PROJECT REPORT ON

“Billing-System”

Submitted By:

Ritish Chauhan, UID- 24MCA20305

Under The Guidance of:

Deepali Saini

OCTOBER, 2024

University Institute of Computing

Chandigarh University,

Mohali, Punjab

CERTIFICATE

This is to certify that Ritish Chauhan (UID- 24MCA20305) have successfully completed Project Report on title **"Billing-System"** at University Institute of Computing under my supervision and guidance in the fulfilment of requirements of first semester, **Master of Computer Applications.** Of Chandigarh University, Mohali, Punjab.

Dr. Abdullah	Deepali Saini
Head of the Department	Project Guide Supervisor
University Institute of Computing	University Institute of Computing

ACKNOWLEDGEMENT

I wish to express my sincere thanks to my project guide, Deepali Saini, for her guidance and support throughout the completion of this project. Her insightful suggestions helped me refine the project scope and encouraged me to delve deeper into the subject matter.

I also want to extend my gratitude to Dr. Abdullah (HOD, University Institute of Computing) for providing an academic atmosphere conducive to learning. Finally, I would like to thank my friends and family for their unwavering support throughout this project.

Date:22.10.2024

Place: Chandigarh University, Mohali, Punjab

Ritish Chauhan, UID- 24MCA20305

ABSTRACT

Technology has indeed changed most things in our lives, and business operations are not left out. The billing and invoicing sector has been one of the highly affected areas as it is now a manual system; instead, an automated one has taken the place of the former in many respects.

Among numerous innovations, the billing system comes as a tool that is based on Python. This efficiency enabled it to help in making the process of making an invoice easier. It eliminated human error by computing all items and summing up all to create a more comprehensive concise bill. It is all in the streamlining of the process meant to create the invoice. In turn, it saves time and much-needed resources for such a process with an assurance that other energies utilized effectively will be on critical activities around the business.

The billing system based on Python is also very flexible and customizable. Business can customize the system according to their needs in respect of discount, tax, and payment options; it is possible that integration does not affect efficiency as a whole.

Hence, the billing system that is based on Python is an excellent innovation in the process of the evolution of business automation. It makes billing a procedure that adds to accuracy through automation of the same. The customization option enables a business to optimize its functioning and provide excellent customer services. With advancing technology, we can very well expect innovative methods to come up in the form of further evolutions in the way in which businesses handle their finances.

TABLE OF CONTENTS

Introduction

Technology has really changed the way of many things in our lives. Business operations are also one of them. Probably the most affected area from this kind of change is the way of billing and issuing invoices. The old system that was manual has now been replaced by automated systems with many benefits.

One such innovation is the billing system using Python, which is a strong and efficient tool for making billing processes easier. It collects all the details of the items, calculates the total amount, and provides a very clear and concise bill. This saves much effort and reduces the scope of errors. With this removal of human error and fastening of the billing process, it gives the business time and resources saved to use in something else in their operation.

Background

Two things about the very competitive business environment that is ongoing today relate to efficiency and accuracy. Most traditional manual billing systems do not live up to such demands, thereby requiring lengthy processes, human mistakes, and delayed payments. In this regard, therefore, automated billing solutions have been increasingly applied in such endeavors. Generally, the whole system increases general operating efficiency by simplifying all processes of billing from an invoice generation process up to payment tracking.

This is one of the automated systems based on Python programming related to billing. It works to create a flexible, customizable billing platform by putting the power of Python to work. Routine functions, including total calculation and application of taxes, among others, are automated here. This allows businesses' precious time to be liberated so that more time may be devoted to core activities. It also reduces the error possibilities due to human mistakes, and accurate and timely billing would be permitted.

Problem Statement

Traditional manual billing systems usually have a few shortcomings:

- Processes are very time consuming, as such practices include several tasks, for example, manual entry, calculations, and billing through invoices in case a firm has many transactions occurring frequently.
- Human error: In most instances, errors can result because of manual calculations and inputting data, hence error billing to customers besides hurting the financial side.

- Less Efficiency: The manual systems and processes do not carry the time efficiency as well because the techniques cannot be performed fast enough to serve the businesses nowadays.
- It is difficult to trace and analyze: the manual systems are a bit complex in tracing sales, analyzing customer behavior, and making insightful reports.
- Such issues call for an effective and efficient billing system to manage such processes so that errors can be avoided and, overall, business processes improved.

Objectives

This is done in Python mainly based on automating the bill, making it more efficient and error-free. A specific aim of this system is that:

- Automatic Entry of data: Manual entries should be minimized whereby a user can input an item detail, number of items, or price in the system directly.
- Totals and Taxes Calculate automatically subtotals, taxes, and grand totals depending on the predefined rates and item prices.
- Proforma invoices with proper cost itemization, price, and customer details are printed.
- It maintains the sales in addition to inventory levels as well as other information to support further analysis and reporting
- The customer's satisfaction and loyalty to the company will improve due to timely and accurate billing.
- For such an activity, reduced time and efforts lead to cost cutting in the manual billing operations.

Literature Review

Existing Billing System

Now available in the market are different types of billing systems which either make use of a manual system or computerized system. In this case, the conventional system relies on paper-based invoices that are calculated by hand and prone to many errors as well as quite time-consuming. In this case, an automated system of billing makes use of a computer and software running through all these processes. It can also be at different complexities, such as from a simple solution based on spreadsheets to quite complex enterprise resource planning.

Python as a Language

Python is a general-purpose language that has gained so much popularity for developing so many applications. Among these, the billing systems became highly popular lately. This language is extremely readable, making it easy to understand. It is excellent to develop efficient and user-friendly software, rich in libraries to develop a robust scalable billing solution like Tkinter in GUI development and NumPy for numerical computations.

Python Libraries for GUI Development

Tkinter is one of the most popular Python libraries when it comes to graphical user interfaces. It makes possible the development of really intuitive and quite easy-to-implement user interfaces. It simplifies developing an interactive, user-friendly system of billing. Developers, while using Tkinter, will be able to create all varieties of UI elements: buttons, labels, text boxes, menus - so that users could work with your program without a hitch.

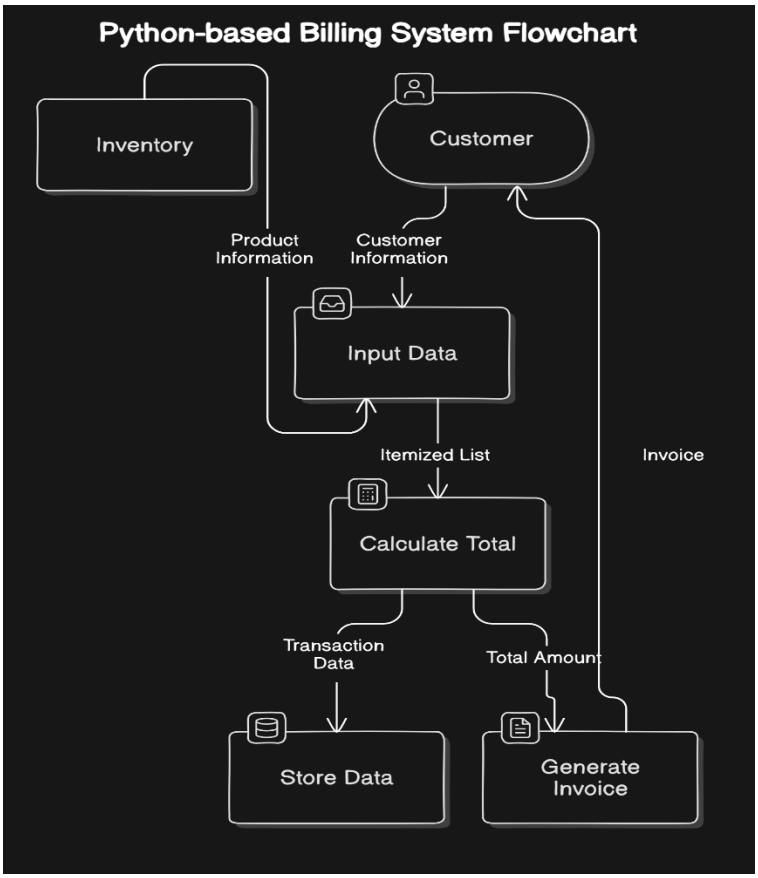
System Design

System Architecture

The billing system will be based on Python with a modular architecture having the following important components:

- User Interface. User-friendly GUI allows users to interact with the system, input data, and view invoices.
- Data Processing Module: This module performs inputting, validation, and calculating totals and taxes.
- Invoice Generation Module: This module generates professional-looking invoices in print format with detail information about items with prices and the customers' information.
- Database: Database will contain a number of information related to the customer, products, and sales data.

Data Flow Diagram



User Interface Design

The user interface explains itself. The design of its portion includes the following:

- Customer Information-Fields for Customer's name, Phone no, and Billing Address.
- Product Details Fields where one may input product name, quantity and price.
- Calculation Area Displays the Sub Total, Tax Amount, and Total
- Preview Space Preview invoice before its generation in order to be printed or saved
- Add, Remove, Calculate Totals, Generate Invoices, Exit.

Implementation

Python Programming

The billing system is implemented by using the core concepts and libraries in Python programming such as Tkinter for GUI. It is divided into various functions and modules that do a given set of tasks:

User Interface:

Tkinter is used to develop an interactive GUI with input fields for customer details, product information, and results of calculations. Buttons are applied to trigger actions such as adding items, removing items, calculating totals, and generating an invoice.

There should be functions that validate user inputs and compute the total quantity of all items bought based on the quantities purchased and rates.

Computing tax at correct tax percentages

There should be an effective use of data structures for keeping relevant data in proper lists and dictionaries, if applicable

Invoice Generation:

Develop a function that is good at generating an invoice into a desired format by using a number for customer ID, items details, quantity and price, tax and a total payable amount

The invoice is printed onto the screen or printer.

Development of GUI

The GUI uses Tkinter, which is a library for Python that helps design graphical user interfaces. Here are the main components used in the GUI:

- Labels: The GUI displays textual information such as "Customer Name," "Product Name, and "Total Amount".
- Entry Fields: The GUI allows the user to add text like the customer's name, product name, and quantity.
- Buttons: The GUI triggers all the actions including adding items, removing items, calculation of the total, and generation of the invoice.
- Text Box: Display the invoice that would follow. The UI is intuitive, user-friendly with good labels, and controls are easy to use.

Testing and Evaluation

Unit Testing

Unit testing is all about testing individual functions and modules of the system. Thus, in case of billing system, following can be written as unit tests of the system:

- Validate input: Confirm that it is validating the user inputs correctly so that no bad entry is done by the users.
- Calculation correctness: Determine whether the subtotal, taxes and grand total computations are being done correctly.
- Invoice generation: The system is generating invoices accurately formatted and information.
- Database operations: Testing proper insertion, retrieval, and updating within the database.

Integration Testing

This is checking the connectivity of the components of a system and how they work as one. The following are some of the integration tests that can be conducted for a billing system:

- User interface and data processing: It should guarantee proper processing of correct input from users in the computation as well as the output while generating an invoice.
- Data Base Interaction: Verify whether the system accepts and retrieves data stored in a database.
- Invoice generation with display: Validate that the system generates the correct invoice and it displays.

User Acceptance Testing

UAT: This is a system by actual end-users for the testing of checking the usability, functionality of the system, and performance as an entire. The following are the user acceptance testing criteria in regard to the above aspects:

- User interface: Is this an intuitive and user-friendly interface?
- Functionality: Do all the required actions the system perform correctly?
- Performance: Is this system response and efficient?
- Error handling: Does the system handle errors well and give information?

Results and Discussion

System Performance

The billing system featured high performance in efficiency as well as accuracy and, of course, friendliness to users. It's because such a system manages to handle as many transactions as possible resulting in time-wasting reductions and error prone reductions. Users, on their part, enjoy use of the very user interface since they manage to put in the data then get their invoice within minimal time spent.

User Feeding

The first user feed, however, is positive to the system. The usability of the system has come out to be simple, the accuracy of the system will be high, and even the time-saving capabilities of it are appreciated by the user. The benefits appreciated for the users are as given below:

- Less manual work: Calculation automation and preparation of invoices reduce the respective time and efforts that normally go into billing activities
- Accuracy: The calculations on the system are accurate, thereby validating the data entered reduces errors and ensures proper billings.
- Improved efficiency: The streamlined workflow and quick generation of invoices improve overall operational efficiency.
- Professional invoices: The system produces professional-looking invoices, which improves the image of the business.

Conclusion

Summary of Findings

The billing system in Python has automated the bill generation process and brought along efficiency and accuracy. In this regard, the acceptance of the system was soundly established because of easy interfaces, robust data handling, and reliable invoice creation. The system has demonstrated very high potential in diminishing all sorts of manual effort as well as reducing mistakes and enhancing general business transactions.

Future Work

This current system can be seen as a base framework for an efficient billing application but still is far from being ideal in many aspects.

Features Enhancement: The incorporation of more features, like inventory management, CRM, and payment gateways, improves the capability of the system.

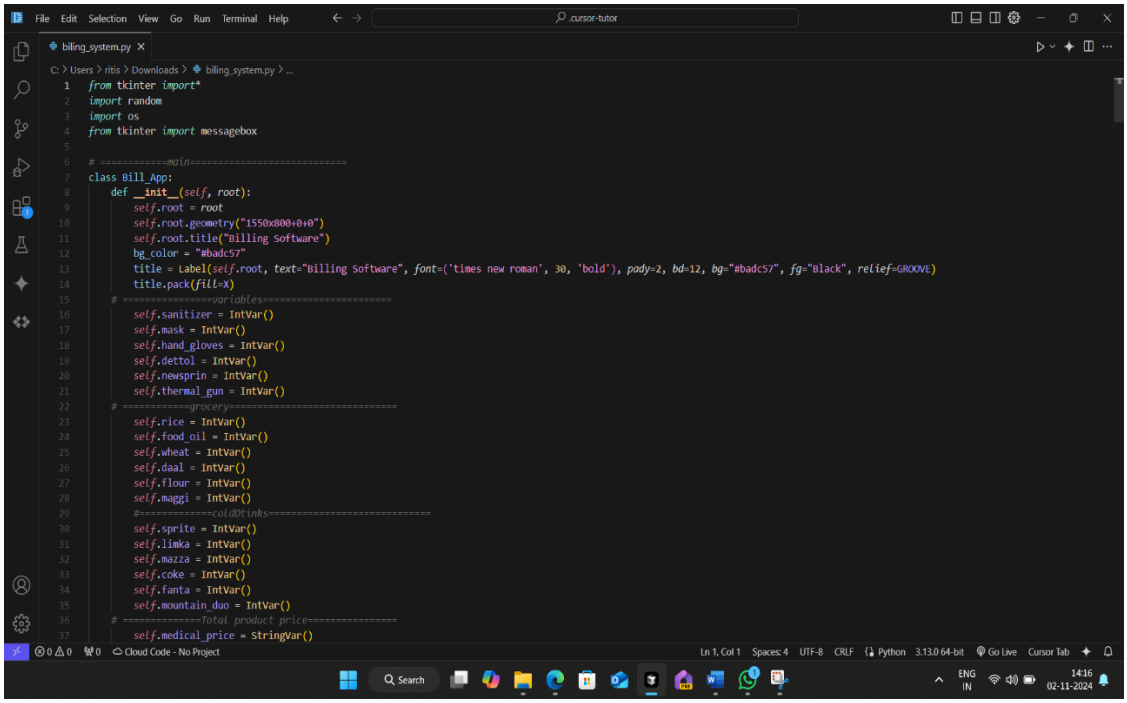
Scalability: The existing system needs to scale up pertaining to increased data sets and more users. This can be optimized by maintaining an optimum database and through efficient algorithms

Security: Ensure that appropriate security measures are implemented so that customer as well as financial information is safe with encryption and secure authentication.

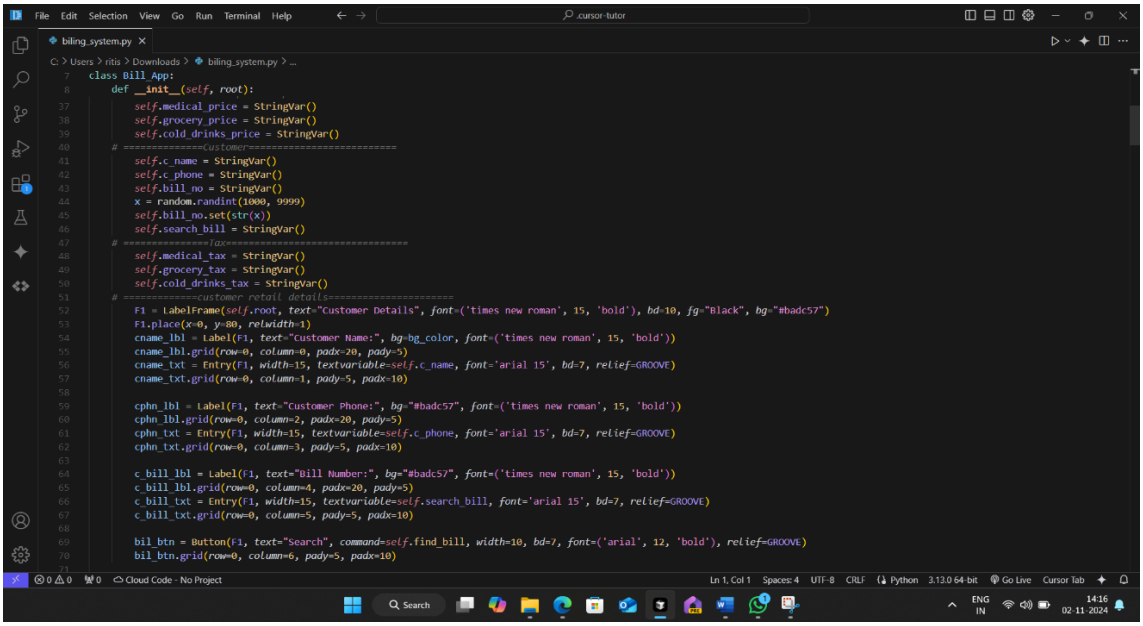
Deploy on Cloud: It needs to be tried to be deployed on cloud platforms for scalability benefits and higher access.

Mobile Compatibility: It needs to be developed so that it will work on mobile. The users need to be accessed from anywhere. There should also be a remote billing option.

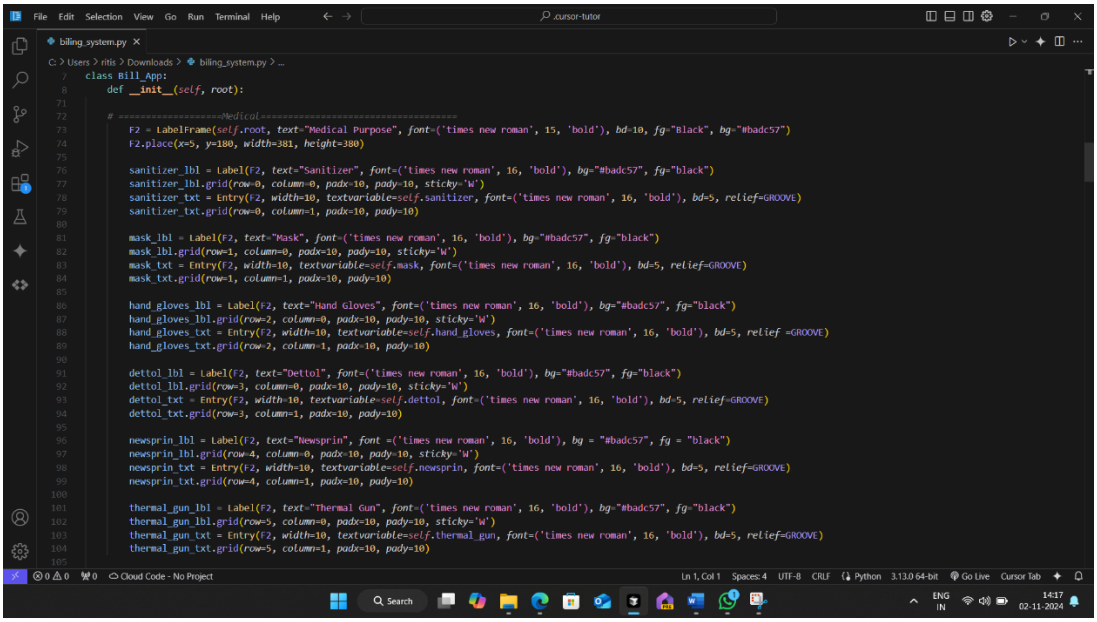
Appendix



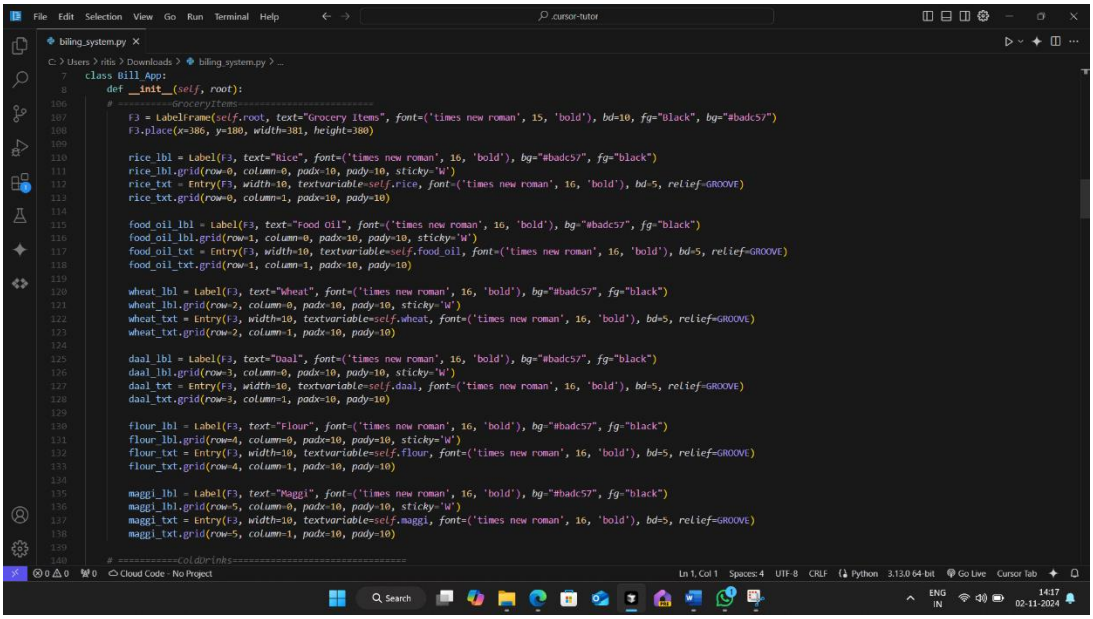
```python
from tkinter import*
import random
import os
from tkinter import messagebox

# ===========main=========================
class Bill_App:
    def __init__(self, root):
        self.root = root
        self.root.geometry("1550x800+0+0")
        self.root.title("Billing Software")
        bg_color = "#badc57"
        title = Label(self.root, text="Billing Software", font=('times new roman', 30, 'bold'), pady=2, bd=12, bg="#badc57", fg="Black", relief=GROOVE)
        title.pack(fill=X)
    # ===============variables=====================
        self.sanitizer = IntVar()
        self.mask = IntVar()
        self.hand_gloves = IntVar()
        self.dettol = IntVar()
        self.newsprin = IntVar()
        self.thermal_gun = IntVar()
    # ==========grocery============================
        self.rice = IntVar()
        self.food_oil = IntVar()
        self.wheat = IntVar()
        self.daal = IntVar()
        self.flour = IntVar()
        self.maggi = IntVar()
        #============coldDtinks=========================
        self.sprite = IntVar()
        self.limka = IntVar()
        self.mazza = IntVar()
        self.coke = IntVar()
        self.fanta = IntVar()
        self.mountain_duo = IntVar()
    # =============Total product price===============
        self.medical_price = StringVar()
```



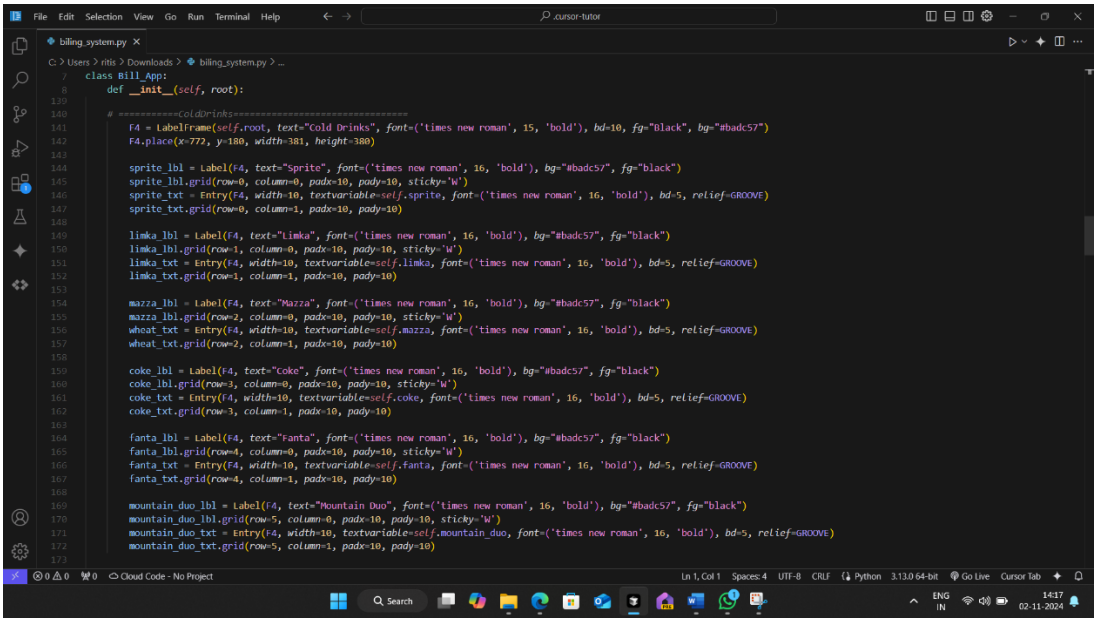
```python
class Bill_App:
    def __init__(self, root):
        self.medical_price = StringVar()
        self.grocery_price = StringVar()
        self.cold_drinks_price = StringVar()
    # =============Customer==========================
        self.c_name = StringVar()
        self.c_phone = StringVar()
        self.bill_no = StringVar()
        x = random.randint(1000, 9999)
        self.bill_no.set(str(x))
        self.search_bill = StringVar()
    # ==============tax=============================
        self.medical_tax = StringVar()
        self.grocery_tax = StringVar()
        self.cold_drinks_tax = StringVar()
    # ==========customer retail details=============
        F1 = LabelFrame(self.root, text="Customer Details", font=('times new roman', 15, 'bold'), bd=10, fg="Black", bg="#badc57")
        F1.place(x=0, y=80, relwidth=1)
        cname_lbl = Label(F1, text="Customer Name:", bg=bg_color, font=('times new roman', 15, 'bold'))
        cname_lbl.grid(row=0, column=0, padx=20, pady=5)
        cname_txt = Entry(F1, width=15, textvariable=self.c_name, font='arial 15', bd=7, relief=GROOVE)
        cname_txt.grid(row=0, column=1, pady=5, padx=10)

        cphn_lbl = Label(F1, text="Customer Phone:", bg="#badc57", font=('times new roman', 15, 'bold'))
        cphn_lbl.grid(row=0, column=2, padx=20, pady=5)
        cphn_txt = Entry(F1, width=15, textvariable=self.c_phone, font='arial 15', bd=7, relief=GROOVE)
        cphn_txt.grid(row=0, column=3, pady=5, padx=10)

        c_bill_lbl = Label(F1, text="Bill Number:", bg="#badc57", font=('times new roman', 15, 'bold'))
        c_bill_lbl.grid(row=0, column=4, padx=20, pady=5)
        c_bill_txt = Entry(F1, width=15, textvariable=self.search_bill, font='arial 15', bd=7, relief=GROOVE)
        c_bill_txt.grid(row=0, column=5, pady=5, padx=10)

        bil_btn = Button(F1, text="Search", command=self.find_bill, width=10, bd=7, font=('arial', 12, 'bold'), relief=GROOVE)
        bil_btn.grid(row=0, column=6, pady=5, padx=10)
```



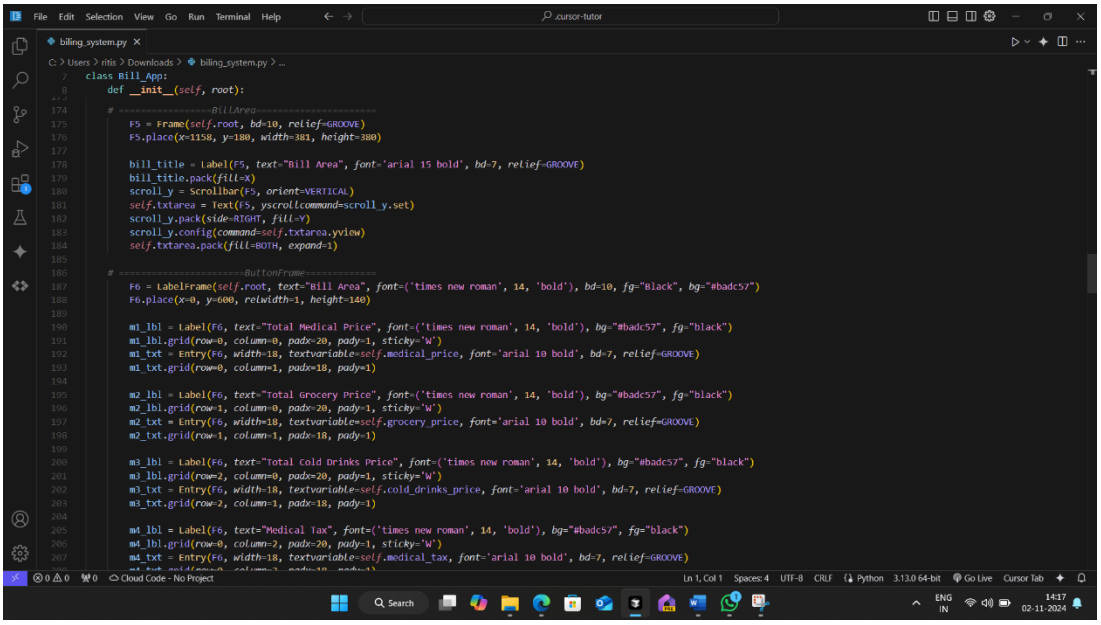
```
class Bill_App:
    def __init__(self, root):

    # ==================Medical=====================================
        F2 = LabelFrame(self.root, text="Medical Purpose", font=('times new roman', 15, 'bold'), bd=10, fg="Black", bg="#badc57")
        F2.place(x=5, y=180, width=381, height=380)

        sanitizer_lbl = Label(F2, text="Sanitizer", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        sanitizer_lbl.grid(row=0, column=0, padx=10, pady=10, sticky='W')
        sanitizer_txt = Entry(F2, width=10, textvariable=self.sanitizer, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        sanitizer_txt.grid(row=0, column=1, padx=10, pady=10)

        mask_lbl = Label(F2, text="Mask", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        mask_lbl.grid(row=1, column=0, padx=10, pady=10, sticky='W')
        mask_txt = Entry(F2, width=10, textvariable=self.mask, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        mask_txt.grid(row=1, column=1, padx=10, pady=10)

        hand_gloves_lbl = Label(F2, text="Hand Gloves", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        hand_gloves_lbl.grid(row=2, column=0, padx=10, pady=10, sticky='W')
        hand_gloves_txt = Entry(F2, width=10, textvariable=self.hand_gloves, font=('times new roman', 16, 'bold'), bd=5, relief =GROOVE)
        hand_gloves_txt.grid(row=2, column=1, padx=10, pady=10)

        dettol_lbl = Label(F2, text="Dettol", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        dettol_lbl.grid(row=3, column=0, padx=10, pady=10, sticky='W')
        dettol_txt = Entry(F2, width=10, textvariable=self.dettol, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        dettol_txt.grid(row=3, column=1, padx=10, pady=10)

        newsprin_lbl = Label(F2, text="Newsprin", font =('times new roman', 16, 'bold'), bg = "#badc57", fg = "black")
        newsprin_lbl.grid(row=4, column=0, padx=10, pady=10, sticky='W')
        newsprin_txt = Entry(F2, width=10, textvariable=self.newsprin, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        newsprin_txt.grid(row=4, column=1, padx=10, pady=10)

        thermal_gun_lbl = Label(F2, text="Thermal Gun", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        thermal_gun_lbl.grid(row=5, column=0, padx=10, pady=10, sticky='W')
        thermal_gun_txt = Entry(F2, width=10, textvariable=self.thermal_gun, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        thermal_gun_txt.grid(row=5, column=1, padx=10, pady=10)
```



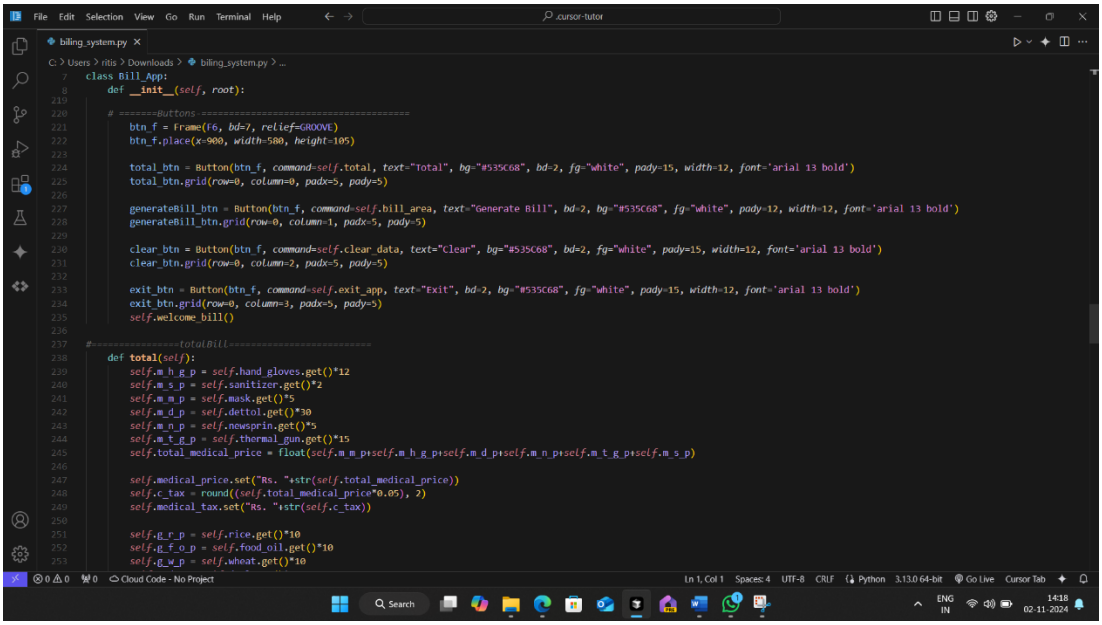
```
class Bill_App:
    def __init__(self, root):
    # =========Groceryitems=========================
        F3 = LabelFrame(self.root, text="Grocery items", font=('times new roman', 15, 'bold'), bd=10, fg="Black", bg="#badc57")
        F3.place(x=386, y=180, width=381, height=380)

        rice_lbl = Label(F3, text="Rice", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        rice_lbl.grid(row=0, column=0, padx=10, pady=10, sticky='W')
        rice_txt = Entry(F3, width=10, textvariable=self.rice, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        rice_txt.grid(row=0, column=1, padx=10, pady=10)

        food_oil_lbl = Label(F3, text="Food Oil", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        food_oil_lbl.grid(row=1, column=0, padx=10, pady=10, sticky='W')
        food_oil_txt = Entry(F3, width=10, textvariable=self.food_oil, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        food_oil_txt.grid(row=1, column=1, padx=10, pady=10)

        wheat_lbl = Label(F3, text="Wheat", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        wheat_lbl.grid(row=2, column=0, padx=10, pady=10, sticky='W')
        wheat_txt = Entry(F3, width=10, textvariable=self.wheat, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        wheat_txt.grid(row=2, column=1, padx=10, pady=10)

        daal_lbl = Label(F3, text="Daal", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        daal_lbl.grid(row=3, column=0, padx=10, pady=10, sticky='W')
        daal_txt = Entry(F3, width=10, textvariable=self.daal, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        daal_txt.grid(row=3, column=1, padx=10, pady=10)

        flour_lbl = Label(F3, text="Flour", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        flour_lbl.grid(row=4, column=0, padx=10, pady=10, sticky='W')
        flour_txt = Entry(F3, width=10, textvariable=self.flour, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        flour_txt.grid(row=4, column=1, padx=10, pady=10)

        maggi_lbl = Label(F3, text="Maggi", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
        maggi_lbl.grid(row=5, column=0, padx=10, pady=10, sticky='W')
        maggi_txt = Entry(F3, width=10, textvariable=self.maggi, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
        maggi_txt.grid(row=5, column=1, padx=10, pady=10)

    # ==========ColdDrinks=================================
```



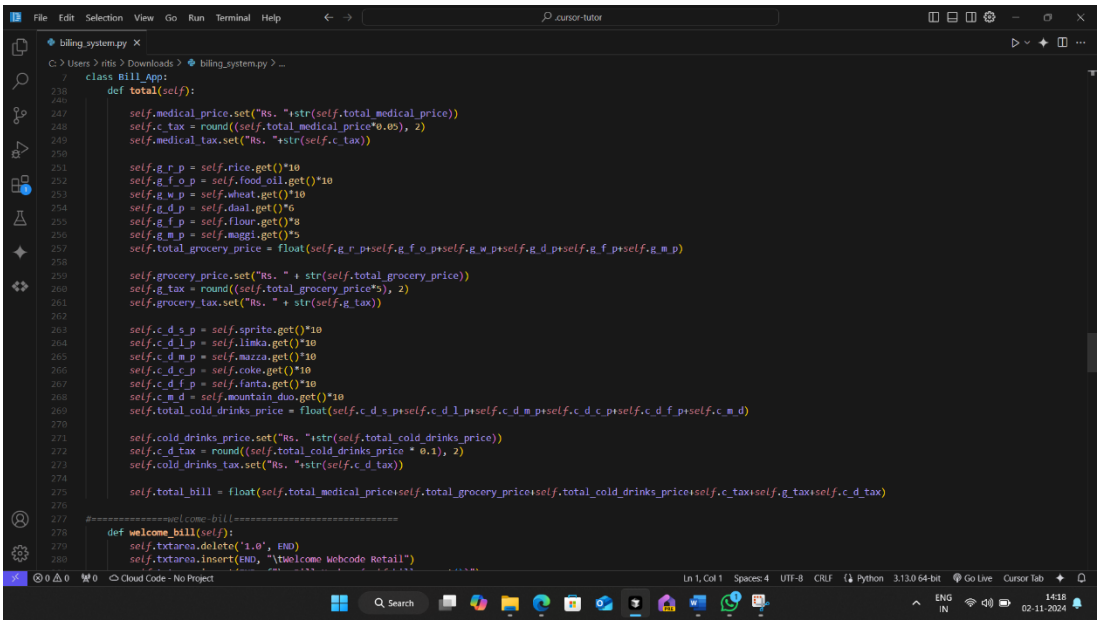
```
    class Bill_App:
        def __init__(self, root):

        # ==========ColdDrinks==============================
            F4 = LabelFrame(self.root, text="Cold Drinks", font=('times new roman', 15, 'bold'), bd=10, fg="Black", bg="#badc57")
            F4.place(x=772, y=180, width=381, height=380)

            sprite_lbl = Label(F4, text="Sprite", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            sprite_lbl.grid(row=0, column=0, padx=10, pady=10, sticky='W')
            sprite_txt = Entry(F4, width=10, textvariable=self.sprite, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            sprite_txt.grid(row=0, column=1, padx=10, pady=10)

            limka_lbl = Label(F4, text="Limka", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            limka_lbl.grid(row=1, column=0, padx=10, pady=10, sticky='W')
            limka_txt = Entry(F4, width=10, textvariable=self.limka, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            limka_txt.grid(row=1, column=1, padx=10, pady=10)

            mazza_lbl = Label(F4, text="Mazza", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            mazza_lbl.grid(row=2, column=0, padx=10, pady=10, sticky='W')
            wheat_txt = Entry(F4, width=10, textvariable=self.mazza, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            wheat_txt.grid(row=2, column=1, padx=10, pady=10)

            coke_lbl = Label(F4, text="Coke", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            coke_lbl.grid(row=3, column=0, padx=10, pady=10, sticky='W')
            coke_txt = Entry(F4, width=10, textvariable=self.coke, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            coke_txt.grid(row=3, column=1, padx=10, pady=10)

            fanta_lbl = Label(F4, text="Fanta", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            fanta_lbl.grid(row=4, column=0, padx=10, pady=10, sticky='W')
            fanta_txt = Entry(F4, width=10, textvariable=self.fanta, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            fanta_txt.grid(row=4, column=1, padx=10, pady=10)

            mountain_duo_lbl = Label(F4, text="Mountain Duo", font=('times new roman', 16, 'bold'), bg="#badc57", fg="black")
            mountain_duo_lbl.grid(row=5, column=0, padx=10, pady=10, sticky='W')
            mountain_duo_txt = Entry(F4, width=10, textvariable=self.mountain_duo, font=('times new roman', 16, 'bold'), bd=5, relief=GROOVE)
            mountain_duo_txt.grid(row=5, column=1, padx=10, pady=10)
```



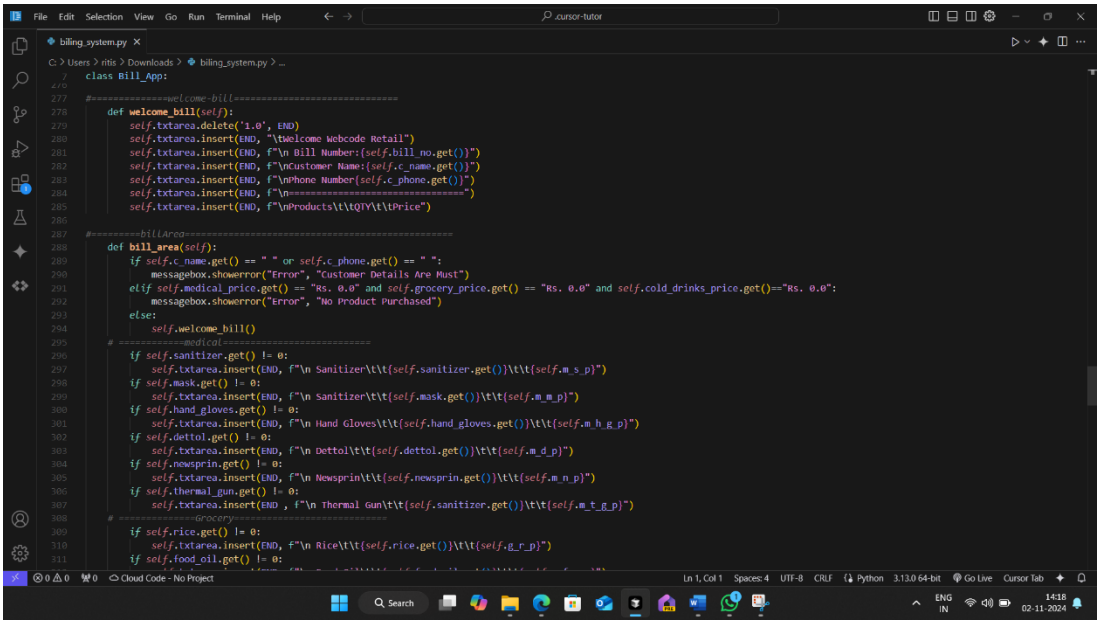
```
    class Bill_App:
        def __init__(self, root):

        # ================BillArea====================
            F5 = Frame(self.root, bd=10, relief=GROOVE)
            F5.place(x=1158, y=180, width=381, height=380)

            bill_title = Label(F5, text="Bill Area", font='arial 15 bold', bd=7, relief=GROOVE)
            bill_title.pack(fill=X)
            scroll_y = Scrollbar(F5, orient=VERTICAL)
            self.txtarea = Text(F5, yscrollcommand=scroll_y.set)
            scroll_y.pack(side=RIGHT, fill=Y)
            scroll_y.config(command=self.txtarea.yview)
            self.txtarea.pack(fill=BOTH, expand=1)

        # ====================ButtonFrame=============
            F6 = LabelFrame(self.root, text="Bill Area", font=('times new roman', 14, 'bold'), bd=10, fg="Black", bg="#badc57")
            F6.place(x=0, y=600, relwidth=1, height=140)

            m1_lbl = Label(F6, text="Total Medical Price", font=('times new roman', 14, 'bold'), bg="#badc57", fg="black")
            m1_lbl.grid(row=0, column=0, padx=20, pady=1, sticky='W')
            m1_txt = Entry(F6, width=18, textvariable=self.medical_price, font='arial 10 bold', bd=7, relief=GROOVE)
            m1_txt.grid(row=0, column=1, padx=18, pady=1)

            m2_lbl = Label(F6, text="Total Grocery Price", font=('times new roman', 14, 'bold'), bg="#badc57", fg="black")
            m2_lbl.grid(row=1, column=0, padx=20, pady=1, sticky='W')
            m2_txt = Entry(F6, width=18, textvariable=self.grocery_price, font='arial 10 bold', bd=7, relief=GROOVE)
            m2_txt.grid(row=1, column=1, padx=18, pady=1)

            m3_lbl = Label(F6, text="Total Cold Drinks Price", font=('times new roman', 14, 'bold'), bg="#badc57", fg="black")
            m3_lbl.grid(row=2, column=0, padx=20, pady=1, sticky='W')
            m3_txt = Entry(F6, width=18, textvariable=self.cold_drinks_price, font='arial 10 bold', bd=7, relief=GROOVE)
            m3_txt.grid(row=2, column=1, padx=18, pady=1)

            m4_lbl = Label(F6, text="Medical Tax", font=('times new roman', 14, 'bold'), bg="#badc57", fg="black")
            m4_lbl.grid(row=0, column=2, padx=20, pady=1, sticky='W')
            m4_txt = Entry(F6, width=18, textvariable=self.medical_tax, font='arial 10 bold', bd=7, relief=GROOVE)
```



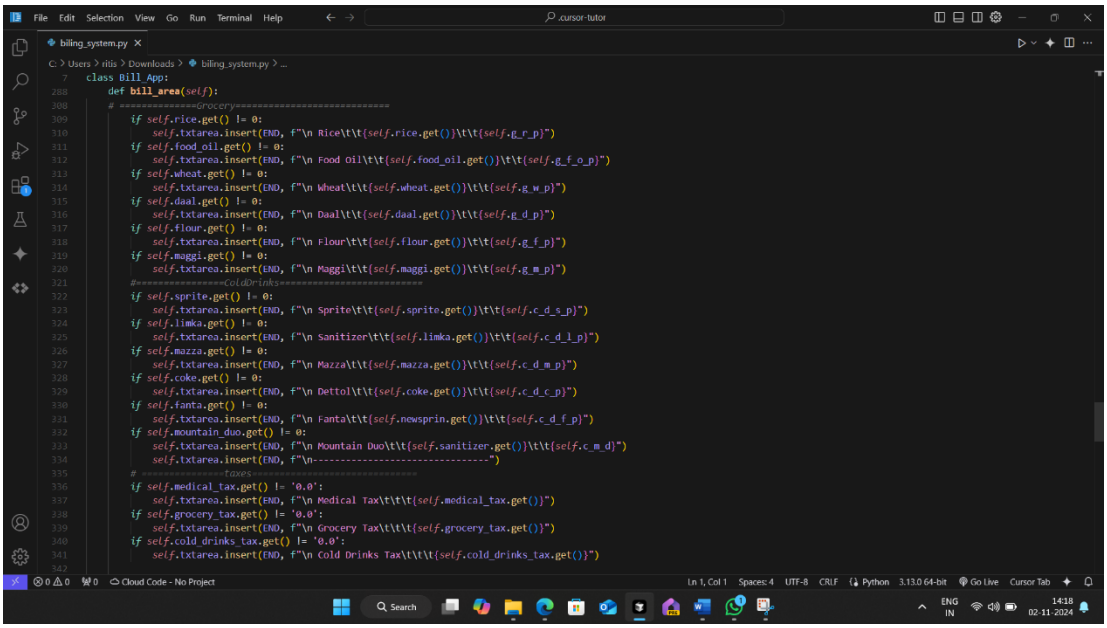
```
class Bill_App:
    def __init__(self, root):

    # =======Buttons=========================================
        btn_f = Frame(f6, bd=7, relief=GROOVE)
        btn_f.place(x=900, width=580, height=105)

        total_btn = Button(btn_f, command=self.total, text="Total", bg="#535C68", bd=2, fg="white", pady=15, width=12, font='arial 13 bold')
        total_btn.grid(row=0, column=0, padx=5, pady=5)

        generateBill_btn = Button(btn_f, command=self.bill_area, text="Generate Bill", bd=2, bg="#535C68", fg="white", pady=12, width=12, font='arial 13 bold')
        generateBill_btn.grid(row=0, column=1, padx=5, pady=5)

        clear_btn = Button(btn_f, command=self.clear_data, text="Clear", bg="#535C68", bd=2, fg="white", pady=15, width=12, font='arial 13 bold')
        clear_btn.grid(row=0, column=2, padx=5, pady=5)

        exit_btn = Button(btn_f, command=self.exit_app, text="Exit", bd=2, bg="#535C68", fg="white", pady=15, width=12, font='arial 13 bold')
        exit_btn.grid(row=0, column=3, padx=5, pady=5)
        self.welcome_bill()

#================totalBill=========================
    def total(self):
        self.m_h_g_p = self.hand_gloves.get()*12
        self.m_s_p = self.sanitizer.get()*2
        self.m_m_p = self.mask.get()*5
        self.m_d_p = self.dettol.get()*30
        self.m_n_p = self.newsprin.get()*5
        self.m_t_g_p = self.thermal_gun.get()*15
        self.total_medical_price = float(self.m_m_p+self.m_h_g_p+self.m_d_p+self.m_n_p+self.m_t_g_p+self.m_s_p)

        self.medical_price.set("Rs. "+str(self.total_medical_price))
        self.c_tax = round((self.total_medical_price*0.05), 2)
        self.medical_tax.set("Rs. "+str(self.c_tax))

        self.g_r_p = self.rice.get()*10
        self.g_f_o_p = self.food_oil.get()*10
        self.g_w_p = self.wheat.get()*10
```



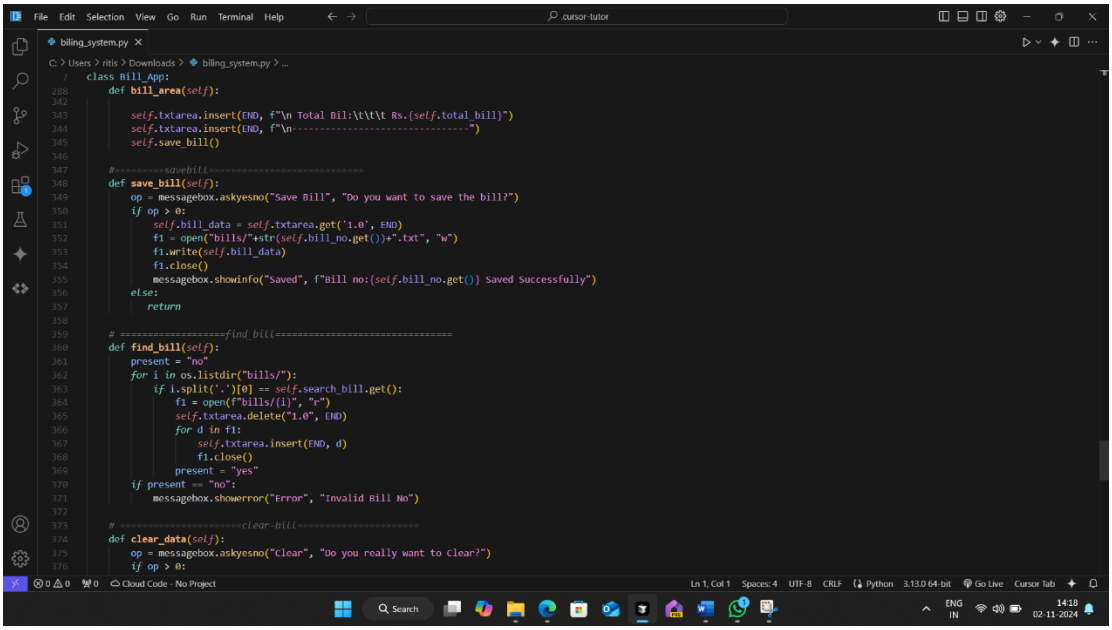
```
class Bill_App:
    def total(self):

        self.medical_price.set("Rs. "+str(self.total_medical_price))
        self.c_tax = round((self.total_medical_price*0.05), 2)
        self.medical_tax.set("Rs. "+str(self.c_tax))

        self.g_r_p = self.rice.get()*10
        self.g_f_o_p = self.food_oil.get()*10
        self.g_w_p = self.wheat.get()*10
        self.g_d_p = self.daal.get()*6
        self.g_f_p = self.flour.get()*8
        self.g_m_p = self.maggi.get()*5
        self.total_grocery_price = float(self.g_r_p+self.g_f_o_p+self.g_w_p+self.g_d_p+self.g_f_p+self.g_m_p)

        self.grocery_price.set("Rs. " + str(self.total_grocery_price))
        self.g_tax = round((self.total_grocery_price*5), 2)
        self.grocery_tax.set("Rs. " + str(self.g_tax))

        self.c_d_s_p = self.sprite.get()*10
        self.c_d_l_p = self.limka.get()*10
        self.c_d_m_p = self.mazza.get()*10
        self.c_d_c_p = self.coke.get()*10
        self.c_d_f_p = self.fanta.get()*10
        self.c_m_d = self.mountain_duo.get()*10
        self.total_cold_drinks_price = float(self.c_d_s_p+self.c_d_l_p+self.c_d_m_p+self.c_d_c_p+self.c_d_f_p+self.c_m_d)

        self.cold_drinks_price.set("Rs. "+str(self.total_cold_drinks_price))
        self.c_d_tax = round((self.total_cold_drinks_price * 0.1), 2)
        self.cold_drinks_tax.set("Rs. "+str(self.c_d_tax))

        self.total_bill = float(self.total_medical_price+self.total_grocery_price+self.total_cold_drinks_price+self.c_tax+self.g_tax+self.c_d_tax)

#=============welcome_bill=============================
    def welcome_bill(self):
        self.txtarea.delete('1.0', END)
        self.txtarea.insert(END, "\tWelcome Webcode Retail")
```



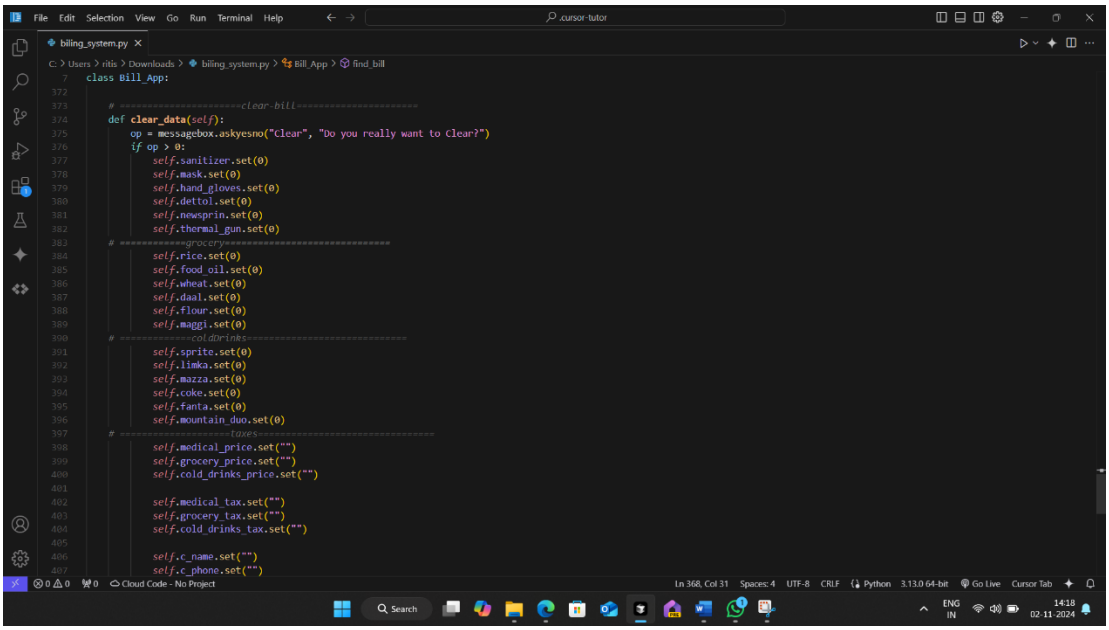
```
class Bill_App:

    #==============welcome-bill=============================
    def welcome_bill(self):
        self.txtarea.delete('1.0', END)
        self.txtarea.insert(END, "\tWelcome Webcode Retail")
        self.txtarea.insert(END, f"\n Bill Number:{self.bill_no.get()}")
        self.txtarea.insert(END, f"\nCustomer Name:{self.c_name.get()}")
        self.txtarea.insert(END, f"\nPhone Number{self.c_phone.get()}")
        self.txtarea.insert(END, f"\n==============================")
        self.txtarea.insert(END, f"\nProducts\t\tQTY\t\tPrice")

    #========billArea=================================================
    def bill_area(self):
        if self.c_name.get() == " " or self.c_phone.get() == " ":
            messagebox.showerror("Error", "Customer Details Are Must")
        elif self.medical_price.get() == "Rs. 0.0" and self.grocery_price.get() == "Rs. 0.0" and self.cold_drinks_price.get()=="Rs. 0.0":
            messagebox.showerror("Error", "No Product Purchased")
        else:
            self.welcome_bill()
        # ===========medical=========================
        if self.sanitizer.get() != 0:
            self.txtarea.insert(END, f"\n Sanitizer\t\t{self.sanitizer.get()}\t\t{self.m_s_p}")
        if self.mask.get() != 0:
            self.txtarea.insert(END, f"\n Sanitizer\t\t{self.mask.get()}\t\t{self.m_m_p}")
        if self.hand_gloves.get() != 0:
            self.txtarea.insert(END, f"\n Hand Gloves\t\t{self.hand_gloves.get()}\t\t{self.m_h_g_p}")
        if self.dettol.get() != 0:
            self.txtarea.insert(END, f"\n Dettol\t\t{self.dettol.get()}\t\t{self.m_d_p}")
        if self.newsprin.get() != 0:
            self.txtarea.insert(END, f"\n Newsprin\t\t{self.newsprin.get()}\t\t{self.m_n_p}")
        if self.thermal_gun.get() != 0:
            self.txtarea.insert(END , f"\n Thermal Gun\t\t{self.sanitizer.get()}\t\t{self.m_t_g_p}")
        # ============Grocery============================
        if self.rice.get() != 0:
            self.txtarea.insert(END, f"\n Rice\t\t{self.rice.get()}\t\t{self.g_r_p}")
        if self.food_oil.get() != 0:
```



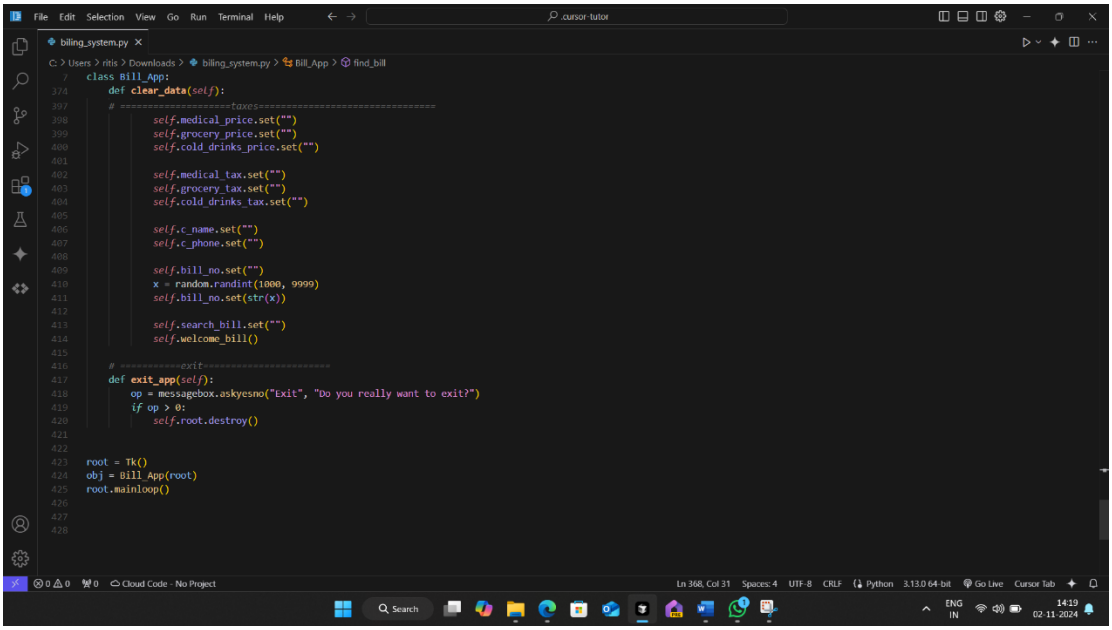
```
class Bill_App:
    def bill_area(self):
        # ============Grocery============================
        if self.rice.get() != 0:
            self.txtarea.insert(END, f"\n Rice\t\t{self.rice.get()}\t\t{self.g_r_p}")
        if self.food_oil.get() != 0:
            self.txtarea.insert(END, f"\n Food Oil\t\t{self.food_oil.get()}\t\t{self.g_f_o_p}")
        if self.wheat.get() != 0:
            self.txtarea.insert(END, f"\n Wheat\t\t{self.wheat.get()}\t\t{self.g_w_p}")
        if self.daal.get() != 0:
            self.txtarea.insert(END, f"\n Daal\t\t{self.daal.get()}\t\t{self.g_d_p}")
        if self.flour.get() != 0:
            self.txtarea.insert(END, f"\n Flour\t\t{self.flour.get()}\t\t{self.g_f_p}")
        if self.maggi.get() != 0:
            self.txtarea.insert(END, f"\n Maggi\t\t{self.maggi.get()}\t\t{self.g_m_p}")
        #==============ColdDrinks=========================
        if self.sprite.get() != 0:
            self.txtarea.insert(END, f"\n Sprite\t\t{self.sprite.get()}\t\t{self.c_d_s_p}")
        if self.limka.get() != 0:
            self.txtarea.insert(END, f"\n Sanitizer\t\t{self.limka.get()}\t\t{self.c_d_l_p}")
        if self.mazza.get() != 0:
            self.txtarea.insert(END, f"\n Mazza\t\t{self.mazza.get()}\t\t{self.c_d_m_p}")
        if self.coke.get() != 0:
            self.txtarea.insert(END, f"\n Dettol\t\t{self.coke.get()}\t\t{self.c_d_c_p}")
        if self.fanta.get() != 0:
            self.txtarea.insert(END, f"\n Fanta\t\t{self.newsprin.get()}\t\t{self.c_d_f_p}")
        if self.mountain_duo.get() != 0:
            self.txtarea.insert(END, f"\n Mountain Duo\t\t{self.sanitizer.get()}\t\t{self.c_m_d}")
            self.txtarea.insert(END, f"\n--------------------------------")
        # ==============Taxes=============================
        if self.medical_tax.get() != '0.0':
            self.txtarea.insert(END, f"\n Medical Tax\t\t\t{self.medical_tax.get()}")
        if self.grocery_tax.get() != '0.0':
            self.txtarea.insert(END, f"\n Grocery Tax\t\t\t{self.grocery_tax.get()}")
        if self.cold_drinks_tax.get() != '0.0':
            self.txtarea.insert(END, f"\n Cold Drinks Tax\t\t\t{self.cold_drinks_tax.get()}")
```



```
class Bill_App:
    def bill_area(self):

        self.txtarea.insert(END, f"\n Total Bil:\t\t\t Rs.{self.total_bill}")
        self.txtarea.insert(END, f"\n-------------------------------")
        self.save_bill()

    #========savebill=============================
    def save_bill(self):
        op = messagebox.askyesno("Save Bill", "Do you want to save the bill?")
        if op > 0:
            self.bill_data = self.txtarea.get('1.0', END)
            f1 = open("bills/"+str(self.bill_no.get())+".txt", "w")
            f1.write(self.bill_data)
            f1.close()
            messagebox.showinfo("Saved", f"Bill no:{self.bill_no.get()} Saved Successfully")
        else:
            return

    # ================find_bill==============================
    def find_bill(self):
        present = "no"
        for i in os.listdir("bills/"):
            if i.split('.')[0] == self.search_bill.get():
                f1 = open(f"bills/{i}", "r")
                self.txtarea.delete("1.0", END)
                for d in f1:
                    self.txtarea.insert(END, d)
                    f1.close()
                present = "yes"
        if present == "no":
            messagebox.showerror("Error", "Invalid Bill No")

    # ====================clear-bill====================
    def clear_data(self):
        op = messagebox.askyesno("Clear", "Do you really want to Clear?")
        if op > 0:
```



```
class Bill_App:

    # ====================clear-bill====================
    def clear_data(self):
        op = messagebox.askyesno("Clear", "Do you really want to Clear?")
        if op > 0:
            self.sanitizer.set(0)
            self.mask.set(0)
            self.hand_gloves.set(0)
            self.dettol.set(0)
            self.newsprin.set(0)
            self.thermal_gun.set(0)
    # ==========grocery=====================================
            self.rice.set(0)
            self.food_oil.set(0)
            self.wheat.set(0)
            self.daal.set(0)
            self.flour.set(0)
            self.maggi.set(0)
    # ============coldDrinks============================
            self.sprite.set(0)
            self.limka.set(0)
            self.mazza.set(0)
            self.coke.set(0)
            self.fanta.set(0)
            self.mountain_duo.set(0)
    # ====================taxes=============================
            self.medical_price.set("")
            self.grocery_price.set("")
            self.cold_drinks_price.set("")

            self.medical_tax.set("")
            self.grocery_tax.set("")
            self.cold_drinks_tax.set("")

            self.c_name.set("")
            self.c_phone.set("")
```



```
class Bill_App:
    def clear_data(self):
    # ====================taxes==============================
            self.medical_price.set("")
            self.grocery_price.set("")
            self.cold_drinks_price.set("")

            self.medical_tax.set("")
            self.grocery_tax.set("")
            self.cold_drinks_tax.set("")

            self.c_name.set("")
            self.c_phone.set("")

            self.bill_no.set("")
            x = random.randint(1000, 9999)
            self.bill_no.set(str(x))

            self.search_bill.set("")
            self.welcome_bill()

    # ==========exit=======================
    def exit_app(self):
        op = messagebox.askyesno("Exit", "Do you really want to exit?")
        if op > 0:
            self.root.destroy()


root = Tk()
obj = Bill_App(root)
root.mainloop()
```