FIT9136 – Algorithms and Programming foundations in Python

Assignment Two

Michael McKay
Student ID: 32270208

Submitted:
11-Apr-2021

1. Scope:

This purpose of this document is to describe how the python script vending_machine.ipynb works.

2. Introduction:

The script vending_machine.ipynb is intended to simulate a working vending machine. The script fulfils a set of defined user requirements.

3. High level design:

There are four classes required for this program. Vending machine class, Location class, Ingredient class and Money class.

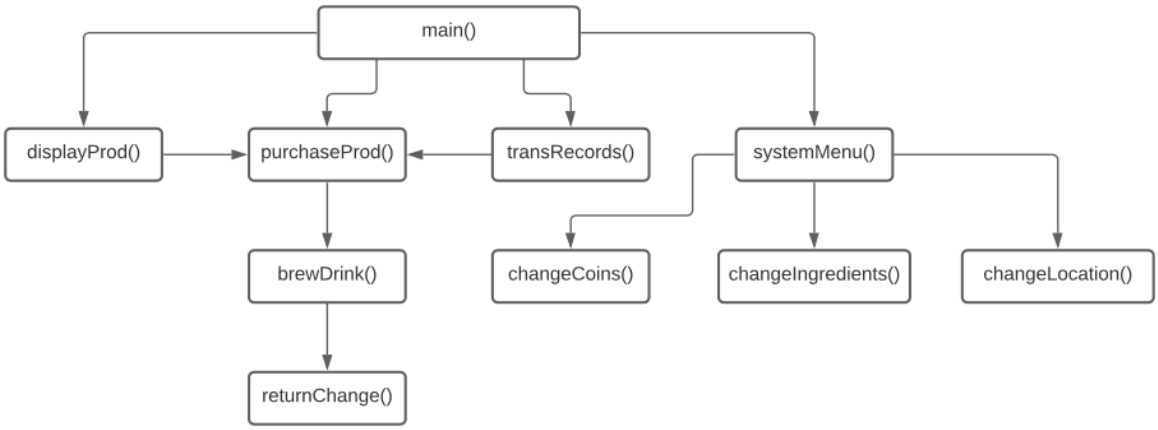
The Location class keeps track of each spot in the vending machine which can hold an item. It stores information such as location name, item name, price, stock, if item requires brewing or not (A drink like coffee needs to be made from scratch, and have ingredients added. But a drink like Coke is just sold as it), how many have been sold, and how much revenue has been generated.

The Money class keeps track of how many coins are in the machine, and how much they are worth. To keep things simple, all monetary values are in cents, and not dollars.

The ingredient class keeps track of the names of ingredients, the stock level, and how many have been dispensed.

The vending machine class is the class above all. When it is generated, it will populate the location, money, and ingredient classes with data from text files included. It will keep track of what items are loaded in the machine, how much money there is, what ingredients, as well as transaction records and statistics.

When the program is executed, it will create the vending_machine class with an input of True. This means the vending machine is online. It can be created offline as well with an input of false. It will then execute the main() function. The rest of the program can be summarised with the below chart:



Main serves as the main menu for the program. It consists of an input statement, and if statement which will execute one of the four functions below it.

displayProd() will show the user what products are available for sale. It will allow to user to go straight to **purchaseProd()**. **transRecords()** is used for showing all transaction records and statistics. User is also able to go straight to **purchaseProd()**. The program **systemMenu()** will allow uses to reset tranactions, reload data from text files, or change values for the three main class Money, Locations and Coins.

The function **purchaseProd()** is the one which does most of the work. When executed it will start a transaction and keep looping through the following code while the transaction is open. There is another nested while loop which continues until the user has selected a product and inserted the correct amount of money. Once these conditions have been met, and the if the drink requires brewing, it will execute the program **brewDrink()**. If the drink does not require brewing the transaction will be complete. The program **brewDrink()** will prompt the user if they want ingredients added. The transaction will be marked complete if a drink is successfully dispensed.

Now it will update transaction records with amount dispensed, and money inputted. It will run the program **returnChange()** to determine how much change to dispense. It will then update the transaction record to reflect this.

There is another program not in the flow chart above which is called often called **validInt()**. The purpose of this program is to take an input and see if it can be turned into an integer or not. This program is called on many things during the execution of the software and is mostly used to validate inputs from the user.

4. User requirements:

Function main():

```
Please choose one of the following:

a. Display products.
b. Choose your products.
c. Purchase, revenue and transaction records.
d. System menu.

Vending Machine Status: Online.


Please enter:
```

This menu appears when the program is first executed. It meets used requirement **S-001**, showing the status of the instrument. The status of the machine is set when the vending machine class object is first defined.

```
if __name__ == '__main__':

    #create class vending_machine to handle all locations, coins and ingredients within the vending machine.
    vending_machine = Vending_Machine(True)

    #execute main program.
    main()
```

If this value is false, the status will appear as offline, and the user will not be able to purchase any goods. The status can be changed by option 'a' within the system menu.

Function displayProd():

The user is then able to display the products available for sale by selecting option 'a' from the main menu. This will run the routine **displayProd()** This meets the requirements for req number **U-002.** This will bring up the following screen.

```
  location_name  item_name  price  stock
0            A1        Tea    150     10
1            A2     Coffee    250      5
2            A3       Coke    350     15
3            A4         OJ    300     20
4            B1      Fanta    350     10
5            B2   Gatorade    500      5
6            B3  Green_Tea    150      5
7            B4     Sprite    350      7
Available Ingredients: Sugar, Milk, Honey, Ginger.


Continue to buy product (y/n)?
```

This routine will allow the user to go straight to purchasing a product. The user can also do this from the transaction menu as well. Meeting the requirements for req number **U-006.**

Function transRecords():

Selecting option 'c' from the main menu will run the function transRecords() which displays the transaction records, the amount of coins in the vending machine, and the amount of ingredients in the machine.

```
Purchase, revenue and transaction records.
                 trans_date location   item  old_stock  new_stock  \
0 2021-04-09 21:34:47.248724       B1  Fanta         10          9
1 2021-04-09 21:34:58.816404       A1    Tea         10          9
2 2021-04-09 21:35:22.382026       A4     OJ         20         19

   old_money_total  new_money_total
0            38000            38350
1            38350            38500
2            38500            38800

       name  value  amount  total_value
0  coin_200    200     104        20800
1  coin_100    100     100        10000
2   coin_50     50     100         5000
3   coin_20     20     100         2000
4   coin_10     10     100         1000

     name  stock  dispensed
0   Sugar     99          1
1    Milk     99          1
2   Honey    100          0
3  Ginger    100          0

  location_name  item_name  dispensed  rev_generated
0            A1        Tea          1            150
1            A2     Coffee          0              0
2            A3       Coke          0              0
3            A4         OJ          1            300
4            B1      Fanta          1            350
5            B2   Gatorade          0              0
6            B3  Green_Tea          0              0
7            B4     Sprite          0              0

Total revenue generated since start up 800c.
```

The routine shows all transaction which have occurred, meeting the requirements for user requirement **D-006**, providing statistical data based on transactions. It will also give the user a list of coins in the instrument, and the amount of each present. This meets user requirement **D-005,** the system must store the coins and record all transactions.

Function purchaseProd():

Selecting option b from the main menu or selecting ‘y’ from the ‘Continue to buy product’ prompt within the display prod screen or the transaction record screen will run the function **purchaseProd().** Initially the user will see the screen below:

```
Welcome!  Please select a product by entering the location name and enter
required amount of coins to dispense a product.

  location_name  item_name  price  stock
0            A1        Tea    150     10
1            A2     Coffee    250      5
2            A3       Coke    350     15
3            A4         OJ    300     20
4            B1      Fanta    350     10
5            B2   Gatorade    500      5
6            B3  Green_Tea    150      5
7            B4     Sprite    350      7

Available Ingredients: Sugar, Milk, Honey, Ginger.
Unavailable products: None.
Product selected: None
Amount required: NA.
Amount entered: 0c
Enter location name of product or enter a coin (Only 200c, 100c, 50c, 20c, 10c allowed, C to cancel):
```

The user will be prompted with a welcome message for the start of the transaction, meeting the requirements for **S-005**, the system shall display welcome and goodbye messages.
The subroutine works by having an outer while loop and an inner while loop. The outer while loop is for the entire transaction, and inner while loop repeats until a user has selected a product and inserted the correct amount of money. Source code for loop is below:

```
# keep on looping while in transaction is in progress.  Transaction will end when routine is cancelled, or
# a beverage has been successfully dispensed.  This way information like coins inserted can be kept within
# context, and user does not need to start progress from scratch in order to select a new drink.
while (transaction_in_prog == True):

    #loop until enough money has been entered and a product has been selected.
    while ((amount_inserted < amount_required) or (product_selected == 'None')):
```

The start of the outer while loop marks a transaction beginning, meeting requirement **U-001**, a new transaction shall be started.

The cost of the product is on display in the table above, satisfying requirement **U-003**, cost of the selected item must be displayed. The user can enter coins by typing in the value of the coin as an integer, meeting the requirement **U-004**, coins must be inserted to get product.

The machine can accept a wide variety of coins:

```
(Only 200c, 100c, 50c, 20c, 10c allowed, C to cancel):
```

As coins are inserted the machine will check to make sure their valid, and one of the coin types above. It uses the following code to do this check:

```
#update the product_selected string if user enters a valid location number.
if validInt(inputStr) == True:

    #update the amount entered and the coin_count list if a valid coin has been inserted.
    if int(inputStr) in coin_list:

        coin_count[coin_list.index(int(inputStr))] += 1
        amount_inserted += int(inputStr)
```

Valid coins are defined in the list coin_list. This meets the requirements **S-010**, the system must accept coins of different amounts and **S-012** the system must check the validity of the coin.

The above table only shows items which are readily available. If the stock falls to 0, it will not be shown. This is achieved by putting a filter on the dataframe used to generate this table. The code for generating and displaying the data frame is below. This meets the requirements for **S-002**, only readily available items must be displayed.

```python
#create a dataframe containing all available locations in machine
location_df = p.DataFrame(vending_machine.return_location_list())

#filter out all locations which don't have any stock
location_df = (location_df.loc[location_df['stock'] > 0])

print('')
print(location_df[['location_name','item_name','price','stock']])
print(ingredStr)
print('')
```

If an item isn't available, it will say so beneath the data frame.

```
  location_name  item_name  price  stock
0            A1        Tea    150     10
1            A2     Coffee    250      5
2            A3       Coke    350     15
4            B1      Fanta    350     10
5            B2   Gatorade    500      5
6            B3  Green_Tea    150      5
7            B4     Sprite    350      7

Available Ingredients: Sugar, Milk, Honey, Ginger.
Unavailable products: OJ.
Product selected: None
Amount required: NA.
Amount entered: 0c
```

The user can enter in the product to be purchased by entering the location name into the user input box. When entered, the output for product selected and amount required will update.

```
Available Ingredients: Sugar,
Unavailable products: OJ.
Product selected: None
Amount required: NA.
Amount entered: 0c
```

```
Available Ingredients: Sugar, Milk, Honey, Ginger.
Unavailable products: OJ.
Product selected: A1
Amount required: 150c
Amount entered: 0c
Enter location name of product or enter a coin (Or
```

This is done by the following code in the **purchaseProd()** function.

```python
#check to see if a valid location name has been inserted.
if inputStr.upper() in loc_list:

    location_name = loc_class[loc_list.index(inputStr.upper())]

    # check to see if there is stock available of the selected item.
    if Location.return_stock(location_name) > 0:

        # update product_selected string, and amount_required.
        product_selected = inputStr.upper()
        amount_required = Location.return_price(location_name)

    else:

        # inform user that the item selected is out of stock.
        print('Product ' + Location.return_item(location_name) + ' currently unavailable.')
```

It will check to see if the input is a valid entry. If it is, it will update the product_selected string, and the amount_required string. It will check to make sure the item is in stock first and will return an error if it isn't. This meets the requirements for **S-009**, the system shall allow a user to select products.

The user can change their order as many times as they like before enough money has been entered to end the inner loop. This meets the requirements for **U-007**, orders shall be changed as required.

The user can cancel at any time by entering 'C' into the input box. This will execute the following code:

```
#cancel transaction if user enters C.
elif inputStr.upper() == 'C':

    #start piecing together error message.
    errorStr = 'Transaction cancelled.  Change returned:' + '\n'

    # loop through coin count, build string containing number of coins in list.
    for index in range(len(coin_list)):
        if int(coin_count[index]) > 0:
            errorStr += str(coin_count[index]) + 'x ' + str(coin_list[index]) + 'c, '

    errorStr = errorStr[:-2]
    errorStr += '.'

    print(errorStr)

    #transaction no longer in progress, end loops.
    product_selected = 'Canceled'
    amount_required = amount_inserted
```

It will set the variables product_selected as 'Canceled', and the amount_required as been equal to the amount_inserted. This will meet the conditions needed to end the inner loop. The subroutine will also return all the money the user as inputted, giving the following message.

```
Transaction cancelled.  Change returned:
1x 100c, 1x 50c, 1x 20c, 1x 10c.
```

This meets the requirements for **U-005**, the transaction shall be cancelled as required. It also meets the requirements for **S-004**, system shall refund the coin if in need.

If the user selects a product and enters enough money to purchase it, the requirements to end the inner loop will be met. This meets the requirements for **S-011**, the system must compare item cost with estimated cost. It will then execute an else statement, which checks if the drink needs to be brewed before been dispensed. The code for this is below:

```
#check if beverage needs to be brewed or not.
prod_to_dispense = loc_class[loc_list.index(product_selected.upper())]
brewing_req = Location.return_brew(prod_to_dispense)

if (brewing_req == 'True'):

    #run brewDrink subroutine if brewing required.
    successful_brew = brewDrink(Location.return_item(prod_to_dispense))

    if (successful_brew == True):
        #if brewing is successful the transaction is complete and can move onto next stage.
        transaction_in_prog = False

    else:
        #if anything other than True was returned it means transaction was cancelled,
        #and product selection should be reset.
        product_selected = 'None'

else:
    #if brewing is not required the transaction is complete and can move to next stage.
    transaction_in_prog = False
```

If brewing is required, it will run the function **brewDrink().**

```
Enter location name of product or enter a coin (Only 200c,
Item Tea ready in 10 seconds.
Item Tea ready in 9 seconds.
Item Tea ready in 8 seconds.
Item Tea ready in 7 seconds.
Item Tea ready in 6 seconds.
Item Tea ready in 5 seconds.
Item Tea ready in 4 seconds.
Item Tea ready in 3 seconds.
Item Tea ready in 2 seconds.
Item Tea ready in 1 seconds.

Insert Sugar automatically? (y/n) (enter "c" to cancel):
```

This will give a countdown to when the drink is ready. It will prompt the user if they'd like ingredients like milk or sugar to be added automatically. If the user selects yes, it will subtract one from the stock of that ingredient. The user can cancel at any time, returning False to this subroutine. A false value will set the product selected back to 'None', restarting the previous loop. This will allow the user to select another drink, without putting their money back in. This further satisfies requirements **U-005,** transaction can be cancelled at any time.

The waiting time above, satisfies requirement **S-006**, the machine must display waiting time and **D-003,** the system shall store Coffee and allow the user to mix sugar manually or automatically by the machine.

If the beverage didn't require brewing or was successfully brewed the function will update the variable transaction_in_prog to False. This will end the transaction. If the transaction was cancelled before this, the user would have been given their change back, meeting the requirement for **S-007**, the system must dispense the selected item only if the amount of coin is inserted, unless it must rollback the transaction. Now transcation is marked as complete, drink it dispensed and number of coins are added to their total in the coin classes.

A good bye message will be given, ending the requirments for **S-005**, system shall display welcome and goodbye messages at beginning and end of a transaction.

```
Item Fanta dispensed successfully.
Change returned: 1x 200c, 1x 50c.
Transaction complete!  Enjoy your beverage.
```

If an ingredient runs out when been added to a beverage, the machine will display a message saying it has been used up. It will then remove this from the list of available ingredients when the user requests a new beverage. This satisifies requirement **S-003**, alert message shall be printed automatically by the machine if any of ingredient is finished.

```
Item Tea ready in 10 seconds.
Item Tea ready in 9 seconds.
Item Tea ready in 8 seconds.
Item Tea ready in 7 seconds.
Item Tea ready in 6 seconds.
Item Tea ready in 5 seconds.
Item Tea ready in 4 seconds.
Item Tea ready in 3 seconds.
Item Tea ready in 2 seconds.
Item Tea ready in 1 seconds.
Insert Sugar automatically? (y/n) (enter "c" to cancel):y
Ingredient Sugar has been consumed.
Sugar inserted.
```

Function systemMenu():

This function is executed when the user selects option ‘d’ from the main menu. It’s the only main menu that will not allow the user to go straight to purchasing a beverage, as it would only be used for service engineers. The menu appears as below:

```
System menu.  Please choose one of the following:

a. Take vending machine offline.
b. Reset transaction records.
c. Reset stock item variables.
d. Reset money variables.
e. Reset ingredient item variables.
f. Change stock items.
g. Add/remove coins.
h. Add/remove ingredients.
i. Return to Main Menu.

Please enter:
```

It will allow the user to take the vending machine offline or put it back online by selecting option ‘a’. The user can reset all transaction records by selecting option ‘b’. The user can reset stock items, money, and ingredients in the machine by selection options ‘c’ though ‘e’. This will reset the lists used to keep track of these items and reload them from the txt files bundled with the python script. These satisfy requirement **S-008**, the system shall allow resetting operation for vending machine supplier. It will also allow user to change variable for the locations/stock items with option ‘f’. Add or remove coins with option ‘g’. And add or remove ingredients with option ‘h’. Pressing ‘I’ will return to main menu.