Table of content

|  |  |  |
| --- | --- | --- |
| Section | Content | Page Number |
| 1 | Introduction | 2 |
| 2 | SQL and GUI | 3 |
| 3 | Detail explanation of each function | 4 |
| 4 | Screenshot of coding and output | 9 |
| 5 | Future work (Suggestion for improvement) | 15 |
| 6 | Conclusion | 15 |

1. **Introduction**

The purpose of this laboratory is to learn from the process of writing a complete program that resembles some basic principles of an inventory system. The language used is Python 3.0 and the method of storing and writing the data to a database is either through excel (csv) or MySQL workbench (SQL). In this case, the SQL was used, and python code was written to assist in the writing of the data to be present in the database. The program’s base functions include, allowing the user to:

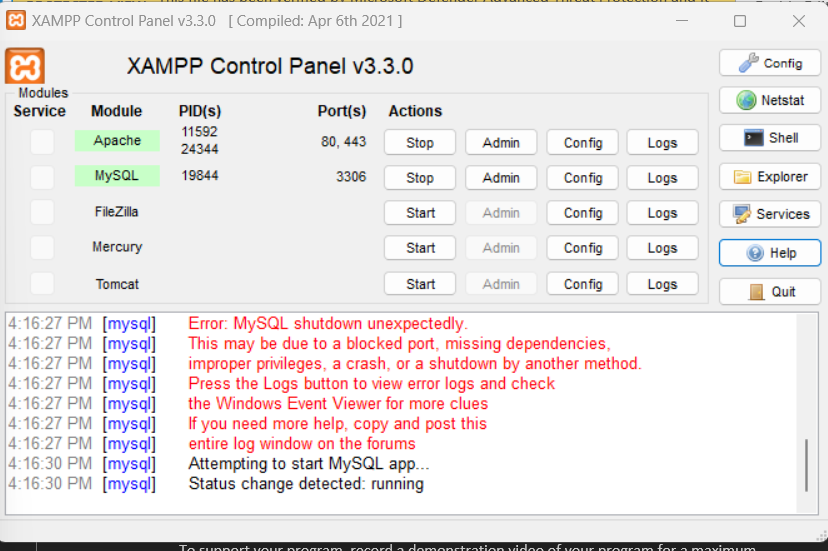
* Add a new item record which will include:
  + A name
  + A price
  + A Quantity to represent stock
  + An expiry date
* Delete an item record
* View all item records present in the database
* Update a specific cell for a record entry
* Search for specific item records present.

Each of these functions will be discussed in depth throughout the rest of this article. Along with future ideas which will hopefully be implemented later. As per the guidelines in the instruction document: a selection statement (use of if), a repetitive statement (multiple while loops and a for loop) and functions (multiple functions) have been included in the construction of the complete program. Screenshots of the code will be included in this document. A ten minute demonstration video and the source code files will be included as separate attachments to the submission along with a logbook submission.

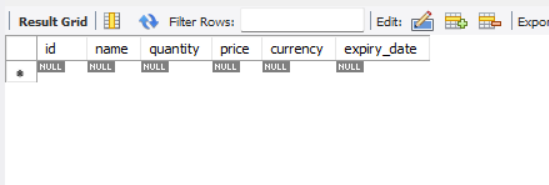
1. **SQL and GUI**

There are some important things to be made aware of before testing the program and evaluating its integrity. Firstly, a simple GUI was created with a button for each corresponding function. Due to GUI’s not being a requirement to the task, a minimalistic approach was taken with the GUI being its only purpose to call the function and input to the terminal in a single execution (for convenience’s sake).

The second matter that is necessary to pay attention to is the nature of the SQL database created and used in the project. The SQL software used to create and manipulate the database was MySQL Workbench which needed local server to run the database on so it could communicate with the program. Here, the use of the third-party open-source software, XAMPP was useful. XAMPP provided an Apache web server for the MariaDB (MySQL) software and is necessary for the proper functionality of the program. Below is a screenshot of the control panel of the program:



After establishing a successful MySQL server connection, the database, inventory was created. This is done by running the command in python by using the cursor class which is imported from the mysql.connector module downloaded earlier. The class presents a method of interaction between the interpreter and MySQL using SQL statements and executing them as if the user manually inputted it into the workbench software themselves. Using the execute function, a database is created, named, referenced using “USE” and a table is created as per the guidelines. If opened in MySQL, the table below will appear:



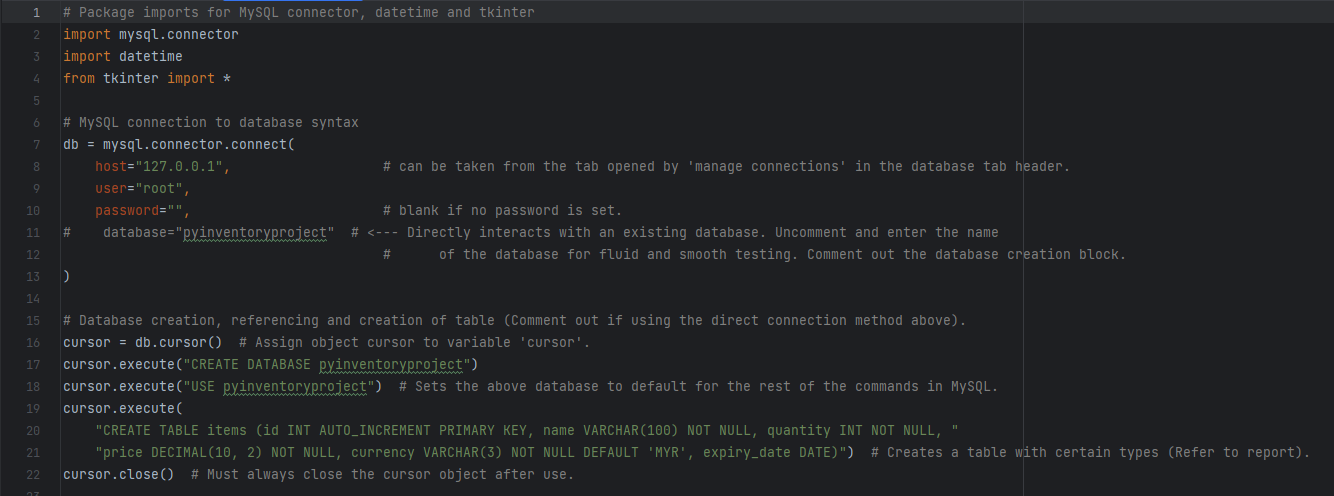
This is the placeholders for the data that will be written by the program into the table. Each field has its data type defined and some fields have a NOT NULL constraint attached to prevent empty cells being present. The id field is assigned as the primary key and a currency field was added as an attempt to improve the immersiveness of the program by using a familiar currency.

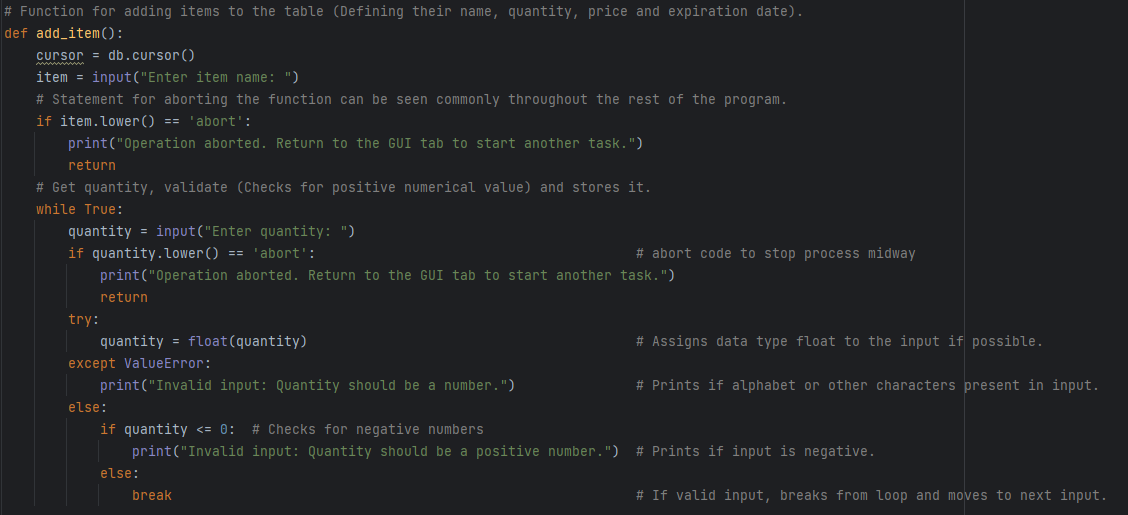
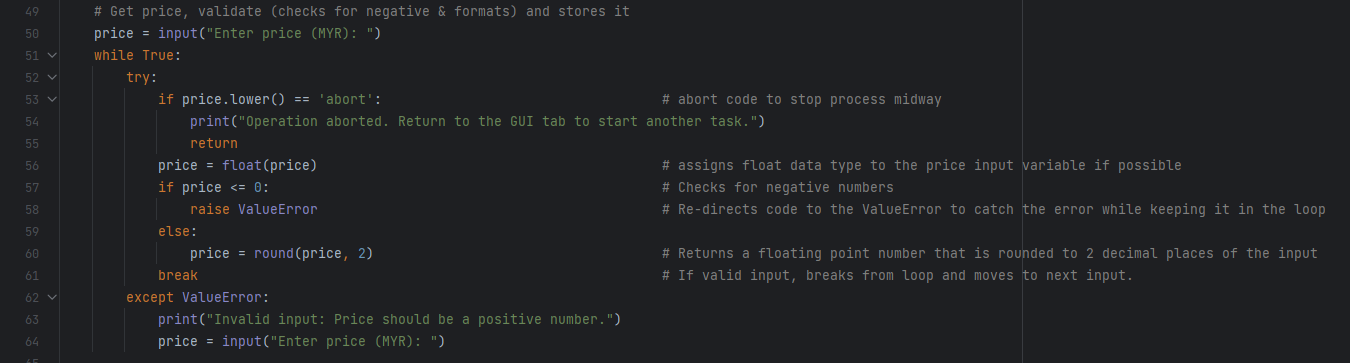
The SQL table is then connected to the python IDE and program with the assistance of the MySQL connector driver, which was installed through pip. The syntax code for connecting the database is inserted into the program. The code contains placeholders for the ‘host’, ‘user’, ‘password’ (blank if not assigned). These are replaced with the currently used values which are displayed in the screenshot in section 4. This establishes a successful connection to the database and the construction of the functions can begin.

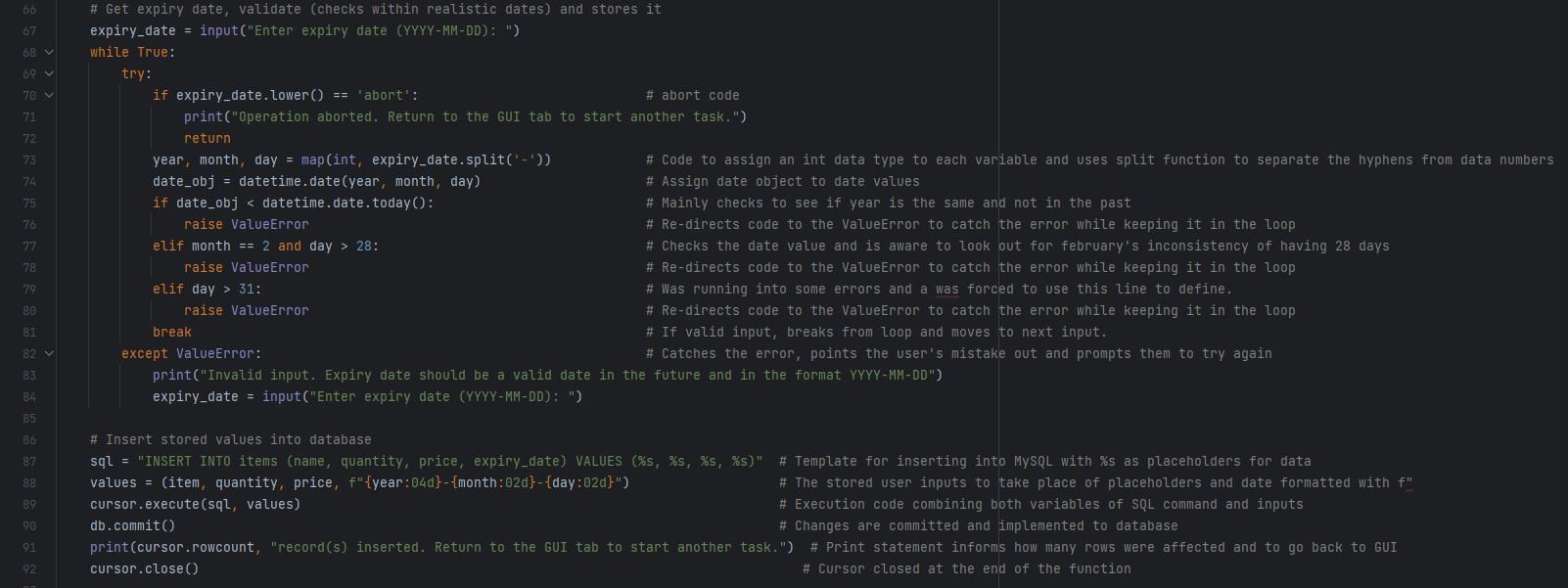
1. **Detailed explanation of each function**

Almost every function in this program has the opening line which is the assignment of cursor object to the variable ‘cursor’. Most functions have the code start with a basic input prompt from the user, which will act as the condition and foundation for the rest of the functionality for this function. It is followed by another selection statement (if) that will be found in the other functions as well. This if statement checks if the user inputted the specific words ‘abort’, the purpose of which is to give the user the option to abort an operation midway if they please. This is credited to the return statement which is accompanied by an explanation print statement.

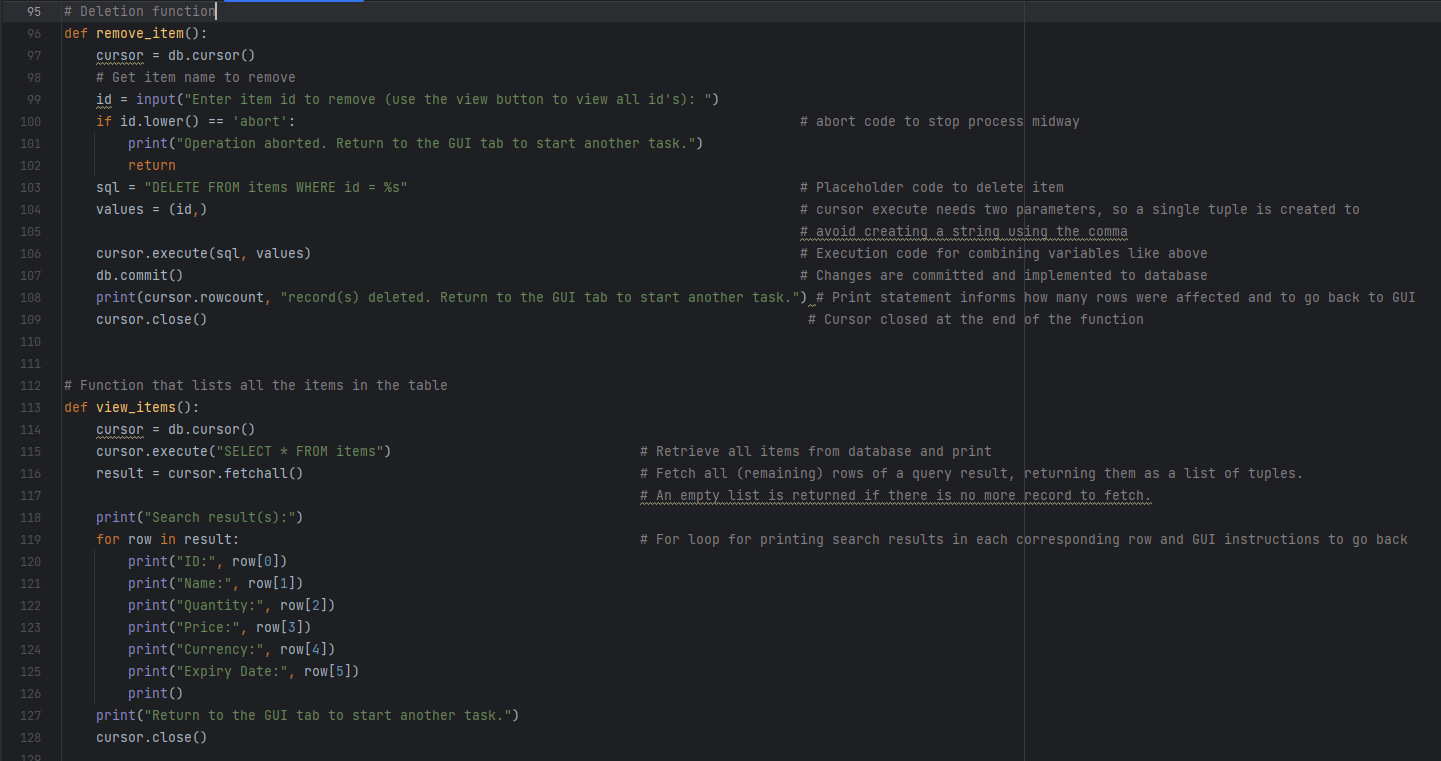
1. add\_item:
   1. The general purpose of this function is to allow the user to insert items into the sql database. It will prompt the user for 4 inputs: item name, quantity of item, price of item and the expiry date of the item. Each of the input stages are equipped with the abort code to allow a return feature at any time.
   2. Firstly, the item code is quite simple. Due to the nature of the program being an inventory system for a superstore, there are no restrictions on the item name. It can be whatever the user wants, thus, it takes the form of a simple input prompt.
   3. Next, the quantity prompt. The quantity prompt is put in a while loop to make sure that the user will not move to the next stage until a valid input is given without breaking the flow of the input. The nature of quantity in a superstore is understood to be strictly numerical but can have leniency as to whether it needs to be in whole number or allow decimal places. Therefore, this code has a data validation section that checks to see if the input is a positive number. It restricts the input type with the assistance of a try except block. It does this by first assuring the input ends up as a float value after the try block, it then creates the except block to catch the error. The except block with the ValueError entity which results in a print statement that points out the user’s mistake in their input. An else block follows with an if statement that ensures the input is a positive number by replying to all inputs that are smaller or equal to zero with a print error statement similar to the one above. The user will be unable to break away from the while loop unless they input a valid quantity.
   4. The price input also uses a while loop, try and except blocks and the abort feature as well. In the case of this field, the data validation is in the form of making sure the price is a positive number and that the format of the number is rounded to two decimal places. It does this by using a method like the one above where price is assigned a float value after a try block and if price is less than or equal to zero, a ValueError is raised. The raise feature redirects the code to the except block which is after the break statement. This placement is necessary due to the positioning of the if-else statement (which can be observed in the later section), which helps keep the data validation in a loop until a valid input is inserted by the user. ValueError triggers a similar print statement that points out the user’s mistake and prompts them to try again.
   5. Price is then followed by the expiry-date field. The code starts with the standard input and the abort code. Like the predecessors, the code makes use of the while loop to ensure the user enters a valid input and a try/except blocks for data validation. The expiry-date data validation is a bit different due to how the date field type operates. Firstly, since the date inputted will be in the format of YYYY-MM-DD (as instructed by the input prompt statement), the program will have to separate the inputted numbers from the hyphens. This is done using the .split function with the hyphen plugged in between the parentheses. An int data type is then assigned to each value for year, month, and day using the map function from the datetime module. Using the features of the datetime module, an object is created that is able read today’s date and compare it to the assigned values to make sure the expiry\_date is in the future. There is also an additional statement that restricts the input of the specific dates in February. This accounts for February’s special circumstance of having only 28 days. There were some issues where the program was accepting non-existent days like day 40, thus, the last line before the break was necessary to fix that. Overall, the code is constructed to improve realism and ensure that each value is a positive whole number within the range of realistic date numbers, if not, a ValueError is raised with similar functionality to the previous value errors. That being a print statement pointing out the mistake followed by a prompt to try again.
   6. Lastly, there is the sql execution block. A variable is assigned to the sql command that would be sent from python to MySQL to trigger the action. Placeholders in the form of %s are seen in this line. Another variable is created to hold the inputted values from the user. The execute function for the mysql.connector module is used to combine the two variable to send a proper fully formed command to the SQL interpreter, this is finalised with the commit command. A print statement is used to confirm a successful insertion of the record and instructions to return to the GUI tab. Like most SQL functions in this program, the cursor is closed last.
2. remove\_item:
   1. Similar to the insert function and other functions, the code will contain the assignment of the cursor, an abort statement, a basic input and a similar set of SQL commands. However, there is a bit of difference in the SQL commands here. The purpose of the function is to delete one record at a time but due to the nature of the sql syntax commands, there is a conflict when executing the combined variables. There still needs to be a second variable to take the place of the place holder but cursor execute needs two parameters, so a single tuple is created to avoid creating a string (which won’t be recognized) by using a comma after the input variable. A print statement is used to confirm a successful deletion of the record and instructions to return to the GUI tab. Like most SQL functions in this program, the cursor is closed last.
3. Update\_items:
   1. An interesting function that was added much later to the program was the update function that would allow users to update existing cells within certain fields. This function still follows the staple structure of abort functions and inputs. The function will require the user to key in three inputs: the column/field name, the updated value and the ID of the item whose column/field cell would be changed as per the updated value. The abort function is available at all stages and there was an attempt to implement a data validation system that was based on the same system in the previous functions, This, however, proved to be more troublesome than it was thought to be with many bugs and issues still remaining. The overall functionality is still present but so are the bugs. Lastly, the SQL command code follows with its respective placeholder and a special placeholder for the column\_name which needed to be formatted in with the .format function due to some complications. The changes are committed and the cursor closed as usual after a verification message of the change is shown to the user.
4. view\_items:
   1. The view function also makes use of the cursor in a similar manner to the above and executes the respective command to SQL syntax. The nature of this function is stoic and requires no input from the user besides pressing a button. The cursor then uses the SELECT \* FROM items command to call all the rows and result = cursor.fetchall() is used to assign a variable to those rows. A for loop is then used to print each row in an easy-to-read format in the terminal with all the fields displayed in a block and a print statement to tab back to the GUI. Like most SQL functions in this program, the cursor is closed last.
5. search\_items:
   1. The usual abort code, cursor assignment, basic input and sql commands are present in this function. It naturally uses a different command with a placeholder for the user input. The code then, once sent to SQL, searches for an exact match for the users input among the item names. The single tuple method along with .fetchall() play an important part of the function. Once located, the row is then printed in the same format as the view function. On the other hand, if the item is not found, an error print statement is then presented to the user. It informs the user of the absence of the item and asks them to return to the GUI for another task. Like most SQL functions in this program, the cursor is closed last.
6. exit\_program:
   1. Lastly, there is the exit function which is perhaps the simplest and smallest of all the functions. It consists of two lines that respectively closes the database and the GUI window.
7. Button functions
   1. Two more functions were added later to make the GUI more user friendly. The goal being to introduce highlight colors when hovering over certain buttons to prevent misclicking a button. For the function to properly work it consists of two functions each with one parameter. Both buttons make use of the .bind function from tkinter module to bind the events ‘enter’ and ‘leave’ to each button. ‘enter’ and ‘leave’ are essentially the events when the cursor moves in proximity within the respective button widget and when it moves out of that area.
      1. On\_enter
         1. So when the mouse cursor moves over the respective button widget, this function is called and will change the color of the button to the defined color as per the function (cyan in this program).
      2. On\_leave
         1. When the cursor leaves that area of the button widget, On\_leave is called and it will change the background color of the button from the defined cyan to the default systembuttonface.
8. **Screenshot of coding and output**

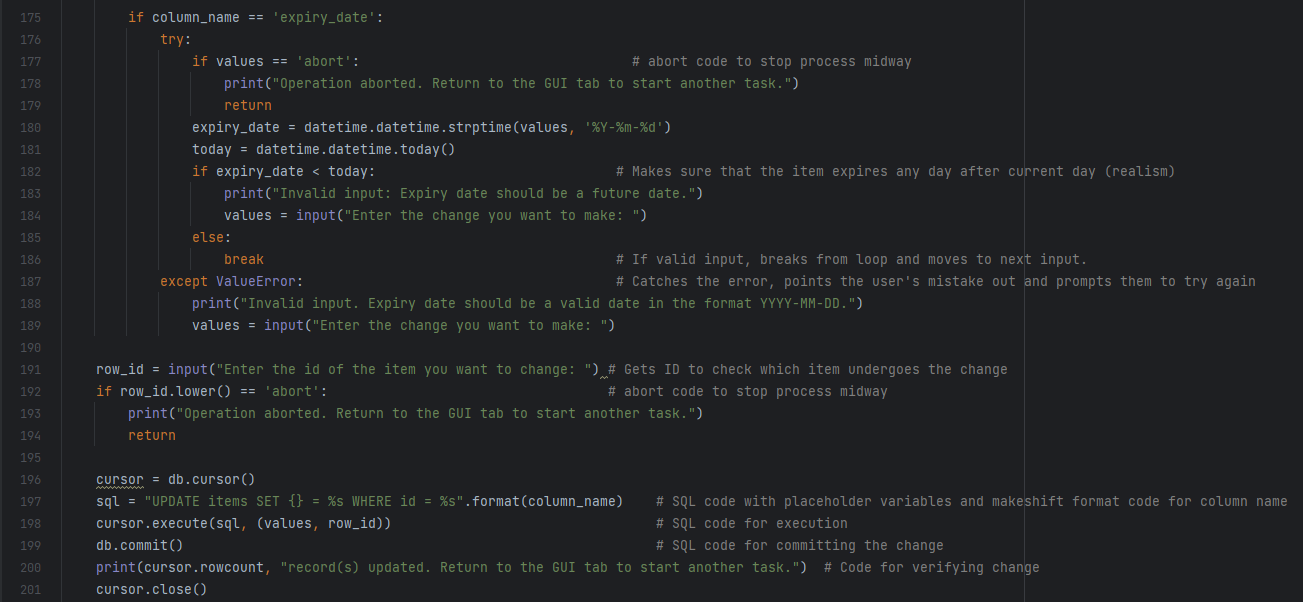
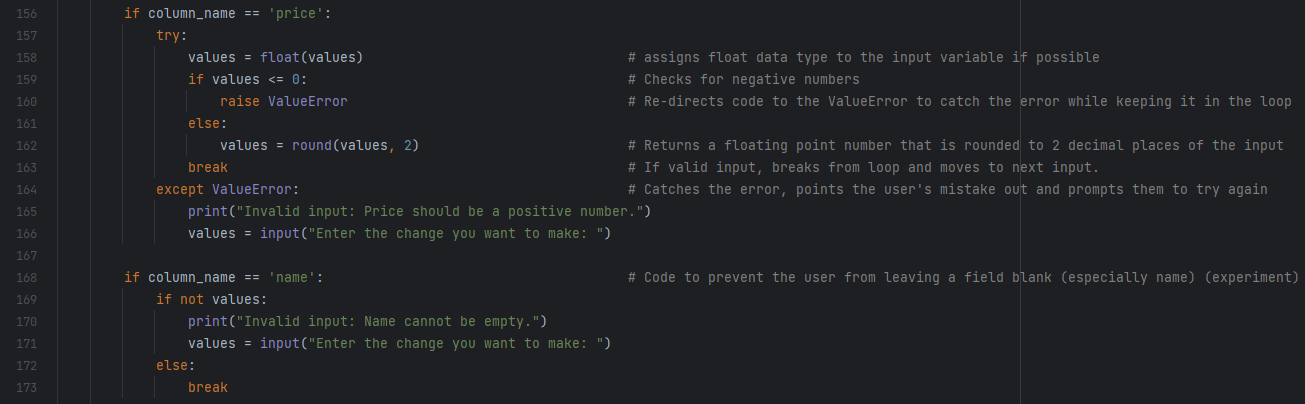
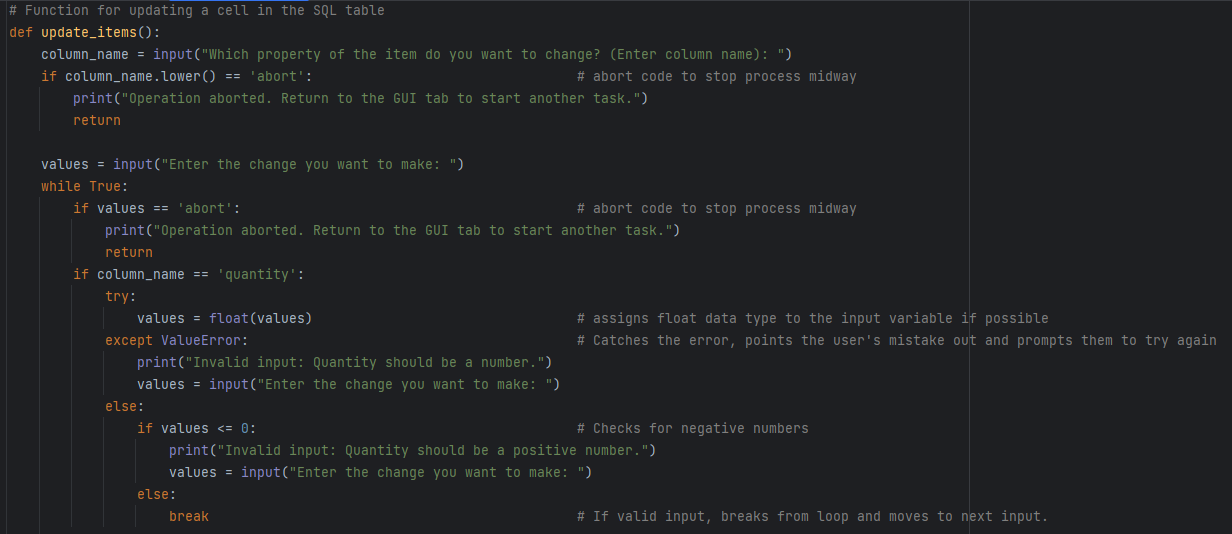
**Imported modules, database and table creation:**

Add\_item function:

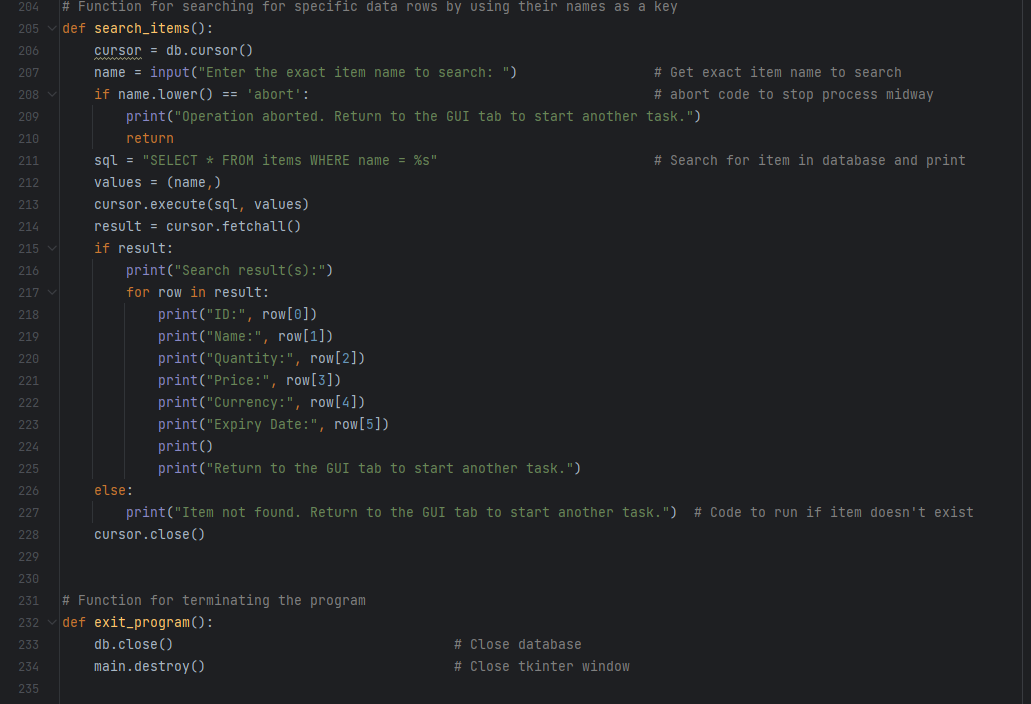


Remove\_item and view\_item function:

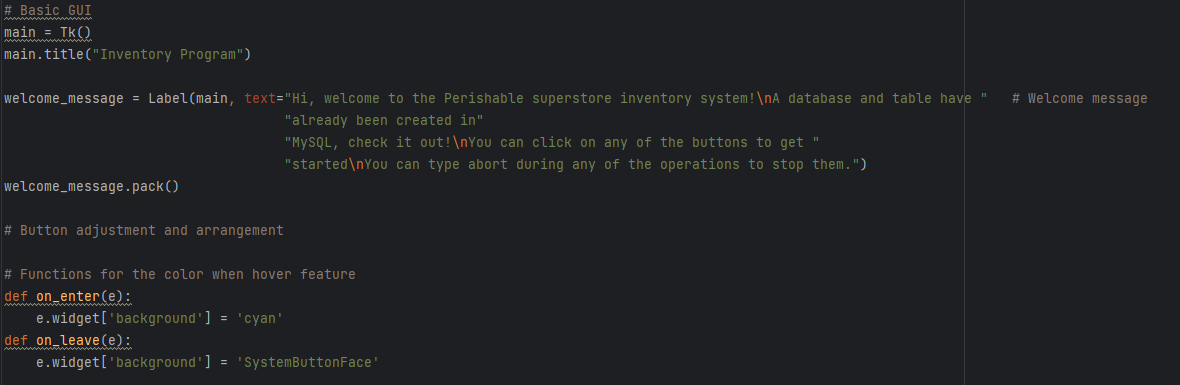




Search\_items and exit\_program functions:



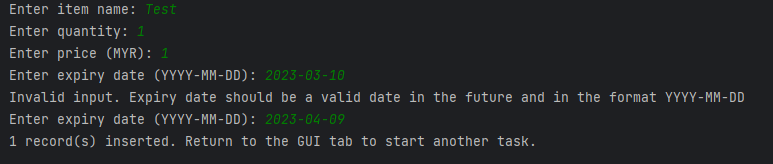
GUI functions and buttons:

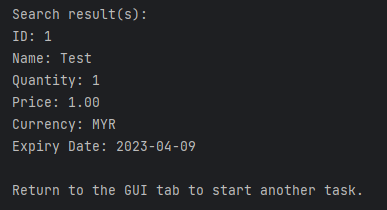




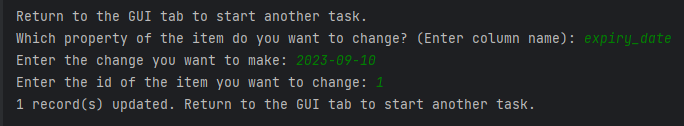
Outputs:

Add\_item:

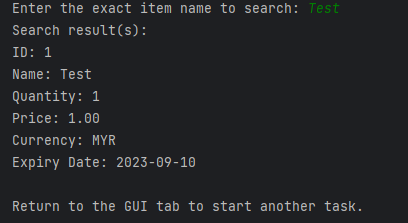


View\_item:

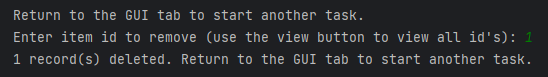
Update\_item:



Search\_item:

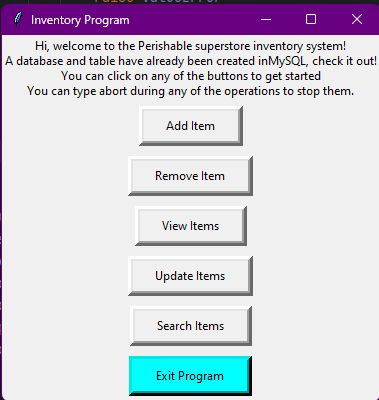


Remove\_item:



Abort code:



GUI and exit\_program:

1. **Future work (Suggestion for improvement)**

Future ideas:

* SQL interaction for updating the databases (Done)
* Having the user name the database upon creation and delete them accordingly
  + Ran into some sever errors that did not seem to have online solutions
  + mysql.connector.errors.ProgrammingError: 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''Salsa” (Database example name) at line 1
  + Ran a multitude of possible solutions to no avail
* Having the user the option to choose multiple other currencies instead of MYR
  + WIP and somewhat unreasonable, thus it just remains and idea for now due to unrealism.
* Better GUI implementation or overhaul into CLI interaction completely
* More accurate expiry-date data validation system
  + Accounting for February’s inaccuracy of 28 days (Done)
  + Proper option to skip the expiry-date system for products that may not have one
  + Leap years and other inconsistencies
* Improve update function
  + Improve the data validation and abort code to be more efficient

1. **Conclusion**

In conclusion, this is a simple program with a simple and friendly GUI. The program can create (and write to an existing) database in MySQL workbench, additionally it also creates a table with fields that match the purpose. It is also able to add rows of data to each field that include quantity and price. It is also able to remove respective rows, view all the rows, update certain cells, search for a specific row and close the program with a button. The code also has some basic error handling and data validation for specific circumstances to improve realism and interaction quality for the user. The simple inventory program could be improved by adjusting existing functions with advanced features or by introducing new functions for advanced features or missing basic ones.