CECS 174 - Project 3

“Vending Machine Change Maker”

Due Date: 10/31/22

Team: Chris Kau and Kathryn Woest

Student Name: Chris Kau

Student ID: 029599906

I certify that this submission is my original work

Chris Kau

Student Name: Kathryn Woest

Student ID: 030131541

I certify that this submission is my original work

Kathryn Woest

Project Report: Programming Project 3 - “Vending Machine Change Maker”

1. **Goal:** The goal of this project is to design a vending machine that’ll give the user their correct change upon entering a price and deposit.

2. **Problem Description:** We're required to implement a limited stock of coins that is able to be dispensed to the user. With this limited stock, we're still required to dispense the correct amount of change, with the least amount of coins possible. And the stock of coins can/will change depending on the user's deposits.

3. **Program Description:**

    a. **Handwritten/typed solution of the problem:** We're going to need to set the coin stock to 25 of every coin first. Then we need to get different user inputs such as their deposit and their purchase price. With their deposit, we're going to need elif statements to change the value of the change stock and update the price that the user has left to pay. And as the user requests the change, we'll need to calculate the amount of coins to give back to the user.

    b. **Decide on *test cases* for your code:**

    Test case for deposit: determines the amount of money the user deposits, or if the user inputs 'c' they'll get their refund back. Input "n", "d", "q", "o", "f", and "c" to ensure all valid cases worked and "g" and "1" to ensure that both numbers and characters would return back the invalid message. The valid inputs led to the program taking those inputs and calculating the rest due or calculating a refund, and the invalid inputs led the program to output an "invalid entry" message.

    Test case for total price: if dollars are present, then it'll print an output with "dollars". If not, it'll only print out "cents". We input prices of 1.95 and 0.95 to test dollar and no dollar situations. 1.95 output "1 dollar and 95 cents" and 0.95 output "95 cents", as we intended.

    Test case to check if the user's purchase price is a non zero number and if it's divisible by 5 cents. We input -5, 1.96, and 1.95 to test negative numbers, numbers not divisible by 5 cents, and then a valid entry. -5 and 1.96 both led to outputs of "illegal price" and had the machine ask them to input a new price or quit. 1.95 correctly continued through the code and asked the user for a deposit.

    Test case to check if there's more money to refund, if true, then it'll run a series of other test cases comparing quarters, dimes, and nickels possible to give back in change. We input exact deposits for the price we input (ie. the price was 5 cents and we input a nickel) and

deposits that were not equal to the price we input (ie. the price was 20 cents and we input a quarter). The exact deposit output a “no change” message and continued with the code, as intended. Additionally, the inexact deposit went through the change statement and gave us change, also as intended (ie. the machine deposited a nickel, if based off the example above).

c. **In simple steps (*pseudo code*) explain the algorithm you want to use to solve the problem for any input.**

1. We initialize the stock: fives = 0, ones = 0, quarters = 25, dimes = 25, and nickels = 25

2. Prints the available coin stock and asks the user to input a purchase price, or if they want to quit the process by entering ‘q’

3. The program will start off with a while loop checking to see if the user’s input (variable price) is not equal to ‘q’. If it’s equal to q, then the program will print out the total stock using a function that calculates the stock into a total price.

4. If the user did not input q, the program will first check to see if the user’s purchase is a non-negative number divisible by 5 cents, if they fail that test, they’ll be asked to re-enter a value

5. If the user pasts all tests, the program will print out the coin stock, take out all the dollars from the price and take out all of the cents in the user’s price

6. There will be another while loop with the condition that checks to see if the user’s amount unpaid is greater than zero. As long as it’s greater than zero this while loop will keep looping

7. The program checks to see if the user’s purchase price is greater than one dollar, and if so it’ll print out the payment due with dollars and cents. Then it’ll ask for the user’s deposit input.

8. Else, if the purchase price is less than a dollar, the program will only print out the payment due in cents. Then It’ll ask for the user’s deposit input.

9. There will be another while loop with the condition that the user’s deposit has not to be ‘n’, ‘d’, ‘q’, ‘o’, ‘f’, and ‘c’. While that condition is true, the loop will print out an illegal selection error and ask the user for their deposit again.

10. Inside the second while loop, there will be a series of test cases (if statements) checking to see which deposit the user made, and the program

will adjust the user's unpaid price value, and will add the corresponding coin/dollar to the stock.

11. However, if the user's deposit is 'c', we'll make a new variable called refund, which will subtract whatever the user's original price is minus the amount the user hasn't paid.

12. Then there will be a series of test cases using if statements to check the largest number of quarters that the machine can give back, then dimes, then nickels. All these are added to the refund and subtracted from the stock.

13. If the machine runs out of stock, and there is still change to give, the program will tell the user that the machine is out of change and to see the store manager for the rest.

14. After all the calculations are done, the program will check to see if the user has paid more than the payment due.

15. If so, we repeat steps 12 and 13 for change but with different variables so they don't get mixed up.

16. If the user pays exactly what is needed, then the program will tell the user there is no change due.

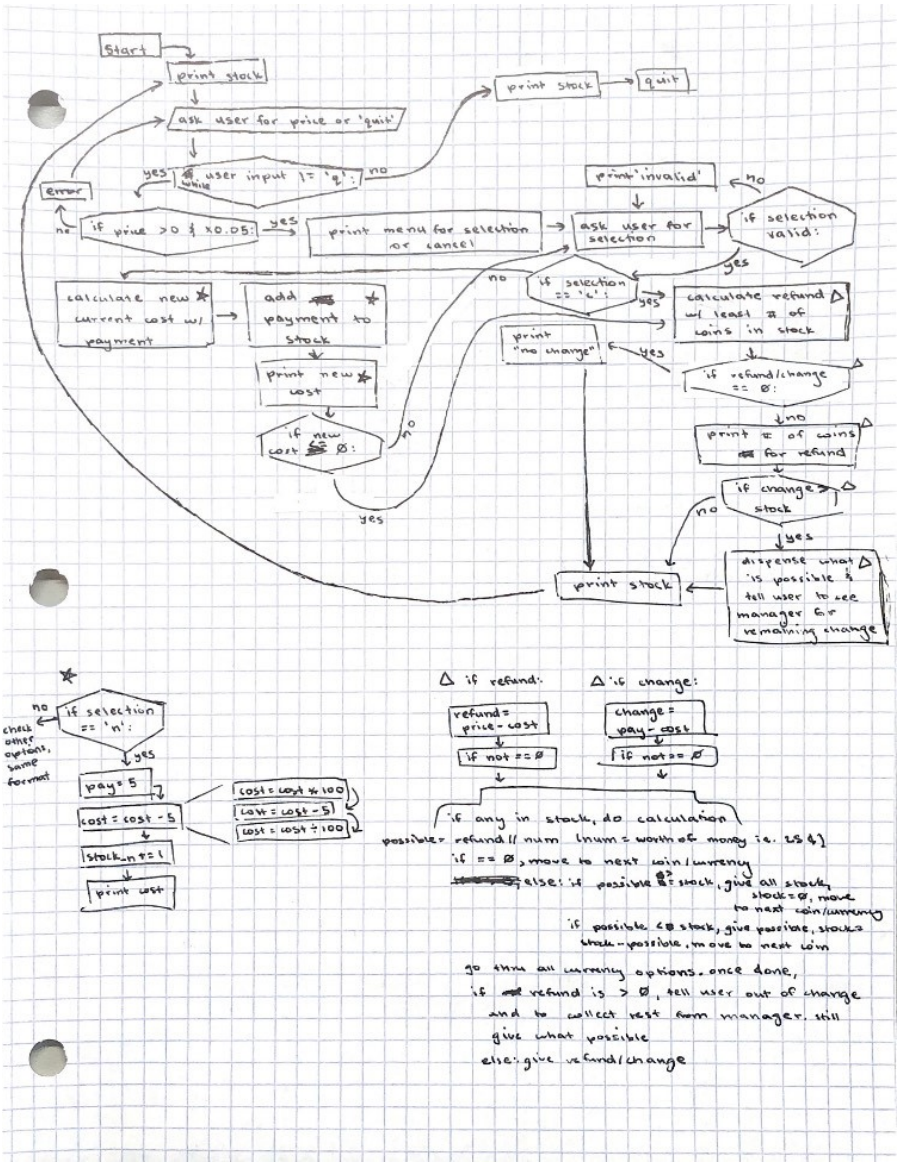
17. The program will then ask for a new purchase price and the program will start all over again.

d. **An *IPO* (top down-tree) diagram.**

Input	**Process**	**Output**
A price (valid)	Enters main code statement branch	Menu statement and a statement asking for deposit
A price (invalid)	Enters invalid input statement branch, then skips to the bottom of the main statement	Prints "illegal price" and re-asks for an input
'q' (quit)	Skips past the main while loop	Prints total stock and quits
'n' (nickel)	Computes new amount left to pay and updates stock	Prints the new amount left to pay and asks for another deposit if necessary. Else, prints any change and asks for a new price/quit
'd' (dime)	Computes new amount left to pay and updates stock	Prints the new amount left to pay and asks for another

		deposit if necessary. Else, prints any change and asks for a new price/quit
‘q’ (quarter)	Computes new amount left to pay and updates stock	Prints the new amount left to pay and asks for another deposit if necessary. Else, prints any change and asks for a new price/quit
‘o’ (one dollar)	Computes new amount left to pay and updates stock	Prints the new amount left to pay and asks for another deposit if necessary. Else, prints any change and asks for a new price/quit
‘f’ (five dollars)	Computes new amount left to pay and updates stock	Prints the new amount left to pay and asks for another deposit if necessary. Else, prints any change and asks for a new price/quit
‘c’ (cancel)	Calculates refund, if needed	Prints amount refunded and asks for a new price/quit

e. **A *flowchart* that corresponds to the major design of your program.**



4. **Program Implementation:**

   a. **What did you use:**

      i. Data types: strings, integers, and floats

      ii. Operations: addition, subtraction, multiplication, integer division, modular function

      iii. Built-in Functions: if-elif-else statements, while loops, input, print, the def function

b. **What was challenging while you were implementing your program?** The challenging part of our program would be trying to implement the refund system. Initially we tried to set it up to where the refund and change calculations would be inside a defined function at the top, which would be called on, but we soon realized that without utilizing lists it wasn't possible to return the updated stock, as it would have to return multiple values, and a function can't return more than one value. We finally implemented them inside the actual code itself, not before it.

c. **What was interesting/straightforward/fun to implement?** Something that was really stratforward to implement would be the test cases, and updating the price with every deposit the user inputted. Something interesting that we implemented was the functions we defined, since this was the first time that we did so on a project. Something fun that we implemented was calculating the total stock at the end of the code, which printed when the user quit.

d. **Did you add any more tests after you implemented the code? If so, which ones and why?** Yes, once all of the code ran and appeared to work, we ran all of the inputs that the example provided in order to ensure that the code did work as the project intended. The inputs all worked properly and the outputs matched the example exactly. We did test many of these values prior to before all of the code was implemented, but never all at once.

e. **How does your program handle bad input?** Our program handles bad input pretty well. With every bad input such as negative numbers, or blank inputs, it'll return a message asking the user to reinput something. The program tests for bad inputs in two locations as well. It first tests to see if the price input is positive and divisible by 5. If not, it asks the user to reinput a price or to quit. It then later tests to ensure that the deposit input is valid. If not, it prints "invalid" and asks for a new deposit.

f. Bugs and/or Errors: One major bug that our code had was that after a user canceled their purchase and received their refund, the code wouldn't skip to asking for a new price but would instead ask the user to continue depositing money for the old price. We fixed this by adding **break** statements after the print statements for the refunds, and the code functioned properly after that.

5. **Conclusion:**
    a. **What went well in this project?** All of the planning and putting our words into coding (with some adjustments) went pretty well. We were able to code the project relatively easily, with a couple minor hiccups, and got the code to run properly in the end.

    b. **What would you do differently given another opportunity (about writing a code, about your study skills, and time management)?** Writing the code went well, so I wouldn't change a thing about that. However, we weren't sure if we were able to use certain functions and tools such as lists which wasted a lot of time since we had to replace a lot of things in our code. And our time management was pretty well for the most part so I wouldn't change a thing about that.

    c. **What are the improvements that might have made the project better or clearer? Please be specific.** We don't really have anything to say about this other than changing the output example they gave us. At first glance, we weren't able to tell that the example was one entire output. It looked like multiple examples.

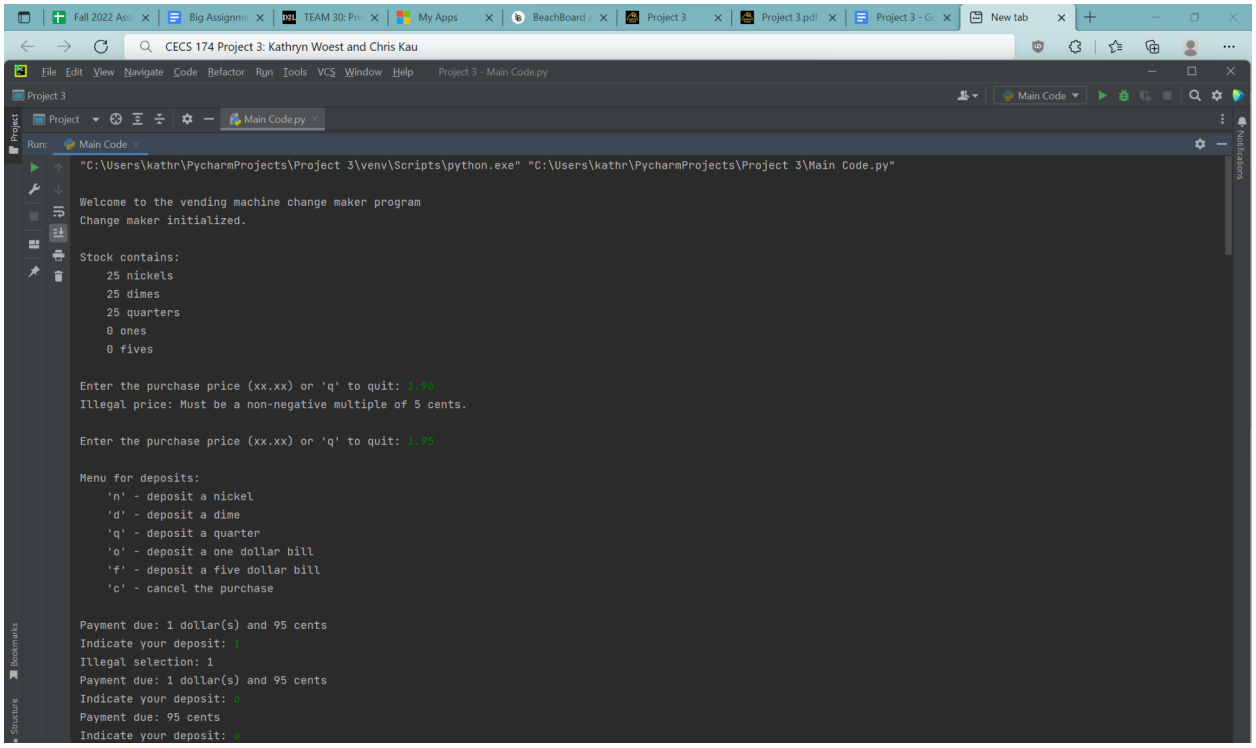
    d. **What are the improvements that the instructors and TAs might have done to promote learning? Please be specific.**

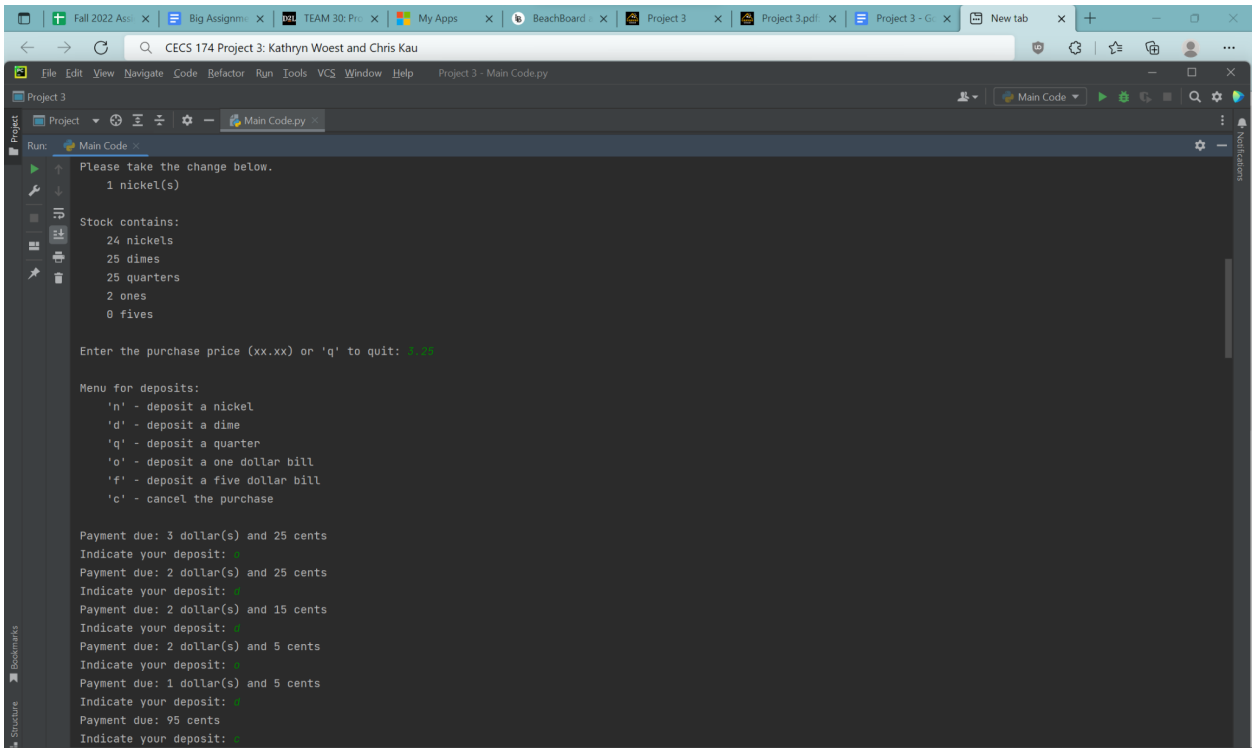
        i. **Chris:** Really the only thing I wish the instructors/TA's would improve upon to promote learning would be to let us start the project our way. An example would be doing a rough code first instead of the flowchart and pseudo code.

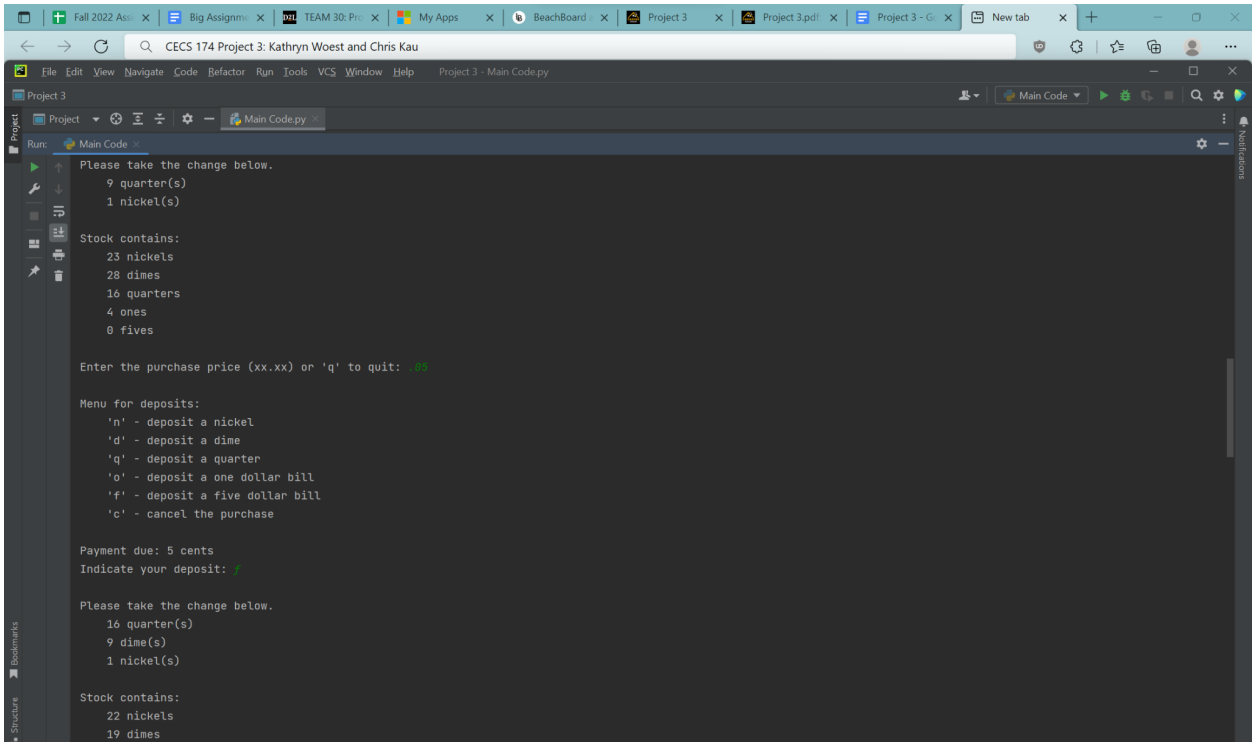
        ii. **Kathryn:** I would appreciate having groups set up prior to our lab meetings. Since our lab section is on Friday, without knowledge of who our partner is for the project before Friday the most we can do is just brainstorm since we don't want our work to deviate too much from our partner's work. We're left with only one week to do the entire flowchart, pseudocode, code, and report on top of other assignments, which is very difficult. If we happen to have a partner project in the future, releasing a group list early to let us contact each other and get started before the lab would be nice.

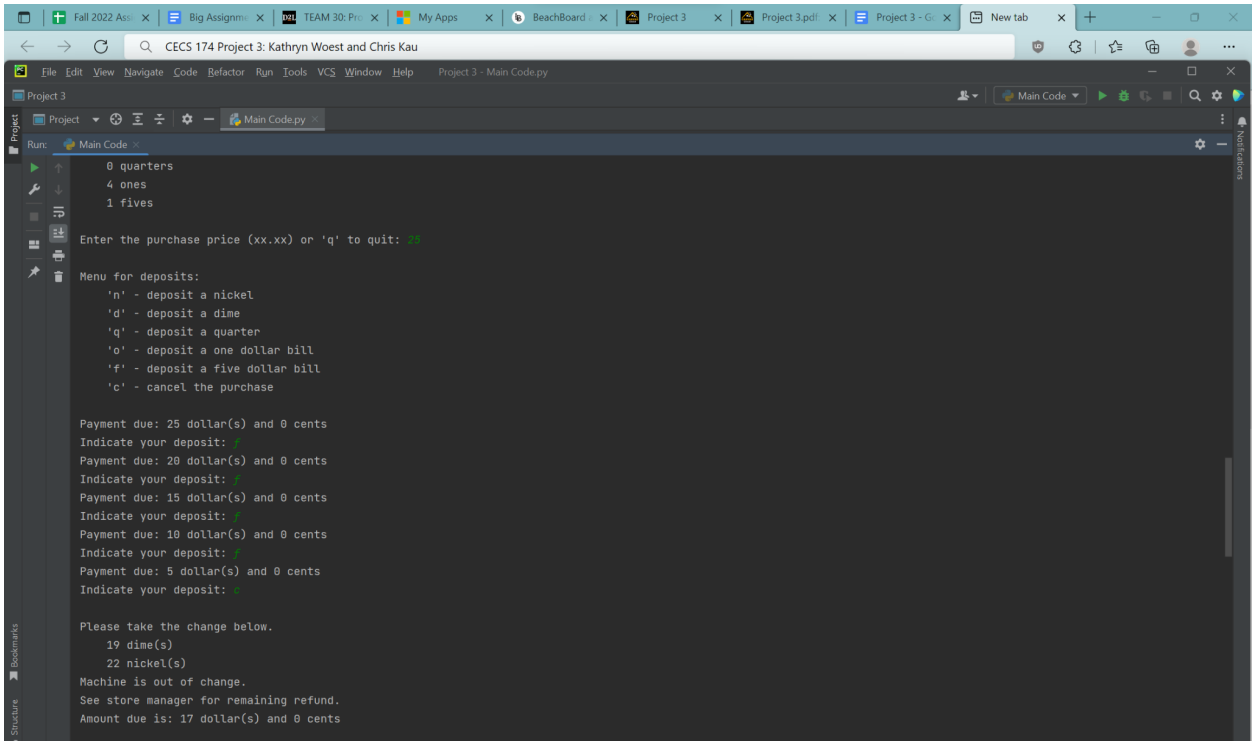
Appendix:

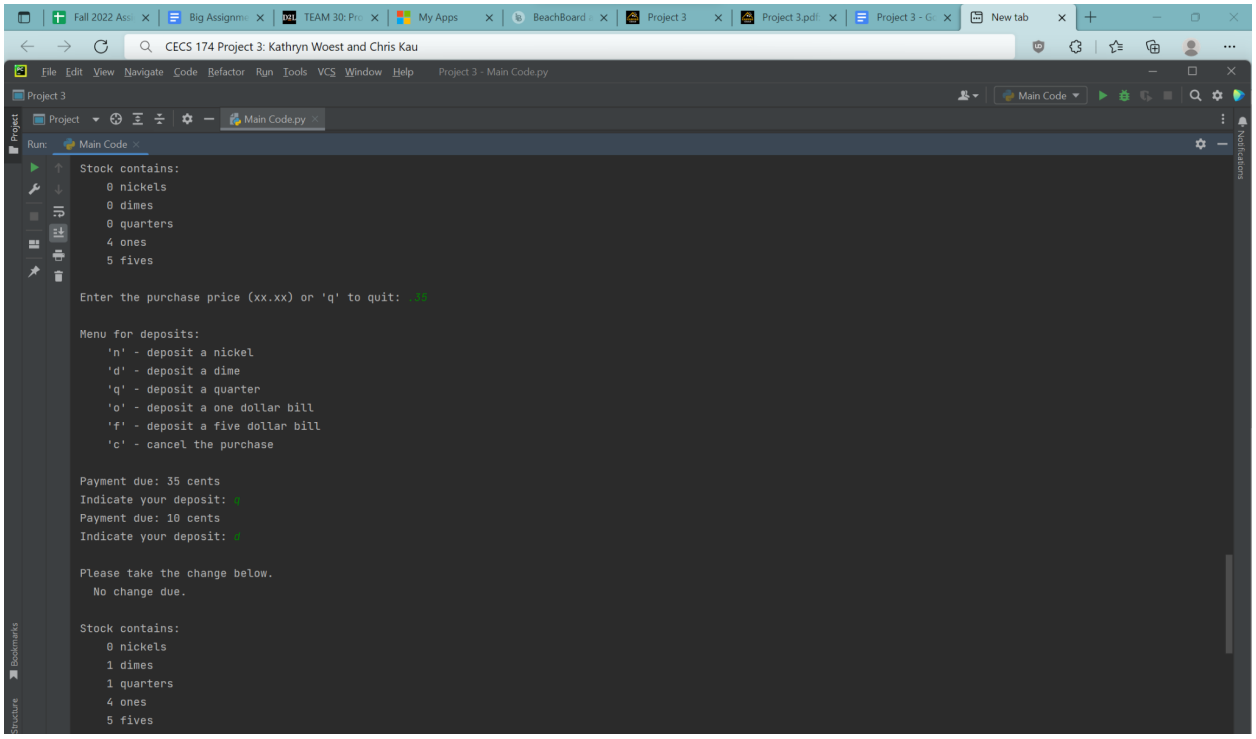
Project 3 Output:

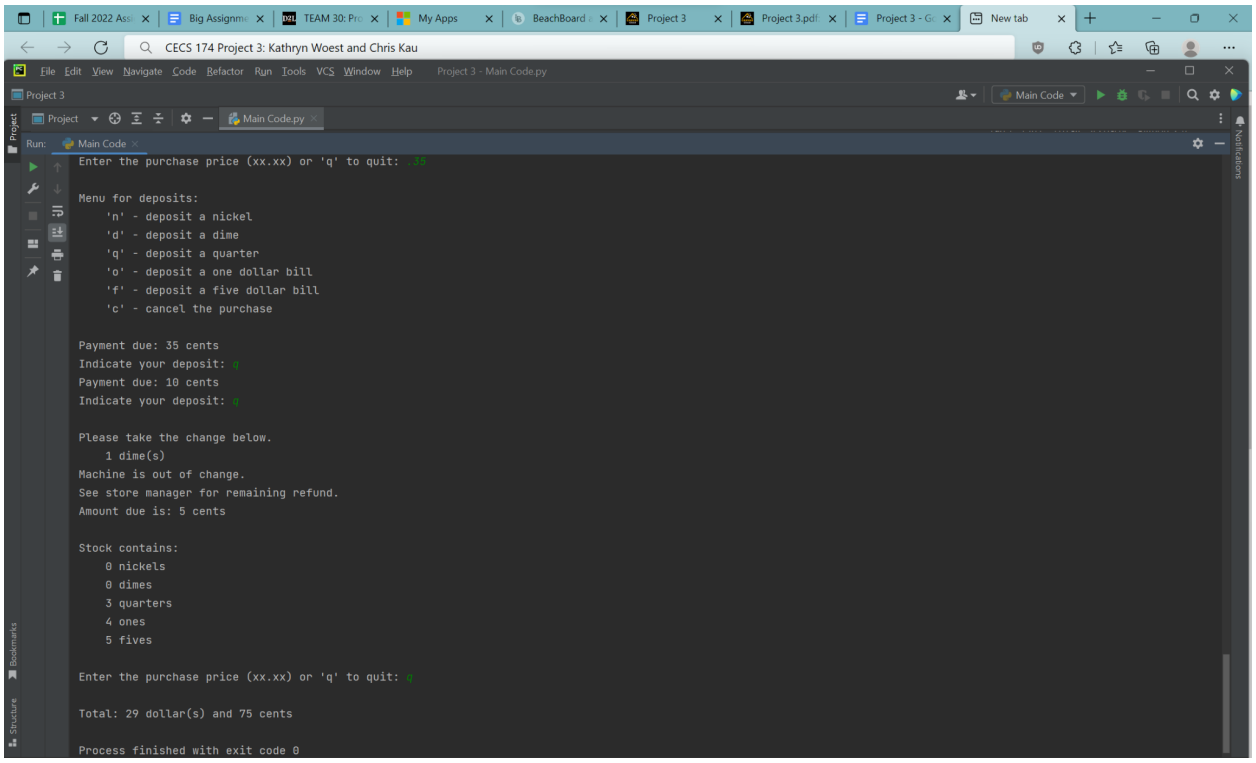




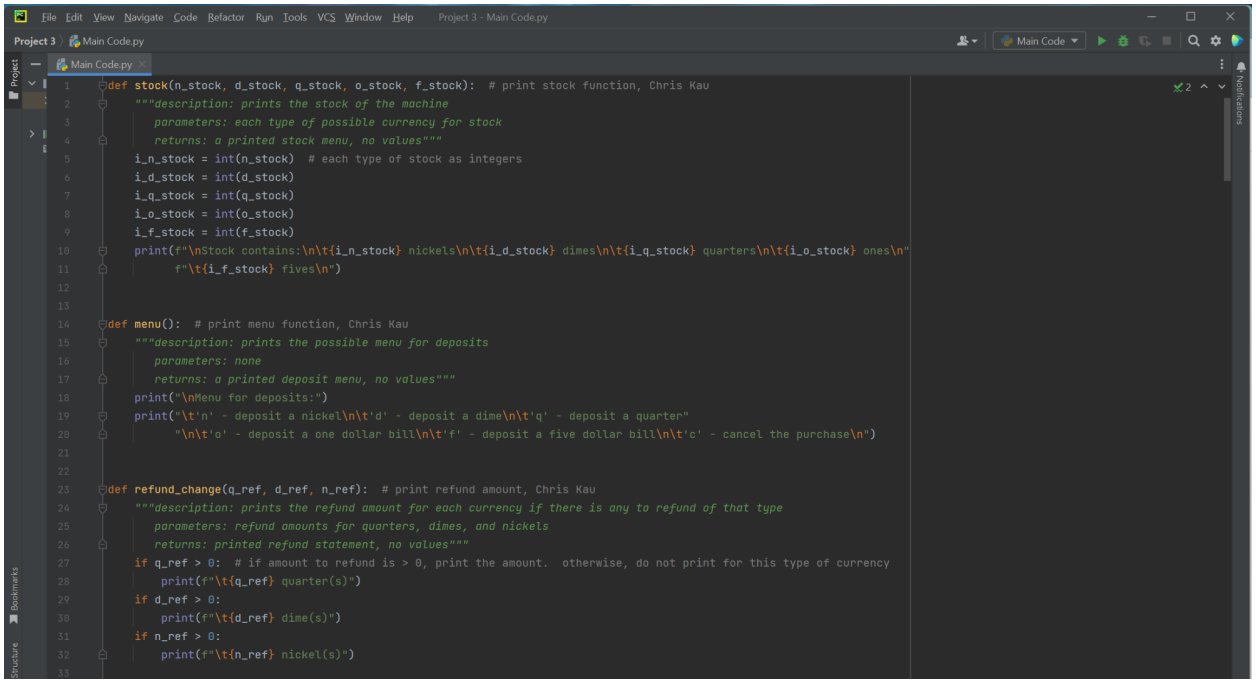

CECS 174 Project 3: Kathryn Woest and Chris Kau
Project 3 - Main Code.py
Main Code
Please take the change below.
9 quarter(s)
1 nickel(s)
Stock contains:
23 nickels
28 dimes
16 quarters
4 ones
0 fives
Enter the purchase price (xx.xx) or 'q' to quit: .85
Menu for deposits:
'n' - deposit a nickel
'd' - 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
Indicate your deposit: f
Please take the change below.
16 quarter(s)
9 dime(s)
1 nickel(s)
Stock contains:
22 nickels
19 dimes


CECS 174 Project 3: Kathryn Woest and Chris Kau
Project 3 - Main Code.py
Main Code
0 quarters
4 ones
1 fives
Enter the purchase price (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 dime(s)
22 nickel(s)
Machine is out of change.
See store manager for remaining refund.
Amount due is: 17 dollar(s) and 0 cents

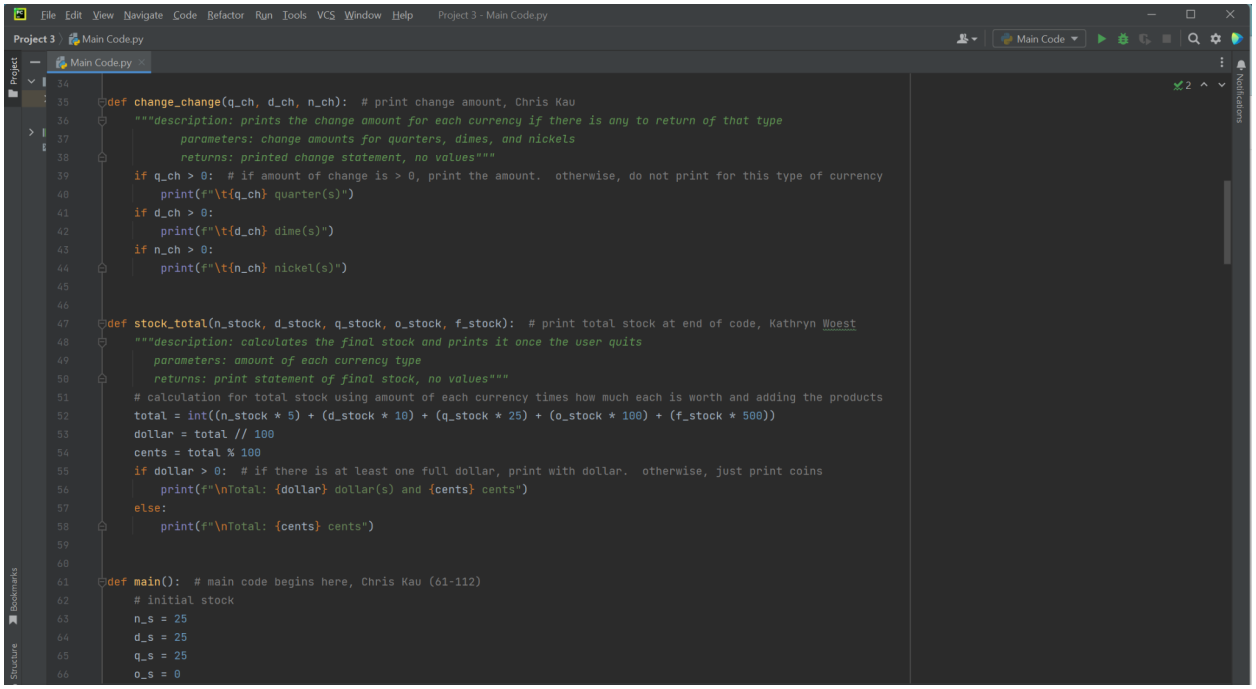

CECS 174 Project 3: Kathryn Woest and Chris Kau
Project 3 - Main Code.py
Main Code.py
Run: Main Code
Stock contains:
0 nickels
0 dimes
0 quarters
4 ones
5 fives
Enter the purchase price (xx.xx) or 'q' to quit: .35
Menu for deposits:
'n' - deposit a nickel
'd' - deposit a dime
'q' - deposit a quarter
'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
Please take the change below.
No change due.
Stock contains:
0 nickels
1 dimes
1 quarters
4 ones
5 fives


CECS 174 Project 3: Kathryn Woest and Chris Kau
Project 3 - Main Code.py
Main Code.py
Run: Main Code
Enter the purchase price (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 dime(s)
Machine is out of change.
See store manager for remaining refund.
Amount due is: 5 cents
Stock contains:
0 nickels
0 dimes
3 quarters
4 ones
5 fives
Enter the purchase price (xx.xx) or 'q' to quit: q
Total: 29 dollar(s) and 75 cents
Process finished with exit code 0

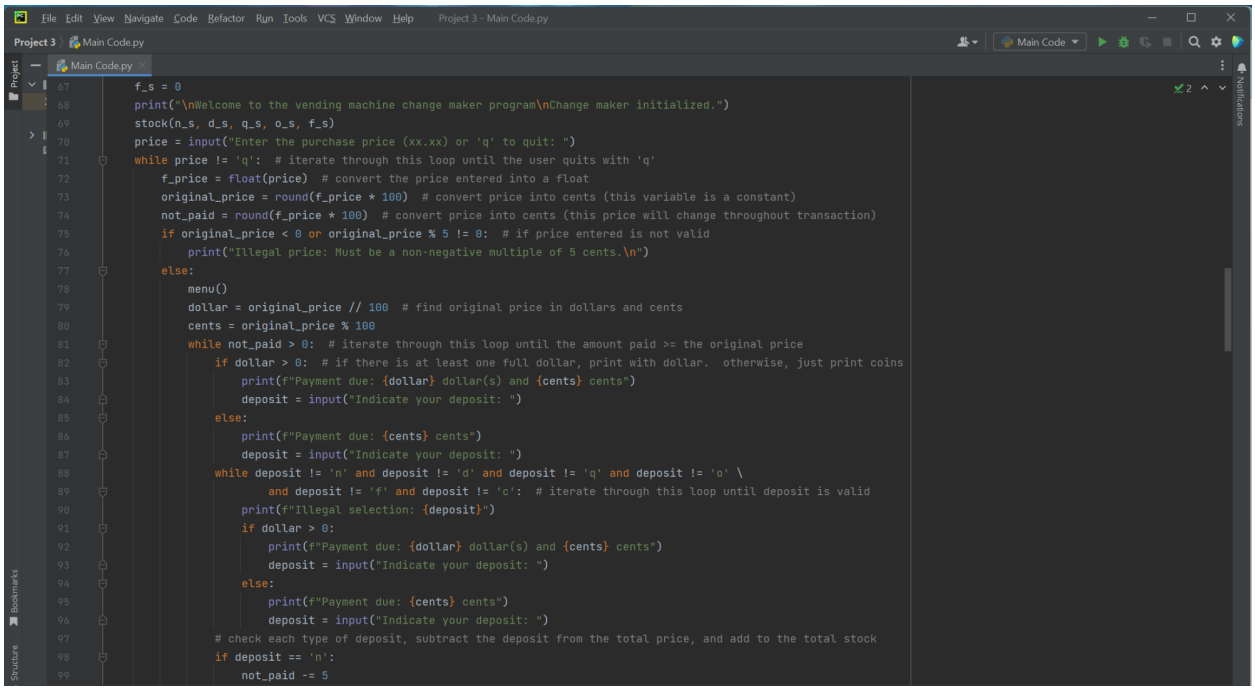
Project 3 Code:



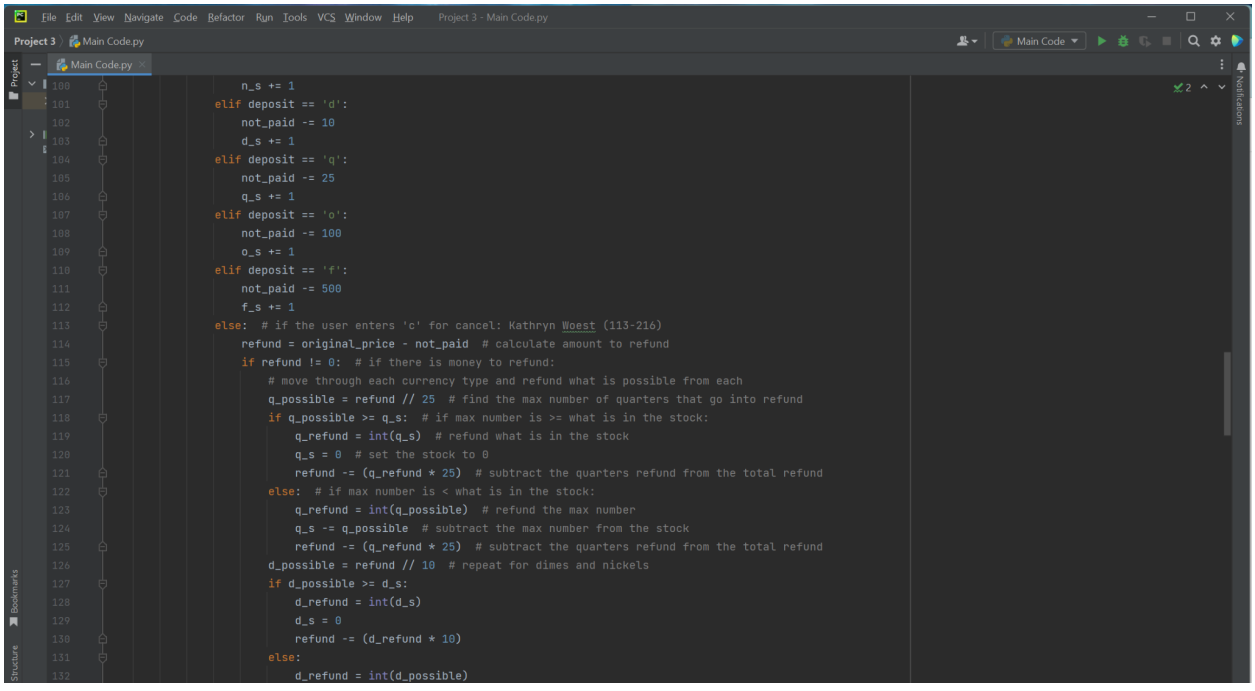
```
def stock(n_stock, d_stock, q_stock, o_stock, f_stock):  # print stock function, Chris Kau
    """description: prints the stock of the machine
       parameters: each type of possible currency for stock
       returns: a printed stock menu, no values"""
    i_n_stock = int(n_stock)  # each type of stock as integers
    i_d_stock = int(d_stock)
    i_q_stock = int(q_stock)
    i_o_stock = int(o_stock)
    i_f_stock = int(f_stock)
    print(f"\nStock contains:\n\t{i_n_stock} nickels\n\t{i_d_stock} dimes\n\t{i_q_stock} quarters\n\t{i_o_stock} ones\n"
          f"\t{i_f_stock} fives\n")


def menu():  # print menu function, Chris Kau
    """description: prints the possible menu for deposits
       parameters: none
       returns: a printed deposit menu, no values"""
    print("\nMenu for deposits:")
    print("\t'n' - deposit a nickel\n\t'd' - deposit a dime\n\t'q' - deposit a quarter"
          "\n\t'o' - deposit a one dollar bill\n\t'f' - deposit a five dollar bill\n\t'c' - cancel the purchase\n")


def refund_change(q_ref, d_ref, n_ref):  # print refund amount, Chris Kau
    """description: prints the refund amount for each currency if there is any to refund of that type
       parameters: refund amounts for quarters, dimes, and nickels
       returns: printed refund statement, no values"""
    if q_ref > 0:  # if amount to refund is > 0, print the amount.  otherwise, do not print for this type of currency
        print(f"\t{q_ref} quarter(s)")
    if d_ref > 0:
        print(f"\t{d_ref} dime(s)")
    if n_ref > 0:
        print(f"\t{n_ref} nickel(s)")
```



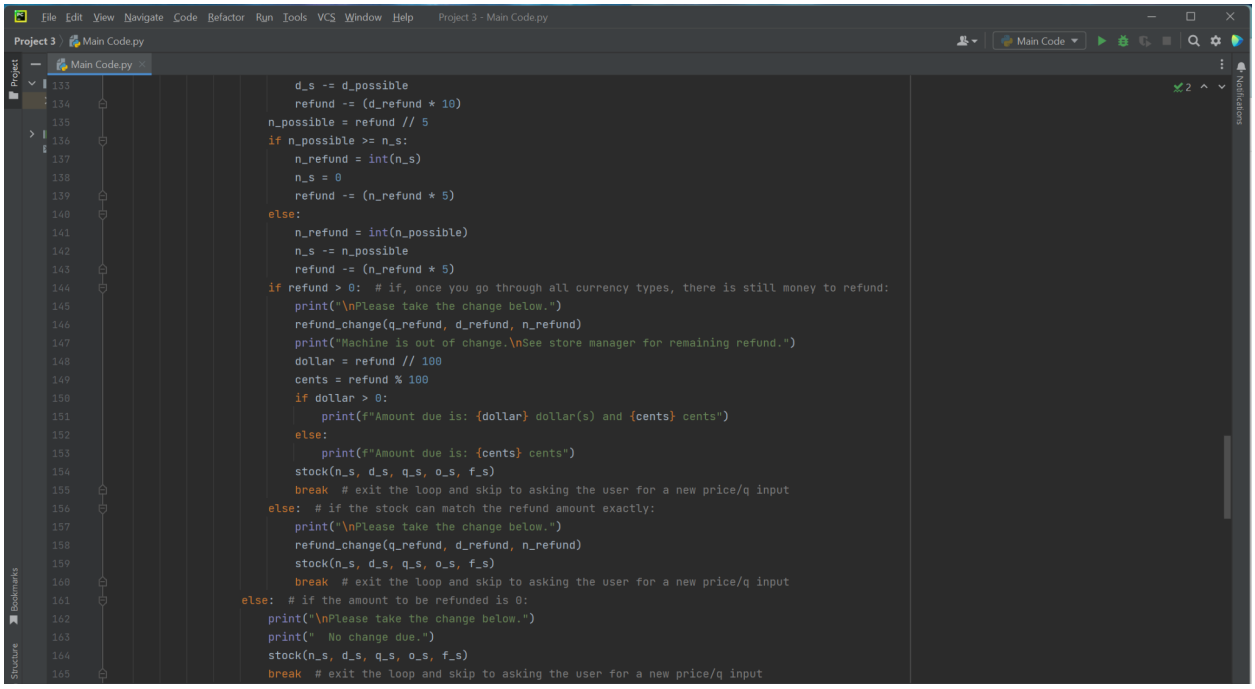
```
def change_change(q_ch, d_ch, n_ch):  # print change amount, Chris Kau
    """description: prints the change amount for each currency if there is any to return of that type
           parameters: change amounts for quarters, dimes, and nickels
           returns: printed change statement, no values"""
    if q_ch > 0:  # if amount of change is > 0, print the amount.  otherwise, do not print for this type of currency
        print(f"\t{q_ch} quarter(s)")
    if d_ch > 0:
        print(f"\t{d_ch} dime(s)")
    if n_ch > 0:
        print(f"\t{n_ch} nickel(s)")


def stock_total(n_stock, d_stock, q_stock, o_stock, f_stock):  # print total stock at end of code, Kathryn Woest
    """description: calculates the final stock and prints it once the user quits
       parameters: amount of each currency type
       returns: print statement of final stock, no values"""
    # calculation for total stock using amount of each currency times how much each is worth and adding the products
    total = int((n_stock * 5) + (d_stock * 10) + (q_stock * 25) + (o_stock * 100) + (f_stock * 500))
    dollar = total // 100
    cents = total % 100
    if dollar > 0:  # if there is at least one full dollar, print with dollar.  otherwise, just print coins
        print(f"\nTotal: {dollar} dollar(s) and {cents} cents")
    else:
        print(f"\nTotal: {cents} cents")


def main():  # main code begins here, Chris Kau (61-112)
    # initial stock
    n_s = 25
    d_s = 25
    q_s = 25
    o_s = 0
```



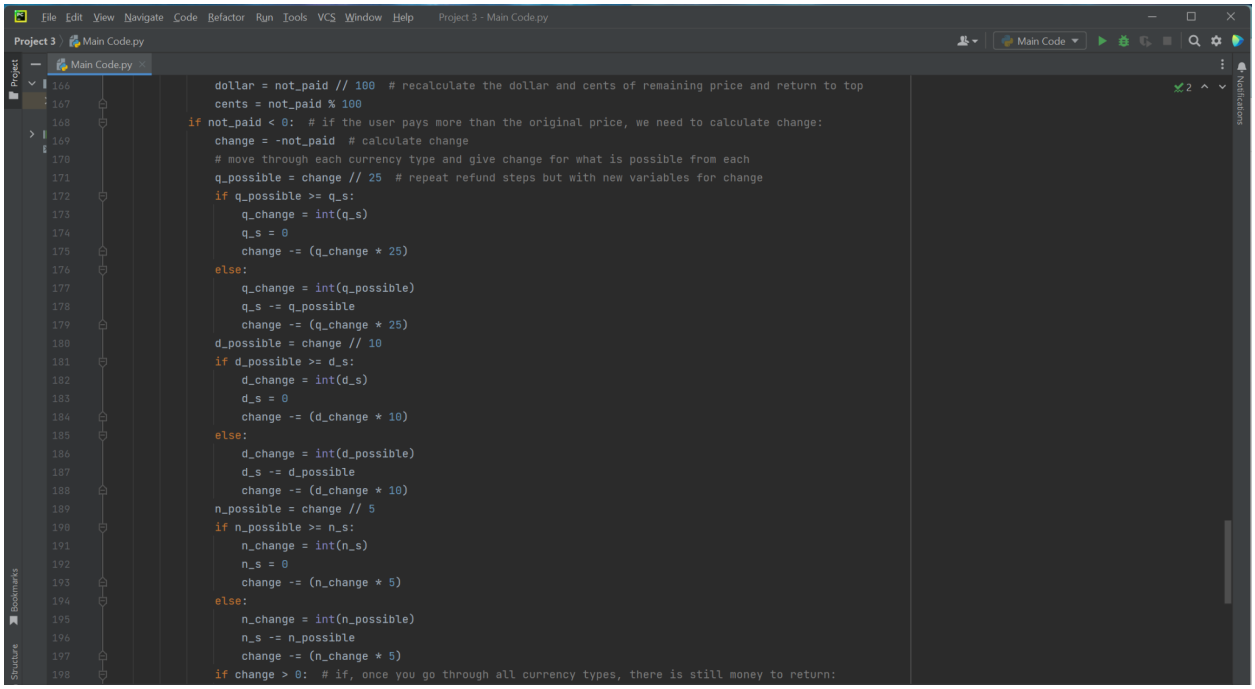
```
    f_s = 0
    print("\nWelcome to the vending machine change maker program\nChange maker initialized.")
    stock(n_s, d_s, q_s, o_s, f_s)
    price = input("Enter the purchase price (xx.xx) or 'q' to quit: ")
    while price != 'q':  # iterate through this loop until the user quits with 'q'
        f_price = float(price)  # convert the price entered into a float
        original_price = round(f_price * 100)  # convert price into cents (this variable is a constant)
        not_paid = round(f_price * 100)  # convert price into cents (this price will change throughout transaction)
        if original_price < 0 or original_price % 5 != 0:  # if price entered is not valid
            print("Illegal price: Must be a non-negative multiple of 5 cents.\n")
        else:
            menu()
            dollar = original_price // 100  # find original price in dollars and cents
            cents = original_price % 100
            while not_paid > 0:  # iterate through this loop until the amount paid >= the original price
                if dollar > 0:  # if there is at least one full dollar, print with dollar.  otherwise, just print coins
                    print(f"Payment due: {dollar} dollar(s) and {cents} cents")
                    deposit = input("Indicate your deposit: ")
                else:
                    print(f"Payment due: {cents} cents")
                    deposit = input("Indicate your deposit: ")
                while deposit != 'n' and deposit != 'd' and deposit != 'q' and deposit != 'o' \
                        and deposit != 'f' and deposit != 'c':  # iterate through this loop until deposit is valid
                    print(f"Illegal selection: {deposit}")
                    if dollar > 0:
                        print(f"Payment due: {dollar} dollar(s) and {cents} cents")
                        deposit = input("Indicate your deposit: ")
                    else:
                        print(f"Payment due: {cents} cents")
                        deposit = input("Indicate your deposit: ")
                # check each type of deposit, subtract the deposit from the total price, and add to the total stock
                if deposit == 'n':
                    not_paid -= 5
```



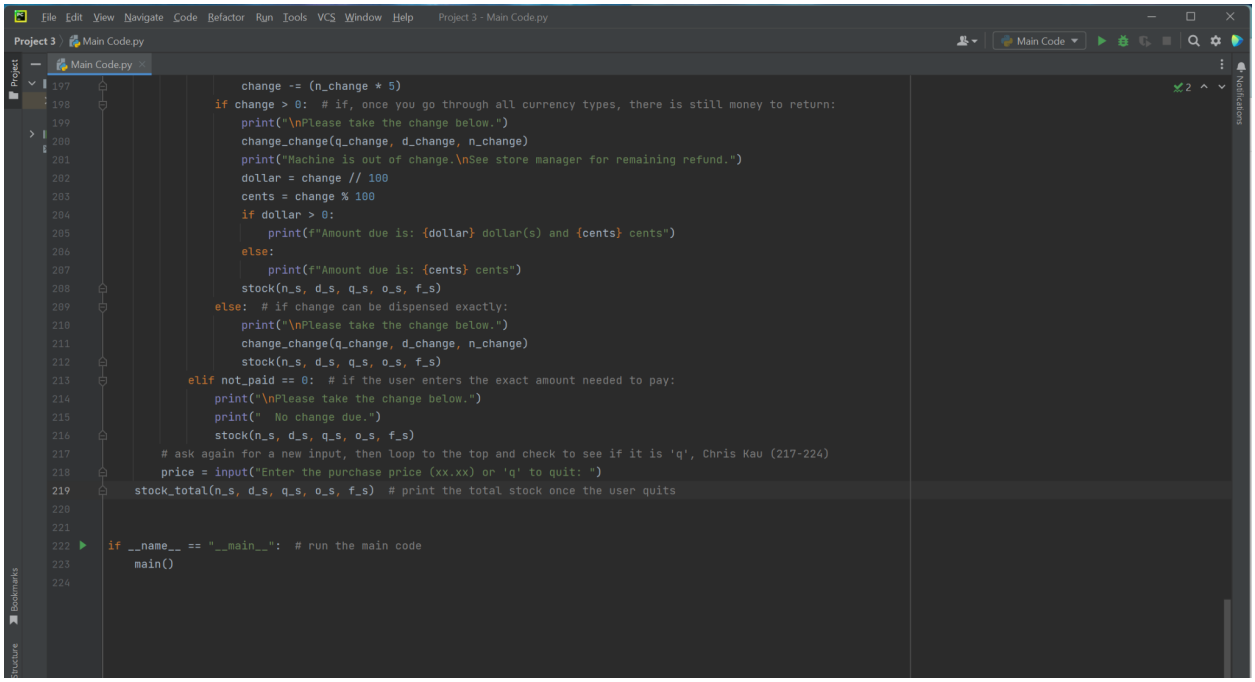
```
                    n_s += 1
                elif deposit == 'd':
                    not_paid -= 10
                    d_s += 1
                elif deposit == 'q':
                    not_paid -= 25
                    q_s += 1
                elif deposit == 'o':
                    not_paid -= 100
                    o_s += 1
                elif deposit == 'f':
                    not_paid -= 500
                    f_s += 1
                else:  # if the user enters 'c' for cancel: Kathryn Woest (113-216)
                    refund = original_price - not_paid  # calculate amount to refund
                    if refund != 0:  # if there is money to refund:
                        # move through each currency type and refund what is possible from each
                        q_possible = refund // 25  # find the max number of quarters that go into refund
                        if q_possible >= q_s:  # if max number is >= what is in the stock:
                            q_refund = int(q_s)  # refund what is in the stock
                            q_s = 0  # set the stock to 0
                            refund -= (q_refund * 25)  # subtract the quarters refund from the total refund
                        else:  # if max number is < what is in the stock:
                            q_refund = int(q_possible)  # refund the max number
                            q_s -= q_possible  # subtract the max number from the stock
                            refund -= (q_refund * 25)  # subtract the quarters refund from the total refund
                        d_possible = refund // 10  # repeat for dimes and nickels
                        if d_possible >= d_s:
                            d_refund = int(d_s)
                            d_s = 0
                            refund -= (d_refund * 10)
                        else:
                            d_refund = int(d_possible)
```



```
                    d_s -= d_possible
                    refund -= (d_refund * 10)
                n_possible = refund // 5
                if n_possible >= n_s:
                    n_refund = int(n_s)
                    n_s = 0
                    refund -= (n_refund * 5)
                else:
                    n_refund = int(n_possible)
                    n_s -= n_possible
                    refund -= (n_refund * 5)
                if refund > 0:  # if, once you go through all currency types, there is still money to refund:
                    print("\nPlease take the change below.")
                    refund_change(q_refund, d_refund, n_refund)
                    print("Machine is out of change.\nSee store manager for remaining refund.")
                    dollar = refund // 100
                    cents = refund % 100
                    if dollar > 0:
                        print(f"Amount due is: {dollar} dollar(s) and {cents} cents")
                    else:
                        print(f"Amount due is: {cents} cents")
                    stock(n_s, d_s, q_s, o_s, f_s)
                    break  # exit the loop and skip to asking the user for a new price/q input
                else:  # if the stock can match the refund amount exactly:
                    print("\nPlease take the change below.")
                    refund_change(q_refund, d_refund, n_refund)
                    stock(n_s, d_s, q_s, o_s, f_s)
                    break  # exit the loop and skip to asking the user for a new price/q input
            else:  # if the amount to be refunded is 0:
                print("\nPlease take the change below.")
                print("  No change due.")
                stock(n_s, d_s, q_s, o_s, f_s)
                break  # exit the loop and skip to asking the user for a new price/q input
```



```
            dollar = not_paid // 100  # recalculate the dollar and cents of remaining price and return to top
            cents = not_paid % 100
        if not_paid < 0:  # if the user pays more than the original price, we need to calculate change:
            change = -not_paid  # calculate change
            # move through each currency type and give change for what is possible from each
            q_possible = change // 25  # repeat refund steps but with new variables for change
            if q_possible >= q_s:
                q_change = int(q_s)
                q_s = 0
                change -= (q_change * 25)
            else:
                q_change = int(q_possible)
                q_s -= q_possible
                change -= (q_change * 25)
            d_possible = change // 10
            if d_possible >= d_s:
                d_change = int(d_s)
                d_s = 0
                change -= (d_change * 10)
            else:
                d_change = int(d_possible)
                d_s -= d_possible
                change -= (d_change * 10)
            n_possible = change // 5
            if n_possible >= n_s:
                n_change = int(n_s)
                n_s = 0
                change -= (n_change * 5)
            else:
                n_change = int(n_possible)
                n_s -= n_possible
                change -= (n_change * 5)
            if change > 0:  # if, once you go through all currency types, there is still money to return:
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project 3 - Main Code.py
Project 3 > Main Code.py
Main Code
Main Code.py
Project
Bookmarks
Structure
Notifications
2
change -= (n_change * 5)
if change > 0: # if, once you go through all currency types, there is still money to return:
print("\nPlease take the change below.")
change_change(q_change, d_change, n_change)
print("Machine is out of change.\nSee store manager for remaining refund.")
dollar = change // 100
cents = change % 100
if dollar > 0:
print(f"Amount due is: {dollar} dollar(s) and {cents} cents")
else:
print(f"Amount due is: {cents} cents")
stock(n_s, d_s, q_s, o_s, f_s)
else: # if change can be dispensed exactly:
print("\nPlease take the change below.")
change_change(q_change, d_change, n_change)
stock(n_s, d_s, q_s, o_s, f_s)
elif not_paid == 0: # if the user enters the exact amount needed to pay:
print("\nPlease take the change below.")
print(" No change due.")
stock(n_s, d_s, q_s, o_s, f_s)
# ask again for a new input, then loop to the top and check to see if it is 'q', Chris Kau (217-224)
price = input("Enter the purchase price (xx.xx) or 'q' to quit: ")
stock_total(n_s, d_s, q_s, o_s, f_s) # print the total stock once the user quits
if __name__ == "__main__": # run the main code
main()