Shop Inventory Counter

Dr. Chu Yih Bing

BER2023 Object Oriented Programming

3/27/23

Name	ID
Mohammed Ahmed Hussien Mohammed	1002163078
Mohamed Tarek Essam	1002163249
Abdullah Ahmed Abdullah Elhaddad	1002060687
Ali Isam Husam Al-Turaihi	1002164751
Abdulrhman Almonajed	1001955264

Contents

Chapter 1: Introduction

A form of digital circuit known as a modulo 10 up/down counter counts up or down from 0 to 9 in steps of 1, then loops back around to 0 again. It is frequently seen in electrical equipment like calculators, timers, and digital clocks.

Python is a well-liked computer language that may be used to model and simulate digital circuits. A modulo 10 up/down counter that can count up or down depending on user input can be made using Python.

With Python, we may utilize conditional statements, loops, and arithmetic operations to build a modulo 10 up/down counter. The built-in modulo operator (%) in Python can be used to make sure that the counter wraps around to 0 once it hits 9.

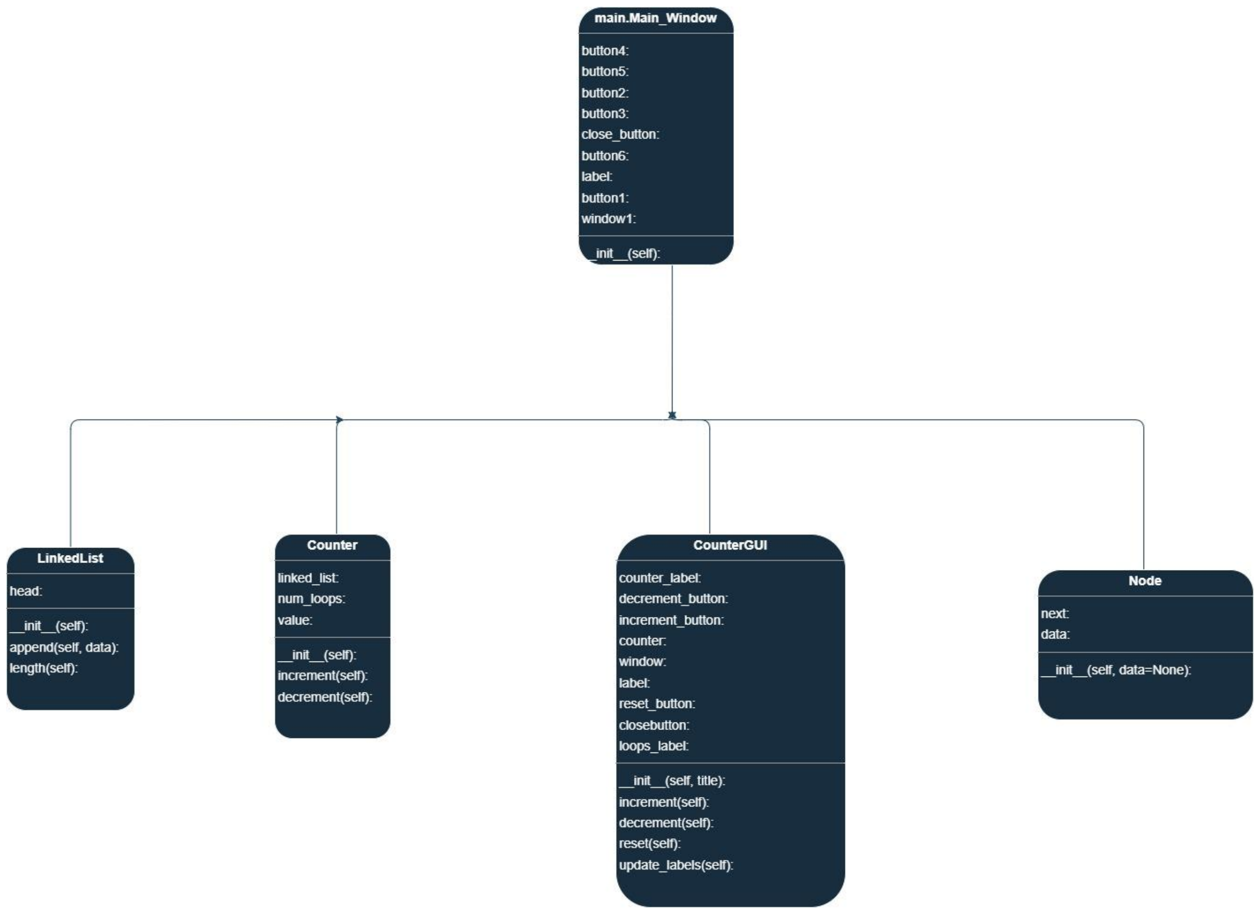
Overall, designing a digital circuit using conditional statements, implementing the circuit using Python code, and testing the circuit to make sure it works are the steps involved in developing a modulo 10 up/down counter in Python.

There are several benefits of using a modulo 10 up/down counter in digital circuits:

1. Effective counting: Without the use of complicated circuitry, the modulo 10 up/down counter may count from 0 to 9 either up or down. It is an efficient and straightforward method for counting digital circuits.
2. Reliability and accuracy: The modulo 10 up/down counter ensure that the count is always precise and constant.
3. Minimal power consumption: The modulo 10 up/down counter is a great option for battery-operated devices because it uses very little power.
4. Cost-effective: The modulo 10 up/down counter is a cheap option for many digital circuit applications since it is a low-cost solution that needs little in the way of circuitry and components.

In conclusion, the modulo 10 up/down counter is a straightforward, effective, and economical method of counting in digital circuits. It is a popular choice for many applications thanks to its precision and dependability, and it is a flexible and adaptable solution thanks to its low power consumption and simple integration.

Chapter 2: UML Diagram



In this diagram, we have 5 classes that all work out together. The first two classes are the Node and LinkedList classes. They work together to create the linked list. The Counter class has two variables which are the number of loops (num_loops) and the value of the counter itself. It has two methods called increment and decrement. The main two classes are the Main_Window and the CounterGUI. The main window includes all the products in forms of buttons so that the user can choose which product they want to select. The counter GUI class is a class that uses the counter class and it is displayed when the user presses on a product button in the main window. Further explanation of the code is in the next chapter.

Chapter 3: The Code, Discussion, & Demonstration

```
# importing necessary libraries
import tkinter as tk
import pickle

# class definition for a node to use in the linked list
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

# linked list definition class
class LinkedList:
    def __init__(self):
        self.head = Node() # using the node class to implement the linked
list

    def append(self, data): # appends a new value to the end of the list
        new_node = Node(data)
        cur = self.head
        while cur.next is not None:
            cur = cur.next
        cur.next = new_node

    def length(self): # returns the length of the list
        cur = self.head
        total = 1  # fix: initialize total to 1 instead of 0
        while cur.next is not None:
            total += 1
            cur = cur.next
        return total

# modulo 10 up/down using linked list
class Counter:
    def __init__(self):
        self.linked_list = LinkedList() # using linked list class to
implement the counter
        self.linked_list.append(0)
        self.num_loops = 15 # set initially to 15, because we want to have
some inventory when using the program
        self.value = 0

    def increment(self): # modulo up
        cur_node = self.linked_list.head
        while cur_node.next is not None:
            cur_node = cur_node.next
        if cur_node.data == 9:
            self.linked_list.append(0)
            self.num_loops += 1
            self.value = 0
        else:
            cur_node.data += 1
            self.value += 1
    def decrement(self): # modulo down
        if self.linked_list.length() == 1 and self.linked_list.head.data
== 0:
            self.value = 0
            return
        cur_node = self.linked_list.head
        prev_node = None
        while cur_node.next is not None:
            prev_node = cur_node
            cur_node = cur_node.next
        if cur_node.data == 0:
            if self.num_loops == 0:
                self.value = 0
            else:
                prev_node.next = None
                if self.linked_list.length() > 1:
                    self.num_loops -= 1
                self.value = 9
        else:
            cur_node.data -= 1
            self.value -= 1

# GUI program
class Main_Window:
    def __init__(self):
        self.window1 = tk.Tk()
        self.window1.title("Inventory Management System")
        self.window1.geometry("400x450")
        self.window1.config(bg='#90C2E7')

        self.label = tk.Label(self.window1, text="Choose Product: ",
bg="#454545", fg="white", font=("Arial", 24))
        self.label.pack()

        # using the lambda function to open a new window whenever any
product's button is clicked
        self.button1 = tk.Button(self.window1, text="Shirts",
command=lambda: self.new_window("Shirts"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button1.pack()

        self.button2 = tk.Button(self.window1, text="T-Shirts",
command=lambda: self.new_window("T-Shirts"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button2.pack()

        self.button3 = tk.Button(self.window1, text="Pants",
command=lambda: self.new_window("Pants"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button3.pack()
        self.button4 = tk.Button(self.window1, text="Shorts",
command=lambda: self.new_window("Shorts"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button4.pack()

        self.button5 = tk.Button(self.window1, text="Socks",
command=lambda: self.new_window("Socks"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button5.pack()

        self.button6 = tk.Button(self.window1, text="Shoes",
command=lambda: self.new_window("Shoes"), bg="#454545", fg="white",
padx=12, pady=8, font=("Arial", 24))
        self.button6.pack()

        self.close_button = tk.Button(self.window1, text="Close",
command=self.window1.quit, bg="#454545", fg="white", padx=12, pady=8,
font=("Arial", 24))
        self.close_button.pack()

        # setting the new_window function to a new class for simpler
coding
    def new_window(self, title):
        CounterGUI(title)

# the class for the product's pop-up window
class CounterGUI:
    def __init__(self, title):
        self.counter = Counter()
        self.title = title



        self.window = tk.Tk()
        self.window.title(title)
        self.window.geometry("300x300")

        self.label = tk.Label(self.window, text=title + "'s inventory")
        self.label.pack()

        self.counter_label = tk.Label(self.window, text="0",
font=("Arial", 24))
        self.counter_label.pack()

        self.increment_button = tk.Button(self.window, text="Bought",
command=self.increment)
        self.increment_button.pack()

        self.decrement_button = tk.Button(self.window, text="Sold",
command=self.decrement)
        self.decrement_button.pack()

        self.reset_button = tk.Button(self.window, text="Reset",
command=self.reset)
```

```
        self.reset_button.pack()

        self.loops_label = tk.Label(self.window, text="Boxes: ",
font=("Arial", 12))
        self.loops_label.pack()

        self.closebutton = tk.Button(self.window, text="Close",
command=self.window.quit)
        self.closebutton.pack()

        self.save_button = tk.Button(self.window, text="Save",
command=self.save_counter)
        self.save_button.pack()

        self.message_label = tk.Label(self.window, text="")
        self.message_label.pack()

        self.load_counter()

        self.update_labels()

        self.window.mainloop()

        # using pickle to save our progress
    def save_counter(self):
        with open(f"{self.title}_counter.pickle", "wb") as f:
            data = {
                "value": self.counter.value,
                "num_loops": self.counter.num_loops,
                "linked_list": self.counter.linked_list
            }
            pickle.dump(data, f)

        # using pickle to load the progress
    def load_counter(self):
        try:
            with open(f"{self.title}_counter.pickle", "rb") as f:
                data = pickle.load(f)
                self.counter.value = data["value"]
                self.counter.num_loops = data["num_loops"]
                self.counter.linked_list = data["linked_list"]
        except FileNotFoundError:
            pass

       # increments the counter by referring to the counter increment
method and then updating the labels
    def increment(self):
        self.counter.increment()
        self.update_labels()

       # decrements the counter by referring to the counter decrement
method and then updating the labels
    def decrement(self):
        self.counter.decrement()
        self.update_labels()

        # method for resetting the counter to 0
    def reset(self):
        self.counter = Counter()
        self.counter.num_loops = 0
        self.update_labels()

       # updates the message that is presented to the user, by seeing the
num_boxes. then updates the message whenever called
    def update_labels(self):
        self.counter_label.config(text=str(self.counter.value))
        self.loops_label.config(text=f"Boxes: {self.counter.num_loops}")

        if self.counter.num_loops > 0 and self.counter.num_loops <= 3:
            message = "Order much more boxes. \nTop seller product."
        elif self.counter.num_loops > 3 and self.counter.num_loops <= 7:
            message = "Order more boxes. \nGood selling product."
        elif self.counter.num_loops > 7 and self.counter.num_loops <= 11:
            message = "Order some boxes. \nRegular product."
        else:
            message = "Don't order any more boxes. \nProduct is not
selling enough."

        self.message_label.config(text=f"Boxes: {self.counter.num_loops} -
" + message, bg="#FF0800", fg="white")

# necessary to run the program
my_window = Main_Window()
my_window.window1.mainloop()
```

The objective of our Assignment is to use linked list and classes to design a modulo 10 up/down counter. So, we chose to implement this counter as an inventory management system. and to make it better and more interactive we decided to do a GUI program. The GUI has a main window which shows different buttons for different products and a close button. If you click any button a new window will pop up, containing a modulo 10 up/down counter. It has the counter displayed with a sell(decrement) button and a bought (increment button) and a few boxes counter that counts one up whenever the counter reaches 9 and increments, not only that but it decrements if it was decremented from 0. We set the number of boxes initially to 15 so we have some inventory to test the program. if a product sells a lot and there are no more boxes left a message will appear to the user alerting him to buy new boxes and informing him that this is a good selling product, and it goes the same way for the products that do not sell much a message will appear to the user. Finally, each product's counter is different and independent on the other one, and we have a save button that saves your progress for later. and of course, a close button to close the window.

An inventory management system GUI application is implemented in the provided Python code. It has two classes that are utilized to build a linked list data structure, named Node and LinkedList. An object that may be used to count the objects in the inventory is created using the class Counter. The application's main window is created lastly using the class Main_Window.

The init function of the Node class only accepts the argument data. The class variables data and after the values of data and None are initialized. The init function of the LinkedList class initializes the head of the linked list to a Node object with None as the default value for the data. Also, it contains a function called append that adds a new node with the supplied data at the end of the linked list. The linked list's length is returned using the length method.

The init function of the Counter class adds a node with the value 0 to the linked list and initializes a LinkedList object. Also, it sets the value and num loops instance variables to 15 and 0, respectively. The valuable of the counter is incremented using the increment method of the Counter class. It initially locates the linked list's final node before determining if its value is nine. If so, the value instance variable is set to 0, the num loops instance variable is increased, and a new node with the value of 0 is appended to the end of the linked list. Otherwise, both the value instance variable and the last node's value are increased.

The Counter class's decrement method is used to lower the counter's value. First, it determines whether the head node's value is 0 and the linked list's length is 1. On the off chance that it is, the worth example variable is set to 0 and the technique returns. If not, it looks for the linked list's final node and checks to see if its value is 0. Assuming that it will be, it checks if the num_loops occasion variable is 0. The instance variable's value is set to zero in this case. If this is not the case, the value instance variable and num_loops instance variables are both decremented, and the linked list's last node is removed from it. The value instance variable is also decreased if the value of the last node is not 0.

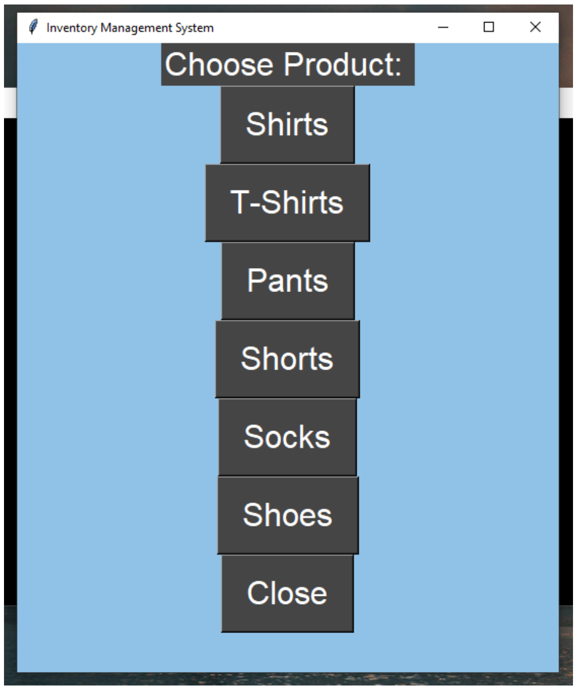
The main window of the program is made using the Main_Window class. A tkinter object is initialized, and its title, geometry, and background color are all set. The new_window function is then called after creating several buttons, each of which has a unique title when clicked.

The GUI for each inventory item is made using the CounterGUI class. Both a Counter object and a tkinter object are initialized. It changes the title of the tkinter object to the value specified by the title parameter. It then generates two labels: one for the counter's value and one for the title of the inventory item. Moreover, it adds three buttons: one for increasing the counter, one for decreasing it, and one for resetting it. It also adds a label to show the quantity of loops and a button to save the counter's current state to a file.

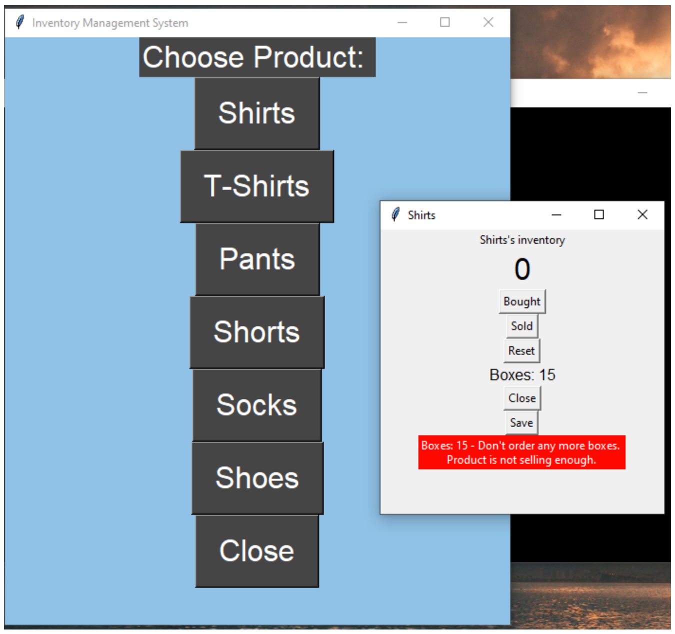
Overall, this Python code uses the tkinter GUI library and a linked list data structure to implement an inventory management system. It is possible to add more features, like the ability to support multiple users and save and load inventory data from a file.

Screenshot example:

We run the program:

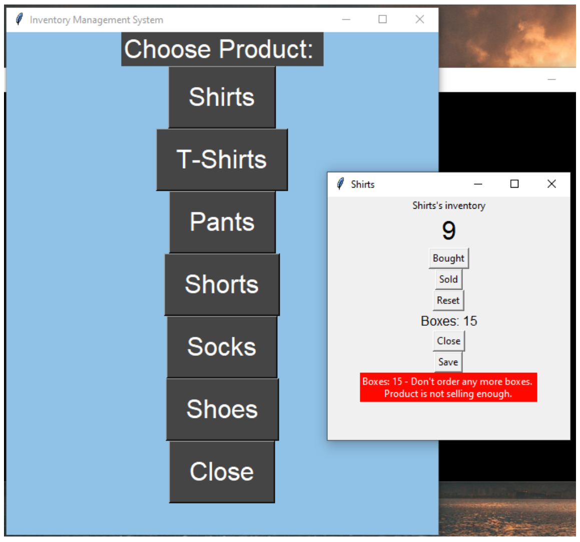


We click on shirts:

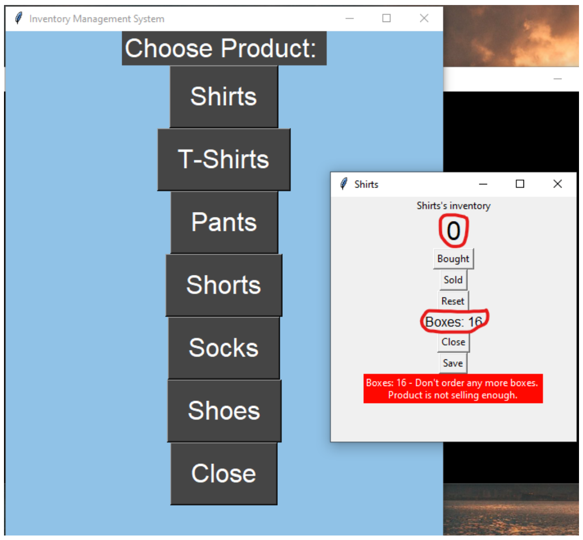


Boxes: 15 - Don't order any more boxes.
Product is not selling enough.

We click on "bought" until it we get on the counter 9 and then we click once again, and this happens:

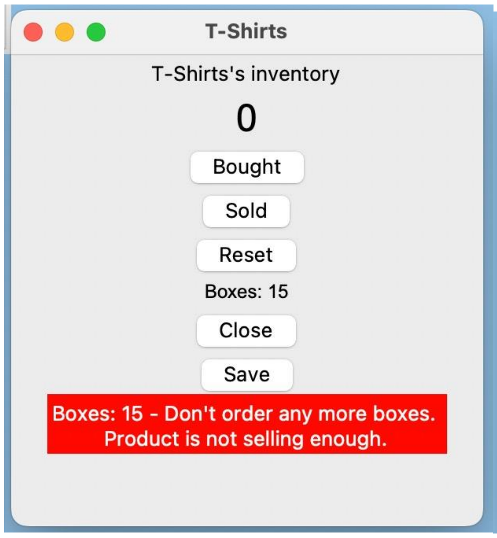


Boxes: 15 - Don't order any more boxes.
Product is not selling enough.



Boxes: 16 - Don't order any more boxes.
Product is not selling enough.

We try the same thing but with the "sold" button (the counter goes backward to zero and then this happens):


T-Shirts

T-Shirts's inventory

0

Bought

Sold

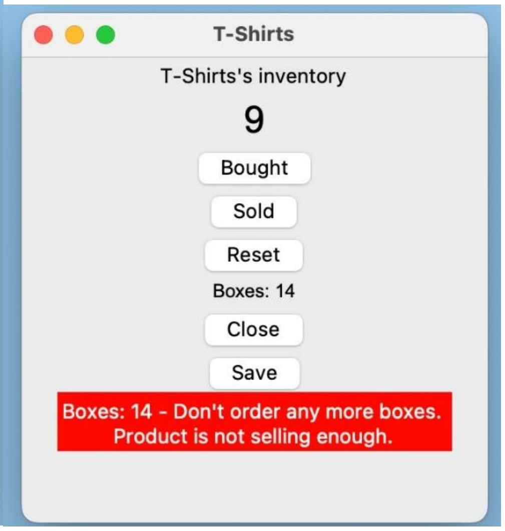
Reset

Boxes: 15

Close

Save

Boxes: 15 - Don't order any more boxes.
Product is not selling enough.


T-Shirts

T-Shirts's inventory

9

Bought

Sold

Reset

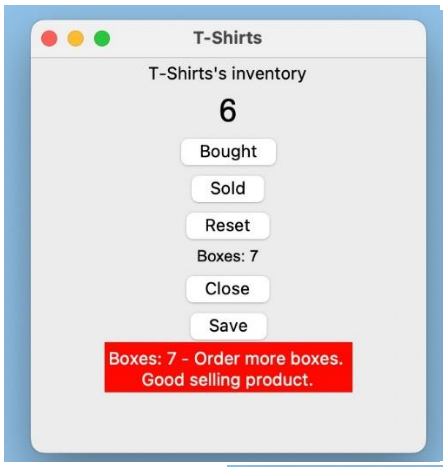
Boxes: 14

Close

Save

Boxes: 14 - Don't order any more boxes.
Product is not selling enough.

We try selling more and more boxes to see the messages displayed to the user.


T-Shirts

T-Shirts's inventory

6

Bought

Sold

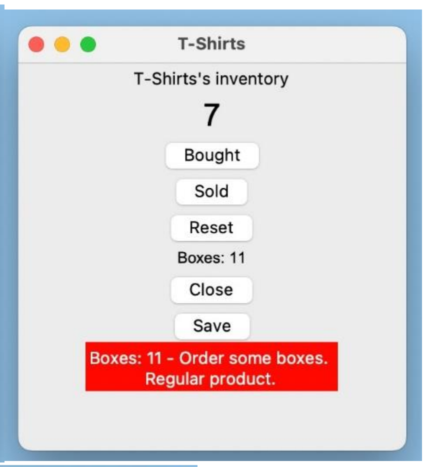
Reset

Boxes: 7

Close

Save

Boxes: 7 - Order more boxes.
Good selling product.


T-Shirts

T-Shirts's inventory

7

Bought

Sold

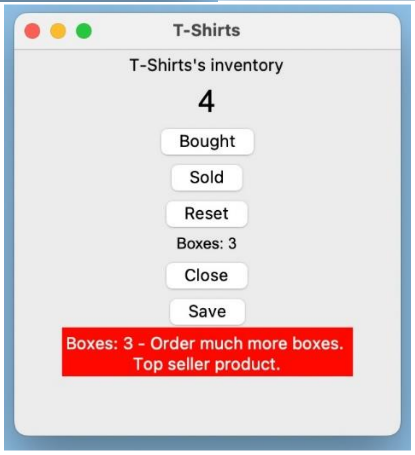
Reset

Boxes: 11

Close

Save

Boxes: 11 - Order some boxes.
Regular product.


T-Shirts

T-Shirts's inventory

4

Bought

Sold

Reset

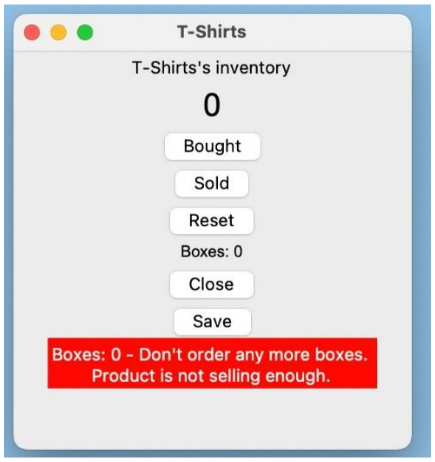
Boxes: 3

Close

Save

Boxes: 3 - Order much more boxes.
Top seller product.

If you click on the “reset” button you get this new message, and everything in the counter resets (inventory from boxes and individual shirts):



Boxes: 0 - Don't order any more boxes.
Product is not selling enough.

When you click on the “save” button, it saves any changes done while running the program. When you save it creates a file that saves the data in the file where you executed the program from. And if you click on “close” button it closes the whole program.

The same goes to all categories of cloths. Every category has the same buttons and same functionality. Except for the last button “close”. It closes the program.

Chapter 4: Conclusion

In conclusion, the development of a Python class-based linked list inventory management system. The system is made up of a primary window and secondary windows for the counters for each product. Using a modulo 10 up/down counter, the application notifies the user when inventory runs low, or sales are strong. The Node and LinkedList classes are used to generate the linked list data structure, while the Counter class is used to implement the counter. The program's main window is made using the Main Window class, and each product's counter's GUI is made using the CounterGUI class. Using the tkinter GUI library, the software enables storing and loading of inventory data from files. Generally, more functionality, including multi-user support, might be added to the application.

In addition, it’s important to note, before getting into the implementation specifics, that an inventory management system like the one discussed in the article may be advantageous for companies of all sizes. Companies may control restocking, price, and promotions by keeping track of inventory levels and sales information. Businesses may cut expenses, increase customer happiness, and eliminate waste by using an effective inventory management system.

Furthermore, classes and linked lists are examples of how object-oriented programming and data structures are effective in software development. These ideas may be used for a variety of programming jobs, from straightforward scripts to intricate programs. Developers may save time and effort while still creating high-quality software by leveraging modular, reusable code.

Finally, the programme emphasizes how crucial user interface design is to the creation of software. The inventory management system is more approachable and intuitive for users thanks to the GUI application mentioned in the article, which offers a user-friendly manner to interact with it. Users' requirements and tastes may be taken into account while developing software, resulting in a product that is both useful and interesting to use.

RUBRIC

Criteria (Marks %)	Score				Student Marks (Score/4 x Marks %)
	4	3	2	1	
Data Declaration (6%)	Data/header is declared properly wherever necessary. All the important information is well structured. Proper indentation, commenting, object/function abstract and instantiate is observed.	Misuse of certain data types. Related information is included but not well structured. General indentation, commenting, object/function abstract and instantiate is observed.	Data/header is not declared. Related information is included but not well structured. General indentation, commenting, object/function abstract and instantiate is observed.	Data/header is partially declared. Related information is included but not well structured. General indentation, commenting, object/function abstract and instantiate is observed.	
Concept Sketch (6%)	The sketch represents well the underlying system design of the work. Proper labelling and addressing is observed. The sequence is well planned, unique and innovative. All the important/ real world features are considered into the program.	The sketch represents overall system design of the work. Proper labelling and addressing is observed. General sequence planning. Some important/real world features are considered into the program.	The sketch represents very basic system design of the work. Limited labelling and addressing is observed. Limited sequencing and planning. A few important/ real world features are considered into the program.	The sketch represents poorly the system design of the work. Improper labelling and addressing is observed. Majority of the sequencing strategy is missing. The program does not considered any real world feature.	
Design Strategy (6%)	Develop new approach or provide innovative solution to solve the problem. The strategy should be short, simple and optimized. Only a few numbers of if,else statement and well use of programming algorithms is observed.	Develop new approach or provide improvement of previous solution to solve the problem. The strategy should be short and simple. Certain usage extent of if,else statement and proper use of programming algorithms is observed.	General solution is provided with or without reference to previous solution. The strategy is not optimized and inconsistent is found throughout the work. Modest level of object-oriented programming skill is observed.	No apparent strategy or poor design solution is provided. The code does not contain fair use of functions, statements, operators and etc. Errors and inconsistencies are found throughout the work.	

Content and Discussion (6%)	In-depth content which explain the understanding of the work. The work is well elaborated with negligible errors. Proper evaluation and testing is conducted. The application is able to function and deployable. All necessary contents are covered in the work.	The work is well elaborated but with some errors found. Evaluation and testing is conducted accordingly. The application contains a few bugs. Some contents explaining the work are missing.	The work is elaborated modestly but contain errors. Limited evaluation and testing. The application contains too many bugs and produce unwanted outcome. Important contents explaining the work are missing.	The work is elaborated poorly and contain many errors. Proper evaluation and testing are not found. The application is either not running or incomplete. No clear understanding of the work is shown.	
Formats and Aesthetics (2%)	Consistent format and labeling.	Consistent format and labeling but with minor errors.	Inconsistent format and labeling. Many errors are found throughout the work.	Poor formatting or the work does not conform to any format or labeling.	
Organization (2%)	Information is presented in a logical and interesting way, which is easy to follow. Purpose is clearly shown and explains the structure of work	Information is presented in a logical manner which is easy to follow. Purpose is clearly shown which assists the structure of work	Work is hard to follow as there is very little continuity. Purpose is shown but does not assist in following the work.	Sequence of information is difficult to follow. No apparent structure or continuity is found. Purpose of work is not clearly shown.	
Reference (2%)	Reference is complete and comprehensive. Consistent and logical referencing system used.	Minor inadequacies in references found. Consistent referencing system used.	Inadequate list of references or references in text. Inconsistent or illogical referencing system used.	Poor referencing or no clear referencing system is shown.	
				Total Marks	