湖北工業大學

HUBEI UNIVERSITY OF TECHNOLOGY

Thesis (project) paper on

"Offline Inventory System"

Created by

Rimon Mahmud

ID: 1811561124

18lq Computer Science

Under The Guidance of

Rong Gao

(Department of Computer Science)

I T Academy

Hubei University of Technology

Wuhan, Hubei Province, P. R. China

Table of contents

Introduction

The Digital Revolution is the shift from mechanical and analog electronic technology to digital electronics. Which began within the latter half of the 20th century, with the adoption and proliferation of digital computers and digital record keeping that continues to this day. Implicitly, the term also refers to the sweeping changes caused by digital computing and communication technologies during this period. It is also known as the Third Industrial Revolution. Analogous to the Digital Revolution marked the start of the Information Age.

Central to the present revolution is mass production and widespread use of digital logic, MOSFETs (The metal–oxide–semiconductor field-effect transistor), microcircuit (IC) chips, and their derived technologies, including computers, microprocessors, digital cellular phones, and therefore the Internet. These technological innovations have transformed traditional production and business techniques.

Technology is around us, and it is integrated into our diurnal lives. Regular tasks now have a brisk and more effective way of doing them thanks to the innovative way in the field of technology. Still, a generally overlooked aspect of these conveniences is also among its most important, videlicet software.

Software is what we suppose when we frequently suppose what we suppose of when we suppose of ultramodern technology. It is what makes the difference between the analog days of history and the digital world moment. However, what is software?

Frequently, the software is used in confluence with the terminal tackle. In simple terms, software is the program and instructions that a computer uses to make itself run. Tackle, the contrary, is the factual material that comprises a computer. Far too frequently, we suppose of getting bias that is more important or the rearmost specifications for smartphones or computers, but the software is what actually makes it work right. It is what we affiliate with, the icons that we click, the programs that we run, and the blank space where we class our words- that is all software. Just as important as good tackle is important, good software is indeed more so.

In addition, because of how constantly we use software, numerous people fail to see its value. Numerous companies with a stoner interface for their guests will do

far better if they have intuitive software. Indeed individuals looking to be competitive in the job request will gain applicable chops if they invest in software knowledge. Below are some further conversations on the significance of software, and where its value lies.

All the rearmost and topmost computer tackle in the world is useless if there is not any software to interpret it. Software is there to use tackle from making sure that, it runs efficiently, to furnishing people with the rearmost functionality upgrades. Indeed aged tackle benefits from bettered software support, videlicet motorist upgrades.

With how reliant the world is on computers, and how computing requires stronger and faster factors to serve, optimization of tackle through clever software tricks is fast getting a necessity. Indeed smartphones currently are filled with calculating tasks that bear advanced-end processing capabilities. And only good software can maximize the tackle to achieve that.

Software is important in the sense that it can indeed mandate how we use our bias. How commodity looks, feels, and functions are each important factors in how we interact with computers, and how efficiently and directly we can use them. Currently, with the amazing progress of software, indeed the most complex of tasks can now be fulfilled through simple input of information.

Type certain information in a specific box, and also you will be suitable to gather results many seconds latterly- a far cry from computers running outstations from half a century gone. This is all thanks to the graphical stoner interface that we all use and enjoy. The telephone's invention gestured a major change in the world. Humans can now communicate presently, and it came to a major corner in mortal history. People can now transfer critical information and instructions anyhow of any distance. With the internet and its coexisting software, this is magnified indeed more. People can unite and communicate vastly easier, and with a multitude of veritably effective tools at their disposal. Participating in work-in-progress systems, making real-time adaptations, and agitating progress are effects that are taken for granted by ultramodern citizens, but these were rather delicate to achieve also. With all the communication and collaboration technology.

Of course, among the most significant advantages that software has handed us is how it makes everything more effective. Accountants can now go through months' worth of data and numbers without having to physically sift through heaps of papers. Utility bills, mortgages, and other finances can be paid ever. Contrivers can

make digital renders of their workshops to reduce time and allow for immediate changes. These are all advancements brought about by software and its constant development. The software has come to be the driving force of invention and advancements, and we can anticipate it to drive further inventions in further times to come. Just as we continuously work on the development and advancement of tackle technology, so do we need to admit the vital part software plays in our world moment? Robotics would not flourish as important as it does now without the combination of slice-edge tackle and ingenious software programming. It is important that we know and understand how software can help make our lives easier.

Five years ago, Marc Andreessen famously stated that "software is eating the planet," and it has and is in ways that he probably could not have imagined even five years ago. Applications are no longer a nice-to-have. They play a central role in how and why businesses operate, and corporations are producing them in unprecedented numbers. In fact, a typical $500 million-plus enterprise today has developed quite 3,079 applications (CIO Magazine) and has a mean of roughly 600 mission-critical applications. Financial services organizations have even more mission-critical applications – with a mean of 800.

What is all this software doing? It is the primary way that companies run their businesses today – from interacting with customers, prospects, and partners to creating business decisions. Software is actually powering our world; it enables the functionality of our banks, power plants, communication, and combat equipment. Hospitals, medical devices, and important infrastructure, like 911 systems and the electric grid, all depend upon software to operate effectively and efficiently.

How did we go from only software companies making software to "every company may be a software company"? It started with companies creating software to make their internal processes more efficient. Then some companies started using it to interact with customers – and the genie was out of the bottle. The bar had been raised, and customers expected to be ready to interact with companies online. To survive, enterprises today need to create not only products and services their customers want but also software that allows customers to interact with the company and its products and services how they want when they want.

However, more is needed to become a software company; you also need to become a fast-moving software company. Customers are demanding that companies not only meet their expectations but continue with them as well. As end users expect ever-greater functionality and mobile accessibility, companies are forced to stay pace with the speed of technological advancement. Moreover, if they are doing not,

their customers will look elsewhere. Ultimately, companies are competing with each other to both create software, and obtain it to market quickly.

In 2014, a Forbes article stated that “becoming Agile will steadily become a requirement just to stay in business. In effect, for many companies, failure to accumulate digital agility will be an existential threat and so, establishing digital agility has become in effect a strategic necessity.” additionally, that was written two years ago; it's an even greater necessity today

In addition, the role of applications in businesses today is not just about making customers happy. Most need software to form business decisions that allow them to function. A recent SANS report concluded, “Most successful businesses are data-driven. for instance, a grocery chain will stock each store with the combination of products most likely to sell. A big-box ironmongery shop will carefully analyze demographic data before expanding to a new location … Without accurate data and applications to draw reliable conclusions, these businesses would fail. Applications allow them to function.”

In the end, it’s not about whether the software will transform your business, but how it'll. A recent article in Entrepreneur Magazine sums it up: “Software has evolved from being not a part of your business to becoming a way to run your business … and now, increasingly, are going to be your business.”

Digital inventory management

After two centuries, a paper-and-pencil Inventory system still works, but given the inconceivable advances in simple-to-use digital Inventory management software, why use similar limited tools? Modern, pall-grounded Inventory systems, like community Suite, are not only affordable but also largely effective at removing the donkeywork of Inventory — lightening your workload and saving you plutocrats on food costs. Keep reading to quest out how digital Inventory management software from community Suite can help time and streamline your eatery operation.

Digital Inventory management allows you to trace Inventory with important, yet simple software. A digital Inventory management system makes it easy to observe how important food you have in stock, offering you superb numerous benefits including increased effectiveness and a dropped workload.

The most effective way to conduct Inventory management is through a system of cycle counting. This means counting a portion of — rather than your entire total Inventory over a set period of days and rotating the list of effects counted every day. Over that destined cycle, all particulars are visited be counted so as to support their value. As an illustration, particulars in Inventory are grouped as A, B, or C particulars according to their value as a chance of total Inventory.

Group particulars regard for 70 percent of Inventory value, B particulars 15 percent, and C particulars the smallest quantum precious 15 percent. While this might sound grueling to take over with pencil-and-paper or indeed a spreadsheet program, it is simple to take over with inventory software that categorizes all of your particulars as A, B, or C and tells you which of them are counted when. Using this type of digital inventory system saves, periods, and boosts productivity.

Numerous drivers have exceptional directors on staff who have their own "system" for inventory management and buying. Still, those unique strategies are not always easy to hold out if they take a holiday or leave for other employment.

A digital system, still, functions identically for everybody and is fluently trained. Once your inventory is entered into a digital inventory system, it's simple to calculate reduction by comparing current inventory to the maximum recent period — not by counting on someone's memory. Through integration with a POS system, deals are compared to inventory a reduction in real-time, flagging any implicit enterprises over loss through waste or theft. It also makes copping a breath because you fete exactly what you have available and what you would like to buy to replenish your stock.

With an electronic digital inventory system accessible via a wireless tablet or smartphone, time spent taking inventory is drastically reduced. A director concerned about whether a named item is grazed rightly can check that count from the device and exclude wasted time spent running to the stock room.

The dated practice of transferring written data to a computer spreadsheet also disappears, as entry into the tablet allows that data to have participated throughout your eatery's entire network. That is especially helpful for multi-unit drivers. Inventory software also guides you thru the tactic in the quickest manner, completing the task briefly in order and leaving you longer to offer some study to staff and guests.

When compared to a handwritten inventory, the convenience of knowledge entry on a tablet is far simpler and further dependable. Crimes or misconstructions made from poor handwriting canceled entries, or mesentery is greatly reduced with a

wireless tablet. Corrections are fluently made and streamlined incontinently to your entire system. In addition, when all counts are accurate, it is easy to buy the right quantum of food, minimizing waste and maximizing gains.

Community Suite digital eatery inventory software simplifies the handling of any eatery through a simple-to-use, easy-to-train system that updates automatically and in real-time to a back-office system and pall database. The time-consuming donkeywork of handwritten inventory management is never excluded, making your eatery run simply, easily, and profitably.

Overview

Through my Inventory-management system, stores can manage their resources, especially their Product information and price, etc. in offline. This invention system will help small stores and local shops. This is a password-protected inventory management system I use Python and thinker Package for this inventory management system. The system is very easy to use for anyone because this is very simple and anybody can modify it easily because this inventory management system source code is written in Python language.

Python could even be a very easy programming language for anyone wanting to get into the field. It is used across fields, including web development, machine learning, data analysis, server management, software building, and more. It is incredibly versatile and thought of easy to hunt out, making it a powerful fit for beginners because its simple syntax is closer to natural language.

Many developers see Python because the foremost useful programing language because it is often used for a variety of tasks across many fields. Python could even be a top choice for processing, which is the gathering and analysis of data to find patterns within. Many popular web and software applications are written in Python, so it is likely that you will encounter it at some stage in your software development career. Learning Python can even be helpful for people that do not work in programming, as it is a well-liked language for automating repetitive tasks.

Many data-focused career paths focus heavily on Python scripting, and it is an important skill for anyone wanting to work as a data analyst or back-end web developer.

If you are inquisitive about learning more about becoming a data analyst, our guide to becoming a knowledge analyst with no experience can help. This line of labor could even be an excellent fit for anyone with a passion for math and strong analytical skills, and our overview of knowledge analyst career paths can help you decide if the field is right for you.

I use python3 in this inventory management system. Python 3 is almost ten years old, yet numerous companies still depend upon Python 2. Also, despite Python 2 support ending doubly agony, numerous still use the language. Both still have their place within the world moment. So, what are the vital differences? Python 2 was launched in 2000; Python 3 was launched in 2008. Python 2 considers the "print" keyword as a statement; Python 3 considers "print" as a function. Python 2 stores strings by ASCII; Python 3 uses Unicode. Python 2 features a more complex syntax than Python 3. Numerous Python 2 libraries are not forward compatible; numerous libraries simply use Python 3. Python discontinued Python 2 support in January 2020; Python 3 remains the foremost popular choice. So numerous companies still calculate on Python 2 — fourteen times after the foreword of Python 3 — because transferring canons between Python 2vs. 3 could indeed be a plenitude of trouble. It could take time. It took Dropbox 3 times to resettle, despite Guido Van Rossum working for them. Therefore, while it is sensible to use Python 3, an edge of familiarity with Python 2 still has its advantages. As an illustration, if your company still uses outdated laws or remains within the process of migrating, some knowledge is effective. Still, Python 2 has primarily lost the interest of numerous inventors. Python 3 is that the newer, easy-to-use, safest, and more important choice. Since Python discontinued Python 2 support, Python 3 is that the no-brainer for new inventors. Starting might be inviting — beyond the Python 2 or 3 debate, the foremost up-to-date language is that the safest choice. In addition, Developers will value Python 3 experience over Python 2. While it is voguish to verse yourself in multiple languages, Python 3 is that the most extensively used and more likely to be salutary for unborn updating.

For creating the graphic user interfaces (GUI) of this inventory management system, I use the Python thinker package.

Python features many GUI frameworks, but Tkinter is the sole framework that is built into the Python standard library. Tkinter has several strengths. It is cross-platform, therefore the identical code works on Windows, macOS, and Linux.

Visual elements are rendered using native OS elements, so applications built with Tkinter appear as if they belong on the platform where they are run.

Although Tkinter is taken under consideration as the de facto Python GUI framework, it is not without criticism. One notable criticism is that GUIs built with Tkinter look outdated. If you would like a shiny, modern interface, then Tkinter will not be what you are trying to seek out.

However, Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where an up-thus-far sheen is unnecessary, and thus the very best priority is to quickly build something functional and cross-platform.

This framework provides Python users with an easy way to create GUI elements using the widgets found in the Tk toolkit. Tk widgets are often used to construct buttons, menus, data fields, etc. during a Python application. Once created, these graphical elements are often associated with or interact with features, functionality, methods, data, or maybe other widgets.

For example, a button widget can accept mouse clicks and may be programmed to perform some kind of action, like exiting the application.

Some definitions

Window

This term has different meanings in several contexts, but generally, it refers to an oblong area somewhere on the user's display screen.

Top-level window

A window that acts as a toddler of the primary window. It will be decorated with the standard frame and controls for the desktop manager. It is often moved around the desktop and can usually be resized.

Widget

The generic term for any of the building blocks that structure an application in a graphical user interface.

Core widgets: The containers: frame, label frame, top level, paned window. The buttons: button, radio button, check button (checkbox), and menu button. The text widgets: label, message, text. The entry widgets: scale, scrollbar, list box, slider, spin box, entry (single line), option menu, text (multiline), and canvas (vector and pixel graphics).

Tkinter provides three modules that allow pop-up dialogs to be displayed: “tk.messagebox” (confirmation, information, warning, and error dialogs), tk.file dialog (single file, multiple files, and directory selection dialogs), and “tk.color” chooser (color picker).

Python 2.7 and Python 3.1 incorporate the "themed Tk" ("ttk") functionality of Tk 8.5 This allows Tk widgets to be easily themed to look like the native desktop environment in which the application is running, thereby addressing a long-standing criticism of Tk (and hence of Tkinter). Some widgets are exclusive to “ttk”, such as the combo box, progress bar, “treeview”, notebook, separator, and size grip.

Frame

In Tkinter, the Frame contrivance is the introductory unit of association for complex layouts. A frame may be a blockish area that can contain other contraptions.

Child and parent

When any contrivance is made, a parent-child relationship is made. For case, if you place a textbook marker inside a frame, the frame is that the parent of the marker.

For database support, I use the sqlite3 database in this inventory management system Project.

SQLite was designed to permit the program to be operated without installing a database management system or taking a database director. Unlike customer-garcon management systems, the SQLite machine has no standalone processes with which the appliance program communicates. rather, a linker integrates the SQLite library — statically or stoutly — into an management that uses SQLite's functionality through simple function calls, reducing quiescence in database operations; for straightforward queries with little concurrency, SQLite performance gains from avoiding the outflow of inter-process communication.

Due to the server less design, SQLite operations bear lower configuration than the customer- garcon databases. SQLite is understood as zero-conf because it does not bear service management (similar to incipiency scripts) or access control grounded on entitlement and watchwords. Access control is handled by means of train-system warrants given to the database train itself. Databases in customer-garcon systems use train-system warrants that give access to the database files only to the daemon process, which handles its cinches internally, allowing concurrent writes from several processes.

SQLite stores the entire database (delineations, tables, indicators, and therefore the word itself) as a single cross-platform train on a host machine, allowing several processes or vestments to pierce the identical database coincidently. It implements this easy design by locking the database train during jotting. Write access may fail with a mistake law, or it is frequently retried until a configurable downtime expires. SQLite read operations are frequently multitasked, however, due to the server less design; writes can only be performed succession ally. This concurrent access restriction does not apply to temporary tables, and it is relaxed in interpretation3.7 as write-ahead logging (WAL) enables concurrent reads and writes. Since SQLite possesses to depend on train-system cinches, it is not the favored choice for write-ferocious deployments.

SQLite uses PostgreSQL as a reference platform." What would PostgreSQL do" is employed to make sense of the SQL standard. One major divagation is that piecemeal from primary keys, SQLite does not apply type checking; the type of a value is dynamic and not rigorously constrained by the schema (although the schema will spark a conversion when storing if such a conversion is potentially reversible). SQLite strives to follow Pastel’s rule.

SQLite is a C library that provides a lightweight disk-based database that does not require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some operations can use SQLite for internal data storehouse. It is also possible to prototype an management using SQLite and harborage the law to a larger database similar to PostgreSQL or Oracle.

Gerhard Haring wrote the sqlite3 module. It provides an SQL interface biddable with the DB- API2.0 specification described by vim 249 and requires SQLite3.7.15 or newer.

SQLite tools are the utmost of the SQL- 92 standards for SQL but lack some features. As an illustration, it only incompletely provides triggers and cannot write

to views (still, it provides rather than triggers that give this functionality). Its support of ALTER TABLE statements is confined.

SQLite uses an unusual type system for a SQL-compatible DBMS rather than assigning a type to a column as in utmost SQL database systems, types are assigned to individual values; in language terms, it's stoutly compartmented. In addition, it has weakly compartmented in a number of the same ways that in Perl 1 can fit a string into an integer column (although SQLite will try to convert the string to an integer first if the columns preferred type is an integer). This adds inflexibility to columns, especially when sure to a stoutly compartmented scripting language. Still, the fashion is not movable to other SQL products. a typical review is that SQLite's type system lacks the word integrity medium handed by statically compartmented columns, although it's frequently emulated with constraints like CHECK(typeof(x) = ' integer'). Strict tables were added in interpretation3.37.1.

Tables typically include a retired rowid indicator column, which provides brisk access. However, SQLite will generally optimize it by treating it as an alias for rowid, causing the contents to be stored as a rigorously compartmented 64- bit inked integer and changing its geste to be kindly like a bus- incrementing column, If a database includes an Integer Primary crucial column. Unborn performances of SQLite may include a command to introspect whether a column has the geste like that of rowid to separate these columns from weakly compartmented, non-auto incrementing Integer Primary Keys. (Failed verification)Version3.6.19 released on October 14, 2009, added support for foreign crucial constraints.

Full support for Unicode case transformations is generally enabled through a voluntary extension. SQLite interpretation3.7.4 first saw the addition of the FTS4 (full-textbook hunt) module, which features advancements over the aged FTS3 module. FTS4 allows druggies to perform full-textbook quests on documents nearly like how hunt machines search spots. Version3.8.2 added support for creating tables without rowid, which can give space and performance advancements. Common table expressions support was added to SQLite in interpretation and added a further ultramodern hunt module called FTS5, the more radical (compared to FTS4) changes taking a bump in interpretation.

In 2015, with the json1 extension and new subtype interfaces, SQLite interpretation3.9 introduced JSON content operation. As of interpretation3.33.0, the utmost supported database size is 281 TB.

Working process

In this chapter of this paper, we are going to discuss about the working process of my inventory management system. For a better understanding of this Inventory management system, first of all, we need to understand the working process of the programming language used here. And we also need to understand the working process of those Python packages that are what is used in this inventory management system. After all, of that, we will discuss in detail the working process of our Inventory management system software.

From the discussion of the overview chapter, we already know that I use python3 as the programming language of my inventory management system. Here is a description of the working process of python 3.

Python is a high-level programming language designed to do many tasks. It is based on the CPython interpreter, which translates the Python code into something the machine can read.

Python gives us the ability to use a lot of modules and packages with our code, which are standard libraries built-in with the interpreter.

Additionally, Python's syntax is pretty simple and easy to learn – often it seems that you are just writing a message to someone else. Just make sure you know the indentation rules.

We can compare Python with other interpreted programming languages such as Java, JavaScript, PHP, and others. However, you might be wondering – what is CPython?

The Python interpreter is called “CPython” and it is written in the C programming language. This is the default implementation for Python.

Actually, any translator starts with the source code analysis. Here the Python interpreter receives the source code and initializes some instructions to do the following things:

It follows the indentation rule and checks the Python syntax. Maybe there are some incorrect lines, so it will stop the program from executing to show the error message.

This phase is called lexical analysis, which means dividing the source code files into a list of tokens

In the following step, the interpreter will generate byte codes. Let us see how that works.

Byte Code Generation

Once the parser of the Python interpreter receives the tokens, it starts to manipulate the lexical tokens. It generates a big structure called the AST (Abstract Syntax Tree).

The interpreter converts this AST to byte code, which means machine language. In Python, the byte code can be saved in a file ending with the “.pyc” extension.

The Python Virtual Machine (PVM)

The Python interpreter initializes its runtime engine called PVM, which is the Python virtual machine.

The interpreter loads the machine language with the library modules and inputs it into the PVM. This converts the byte code into executable code such as 0 s and 1s (binary). And then it prints the results.

Note that if an error happens during the PVM process, the executor will terminate the operation immediately to display the error.

Python source code

Python source files use the ".py" extension and are called "modules." With a Python module hello.py, the easiest way to run it is with the shell command "python hello.py Alice" which calls the Python interpreter to execute the code in hello.py, passing it the command line argument "Alice". See the official doc’s page for all the different options you have when running Python from the command line.

Imports, Command-line arguments, and len()

The outermost statements in a Python file, or "module", do its one-time setup — those statements run from top to bottom the first time the module is imported somewhere, setting up its variables and functions. A Python module can be run directly — as above "python hello.py Bob" — or it can be imported and used by some other module. When a Python file is run directly, the special variable "__name__" is set to "__main__". Therefore, it's common to have the boilerplate if __name__ ==... shown above to call a main() function when the module is run directly, but not when the module is imported by some other module.

In a standard Python program, the list sys. argv contains the command-line arguments in the standard way with sys. argv[0] being the program itself, sys. argv is the first argument, and so on. If you know about argc or the number of arguments, you can simply request this value from Python with len(sys. argv), just like we did in the interactive interpreter code above when requesting the length of a string. In general, len() can tell you how long a string is, the number of elements in lists and tuples (another array-like data structure), and the number of key-value pairs in a dictionary.

User-defined Functions

The def keyword defines the function with its parameters within parentheses and its code indented. The first line of a function can be a documentation string ("docstring") that describes what the function does. The docstring can be a single-line or a multi-line description as in the example above. (Yes, those are "triple quotes," a feature unique to Python!) Variables defined in the function are local to that function, so the "result" in the above function is separate from a "result" variable in another function. The return statement can take an argument, in which case that is the value returned to the caller.

Indentation

One unusual Python feature is that the whitespace indentation of a piece of code affects its meaning. A logical block of statements such as the ones that make up a function should all have the same indentation, set in from the indentation of their parent function or "if" or whatever. If one of the lines in a group has a different indentation, it is flagged as a syntax error.

Python's use of whitespace feels a little strange at first, but it's logical and I found I got used to it very quickly. Avoid using TABs as they greatly complicate the indentation scheme (not to mention TABs may mean different things on different platforms). Set your editor to insert spaces instead of TABs for Python code.

A common question beginners ask is, "How many spaces should I indent?" According to the official Python style guide (PEP 8), you should indent with 4 spaces. (Fun fact: Google's internal style guideline dictates indenting by 2 spaces!)

More on Modules and their Namespaces

Suppose you have a module "binky.py" which contains a "def foo ()". The fully qualified name of that foo function is "binky. foo". In this way, various Python modules can name their functions and variables whatever they want, and the variable names will not conflict — module1.foo is different from module2.foo. In the Python vocabulary, we would say that binky, module1, and module2 each have their own "namespaces," which as you can guess are variable name-to-object bindings.

For example, we have the standard "sys" module that contains some standard system facilities, like the argv list, and exit() function. With the statement "import sys", you can then access the definitions in the sys module and make them available by their fully-qualified name, e.g. sys. exit (). (Yes, 'sys' has a namespace too!)

There is another import form that looks like this: "from sys import argv, exit". That makes argv and exit() available by their short names; however, we recommend the original form with the fully-qualified names because it's a lot easier to determine where a function or attribute came from.

There are many modules and packages, which are bundled with a standard installation of the Python interpreter, so you do not have to do anything extra to use them. These are collectively known as the "Python Standard Library." Commonly used modules/packages include:

sys — access to exit(), argv, stdin, stdout, ...

re — regular expressions

os — operating system interface, file system

In the overview chapter, we discuss that I used the Python Tkinter package for developing the graphical user interface (GUI) of my inventory management system. Here is a description of the working process of the Python Tkinter package.

Python has numerous GUI framings, but Tkinter is the incomparable framework that's erected into Python's usual library. Tkinter has several potencies. It is cross-platform, so the identical code works on Windows, macOS, and Linux. Visual elements are rendered applying native operating system elements, so applications made up with Tkinter appear as if they belong on the platform where they're run.

Although Tkinter is considered the de facto Python GUI frame, it isn't without review. One star review is that GUIs made up with Tkinter look outdated. However, a new-fashioned interface, also Tkinter may not be what you're seeming for If you demand a candescent.

still, Tkinter is feather light and fairly effortless to apply compared to other frameworks. This makes it a compelling choice for erecting GUI operations in Python, especially for applications where an ultramodern luster is gratuitous, and the top precedence is to snappily make commodity running and cross-platform.

Tkinter Programming

Tkinter is an unremarkable GUI library for Python. Python when associated with Tkinter provides a fast and easy way to produce GUI applications. Tkinter provides an important object-acquainted interface to the Tk GUI toolkit.

Creating a GUI application applying Tkinter is a simple task. All you want to do is perform the following way Import the Tkinter module.

produce the GUI application main window.

Add one or further of the below-mentioned widgets to the GUI operation.

Enter the main event loop to take action against each event touched off by the user.

Tkinter uses so-called event progressions for permitting the user to define which events, both special and common; he or she wants to bind to instructors. It's the first argument" event" of the binding system. The event sequence is given as a string, using the following syntax:

<modifier-type-detail>

The type field is the essential part of an event specifier, whereas the "modifier" and "detail" fields aren't obligatory and are overlooked in many cases. they're wont to provide additional information for the chosen "type". The event "type" describes the sort of event to be bound, e.g. actions like mouse clicks, key presses, or the widget got the input focus.

Event Description

<Button>

A mouse button is pressed with the mouse pointer over the widget. The detail part specifies which button, e.g. the event <Button-1>, the middle button by <Button-2>, and the rightmost mouse button by <Button-3> define the left mouse button.<Button-4> defines the scroll-up event on mice with wheel support and <Button-5> the scroll-down.

Still, Tkinter will automatically" snare" the mouse needle, If you press a mouse button over a widget and observe it pressed. farther mouse events like Motion and Release events will be transferred to the current widget, indeed if the mouse is moved outside the current widget. The current position, relative to the widget, of the mouse pointer is handed in the x and y members of the event object passed to the message. You can use Button Press rather than Button, or indeed leave it out fully and< 1> are all duplicates.

<Motion>

The mouse is moved with a mouse button being held down. To specify the left, middle, or right mouse button use <B1-Motion>, <B2-Motion>, and <B3-Motion> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y

<Button Release>

The event, if a button is released. To specify the left, middle, or right mouse button use <ButtonRelease-1>, <ButtonRelease-2>, and <ButtonRelease-3> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y

<Double-Button>

Similar to the Button event, see above, but the button is double-clicked instead of a single click. To specify the left, middle, or right mouse button use <Double-Button-1>, <Double-Button-2>, and <Double-Button-3> respectively.

You can use Double or Triple as prefixes. Note that if you bind to both a single click (<Button-1>) and a double click (<Double-Button-1>), both bindings will be called.

<Enter>

The mouse pointer entered the widget.

Attention: This does not mean that the user pressed the Enter key! <Return> is used for this purpose.

<Leave>

The mouse pointer left the widget.

<Focus In>

The keyboard focus was moved to this widget or a child of this widget.

<Focus Out>

The keyboard focus was moved from this widget to another widget.

<Return>

The user pressed the Enter key. You can bind to virtually all keys on the keyboard: The special keys are Cancel (the Break key), Backspace, Tab, Return(the Enter key), Shift_L (any Shift key), Control_L (any Control key), Alt_L (any Alt key), Pause, Caps Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, NumLock, and Scroll Lock.

<Key>

The user pressed any key. The key is provided in the char member of the event object passed to the callback (this is an empty string for special keys).

A

The user typed an "a" key. Most printable characters can be used as is. The exceptions are space (<space>) and less than (<less>). Note that one is a keyboard binding, while <1> is a button binding.

<Shift-Up>

The user pressed the Up arrow while holding the Shift key pressed. You can use prefixes like Alt, Shift, and Control.

<Configure>

The size of the widget changed. The new site is provided in the width and height attributes of the event object passed to the callback. On some platforms, it can mean that the location changed.

We already know from the overview chapter I used sqlite3 as the database package of my inventory management system. Here is a short description of the working process of the python sqlite3.

The sqlite3 module

The sqlite3 module provides a straightforward interface for interacting with SQLite databases. A connection object is created by applyingsqlite3.connect (); the connection must be closed at the end of the session with the. Close () command. While the connection is open, any relations with the database bear you to make a cursor object with the. cursor () command. The cursor is also ready to perform all kinds of operations with. execute (). It's also possible to connect to an SQLite database that resides rigorously in memory(and not in a train) by passing the special string" memory" intosqlite3.connect(). For illustration, sqlite3.connect (" memory"). A" memory" SQLite database will vanish as soon as your Python program exits. This might be accessible if you want a temporary sandbox to try commodity out in SQLite, and do not need to persist any data after your program exits.

Queries

One of the most common ways to interact with a database is by querying and reacquiring data grounded on some hunt parameters. Use a SELECT statement string. The query is returned as a single tuple or a tuple of tuples. Add a WHERE statement to sludge your results grounded on some parameter.

Accessing data kept in SQLite by applying Python and Pandas

Applying pandas, we can import the effects of a SQLite query into a data framework. Remark that you can apply the duplicate SQL commands syntax that we applied in the SQLite assignment.

Storing data:

Keeping your data in a SQLite database can give substantial performance advancements when reading/ jotting compared to CSV. The difference in performance becomes more conspicuous as the size of the dataset grows. We can also use pandas to produce new tables within a SQLite database. Then, we run a replay of an exercise we did ahead with CSV lines using our SQLite database.

We first read our check data, also elect only those check results for 2002, and also save it out to its own table so we can work with it on its own latterly.
Adding Data to the SQLite Database
Insert rows into a table in the SQLite database from a Python program using the sqlite3 module. To fit rows into a table in a SQLite database, you apply the following way
First, connect to the SQLite database by creating a Connection object.
Second, produce a Cursor object by calling the cursor system of the Connection object. Third, execute an INSERT statement. However, you use the question mark (?) as the placeholder for each argument,
If you want to pass arguments to the INSERT statement. To produce a new SQLite database from a Python program. When you connect to a SQLite database train that does not live, SQLite automatically creates a new database for you. To breed a database, first, you have to produce a Connection object that represents the database applying the connect() function of the sqlite3 module.

How to use the SQLite INSERT statement to insert new rows into a table.
To insert data into a table, you use the INSERT statement. SQLite provides various forms of INSERT statements that allow you to insert a single row, multiple rows, and default values into a table.
In addition, you can insert a row into a table using data provided by a SELECT statement.
SQLite INSERT – inserting a single row into a table

Modifying Data in the SQLite Database
The UPDATE statement in SQL is used to update the data of an existing table in the database. We can update single columns as well as multiple columns using the UPDATE statement as per our requirement.

Syntax:
DATE table_name SET column1 = value1, column2 = value2…
WHERE condition;

In the above syntax, the SET statement is used to set new values to the particular column, and the WHERE clause is used to select the rows for which the columns are needed to be updated.

As of now, the ‘SqliteDb_developers’ table contains six rows, so let’s update the salary of a developer whose id is 4. To perform SQLite UPDATE query from Python, you need to follow these simple steps:

Connect to MySQL from Python
Refer to Python SQLite database connection to connect to SQLite database from Python using the sqlite3 module.
Prepare a SQL Update Query
Prepare an update statement query with data to update. Mention the column name we want to update and its new value.
For example, UPDATE table_name SET column1 = value1, column2 = value2..., columnN = valueN WHERE [condition];
Execute the UPDATE query, using the cursor.execute()
This method executes the operation stored in the UPDATE query.
Commit your changes
After the successful execution of the SQLite update query, Don't forget to commit your changes to the database using the connection. comit().
Extract the number of rows affected
After a successful update operation, use a cursor. rowcount method to get the number of rows affected. The count depends on how many rows you are updating.
Verify the result using the SQL SELECT query
Execute an SQLite select query from Python to see the new changes
Close the cursor object and database connection object
Using the cursor. clsoe() and connection.clsoe() method to close SQLite connections once the update operation completes.

Reading Data
To query data in an SQLite database from Python, you use these steps:
First, establish a connection to the SQLite database by creating a Connection object.
Next, create a Cursor object using the cursor method of the Connection object.
Then, execute a SELECT statement.
After that, call the fetchall() method of the cursor object to fetch the data.
Finally, loop the cursor and process each row individually.

Delete data.
In order to delete data in the SQLite database from a Python program, you use the following steps:
First, establish a connection to the SQLite database by creating a Connection object using the connect() function.
Second, to execute a DELETE statement, you need to create a Cursor object using the cursor() method of the Connection object.

Third, execute the DELETE statement using the execute() method of the Cursor object. In case you want to pass the arguments to the statement, you use a question mark (?) for each argument

After having a depth brief of the using materials for my inventory management software system. Now we can discuss the working process of my inventory management system software.

After running the .py file of my inventory management system project. We will see a pop-up window come to our screen. The function of this window has been defined on our code in the root.

In this window, we have to button login and exit. If you want to use this inventory management system, you need to click on the login button unless you can click on the exit button.

After clicking the login button on the home on the first window, we will see another window will come up. This window is the user login window of my inventory management system. The function of this window has been defined on our code in the “def Login(event=None):” function.

```
def Login(event=None):
    global admin id
    Database()
    if USERNAME.get == "" or PASSWORD.get() == "":
        lbl_result.config(text="Please complete the required field!", fg="red")
    else:
        cursor.execute("SELECT * FROM `admin` WHERE `username` = ? AND `password` = ?",
                       (USERNAME.get(), PASSWORD.get()))
        if cursor.fetchone() is not None:
            cursor.execute("SELECT * FROM `admin` WHERE `username` = ? AND `password` = ?",
                           (USERNAME.get(), PASSWORD.get()))
            data = cursor.fetchone()
            admin_id = data[0]
            USERNAME.set("")
            PASSWORD.set("")
            lbl result.config(text="")
            #smain()
            #hm()
            ShowHome()
        else:
            lbl_result.config(text="Invalid username or password", fg="red")
            USERNAME.set("")
            PASSWORD.set("")
    cursor.close()
    conn.close()
```

In this function, the program asks the user to input a username and a password. When the user hit the login button, the program will check the username and password with our stored username and password. If the given password and

username match with the stored username and password then the program goes forward and another Windows will come on screen.

This window is the main window of our inventory management system. in this Windows, the screen in the login window will disappear. The function of disappearing that login window has been defined on our code on the “def smain():” function.

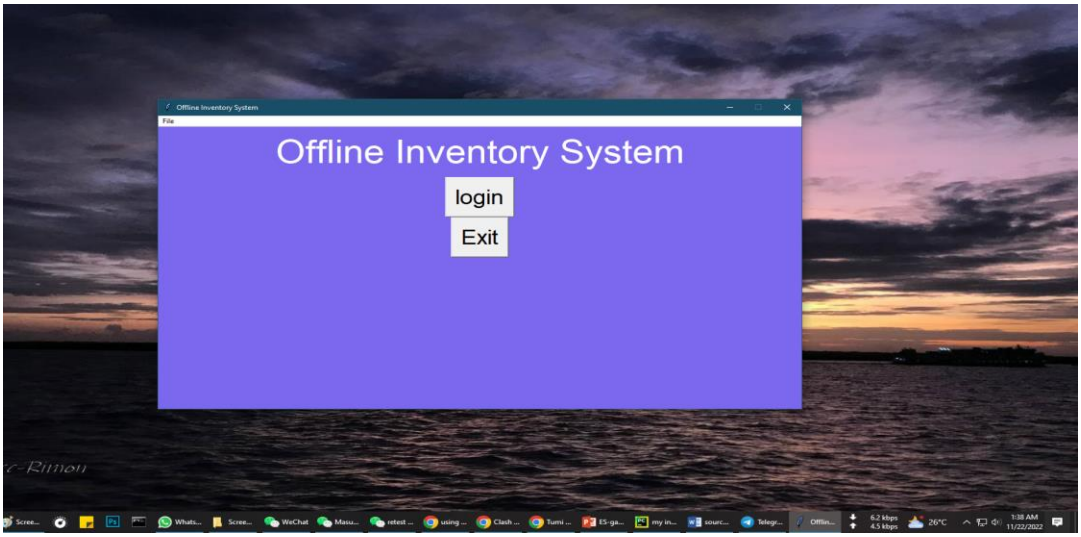
Now in the main window, we have three parts. In the upper lip part, we have boxes to get input for our user and in the lower part, we have buttons to do actions on the inputs of our user. And on the right side of this main window, we can see our stored data. These data are stored in our function bysqllite3.

To insert data in our database portion, we need to fill up the input boxes and click on the edit button. Now we can see our data on the right side of the window.

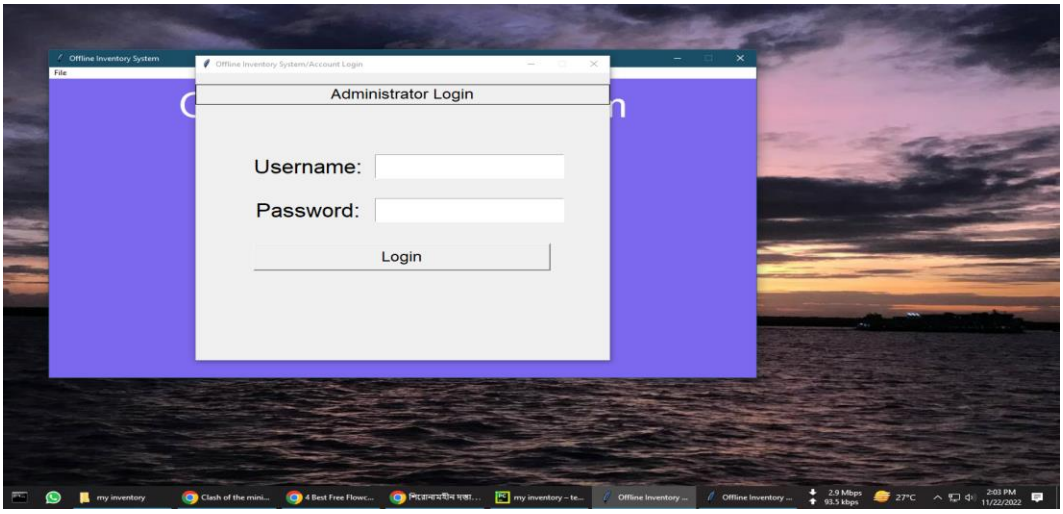
Using Process

For using my inventory management system you need to install Python 3 on your computer after installing Python 3 you should download some packages like thinker and SQL light or you can run the requirement.txt II in your system.

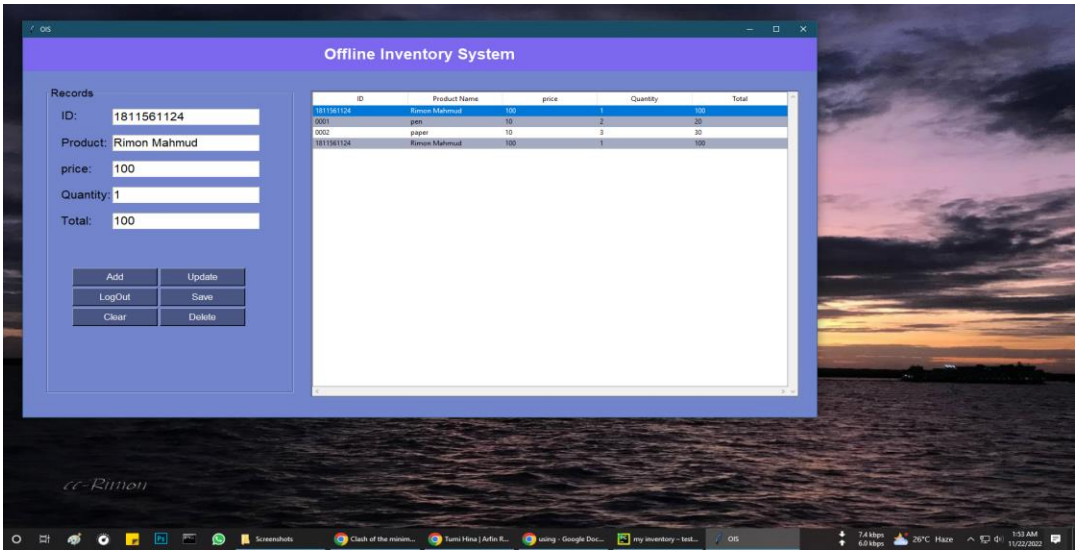
Next, you need to run the .py file on your computer. Now you can see a popup Windows have two buttons one login and another one exit. To login in you need to click the login button.



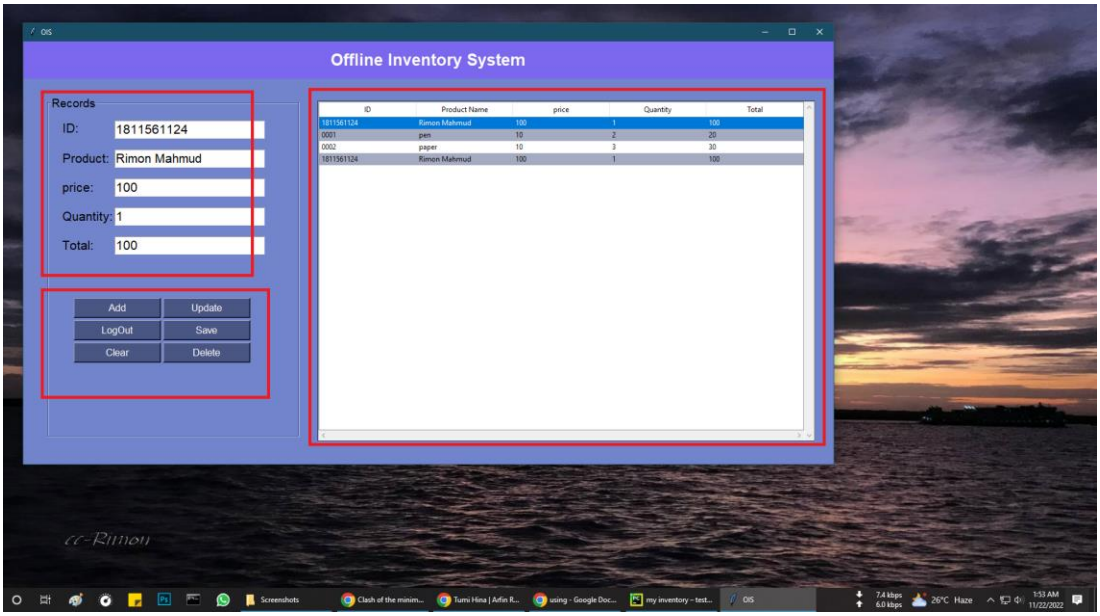
After pressing the login button, you will able to see the login form of my inventory management system. In this window, you need to put in a username and a password.



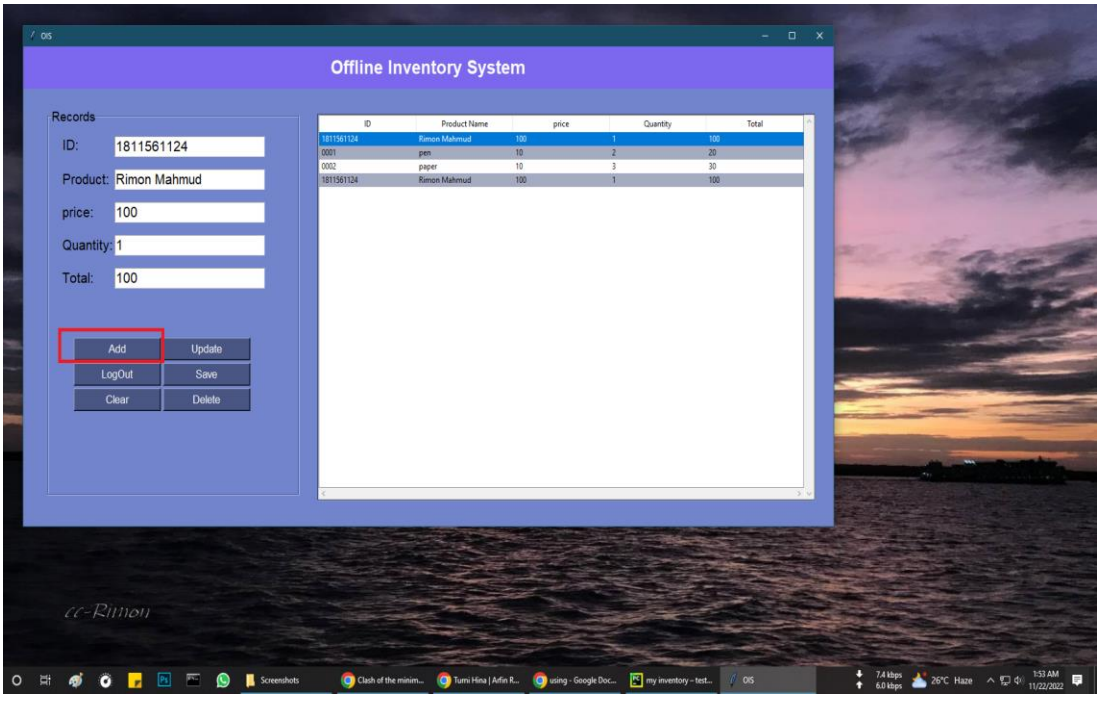
The default password and username of my inventory system are username= admin and password= admin. After filling in the username and password box, you have to click on the login button.



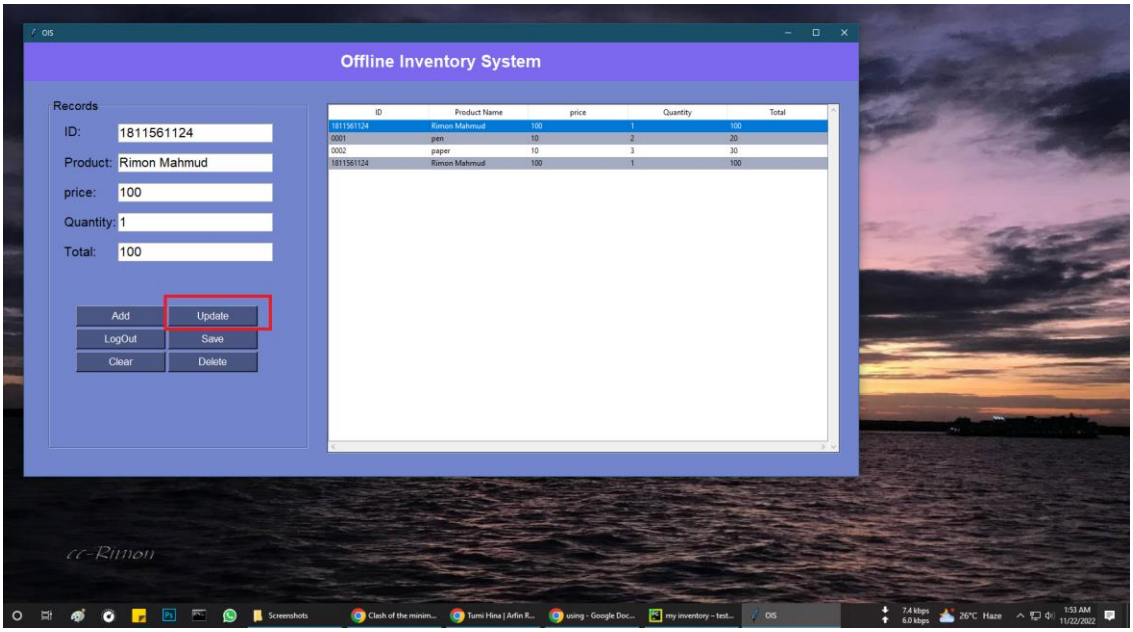
Now we are in the main window of the inventory management system. Here we have three parts. In the upper left part, we have boxes to get input for our user and in the lower part, we have buttons to do actions on the inputs of our user. And on the right side of this main window, we can see our stored data.



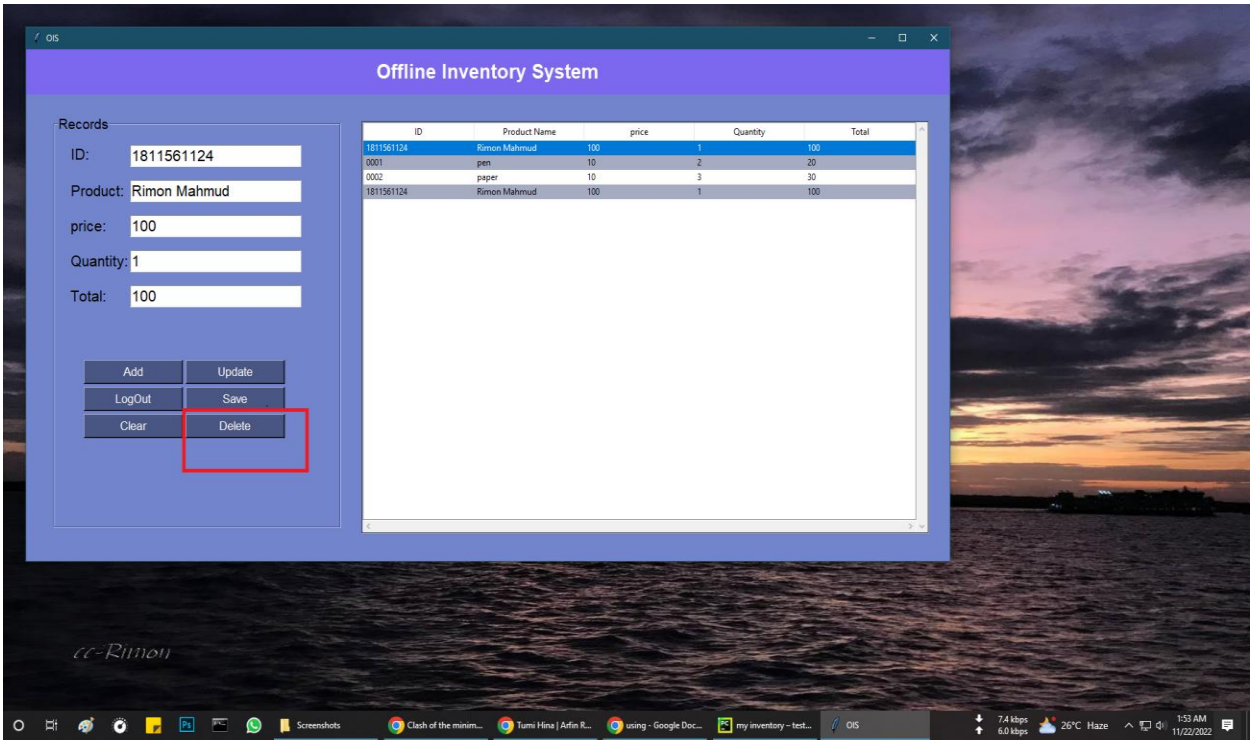
For adding data in our system we need to feel up the left side boxes of our Window and then click on the add button. Now you can see your data on the left side of the window.



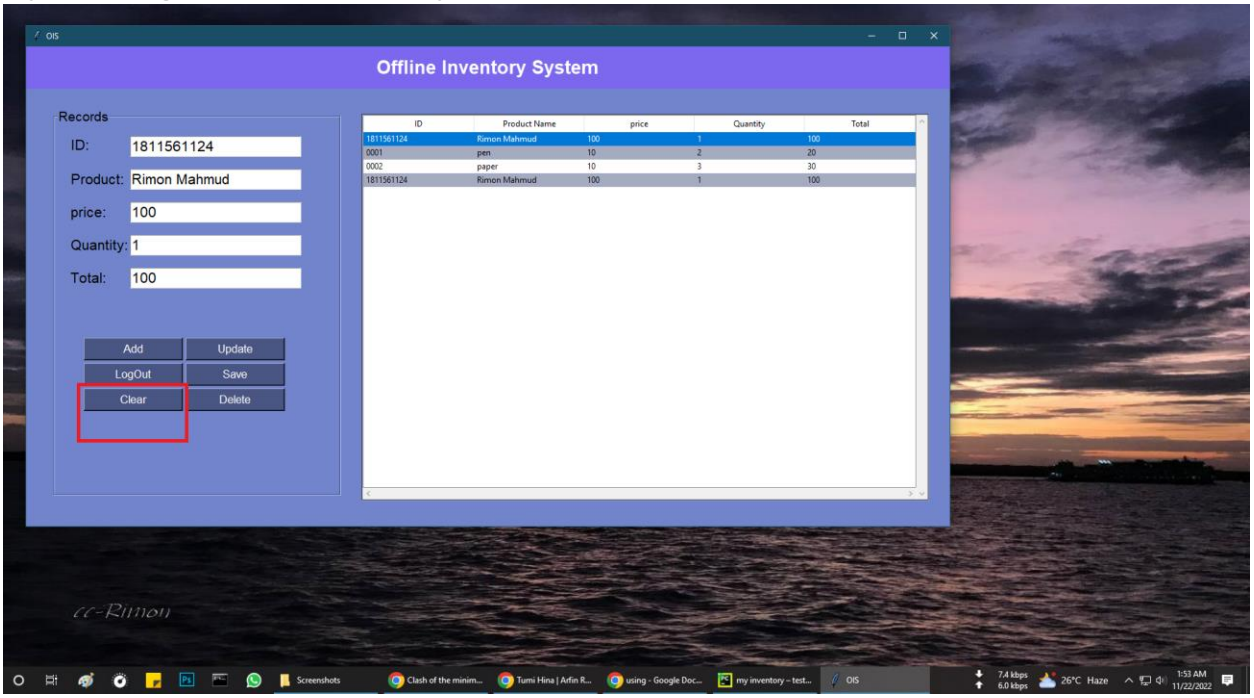
For an update on any entry, you have to select the row of that entry from the data box. Then you can edit the record field and save it.



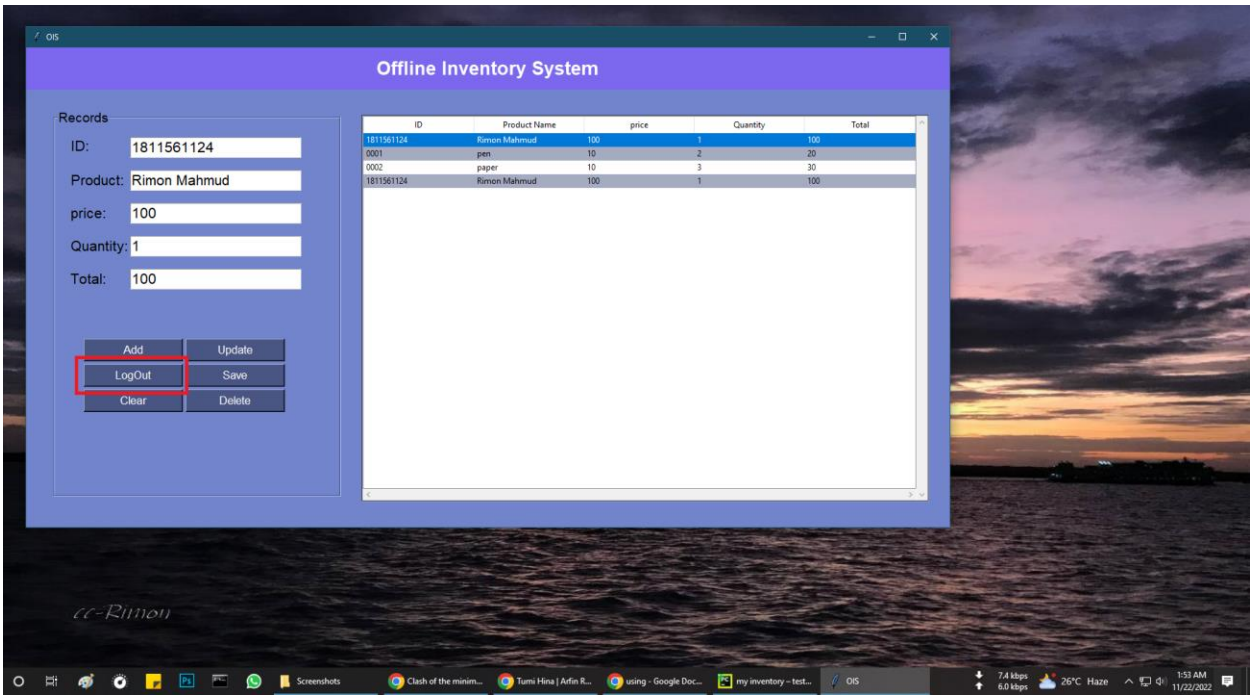
To delete, you need to select the row from the data box and then press the delete button.



By clicking the clear button, you can clear the whole data box at once.



When you click the logout button, the software will stop and shut down.



Conclusion

One of the most valuable assets of a company is its inventory. In colorful diligence, similar to retail, food services, and manufacturing, a lack of inventory can have mischievous goods. Away from being a liability, the inventory can also be considered a threat. It can be prone to theft, damage, and corruption. Having a large inventory can also lead to a reduction in deals.

Anyhow, of the size of your company, having a proper inventory management system is veritably important for any business. It can help you keep track of all your inventories and determine the exact prices. It can also help you manage unforeseen changes in demand without immolating client experience or product quality. This is especially important for brands looking to come to a further client-centric association.

Balancing the pitfalls of overstock and dearth is an especially grueling process for companies with complex inventory chains. A company's inventory is generally a current asset that it plans to vend within a time. It must be measured and counted regularly to be considered a current asset.

inventory management is a veritably complex but essential part of the inventory chain. An effective inventory management system helps to reduce stock-related costs similar to warehousing, carrying, and ordering costs. As you have read over, there are different ways that businesses can use to simplify and optimize stock management processes and control systems.

Products are being bought, packed, and returned across all channels — in-store, online, and through mobile management, packed from both stores and storages – in every combination possible, as consumers are forming advanced prospects and demands from their retailers.

These Omni channel sweats complicate inventory shadowing, as products can be moved to multiple locales during their selling lifecycle due to erratic returns, canceled orders, and shipping changes brought on by new shopping technology.

Therefore, accurate management of this inventory is critical and may bear newer technology to track duly and produce a positive client experience. Inventory management systems are precious tools in helping to give a clear view of all stock, allowing retailers to more handle effectively such as inventory allocation, redundant inventory, and other stock-related processes. Then, we will take you

through 6 major reasons why streamlining your inventory management system is worth your time and plutocrat.

Benefits of an inventory management system

Trusting the report system

Having stock Situations constantly up to date builds trust in the inventory management software by those who use it. This leads to better task performance, as you and your workers can be certain that the offers you are putting together or the purchases you are approving will be accurate and salutary for the company.

In turn, you will all use the inventory management technology more, performing better data collection that can also be used for unborn business opinions and platoon collaboration strategies.

Satisfying your guest's buyers

Inventory management systems mean a clearer view of reporting. Better, more accurate reporting leads to better fulfillment and lower or over-shipping and selling. You can also vend guests what they want because you are certain that the stock you see is the stock that is in your storehouse. This helps you vend and transport what you have, leading to smaller unfulfilled orders and bummers.

Detecting problems sooner

inventory management systems give a real-time view of stock situations, allowing retailers to cover and incontinently reply if situations ever go amiss, indicating that the commodity is wrong. Being suitable to spot problems sooner, and thereby make changes briskly, eventually saves the company plutocrat by limiting the losses from an unnoticed issue.

Investing your cash better

Having a real-time view of inventory situations with full confidence in the delicacy of the figures allows you to be suitable to know before what products aren't dealing and will remain leftover for out-price deals. This allows for better-planned deals. Also, these call-outs for poor-performing products allow you to make changes to your orders sooner and alter noncommercial buying strategies going forward. Eventually, good inventory management allows for smarter business opinions and better cash inflow.

Missing Smaller deals openings

Knowing exactly what you have allows you to more feed your off-price buyers and gives you the capability to identify specific buyers by their purchasing history and habits. This also enables you to reach out to a specific buyer with specific product assortments provisioned to that buyer. In-stock product identification and buyer product preference lead to better-provisioned offers and therefore an increased chance of a trade that would else have not passed.

Improving profit perimeters with advanced returns

Accurate stock Situations modernize automatically with new inventory management system technology. This gives you access to dealing with the product still in stock as soon as possible, leading to lower plutocrats lost on storehouse space. Also, the hastily the product is vended, the more advanced the price it can be vended since the product loses value every day it sits in the storehouse. An advanced selling price leads to advanced financial returns on that product.

Disadvantages of the inventory management system

Bureaucracy: indeed though inventory management allows workers at every position of the company to read and manipulate company stock and product inventory, the structure needed to make such a system adds a sub caste of bureaucracy to the whole process and the business in general. In cases where inventory control is in-house, this includes the number of new hires that are not present to regulate the storehouse and grease deals. In cases where the inventory management is in the hands of a third party, the cost is a subscription price and a dependence on another separate company to manage its structure. No matter the choice you go for, it translates to an advanced overhead cost and further layers of management between the proprietor and the client. From the standpoint of the client, a problem that requires elderly management to handle will take a longer period of time before it'll be trashed out.

Impersonal touch: another disadvantage of inventory management is a lack of particular touch. Large inventory chain management systems make products more accessible across the globe and utmost give client service support in case of difficulty, but the increase in structure can frequently mean a drop in the particular touch that helps a company to stand out above the rest. For case, the deals director of a small manufacturing company that sells plumbing inventories to original plumbers can throw in a redundant box of washers or elbows at no charge to the client without raising any admonitions. This is done for the sake of client relations and frequently makes the client feel like he is special. While free accouterments

can also be handed under inventory operation, processing time and paperwork make carrying the material feel more like a chore for the client or indeed an annuity.

Product problem: indeed though inventory management can reveal to you the quantum of stock you have at hand and the quantum that you have fended off, it can also hide product problems that could lead to client service disasters. Since the management places, nearly all of its focus on inventory management to the detriment of quality control, broken or incorrect particulars that would typically be discarded are packed along with wholesome particulars.

Increased space is demanded to hold the inventory: in order to hold inventory, you will need to have space so unless the goods you deal in are really small in size, also you'll need a storehouse to store them. In addition, you will also need to buy shelves and racks to store your goods, forklifts to move around the stock, and of course staff. The optimum position of inventory for a business could still be a lot of goods and they will need space to be stored in and in some cases fresh functional costs to manage the inventory. This will in turn increase costs and influence negatively the quantum of profit the business makes.

Complexity: some styles and strategies of inventory management can be fairly complex and delicate to understand on the part of the staff. This may affect the need for workers to suffer training in order to grasp how the system works.

Some inventory management systems similar to the fixed order period system impel a periodic review of all particulars. This itself makes the system a bit hamstrung.

High perpetration costs: some inventory management systems can come at a high price because the business needs to install technical systems and software in order to use them. This can be problematic for large businesses, which operate in delicate locales. Indeed, after installing the expensive system, it still needs to be maintained and upgraded on a regular base, therefore incurring further costs.

Indeed, with an effective inventory management system, you can control but not exclude business threats.

The control of inventory is complex because of the numerous functions it performs. It should therefore be viewed as a participating responsibility.

Holding inventory can affect a lesser threat of loss to devaluation (changes in price).

In this concluding chapter, I also want to talk something about the used packages in my project.

Lift the hood on nearly any business operation, and you will reveal some way to store and use structured data. Whether it is a customer-side app, an app with a web front-end, or an edge-device app, chances are it needs a bedded database of some kind. SQLite is an embeddable open-source database, written in C and query able with conventional SQL, designed to cover those use cases further. SQLite is designed to be presto, movable, and dependable, whether you are storing only kilobytes of data or multi-gigabyte blobs.

One of SQLite's topmost advantages is that it can run nearly anywhere. SQLite has been ported to a wide variety of platforms Windows, macOS, Linux, iOS, Android, and more. Windows druggies in particular can use precompiled binaries for regular Win32, UWP, WinRT, and. Net. Whatever the deployment target is for your app, odds are there is an edition of SQLite available for it or a way to harborage the C source law to that target. Apps that use SQLite do not have to be written in any particular language, as long as there is some way to bind and work with external libraries written inC. SQLite's binaries are tone-contained, so they bear no particular magic to emplace — they can simply be dropped into the same directory as the operation. Numerous languages have high-position tapes for SQLite as a library and can use that in confluence with other database access layers for the language. Python, for case, packets the SQLite library as a standard-issue element with the stock interpretation of the Python practitioner. In addition, third parties have written a wide variety of ORMs and data layers that use SQLite, so you are not wedged penetrating SQLite via raw SQL strings (which isn't only clumsy but also potentially dangerous). Eventually, the source law for SQLite is the public sphere so that it can be reused in other programs with no practical restrictions.

SQLite advantages.

The most common and egregious use case for SQLite is serving as a conventional, table-acquainted relational database. SQLite supports deals and infinitesimal actions, so a program crash or indeed a power outage will not leave you with a spoiled database. SQLite has features set up in advanced-end databases similar to full-textbook indexing and support for JSON data. Operation data generally stuffed into semi-structured formats like YAML or XML can be stored as SQLite tables, allowing the data to be penetrated more fluently and reused more snappily. SQLite also provides a fast and important way to store configuration data for a program. Rather than parsing a training format like YAML, an inventor can use SQLite as an interface to those lines — frequently far more briskly than operating on them manually. SQLite can work with in-memory data or external lines (e.g., CSV lines) as if they were native database tables, furnishing a handy way to query that data. Because SQLite is a single standalone binary, it is easy to emplace with an app and move it with the app as demanded. Each database created by SQLite also comprises a single train, which can be compacted or optimized by way of SQL commands. Third-party double extensions for SQLite add indeed more functionality. SQLCipher adds 256- bit AES encryption to SQLite database lines. Another, SQLite- Bloom filter, allows you to produce bloom pollutants from data in a given field. Numerous other third-party systems give fresh driving for SQLite, similar to the Visual Studio Code extension that allows browsing databases from within Visual Studio Code or the LiteCLI interactive command line for SQLite. A curated list of SQLite coffers on GitHub includes numerous further.

Why I used Tkinter

Python Tkinter is the most favored package used for creating nice GUIs for operations as it has a variety of styles like pack(), grid(), and place() for figure operation. It has standard attributed confines, sources, colors, cursors, anchors, and bitmaps for a better GUI. In addition, it has a vast array of contraptions to choose from and is by far the easiest to use. The combination of all these features makes Python Tkinter makes it veritably popular among Python inventors and makes it a good tool to use.

Concentrated approach

The layered approach used in designing Tkinter gives Tkinter all of the advantages of the TK library. Thus, at the time of creation, Tkinter inherited the benefits of a GUI toolkit that had been given time to develop. This makes the early performances of Tkinter a lot more stable and dependable than if they had been

rewritten from scrape. In addition, the conversion from Tcl/ Tk to Tkinter is really trivial, so Tk programmers can learn to use Tkinter veritably and fluently.

Accessibility

Tkinter is veritably intuitive, and thus quick and effortless. The Tkinter perpetration hides the detailed and complicated calls in simple, intuitive styles. This is a durability of the Python way of allowing since the language excels at snappily erecting prototypes. It's thus anticipated that its favored GUI library be enforced using the same approach. For example, here is the code for a typical "Hello world"-like application:

```
from Tkinter import *
root = Tk( )
root.title("A simple application")
root.mainloop( )
```

The first 2 lines allow for the creation of a complete window. Compared to MFC programming, it makes no doubt that Tkinter is simple to use. The third line sets the caption of the window and the fourth one makes it enter its event loop.

Portability

Python scripts that apply Tkinter do not bear variations to be ported from one platform to the other. Tkinter is available for any platform that Python is enforced for, videlicet Microsoft Windows, X Windows, and Macintosh. This gives it a great vantage over the utmost contending libraries, which are frequently confined to one or two platforms. In addition, Tkinter will give the native look and sense of the specific platform it runs on.

Availability

Tkinter is now included in any Python distribution. Thus, no supplementary modules are needed in order to run scripts using Tkinter

In conclusion, every modern system and application has some casualties and advantages. The inventory management system also has some casualties and some advantages but in this era of the Information Technology revolution, we cannot

ignore the digitalization of things. On the other hand, this digitalization helps us how to make easy e our daily activities.
About my inventory management system is very light and very easy to use the system it doesn't take too many resources from the computer but it helps a lot to manage a business it is very easy to update it is very easy updatable because of the used programming language and its packages.

Source Code

```
from tkinter import *
import tkinter.messagebox as tkMessageBox
import sqlite3

import tkinter as tk
from tkinter import Button, StringVar, ttk

root = Tk()
root.title("Offline Inventory System")

width = 1024
height = 520
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
root.geometry("%dx%d+%d+%d" % (width, height, x, y))
root.resizable(0, 0)
root.config(bg="mediumslateblue")

#=======================================VARIABLES=====================================

USERNAME = StringVar()
PASSWORD = StringVar()

#=====================================METHODS=========================================

def Database():
    global conn, cursor
    conn = sqlite3.connect("pythontut.db")
    cursor = conn.cursor()
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS `admin` (admin_id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, username TEXT, password TEXT)")
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS `product` (product_id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, product_name TEXT, product_qty TEXT, product_price TEXT)")
    cursor.execute("SELECT * FROM `admin` WHERE `username` = 'admin' AND `password` =
'admin'")
    if cursor.fetchone() is None:
        cursor.execute("INSERT INTO `admin` (username, password) VALUES('admin',
'admin')")
        conn.commit()
```

```
def Exit():
    result = tkMessageBox.askquestion('Offline Inventory System', 'Are you sure you
want to exit?', icon="warning")
    if result == 'yes':
        root.destroy()
        exit()

def ShowLoginForm():
    global loginform
    loginform = Toplevel()
    loginform.title("Offline Inventory System/Account Login")
    width = 600
    height = 500
    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    loginform.resizable(0, 0)
    loginform.geometry("%dx%d+%d+%d" % (width, height, x, y))
    LoginForm()


def LoginForm():
    global lbl_result
    TopLoginForm = Frame(loginform, width=600, height=100, bd=1, relief=SOLID)
    TopLoginForm.pack(side=TOP, pady=20)
    lbl_text = Label(TopLoginForm, text="Administrator Login", font=('arial', 18),
width=600)
    lbl_text.pack(fill=X)
    MidLoginForm = Frame(loginform, width=600)
    MidLoginForm.pack(side=TOP, pady=50)
    lbl_username = Label(MidLoginForm, text="Username:", font=('arial', 25), bd=18)
    lbl_username.grid(row=0)
    lbl_password = Label(MidLoginForm, text="Password:", font=('arial', 25), bd=18)
    lbl_password.grid(row=1)
    lbl_result = Label(MidLoginForm, text="", font=('arial', 18))
    lbl_result.grid(row=3, columnspan=2)
    username = Entry(MidLoginForm, textvariable=USERNAME, font=('arial', 25),
width=15)
    username.grid(row=0, column=1)
    password = Entry(MidLoginForm, textvariable=PASSWORD, font=('arial', 25),
width=15, show="*")
    password.grid(row=1, column=1)
    btn_login = Button(MidLoginForm, text="Login", font=('arial', 18), width=30,
command=Login)
    btn_login.grid(row=2, columnspan=2, pady=20)
    btn_login.bind('<Return>', Login)


def Login(event=None):
    global admin_id
    Database()
    if USERNAME.get == "" or PASSWORD.get() == "":
        lbl_result.config(text="Please complete the required field!", fg="red")
    else:
        cursor.execute("SELECT * FROM `admin` WHERE `username` = ? AND `password` =
?",
                       (USERNAME.get(), PASSWORD.get()))
        if cursor.fetchone() is not None:
            cursor.execute("SELECT * FROM `admin` WHERE `username` = ? AND `password`
= ?",
                           (USERNAME.get(), PASSWORD.get()))
            data = cursor.fetchone()
```

```
            admin_id = data[0]
            USERNAME.set("")
            PASSWORD.set("")
            lbl_result.config(text="")
            #smain()
            #hm()
            ShowHome()
        else:
            lbl_result.config(text="Invalid username or password", fg="red")
            USERNAME.set("")
            PASSWORD.set("")
    cursor.close()
    conn.close()



#========================================From s main ================================


def ShowHome():
    root.withdraw()
    smain()



def smain():

    loginform.destroy()

    win = tk.Tk()
    win.geometry("1350x700+0+0")
    win.title("OIS")

    # ---------Background Color
    win.config(bg="#7285CC")

    #--------- Adding some style
    style = ttk.Style()

    # --------Pick a theme
    style.theme_use("default")

    style.configure("Treeview",
                    background="white",
                    foreground="black",
                    rowheight=25,
                    fieldbackground="white"
                    )

    #--------- Change selected color
    style.map(
        "Treeview",
        background=[("selected", "darkred")]
    )

    #-------- Top Menu

    title_label = tk.Label(
        win,
        text="Offline Inventory System",
        font=("Arial", 20, "bold"),
        padx=15,
```

```
        pady=15,
        border=0,
        relief=tk.GROOVE,
        bg="mediumslateblue",
        foreground="white"
    )
    title_label.pack(side=tk.TOP, fill=tk.X)

    #------ Left Menu

    detail_frame = tk.LabelFrame(
        win, text="Records",
        font=("Arial", 14),
        bg="#7285CC",
        foreground="black",
        relief=tk.GROOVE
    )
    detail_frame.place(x=40, y=90, width=420, height=565)

    #--------- Data Frame

    data_frame = tk.Frame(
        win,
        bg="#495582",
        relief=tk.GROOVE
    )
    data_frame.place(x=490, y=98, width=830, height=565)

    #--------- Label with Entry

    id_lab = tk.Label(
        detail_frame,
        text="ID:",
        font=("Arial", 16),
        bg="#7285CC",
        foreground="black"
    )
    id_lab.place(x=20, y=15)

    id_ent = tk.Entry(
        detail_frame,
        bd=1,
        font=("arial", 16),
        bg="white",
        foreground="black",
    )
    id_ent.place(x=110, y=17, width=250, height=30)

    # 2
    Productname_lab = tk.Label(
        detail_frame,
        text="Product:",
        font=("Arial", 16),
        bg="#7285CC",
        foreground="black"
    )
    Productname_lab.place(x=20, y=65)

    Productname_ent = tk.Entry(
        detail_frame,
        bd=1,
        font=("arial", 16),
        bg="white",
```

```
        foreground="black",
    )
    Productname_ent.place(x=110, y=65, width=250, height=30)


    # 4
    price_lab = tk.Label(
        detail_frame,
        text="price:",
        font=("Arial", 16),
        bg="#7285CC",
        foreground="black"
    )
    price_lab.place(x=20, y=113)

    price_ent = tk.Entry(
        detail_frame,
        bd=1,
        font=("arial", 16),
        bg="white",
        foreground="black",
    )
    price_ent.place(x=110, y=113, width=250, height=30)

    # 5
    quantity_lab = tk.Label(
        detail_frame,
        text="Quantity:",
        font=("Arial", 16),
        bg="#7285CC",
        foreground="black"
    )
    quantity_lab.place(x=20, y=161)

    quantity_ent = tk.Entry(
        detail_frame,
        bd=1,
        font=("arial", 16),
        bg="white",
        foreground="black",
    )
    quantity_ent.place(x=110, y=161, width=250, height=30)



    # 7
    Total_lab = tk.Label(
        detail_frame,
        text="Total:",
        font=("Arial", 16),
        bg="#7285CC",
        foreground="black"
    )
    Total_lab.place(x=20, y=209)

    Total_ent = tk.Entry(
        detail_frame,
        bd=1,
        font=("arial", 16),
        bg="white",
        foreground="black",
    )
    Total_ent.place(x=110, y=209, width=250, height=30)
```

```
    # Database frame

    main_frame = tk.Frame(
        data_frame,
        bg="#495582",
        bd=2,
        relief=tk.GROOVE
    )
    main_frame.pack(fill=tk.BOTH, expand=True)

    y_scroll = tk.Scrollbar(main_frame, orient=tk.VERTICAL)
    x_scroll = tk.Scrollbar(main_frame, orient=tk.HORIZONTAL)

    # Treeview database

    inventory_table = ttk.Treeview(main_frame, columns=(
        "ID", "Product Name", "price", "Quantity", "Total",
    ), yscrollcommand=y_scroll.set, xscrollcommand=x_scroll.set)

    y_scroll.config(command=inventory_table.yview)
    x_scroll.config(command=inventory_table.xview)

    y_scroll.pack(side=tk.RIGHT, fill=tk.Y)
    x_scroll.pack(side=tk.BOTTOM, fill=tk.X)

    inventory_table.heading("ID", text="ID")
    inventory_table.heading("Product Name", text="Product Name")
    inventory_table.heading("price", text="price")
    inventory_table.heading("Quantity", text="Quantity")
    inventory_table.heading("Total", text="Total")


    inventory_table["show"] = "headings"

    inventory_table.column("ID", width=100)
    inventory_table.column("Product Name", width=100)
    inventory_table.column("price", width=100)
    inventory_table.column("Quantity", width=100)
    inventory_table.column("Total", width=100)
    inventory_table.pack(fill=tk.BOTH, expand=True)

    # Default data

    data = [

    ]

    # Create stripped row tags
    inventory_table.tag_configure("oddrow", background="white")
    inventory_table.tag_configure("evenrow", background="#A5ABC1")

    global count
    count = 0
    for record in data:
        if count % 2 == 0:
            inventory_table.insert(parent="", index="end", iid=count, text="",
values=(
            record[0], record[1], record[2], record[3], record[4],record[5],
record[6], record[7]), tags=("evenrow"))
        else:
```

```
            inventory_table.insert(parent="", index="end", iid=count, text="",
values=(
            record[0], record[1], record[2], record[3], record[4], record[5],
record[6], record[7]), tags=("oddrow"))

        count += 1




    # Functions

    # Add Function
    def add_record():
        inventory_table.tag_configure("oddrow", background="white")
        inventory_table.tag_configure("evenrow", background="#A5ABC1")

        global count
        if count % 2 == 0:
            inventory_table.insert(parent="", index="end", iid=count, text="",
values=(
                id_ent.get(),
                Productname_ent.get(),
                price_ent.get(),
                quantity_ent.get(),
                Total_ent.get(),


            ),
                                tags=("evenrow")
                                )
        else:
            inventory_table.insert(parent="", index="end", iid=count, text="",
values=(
                id_ent.get(),
                Productname_ent.get(),
                price_ent.get(),
                quantity_ent.get(),
                Total_ent.get(),


            ),
                                tags=("oddrow")
                                )
        count += 1


    # Delete All Function
    def delete_all():
        for record in inventory_table.get_children():
            inventory_table.delete(record)


    # Delete One Function
    def delete_one():
        x = inventory_table.selection()[0]
        inventory_table.delete(x)


    # Select Record
    def select_record():
        id_ent.delete(0, END)
        Productname_ent.delete(0, END)
```

```
        price_ent.delete(0, END)
        quantity_ent.delete(0, END)
        Total_ent.delete(0, END)
        selected = inventory_table.focus()

        values = inventory_table.item(selected, "values")
        id_ent.insert(0, values[0])
        Productname_ent.insert(0, values[1])
        price_ent.insert(0, values[2])
        quantity_ent.insert(0, values[3])
        Total_ent.insert(0, values[4])



    # Update Button
    def update_record():
        selected = inventory_table.focus()
        inventory_table.item(selected, text="", values=(
        id_ent.get(), Productname_ent.get(), price_ent.get(), quantity_ent.get(),
Total_ent.get(), ))

        id_ent.delete(0, END)
        Productname_ent.delete(0, END)
        price_ent.delete(0, END)

        quantity_ent.delete(0, END)

        Total_ent.delete(0, END)


        # Clear boxes
        id_ent.delete(0, END),
        Productname_ent.delete(0, END),
        price_ent.delete(0, END),

        quantity_ent.delete(0, END),

        Total_ent.delete(0, END),



    #----------------------- Buttons -------

    btn_frame = tk.Frame(
        detail_frame,
        bg="#7285CC",
        bd=0,
        relief=tk.GROOVE
    )
    btn_frame.place(x=40, y=309, width=310, height=130)

    #-------- Add Button
    add_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="Add",
        bd=2,
        font=("Arial", 13), width=15,
        command=add_record
    )
    add_btn.grid(row=0, column=0, padx=2, pady=2)
```

```
    #--------- Update Button
    update_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="Update",
        bd=2,
        font=("Arial", 13), width=15,
        command=select_record
    )
    update_btn.grid(row=0, column=1, padx=2, pady=2)

    # --------- LogOut Button
    LogOut_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="LogOut",
        bd=2,
        font=("Arial", 13), width=15,
        #command=Logout

    )
    LogOut_btn.grid(row=1, column=0, padx=2, pady=2)

    #-------- Save Button
    cal_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="Save",
        bd=2,
        font=("Arial", 13), width=15,
        command=update_record
    )
    cal_btn.grid(row=1, column=1, padx=2, pady=2)

    #-------- Save Button
    save_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="Clear",
        bd=2,
        font=("Arial", 13), width=15,
        command=delete_all
    )
    save_btn.grid(row=2, column=0, padx=2, pady=2)

    #------ Delete Button
    delete_btn = tk.Button(
        btn_frame,
        bg="#495582",
        foreground="white",
        text="Delete",
        bd=2,
        font=("Arial", 13), width=15,
        command=delete_one
    )
    delete_btn.grid(row=2, column=1, padx=2, pady=2)

    win.mainloop()
```

```
# ================================MENUBAR WIDGETS================================

menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Account", command=ShowLoginForm)
filemenu.add_command(label="Exit", command=Exit)
menubar.add_cascade(label="File", menu=filemenu)
root.config(menu=menubar)


#==================================FRAME==========================================

Title = Frame(root, bd=0, relief=SOLID)
Title.pack(pady=10)

# ================================LABEL WIDGET=====================================


lbl_display = Label(Title, text="Offline Inventory System", font=('arial', 45),
                    bg="mediumslateblue",
                    foreground="white")
lbl_display.pack( )

login_button = Button(root, text="login", font=("arial, 28"), command=ShowLoginForm)
login_button.pack()

Exit_button = Button(root, text="Exit", font=("arial, 28"), command=Exit)
Exit_button.pack()


#
==================================INITIALIZATION===================================

if __name__ == '__main__':
    root.mainloop()
```

References

https://en.wikipedia.org/wiki/Digital_Revolution#Socio-economic_impact
https://www.unionsquareawards.org/why-software-important/
https://www.veracode.com/blog/managing-appsec/role-applications-todays-digital-world

https://www.synergysuite.com/blog/let-a-digital-inventory-system-lighten-your-workload/#:~:text=What%20is%20digital%20inventory%20management,efficiency%20and%20a%20decreased%20workload.

https://bootcamp.pe.gatech.edu/blog/easiest-programming-languages-to-learn/
https://www.codingdojo.com/blog/python-2-vs-python-3-which-is-better-to-learn
https://realpython.com/python-gui-tkinter/
https://www.activestate.com/resources/quick-reads/what-is-tkinter-used-for-and-how-to-install-it/
https://en.wikipedia.org/wiki/Tkinter
https://en.wikipedia.org/wiki/SQLite
https://docs.python.org/3/library/sqlite3.html#:~:text=SQLite%20is%20a%20C%20library,SQLite%20for%20internal%20data%20storage.

https://datacarpentry.org/python-ecology-lesson/09-working-with-sql/index.html#:~:text=sqlite3%20provides%20a%20SQL%2Dlike,the%20CSV%20and%20SQL%20formats.
https://www.sqlitetutorial.net/sqlite-python/sqlite-python-select/
https://www.digitalocean.com/community/tutorials/how-to-use-the-sqlite3-module-in-python-3
https://www.sqlitetutorial.net/sqlite-python/insert/#:~:text=To%20insert%20rows%20into%20a%20table%20in%20SQLite%20database%2C%20you,Third%2C%20execute%20an%20INSERT%20statement.
https://www.sqlitetutorial.net/sqlite-insert/
https://realpython.com/python-gui-tkinter/
https://www.tutorialspoint.com/python/python_gui_programming.htm
https://python-course.eu/tkinter/events-and-binds-in-tkinter.php
https://developers.google.com/edu/python/introduction
https://www.freecodecamp.org/news/what-is-python-beginners-guide/

https://www.infoworld.com/article/3331923/why-you-should-use-sqlite.html?upd=1668861800220
https://sites.google.com/site/adithie1121010001/advantages-of-tkinter
https://www.profitableventure.com/advantages-disadvantages-inventory-management/