- Return remaining balance as change

Flowchart

Below is a flowchart displaying the basic structure of the code. It shows the process of entering funds, selecting items, validating both the selection and the funds and finally returning change. It can also loop allowing for the user to purchase multiple items.



Technical description:

The vending machine program utilizes several functions in order to streamline the coding process and avoid creating large blocks of code that are harder to debug. The first step is to create 3 separate dictionaries that utilize the same key. The first dictionary contains item names and uses item codes as the key. The second dictionary contains item prices and uses the same key. The third dictionary contains the item stock and, like the former 2, also utilizes the same key. Utilizing the same key allows us to input the same key into multiple different dictionaries and obtain different details about the same item. It also greatly simplifies the process of turning the dictionaries into a unified table. The receipt list is also initialized. This is where purchase details are stored. These details will be displayed at the end of the program.

The first defined function is the “display_items()” function. The purpose of this function is to display the contents of all 3 dictionaries in a table-like structure. This will give the user an overview of an item’s code, name, price and its availability(stock). In order to create a table, I used string formatting to first create the headers and then I used a for loop alongside string formatting to fill out the details. The for loop runs through all the keys in the item name dictionary. Since they have the same keys, this key is also inserted into the other two dictionaries which allows us to display all aspects of our item. It is important to note that, due to the fact that monetary values are stored in cents, any balance or price displays will be divided by 100 to display them as dollars and not just cents. 2f is used when formatting the price in order to display it to 2 decimal places.

The next function is the “get_funds” function. This is used to obtain a user input that will be their balance. Float data type is used so that cents can be accounted for. Int does not accept decimal places. The input is then multiplied by 100 in order to obtain the balance in cents. In order to use primarily int data type, all monetary calculations are performed in cents. They are only converted into the float data type for display purposes. In the main body of code, a loop is created that can continuously call the get_funds function if the user so desires. This allows for funds to be added several times and simulates real world vending machines where multiple separate notes are added.

The buy_item function is the first function to accept arguments. The balance is required. This function asks the user to input the item code. First we check if the code is valid. Then we verify if the item is in stock and finally we verify if there are enough funds to make the purchase. If any of these checks are failed, an appropriate message is output and the balance is returned to the main body without being spent. In the main body, a loop will ask if the user wishes to continue buying. If yes the function will be called once

more. If the checks are passed then the cost will be deducted from the balance and it will return the new balance. Additionally, this transaction is recorded in the receipt list we initialized earlier.

The `print_receipt` function works similarly to the `display_items` function. They both use string formatting and for loops to go through all the data contained in the list/dictionaries. The difference is that this function has a total variable that sums up the cost of all purchases and is increased with every new iteration of the loop. The total as well as the balance taken as an argument are displayed at the bottom of the receipt.

The final function is `main`, the main body of the code. This is where all the functions are called. While true statements are used throughout the code in order to loop until the user meets a certain condition.

Critical reflection

The vending machine code works but there are several features I was unable to add either due to lack of skill or lack of time:

- One feature that proved difficult to implement was a purchase of several feature. This would allow the user to set an amount of items once they had selected a valid item. This complicated the verification process and made creating the receipt more difficult.
- Another feature that wasn't implemented was a menu system. This would allow the user to call upon the various functions themselves but the `ELIF` statements were too long and it created a large block of code that was difficult to debug.
- Due to lack of knowledge, I was unable to properly create a table. According to my research, there is a library called `tabulate` which allows for simplified table creation; however, despite my best efforts, any attempts at implementation led to more errors.
- When looking for a way to improve my code, a persistent inventory was suggested which would retain updated stock amounts but this proved to be too difficult.
- Creating a VAT system greatly complicated the code.
- A conditional discount was attempted but factoring all of this while simultaneously updating the balance was too complicated. I kept running into an issue where

funds were insufficient when attempting a purchase but the final change calculations showed enough money.

- The code assumes the user will only use cash. Card payments would not require such specialized change calculations

These were the features I was unable to implement due to various challenges and personal limitations. This vending machine program allowed me to display most of my acquired programming skills. Its diverse functions required different coding knowledge in order to properly implement. The biggest challenge was nailing down string formatting in order to make the display more visually appealing.

Appendix

```
#Create 3 dictionaries with the same key
item_code = {
    "1A": "Coke", "1B": "Diet Coke",
    "1C": "Pepsi", "1D": "Coke Zero",
    "2A": "7Up", "2B": "Fanta",
    "2C": "Sprite", "2D": "Mirinda",
    "3A": "Gatorade", "3B": "Orange Juice",
    "3C": "Iced Tea", "3D": "Water"
}

item_stock = {
    "1A": 10, "1B": 10,
    "1C": 10, "3A": 10,
    "3D": 10, "1D": 10,
    "2C": 10, "2A": 10,
    "2B": 10, "2D": 10,
    "3B" : 10, "3C" : 10,
}

item_price = {
    "1A": 260, "1B": 350,
    "1C": 230, "1D": 325,
    "2A": 300, "2B": 255,
    "2C": 275, "2D": 270,
    "3A": 610, "3B": 475,
    "3C": 445, "3D": 100
}

#Create a list to store purchases
receipt = []

#Define fuction to display vending machine inventory and prices
def display_items():
    #Print headers for display table using f to format it
    print(f"{'Code':<5} {'Name':<15} {'Price($)':<10} {'Stock':<5}")
    print("=" * 40)
    #Loop that goes through all dictionaries using the same key
    for code in item_code:
        price = item_price[code] / 100
        stock = item_stock[code]
```

```

        #Print dictionary contents under each header. price is formatted
so that it shows 2 decimal places
        print(f"{code:<5} {item_code[code]:<15} {price:<10.2f}
{stock:<5}")
        print("-" * 40)
        print("=" * 40)

#Function to accept valid user funds
def get_funds():
    #Will loop until valid funds are entered
    while True:
        amount = float(input("Enter funds: $"))
        #Check if user input is valid
        if amount > 0:
            #Return statement will automatically break the loop
            return int(amount * 100)
        elif amount <= 0:
            print("Amount must be positive.")
        else:
            print("Invalid amount. Try again.")

#Function that handles the buying process
def buy_item(bal):
    code = input("Enter item code: ").upper()
    #Validate user entry
    if code not in item_code:
        print("Invalid item code.")
        return bal
    #Verify stock availability
    if item_stock[code] == 0:
        print(f"{item_code[code]} is out of stock.")
        return bal

    price = item_price[code]
    #Check if funds are sufficient
    if bal < price:
        print("Insufficient funds.")
        return bal
    #Update stock and display remaining balance
    item_stock[code] -= 1

```

```

    bal -= price
    print(f"Dispensing {item_code[code]}")
    print(f"Remaining balance: ${bal / 100:.2f}")
    #Record transaction in list
    receipt.append((item_code[code], price))
    #Return remaining balance
    return bal

#Function to display receipt which includes all purchased items, total
price and change
def print_receipt(bal):
    #Header
    print("=" * 30)
    print("          RECEIPT")
    print("=" * 30)
    #Check if items were purchased
    if not receipt:
        print("No items purchased.")
    else:
        total = 0
        #Display purchased items along with price
        for name, price in receipt:
            print(f"{name:<15} ${price / 100:.2f}")
            #Sum up total cost
            total += price

        print("-" * 30)
        print(f"{'Total':<15} ${total / 100:.2f}")

    print(f"{'Change':<15} ${bal / 100:.2f}")
    print("=" * 30)

#Function to calculate change to be returned
def calc_change(bal):
    #Smallest denomination of change is 25 cents
    quarters = bal // 25
    change = quarters * 25
    print(f"Your change is ${change / 100:.2f}")

#Main function

```

```

def main():
    bal = 0

    while True:
        display_items()

        while True:
            #Call function to accept user input
            bal += get_funds()
            print(f"Current balance: ${bal / 100:.2f}")
            #Ask user if they want to input more money
            cont = input("Add more funds? (Y/N): ").upper()
            #Break loop if user does not want to continue
            if cont != "Y":
                break

        while True:
            #Call function to purchase items
            bal = buy_item(bal)
            #Ask user if they want to buy more items
            cont = input("Buy another item? (Y/N): ").upper()
            #Break loop if they do not want to
            if cont != "Y":
                break

        #Final check if user wants to buy more items
        cont = input("Make another transaction? (Y/N): ").upper()
        if cont != "Y":
            break

    #Print both receipt and change with a minimum denomination of 25c
    print_receipt(bal)
    calc_change(bal)

#Run program
if __name__ == "__main__":
    main()

```

