CECS 174 - Project 3
"Vending Machine Change Maker"
Due date: 10/28/22

Team: Lazy Coders

Student Name: Tyler Pham
Student ID: 029553223

I certify that this submission is my original work

Tyler Duy Pham

Student Name: Parth Balchandani
Student ID: 029629403

I certify that this submission is my original work

Parth Deepakkumar Balchandani

Project Report: Programming Project 3 - "Vending Machine Change Maker"

1. **Goal:** The goal of this project is to create a change maker for a vending machine that will only accept price inputs that are non-negative and are a multiple of 0.05, and will continue to ask for price at the end of each iteration if the user does not input 'q' or the user enters an invalid input. This program will then ask the user to deposit money to pay for their item (their price input), only accepting nickels (0.05), dimes (0.10), quarters (0.25), ones (1.00), and fives (5.00). The user may also choose to cancel the purchase and get a full refund. After the user completes depositing their money or cancels the purchase, the program shall display the correct change using the lowest amount of coins possible and must not dispense more coins than are in stock, display the updated stock, and finally ask the user for another price input or 'q' to quit. When the user quits, it shall display the total amount of money in stock.

2. **Problem Description:** In programming this Project 3, we had to use many if-statements for a number of inputs from the user. These if-statements are used to solve problems like ensuring the inputs from the user are valid, and that the correct outputs correspond to the inputs. Furthermore, we did make use of functions, but we only used null functions that do not return a value but instead print values, it is to make the main program more readable so as to not overflow it with print(...). We have used the 'continue' reserved statement once for the verification of user input to repeat part of the code that is necessary to repeat, which will be explained in detail.

3. **Program Description:**

There are three main problems that we have to find the solution to in programming this Project. The first problem is how to verify the user's input for price (which will be stored in the variable "price_input"). We must verify that the price is a multiple of 5 cents and that it is not negative. To simplify the operation, after getting the price_input, the program will enter the while loop and assign the variable 'price_as_whole' as 100 * price_input. In our program, we will round the operation that assigns 'price_as_whole' to ensure that the math operation that involves floats does not return any odd values (ex: 14.9999999999 instead of just simply 15).

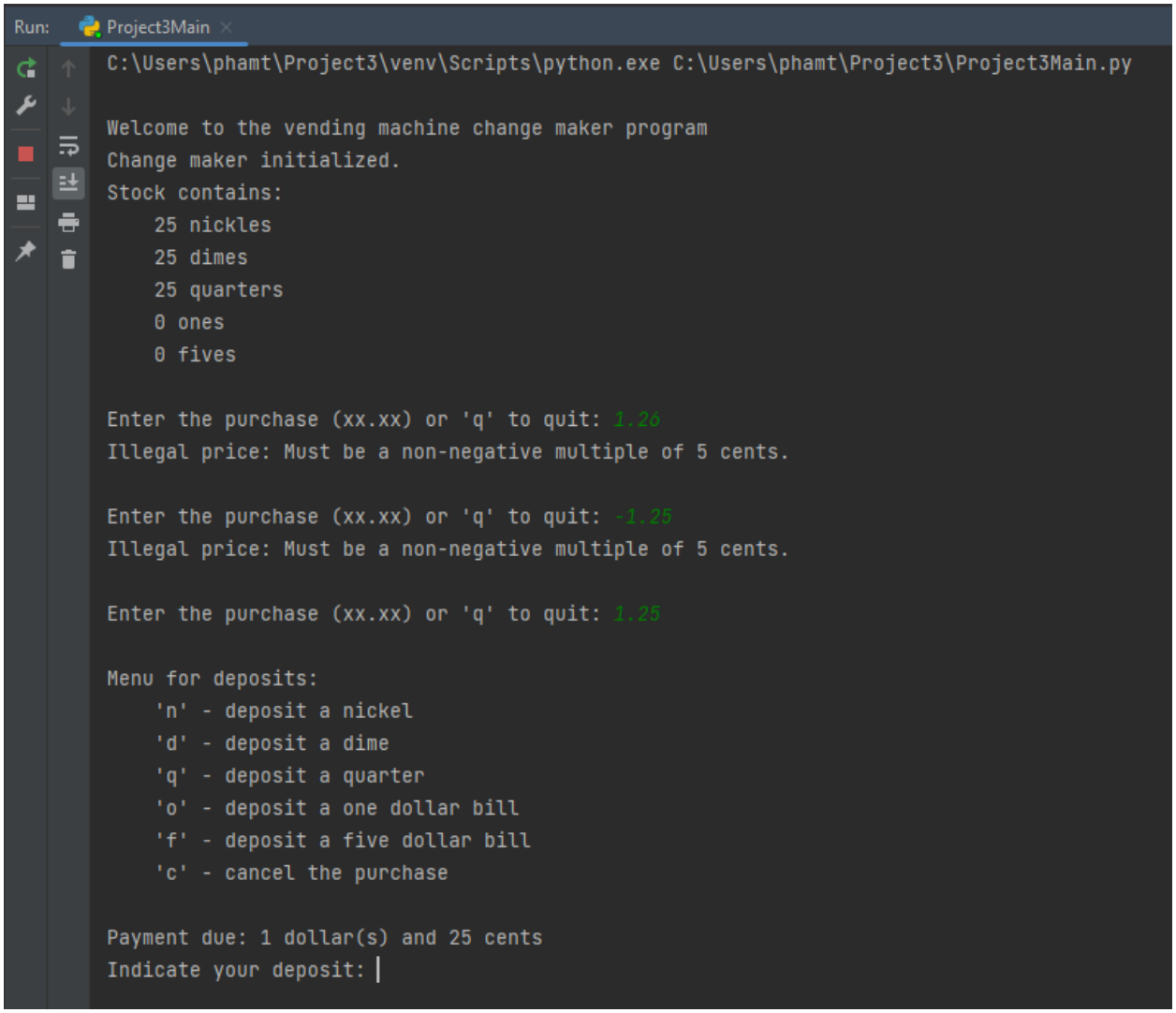
This is how this part of the program looks:

```
# This is the start of the script code for Project 3 by Tyler Pham and Parth Balchandani created in 10/21/2022.

# Stock of Coins and Dollars
nickels = 25
dimes = 25
quarters = 25
ones = 0
fives = 0

# Print welcome message for start of the program
print("\nWelcome to the vending machine change maker program\nChange maker initialized.")
```

```
def stock():  # Function that displays stock amounts
    print("Stock contains: ")
    print(f"\t{nickels} nickles")
    print(f"\t{dimes} dimes")
    print(f"\t{quarters} quarters")
    print(f"\t{ones} ones")
    print(f"\t{fives} fives\n")


def menu():  # Function that displays the menu
    print("\nMenu for deposits: ")
    print("\t\'n\' - deposit a nickel")
    print("\t\'d\' - deposit a dime")
    print("\t\'q\' - deposit a quarter")
    print("\t\'o\' - deposit a one dollar bill")
    print("\t\'f\' - deposit a five dollar bill")
    print("\t\'c\' - cancel the purchase\n")
```

```
60      stock()
61      price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")
62      while price_input != 'q':
63          price_input = float(price_input)
64          price_as_whole = round(price_input * 100)
65          # Price Verification
66          if (price_as_whole < 0) or (price_as_whole % 5 != 0):
67              print("Illegal price: Must be a non-negative multiple of 5 cents.\n")
68              price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")
69              continue
70          menu()
```

After stock() prints the amount of stock in the change-maker machine based on the variables under '# Stock of Coins and Dollar', the program will ask for a price_input. After "while price_input != 'q'" checks that price_input is not 'q', price_input will then be converted into a float so that it can be used in a math equation to determine the value of the variable 'price_as_whole'. The variable 'price_as_whole' is simply 100 * price_input, and an if-statement will check if this variable is negative or if it is not divisible by 5. If any of these conditions are true, it will print "Illegal Price: Must be a non-negative multiple of 5 cents."), then asks price_input again, and finally the 'continue' reserved word will go back to the start of the while loop in the screenshot. If the user enters a price input that is a non-negative multiple of 5 cents, then the if-statement does not run and continues on with the rest of the loop, otherwise it will continue to ask for a price_input. The user will know if their price_input is legal if the menu is printed, which will print the available inputs for the next part of the program.
The example below shows how the menu will not print until a legal price (one that is non-negative and multiple of 5 cents):



Our next problem is how to implement deposits, these are what pay for the imputed price that is stored in price_input and now stored as a whole number in price_as_whole. Our solution to this is to have the input for deposits be stored in the variable 'deposit_input' and using many if-statements determine what actions the program shall take.

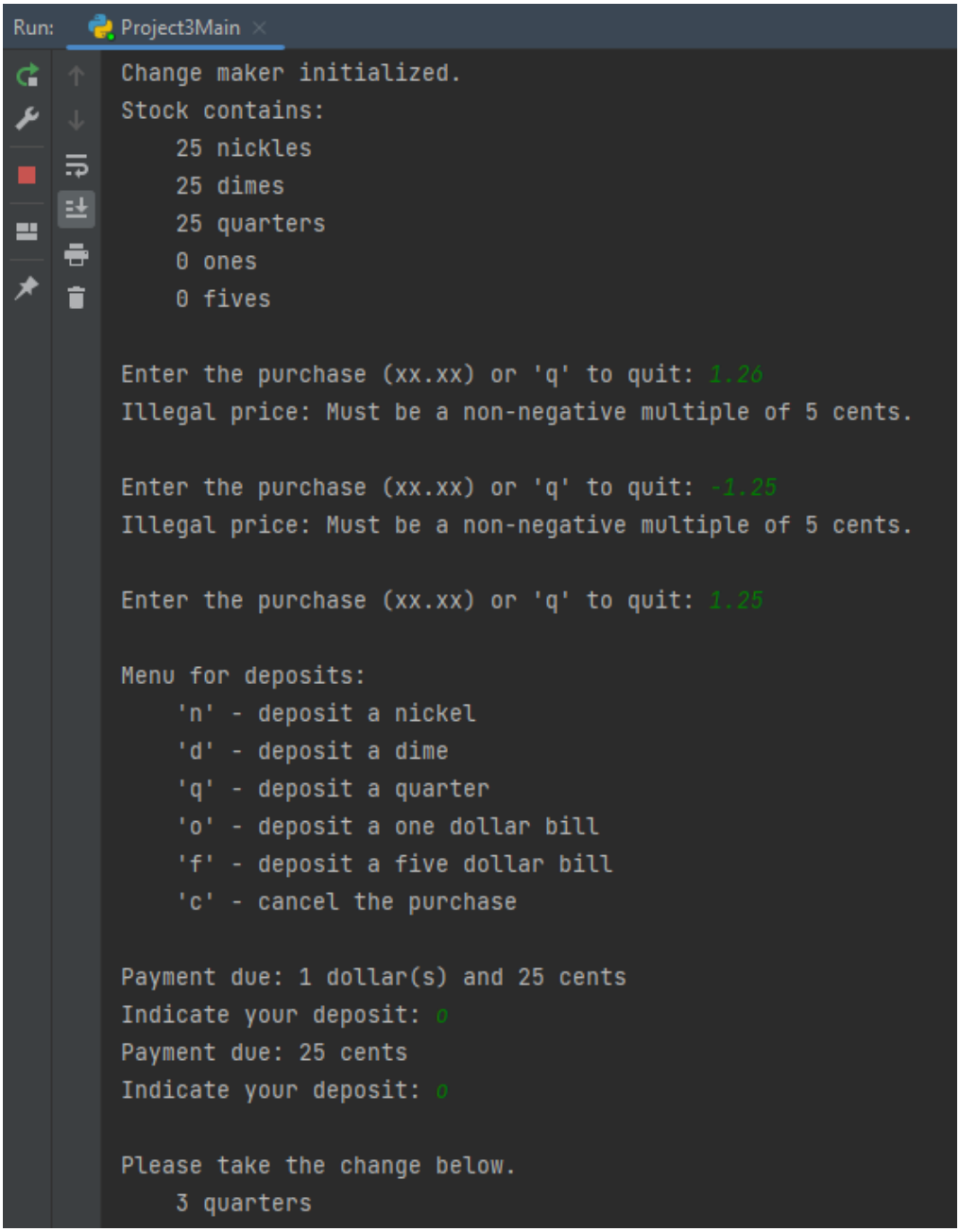
Below is the code for this section of the program (and the payment_due function):

```
33  def payment_due(total_cents):  # Function that displays payment due.
34      bills = int(total_cents / 100)
35      coins = int(total_cents % 100)
36      if bills >= 1:
37          print(f"Payment due: {bills} dollar(s) and {coins} cents")
38      else:
39          print(f"Payment due: {coins} cents")
```

```
    menu()
    # Deposit
    deposit_input = 0
    while deposit_input != 'c' and price_as_whole > 0:
        payment_due(price_as_whole)
        deposit_input = input("Indicate your deposit: ")
        if deposit_input == 'n':  # Deposits 1 nickel and adds it to stock
            price_as_whole -= 5
            nickels += 1
        elif deposit_input == 'd':  # Deposits 1 dime and adds it to stock
            price_as_whole -= 10
            dimes += 1
        elif deposit_input == 'q':  # Deposits 1 quarter and adds it to stock
            price_as_whole -= 25
            quarters += 1
        elif deposit_input == 'o':  # Deposits 1 one-dollar bill and adds it to stock
            price_as_whole -= 100
            ones += 1
        elif deposit_input == 'f':  # Deposits 1 five-dollar bill and adds it to stock
            price_as_whole -= 500
            fives += 1
        elif deposit_input == 'c':  # Cancels the entire payment
            price_as_whole -= round(price_input * 100)
        else:  # Detects bad inputs
            print(f"Illegal selection: {deposit_input}")
```

To initialize the while loop, we had to assign deposit_input the value of 0 so that it would enter the while loop. Then, the amount of payment due will be displayed, showing the valid price_input in terms of dollars and cents. The first five if- and elif-statements are very similar to each other, simply subtracting from price_as_whole and adding to the correct stock variables defined at the beginning of the program. For example, if the user inputs ‘q’, price_as_whole will have 25 subtracted from it, and a quarter will be added to the stock (quarter = quarter + 1). The most interesting part of this part is the last elif-statement “elif deposit_input == ‘c’:”, as some graders may ask “why not simply put this as your else-statement”. The else-statement the screenshot is used to detect any inputs that are not recognized for deposits, which are shown in the menu for deposits. The last elif-statement will subtract from price_as_whole by its initial value, which was defined as “round(price_input * 100). This is done so that when it comes time to calculate change, the correct value for change will be calculated with the negative value of price_as_whole, and that is change owed to the buyer. Whether the buyer has deposited some money and decides to cancel, or cancels immediately, the correct change will be calculated in the next section of the program due to these if-, elif-, and else-statements. After each valid input for deposit, the program will print how much more money is due, which is done via the function “payment_due(price_as_whole)”.

Below is an example of this section following the previous example:



As you can see, after inputting ‘o’, one dollar is subtracted from the payment due, and input for deposit is asked again. When inputting another ‘o’ to deposit another one dollar, the loop ends as the program detects the payment due is less than 0, and continues to the change section of the program.

The next and third problem is the change determination section, which shall give to the buyer the minimal number of coins as change. In order to simply calculate the amount of change, we need to take the absolute value of price_as_whole, which will be a negative value from the previous section, and assign that value to the variable ‘change’. Then, while change is not 0, a

series of if-, elif-statements will run to determine the minimum amount of coins to deposit as change. Three new variables ‘quarters_change’, ‘dimes_change’, ‘nickels_change’ will be used to determine how much of a coin can be dispensed to give as change, how much to subtract to give the left-over change, and how much of the coin will be removed from stock.

Below is this part of the program:

```
95      # Change determined from price as a whole
96      change = abs(price_as_whole)
97      print("\nPlease take the change below.")
98      if change != 0:
99          while change > 0:
100             quarters_change = change // 25
101             dimes_change = change // 10
102             nickels_change = change // 5
103             if quarters != 0 and change >= 25:
104                 if quarters < quarters_change:
105                     quarters_change = quarters
106                 change -= 25 * quarters_change
107                 quarters -= quarters_change
108                 print(f"\t{quarters_change} quarters")
109             elif dimes != 0 and change >= 10:
110                 if dimes < dimes_change:
111                     dimes_change = dimes
112                 change -= 10 * dimes_change
113                 dimes -= dimes_change
114                 print(f"\t{dimes_change} dimes")
115             elif nickels != 0 and change >= 5:
116                 if nickels < nickels_change:
117                     nickels_change = nickels
118                 change -= 5 * nickels_change
119                 nickels -= nickels_change
120                 print(f"\t{nickels_change} nickels")
121             else:
122                 print("Machine is out of change.\nSee store manager for remaining refund.")
123                 change_due(change)
124                 change = 0
125     else:
126         print("\tNo change due.")
```

```
41
42  def change_due(total_cents):  # Function that displays the amount of change left if there are no more coins.
43      bills = int(total_cents / 100)
44      coins = int(total_cents % 100)
45      if bills >= 1:
46          print(f"Amount due is: {bills} dollar(s) and {coins} cents")
47      else:
48          print(f"Amount due is: {coins} cents")
49
```

Let’s go through this section of the code step-by-step. First, the variables ‘quarters_change’, ‘dimes_change’, and ‘nickels_change’ determine the maximum number of quarters, dimes, and

nickels that can be given as change based on the current amount of change. These variables will change in value after each iteration of the while loop 'while change > 0'. Next, the if-statements and elif-statements will determine if there are quarters, dimes, or nickels present in stock to give as change. For example, the first if-statement says 'if quarters != 0 …' checks to see if there are quarters in stock, while the latter half of the if statement '... change >= 25' checks to see if the change is greater than 25 cents. Each elif-statement after the if-statement follows this same process but with adjusted numbers. Finally, there is an if-statement inside each of outer if- and elif-statements, that check if the change variables (i.e. quarters_change) is greater than the amount of stock (i.e. quarters), this is to prevent overflow by changing the variable that calculates the maximum amount of change that can be given to the current amount in stock. Once the program is done checking to see if there is overflow, it will subtract from the variable 'change' by the amount of coins times that coin's value (for example: if quarters_change is 2 and there are enough quarters to give to the buyer in stock, then variable 'change' will have 2 * 25 subtracted from it, which is 50 cents). The program will then subtract from the variable in stock by however many coins were given as change to the buyer. The final else-statement in the while loop is used to know if there are not enough coins in stock to give as change to the buyer, where it will print "Machine is out of change. See store manager for remaining refund." and the program will print out what change is owed to the buyer using the function 'change_due()'. The else-statement outside the while loop prints "No change due." which will print on the rare occurrence that the buyer pays with the exact amount of money. Once the full amount of change is given, the machine runs out of stock coins to give, or there is no change the program will move on to the restarting stage.

```
127     # Restart
128     print()
129     stock()
130     price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")
```

The final section (or second to final section) of this program is what we call the 'Restart Section', in which the program will display the updated stock and will prompt the user to input another price to store in the 'price_input' variable. After the user inputs a price, it will go all the way back to the top of the 'while price_input != 0' loop and verify the input.

By now, our output will look like this following the previous inputs:

```
Enter the purchase (xx.xx) or 'q' to quit: 1.26
Illegal price: Must be a non-negative multiple of 5 cents.

Enter the purchase (xx.xx) or 'q' to quit: -1.25
Illegal price: Must be a non-negative multiple of 5 cents.

Enter the purchase (xx.xx) or 'q' to quit: 1.25

Menu for deposits:
    'n' - deposit a nickel
    'd' - deposit a dime
    'q' - deposit a quarter
    'o' - deposit a one dollar bill
    'f' - deposit a five dollar bill
    'c' - cancel the purchase

Payment due: 1 dollar(s) and 25 cents
Indicate your deposit: o
Payment due: 25 cents
Indicate your deposit: o

Please take the change below.
    3 quarters

Stock contains:
    25 nickles
    25 dimes
    22 quarters
    2 ones
    0 fives

Enter the purchase (xx.xx) or 'q' to quit:
```

As you can see, our program has given the correct change with the smallest number of coins possible (which was 3 quarters), subtracted 3 quarters from the stock, and added two one-dollar bills to the stock. After printing the stock, the program prompts the user to enter the purchase (price_input). We will show what happens when we input ‘q’.

```
131
132     totals = (nickels * 5) + (dimes * 10) + (quarters * 25) + (ones * 100) + (fives * 500)
133     total(totals)
134
```

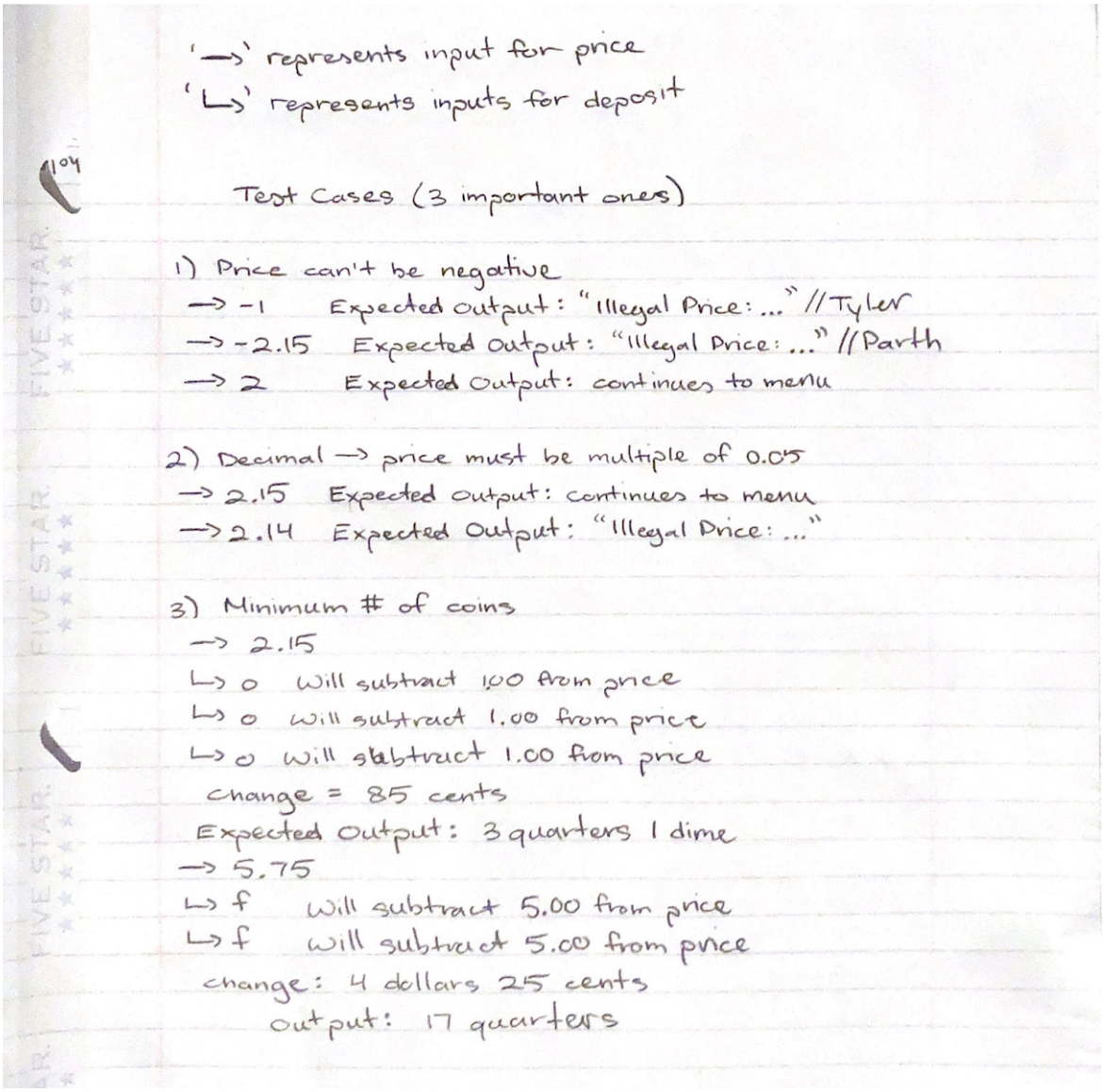
```python
def total(total_amount):  # Function that displays total amount of money in stock; The final amount.
    bills = int(total_amount / 100)
    coins = int(total_amount % 100)
    if bills >= 1:
        print(f"Total: {bills} dollar(s) and {coins} cents")
    else:
        print(f"Total: {coins} cents")
```

The true final section of the code occurs when the user inputs ‘q’ when prompted to enter a valid price. A new variable ‘totals’ will calculate the total amount in terms of cents of money that is the machine’s stock.

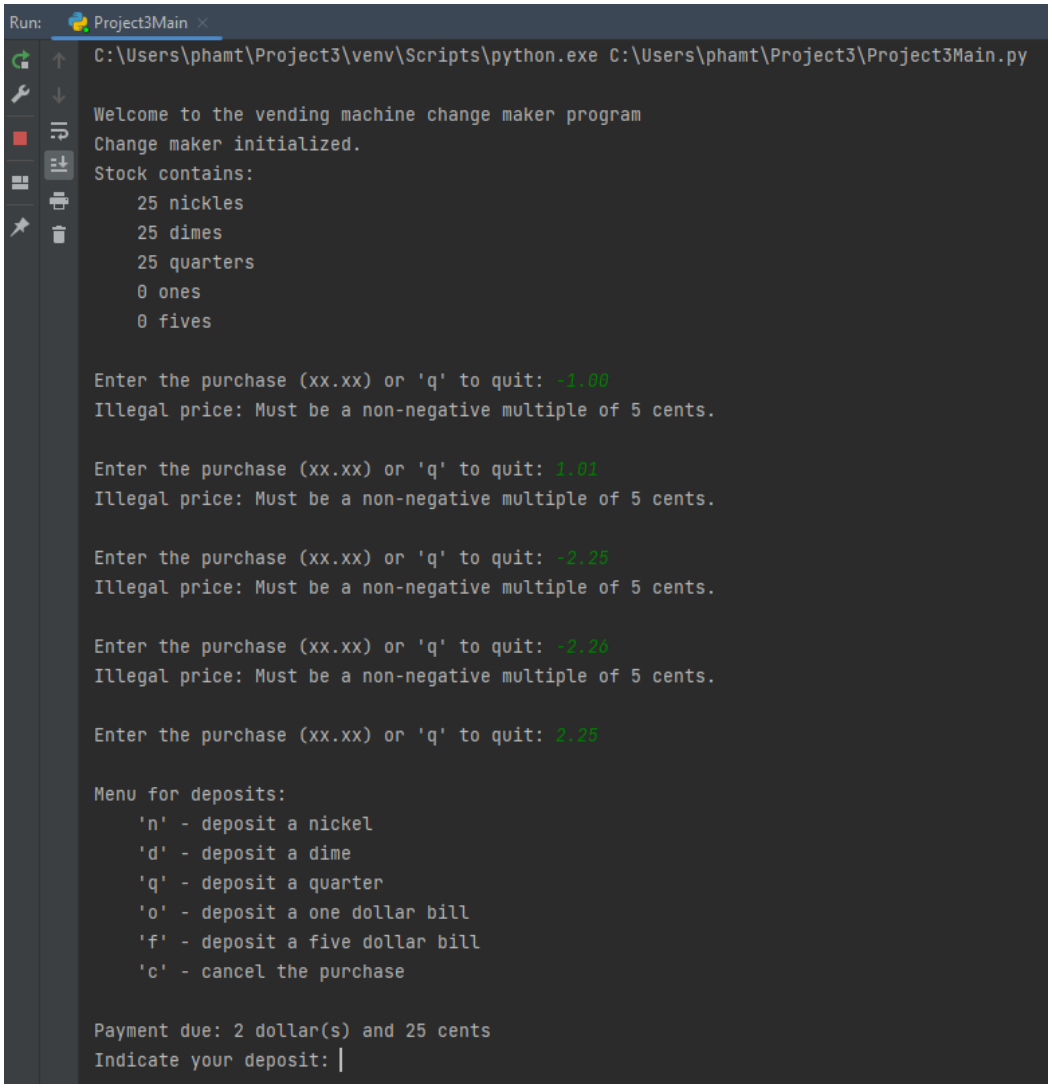
Below is our total and our program finishes:

```
Please take the change below.
    3 quarters

Stock contains:
    25 nickles
    25 dimes
    22 quarters
    2 ones
    0 fives

Enter the purchase (xx.xx) or 'q' to quit: q

Total: 11 dollar(s) and 25 cents

Process finished with exit code 0
```

Test Cases:

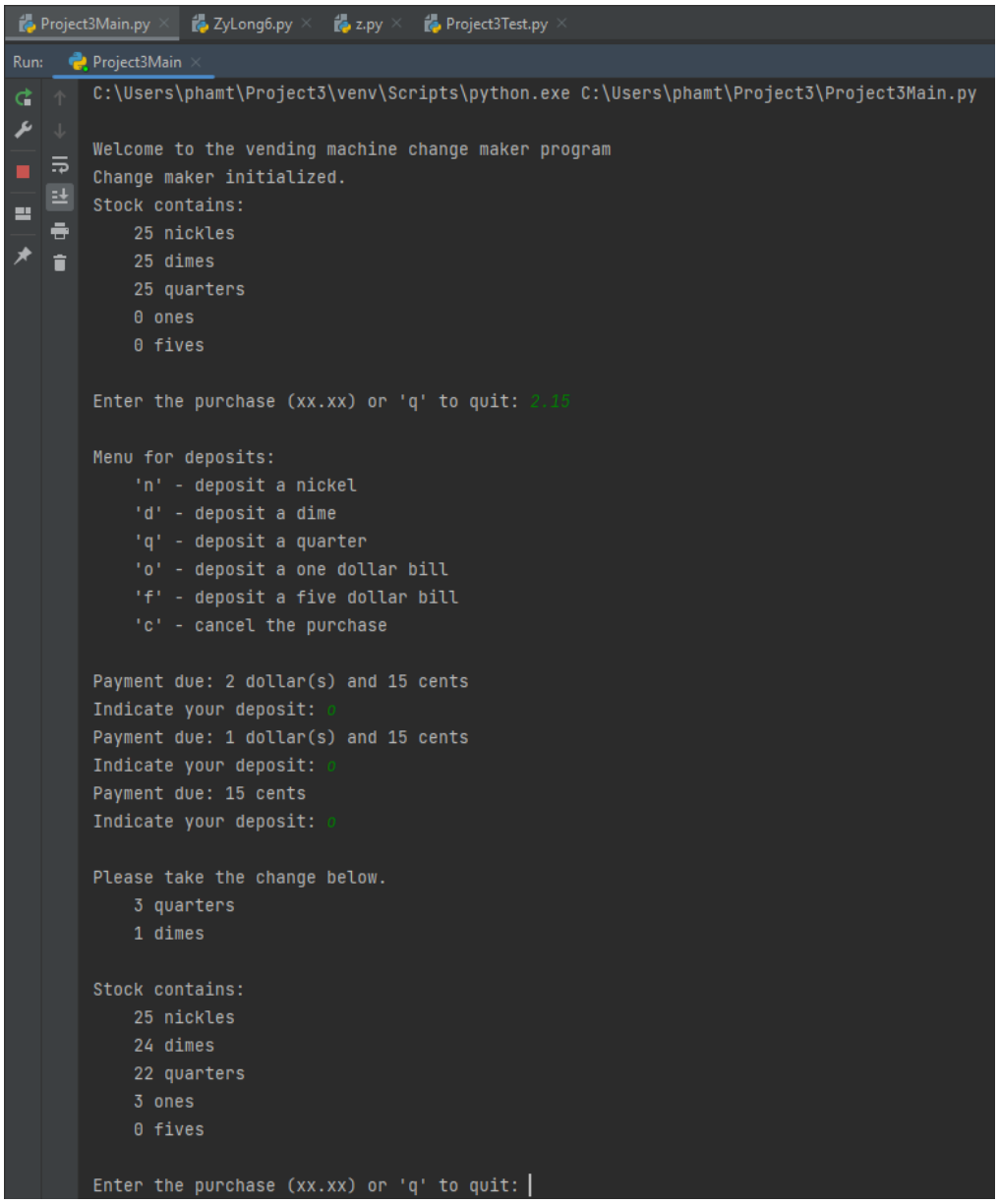


Test Case 1 - It is important to verify the user's input for purchase price as it is instructed in the Project specifications and makes making change simpler. Our test case will include the input values (-1.00, 1.01, -2.25, -2.26, 2.25) Below is our test case in the program and it outputs as expected (Illegal, Illegal, Illegal, Illegal, legal):

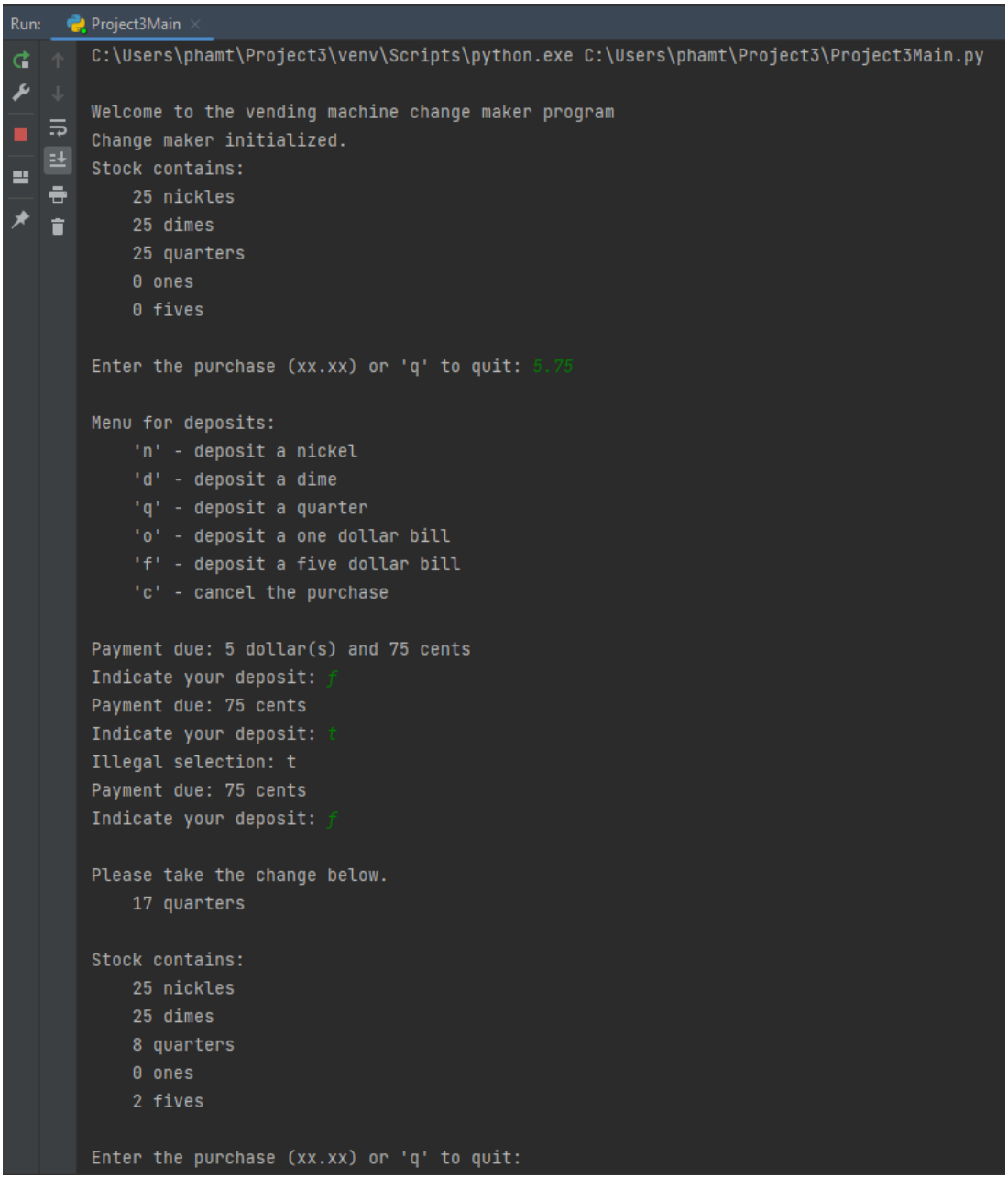


If the user's input is negative, the program shall inform the user that the price is illegal because it is either a negative float or not a multiple of 5 cents. After inputting '2' which is a legal price because it is both non-negative and a multiple of 5 cents, the program prints the menu.

Test Case 2 & 3 - We must ensure that when the user inputs for deposit after being prompted 'Indicate your deposit: ', that we must verify the input and also ensure that each input subtracts from the payment due correctly. Then, the correct amount of change must result from our inputs for deposits. The first screenshot shows the price 2.15, and after placing 3 one-dollar bills, the price is subtracted to 1.15, then 0.15, and finally moves on to calculating change. Change should be 85 cents, with the least amount of coins being 3 quarters and 1 dime. Below is our example that passes our test:



The second screenshot, the inputs are 5.75, which is legal input for price. Then, we input ‘f’, ‘t’, ‘f’. We expect that after inputting ‘t’ that the program will print “Illegal Selection: t”. For the two inputs ‘f’, we expect payment to decrease from 5.75 to 0.75 (5 dollars and 75 cents to 75 cents). Below is the second screenshot:


Run:
Project3Main
C:\Users\phamt\Project3\venv\Scripts\python.exe C:\Users\phamt\Project3\Project3Main.py
Welcome to the vending machine change maker program
Change maker initialized.
Stock contains:
25 nickles
25 dimes
25 quarters
0 ones
0 fives
Enter the purchase (xx.xx) or 'q' to quit: 5.75
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 5 dollar(s) and 75 cents
Indicate your deposit: f
Payment due: 75 cents
Indicate your deposit: t
Illegal selection: t
Payment due: 75 cents
Indicate your deposit: f
Please take the change below.
17 quarters
Stock contains:
25 nickles
25 dimes
8 quarters
0 ones
2 fives
Enter the purchase (xx.xx) or 'q' to quit:

Pseudo Code:

Tyler Pham (029553223)

Hand written Solution

Program shall not end to price_input == 'q'; while price_input != 'q'

Inputs for:

price → stored in variable "price_input"

Since Python calculates floats in an odd way, our program shall store the price as a whole number.

price_as_whole ← price • 100

ex: if price input is 2.15

then price_as_whole will be 215.

To verify price is non-negative and multiple of 0.05 we will compare with the if-statement:

if (price_as_whole) < 0 or (price_as_whole % 5 != 0):

print ("Illegal Price: ...")

get another price input

continue ← reserved word to start again at loop beginning.

deposit → stored in variable "deposit_input"

Valid inputs are:

'n' – subtract 5 from price_as whole and adds 1 to nickels

'd' – subtracts 10 cents from price and adds 1 to dimes

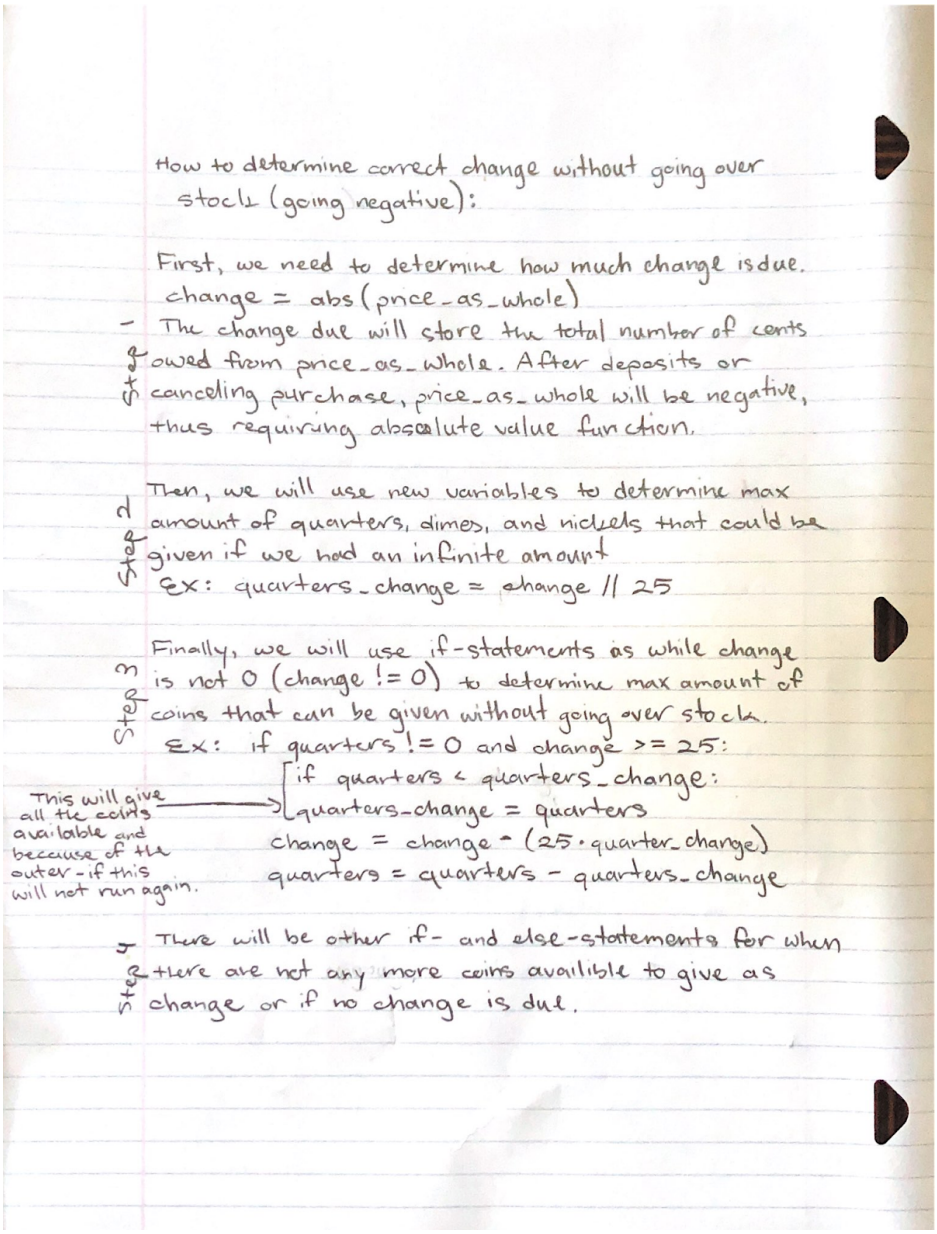
Same as 'n' and 'd' for valid inputs 'q', 'o', 'f' but with adjusted subtractions (25 cents, 1 dollar, 5 dollar)

'c' – cancels the purchase, gives back any money given in deposit

price_as_whole = price_as_whole – (price · 100)

This equation should give what change will be owed if user cancels the purchase.

While deposit_input != 'c' and price_as_whole < 0



How to determine correct change without going over stock (going negative):

First, we need to determine how much change is due.
change = abs(price_as_whole)
Step 1 - The change due will store the total number of cents owed from price_as_whole. After deposits or canceling purchase, price_as_whole will be negative, thus requiring absolute value function.

Step 2 Then, we will use new variables to determine max amount of quarters, dimes, and nickels that could be given if we had an infinite amount
Ex: quarters_change = change // 25

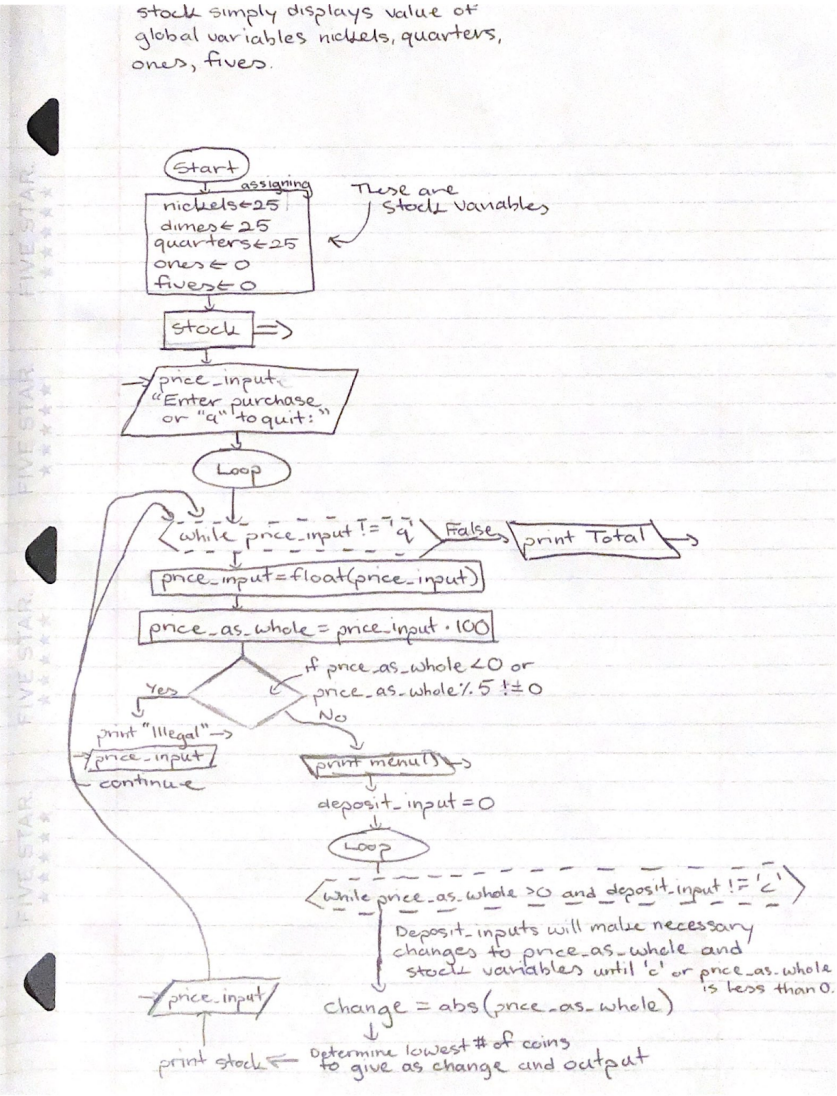
Step 3 Finally, we will use if-statements as while change is not 0 (change != 0) to determine max amount of coins that can be given without going over stock.
Ex: if quarters != 0 and change >= 25:
    if quarters < quarters_change:
        quarters_change = quarters
    change = change - (25 * quarter_change)
    quarters = quarters - quarters_change

This will give all the coins available and because of the outer - if this will not run again.

Step 4 - There will be other if- and else-statements for when there are not any more coins available to give as change or if no change is due.

IPO flowchart:

Our idea was so big that it cannot be handled in RAPTOR, so I have to draw a condensed version instead:



4. **Program Implementation:**

The data types that are used in this program are primarily floats and integers, as well as strings. The most used mathematical operations used in our program are integer division, division, and modulo operators. When we get our input for price stored in the variable 'price_input' we store it as a string in all areas of the program, this is so that we can check to see if the input is 'q' so that the "while price_input != 'q'" will not run another iteration. After that, price_input is converted into a float so that it can then be used in the equation to get the value of 'price_as_whole' which is simply price_input * 100. Modulo is used to see if the input for price is a multiple of 5 cents in verifying the price. Comparison operators such as '==' and '!=' were used in the verification process, and other processes such as during the Deposit section and the Change section of the program.

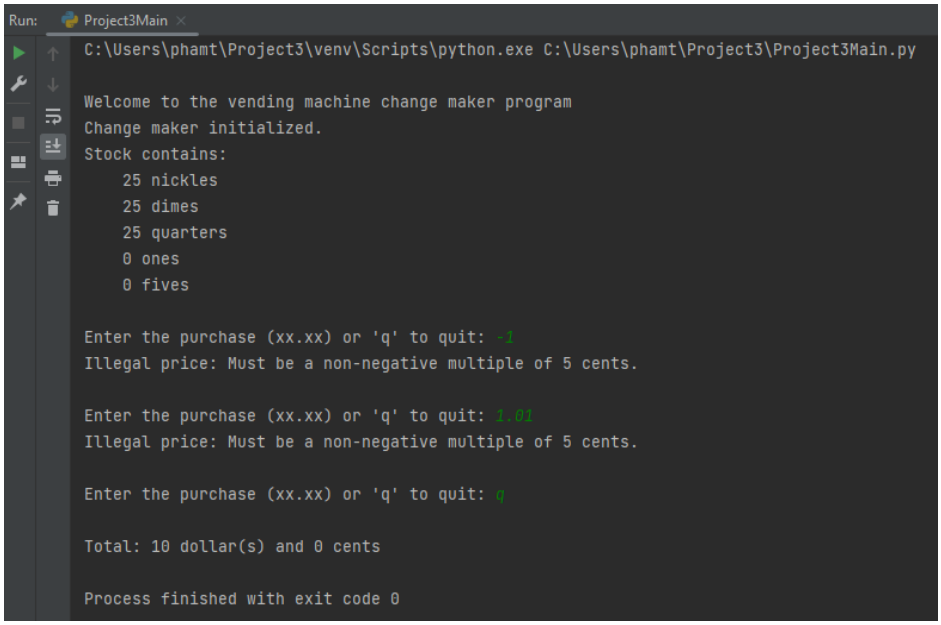
We would say that the most difficult part of implementing our program is the Change section of our program. The section that determines how to output the correct change using the lowest amount of coins and not going over what is in stock. In our program, we ended up

having more if-statements inside the if- and elif-statements to calculate for change overflow, and the Change section is easily the largest part of our code.

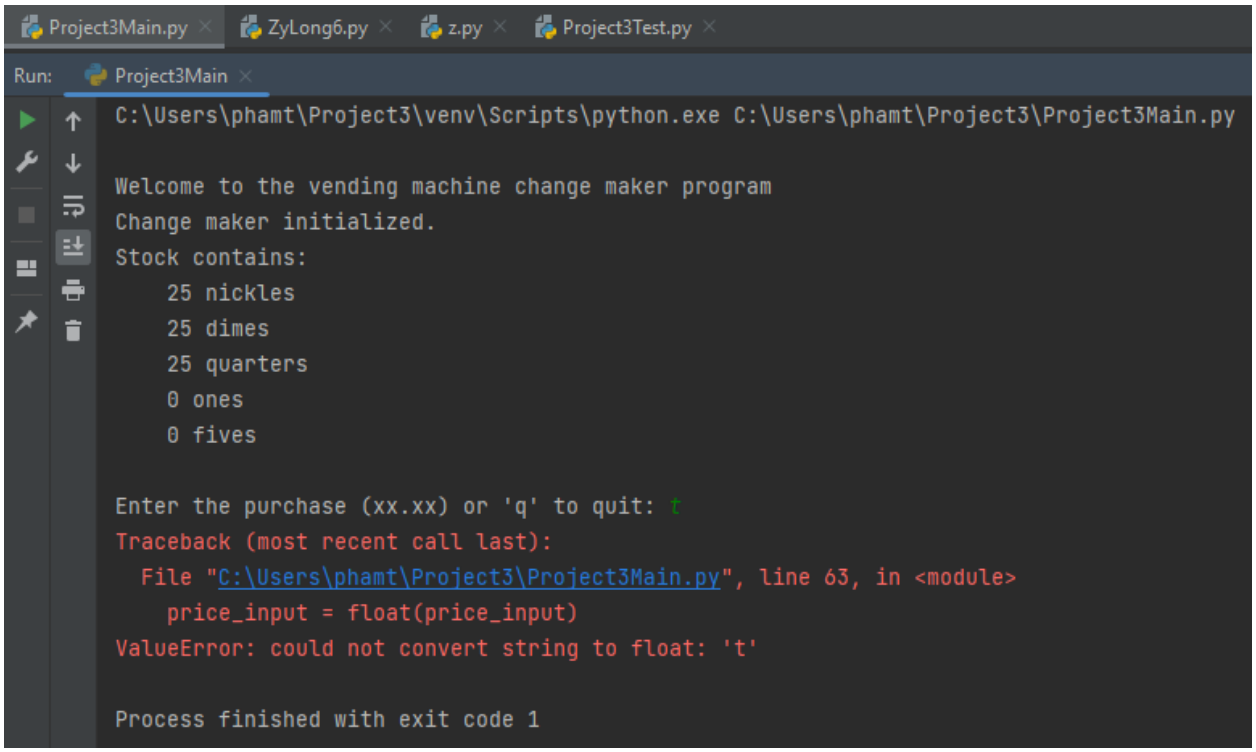
To be honest, the Change section was not easy but it was the most fun part to implement in the program. There was some trial and error, but overall the logic is very easy. Essentially, when reading the section under '# Change', you can think of it as 'if quarters != 0 and change >= 25: ' as "if there are some quarters in stock and leftover change is at least 25 cents". Then the if-statement within the first one 'if quarters < quarters_change: ' is thought up as "if there are not enough quarters to give as change, give as much as we can before moving on to the next biggest coin". This logic is repeated for dimes and nickels, with the else-statement inside the while-loop simply saying that there are no more coins for the change-maker to give.

We only made one small edit to our test cases, and that was to combine Test Cases 2 and 3. Other than that, we could have added a test case for the total amount of money in stock, but that part of the code is very easy to implement and does not need to be written down or requires its own test case. Simply adding up all the money that is in stock as money value with a calculator is a good enough test.

In terms of bad input that has to be verified, inputs for the variable 'price_input' after being prompted "Enter the purchase (xx.xx) or 'q' to quit: " can only verify inputs that are floats and 'q'.



However, our program, the way we designed it, cannot handle inputs that are any other string. For example, if we input 't' instead of 'q' or a float that can be verified, the below will result:



As far as we know, this error does not need to be fixed as it results from an input that does not need to be verified. Also, since we have not learned about exceptions yet, which would be used to combat the ValueError, this error cannot be fixed by current means that we know of. In all other areas where an input is required, such as being prompted “Indicate your deposit: ” which is stored in the variable ‘deposit_input’, it can handle any kind of bad input. It is only the input for purchase ‘price_input’ that cannot handle bad string inputs.

5. **Conclusion:**

The things that went well in this project is problem-solving, figuring out how to get correct outputs in terms of change with the least amount of coins possible. Getting the user’s inputs for prices and deposits were also fairly simple. Figuring out how to verify the price input was at least somewhat difficult but the code itself is fairly simple. I know that we should avoid the ‘continue’ and ‘break’ reserved words because they are somewhat hard to spot but it was the simplest way I could have the price be verified.

If given another opportunity and able to use different code, I would add exceptions to my code to reduce the amount of input errors. In terms of study skills and time management, I have done well in both aspects of this project. It is only the coding that I can improve on a bit with better techniques that we will be learning.

Some improvements that can be made to the project to be clearer or better is that perhaps the assignment should restrict what kind of code we can use. Such as ‘do not use null functions’ or something of that sort. Not that it would change the code all that much in my opinion, but it would be interesting.

To improve learning, perhaps there can be some pointers as to how to verify the input for price. Although, maybe leaving this problem open ended would let students be open-minded about solutions to the problem of verifying the price input, an input that can either be a float or a string.

Appendix:

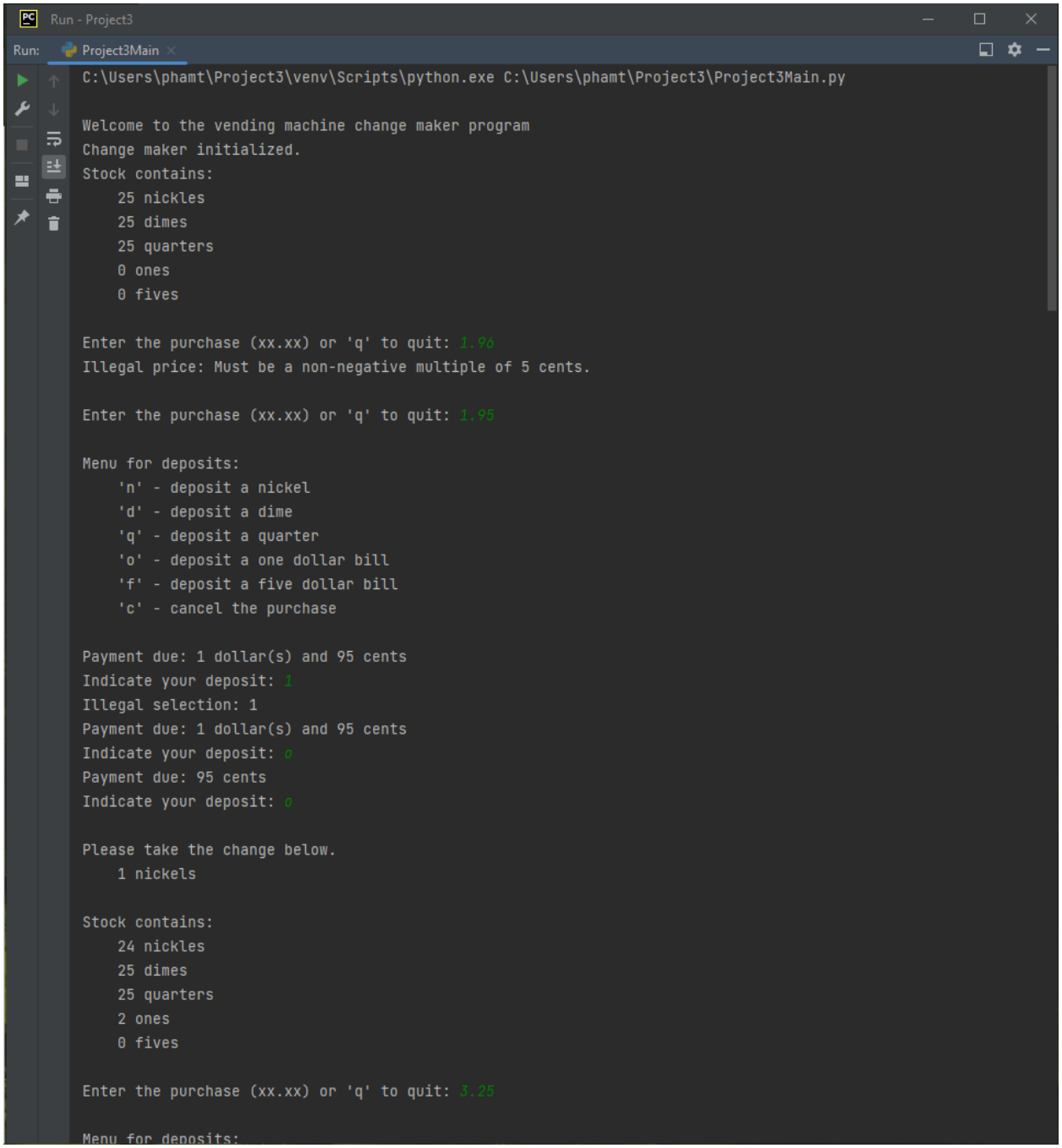
Project 3 Source Code:

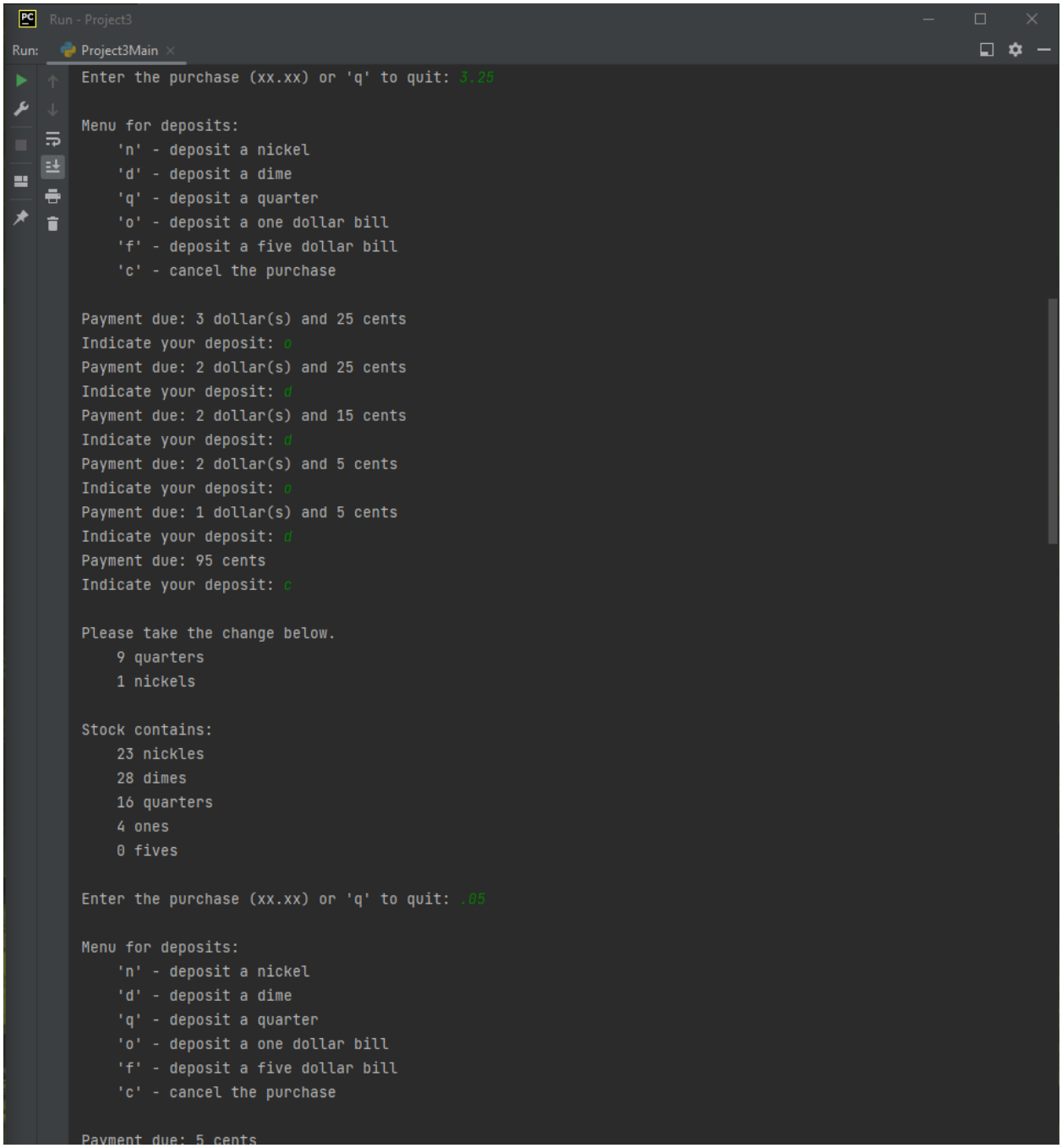
```
# This is the start of the script code for Project 3 by Tyler Pham and
Parth Balchandani created in 10/21/2022.

# Stock of Coins and Dollars
nickels = 25
dimes = 25
quarters = 25
ones = 0
fives = 0

# Print welcome message for start of the program
print("\nWelcome to the vending machine change maker program\nChange maker
initialized.")


def stock():  # Function that displays stock amounts
    print("Stock contains: ")
    print(f"\t{nickels} nickles")
    print(f"\t{dimes} dimes")
    print(f"\t{quarters} quarters")
    print(f"\t{ones} ones")
    print(f"\t{fives} fives\n")


def menu():  # Function that displays the menu
    print("\nMenu for deposits: ")
    print("\t\'n\' - deposit a nickel")
    print("\t\'d\' - deposit a dime")
    print("\t\'q\' - deposit a quarter")
    print("\t\'o\' - deposit a one dollar bill")
    print("\t\'f\' - deposit a five dollar bill")
    print("\t\'c\' - cancel the purchase\n")


def payment_due(total_cents):  # Function that displays payment due
    bills = int(total_cents / 100)
    coins = int(total_cents % 100)
    if bills >= 1:
        print(f"Payment due: {bills} dollar(s) and {coins} cents")
    else:
        print(f"Payment due: {coins} cents")
```

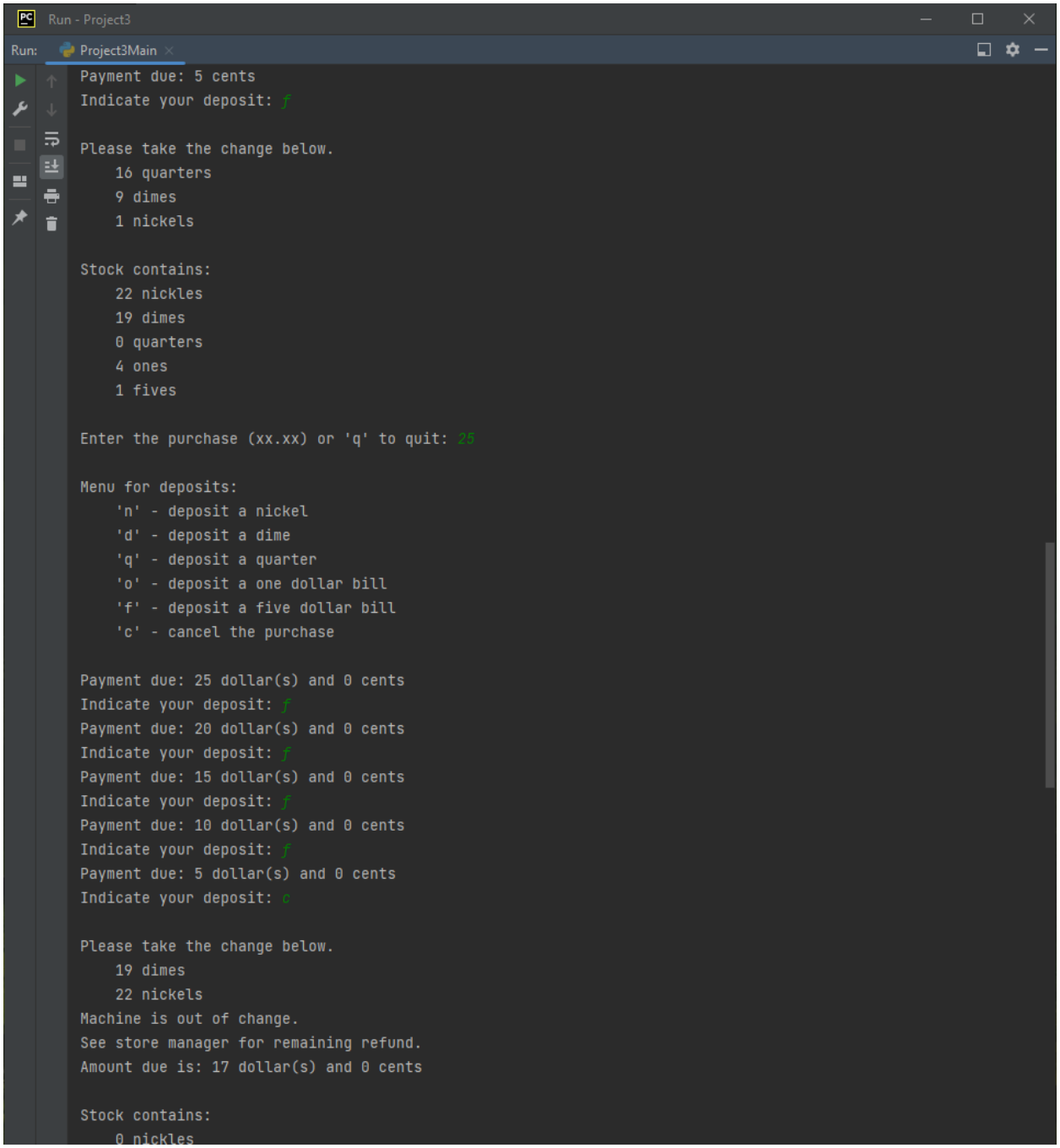
```
def change_due(total_cents):  # Function that displays the amount of change
left if there are no more coins.
   bills = int(total_cents / 100)
   coins = int(total_cents % 100)
   if bills >= 1:
       print(f"Amount due is: {bills} dollar(s) and {coins} cents")
   else:
       print(f"Amount due is: {coins} cents")


def total(total_amount):  # Function that displays total amount of money in
stock; The final amount.
   bills = int(total_amount / 100)
   coins = int(total_amount % 100)
   if bills >= 1:
       print(f"\nTotal: {bills} dollar(s) and {coins} cents")
   else:
       print(f"\nTotal: {coins} cents")


stock()
price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")
while price_input != 'q':
   price_input = float(price_input)
   price_as_whole = round(price_input * 100)
   # Price Verification
   if (price_as_whole < 0) or (price_as_whole % 5 != 0):
           print("Illegal price: Must be a non-negative multiple of 5
cents.\n")
       price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")
       continue
   menu()
   # Deposit
   deposit_input = 0
   while deposit_input != 'c' and price_as_whole > 0:
       payment_due(price_as_whole)
       deposit_input = input("Indicate your deposit: ")
       if deposit_input == 'n':  # Deposits 1 nickel and adds it to stock
           price_as_whole -= 5
           nickels += 1
       elif deposit_input == 'd':  # Deposits 1 dime and adds it to stock
           price_as_whole -= 10
           dimes += 1
         elif deposit_input == 'q':  # Deposits 1 quarter and adds it to
stock
           price_as_whole -= 25
           quarters += 1
       elif deposit_input == 'o':  # Deposits 1 one-dollar bill and adds it
to stock
           price_as_whole -= 100
           ones += 1
        elif deposit_input == 'f':  # Deposits 1 five-dollar bill and adds
it to stock
           price_as_whole -= 500
           fives += 1
```

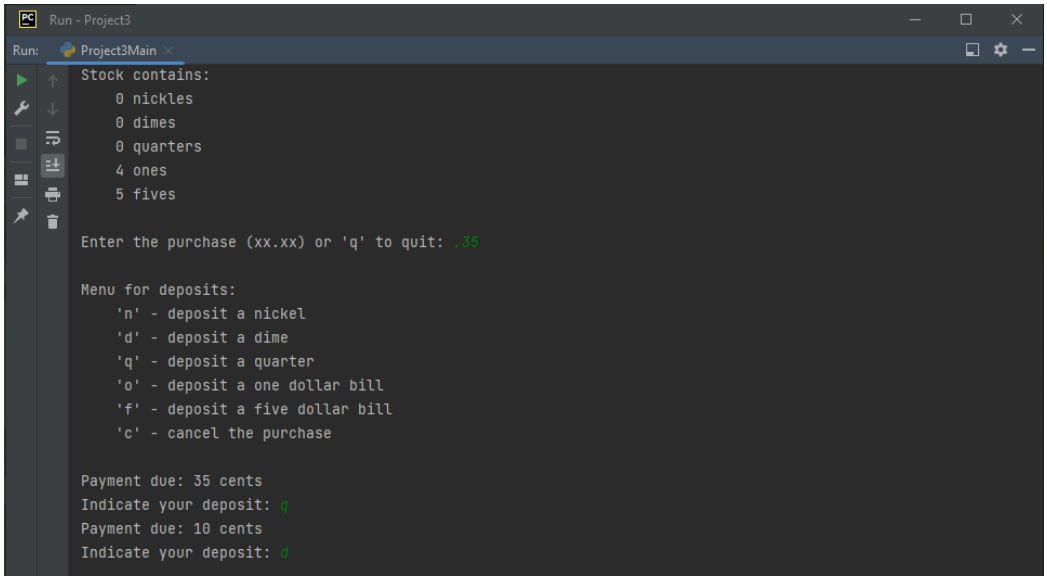
```
        elif deposit_input == 'c':  # Cancels the entire payment
            price_as_whole -= round(price_input * 100)
        else:  # Detects bad inputs
            print(f"Illegal selection: {deposit_input}")
    # Change determined from price as a whole
    change = abs(price_as_whole)
    print("\nPlease take the change below.")
    if change != 0:
        while change > 0:
            quarters_change = change // 25
            dimes_change = change // 10
            nickels_change = change // 5
            if quarters != 0 and change >= 25:
                if quarters < quarters_change:
                    quarters_change = quarters
                change -= 25 * quarters_change
                quarters -= quarters_change
                print(f"\t{quarters_change} quarters")
            elif dimes != 0 and change >= 10:
                if dimes < dimes_change:
                    dimes_change = dimes
                change -= 10 * dimes_change
                dimes -= dimes_change
                print(f"\t{dimes_change} dimes")
            elif nickels != 0 and change >= 5:
                if nickels < nickels_change:
                    nickels_change = nickels
                change -= 5 * nickels_change
                nickels -= nickels_change
                print(f"\t{nickels_change} nickels")
            else:
                print("Machine is out of change.\nSee store manager for
remaining refund.")
                change_due(change)
                change = 0
    else:
        print("\tNo change due.")
    # Restart
    print()
    stock()
    price_input = input("Enter the purchase (xx.xx) or \'q\' to quit: ")

totals = (nickels * 5) + (dimes * 10) + (quarters * 25) + (ones * 100) +
(fives * 500)
total(totals)
```

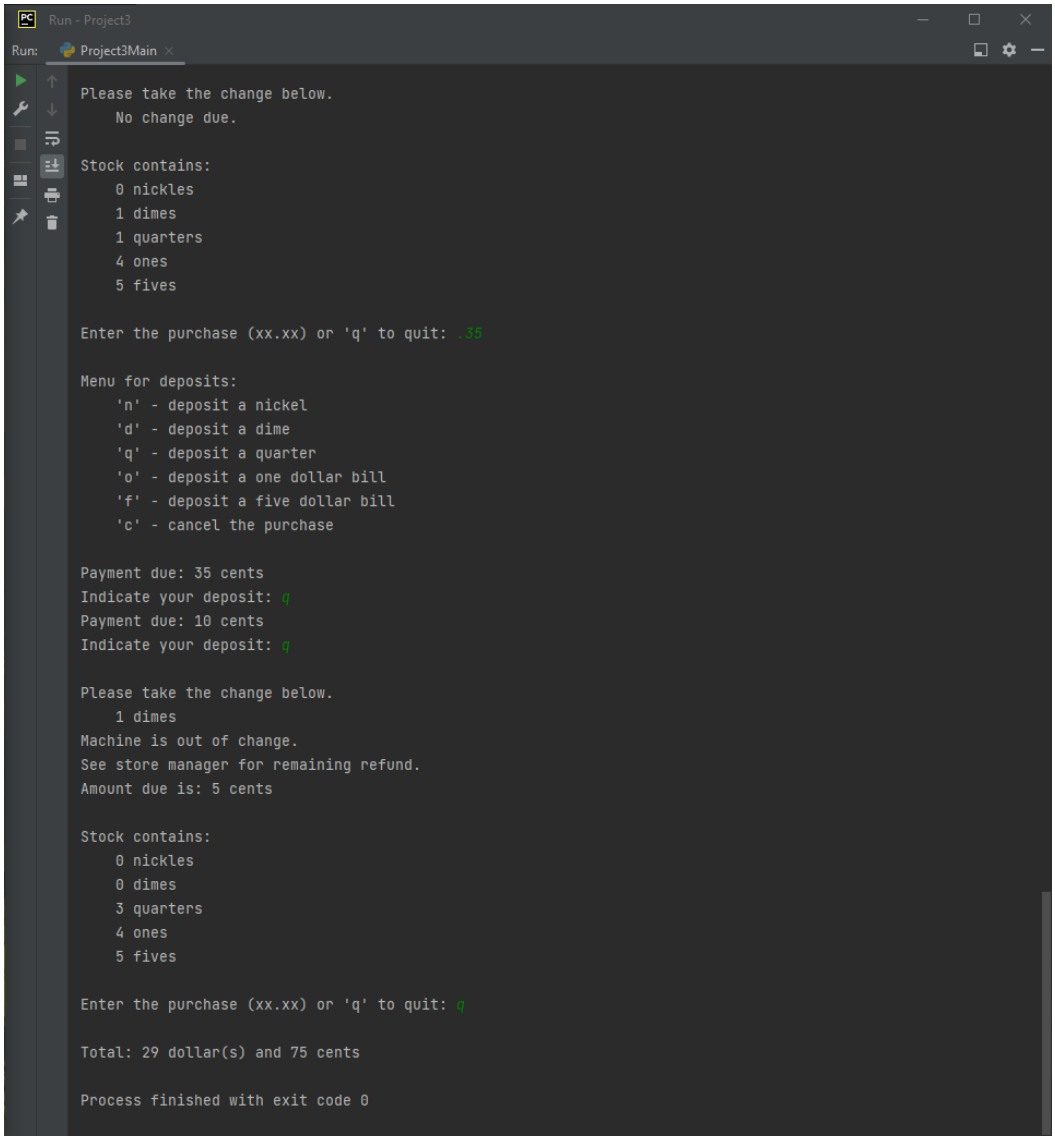
Project 3 Output (Using Instructor's inputs in PDF; Inputs are green):


Run - Project3
Run:
Project3Main
C:\Users\phamt\Project3\venv\Scripts\python.exe C:\Users\phamt\Project3\Project3Main.py
Welcome to the vending machine change maker program
Change maker initialized.
Stock contains:
25 nickles
25 dimes
25 quarters
0 ones
0 fives
Enter the purchase (xx.xx) or 'q' to quit: 1.96
Illegal price: Must be a non-negative multiple of 5 cents.
Enter the purchase (xx.xx) or 'q' to quit: 1.95
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 1 dollar(s) and 95 cents
Indicate your deposit: 1
Illegal selection: 1
Payment due: 1 dollar(s) and 95 cents
Indicate your deposit: o
Payment due: 95 cents
Indicate your deposit: o
Please take the change below.
1 nickels
Stock contains:
24 nickles
25 dimes
25 quarters
2 ones
0 fives
Enter the purchase (xx.xx) or 'q' to quit: 3.25
Menu for deposits:


Run - Project3
Run:
Project3Main
Enter the purchase (xx.xx) or 'q' to quit: 3.25
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 3 dollar(s) and 25 cents
Indicate your deposit: o
Payment due: 2 dollar(s) and 25 cents
Indicate your deposit: d
Payment due: 2 dollar(s) and 15 cents
Indicate your deposit: d
Payment due: 2 dollar(s) and 5 cents
Indicate your deposit: o
Payment due: 1 dollar(s) and 5 cents
Indicate your deposit: d
Payment due: 95 cents
Indicate your deposit: c
Please take the change below.
9 quarters
1 nickels
Stock contains:
23 nickles
28 dimes
16 quarters
4 ones
0 fives
Enter the purchase (xx.xx) or 'q' to quit: .05
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 5 cents


Run - Project3
Run:
Project3Main
Payment due: 5 cents
Indicate your deposit: f
Please take the change below.
16 quarters
9 dimes
1 nickels
Stock contains:
22 nickles
19 dimes
0 quarters
4 ones
1 fives
Enter the purchase (xx.xx) or 'q' to quit: 25
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 25 dollar(s) and 0 cents
Indicate your deposit: f
Payment due: 20 dollar(s) and 0 cents
Indicate your deposit: f
Payment due: 15 dollar(s) and 0 cents
Indicate your deposit: f
Payment due: 10 dollar(s) and 0 cents
Indicate your deposit: f
Payment due: 5 dollar(s) and 0 cents
Indicate your deposit: c
Please take the change below.
19 dimes
22 nickels
Machine is out of change.
See store manager for remaining refund.
Amount due is: 17 dollar(s) and 0 cents
Stock contains:
0 nickles


Run - Project3
Run:
Project3Main
Stock contains:
0 nickles
0 dimes
0 quarters
4 ones
5 fives
Enter the purchase (xx.xx) or 'q' to quit: .35
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 35 cents
Indicate your deposit: q
Payment due: 10 cents
Indicate your deposit: d


Run - Project3
Run:
Project3Main
Please take the change below.
No change due.
Stock contains:
0 nickles
1 dimes
1 quarters
4 ones
5 fives
Enter the purchase (xx.xx) or 'q' to quit: .35
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'o' - deposit a one dollar bill
'f' - deposit a five dollar bill
'c' - cancel the purchase
Payment due: 35 cents
Indicate your deposit: q
Payment due: 10 cents
Indicate your deposit: q
Please take the change below.
1 dimes
Machine is out of change.
See store manager for remaining refund.
Amount due is: 5 cents
Stock contains:
0 nickles
0 dimes
3 quarters
4 ones
5 fives
Enter the purchase (xx.xx) or 'q' to quit: q
Total: 29 dollar(s) and 75 cents
Process finished with exit code 0