UNIVERSITY OF
Southampton
MALAYSIA

Inventory System

Soo Yung Ting

33354456

UNIVERSITY OF SOUTHAMPTON

(MALAYSIA CAMPUS)

17th February 2023

UNIVERSITY OF Southampton MALAYSIA

Declaration of Academic Integrity

Please sign to indicate that you have read and accepted the following statements. Your assignment will not be accepted without this declaration.

I confirm that:

1. I have read and understood the University's Academic Integrity Guidance for Students and that in the attached submission I have worked within the expectations of the Regulations Governing Academic Integrity.
2. I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
3. I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

The extent to which I have worked with others is as follows:

This submission is inadmissible without a written signature below (applies to hard copy only).

Name: Soo Yung Ting

Signature: [illegible]

Date: 17th February 2023

Table of Content

Section	Content	Page Number
1	Introduction	4 - 5
2	Detail explanation of each function	5 - 17
3	Screenshot of each coding	17 - 28
4	Future work (suggestion for improvement)	29
5	Conclusion	29
6	References	29 - 30
7	Appendix	30

1. Introduction

Problem Statement

As a manager of an in-store shop, they deal with a lot of tangible goods that need precise tracking and administration (Hashmicro, 2022) [1]. They now lack an effective mechanism to control their inventory levels, which has led to overstocking, lost sales opportunities, and increased expenditures for the company (Ginsberg, 2022) [3]. They are looking for an inventory management system that can give them up-to-the-second inventory updates, interface with our point-of-sale system, and give them accurate data so they can decide on restocking, pricing, and marketing in a well-informed manner (courses.lumenlearning.com, n.d.) [2]. Additionally, they desire a user-friendly, adaptable system that can accommodate their company's expanding needs. It would be advantageous to be able to create reports and analyse their inventory data in graphs.

Definition of an Inventory Program

According to Unleashed Software (2019) [4], businesses can measure and manage their inventory levels using a software programme called an inventory management system. A centralised view of inventory data, including stock levels, sales, and orders, is provided by the system, which also has the ability to update inventory levels in real-time.

Benefits

For businesses of all sizes in a variety of industries, inventory management systems have emerged as a vital tool (Jenkins, 2020) [7]. These systems are used in the retail industry to balance stock levels with client demand, minimising overstocking and unnecessary expenses (Inventory, 2023) [8]. Similar to this, according to Abu Zwaida, Pham and Beauregard (2021) [9], inventory management systems in manufacturing regulate the flow of raw materials and finished items, boosting operational effectiveness and reducing waste. Healthcare organisations may keep track of and manage medical supplies, assuring their availability and reducing the chance of shortages. They can also keep track of product expiration dates and guarantee proper handling. Systems for managing inventories have many advantages for firms. Based on WiSys (2021) [6], real-time inventory monitoring avoids under- or overstocking, lowering maintenance costs and ensuring product availability. By monitoring the flow of commodities from suppliers to customers, the system can improve supply chain operations. The method increases the accuracy of demand forecasting, enabling companies to better manage production and purchasing (BigCommerce, 2022) [5]. Finally, by sending alerts for low inventory levels and unusual activity in the warehouse, the system lowers the possibility of product theft or loss (Velthoen, 2015) [10].

Importance

For any company that deals with tangible things, the value of an inventory management system cannot be emphasised. According to Keenan (2022) [16], an inventory management system gives firms real-time access to their stock levels, assisting them in preventing product overstock or understock. By making sure that the products are always in stock, can save waste, lower inventory carrying costs, and boost customer satisfaction. Systems for managing inventory can also assist companies in making data-driven decisions about pricing, marketing, and restocking (Rickerby, 2022). An inventory management system can increase profitability and provide a business with a competitive edge in the market by streamlining operations and decreasing inefficiencies. Any company that wishes to enhance its supply chain management, boost productivity, and maximise profitability should invest in an inventory management system (Atnafu and Balda, 2018) [17].

Real-World Application

Systems for inventory management are becoming an essential tool for businesses of all sizes in a variety of industries. Companies in the retail industry can utilise inventory management systems to prevent overstocking and avoidable expenses while making sure there are enough products on hand to meet client demand (AltexSoft, 2021) [11]. Similar to this, an inventory management system in manufacturing can aid in managing the flow of raw materials and completed items, reducing waste and boosting operational effectiveness. Medical supply tracking and management can be done by healthcare organisations using inventory management software, reducing the risk of shortages and ensuring that vital commodities are always on hand. In addition to keeping track of product expiration dates, these systems can help make sure that products are used in the proper sequence. Businesses may cut expenses, boost productivity, and improve customer service by implementing inventory management systems (Asset Infinity Blog, 2019) [12].

A number of significant businesses have put inventory management systems in place to streamline their operations and increase efficiency. According to Publisher (2015) [13], Walmart's one of the biggest retailers in the world, uses Retail Link to monitor sales data and control inventory levels. Walmart may use this method to track the success of specific products in real-time and modify inventory levels, accordingly, cutting waste and guaranteeing those in-demand items are always available. Amazon, on the other hand, manages its enormous product inventories using a sophisticated inventory management system. Using algorithms, the company's system forecasts demand and modifies inventory levels to ensure that products are accessible when clients need them. As a result, Amazon can now operate with a high degree of dependability and efficiency, minimising waste and cutting expenses (Amazon, 2022) [14]. In the healthcare sector, Cardinal Health's WaveMark system, which tracks medical supplies using RFID technology, aids in real-time usage monitoring, lowering the possibility of shortages and enhancing patient outcomes (howard, 2021) [15].

Success Criteria

1. **Accuracy:** The system must be able to accurately pinpoint inventory items' amounts as well as the areas of those items in the storeroom or storage area.

2. **Efficiency:** The system should be capable of quickly executing operations and producing records on inventory levels, restocking needs, and other relevant metrics.

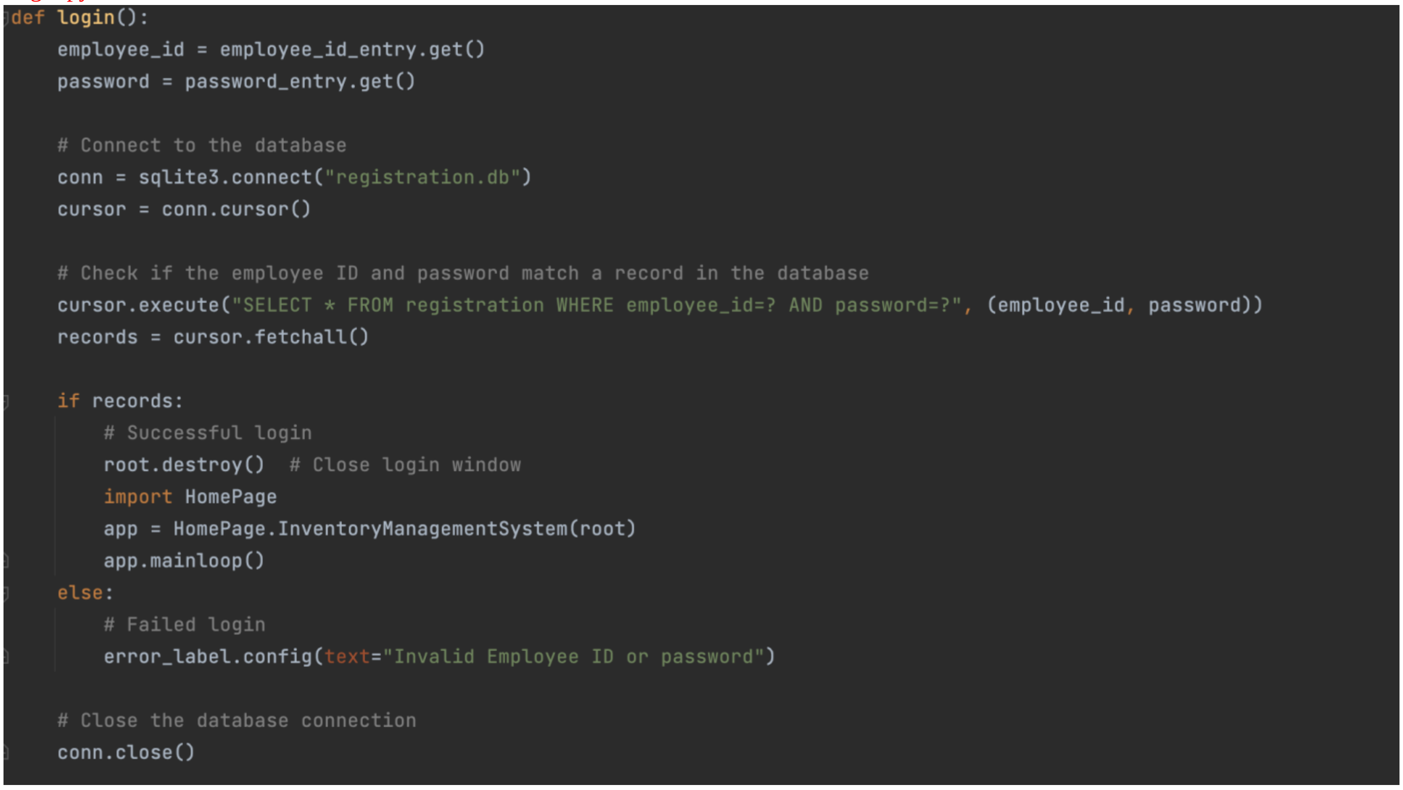
3. **User-Friendliness:** The system should have an easy-to-use user interface and straightforward instructions to make it straightforward for all staff members who need to interact with it to use.

4. **Security:** The system needs to be secure, with the appropriate access controls in position to avoid unauthorized access or data manipulation.

5. **Cost-Effectiveness:** The system should produce a rewarding outcome and be reasonably priced when considering the profits, it will bring the company.

2. Detail explanation of each function (Explain the function – def…)

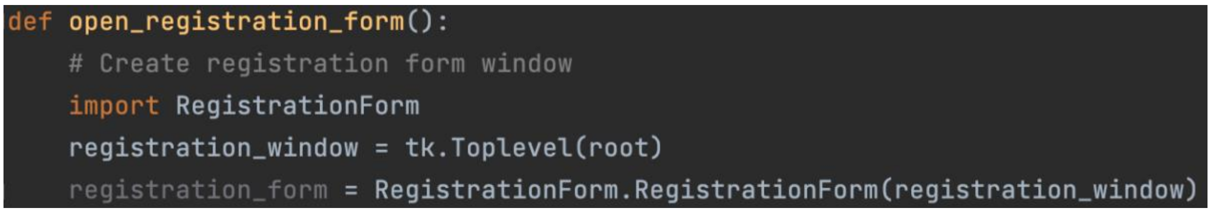
Login.py



```python
def login():
    employee_id = employee_id_entry.get()
    password = password_entry.get()

    # Connect to the database
    conn = sqlite3.connect("registration.db")
    cursor = conn.cursor()

    # Check if the employee ID and password match a record in the database
    cursor.execute("SELECT * FROM registration WHERE employee_id=? AND password=?", (employee_id, password))
    records = cursor.fetchall()

    if records:
        # Successful login
        root.destroy()  # Close login window
        import HomePage
        app = HomePage.InventoryManagementSystem(root)
        app.mainloop()
    else:
        # Failed login
        error_label.config(text="Invalid Employee ID or password")

    # Close the database connection
    conn.close()
```

This function's objective is to verify an employee's identity by determining whether the 'employee_id' and 'password' they have entered correspond to a record in the database. It accomplishes this by first retrieving the 'employee_id' and 'password' from the corresponding fields on the login form labelled 'employee_id_entry' and 'password_entry'. The 'sqlite3' library is then used by the function to establish a connection to the 'registration.db' SQLite database. To run SQL commands on the database, it constructs a cursor object. The function then runs a 'SELECT' SQL query on the 'registration' table using the arguments from the 'id' and 'password' fields. The query determines whether there is a record in the table with an 'employee_id' matching the 'employee_id' supplied and a 'password' matching the password entered.

The function opens a new window that displays the home page of the inventory management system using an 'InventoryManagementSystem' object, destroys the login window, and logs the user in if there is a record that matches both the 'employee_id' and 'password'. The function shows an error message on the login form using a label called error label if there is no record that matches the 'employee_id' and 'password' entered. The function then uses the 'close()' method of the connection object to terminate the database connection.

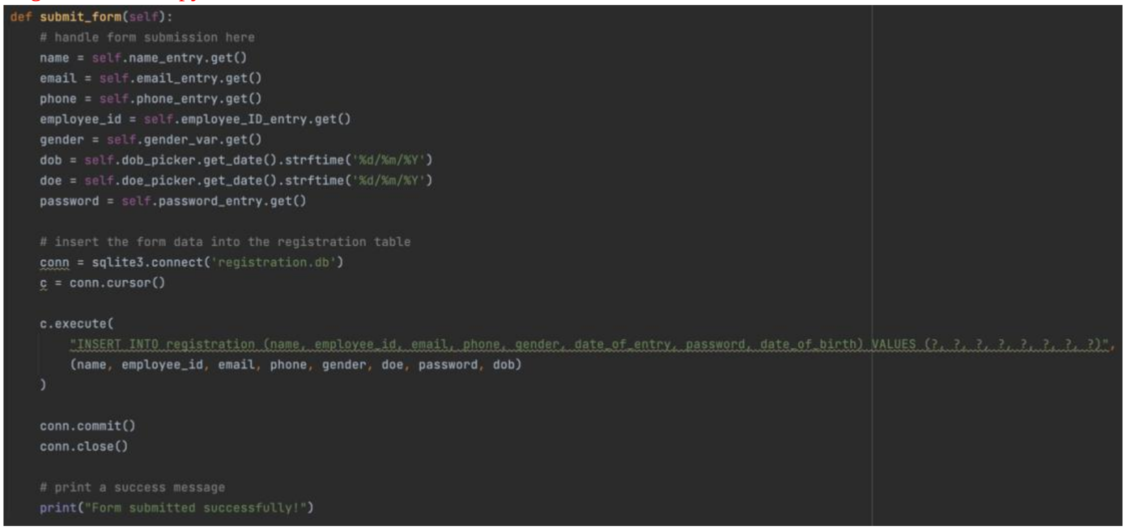


```python
def open_registration_form():
    # Create registration form window
    import RegistrationForm
    registration_window = tk.Toplevel(root)
    registration_form = RegistrationForm.RegistrationForm(registration_window)
```

This function opens a brand-new display with a sign-up sheet for a first-time worker to fill out. To do this, it brings in a module named 'RegistrationForm', containing the graphics user interface (GUI) code. Afterward, it produces a new top-level window with the 'Toplevel()'

technique of the ‘tk’ module, which is employed by ‘Tkinter’, the ‘Python GUI toolkit’, to generate fresh windows. This new window is composed as a descendant of the main window ‘(root)’ in which the function is designated. Eventually, the function sets up a novel instance of the ‘RegistrationForm’ class, passing the freshly made registration window to the constructor as an argument. This permits the recent sign-up window to showcase the ‘RegistrationForm’ entity.

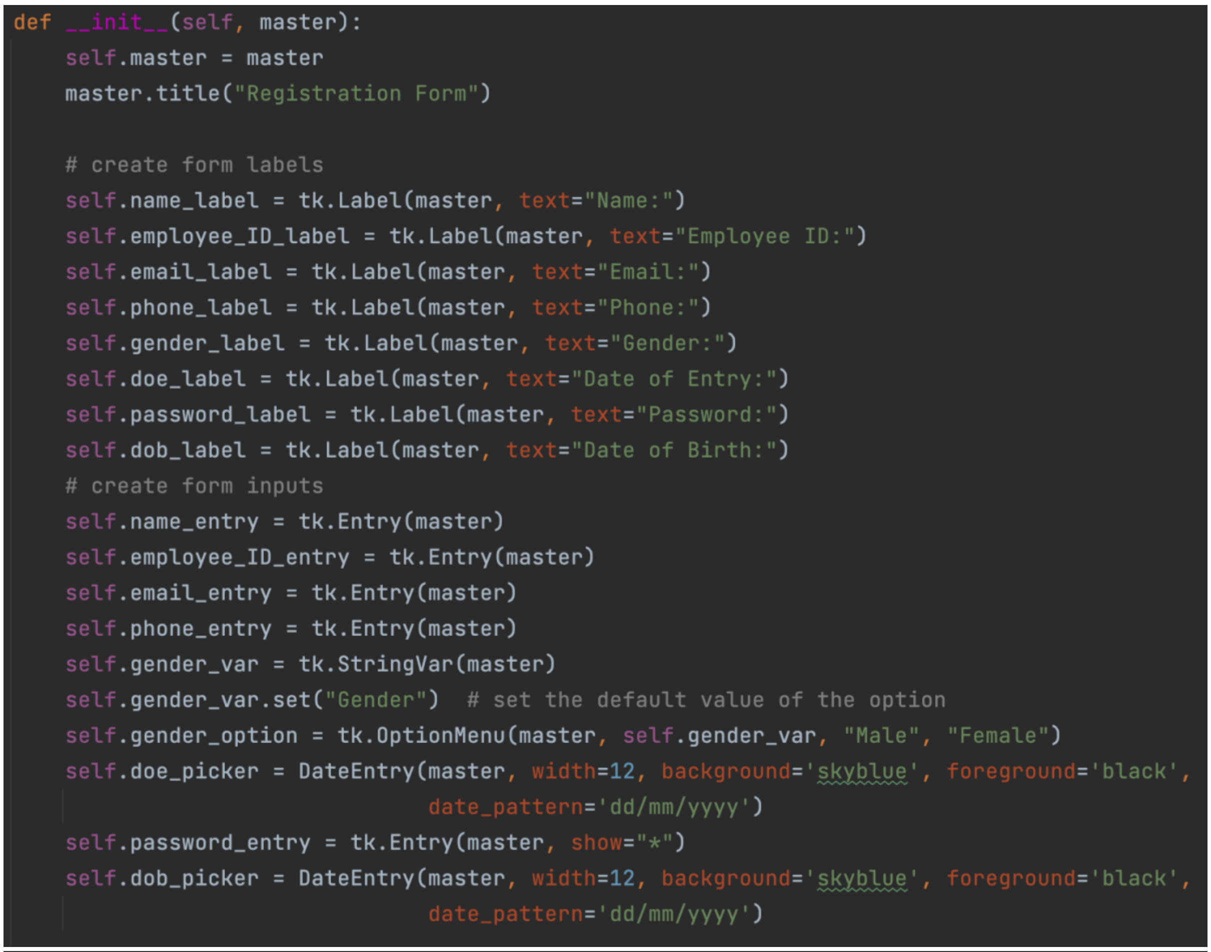
RegistrationForm.py



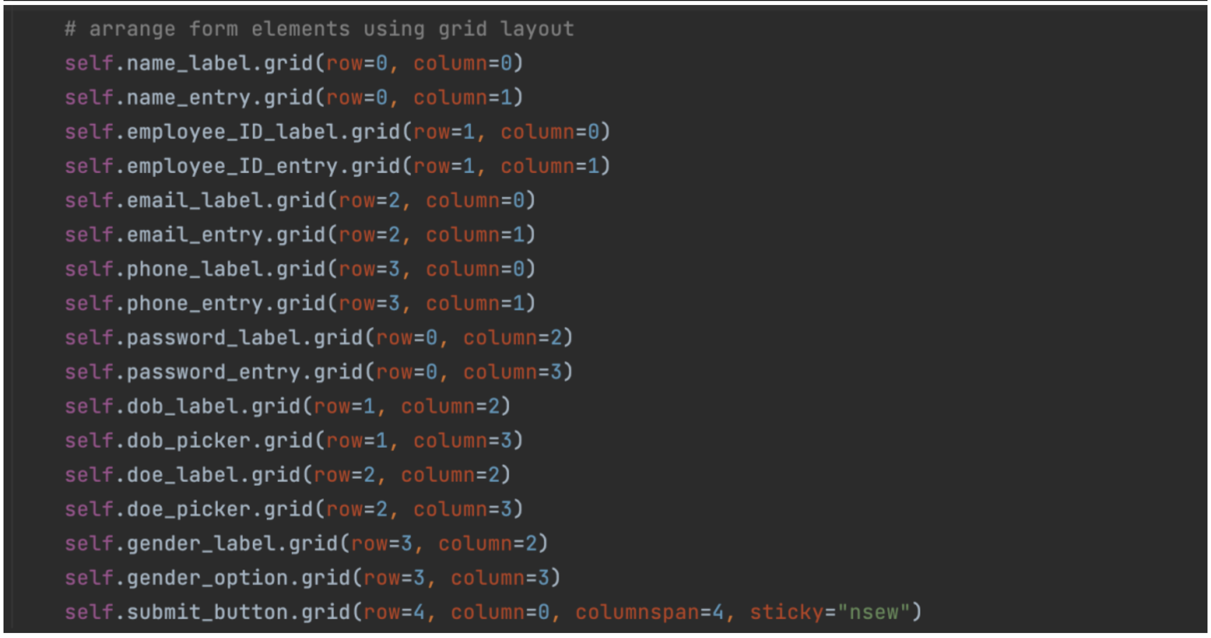
```python
def submit_form(self):
    # handle form submission here
    name = self.name_entry.get()
    email = self.email_entry.get()
    phone = self.phone_entry.get()
    employee_id = self.employee_ID_entry.get()
    gender = self.gender_var.get()
    dob = self.dob_picker.get_date().strftime('%d/%m/%Y')
    doe = self.doe_picker.get_date().strftime('%d/%m/%Y')
    password = self.password_entry.get()

    # insert the form data into the registration table
    conn = sqlite3.connect('registration.db')
    c = conn.cursor()

    c.execute(
        "INSERT INTO registration (name, employee_id, email, phone, gender, date_of_entry, password, date_of_birth) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
        (name, employee_id, email, phone, gender, doe, password, dob)
    )

    conn.commit()
    conn.close()

    # print a success message
    print("Form submitted successfully!")
```

When a user completes the registration form, the function ‘submit_form(self)’ defined in this code is triggered. This function's purpose is to extract the information provided by the user from the registration form and add it to an SQLite database. This is achieved firstly by collecting the data the user has entered in each registration form field, such as ‘name’, ‘email’, ‘employee_id’, ‘gender’, ‘dob’ (date of birth), ‘doe’ (date of entry), and ‘password’. Next, the method connects to the ‘registration.db’ SQLite database using the ‘sqlite3’ library and creates a cursor object to execute SQL queries on the database.

Utilizing the user-supplied values as arguments, the procedure then carries out an ‘INSERT’ SQL query on the ‘registration’ table. This action adds the user's registration data as a new line in the table. The connection object's ‘commit()’ method is utilized to confirm the transformations made to the database after the query has been completed. The method ultimately concludes by using the connection object's ‘close()’ method to close the database connection. The process finishes by presenting a success message in the console to demonstrate that the registration form was successfully submitted.



```python
def __init__(self, master):
    self.master = master
    master.title("Registration Form")

    # create form labels
    self.name_label = tk.Label(master, text="Name:")
    self.employee_ID_label = tk.Label(master, text="Employee ID:")
    self.email_label = tk.Label(master, text="Email:")
    self.phone_label = tk.Label(master, text="Phone:")
    self.gender_label = tk.Label(master, text="Gender:")
    self.doe_label = tk.Label(master, text="Date of Entry:")
    self.password_label = tk.Label(master, text="Password:")
    self.dob_label = tk.Label(master, text="Date of Birth:")
    # create form inputs
    self.name_entry = tk.Entry(master)
    self.employee_ID_entry = tk.Entry(master)
    self.email_entry = tk.Entry(master)
    self.phone_entry = tk.Entry(master)
    self.gender_var = tk.StringVar(master)
    self.gender_var.set("Gender")  # set the default value of the option
    self.gender_option = tk.OptionMenu(master, self.gender_var, "Male", "Female")
    self.doe_picker = DateEntry(master, width=12, background='skyblue', foreground='black',
                                date_pattern='dd/mm/yyyy')
    self.password_entry = tk.Entry(master, show="*")
    self.dob_picker = DateEntry(master, width=12, background='skyblue', foreground='black',
                                date_pattern='dd/mm/yyyy')
```



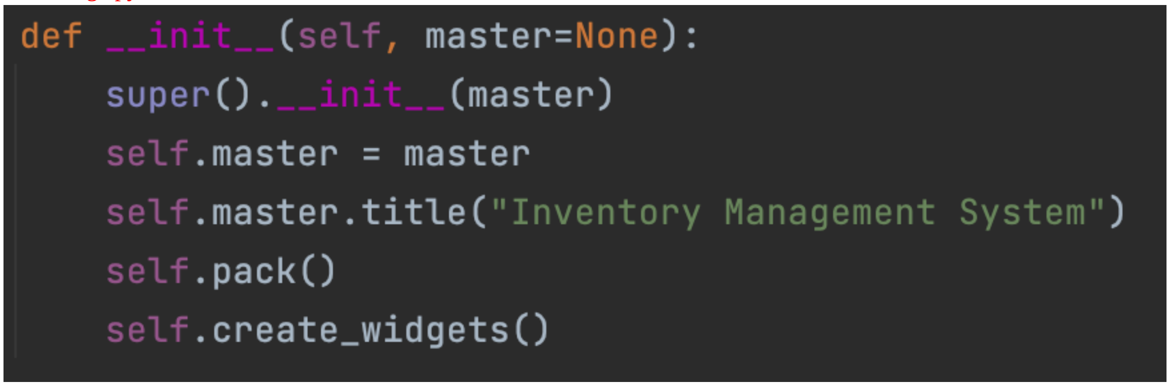
```python
    # arrange form elements using grid layout
    self.name_label.grid(row=0, column=0)
    self.name_entry.grid(row=0, column=1)
    self.employee_ID_label.grid(row=1, column=0)
    self.employee_ID_entry.grid(row=1, column=1)
    self.email_label.grid(row=2, column=0)
    self.email_entry.grid(row=2, column=1)
    self.phone_label.grid(row=3, column=0)
    self.phone_entry.grid(row=3, column=1)
    self.password_label.grid(row=0, column=2)
    self.password_entry.grid(row=0, column=3)
    self.dob_label.grid(row=1, column=2)
    self.dob_picker.grid(row=1, column=3)
    self.doe_label.grid(row=2, column=2)
    self.doe_picker.grid(row=2, column=3)
    self.gender_label.grid(row=3, column=2)
    self.gender_option.grid(row=3, column=3)
    self.submit_button.grid(row=4, column=0, columnspan=4, sticky="nsew")
```

This function defines the '__init__' method for the 'RegistrationForm' class. The 'RegistrationForm' class is a custom widget that inherits from the 'tk.Frame' class in the Tkinter library, which is a standard Python library for creating graphical user interfaces. The '__init__' method is called when an instance of the 'RegistrationForm' class is created. The master parameter is the parent widget that the 'RegistrationForm' is placed in. The method sets the title of the 'master' window to "Registration Form". It then creates labels for each input field in the registration form using the 'tk.Label' widget. The labels are given text that describes the field.

Next, the method creates input fields for the registration form using the 'tk.Entry' widget. These input fields are where the user can enter their name, email, phone number, employee ID, password, and date of birth. A drop-down menu is also created using the 'tk.OptionMenu' widget for selecting the user's gender. The 'StringVar' object 'gender_var' is used to store the current value of the gender option. Two date pickers are also created using the 'DateEntry' widget. One for the date of entry and the other for the date of birth. These date pickers allow the user to select a date from a calendar widget.

Finally, a submit button is created using the 'tk.Button' widget. The 'submit_form' method is called when the button is clicked. All the form elements are then arranged on the window using the 'grid' geometry manager.

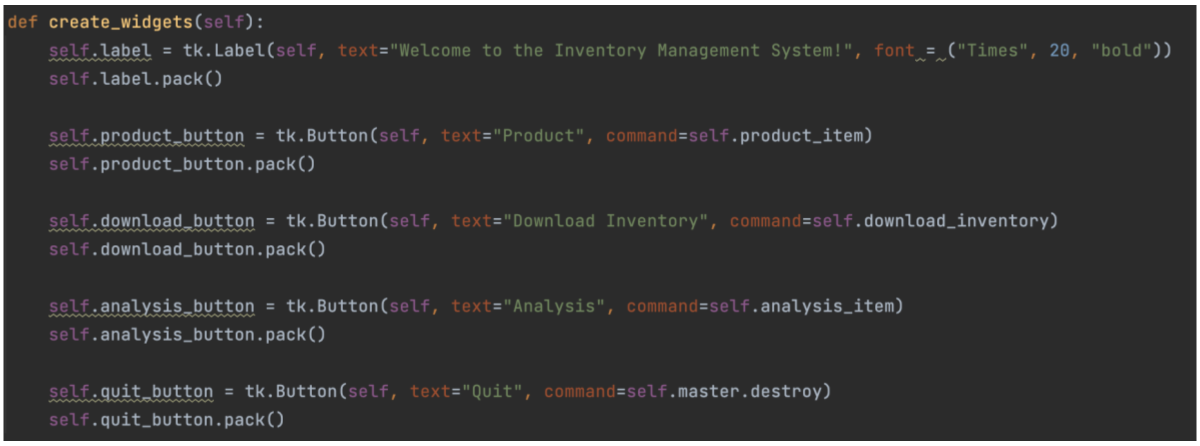
HomePage.py



```
def __init__(self, master=None):
    super().__init__(master)
    self.master = master
    self.master.title("Inventory Management System")
    self.pack()
    self.create_widgets()
```

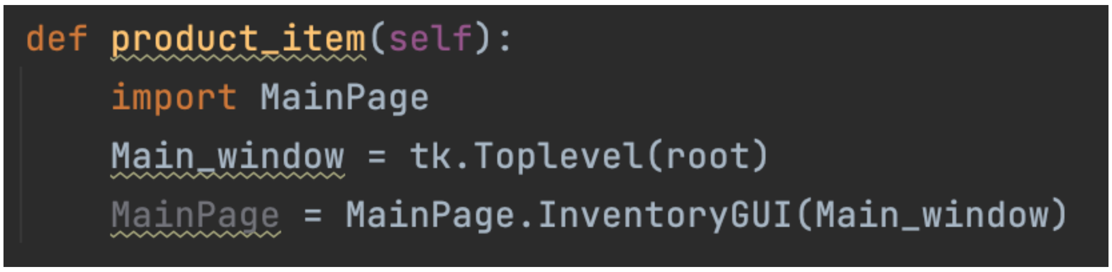
This is the constructor for the ‘InventoryManagementSystem’ class, which initializes the GUI for the application. The ‘master’ parameter refers to the parent widget, which is the main application window. If no value is provided for ‘master’, it defaults to ‘None’. The constructor then calls the constructor of the superclass ‘tk.Tk’ by using the ‘super()’ function to set up the window. The ‘self.master.title()’ method is called to set the title of the window to "Inventory Management System". The ‘self.pack()’ method is called to add the main widget to the window.

Finally, the ‘self.create_widgets()’ method is called to create the various widgets that make up the GUI, including buttons, labels, and text boxes. This method is called automatically when an instance of the ‘InventoryManagementSystem’ class is created.



```
def create_widgets(self):
    self.label = tk.Label(self, text="Welcome to the Inventory Management System!", font = ("Times", 20, "bold"))
    self.label.pack()

    self.product_button = tk.Button(self, text="Product", command=self.product_item)
    self.product_button.pack()

    self.download_button = tk.Button(self, text="Download Inventory", command=self.download_inventory)
    self.download_button.pack()

    self.analysis_button = tk.Button(self, text="Analysis", command=self.analysis_item)
    self.analysis_button.pack()

    self.quit_button = tk.Button(self, text="Quit", command=self.master.destroy)
    self.quit_button.pack()
```

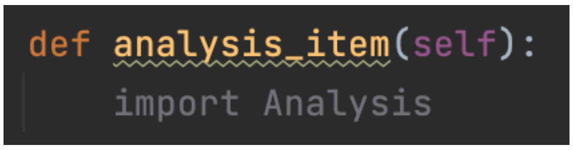
The method first creates a label widget and sets its text to “Welcome to the Inventory Management System!” using the ‘tk.Label()’ constructor. The ‘font’ parameter is also used to set the font of the label to ‘Times’ with a size of 20 and a weight of ‘bold’. The label is then packed into the GUI window using the ‘pack()’ method. Next, the method creates three button widgets using the ‘tk.Button()’ constructor. Each button is given a text label and a command to execute when clicked. The ‘product_button’ is linked to the ‘product_item()’ method, the ‘analysis_button’ is linked to the ‘analysis_item()’ method, and the ‘download_button’ is linked to the ‘download_inventory()’ method. Finally, a ‘quit_button’ is created and linked to the ‘destroy()’ method of the GUI window, which terminates the application when clicked. Each of the buttons is then packed into the GUI window using the ‘pack()’ method.



```
def product_item(self):
    import MainPage
    Main_window = tk.Toplevel(root)
    MainPage = MainPage.InventoryGUI(Main_window)
```

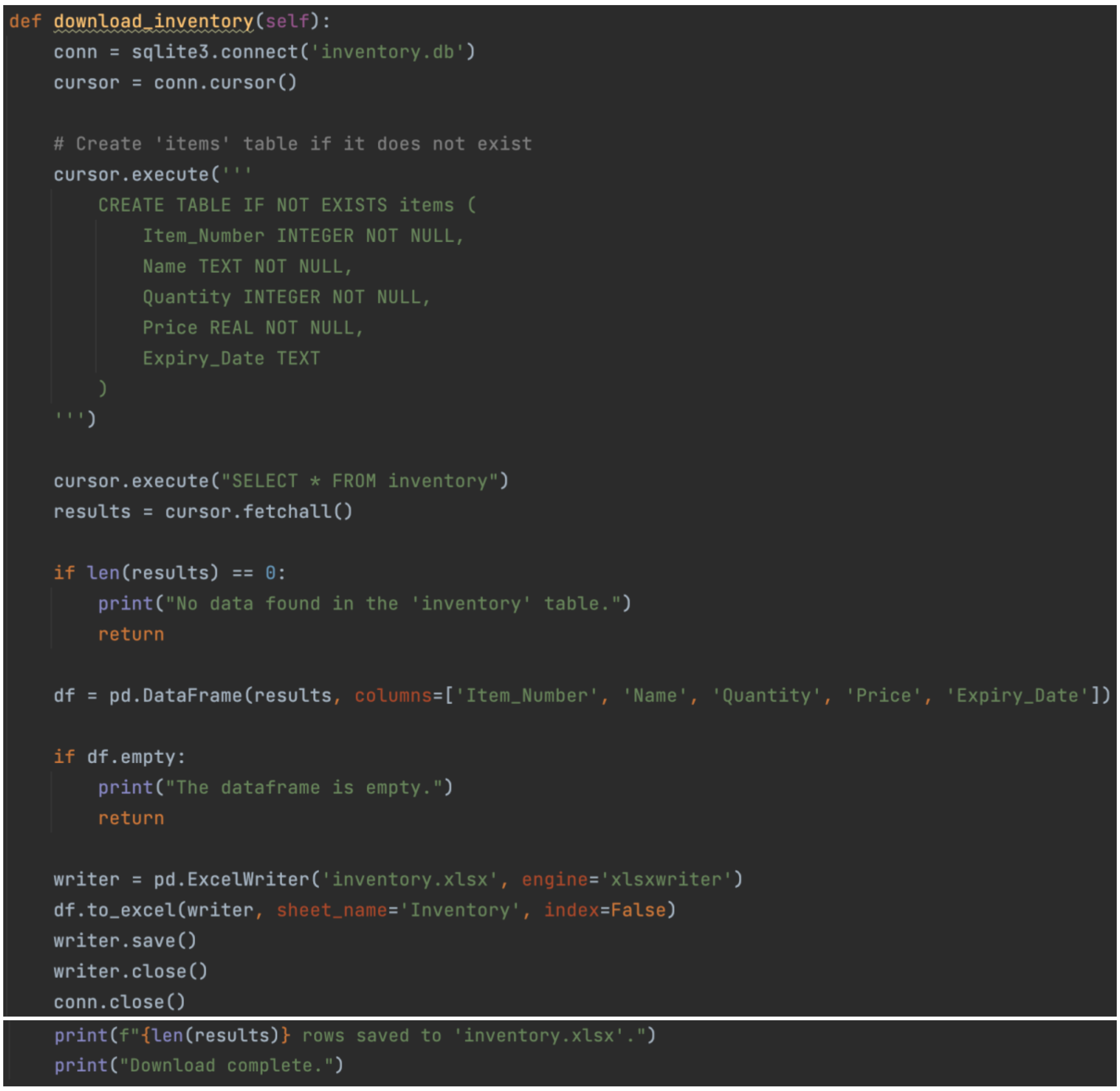
The ‘InventoryManagementSystem’ class includes the ‘product_item’ method. When it is activated, it imports the ‘MainPage’ module and uses the ‘Toplevel()’ procedure to make a fresh top-level window that is not connected to the main window. In order to construct the new window over the main window, it gives the ‘root’ object, which is the main window, as an argument to the ‘Toplevel()’ method. Next, after supplying the freshly created top-level window as an argument, the function instantiates the ‘InventoryGUI’ class, which is specified in the ‘MainPage’ module. To do this, invoke the constructor of the ‘InventoryGUI’ class while providing it the freshly generated ‘Main_window’.

Subsequently, a variable with the same name as the module is given to the ‘MainPage’ specimen, which may be perplexing but is satisfactory in Python. The ‘MainPage’ serves as the primary window of the inventory management system, which displays a tabular format of product data and allows for selections to include, modify, and eliminate products, which is made by the constructor of the ‘InventoryGUI()’ capability.



```
def analysis_item(self):
    import Analysis
```

The ‘analysis_item’ function is a method of the ‘InventoryManagementSystem’ class that is called when the user clicks the ‘Analysis’ button on the GUI. This function imports the ‘Analysis’ module, which presumably contains code for generating and displaying a graph analysis of the item number and its quantity related to the inventory management system.



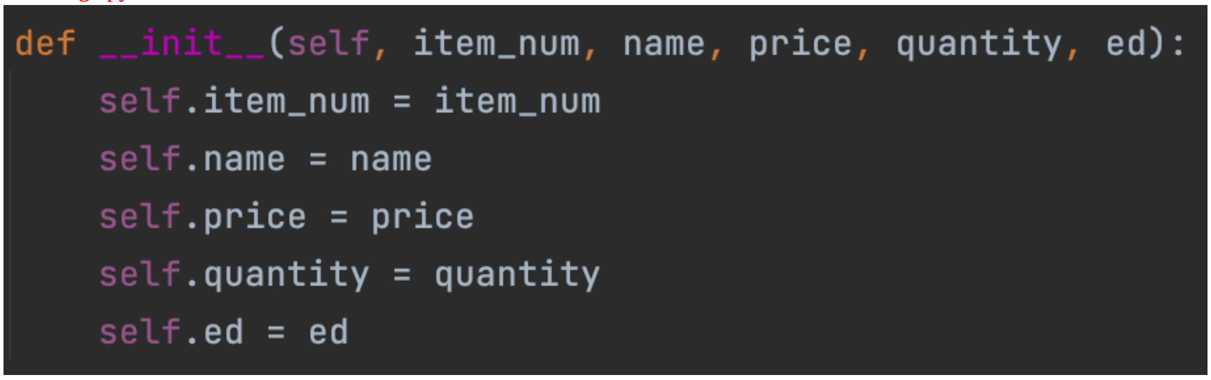
```
def download_inventory(self):
    conn = sqlite3.connect('inventory.db')
    cursor = conn.cursor()

    # Create 'items' table if it does not exist
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS items (
            Item_Number INTEGER NOT NULL,
            Name TEXT NOT NULL,
            Quantity INTEGER NOT NULL,
            Price REAL NOT NULL,
            Expiry_Date TEXT
        )
    ''')

    cursor.execute("SELECT * FROM inventory")
    results = cursor.fetchall()

    if len(results) == 0:
        print("No data found in the 'inventory' table.")
        return

    df = pd.DataFrame(results, columns=['Item_Number', 'Name', 'Quantity', 'Price', 'Expiry_Date'])

    if df.empty:
        print("The dataframe is empty.")
        return

    writer = pd.ExcelWriter('inventory.xlsx', engine='xlsxwriter')
    df.to_excel(writer, sheet_name='Inventory', index=False)
    writer.save()
    writer.close()
    conn.close()
    print(f"{len(results)} rows saved to 'inventory.xlsx'.")
    print("Download complete.")
```

The inventory info is exported into an Excel document from an SQLite database utilizing the ‘download_inventory’ capacity. It begins by exploiting the ‘sqlite3’ library to set up a connection to the SQLite database. In the event that a ‘items’ table doesn’t already exist, the function at that point runs a SQL explanation to make it. The segments for ‘Item_Number,’ ‘Name,’ ‘Quantity,’ ‘Price,’ and ‘Expiry_Date’ can be found in the items table. The function at that point runs a SQL command to recover all lines from the ‘inventory’ table and spares the ‘results’ in a variable called ‘results’.

The function presents a message to the console noting that no data was discovered in the ‘inventory’ table if there are no results, then it terminates. If there are outcomes, the function produces a Pandas DataFrame with the columns ‘Item_Number,’ ‘Name,’ ‘Quantity,’ ‘Price,’ and ‘Expiry_Date’ and utilizes the ‘results’ variable as a placeholder. The function then inspects if the DataFrame is vacant. If it is vacant, the function presents a notification to the console mentioning that the DataFrame is empty and then terminates.

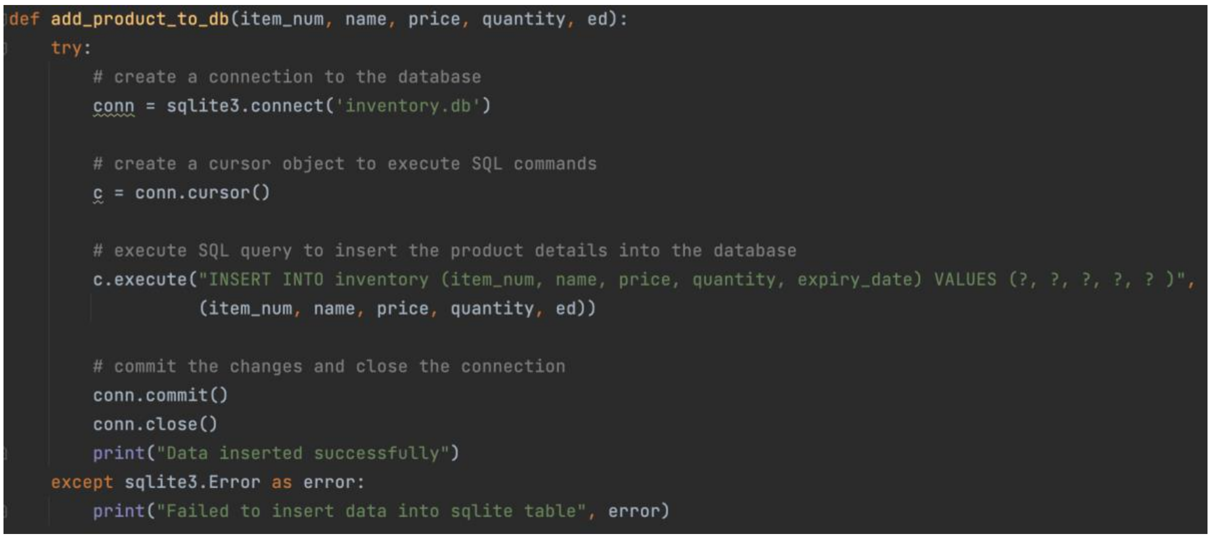
The function creates a fresh Excel file referred to as ‘inventory.xlsx’ if the DataFrame is not empty using the ‘xlsxwriter’ engine. It then writes the DataFrame to a sheet called ‘Inventory’ within the Excel file, sets the ‘index’ parameter to ‘False’ to prevent the DataFrame's index from being included in the sheet, saves the file, and closes the writer object. Finally, the function prints a message to the console indicating how many rows were saved to the Excel file and that the download is complete. The function then closes the connection to the SQLite database.

MainPage.py



```python
def __init__(self, item_num, name, price, quantity, ed):
    self.item_num = item_num
    self.name = name
    self.price = price
    self.quantity = quantity
    self.ed = ed
```

The special constructor method, also referred to as '__init__', is exclusively used when a fresh instance of the 'Product' class is created. This process is referred to by the 'self' parameter. Five distinct requirements are imposed by the method: 'item_num', 'name', 'price', 'quantity', and 'ed' (expiration date). These represent a product's features, such as its designation, item number, cost, quantity, and expiry date. Using the supplied inputs, the method sets up the attributes 'item_num', 'name', 'price', 'quantity', and 'ed'. which are exclusive to every item class instance. These components indicate the condition of a product article.

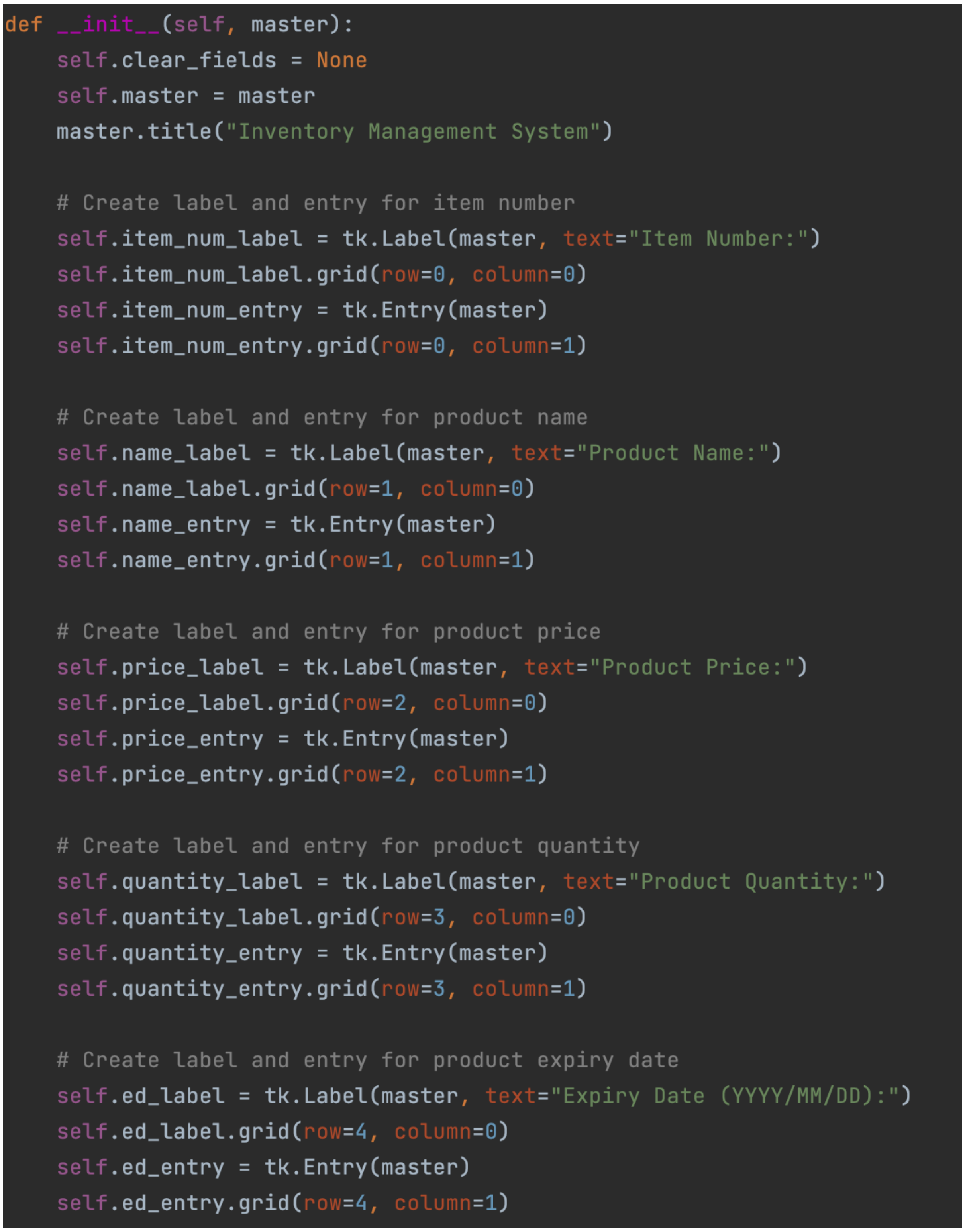


```python
def add_product_to_db(item_num, name, price, quantity, ed):
    try:
        # create a connection to the database
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to insert the product details into the database
        c.execute("INSERT INTO inventory (item_num, name, price, quantity, expiry_date) VALUES (?, ?, ?, ?, ? )",
                  (item_num, name, price, quantity, ed))

        # commit the changes and close the connection
        conn.commit()
        conn.close()
        print("Data inserted successfully")
    except sqlite3.Error as error:
        print("Failed to insert data into sqlite table", error)
```

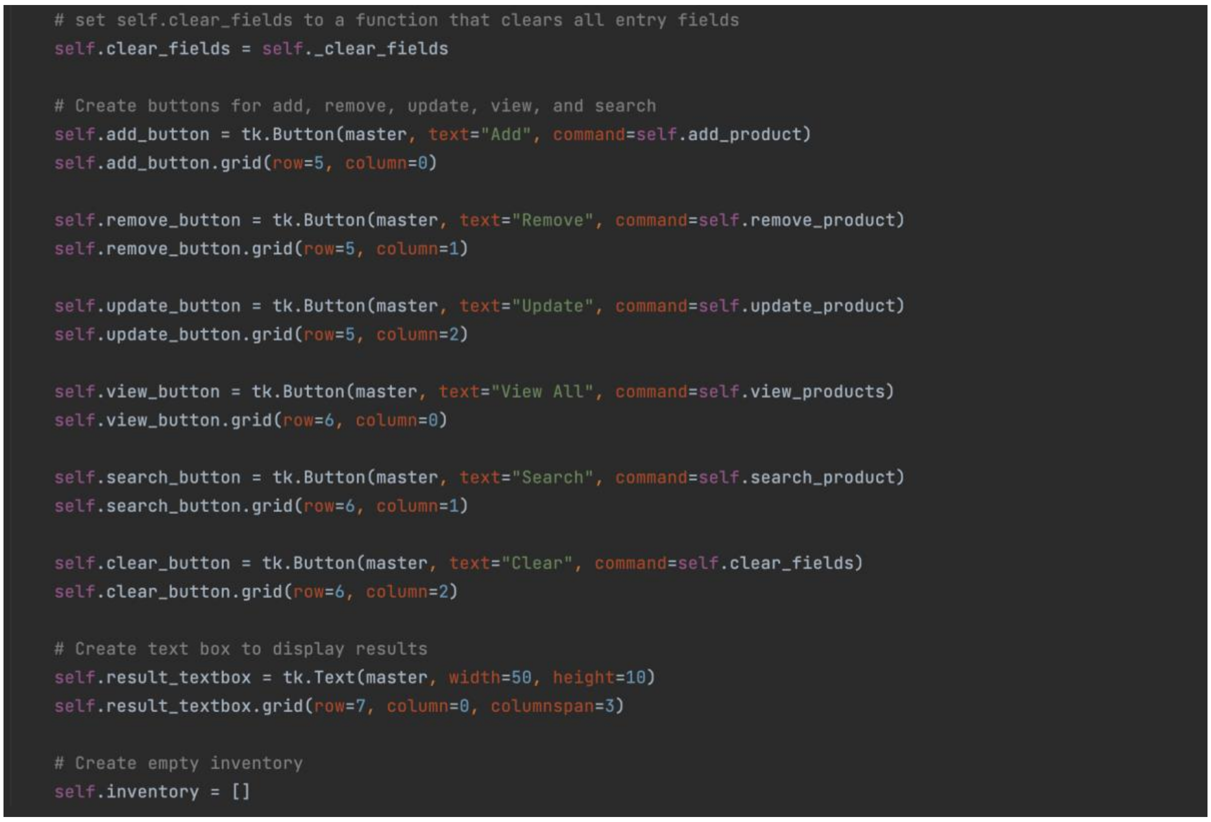
In the inventory database, the 'add_product_to_db' function stores data concerning products. This function has five fields: 'item_num', 'name', 'price, 'quantity', and 'ed'. These arguments represent the item number, name, cost, amount, and expiration date of the product. To interact with the SQLite database, this function utilizes Python's native SQLite3 module. The 'sqlite3.connect' method is utilized to establish a link to the database as the procedure's initial step. This method produces a connection object that can be employed to communicate with the database. The connect method takes as an argument the name of the database file. Subsequently, the 'conn.cursor()' method is used to initiate a cursor object.

SQL instructions are executed inside the database using the cursor object. This time, the cursor will implement a SQL query to insert the product information into the database. The cursor object's 'execute' method takes the SQL query as a string construction and executes it. The 'inventory' table in the database is populated with the product details through the query, which is an INSERT statement. When the query is activated, the parameterized values demonstrated in the query as question marks are supplanted by actual values. The values are given to the 'execute' procedure as a bundle in the subsequent debate.

Once the query has been executed, the database modifications are ratified by utilising the 'conn.commit()' technique. Then, the 'conn.close()' procedure is employed to shut down the database link. In the event that a difficulty arises during the query operation, an exception is thrown, and the issue will be displayed on the screen. The 'print' command sends a mistake message to the console after this exception is detected via a try-except block. This facility is useful when a business needs to monitor its stock by storing product details in a database. Inputting data about the goods into a repository enables the organization to rapidly access, revise, and erase documents as needed. Further capabilities can be incorporated into the process, such as verifying for repeat entries or altering already present records.



```python
def __init__(self, master):
    self.clear_fields = None
    self.master = master
    master.title("Inventory Management System")

    # Create label and entry for item number
    self.item_num_label = tk.Label(master, text="Item Number:")
    self.item_num_label.grid(row=0, column=0)
    self.item_num_entry = tk.Entry(master)
    self.item_num_entry.grid(row=0, column=1)

    # Create label and entry for product name
    self.name_label = tk.Label(master, text="Product Name:")
    self.name_label.grid(row=1, column=0)
    self.name_entry = tk.Entry(master)
    self.name_entry.grid(row=1, column=1)

    # Create label and entry for product price
    self.price_label = tk.Label(master, text="Product Price:")
    self.price_label.grid(row=2, column=0)
    self.price_entry = tk.Entry(master)
    self.price_entry.grid(row=2, column=1)

    # Create label and entry for product quantity
    self.quantity_label = tk.Label(master, text="Product Quantity:")
    self.quantity_label.grid(row=3, column=0)
    self.quantity_entry = tk.Entry(master)
    self.quantity_entry.grid(row=3, column=1)

    # Create label and entry for product expiry date
    self.ed_label = tk.Label(master, text="Expiry Date (YYYY/MM/DD):")
    self.ed_label.grid(row=4, column=0)
    self.ed_entry = tk.Entry(master)
    self.ed_entry.grid(row=4, column=1)
```

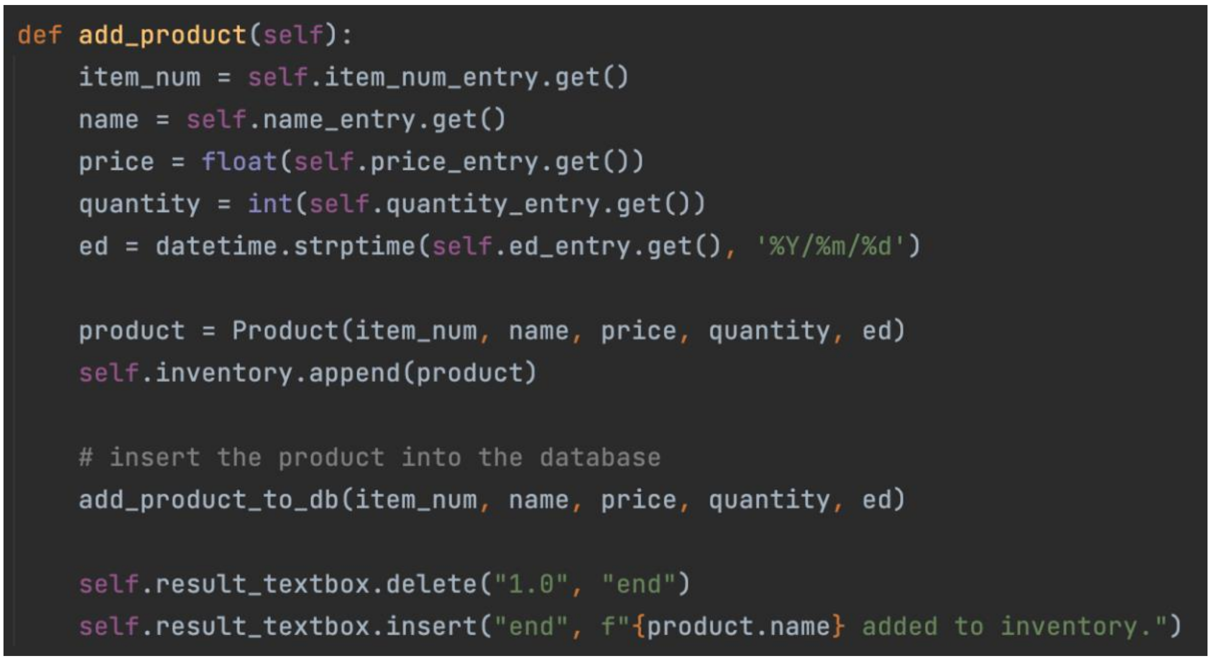


```python
        # set self.clear_fields to a function that clears all entry fields
        self.clear_fields = self._clear_fields

        # Create buttons for add, remove, update, view, and search
        self.add_button = tk.Button(master, text="Add", command=self.add_product)
        self.add_button.grid(row=5, column=0)

        self.remove_button = tk.Button(master, text="Remove", command=self.remove_product)
        self.remove_button.grid(row=5, column=1)

        self.update_button = tk.Button(master, text="Update", command=self.update_product)
        self.update_button.grid(row=5, column=2)

        self.view_button = tk.Button(master, text="View All", command=self.view_products)
        self.view_button.grid(row=6, column=0)

        self.search_button = tk.Button(master, text="Search", command=self.search_product)
        self.search_button.grid(row=6, column=1)

        self.clear_button = tk.Button(master, text="Clear", command=self.clear_fields)
        self.clear_button.grid(row=6, column=2)

        # Create text box to display results
        self.result_textbox = tk.Text(master, width=50, height=10)
        self.result_textbox.grid(row=7, column=0, columnspan=3)

        # Create empty inventory
        self.inventory = []
```

The ‘__init__’ method of the class is specified by the code supplied. This function is employed to set up the characteristics and values of an object of the class when it is created. The only parameter given to the constructor is an example of the ‘Tk’ class called ‘master’. This is the primary window of the graphical user interface program. The constructor applies the ‘title’ technique to alter the header of the main window to “Inventory Management System”.

The constructor then generates labels and entry fields for the product name, price, quantity, and expiration date by using the ‘Label’ and ‘Entry’ classes from the ‘tkinter’ module. The ‘row’ and ‘column’ parameters of the grid method are used to arrange these widgets in a ‘grid’, with each widget's location being specified. Following this, the constructor defines the self-function after building the entry fields. The entry fields are all cleared by the function ‘clear_fields’. Later, the function ‘add_product’ is used to add the product details to the listbox and the file.

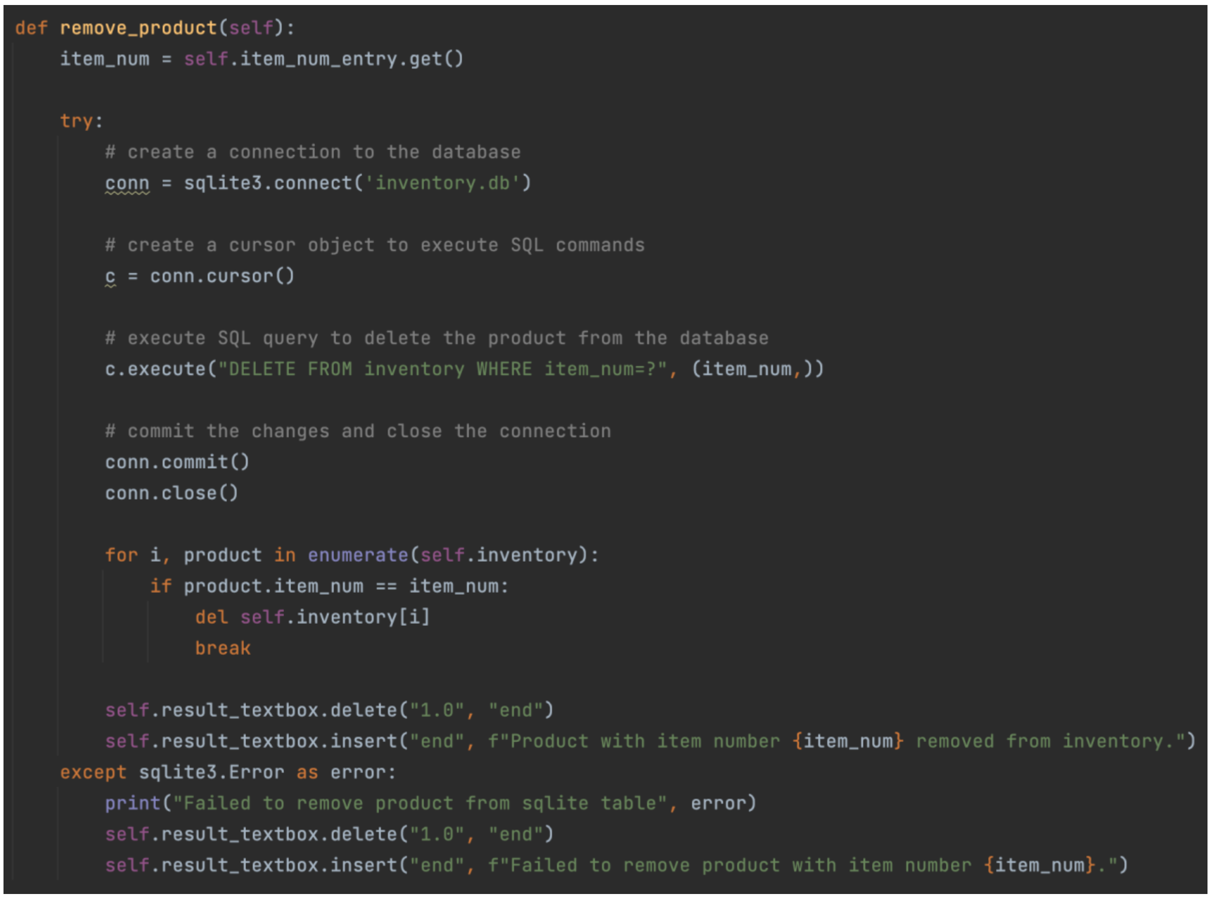
The ‘clear_fields’ attribute is given this task. A button which, when clicked, runs the ‘self.clear_fields’ function is also generated by the constructor, plus buttons for introducing, removing, refreshing, observing, and scanning products. Likewise, the ‘grid’ process is utilized to arrange these buttons on the grid. The ‘Text’ class is employed in the constructor to establish a text box widget that will demonstrate the results of the user's activities. This widget is placed on the grid as well. The constructor also initializes an empty list ‘self.inventory’, which will be used to store the product details entered by the user.



```python
def add_product(self):
    item_num = self.item_num_entry.get()
    name = self.name_entry.get()
    price = float(self.price_entry.get())
    quantity = int(self.quantity_entry.get())
    ed = datetime.strptime(self.ed_entry.get(), '%Y/%m/%d')

    product = Product(item_num, name, price, quantity, ed)
    self.inventory.append(product)

    # insert the product into the database
    add_product_to_db(item_num, name, price, quantity, ed)

    self.result_textbox.delete("1.0", "end")
    self.result_textbox.insert("end", f"{product.name} added to inventory.")
```

The ‘add_product’ method is called when the user clicks on the “Add” button in the GUI. Its purpose is to add a new product to the inventory and insert its details into the SQLite database. First, the method retrieves the user input from the text entry fields for item number, name, price, quantity, and expiry date. These values are extracted using the ‘get’ method of each corresponding text entry widget. The item number, name, and expiry date are strings, while the price and quantity are expected to be numeric values, so they are converted to the appropriate types using the ‘float’ and ‘int’ functions, respectively. Next, a ‘Product’ object is created using the input values. The ‘Product’ class is defined elsewhere in the code and represents a single product in the inventory. The product variable holds this newly created ‘Product’ object. After creating the ‘Product’ object, the method adds it to the ‘inventory’ list by appending it to the end of the list. This list holds all the products in the inventory and is also defined elsewhere in the code.

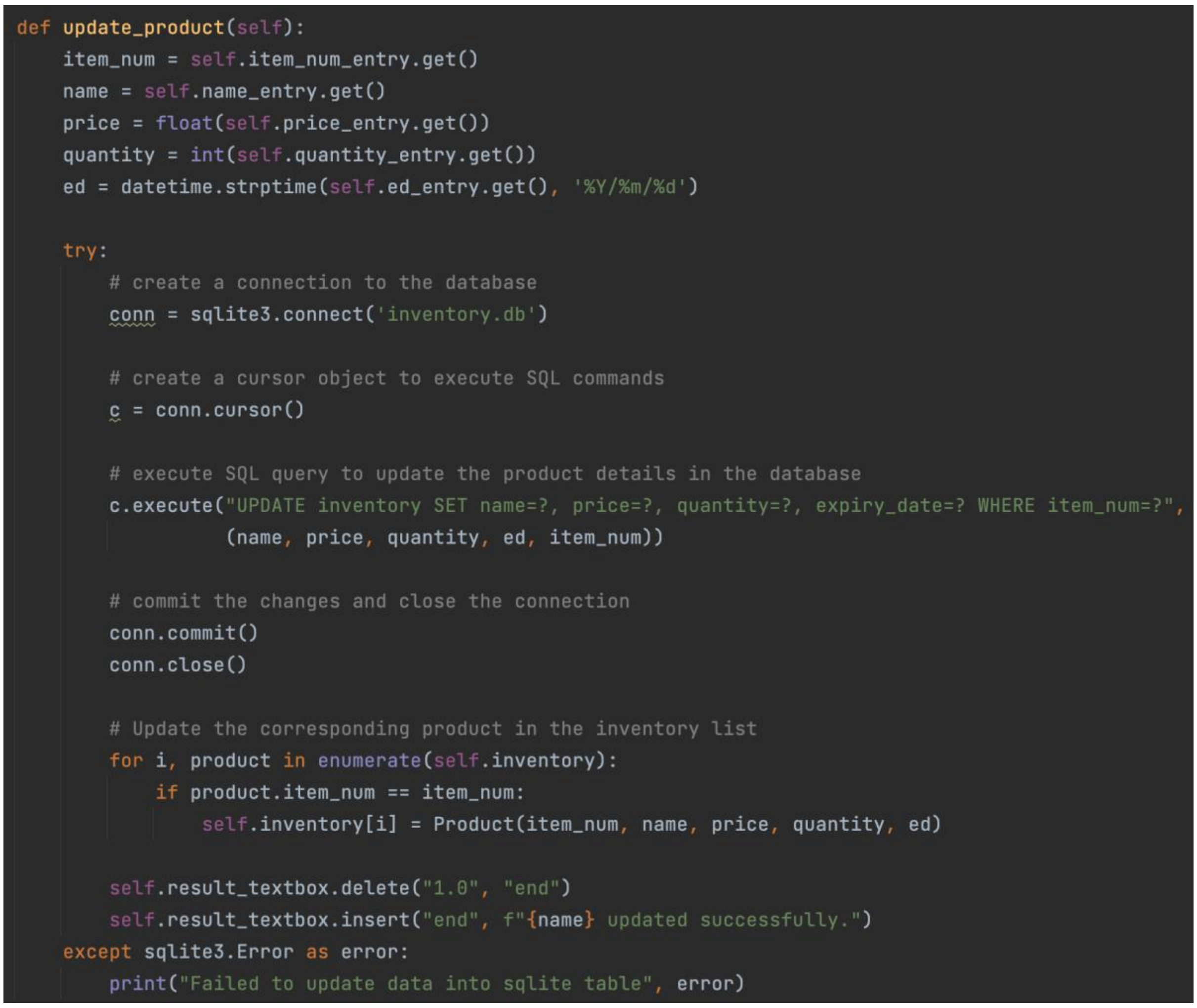
Finally, the method calls the ‘add_product_to_db’ function, passing in the product details as arguments. This function is defined elsewhere in the code and is responsible for inserting the product data into the SQLite database. It uses SQL commands to insert a new row into the ‘inventory’ table with the product details. After the product has been added to the inventory and inserted into the database, the method updates the ‘result_textbox’ widget with a message indicating that the product has been added. It does this by deleting the current contents of the widget using the delete method and then inserting the message using the ‘insert’ method. The ‘end’ parameter tells the ‘insert’ method to insert the text at the end of the widget. The final message displayed to the user is in the format “{product name} added to inventory.”.



```python
def remove_product(self):
    item_num = self.item_num_entry.get()

    try:
        # create a connection to the database
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to delete the product from the database
        c.execute("DELETE FROM inventory WHERE item_num=?", (item_num,))

        # commit the changes and close the connection
        conn.commit()
        conn.close()

        for i, product in enumerate(self.inventory):
            if product.item_num == item_num:
                del self.inventory[i]
                break

        self.result_textbox.delete("1.0", "end")
        self.result_textbox.insert("end", f"Product with item number {item_num} removed from inventory.")
    except sqlite3.Error as error:
        print("Failed to remove product from sqlite table", error)
        self.result_textbox.delete("1.0", "end")
        self.result_textbox.insert("end", f"Failed to remove product with item number {item_num}.")
```

The ‘remove_product’ function is responsible for eliminating a product from the storehouse. It collects the product's item number from the GUI’s entry field and runs a DELETE SQL request on the inventory table with that number as a parameter. It will spot any mistakes that may arise during the performance of the SQL command and inform the user via the GUI's ‘result_textbox’ if they do. After deleting the product from the database, the function will search for the product with the same item number in the inventory listing.

Once the piece is found, it erases the relevant component to eliminate the product from the stock rundown. To cycle through the stock list and monitor every component’s list, the capacity uses the ‘enumerate’ work. It searches for the thing to be erased, disposes of it utilizing the ‘delete’ proclamation, and afterward leaves the circle. When the evacuation has been effectively finished or not, the capacity refreshes the GUI’s ‘result_textbox’ to show a message. On the off chance that the evacuation was fruitful, the thing will be recorded as having been taken out from stock. A message specifying that the item could not be taken out will be displayed if the extraction was unsuccessful.

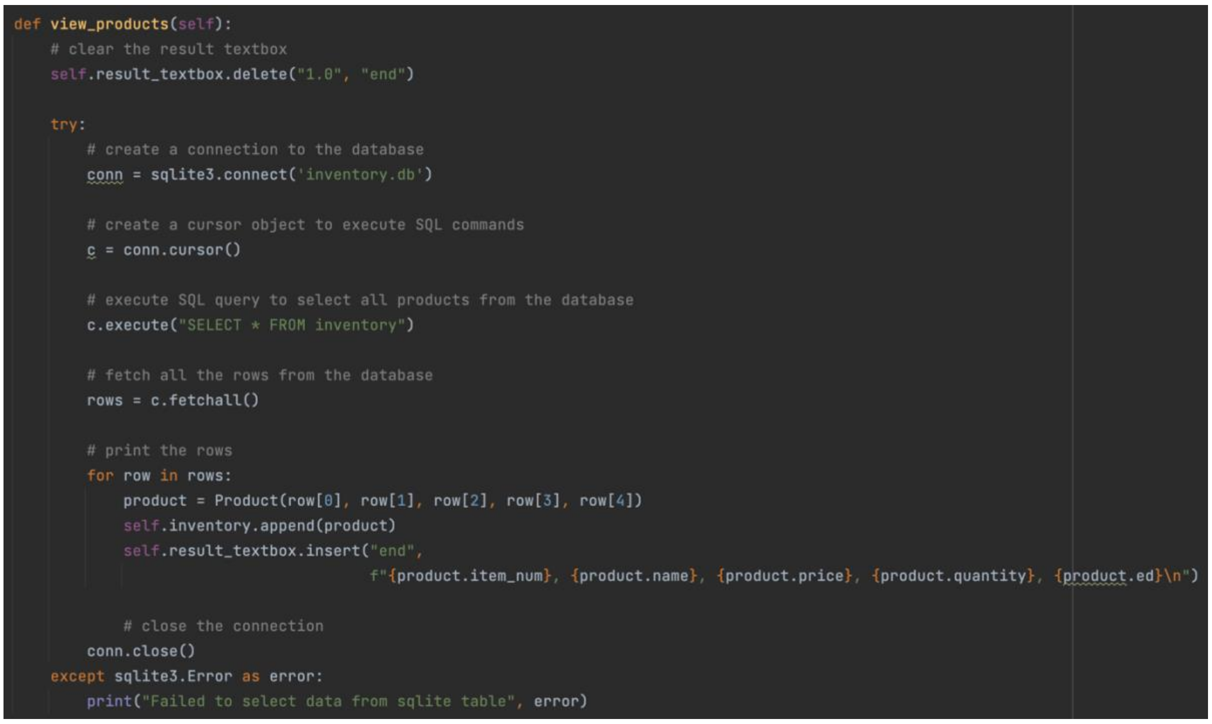


```python
def update_product(self):
    item_num = self.item_num_entry.get()
    name = self.name_entry.get()
    price = float(self.price_entry.get())
    quantity = int(self.quantity_entry.get())
    ed = datetime.strptime(self.ed_entry.get(), '%Y/%m/%d')

    try:
        # create a connection to the database
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to update the product details in the database
        c.execute("UPDATE inventory SET name=?, price=?, quantity=?, expiry_date=? WHERE item_num=?",
                  (name, price, quantity, ed, item_num))

        # commit the changes and close the connection
        conn.commit()
        conn.close()

        # Update the corresponding product in the inventory list
        for i, product in enumerate(self.inventory):
            if product.item_num == item_num:
                self.inventory[i] = Product(item_num, name, price, quantity, ed)

        self.result_textbox.delete("1.0", "end")
        self.result_textbox.insert("end", f"{name} updated successfully.")
    except sqlite3.Error as error:
        print("Failed to update data into sqlite table", error)
```

The ‘update_product’ method updates the details of a product in the inventory. It takes input from the user in the form of the item number, name, price, quantity, and expiry date of the product. First, the method gets the input values using the ‘get()’ method of the ‘Entry’ widget for each input field. The ‘get()’ method returns a string, so the values for ‘price’ and ‘quantity’ are converted to ‘float’ and ‘int’ respectively. Then, the method attempts to create a connection to the SQLite database using the ‘connect()’ method of the ‘sqlite3’ module. If successful, a cursor object is created using the ‘cursor()’ method of the connection object.

The method then executes an SQL query to update the product details in the database using the ‘execute()’ method of the cursor object. The query uses placeholders to pass in the updated values for the name, price, quantity, expiry date, and item number. These values are passed as a tuple as the second argument to the ‘execute()’ method. If the query is executed successfully, the changes are committed to the database using the ‘commit()’ method of the connection object and the connection is closed using the ‘close()’ method. The method then updates the corresponding product in the inventory list by iterating over the list using the ‘enumerate()’ function to access the index of each element as well as its value. If the ‘item_num’ of the product matches the ‘item_num’ input value, the product details are updated using the ‘Product’ class.

Finally, the method clears the ‘result_textbox’ using the ‘delete()’ method and inserts a message indicating that the product was updated successfully. If an error occurs while updating the product details in the database, the method catches the error using a ‘try’ and ‘except’ block and prints an error message to the console. It also updates the ‘result_textbox’ with a message indicating that the product update failed.

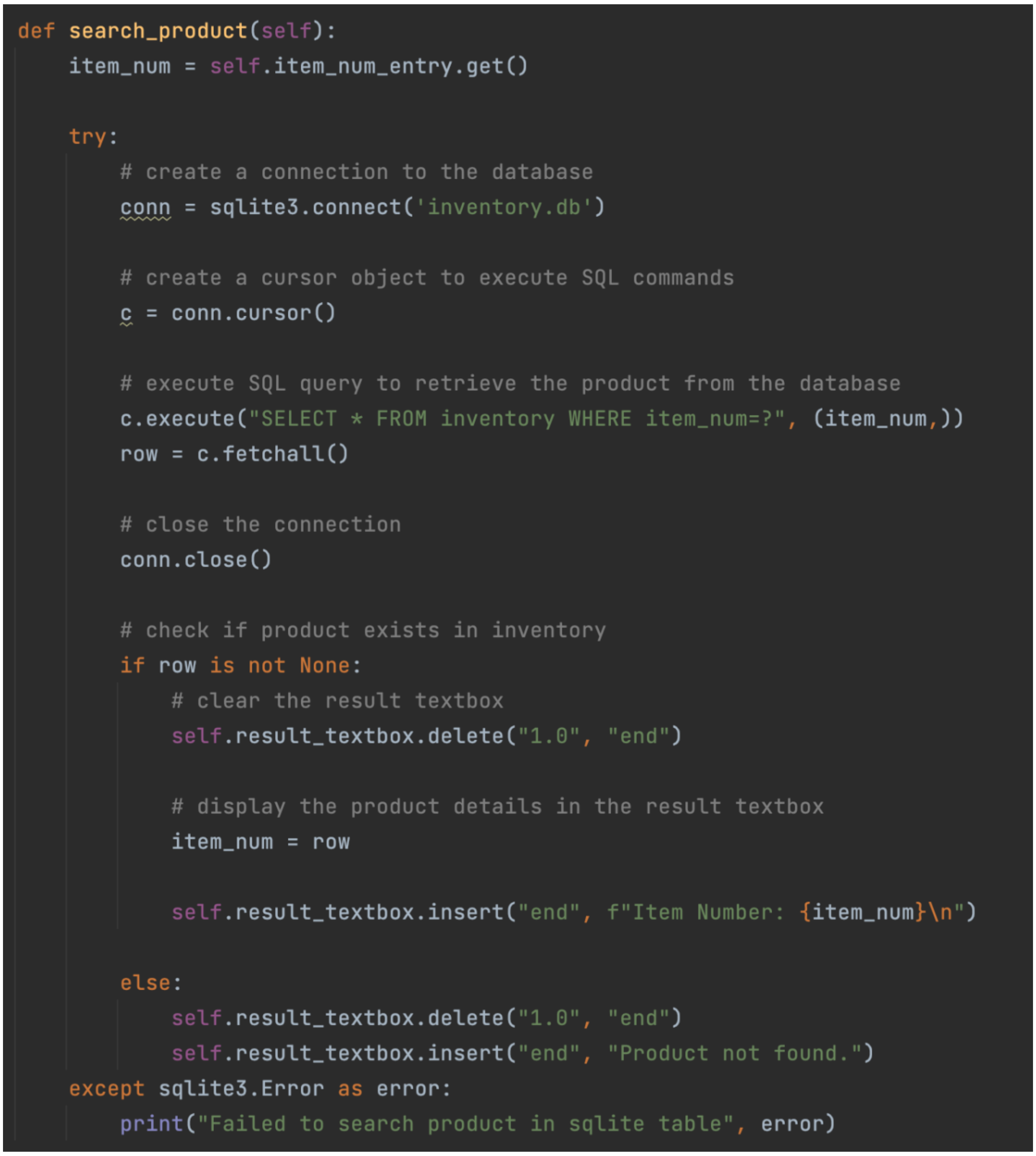


```python
def view_products(self):
    # clear the result textbox
    self.result_textbox.delete("1.0", "end")

    try:
        # create a connection to the database
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to select all products from the database
        c.execute("SELECT * FROM inventory")

        # fetch all the rows from the database
        rows = c.fetchall()

        # print the rows
        for row in rows:
            product = Product(row[0], row[1], row[2], row[3], row[4])
            self.inventory.append(product)
            self.result_textbox.insert("end",
                                       f"{product.item_num}, {product.name}, {product.price}, {product.quantity}, {product.ed}\n")

            # close the connection
        conn.close()
    except sqlite3.Error as error:
        print("Failed to select data from sqlite table", error)
```

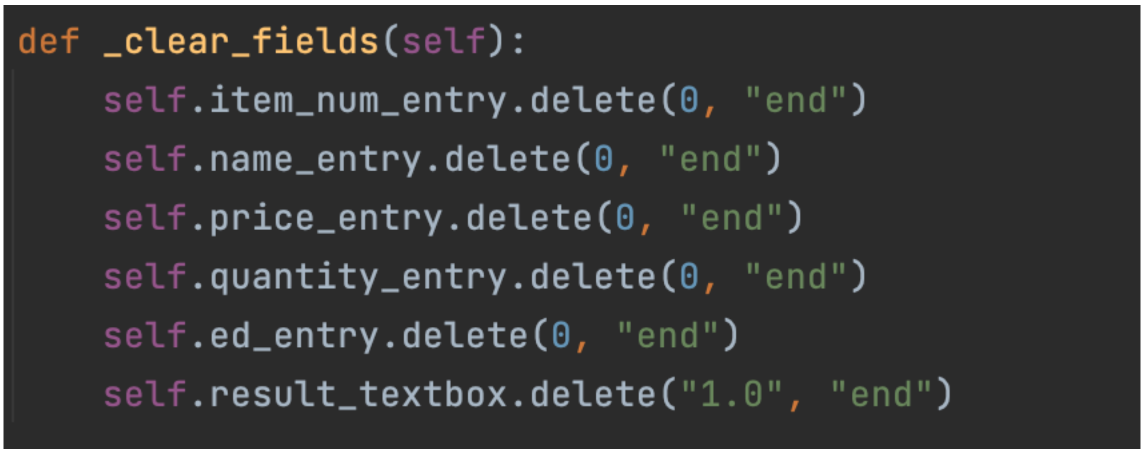
The ‘view_products()’ method is a component of a Python-coded GUI for an inventory management system. Whenever a user desires to view every item currently in stock, they can invoke this procedure. To begin, the text box where the results will be presented must be cleared. This is done with the assistance of the ‘result_textbox’ object’s ‘delete()’ method. Subsequently, the method strives to link up with the ‘inventory.db’ sqlite database file. The database file must be in the same folder as the Python script.

The ‘connect()’ command of the ‘sqlite3’ module is used to achieve this. When the link is properly established, a cursor object called ‘c’ is generated to execute SQL statements on the database. The next step is to execute an SQL query to pick out all elements from the database’s ‘inventory’ table. This is done through the cursor object’s ‘execute()’ function. The SQL command “SELECT * FROM inventory” is used to acquire all of the columns and rows from the ‘inventory’ table. Subsequently, the ‘fetchall()’ method is applied to collect all of the rows retrieved from the query as a list of tuples. Every tuple is indicative of a row from the inventory table, with the values for all its columns included.

All the rows extracted from the database are gone through one-by-one using a ‘for’ loop. The data in each row is utilized to create a new ‘Product’ object. After that, the ‘inventory’ list, which contains ‘Product’ objects indicating the latest state of the inventory, is updated to include the ‘Product’ object. Lastly, the values of the ‘Product’ object is displayed on the GUI using the ‘insert()’ method of the ‘result_textbox’ object. To make sure the new text is added at the end of the current text in the textbox, the ‘end’ parameter of the ‘insert()’ method is used. The ‘Product’ entity’s characteristics are employed to create a phrase, which is then put into the textbox. Making use of the ‘close()’ feature of the connection entity, the database link is at long last severed. A misstep notification is printed to the console if one appears while the procedure is being utilized.


```
def search_product(self):
    item_num = self.item_num_entry.get()

    try:
        # create a connection to the database
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to retrieve the product from the database
        c.execute("SELECT * FROM inventory WHERE item_num=?", (item_num,))
        row = c.fetchall()

        # close the connection
        conn.close()

        # check if product exists in inventory
        if row is not None:
            # clear the result textbox
            self.result_textbox.delete("1.0", "end")

            # display the product details in the result textbox
            item_num = row

            self.result_textbox.insert("end", f"Item Number: {item_num}\n")

        else:
            self.result_textbox.delete("1.0", "end")
            self.result_textbox.insert("end", "Product not found.")
    except sqlite3.Error as error:
        print("Failed to search product in sqlite table", error)
```
The inventory management system's 'search_product()' function allows an individual to input an item number to look for a certain commodity in the inventory. The 'item_num_entry' attribute is used by the function to acquire the item number that was keyed in by the user. Following that, a tie-in is made to the 'inventory.db' database utilizing the 'sqlite3.connect()' system and creates a cursor object 'c' to execute SQL operations on the database. Subsequently, the user-indicated item number is acquired from the database using the SQL command "SELECT". The person enters an SQL command and a product number as parameters, which are sent to the cursor item's 'execute()' function. Subsequently, to get each row that fulfils the search criteria, the cursor item's 'fetchall()' method is utilized. The 'row' variable stores the obtained rows. Once the database has given back the item details, the bond is broken by calling the 'close()' method.

To ascertain the presence of the item in stock, the function double-checks that the obtained 'row' variable is not 'None'. The 'delete()' method is applied to erase the result textbox in the occurrence that the product is discovered, and the info of the product is screened there through the 'insert()' technique. If the product cannot be identified, the result textbox is cleared and the sentence "Product not found." is displayed. In the result textbox, it has been stated. If an issue arises while looking for a product in the database, the function utilizes the 'print()' function to output a mistake notification and show the message "Failed to locate product in sqlite table" in the output textbox.

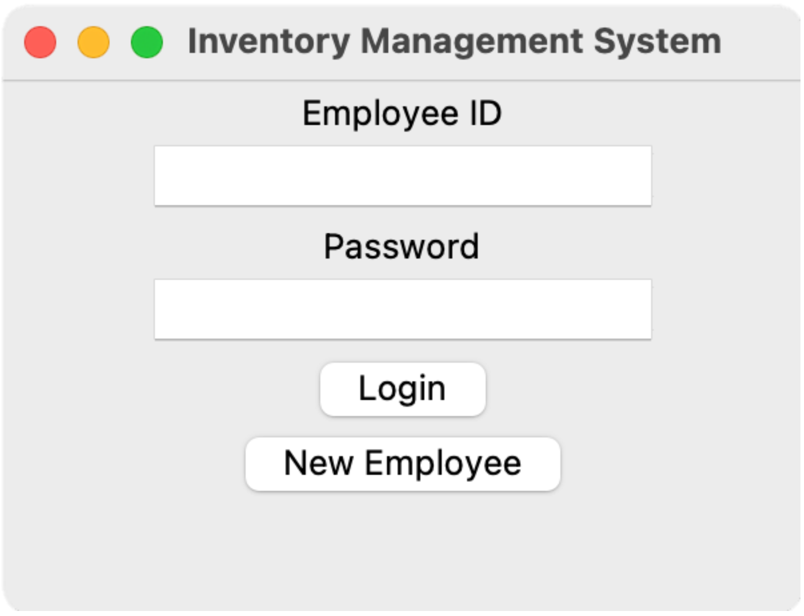

```
def _clear_fields(self):
    self.item_num_entry.delete(0, "end")
    self.name_entry.delete(0, "end")
    self.price_entry.delete(0, "end")
    self.quantity_entry.delete(0, "end")
    self.ed_entry.delete(0, "end")
    self.result_textbox.delete("1.0", "end")
```

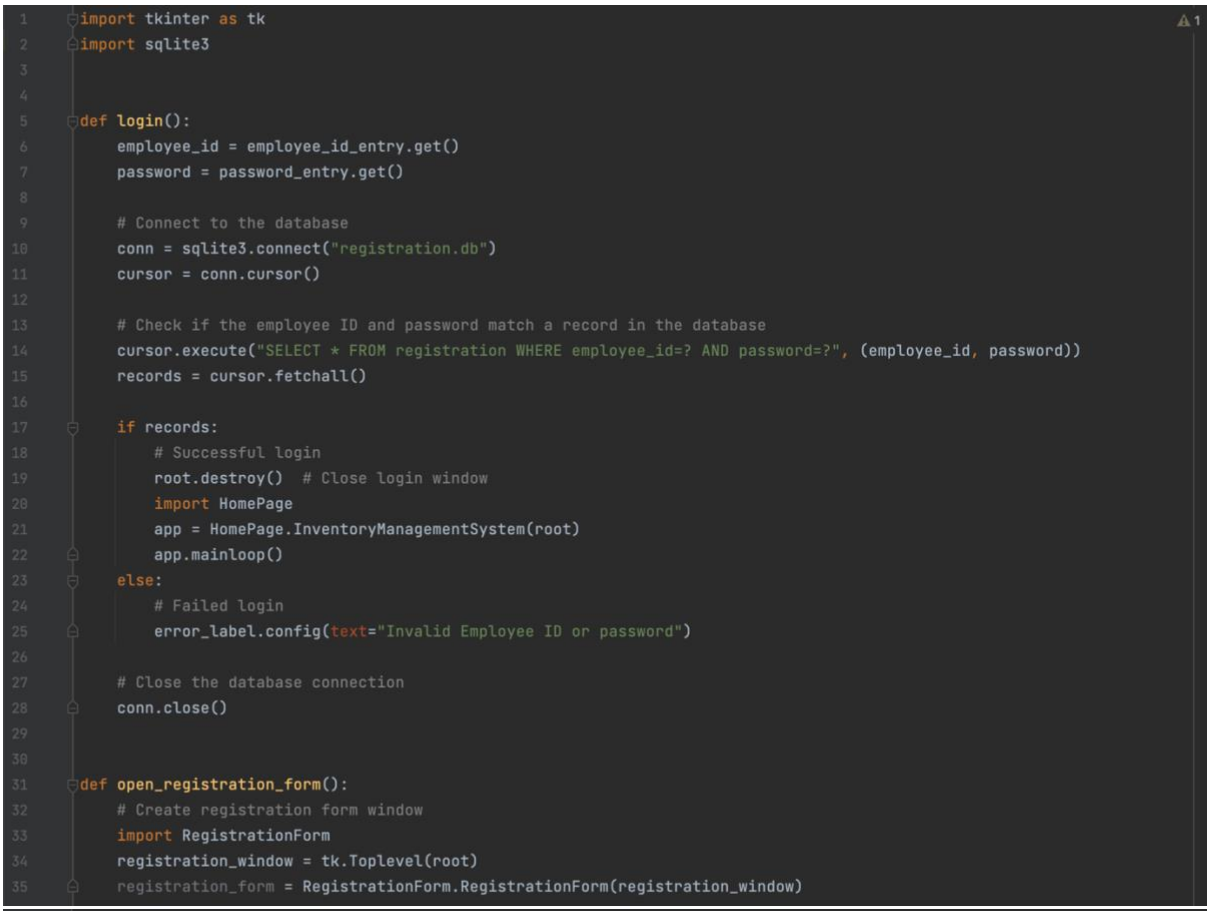
The private helper method '_clear_fields' wipes off the data from the GUI result textbox and all input fields. Every time the user clicks the 'Clear' button in the GUI or performs any other manoeuvre that modifies the displayed info, this method is called. It carries out its task by using the 'delete' method to get rid of any existing text in each input box. The 'delete' method necessitates two parameters - the start index and the end index of the text to be removed. In this case, the start index is set to 0, signifying the deletion will start at the start of the field, and the end index is specified as 'end', which means the deletion will go right up to the conclusion of the field.

The result textbox's 'erase' method is subsequently utilized by the procedure to get rid of any existing words there. The 'clear' method requires the same two parameters as the input fields; in this case, the start index is '1.0', meaning the deletion commences with the first character of the first line, and the end index is 'end', meaning the deletion carries on up to the last letter of the text in the textbox.

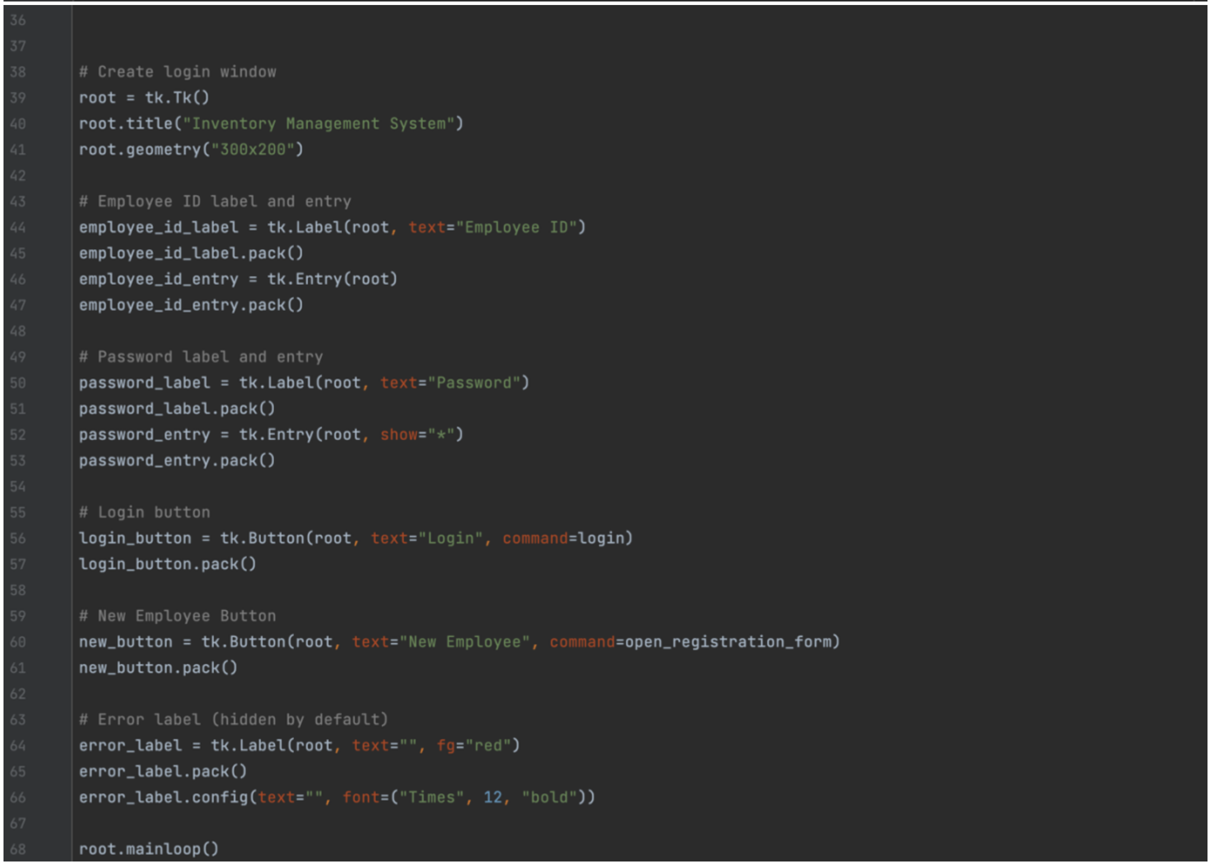
<u>3. Screenshot of coding and output (Explain the general output of whole code)</u>

Login.py





```python
import tkinter as tk
import sqlite3


def login():
    employee_id = employee_id_entry.get()
    password = password_entry.get()

    # Connect to the database
    conn = sqlite3.connect("registration.db")
    cursor = conn.cursor()

    # Check if the employee ID and password match a record in the database
    cursor.execute("SELECT * FROM registration WHERE employee_id=? AND password=?", (employee_id, password))
    records = cursor.fetchall()

    if records:
        # Successful login
        root.destroy()  # Close login window
        import HomePage
        app = HomePage.InventoryManagementSystem(root)
        app.mainloop()
    else:
        # Failed login
        error_label.config(text="Invalid Employee ID or password")

    # Close the database connection
    conn.close()


def open_registration_form():
    # Create registration form window
    import RegistrationForm
    registration_window = tk.Toplevel(root)
    registration_form = RegistrationForm.RegistrationForm(registration_window)
```

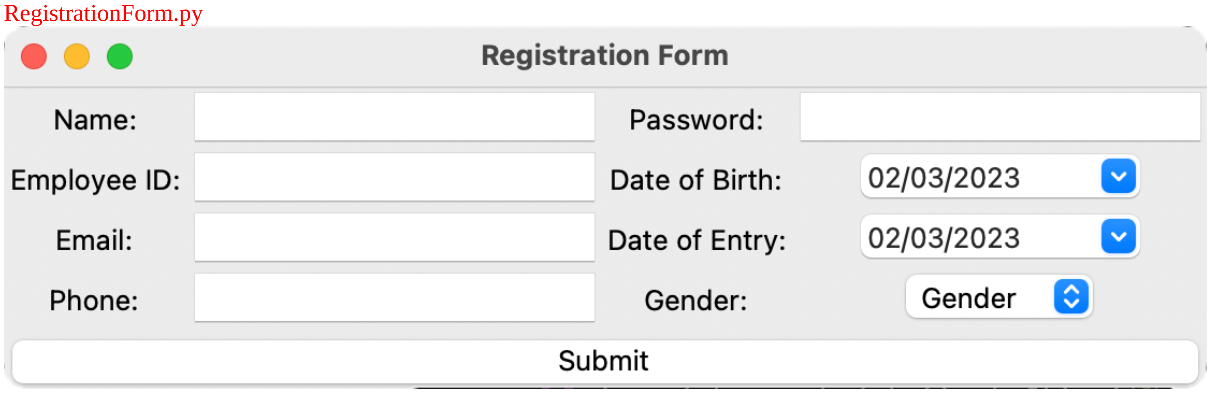


```python


# Create login window
root = tk.Tk()
root.title("Inventory Management System")
root.geometry("300x200")

# Employee ID label and entry
employee_id_label = tk.Label(root, text="Employee ID")
employee_id_label.pack()
employee_id_entry = tk.Entry(root)
employee_id_entry.pack()

# Password label and entry
password_label = tk.Label(root, text="Password")
password_label.pack()
password_entry = tk.Entry(root, show="*")
password_entry.pack()

# Login button
login_button = tk.Button(root, text="Login", command=login)
login_button.pack()

# New Employee Button
new_button = tk.Button(root, text="New Employee", command=open_registration_form)
new_button.pack()

# Error label (hidden by default)
error_label = tk.Label(root, text="", fg="red")
error_label.pack()
error_label.config(text="", font=("Times", 12, "bold"))

root.mainloop()
```

This Python code produces a rudimentary graphical user interface (GUI) log-in window for an inventory control system. ‘tkinter’ is a module employed to construct the GUI. In order to gain access to the system, the individual must submit their employee ID and password. If the log-in process is successful, the window will close, and the user will be directed to the inventory regulation system’s home page. Failed log-in attempts will result in an error message being presented.

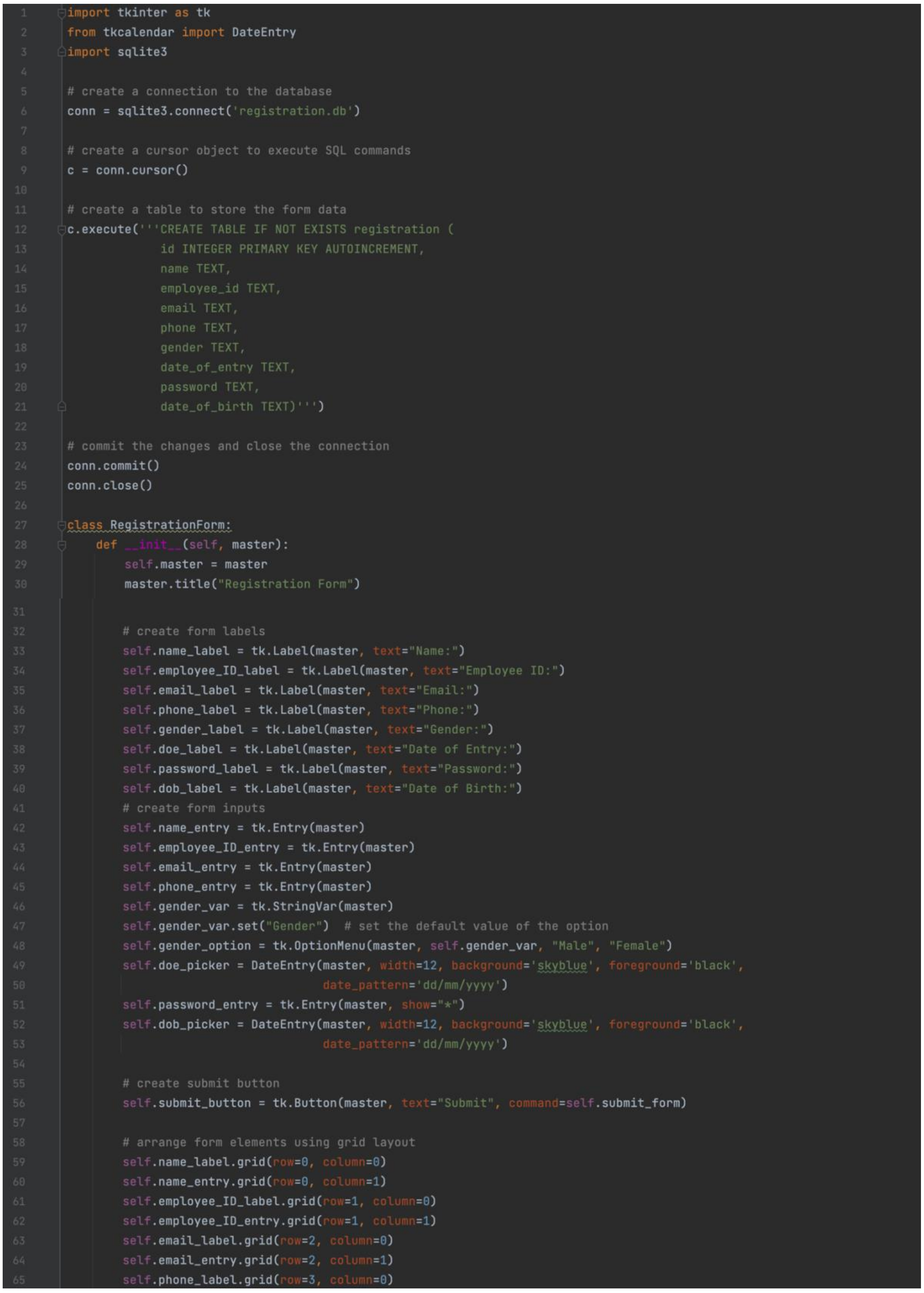
An SQLite database labelled ‘registration.db’ is utilized to compare the login details. To evaluate if the employee ID and password match a record in the database, the code links to the database utilizing the SQLite3 module and runs a query. If there is a match, the person is given access to the system. The login window has a “New Employee” button which, when pressed, brings up a new window where the user can input their registration information. The same SQLite database employed for logging in also stores the registration information.

RegistrationForm.py

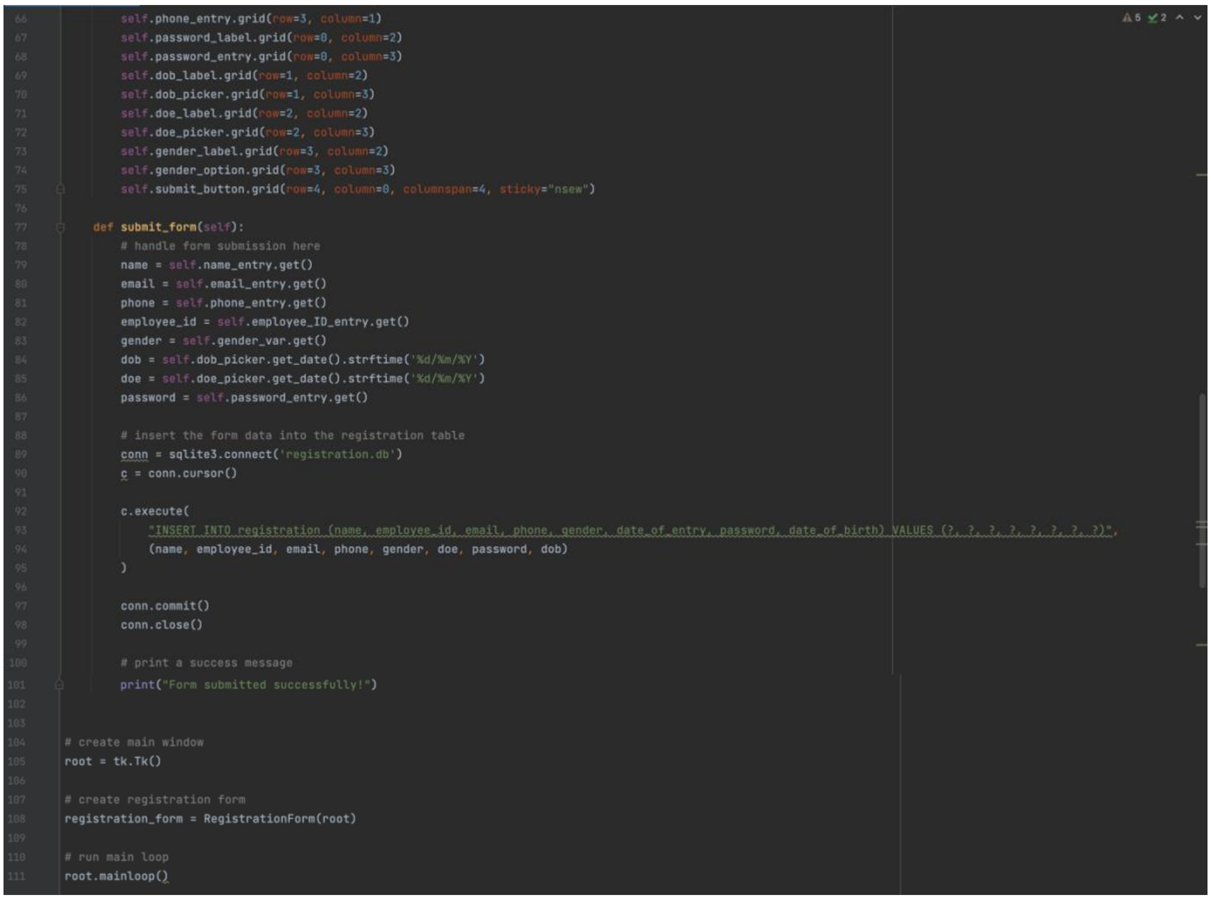


Registration Form

Name:		Password:	
Employee ID:		Date of Birth:	02/03/2023
Email:		Date of Entry:	02/03/2023
Phone:		Gender:	Gender



```
import tkinter as tk
from tkcalendar import DateEntry
import sqlite3

# create a connection to the database
conn = sqlite3.connect('registration.db')

# create a cursor object to execute SQL commands
c = conn.cursor()

# create a table to store the form data
c.execute('''CREATE TABLE IF NOT EXISTS registration (
             id INTEGER PRIMARY KEY AUTOINCREMENT,
             name TEXT,
             employee_id TEXT,
             email TEXT,
             phone TEXT,
             gender TEXT,
             date_of_entry TEXT,
             password TEXT,
             date_of_birth TEXT)''')

# commit the changes and close the connection
conn.commit()
conn.close()

class RegistrationForm:
    def __init__(self, master):
        self.master = master
        master.title("Registration Form")

        # create form labels
        self.name_label = tk.Label(master, text="Name:")
        self.employee_ID_label = tk.Label(master, text="Employee ID:")
        self.email_label = tk.Label(master, text="Email:")
        self.phone_label = tk.Label(master, text="Phone:")
        self.gender_label = tk.Label(master, text="Gender:")
        self.doe_label = tk.Label(master, text="Date of Entry:")
        self.password_label = tk.Label(master, text="Password:")
        self.dob_label = tk.Label(master, text="Date of Birth:")
        # create form inputs
        self.name_entry = tk.Entry(master)
        self.employee_ID_entry = tk.Entry(master)
        self.email_entry = tk.Entry(master)
        self.phone_entry = tk.Entry(master)
        self.gender_var = tk.StringVar(master)
        self.gender_var.set("Gender")  # set the default value of the option
        self.gender_option = tk.OptionMenu(master, self.gender_var, "Male", "Female")
        self.doe_picker = DateEntry(master, width=12, background='skyblue', foreground='black',
                                    date_pattern='dd/mm/yyyy')
        self.password_entry = tk.Entry(master, show="*")
        self.dob_picker = DateEntry(master, width=12, background='skyblue', foreground='black',
                                    date_pattern='dd/mm/yyyy')

        # create submit button
        self.submit_button = tk.Button(master, text="Submit", command=self.submit_form)

        # arrange form elements using grid layout
        self.name_label.grid(row=0, column=0)
        self.name_entry.grid(row=0, column=1)
        self.employee_ID_label.grid(row=1, column=0)
        self.employee_ID_entry.grid(row=1, column=1)
        self.email_label.grid(row=2, column=0)
        self.email_entry.grid(row=2, column=1)
        self.phone_label.grid(row=3, column=0)
```

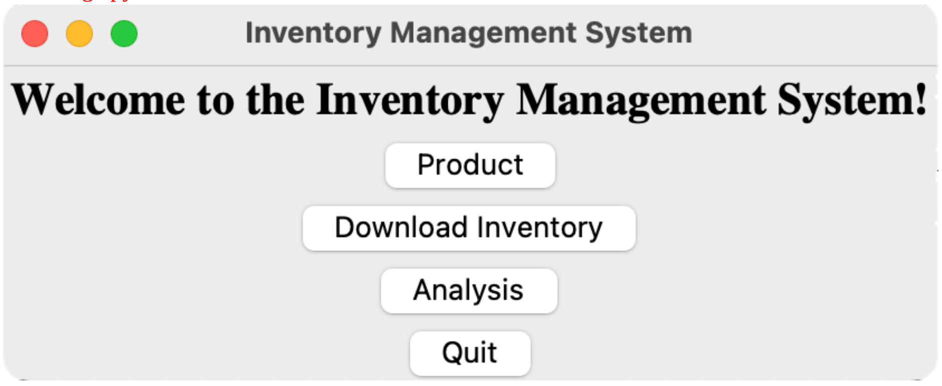


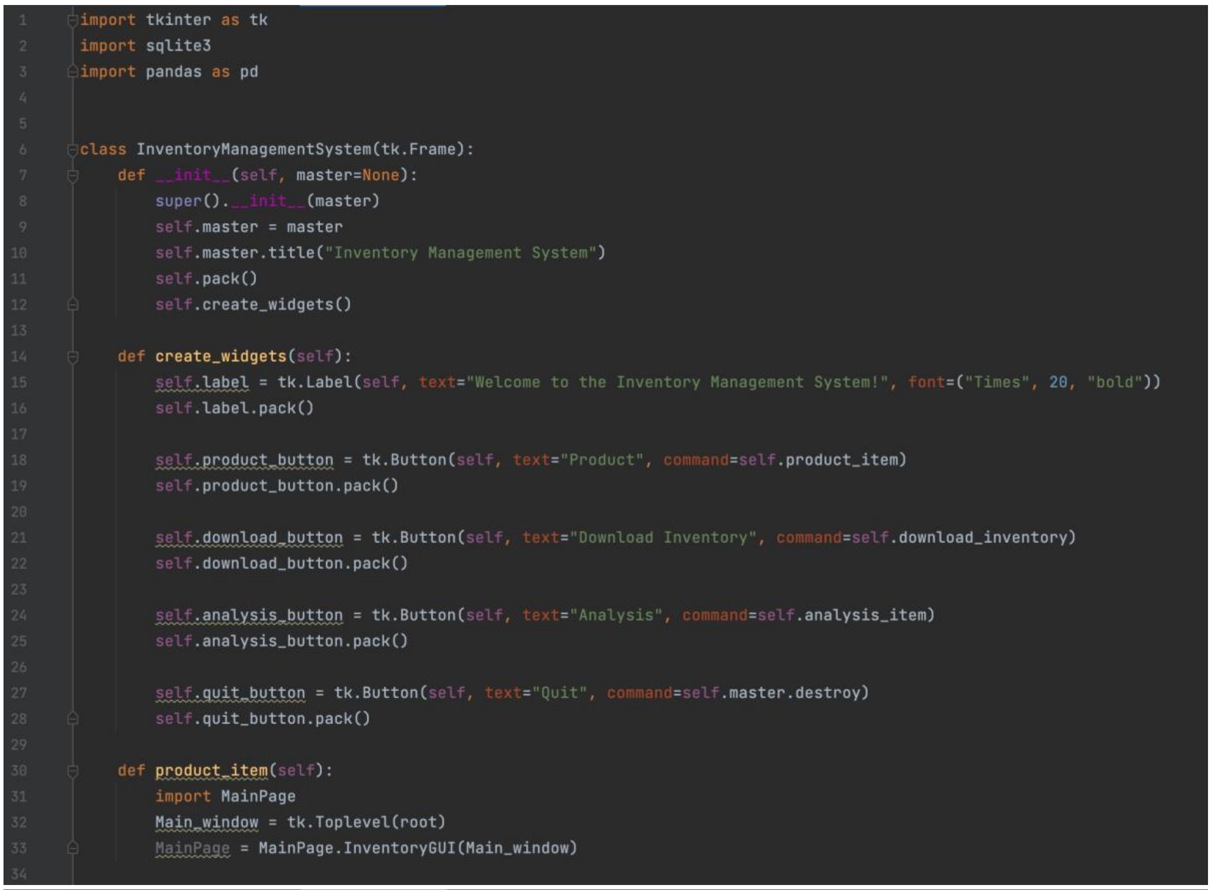
```python
        self.phone_entry.grid(row=3, column=1)
        self.password_label.grid(row=0, column=2)
        self.password_entry.grid(row=0, column=3)
        self.dob_label.grid(row=1, column=2)
        self.dob_picker.grid(row=1, column=3)
        self.doe_label.grid(row=2, column=2)
        self.doe_picker.grid(row=2, column=3)
        self.gender_label.grid(row=3, column=2)
        self.gender_option.grid(row=3, column=3)
        self.submit_button.grid(row=4, column=0, columnspan=4, sticky="nsew")

    def submit_form(self):
        # handle form submission here
        name = self.name_entry.get()
        email = self.email_entry.get()
        phone = self.phone_entry.get()
        employee_id = self.employee_ID_entry.get()
        gender = self.gender_var.get()
        dob = self.dob_picker.get_date().strftime('%d/%m/%Y')
        doe = self.doe_picker.get_date().strftime('%d/%m/%Y')
        password = self.password_entry.get()

        # insert the form data into the registration table
        conn = sqlite3.connect('registration.db')
        c = conn.cursor()

        c.execute(
            "INSERT INTO registration (name, employee_id, email, phone, gender, date_of_entry, password, date_of_birth) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
            (name, employee_id, email, phone, gender, doe, password, dob)
        )

        conn.commit()
        conn.close()

        # print a success message
        print("Form submitted successfully!")


# create main window
root = tk.Tk()

# create registration form
registration_form = RegistrationForm(root)

# run main loop
root.mainloop()
```

A GUI utilising the Python programming language is developed to establish a registration form. This form will collect personal information, such as names, email addresses, telephone numbers, dates of birth and entry dates. The tkinter library, the tkcalendar module and the SQLite database engine are all implemented by the program to create the GUI, present date selection and store form data, respectively. The sqlite3 library is used to generate a database connection object at the start of the program. This connection is then used to construct a cursor object which can execute SQL statements.

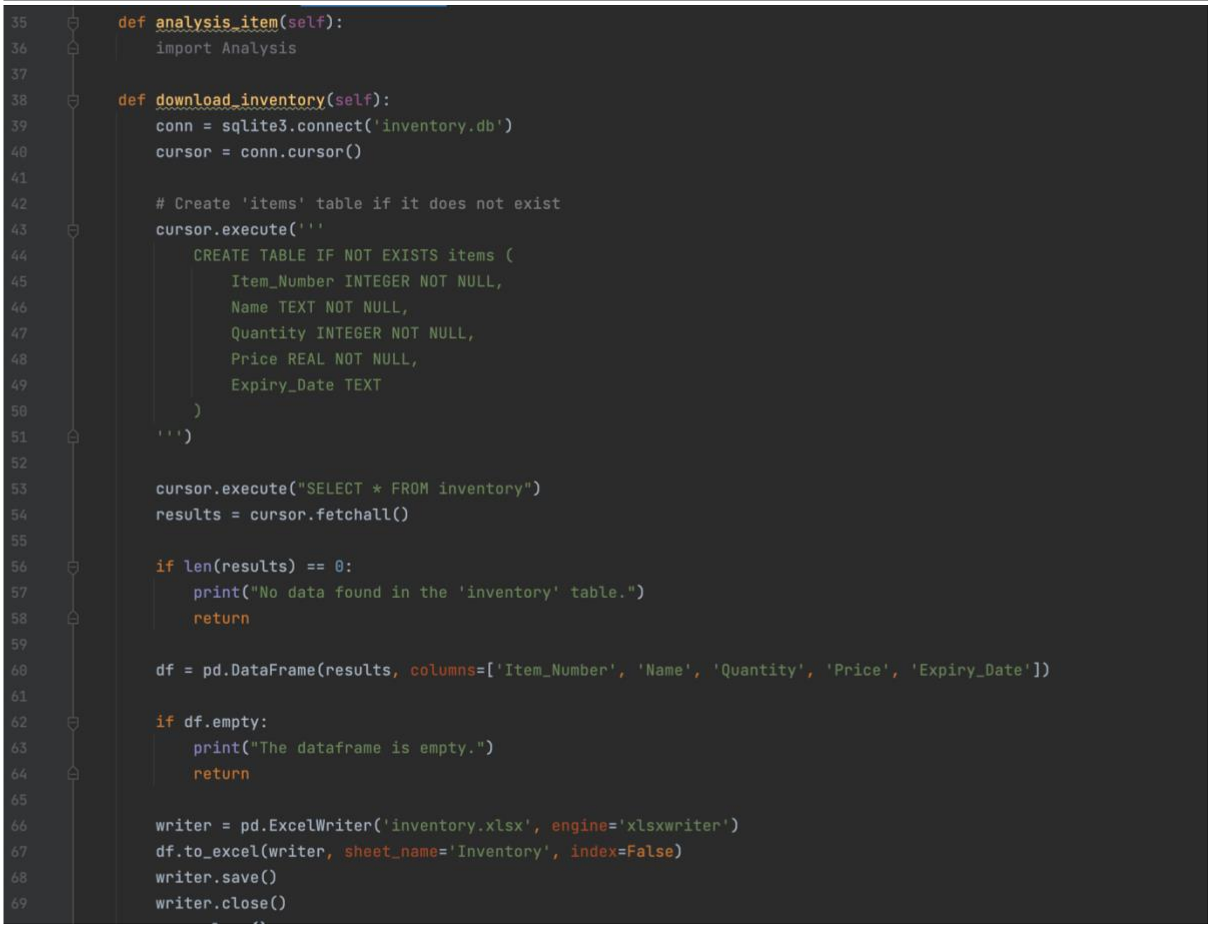
The program then makes a registration table in the database with fields for every form area. Subsequent to forming the database, the program forms a RegistrationForm class which hails from the tk. The tkinter frame class. For each of the form fields, this class includes the code required to fabricate labels and input fields. The form additionally has a submit button, which initiates a function that retrieves the values from the input fields and places them into the database's registration table. Eventually, the system initiates the RegistrationForm class, builds the primary window, and includes the form to it. Then, the essential loop of the program starts running, waiting for input from the user and dynamically updating the graphical interface as needed.

HomePage.py

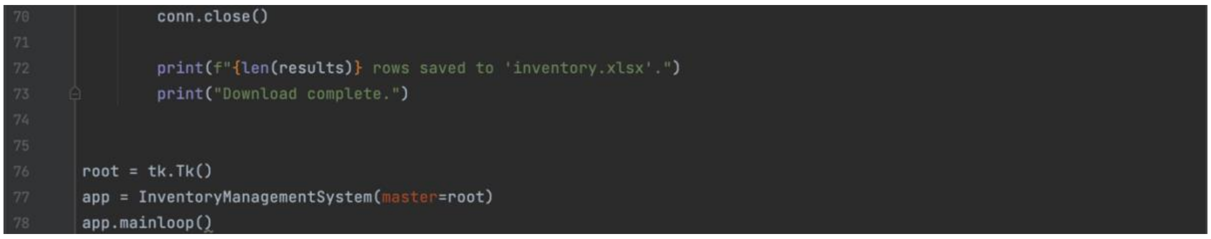

Inventory Management System
Welcome to the Inventory Management System!
Product
Download Inventory
Analysis
Quit



```python
import tkinter as tk
import sqlite3
import pandas as pd


class InventoryManagementSystem(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.master.title("Inventory Management System")
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.label = tk.Label(self, text="Welcome to the Inventory Management System!", font=("Times", 20, "bold"))
        self.label.pack()

        self.product_button = tk.Button(self, text="Product", command=self.product_item)
        self.product_button.pack()

        self.download_button = tk.Button(self, text="Download Inventory", command=self.download_inventory)
        self.download_button.pack()

        self.analysis_button = tk.Button(self, text="Analysis", command=self.analysis_item)
        self.analysis_button.pack()

        self.quit_button = tk.Button(self, text="Quit", command=self.master.destroy)
        self.quit_button.pack()

    def product_item(self):
        import MainPage
        Main_window = tk.Toplevel(root)
        MainPage = MainPage.InventoryGUI(Main_window)

```



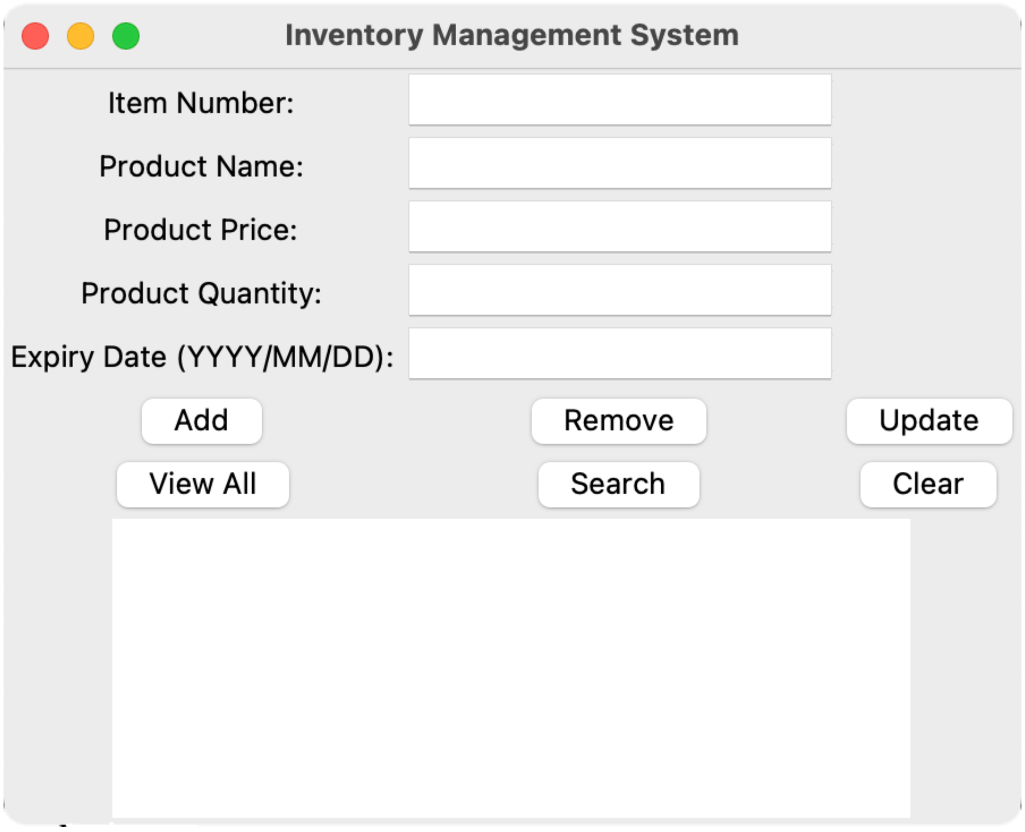
```python
    def analysis_item(self):
        import Analysis

    def download_inventory(self):
        conn = sqlite3.connect('inventory.db')
        cursor = conn.cursor()

        # Create 'items' table if it does not exist
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS items (
                Item_Number INTEGER NOT NULL,
                Name TEXT NOT NULL,
                Quantity INTEGER NOT NULL,
                Price REAL NOT NULL,
                Expiry_Date TEXT
            )
        ''')

        cursor.execute("SELECT * FROM inventory")
        results = cursor.fetchall()

        if len(results) == 0:
            print("No data found in the 'inventory' table.")
            return

        df = pd.DataFrame(results, columns=['Item_Number', 'Name', 'Quantity', 'Price', 'Expiry_Date'])

        if df.empty:
            print("The dataframe is empty.")
            return

        writer = pd.ExcelWriter('inventory.xlsx', engine='xlsxwriter')
        df.to_excel(writer, sheet_name='Inventory', index=False)
        writer.save()
        writer.close()
```

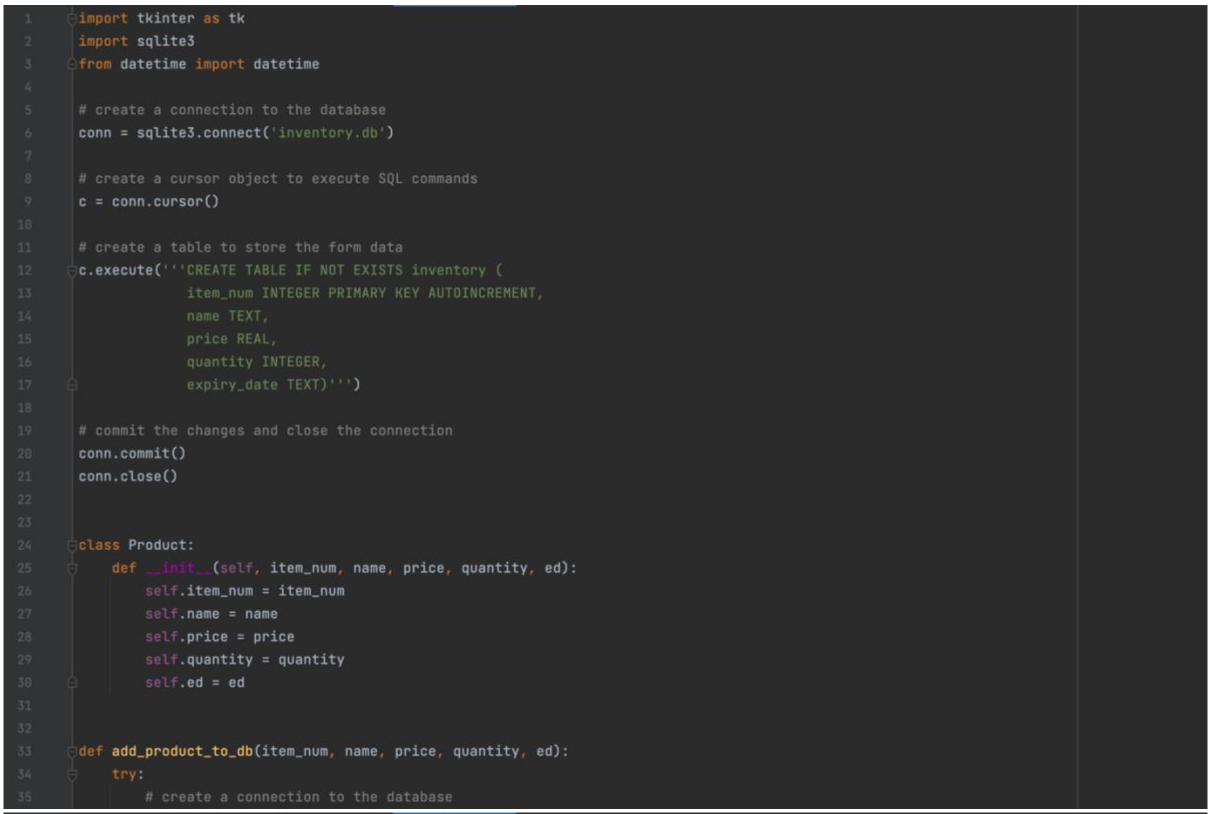

```
            conn.close()

            print(f"{len(results)} rows saved to 'inventory.xlsx'.")
            print("Download complete.")


    root = tk.Tk()
    app = InventoryManagementSystem(master=root)
    app.mainloop()
```

InventoryManagementSystem, a subclass of tk.Frame, is the name of the program's primary class. It entails two functions: create_widgets(), which defines the GUI's labels and buttons, and init(), which arranges and dispatches the GUI window with a title. The create_widgets() method produces a Label widget that conveys a salutation and four Button widgets: one to reveal product information, one to acquire inventory data as an Excel file, one to exhibit analysis of the inventory data, and one to terminate the program. Each button has a precise command linked with it that manages how it operates.

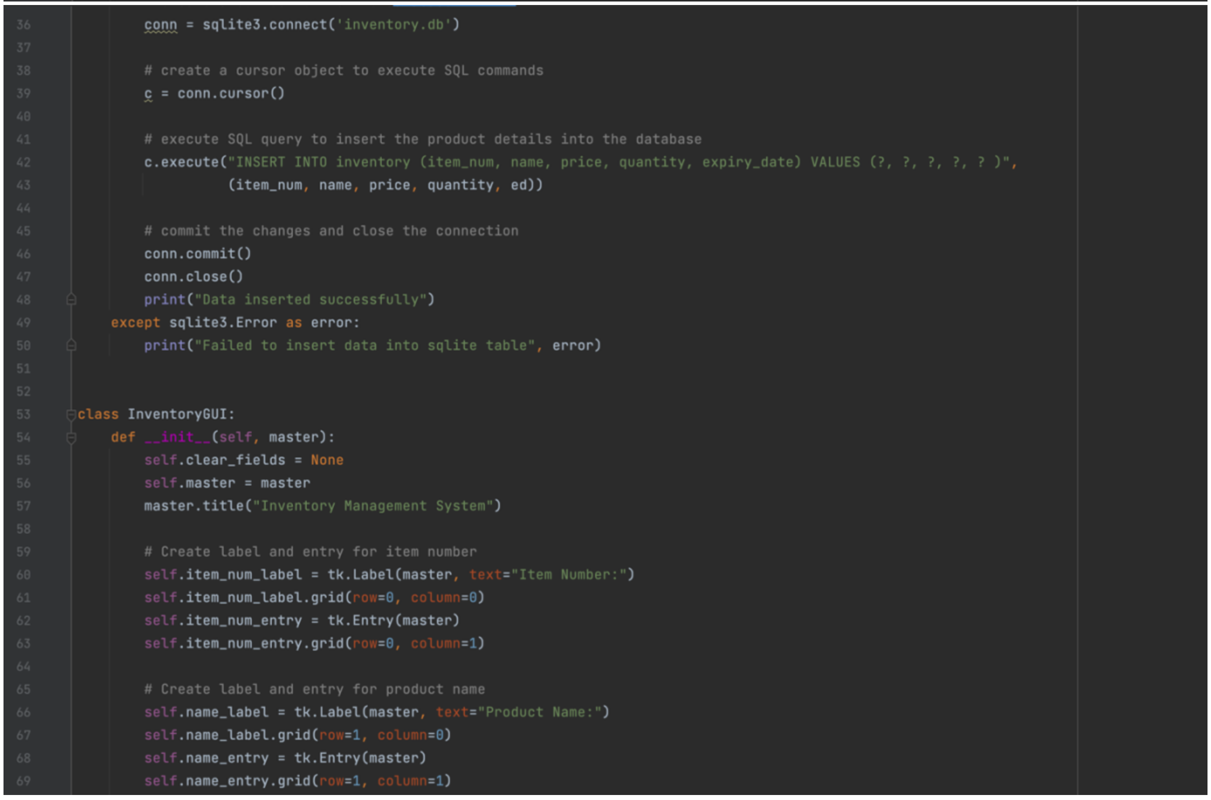
When the "Product" button is clicked, the product_item() method is activated, which imports the MainPage.py module and brings up a new Toplevel window to show the product stock. Clicking the "Analysis" button starts the analysis_item() method, which imports the Analysis module and examines the inventory data. When the "Download Inventory" button is clicked, the download_inventory() method is initiated, which collects the inventory data from a SQLite database, converts it into a Pandas DataFrame, and stores it as an Excel file. The program then gets the InventoryManagementSystem class instance up and running and sets it in motion with the mainloop() method. Once the "Quit" command is engaged, the program comes to a close and the GUI window gets wiped out. To sum it up, this code sets up a simple GUI for an inventory management system that gives the user the capacity to check, analyze, and obtain the inventory data.

MainPage.py

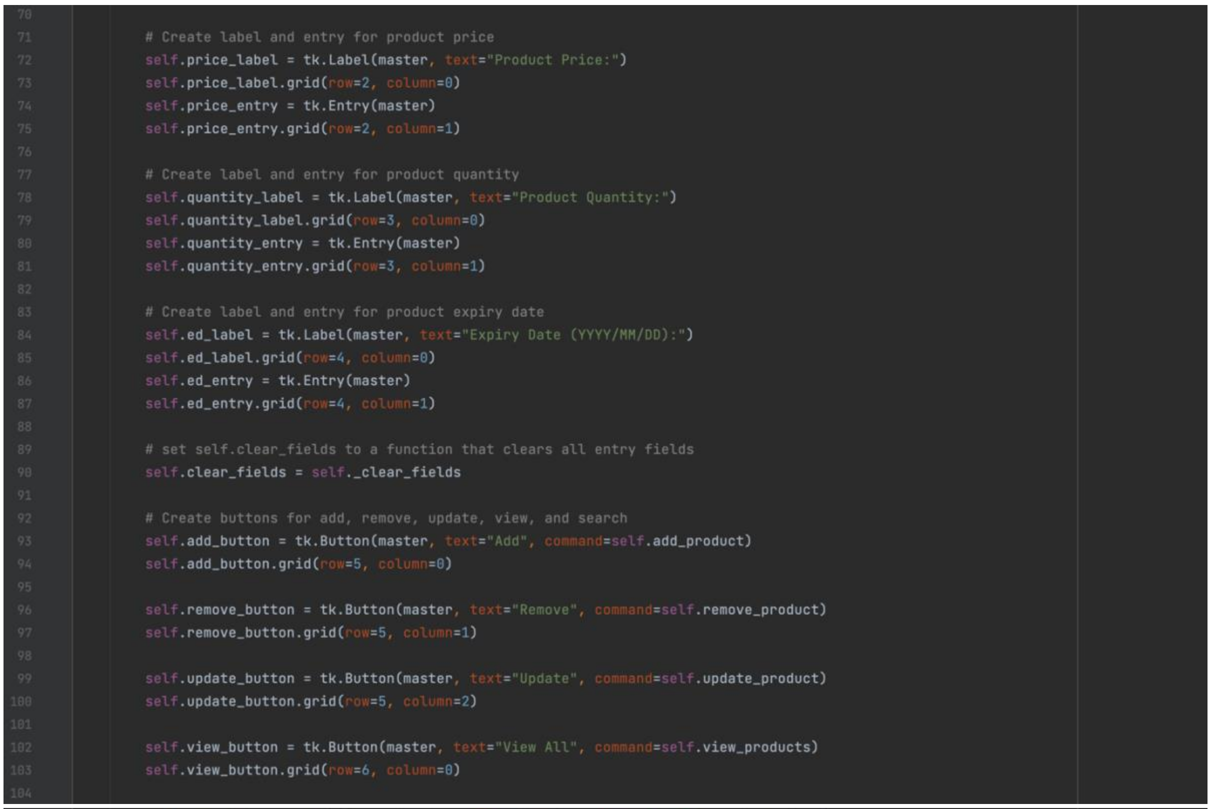




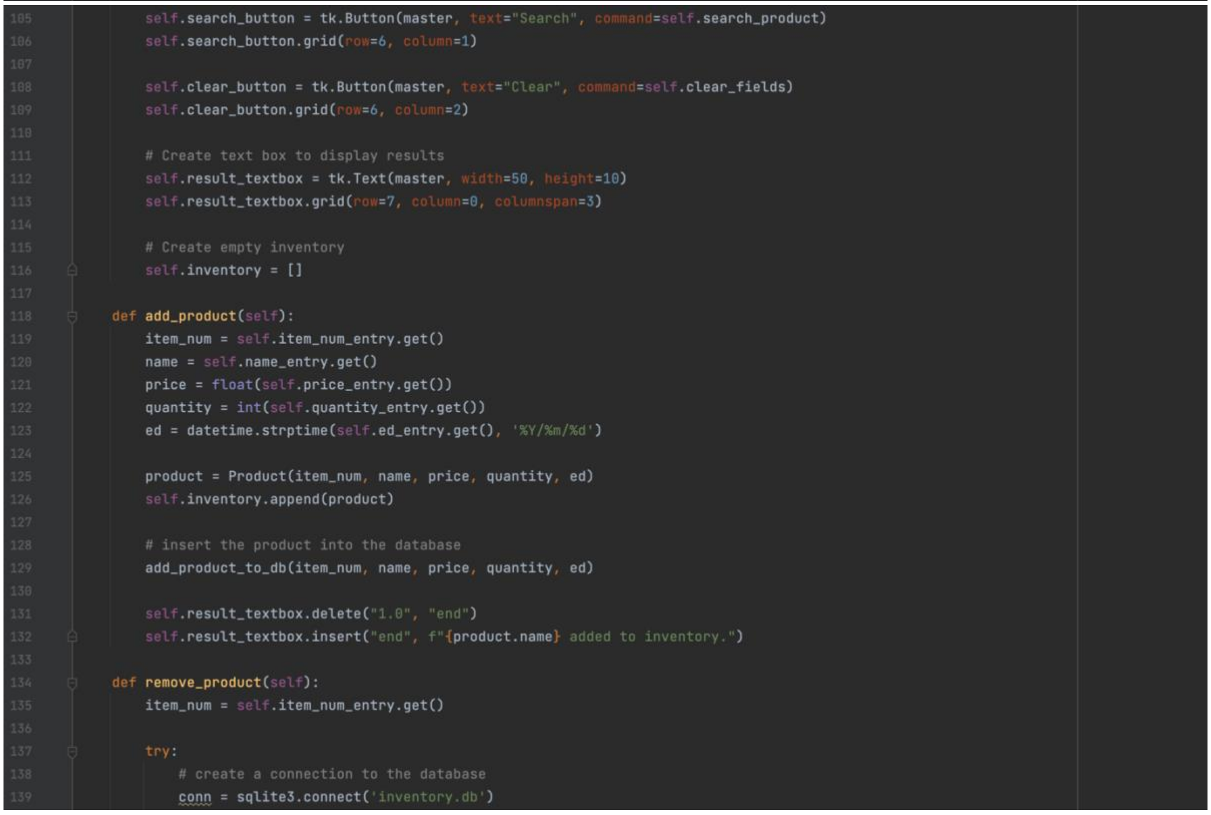
```python
import tkinter as tk
import sqlite3
from datetime import datetime

# create a connection to the database
conn = sqlite3.connect('inventory.db')

# create a cursor object to execute SQL commands
c = conn.cursor()

# create a table to store the form data
c.execute('''CREATE TABLE IF NOT EXISTS inventory (
             item_num INTEGER PRIMARY KEY AUTOINCREMENT,
             name TEXT,
             price REAL,
             quantity INTEGER,
             expiry_date TEXT)''')

# commit the changes and close the connection
conn.commit()
conn.close()


class Product:
    def __init__(self, item_num, name, price, quantity, ed):
        self.item_num = item_num
        self.name = name
        self.price = price
        self.quantity = quantity
        self.ed = ed


def add_product_to_db(item_num, name, price, quantity, ed):
    try:
        # create a connection to the database
```



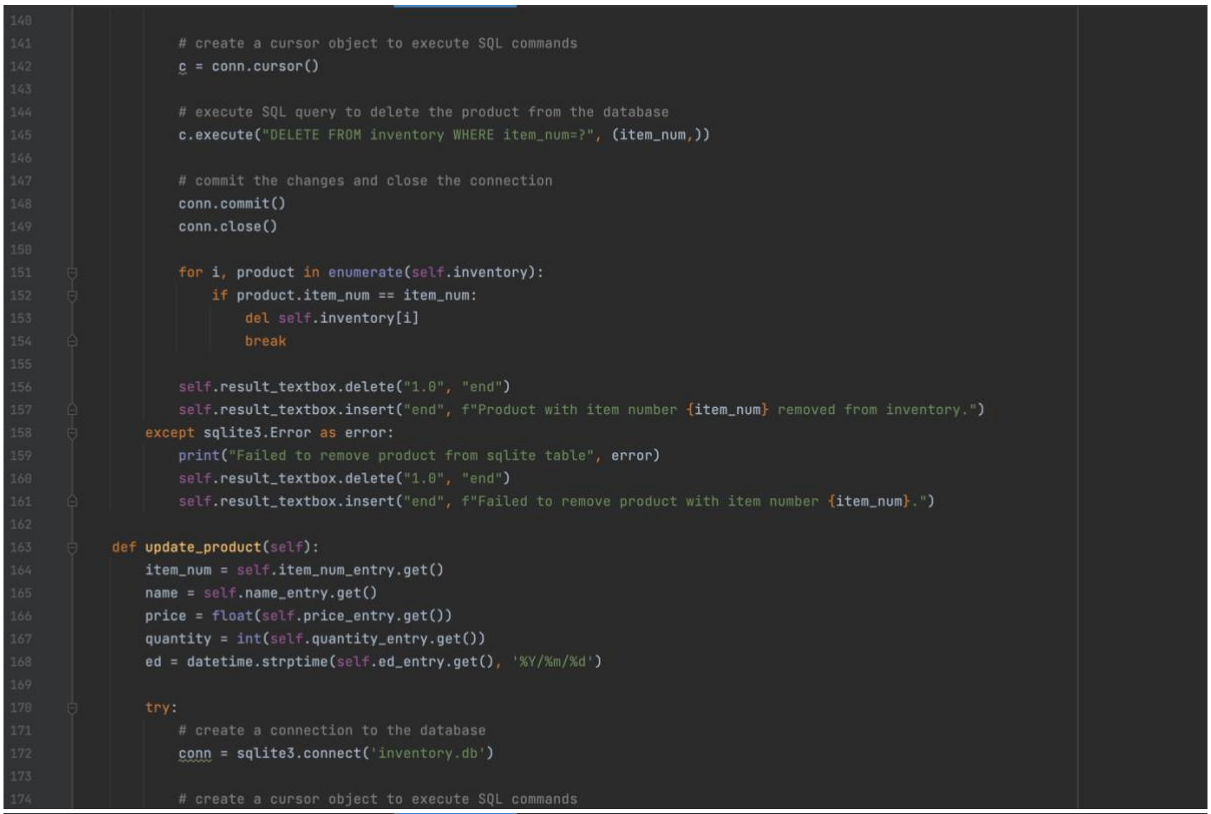
```python
        conn = sqlite3.connect('inventory.db')

        # create a cursor object to execute SQL commands
        c = conn.cursor()

        # execute SQL query to insert the product details into the database
        c.execute("INSERT INTO inventory (item_num, name, price, quantity, expiry_date) VALUES (?, ?, ?, ?, ? )",
                  (item_num, name, price, quantity, ed))

        # commit the changes and close the connection
        conn.commit()
        conn.close()
        print("Data inserted successfully")
    except sqlite3.Error as error:
        print("Failed to insert data into sqlite table", error)


class InventoryGUI:
    def __init__(self, master):
        self.clear_fields = None
        self.master = master
        master.title("Inventory Management System")

        # Create label and entry for item number
        self.item_num_label = tk.Label(master, text="Item Number:")
        self.item_num_label.grid(row=0, column=0)
        self.item_num_entry = tk.Entry(master)
        self.item_num_entry.grid(row=0, column=1)

        # Create label and entry for product name
        self.name_label = tk.Label(master, text="Product Name:")
        self.name_label.grid(row=1, column=0)
        self.name_entry = tk.Entry(master)
        self.name_entry.grid(row=1, column=1)
```



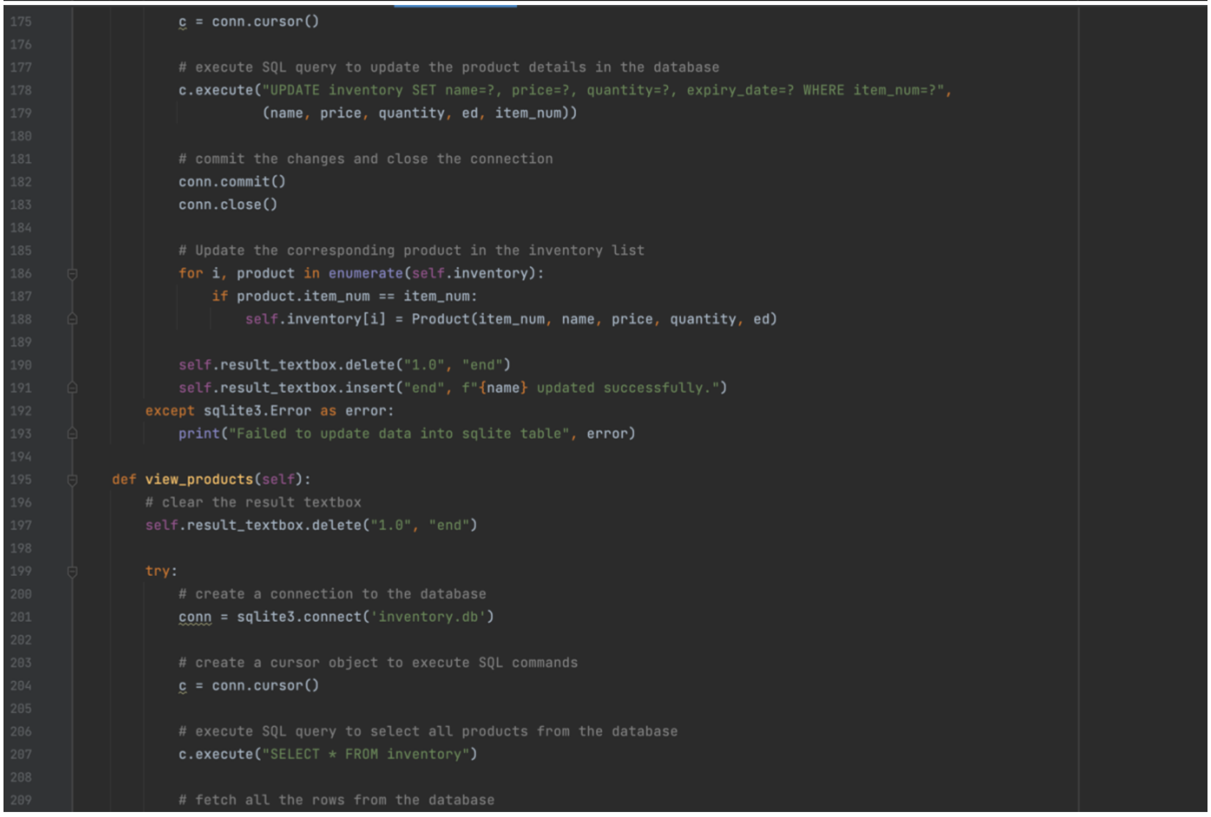
```python

        # Create label and entry for product price
        self.price_label = tk.Label(master, text="Product Price:")
        self.price_label.grid(row=2, column=0)
        self.price_entry = tk.Entry(master)
        self.price_entry.grid(row=2, column=1)

        # Create label and entry for product quantity
        self.quantity_label = tk.Label(master, text="Product Quantity:")
        self.quantity_label.grid(row=3, column=0)
        self.quantity_entry = tk.Entry(master)
        self.quantity_entry.grid(row=3, column=1)

        # Create label and entry for product expiry date
        self.ed_label = tk.Label(master, text="Expiry Date (YYYY/MM/DD):")
        self.ed_label.grid(row=4, column=0)
        self.ed_entry = tk.Entry(master)
        self.ed_entry.grid(row=4, column=1)

        # set self.clear_fields to a function that clears all entry fields
        self.clear_fields = self._clear_fields

        # Create buttons for add, remove, update, view, and search
        self.add_button = tk.Button(master, text="Add", command=self.add_product)
        self.add_button.grid(row=5, column=0)

        self.remove_button = tk.Button(master, text="Remove", command=self.remove_product)
        self.remove_button.grid(row=5, column=1)

        self.update_button = tk.Button(master, text="Update", command=self.update_product)
        self.update_button.grid(row=5, column=2)

        self.view_button = tk.Button(master, text="View All", command=self.view_products)
        self.view_button.grid(row=6, column=0)

```



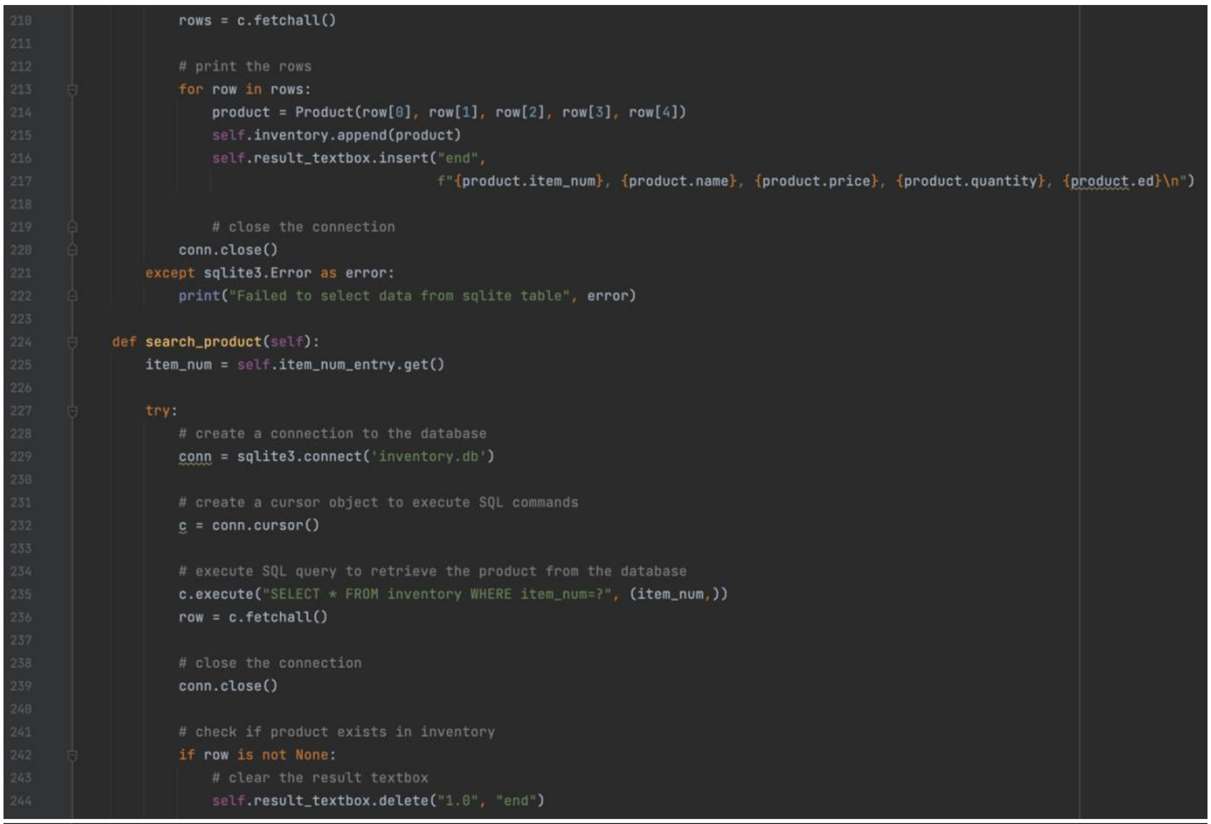
```python
        self.search_button = tk.Button(master, text="Search", command=self.search_product)
        self.search_button.grid(row=6, column=1)

        self.clear_button = tk.Button(master, text="Clear", command=self.clear_fields)
        self.clear_button.grid(row=6, column=2)

        # Create text box to display results
        self.result_textbox = tk.Text(master, width=50, height=10)
        self.result_textbox.grid(row=7, column=0, columnspan=3)

        # Create empty inventory
        self.inventory = []

    def add_product(self):
        item_num = self.item_num_entry.get()
        name = self.name_entry.get()
        price = float(self.price_entry.get())
        quantity = int(self.quantity_entry.get())
        ed = datetime.strptime(self.ed_entry.get(), '%Y/%m/%d')

        product = Product(item_num, name, price, quantity, ed)
        self.inventory.append(product)

        # insert the product into the database
        add_product_to_db(item_num, name, price, quantity, ed)

        self.result_textbox.delete("1.0", "end")
        self.result_textbox.insert("end", f"{product.name} added to inventory.")

    def remove_product(self):
        item_num = self.item_num_entry.get()

        try:
            # create a connection to the database
            conn = sqlite3.connect('inventory.db')
```



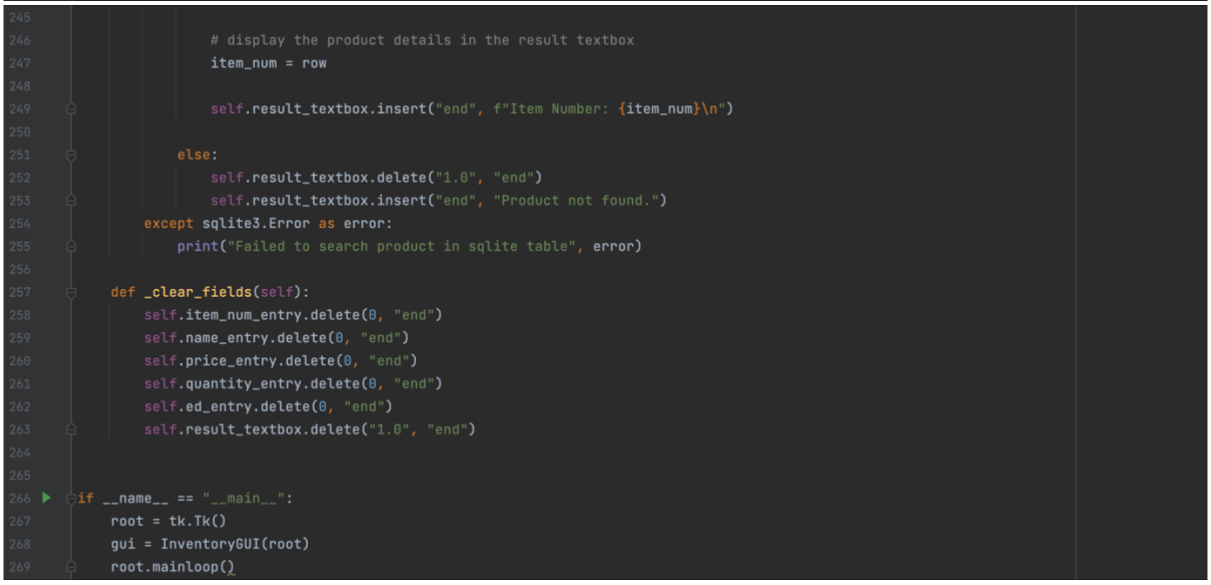
```python

            # create a cursor object to execute SQL commands
            c = conn.cursor()

            # execute SQL query to delete the product from the database
            c.execute("DELETE FROM inventory WHERE item_num=?", (item_num,))

            # commit the changes and close the connection
            conn.commit()
            conn.close()

            for i, product in enumerate(self.inventory):
                if product.item_num == item_num:
                    del self.inventory[i]
                    break

            self.result_textbox.delete("1.0", "end")
            self.result_textbox.insert("end", f"Product with item number {item_num} removed from inventory.")
        except sqlite3.Error as error:
            print("Failed to remove product from sqlite table", error)
            self.result_textbox.delete("1.0", "end")
            self.result_textbox.insert("end", f"Failed to remove product with item number {item_num}.")

    def update_product(self):
        item_num = self.item_num_entry.get()
        name = self.name_entry.get()
        price = float(self.price_entry.get())
        quantity = int(self.quantity_entry.get())
        ed = datetime.strptime(self.ed_entry.get(), '%Y/%m/%d')

        try:
            # create a connection to the database
            conn = sqlite3.connect('inventory.db')

            # create a cursor object to execute SQL commands
```



```python
            c = conn.cursor()

            # execute SQL query to update the product details in the database
            c.execute("UPDATE inventory SET name=?, price=?, quantity=?, expiry_date=? WHERE item_num=?",
                      (name, price, quantity, ed, item_num))

            # commit the changes and close the connection
            conn.commit()
            conn.close()

            # Update the corresponding product in the inventory list
            for i, product in enumerate(self.inventory):
                if product.item_num == item_num:
                    self.inventory[i] = Product(item_num, name, price, quantity, ed)

            self.result_textbox.delete("1.0", "end")
            self.result_textbox.insert("end", f"{name} updated successfully.")
        except sqlite3.Error as error:
            print("Failed to update data into sqlite table", error)

    def view_products(self):
        # clear the result textbox
        self.result_textbox.delete("1.0", "end")

        try:
            # create a connection to the database
            conn = sqlite3.connect('inventory.db')

            # create a cursor object to execute SQL commands
            c = conn.cursor()

            # execute SQL query to select all products from the database
            c.execute("SELECT * FROM inventory")

            # fetch all the rows from the database
```



```python
            rows = c.fetchall()

            # print the rows
            for row in rows:
                product = Product(row[0], row[1], row[2], row[3], row[4])
                self.inventory.append(product)
                self.result_textbox.insert("end",
                                           f"{product.item_num}, {product.name}, {product.price}, {product.quantity}, {product.ed}\n")

                # close the connection
            conn.close()
        except sqlite3.Error as error:
            print("Failed to select data from sqlite table", error)

    def search_product(self):
        item_num = self.item_num_entry.get()

        try:
            # create a connection to the database
            conn = sqlite3.connect('inventory.db')

            # create a cursor object to execute SQL commands
            c = conn.cursor()

            # execute SQL query to retrieve the product from the database
            c.execute("SELECT * FROM inventory WHERE item_num=?", (item_num,))
            row = c.fetchall()

            # close the connection
            conn.close()

            # check if product exists in inventory
            if row is not None:
                # clear the result textbox
                self.result_textbox.delete("1.0", "end")
```



```python

                # display the product details in the result textbox
                item_num = row

                self.result_textbox.insert("end", f"Item Number: {item_num}\n")

            else:
                self.result_textbox.delete("1.0", "end")
                self.result_textbox.insert("end", "Product not found.")
        except sqlite3.Error as error:
            print("Failed to search product in sqlite table", error)

    def _clear_fields(self):
        self.item_num_entry.delete(0, "end")
        self.name_entry.delete(0, "end")
        self.price_entry.delete(0, "end")
        self.quantity_entry.delete(0, "end")
        self.ed_entry.delete(0, "end")
        self.result_textbox.delete("1.0", "end")


if __name__ == "__main__":
    root = tk.Tk()
    gui = InventoryGUI(root)
    root.mainloop()
```

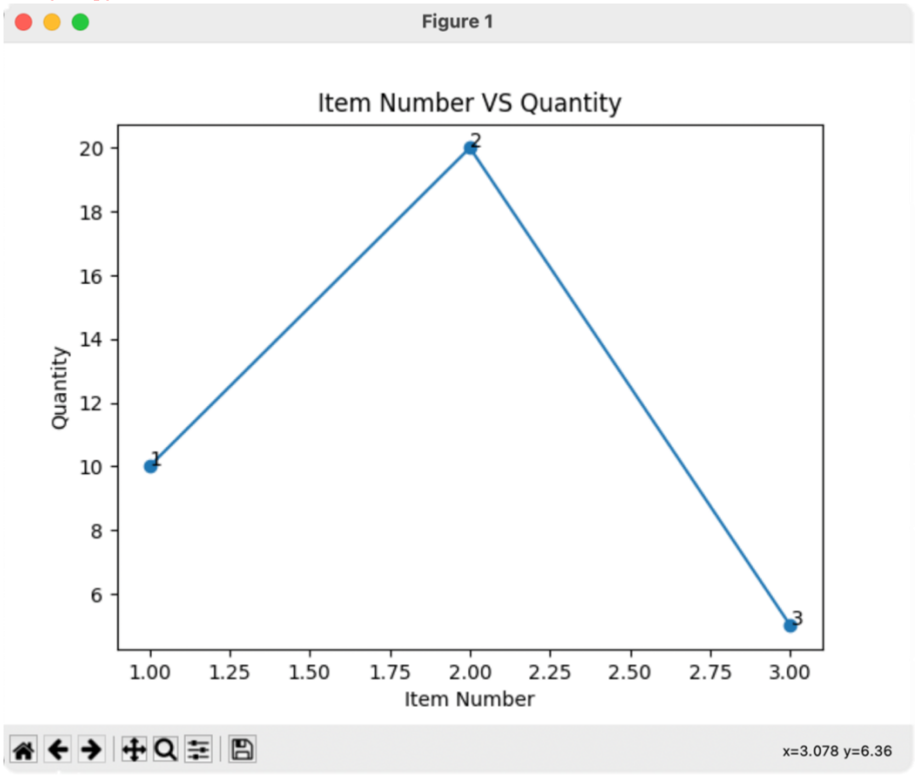
By employing this code, a person can conveniently add, remove, update, view, and search for products in a comprehensible supply management program. Bringing in the essential libraries, Tkinter for the graphical user interface, SQLite for the database, and datetime for working with dates, is the initial step in the code. The database link and the cursor object required to run SQL commands are then established. A table is generated that holds the stock sheet data, with fields for the item number, name, rate, amount, and expiry date. The Product class is then introduced, which is used to symbolize each item in the inventory. The add_product_to_db function is established to enable the addition of a new item to the database.

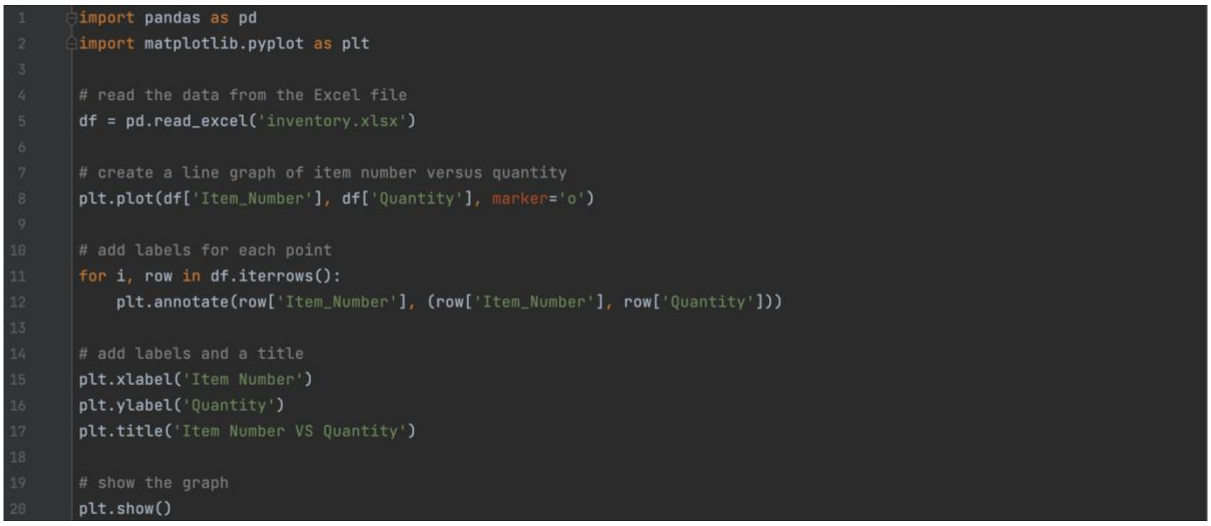
The GUI is constructed with the help of the InventoryGUI class. The window is launched by the init function, which also adds buttons and text fields for the user to interact with. Every button is linked to a particular action, such as adding or deleting an item. Additionally, the GUI has a text box where the results of each action are displayed.

Once the “Add” button is pushed by the user, the add_product function is enacted. The add_product_to_db function is called to add the newly formed Product object to the database after collecting the user-submitted data from the entry fields, incorporating it to the stock list, and forming a fresh Product object. The user clicks the “Remove” button, initiating the remove_product function. The user-submitted item number is obtained and then the corresponding product is taken away from the inventory list and database.

By substituting the figures in the repository, the update_product function facilitates patrons to adjust a product's details. When a person clicks the “View All” button, the view_products action is set into motion. The text box shows every single one of the items that were taken from the database. The “Search” button is pushed by the individual, which activates the search_product function. The search term that was entered by the user is collected, and any correlating items are displayed in the text box. When the person presses the “Clear” button, the code's clear_fields function is triggered. All of the input boxes in the GUI are wiped out. Essentially, this code supplies a user-friendly GUI and a fundamental inventory control system. It enables the user to execute standard inventory control assignments, such as introducing and extracting items, and stores the inventory data in a database.

Analysis.py





```
import pandas as pd
import matplotlib.pyplot as plt

# read the data from the Excel file
df = pd.read_excel('inventory.xlsx')

# create a line graph of item number versus quantity
plt.plot(df['Item_Number'], df['Quantity'], marker='o')

# add labels for each point
for i, row in df.iterrows():
    plt.annotate(row['Item_Number'], (row['Item_Number'], row['Quantity']))

# add labels and a title
plt.xlabel('Item Number')
plt.ylabel('Quantity')
plt.title('Item Number VS Quantity')

# show the graph
plt.show()
```

Using the "import" statement, this code incorporates the Pandas and Matplotlib Python libraries. The "pd.read_excel" function of Pandas is then utilized to read the data from the "inventory.xlsx" Excel file and store it in the "df" variable. Afterward, a line graph is formed with item number versus quantity. The "plt.plot" function in Matplotlib is then utilized to generate a line chart where the horizontal axis is the magnitude, and the vertical axis is the item numbers. Each point will have round markers if the "marker" argument is set to "o".

The code goes through each row of the "df" variable using the "iterrows" function, incorporating markers for each spot as it does so. The "plt.annotate" function is utilized to add tags at each spot, with the first argument representing the label content (in this case, the item number), and the second argument representing the point's location. After labelling the points, the code adds a label for the x-axis, a label for the y-axis, and a title to the graph using the "plt.xlabel", "plt.ylabel", and "plt.title" functions, respectively. Finally, the graph is displayed using the "plt.show" function.

This code has been designed to present the inventory amounts graphically, enabling trends to be observed and decisions to be taken related to stock control. To avert depletion of items or overstocking, it is important to observe the levels of inventory in an inventory management system. The user can distinguish which items must be replenished and which may be oversupplied expeditiously due to the line graph this code generates which demonstrates the stock levels. This file is a beneficial asset for inventory directors to take data-driven decisions regarding stock management. As a result of it, they can hastily identify which products might require restocking and keep tabs on variations in inventory levels over time.

4. Future work (Suggestion for improvement)

The features listed below could be included in a software program to improve user experience and augment productivity. To begin, by analysing a month's best sales, organizations can acquire insight about which products are popular and make judicious decisions. By doing so, firms can then concentrate on advertising and stocking those items more intensively after discovering which are most desired by patrons. This capability can help organizations in formulating sound decisions regarding cost, supply organization, and promotional activities. Another way that firms can avert stockouts and squandering is by sending out signal notifications when a product's expiry date is imminent or when there is a low inventory. By allowing them to swiftly restock their supplies or remove out-of-date items from the shelves, this can aid companies in preventing wastefulness and running out of stock. Thirdly, having a "Forgot Password" button on the login page can save time and hassle for users who overlook their password by enabling them to expeditiously recover their account. Those who ordinarily need to undergo an extended password retrieval system can gain from this as it can conserve them time and exertion. Fourthly, having a back button can afford users the independence to transition between pages without having to start over.

Lastly, providing a readily available aid button allows users to obtain assistance whenever they experience problems. Without having to turn to external sources for answers or contact customer support, this can help users fix issues promptly and efficiently with better user interface. Sixth, having an index on the chart that reveals when the stock must be replenished can assist personnel to stay on top of inventory management. By alerting them when the amount of a product drops below the required amount, this can help personnel sustain stock control. Finally, displaying data in a tabular format on the MainPage.py can ease users' viewing and comprehension of data. These characteristics enable applications to be more user-friendly, competent, and productive.

5. Conclusion

The in-store store has encountered numerous issues due to the lack of a proficient stock system, including oversupply, lost income opportunities, and augmented costs. As a supervisor, they are aware of the necessity for a stock arrangement that can render real-time stock updates, link up with their point-of-sale mechanism, and give precise information for restocking, pricing, and marketing decisions. They also need a system that is easy to use and flexible enough to adjust to the changing needs of their growing business. It would also be advantageous if the system could generate reports and analyze stock data pictorially. The utilization of an inventory management system can assist the in-store store surpass its issues, resulting in better stock control, sounder judgement-making, and ultimately bigger earnings.

The merits and drawbacks of instituting such a system ought to be deliberated on with due diligence, and the personnel entrusted with keeping the system current ought to be devoted to ensuring its success. Through the adoption of an automated inventory management system, a business can benefit from immediate stock data, precise figures, the capacity to fabricate reports, and the capacity to interpret inventory data in graphs. This can provide valuable insight into inventory levels and sales trends, which can refine supply chain management and client pleasure, ultimately setting the business up for long-term prosperity.

Including a convenient interface that made it effortless to input and access inventory data was one of the task's greatest accomplishments. People can readily traverse through the diverse capacities of the stock administration framework due to the GUI's unpretentious and natural plan. Moreover, the GUI was devised to give a definite visual portrayal of the stock levels, empowering clients to screen and administer their inventory levels with no problem. The code's capacity to incorporate, eliminate, update, see, and search for items just as download the stock data into an excel document was another creative component. Clients had the option to proficiently oversee their stock and keep an eye on their items continuously because of this usefulness. Additionally, scrutinizing the title and amount of the item presented users with a straightforward way to evaluate their supplies and make sound decisions concerning acquisition, fabrication, and marketing tactics using a graph.

Analysing the code and ensuring that all the features operated as expected was one of the issues we faced in the project. We overcame this by consulting different coding manuals, rigorously checking the code, and spotting any faults or mistakes. Then, using a variety of debugging strategies we were able to make the code run as planned. Another challenge was improving the interface's user-friendliness and visual appeal. We solved this by making thoughtful plans and changes, while also incorporating feedback from customers and colleagues to improve the look of the interface.

Upcoming changes to the stock control system could include the integration of fresh elements and capabilities. For instance, the introduction of an alert mechanism that warns customers when stocks fall beneath a predetermined level would aid in avoiding shortages and help reduce waste. Additionally, incorporating a predictive analytics application that forecasts inventory demand based on past figures would assist companies in managing their inventory levels more efficiently and economically. In sum, the project provided invaluable knowledge about utilizing Python to construct an effective and productive inventory management system.

6. References

[1] Hashmicro (2022). *Inventory Management Software: A Need for the Retail Industry*. [online] BusinessTech. Available at: https://www.hashmicro.com/blog/inventory-management-software-need-retail-industry/ [Accessed 22 Feb. 2023].

[2] courses.lumenlearning.com. (n.d.). *Inventory Levels | Retail Management*. [online] Available at: https://courses.lumenlearning.com/wm-retailmanagement/chapter/determining-product-inventory-levels/ [Accessed 22 Feb. 2023].

[3] Ginsberg, B. (2022). *How To Successfully Manage Inventory In Retail Stores*. [online] ApparelMagic. Available at: https://apparelmagic.com/how-to-successfully-manage-inventory-in-retail-stores/ [Accessed 22 Feb. 2023].

[4] Unleashed Software. (2019). *What Is An Inventory Management System? Your Complete Guide*. [online] Available at: https://www.unleashedsoftware.com/inventory-management-guide/inventory-management-systems [Accessed 22 Feb. 2023].

[5] BigCommerce (2022). *Inventory Management: A Guide to Success + Techniques*. [online] BigCommerce. Available at: https://www.bigcommerce.com/articles/ecommerce/inventory-management/ [Accessed 22 Feb. 2023].

[6] WiSys (2021). *Why Real-Time Inventory Management Matters To Your Business*. [online] Supply Chain Management Solutions for Macola and SAP Business One. Available at: https://www.wisys.com/why-real-time-inventory-management-matters-to-your-business/ [Accessed 22 Feb. 2023].

[7] Jenkins, A. (2020). *What is Inventory Management? Benefits, Types, & Techniques*. [online] Oracle NetSuite. Available at: https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management.shtml [Accessed 23 Feb. 2023].

[8] Inventory, C.F. (2023). *How to Avoid Overstocking and Understocking? Effective and Efficient Ways*. [online] CFBLOG. Available at: https://cashflowinventory.com/blog/overstocking-and-understocking/ [Accessed 23 Feb. 2023].

[9] Abu Zwaida, T., Pham, C. and Beauregard, Y. (2021). Optimization of Inventory Management to Prevent Drug Shortages in the Hospital Supply Chain. *Applied Sciences*, [online] 11(6), p.2726. doi:https://doi.org/10.3390/app11062726.

[10] Velthoen, J. (2015). *Preventing Theft with Inventory Management*. [online] QStock Inventory. Available at: https://www.qstockinventory.com/blog/preventing-theft-with-inventory-management/ [Accessed 23 Feb. 2023].

[11] AltexSoft. (2021). *Inventory Management Software: How Technology Addresses Inventory-Related Issues*. [online] Available at: https://www.altexsoft.com/blog/inventory-management-software/ [Accessed 23 Feb. 2023].

[12] Asset Infinity Blog. (2019). *What are the Objectives of the Inventory Management System? - Asset Infinity*. [online] Available at: https://www.assetinfinity.com/blog/inventory-management-system-objectives [Accessed 23 Feb. 2023].

[13] Publisher, A. removed at request of original (2015). *11.7 Data Asset in Action: Technology and the Rise of Wal-Mart*. [online] open.lib.umn.edu. Available at: https://open.lib.umn.edu/informationsystems/chapter/11-7-data-asset-in-action-technology-and-the-rise-of-wal-mart/ [Accessed 23 Feb. 2023].

[14] Amazon (2022). *Inventory Management 101: Essentials for Ecommerce Businesses*. [online] sell.amazon.com. Available at: https://sell.amazon.com/learn/inventory-management [Accessed 23 Feb. 2023].

[15] howard, sofya (2021). *RFID's Role in Managing Inventory for Medical Devices and Critical Supplies*. [online] RAIN RFID. Available at: https://rainrfid.org/rfids-role-in-managing-inventory-for-medical-devices-and-critical-supplies/ [Accessed 23 Feb. 2023].

[16] Keenan, M. (2022). *What Is Inventory Management? How to Manage and Improve Stock Flow*. [online] Shopify. Available at: https://www.shopify.com/retail/inventory-management [Accessed 23 Feb. 2023].

[17] Atnafu, D. and Balda, A. (2018). The impact of inventory management practice on firms' competitiveness and organizational performance: Empirical evidence from micro and small enterprises in Ethiopia. *Cogent Business & Management*, [online] 5(1), pp.1–16. doi:https://doi.org/10.1080/23311975.2018.1503219.

[18] Rickerby, M. (2022). Inventory Management Guide: Techniques & 4 Good Examples. [online] 17 Mar. Available at: https://www.extensiv.com/blog/inventory-management [Accessed 3 Mar. 2023].

7. Appendix

Python Tutorial/Help Websites Used

https://www.w3schools.com/python/matplotlib_line.asp
https://www.w3schools.com/python/matplotlib_labels.asp
https://www.w3schools.com/python/matplotlib_markers.asp
https://www.codespeedy.com/plot-data-from-excel-file-in-matplotlib-python/
https://youtu.be/pd-0G0MigUA
https://youtu.be/byHcYRpMgI4
https://www.sqlitetutorial.net/
https://www.codespeedy.com/export-data-from-mysql-to-the-excel-sheet-through-python/
https://youtu.be/hSS3raFtEEo

Demonstration Video

Inventory Management System (1).mp4