A.S.C

PROJECT

AUTOMATIC . STOCK . CHECKER

By Christopher Greer

Contents

Introduction

Purpose of this Document

The purpose for this documents creation is in order to summarise the required functional requirements of the A.S.C system. Due to this the points made in this document are not necessarily the whole solution only a guideline for the systems functionality.

What is the Problem and why is it Possible?

The problem I am hoping to solve with the proposed A.S.C program is to aid in the organisation of stock at a local Jewellers of Bentley and Skinner. This is as they repeatedly and easily can lose track of stock due to them using a paper based system and as well as out dated versions of Microsoft Office and Windows. These outdated versions lack the modern methods of organisation that can be seen in newer versions of software. However obtaining the newer programs is costly and is not of high priority for the business. Therefore by allowing this project to be completed for them they will save on costs.

This problem is unique in the fact that it can use computational methods as a way to solve it. For example most of the staff are using programs with simplistic UI's meaning that Abstraction plays a big roll. This is as the more complex parts of the systems will have to be hidden via an easy to use UI such as a menu or table. Decomposition will allow me to split the program down into much smaller modules so they can be completed much easier than before.

Why can this be solved?

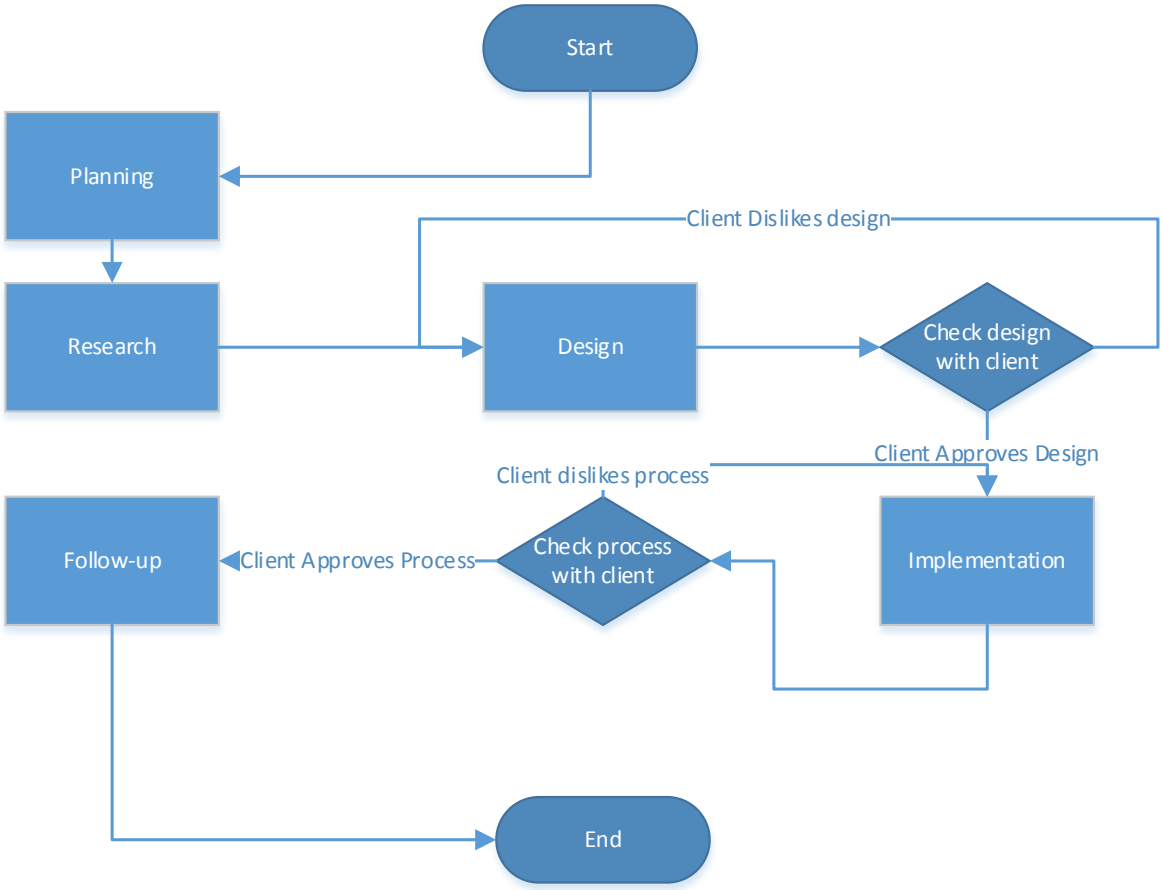
This solution can be solved by software as a dedicated program will be able to constantly be looking at the stock and be able to check everything quickly. This is as a person checking the stock would take a few hours where a PC program could do it in minutes. Therefore by using the software the productivity of the shop will increase and could then lead to multiplier effect bring in more profit.

Timescale

This is an estimated copy of a timescale showing every step of the project right up to the intended completion date of December 2016 where the finished product should be implemented into the workplace.

Task Name	Jul-16	Aug-16	Sep-16	Oct-16	Nov-16	Dec-16	Jan-17
Planning							
Research							
Design							
Implementation							
Follow-Up							

Flow-Chart of Time Scale



Feasibility Study

This feasibility study is used to ensure that the creation of this system is achievable in all areas it needs to be. These areas being Technological, Economically and Legally as well as in terms of the businesses Operations and Schedule. This approach is called TELOS for short. For this feasibility report to be done correctly all of these bases must be covered by the study.

P.E.S.T Analysis

The PEST analysis aims to look at the Political, Economic, Social and Technological impacts of the given project. This analysis covers more areas such as Politically and Socially which both could have an impact on the system as well. This is as Political impacts explore the legality of the system and social impacts explores the how time affects the program. This therefore incorporates the Schedule of the system. Therefore this analysis covers four of the TELOS sections in total.

Political Impacts **Data Protection Act 1998** Any personal data of staff that will be stored on the system will have to be allowed by that member of staff.	**Economic Impacts** **Increase in Online Sales** Due to the prospect of increased ability of integration the website can directly tap into the database allowing for more people to be targeted and therefore bringing more people into the store **Upfront Cost** The system being implemented will have low a cost due to it using an already owned low budget computer. Due to this the cost should remain low however some obsolete technology may have to be exchanged in order to fully functionalise the system.
Social Impacts **Less Time Wasted** The interface of the system will allow data to be enter a lot quicker therefore saving time allowing it to be dedicated elsewhere. **Data Vulnerability** Due to the data being entered on to a Database anyone with access to the Database will have access to the data. However the database can be made more secure via many methods such as encryption and compartmentalisation. However there is always a risk.	**Technological Impacts** **More of the Business Online** Due to the Database having the ability to being put online data can be accessed at any time in the store. **Older Systems Maybe Obsolete** Due to the nature of the database some older devices may become obsolete. This therefore may cut them out of being able to take advantage of the new system.

S.W.O.T Analysis

The SWOT analysis analyses the strengths and weakness of the system as well as Opportunities and threats to the system.

Strengths **Organisation** Due to the easy way a database functions organising stock is a lot easier	**Weaknesses** **Time** Due to the huge amount of stock the shop may have it may take time to upload all of the information within the first time
Opportunities **Easily Updated Website** Due to the intended functionality of the program it could easily be hooked up to a website to show a more updated list	**Threats** **Security** With the database being connected to the internet it may be possible to hack the database and retrieve any information stored within it

Analysis of the Problem

Research Evidence

Example Documents

Blank Questionnaire

Questionnaire

1. What is your role in the business?

2. How would you rate your Technical knowledge?

   Low Medium High

3. Why would you rate it this way?

4. How many items of stock do you usually have in store?

5. How many staff members will use the system?

6. How secure does the system need to be?

   Authorised Only Authorised and Guests Anyone

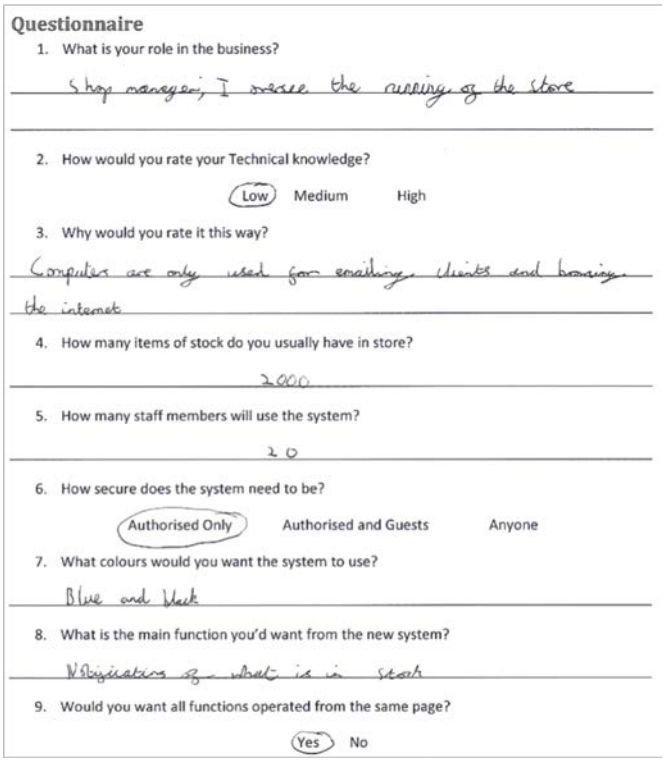
7. What colours would you want the system to use?

8. What is the main function you'd want from the new system?

9. Would you want all functions operated from the same page?

   Yes No

Shop Manager's Questionnaire



Questionnaire

1. What is your role in the business?

   Shop manager; I oversee the running of the store

2. How would you rate your Technical knowledge?

   (Low) Medium High

3. Why would you rate it this way?

   Computers are only used for emailing, clients and browsing the internet

4. How many items of stock do you usually have in store?

   2000

5. How many staff members will use the system?

   20

6. How secure does the system need to be?

   (Authorised Only) Authorised and Guests Anyone

7. What colours would you want the system to use?

   Blue and black

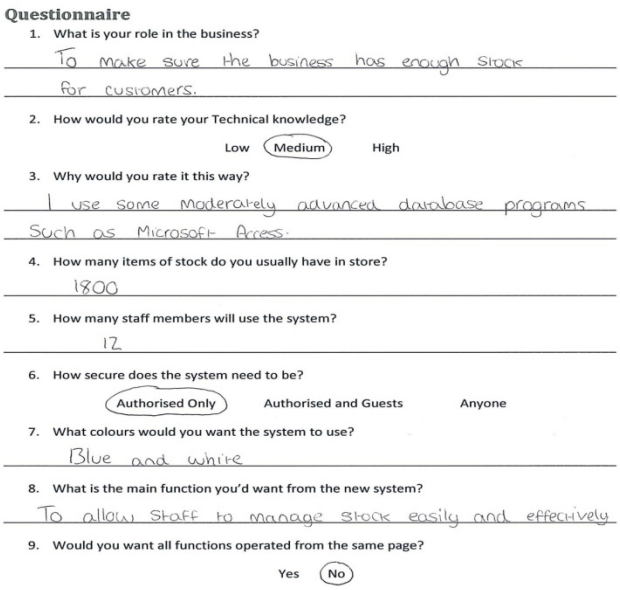
8. What is the main function you'd want from the new system?

   Notifications of what is in stock

9. Would you want all functions operated from the same page?

   (Yes) No

Stock Manager



Questionnaire

1. What is your role in the business?

   To make sure the business has enough stock for customers.

2. How would you rate your Technical knowledge?

   Low (Medium) High

3. Why would you rate it this way?

   I use some moderately advanced database programs such as Microsoft Access.

4. How many items of stock do you usually have in store?

   1800

5. How many staff members will use the system?

   12

6. How secure does the system need to be?

   (Authorised Only) Authorised and Guests Anyone

7. What colours would you want the system to use?

   Blue and white

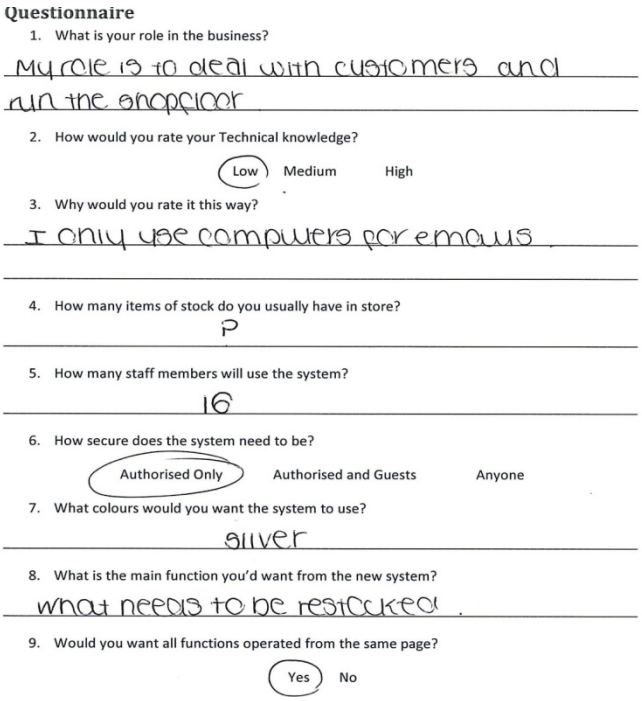
8. What is the main function you'd want from the new system?

   To allow staff to manage stock easily and effectively

9. Would you want all functions operated from the same page?

   Yes (No)

Shop Assistant



Questionnaire

1. What is your role in the business?

My role is to deal with customers and run the shopfloor

2. How would you rate your Technical knowledge?

(Low) Medium High

3. Why would you rate it this way?

I only use computers for emails.

4. How many items of stock do you usually have in store?

P

5. How many staff members will use the system?

16

6. How secure does the system need to be?

(Authorised Only) Authorised and Guests Anyone

7. What colours would you want the system to use?

silver

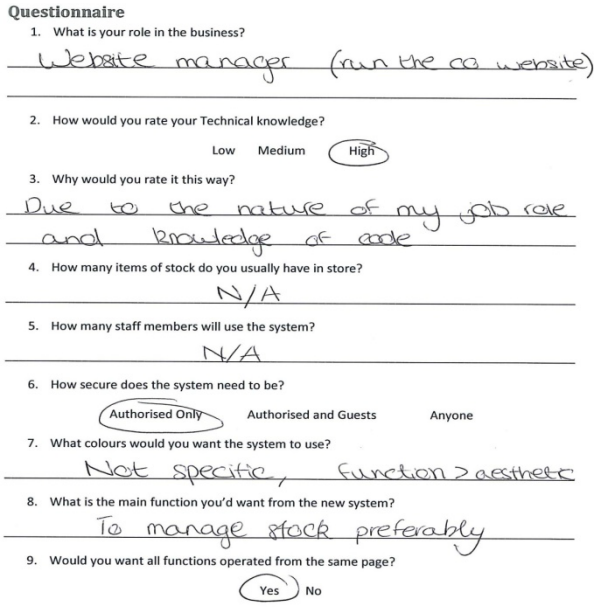
8. What is the main function you'd want from the new system?

what needs to be restocked.

9. Would you want all functions operated from the same page?

(Yes) No

Website Manager



Questionnaire

1. What is your role in the business?

Website manager (run the co. website)

2. How would you rate your Technical knowledge?

Low Medium (High)

3. Why would you rate it this way?

Due to the nature of my job role and knowledge of code

4. How many items of stock do you usually have in store?

N/A

5. How many staff members will use the system?

N/A

6. How secure does the system need to be?

(Authorised Only) Authorised and Guests Anyone

7. What colours would you want the system to use?

Not specific, function > aesthetic

8. What is the main function you'd want from the new system?

To manage stock preferably

9. Would you want all functions operated from the same page?

(Yes) No

Questionnaire and Interview Analysis

In order to find out more about the client and their requirements I constructed a questionnaire of nine questions. The blank questionnaire can be seen above with the questions which range from who that member if staff is to what they except from the system. These questions are important if I wish to build up a successful user profile of each staff member as well as create useful requirements for what the system needs.

For example from the shop manager's questionnaire I was able to find out that "he oversees the running of the store". This is important as it proves any detail that is added to the program must be run through him first as he gets the final say in how his business is ultimately run. This gathered information has been added to his user profile below. Also he has rated himself with a low Technical

knowledge as he believes “Computers are only used for emailing his clients and browsing the internet”. Therefore in his user profile I have stated that he only uses computers for their basic functions such as email and web browsers.

From the stock managers questionnaire I was able to find out that he had more computer knowledge then the shop manager and shop assistant. This is as he “uses advanced database programs such as Microsoft Access”. However on further inspection I was able to find out that his use of Access was very basic as he used it as an alternative to Excel. Therefore he differs from the rest of the store who decide to store entities in excel.

The website manager’s questionnaire gave me an insight on what the website mangers role in the business entails. This is that the website manager “runs the website”, on further insight from an interview I was able to find out that this involves the writing of HTML and CSS code. For this reason and the information stated in the questionnaire he has been given a rating of High technical knowledge.

Shop Assistants questionnaires and interviews gave me good information which allowed me to build a good user profile for them. For example from the questionnaires I manged to gather that their main role in the business is to “deal with customers and run the shop floor”. This is an important factor in the user profiles as knowing what their role is allows me to build a system which can better benefit the user. I also know from this research that they have a low technical knowledge rating as they use computers in the same way as the shop manager.

From all of the questionnaires I was also able to gather some of the metrics of the system these being that there would be 2000 items of stock and 20 staff members. This is as the shop manager said there would be 2000 items and the stock manager said there would be 1800 items. Due to the uncertainty in these results I decided to take the higher of these two to make sure that the system was built to last. From the shop manager’s questionnaire and the shop assistants I was able to gather the metric for the number of staff members. Again I took the highest result given to me in case the shop were to expand or more users needed to be added then intended. These metrics are important as they allow me to build a system which is big enough and capable enough to hold the required information.

From these questionnaires I was also able to gather the fact that only authorised staff members should access the system which means that a log in system will have to be added to the systems requirements. This is as in all of the questionnaires the Users voted on only authorised personal having access to the system. Also each of the members of staff suggested a colour scheme for the ASC program. Half of the staff members suggested blue whilst the other half suggested a lighter colour to accompany it such as white or silver. However the website manager raised a good point of function should come before the aesthetic. Also the questionnaires specified that most users want easy access to all functions all at once meaning a menu format could be added. Though this could create a cluttered UI so it may have to have sub-Menus coming from the main menu

Another potential requirement that could be added is the ability to send notifications to the appropriate user. This is an idea that was raised by the shop manager in his questionnaire and interview along with the shop assistants and implementing this would allow the Automatic Stock Checker to become automated.

The interviews allowed me the ability to ask more in depth questions about the responses given in the questionnaire and to gather more information in general. For example the website managers’ questionnaire was a bit lacking in some places and so the interview allowed me to verify some of his

answers. For example in the questionnaire when asked what he wants from the system he responded “To manage stock preferably”. Therefore in his interview I asked how having the ability to do this would benefit him. In response I was told “If it was to be stored in a database that database may be able to be linked to the website”. This is as at the moment everyone uses Excel spreadsheets other than the stock manager who uses Access. Therefore by using the database as my main storage it can have many other benefits.

Another interview with shop assistants allowed me to ask more about the required functions they wished for from the ASC program. This is as the answer I retrieved from the questionnaire was that they wanted the restock notification like the shop manager did. From the interview I was also told another function that could be a good idea is one to “order more of an item”. This requirement has been added to the requirements list below.

Speaking of the restock notification this was something that came up in my interview with the shop manager who first came up with the idea. From the interview I managed to gather more details about how he would use that particular function. The answer in which I was given was that he’d wish to receive the notification on multiple devices so he could always keep tabs on what had run out of stock. Also he asked for a search function for when he is in the business and serving customers. This would allow him to quickly find a specific item the customer asked him about without having to scroll through a huge list. Due to this justification the search function has been added to the list of requirements as well.

Research Analysis

Stakeholders

The stakeholders are important for the project as they will be the ones who this software is being made for. For this project the stakeholders shall be:

Stake Holder	Who they are	How they will use it	Needs
The Shop Manager	They oversee the running of the whole store	Receive notifications from the program to know what is in and out of stock	To see what is in the shops stock via notifications and search function
Stock Assistant	Makes sure that the business has enough stock. Reports to the shop manager.	Whole stock in one place at one time so it is easier to manage	Assess the situation of the shops stock Order more materials when they run out
Shop Assistant	Runs the shop floor and handle all of the customers	Answer customers questions by checking the stock list	Easily restock more items and see what is in stock.
Website Manager	Manages the shops website and advertising on the internet.	Use this software to update the stock shown on the website	The information to be stored in a database so they can easily link it to the database.

These stakeholders can be accessed at any time in order to confirm or test different solutions in order to make it as user friendly to them as possible.

Breakdown of Methods

The main method of research was through a primary data collection of a Questionnaire which was given to the business in order to find out what they want from the system. This was done via a variety of questions in order to create a criteria for the program to be compared to. From this questionnaire I managed to gather the metrics that would be needed in the system. These metrics include the amount of stock that will have to be entered into the system as well as a count of the total amount of staff that would be required to use the program.

From this information I was able to give an average on the amount of disk space that the system would take up. This is by using 8KB per user, 8KB per item and 4KB per screen of the program. The need for this was in order to make sure that the PI that would be used to store the bulk of the program had enough room to store all of the information on. Fortunately this was the case. I was also able to gather how their current stock system works so I will be able to create a better digital version of it. This is important as it will need to be familiar and easily accessible to all of the future users of the software.

I also used did some secondary data collection via the internet were I looked at how other stock programs function. From this I was able to gather that there were three types of view, the most common of them work via a search function or they just give a table view of all of the stock. Another method that is used is via a hand held barcode scanner that is used on a product to see how many of them are in stock. This method is far more complex then I will be able to create due to the limits of the technology available to me.

From all of this research I was able to create a detailed requirements list that will allow me to create the software that my clients will expect.

User Profiles

The users are an important part for the functionality of the finished product as they will be the ones who will be using the software on a daily basis. From recent interviews and questionnaires I have gathered the following information on each type of user;

User	Experience	Technical Knowledge	Other Characteristics
The Shop Manager	Will have been in the shop the longest and therefore will have the most idea on how the shop functions and is therefore the biggest candidate for understanding how the software will have to be integrated	Low; Only uses computers basic functions when working and uses nothing more than emailing customers and web browsers.	Is in charge of the shop and will therefore have the biggest say on if the A.S.C meets the requirements
Stock Assistant	Is the primary target of the A.S.C software due to them very experienced in sorting stock. This makes them and ideal candidate for creating the Functional Specification as they	Medium; Will use more advanced functions of spreadsheets and basic functions of Access. More experienced than others.	Could fix database if it were to break

	will know how A.S.C will have to function		
Shop Assistant	Will be dealing directly with clients and will therefore be a secondary target as they will know how the U.I will need to be displayed in order to allow an easier transition of information to clients.	Low; Mainly using phones, emails and spreadsheets in a basic fashion.	N/A
Website Manager	Very little experience in terms of stock sorting but they have a vast experience in updating the websites stock. Therefore they will know how the U.I, User Interface will have to be displayed for the customers view.	High; Uses computing and coding in their everyday life and will therefore be the most experienced of the group.	May not be situated on site and therefore may be hard to contact.

Metrics

From the calculations given above in the research analysis I have managed to construct the given values the metrics and requirements for the system. This being that there is 8KB per user, 8KB per item and 4KB per screen of the program. From the given information it is clear to see that I have enough space in the PI for the program to be stored and run effectively. The No. of entities and staff are taken from the interviews and questionnaires done with staff. The highest units given to me have been used in order to make sure the system is ready for the largest amount of strain possible.

Item Name	Amount Of Item, Estimated
Entities	2000
Database tables	2
Users	20
Screens	5
Disk Space	16gb(Available)
Space Needed (Approx.)	2.02Mb

Requirements

This list of requirements will assist me later in my design and development stages as these functions will allow we to break the problem down into parts which are a lot easier to construct. This deconstruction of these essential parts is known as decomposition. By using it the problem shall become a lot easier to solve. This is as multiple smaller problems are easier to solve individually than one large one. Due to this the design, development and iteration of the project should be relatively quick as I won't have to think about the whole picture all at once.

Priority Levels:

M – Must Have
S – Should Have
C – Could Have
W – Would like to have

Function Id	PROGRAM Functional description	Priority Level	Key Analysis Points/Notes
UI	Is the main interface that will be used by the user when the software is completed	S	Is an essential module as it will be required to be the main face of A.S.C program
Search	A function that will be used to search through all entities of the Database in order to find a specific entity	S	Is not an essential module but will make the program more efficient and ease its use for the user
Input	Most important function as it allows data entities to be added to the system as well as altered if the unit changes	M	Without this function the system would be unable to function as there would be no way to add to the database
Alter	Second most important function as it allows the alteration of data entities already in the system	M	This function is provided as a means to amend any user errors in the system.
Restock	This is a proposed function that would allow the automatic request for more of an item to be done with a click of a button	C	Though this is not a critical feature it would help the business as it would be situated next to all the entities descriptions.
Back	This is a function that will allow the user to return to the previous page	M	This is a critical function of the software as without it the user will get stuck on one screen and will not be able to return to the menu
Sign In	This is a function that will check a secure database for the correct sign in details	M	Without this function the software will be very unsecure as there will be no way of stopping random people looking at the data
Out of Stock Notification	This is a notification that will be sent to one of the authorised members of staff via email in order to tell them to order more stock.	C	This function is one intended to help the functionality of the business by allowing it to become more efficient. It is not an essential module though but one suggested by the shops manager

Testing Criteria

The criteria in which the final product shall be tested against has been given below. Each point of the criteria has been made specifically to assess at least one of the requirements given above. These criteria are important when considering the User especially if the criteria involves the usability of the finished product.

Testing Criteria Point	Reason for Creation
1.Easily search through all items in stock	This criteria point was made to test the function with the ID of "Search". This is in order to make sure the function works to its full intended purpose.
2.Easily edit any items details	This point was made to so that the function with the ID of "Alter" can be tested as this is the second most important function.
3.All information is secure and can only be accessed by authorised personal	This tests the "Sign In" ID'd function as it was one that was raised as being very important by the shop manager and other staff members.
4.Automatic alert if stock reaches zero	This tests whether the "Out of Stock Notification" works to its intended use of being able to notify staff members.
5.Account details can be changed	This is another criteria that tests the "Alter" function from a second viewpoint.
6.New stock can be inputted by appropriate users	This tests the "Input" function to make sure users can add new stock
7.The screens can be navigated easily	This criteria test whether the function dubbed "Back" operates at its full extent.
8.The restock button orders more stock from the retailer	This final criteria is made especially for the function "Restock".

Limitations

Potential limitations of this project is the fact that the more items in stock may prolong a search or loading functions time cycle. This is as the more items of stock there are to search through the longer it will take for the system to check the criteria against each given item. If this is the case then users may get impatient with the system and it may fail to speed up productivity of the system.

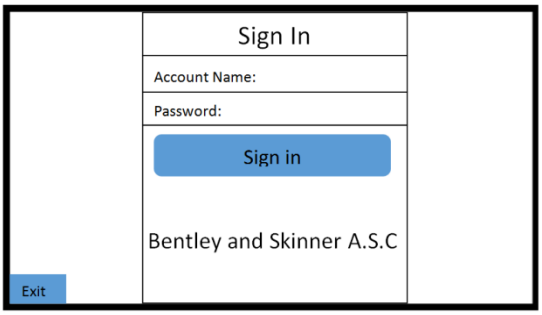
Another potential limitation is the ability of sending notifications automatically when they reach zero. This is as it would require the program to be running all the time which is something that may not be able to occur. Also the ability of receiving the given notifications has not yet been determined as it may have to be an inter-program notification and not have any external output unless you're on the program.

Feedback from the questionnaires suggests that some users wish for a quick and easy access to all functions at once. However I believe this may lead to the main screen of the program to feel bloated and not work effectively. This is as the main screen being filled with all the given functions may lead to confusion and the individual functions in themselves not working properly. This is as when each one is called all the widgets would have to be removed and replaced and the more there are to be replaced the more potential problems could occur.

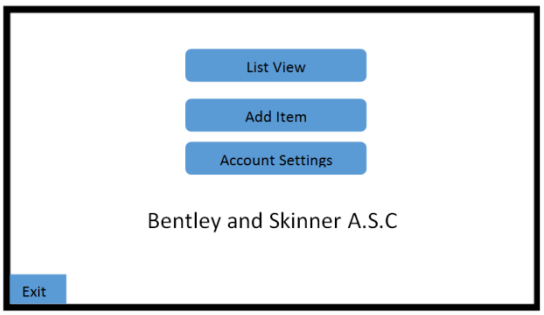
The way in which items would be stored, e.g. by a database, makes it difficult for multiple users to access it all at once. The main may to bypass this is for the database to be located on the PI which is connected in a network to all other PC's. The fetch address in the database would then have to be changed to that of the PI so all users can access the PI and therefore the database stored within it.

Design of the Solution

Screen Concepts

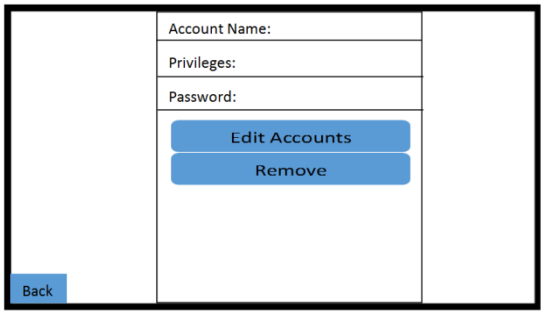


Sign in



Main Menu

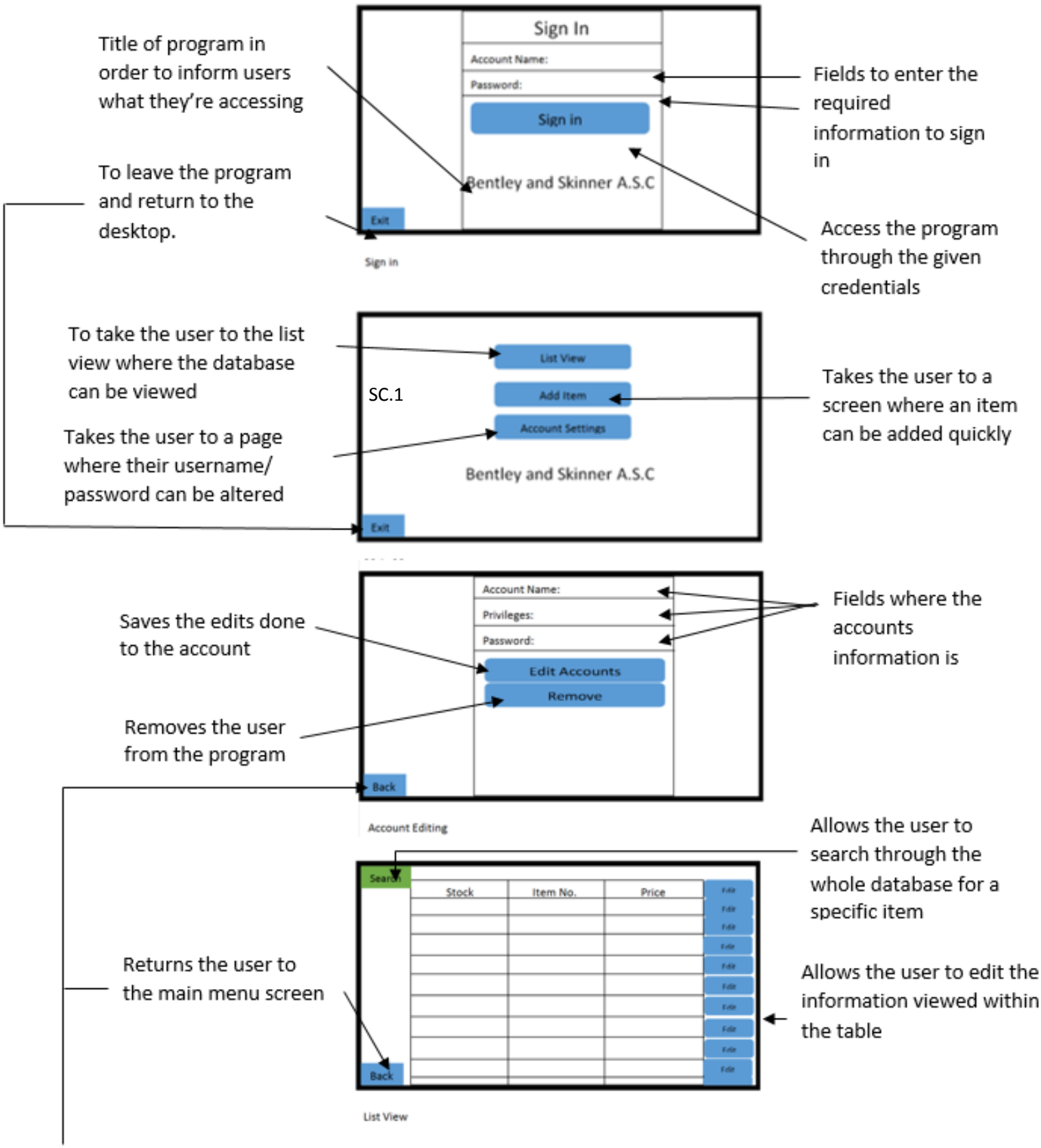
List View

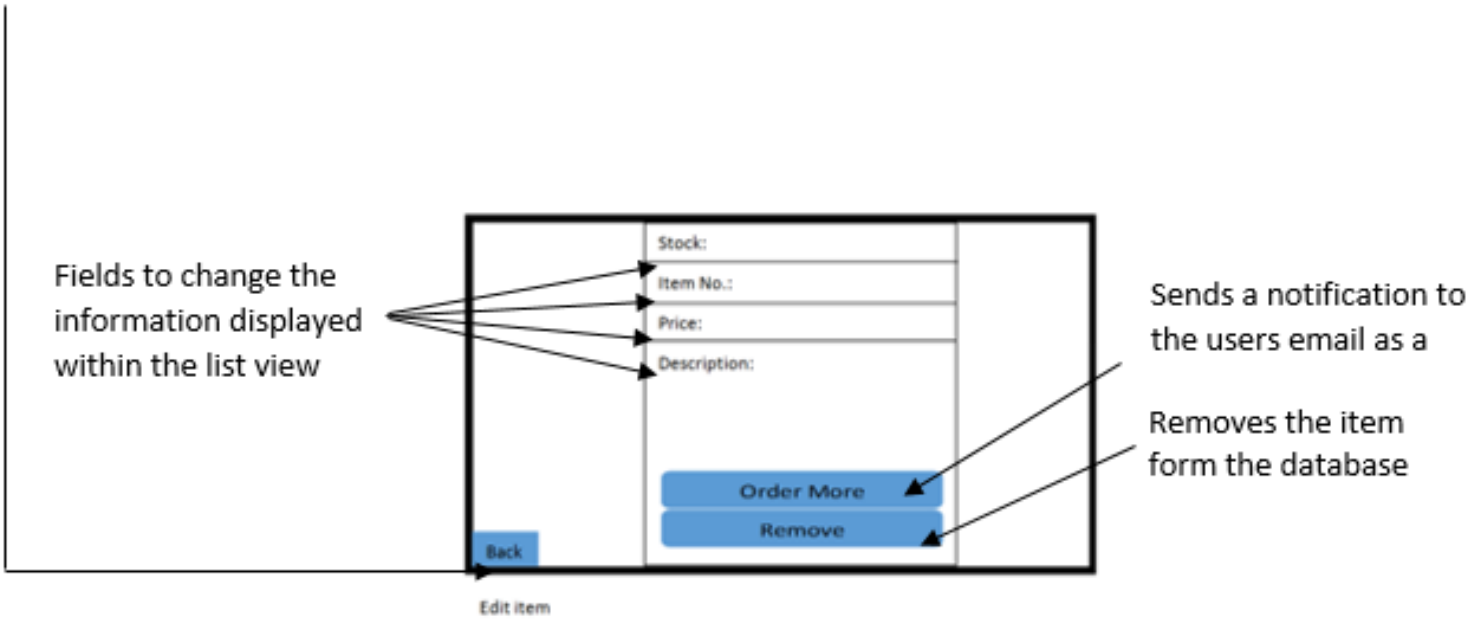


Account Editing

Edit item

Screen Annotations





User-Interface Description

The User Interface needs to be very simplistic that is why the layout has been designed the way it has. For example the main menu, SC.1, only has three buttons which take up most of the screen within the centre column therefore using the simple centre stage effect. Due to this the main buttons are placed here filling up most of the space and giving the main functions of the system. This allows for quick access of the system due to the simplistic styling of the UI.

The list view, SC.2, allows quick visibility of all stock items which should be organised in numerical/alphabetical order. This allows the user to scroll through all of their products to find a specific item. However there is a search bar located in the top right in order to find a specific item much quicker than usual. This does mean that the user may need to know what the item's no is beforehand though a description can be found once the item has been clicked on. The list view shows the amount of that item, the items number/name as well as the price of the item.

Once an item has been clicked on they can edit that item's details, this can be seen in SC.4. In the edit item screen the user can edit the amount that is in stock, the item's name/ number, the price of the item as well as a description of what the item is. There are also the options to order more of an item and remove the item from the list.

In SC.3 the account settings screen can be viewed, this screen is mainly for administrators but some functions can be accessed by other members. For example the admin will be able to change other accounts names and privileges but an administrator will be able to view and edit these and also remove accounts all together. The user can also change their password from this page which is needed in order to keep the information secure. This is why SC.5 exists as it is the sign in screen that will determine what specific users can access and what they cannot as well as stopping any person from accessing the information which needs to be kept as secure as possible.

On each screen a back button is located the bottom right hand corner in order to return the user back to a previous screen. This is the same across all of the UI's except for the main menu where the back button is replaced by an exit button which will allow the user to exit the program.

Description

The original problem that ASC was designed to solve can be decomposed into many smaller problems as can be seen in the diagram. The main problem can be split down into a UI as well as the security issue that the software raises. This security problem can be solved via a sign in in function as well as an authorisation level of each user. These can be either a user or an administrator each will

allow a different level of access to the program e.g. hiding certain functions. All of this comes via the sign in function which allows for different users to operate the program.

The sign in function works via the widgets of the Log In screen. These widgets check the user name and password entered in the screen with those found in the database. If one of the entries matches the entered information then the user will be signed in with that profile. When that profile has been signed into the authority of it is tested, depending on its level depends what widgets it has issued on the menu. For instance there are two types administrator and user. The administrator can change the information of other users including user name password and their authority.

The database viewer has one simple function of allowing the users to see what is in stock, the simplest way to do this is for a table to fetch and display the information from the database. A scroll function could be implemented to allow all 2000 items to be scrolled through. However it is probably best for this not happen as it will make the UI messy. Instead the search function will be used to display the information that the user wants, this will occur via an entry field. A string shall be entered here that will be checked via a like SQL. This will then return the data from the database that the user wishes to view. Therefore this will remove the need to scroll and will leave the UI less cluttered. There will also need to be a back button displayed somewhere on the screen so people can return to the main menu. This will allow for people to be able to access other functions easier as the main menu is the place where all functions can be accessed.

The main menu will have a simplistic design to meet with need for it to be accessible by many different people even with some that aren't very good at computers. It will also allow all the functions to be accessed from the same place at the same time. To do this the menu will be made up of buttons each adhering to a different function of the program. The three main functions will be table view, account settings as well as a quick edit where items can easily be listed to the data base without going through the table view. This is important as the main role of the program is to increase the efficiency of the stock sorting within the store.

The quick edit screen will have the same layout and function of the edit screen from the table viewer. This will be a table format where the name, the price, the amount in stock aswell as a description and item code of an item can be edited. This will be by changing the text in the fields and clicking the edit/save button. However if an item doesn't already exist in the database that item can be added via an add item button or removed by the remove button. This layout is very simplistic and has a similar layout to the main menu which is very important as the users need to feel as if they are still in the same program and can easily do what they wish.

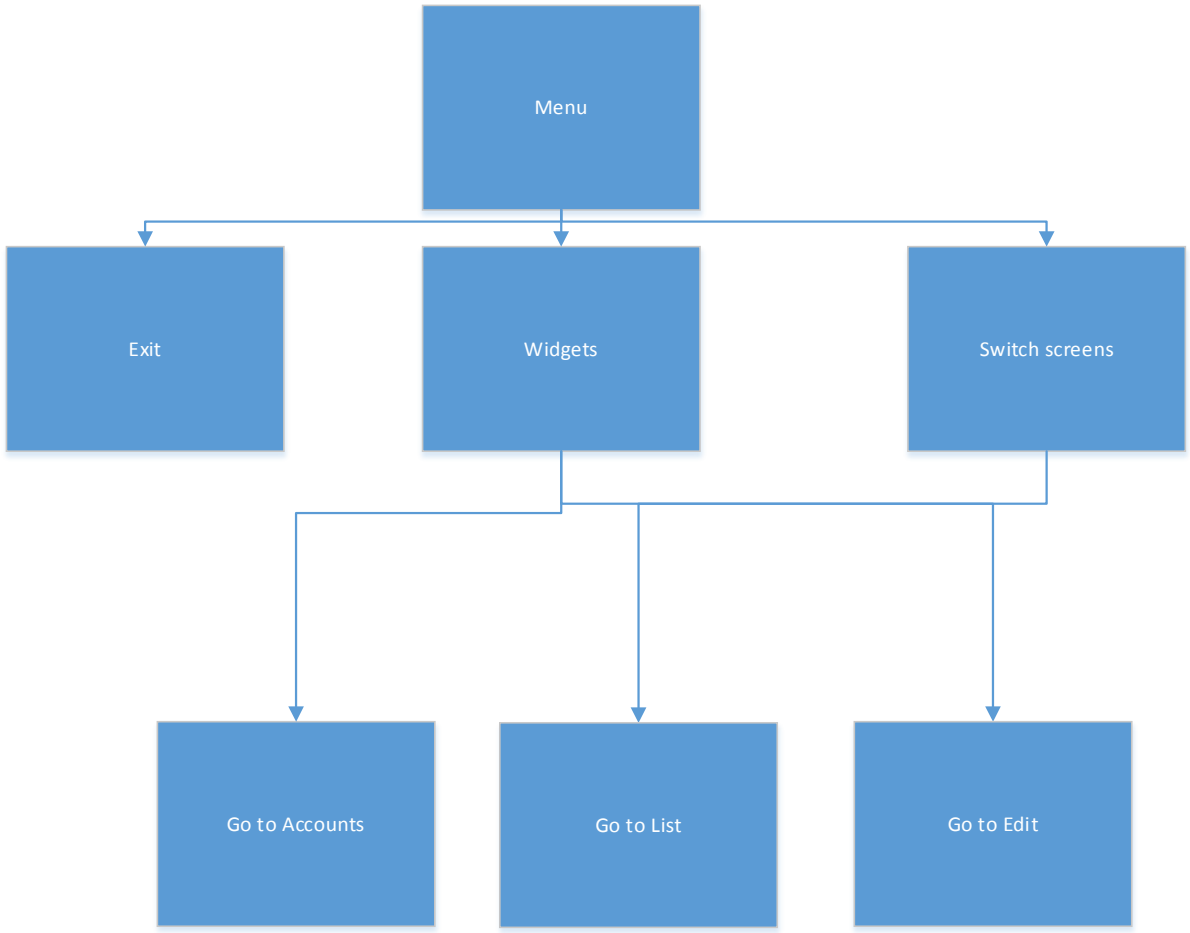
Following with this theme the account settings screen is the same as the edit screen just with different labels but similar buttons. However depending on the clearance of the user it will change what they're able to do here. For example an administrator will be able to edit everyone's information but a standard user will only be able to change theirs. This information will be changed in the fields name, password and authority but only administrators will have access to the authority field. There will also be the buttons save/edit user, add user and remove user with only administrators having access to the latter two. Another thing that only administrators will have access to is the search field located in the top left which will allow them to access other users information.

Flowcharts

These diagrams are essential at helping to break down the problem into its smaller constituents, which is otherwise known as decomposition, allowing me to see what components go into each screen. Although these only provide a basic insight into how the program should work it is still very valuable in terms of me designing the system.

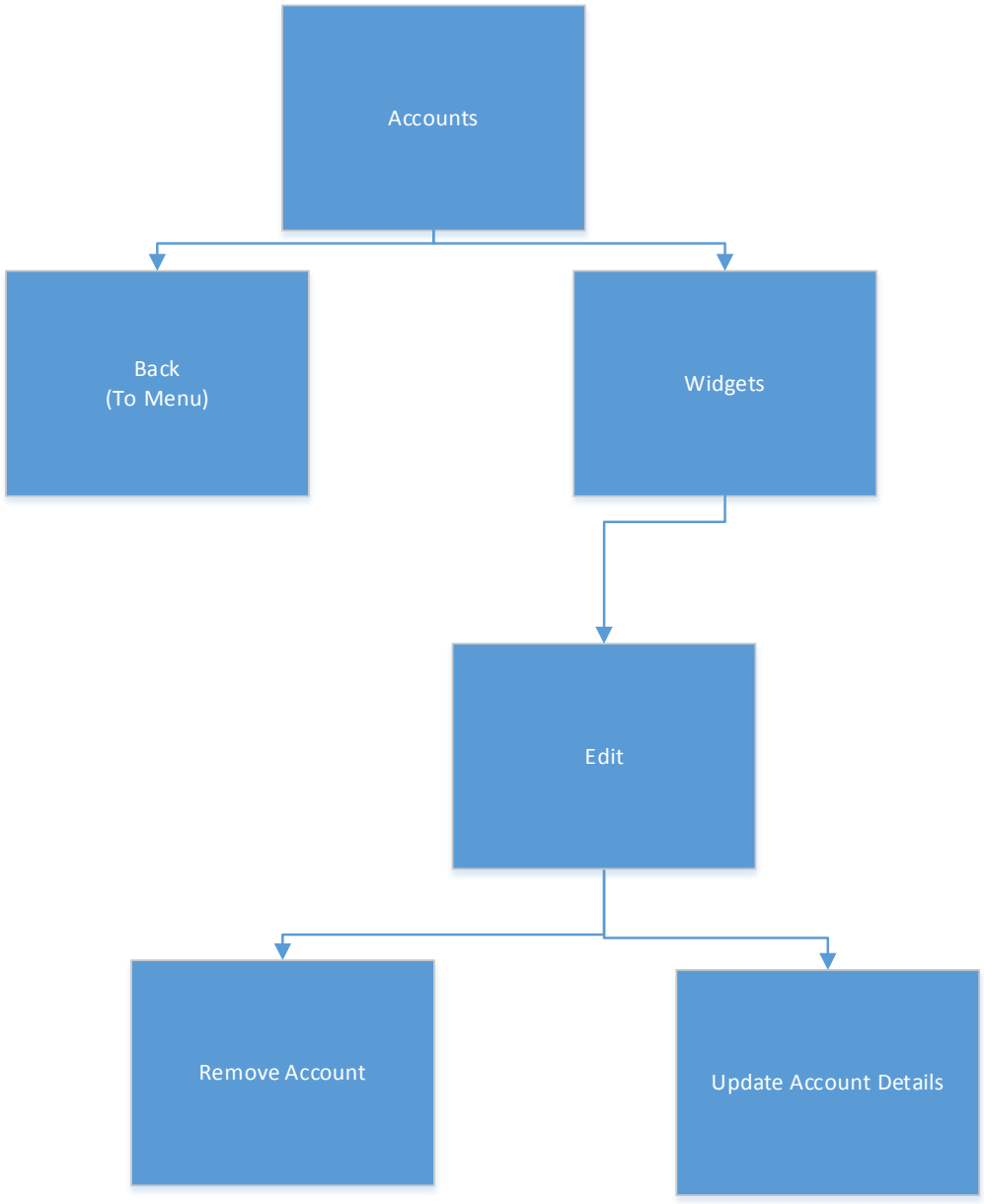
Flow Chart: Menu

Located below is the flowchart for what is needed in order for the menu to function which includes details such as all essential widgets as well as the main function of the screen. In this case these are the widgets “Go to Accounts”, “Go to List” and “Go to Edit” with their main function being to switch between the different screens. The other function that is attributed to this screen is the ability to exit the program which is only found in one other screen being the sign in screen.



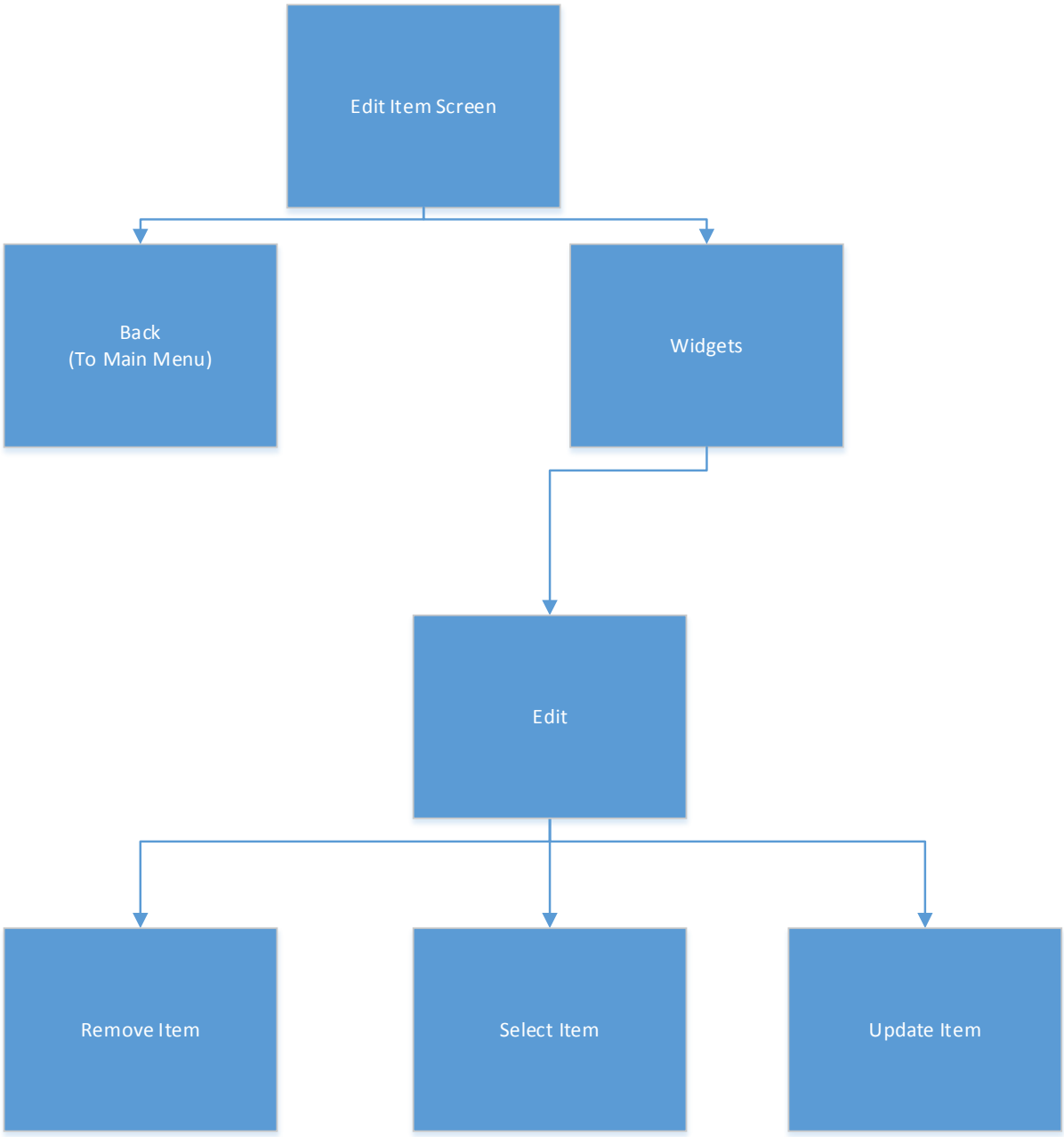
Flow Chart: Accounts

This flow chart details the functions and widgets that will be needed for the accounts screen module. The needed widgets for this screen shall be the buttons “Remove” and “Update” with the functions to “Remove” and “Update” an account respectively. The main function for this screen shall be to edit the account that has been chosen by that user.



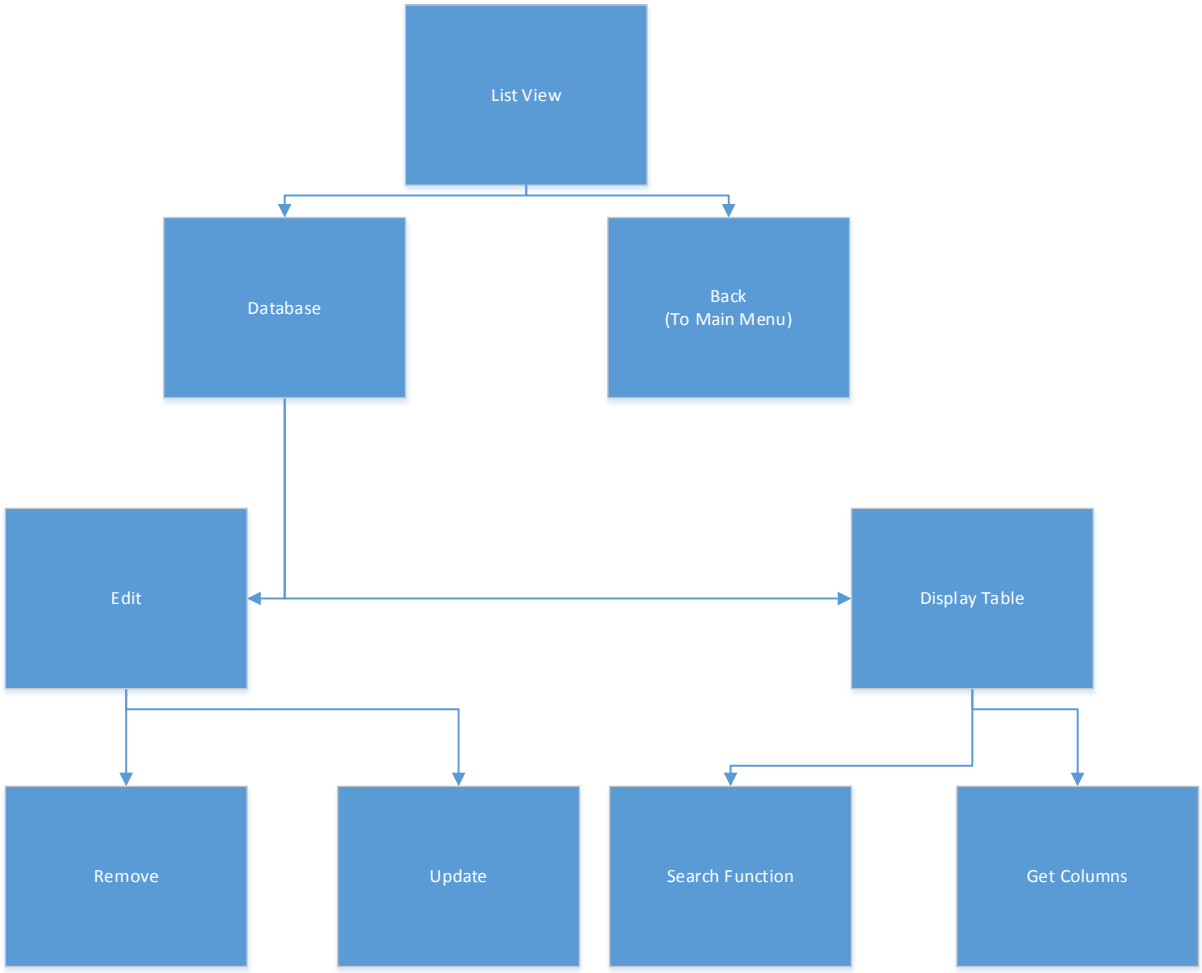
Flow Chart: Edit Item Screen

This flowchart displays everything that is to be inserted into the Edit Item Screen including its widgets and any functions that those widgets may hold. In terms of widgets it shall need buttons titled “Remove”, “Select” and “Update” each having the function to either “Remove”, “Select” or “Update” the selected Items details.



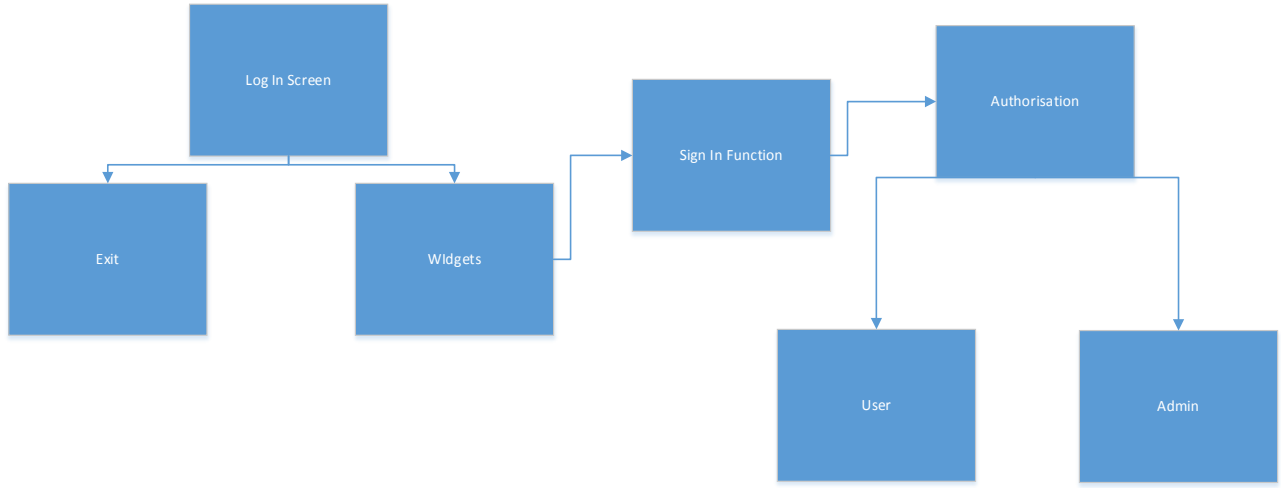
Flow Chart: List View

This flowchart is by far the most complex system and shows everything that shall be included in the list view screen including its widgets and functions. This screen will have buttons to “Remove”, “Search” and “Update” the information. The main functions for this screen will be to “Display the Table”, “Retrieve Information from the Database”, “Set up the Columns” and “Search for the Item via a Criteria”.



Flow Chart: Log In

The diagram below is the flowchart for the Log in Screen which shall be the first screen the user shall see. It shall include one main button to sign the user in to the program and shall be displayed to the user as “Sign In”. Its main functions shall be to enter the program as well as identify the authorisation of the user although this shall be done during the sign in process. This screen shall also have the ability to exit the program and is only the second screen that will have this function.



Pseudo Code

Displayed below is the pseudo code for the system which acts as a first iteration of this software’s development. Here the basic concepts of how the system shall be coded has been laid out but there are still many smaller problems that are yet to be solved. This is as Pseudo code is very simple in its nature and cannot be read by a computer. This therefore results in it being very unlikely that the code in its current state would operate to its full effectiveness. This is as there is no way to test the pseudo code and to remove any problems unless they are seen with the naked eye.

Key to the Pseudo Commands:

Label: A text widget which displays the intended text
Entry: This is an entry field that users can type into for the program to extract significant details.
Button: This is a widget that can be clicked in order to run the function that is assigned to it.
If: this is a statement that runs the given code if given requirements are met
Connect: is used to connect to a Database with the given name
Remove: Is used to remove entries from a database as an SQL command
Open: Used to load another screen or open a document
Print: Used to give text to the user to help them operate the program
Select: This is an SQL command used to gather information from a database
Alter: Is an SQL command used in a database to change the information already stored there
Insert: This is another SQL command that is used within a database to input new information
End: Stops the named process

```
Create Sign IN Screen
        Label "Sign In"
        Label "Account Name"
        Entry "Account Name"
        Label "Password"
        Entry "Password"
        Label "Bentley and Skinner A.S.C"
        Button "Sign In"
        Button "Exit"

        CurrentUser<-Entry "Account Name"
        CurrentPassword<-Entry "Password"

        if Button "Sign In" clicked
                connect to user.db
                if Username==CurrentUser
                        if Password==CurrentPassword
                                Remove all Widgets
                                Open Frame "Menu"
                        else
                        print "Your UserName/Password was wrong"
                else
                print "Your UserName/Password was wrong"

        if Button "Exit" clicked
                End All Processes
```

Sign In Pseudo

This is the Pseudo code for the sign in screen. It starts off with creating the sign in page by creating the widgets listed below that command. After this the new variables are stated, these being “CurrentUser” and “CurrentPassword”. The contents of these are the entries “Account Name” and “Password”.
The intended inputs of the system would be:
The Accounts Name
The Password
Both of these are checked against the declared variables listed above in the “IF” statements given below the declaration. If they’re the same as these variables then it will lead to one of the outputs occurring. In this case moving to the main menu.
The intended outputs from this screen will be:
Going to the Main Menu
Exiting from the program

```
Create Menu Screen
        Button"List View"
        Button "Add Item"
        Button "Edit Accounts"
        Label "Bentley and Skinner A.S.C"
        Button "Exit"

        if Button "List View" clicked
                Remove all Widgets
                Open Frame "List View"

        if Button "Add Item" clicked
                Remove all Widgets
                Open Frame "Add Item"

        if Button "Edit Accounts" clicked
                Remove all Widgets
                Open Frame "Account Settings"

        if Button "Exit" clicked
                End All Processes
```

Menu Pseudo

This demonstrates the Pseudo code for the Menu screen. This page begins, as with all the other screens, with the creation of the screen by loading the widgets that shall be displayed on the screen. Due to this page simply being a navigation screen no variables have to be declared all that has to be done is to give the loaded widgets function. These are what are expressed in the “IF” statements.
There are no intended out puts for this screen but there are four intended outputs. These outputs being:
Moving to the “List View” screen
Moving to the “Add Item” screen
Moving to the “Edit Accounts” screen
Exiting from the program

```
Create Add Item Screen
        Label "Stock"
        Entry "Stock"
        Label "Item No."
        Entry "Item No."
        Label "Price"
        Entry "Price"
        Label "Description"
        Entry "Description"
        Button "Order More"
        Button "Add Item"
        Button "Remove"
        Button "Back"

        NewName<-Entry "Stock"
        No<-Entry "Item No"
        NewPrice<-Entry "Price"
        NewDescription<-Entry "Description

        if Button "OrderMore" clicked
                Open "Notifications.docx"
                Write NewName,"Is running low or is out of stock"
                Close file

        if Button "Add Item" clicked
                Insert NewName, No, NewPrice, NewDescription into StockTable

        if Button "Remove" clicked
                connect to stock.db
                Remove Stock, ItemNo, Price, Description where ItemNo = NO


        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

Add Item Pseudo

The “Add Item” screen begins with the loading of all of the tkinter widgets in order to create the screens UI for the user. After this the program shall declare 4 new variables being “NewName”, “No”, “NewPrice” and “NewDescription”. The contents of these variables shall be the entries “Stock”, “ItemNo”, “Price” and “Description”.
Using these newly declared variables the second “IF” statement can run adding the given entries to the database. There are four intended inputs for this screen being the four given to the variables:

- Items Name
- Items No
- Items Price
- Items Description

There are also four intended outputs for this screen as well:

- A notification is written in to a Word Document
- The Item is added to the database
- The item is removed from the database
- The user is returned to the main menu

```
Create List view Screen
        Entry "Search"
        Label "Stock"
        Label "Item No."
        Label "Price"
        Button "Back"

        NewSearch<-Entry "Search"

        if Button "Search" clicked
                connect to stock.db
                cursor<-Select Stock, Price, Description where ItemNo or Stock = Search
                for rows in cursor =n
                x=0
                if x=<n
                        label "Stock[n]"
                        label "Price[n]"
                        label "Description[n]"
                        button "Edit"
                        x+1
                else:
                        return

        if Button "Edit" clicked
                Remove all Widgets
                Open Frame "Add Item"

        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

List Pseudo

In this screen, “List View”, it begins by loading in all of the required widgets that the user shall need on that screen. After this one variable is declared which in this case is the “NewSearch” variable. This variables contents is from the entry field called “Search”. After this three if statements are given which create the intended outputs for this screen. In this case the outputs are:

- Items are loaded depending on what is in the contents of the search field.
- Users are taken to the Add Item screen to edit an item’s details
- Users are taken to the main menu screen

The inputs for this screen would be the contents of the declared variables.in this case they are :

- The entry to the search field.

```
Create Account Edit Screen
        Label "Account Name"
        Entry "Account Name"
        Label "Privileges"
        Entry "Privileges"
        Label "Password"
        Entry "Password"
        Button "Edit Accounts"
        Button "Remove"
        Button "Back"

        NewName<-Entry "Account Name"
        NewPrivileges<-Entry "Privileges"
        NewPassword<-Entry "Password"

        if Button "Edit Accounts" clicked
                connect to user.db
                Select Username, Password, Privileges where Username = current user
                Alter values to NewName, NewPassword, NEwPassword

        if Button "Remove" clicked
                connect to user.db
                Remove Username, Password, Privileges where Username = current user


        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

Account Pseudo

The pseudo code for the account screen is very simple in its nature. The first thing it does is to create the screen it's self this is done by loading up loads of widgets which are listed below this.

Under this a new variable is declared "NewName" which is the entry field "Account Name". After this another new variable is declared this time being "NewPrivilleges" which is the entry field called "Privileges". The final variable to be declared is the "NewPassword" variable which is the entry field called "Password". All of these entry fields are loaded above.

The three if statements listed below outline what happens when each of the button widgets is clicked.

The expected output from these functions would be:

- To return to the previous page for the back button
- To remove the current users from the database for the remove budget
- To edit the account details for the Edit button

Design of the Data Structure

Stock	
ID	
Item Name	TEXT NOT NULL
Item No	TEXT NOT NULL
Prices	INTEGER NOT NULL
Description	TEXT NOT NULL

Sign In	
ID	
Account Name	TEXT NOT NULL
Password	TEXT NOT NULL
Privileges	TEXT NOT NULL

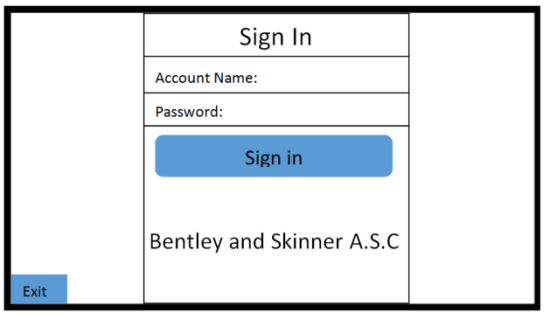
The design of the database is very simple as there are only a few entities that need to be added in the tables. Due to these tables having no common key they will remain completely separated from each other with no relationships between them. All of the fields will have the NOT NULL attribute as they can't be left empty as if this is done there may be problems with the retrieval function. Most of the entities in the database will have the TEXT type applied to them to ensure that this is the only thing accepted by them. The Prices entity has an INTEGER type as the price entity is made up of lots of numbers.

Using this design the database will be able to be constructed either by a DBMS (Data Base Management System) or by SQL commands. These are commands such as "Create Table" as well as "INSERT" as these will both create the table as well as add the values needed to the database. These values can then be retrieved by the "SELECT" SQL commands and displayed to the user. This forms an essential part of the both the Search function and the Log In function. This is as the retrieval of such information is needed as a core part of these functions. This can be seen within the pseudo code that has been given above.

Developing the Solution

Prototyping

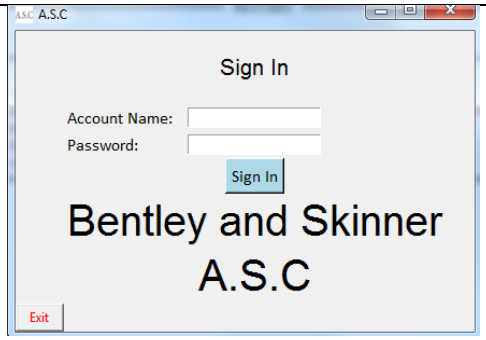
Sign In



Concept of Sign In

```
Create Sign IN Screen
        Label "Sign In"
        Label "Account Name"
        Entry "Account Name"
        Label "Password"
        Entry "Password"
        Label "Bentley and Skinner A.S.C"
        Button "Sign In"
        Button "Exit"

        CurrentUser<-Entry "Account Name"
        CurrentPassword<-Entry "Password"

        if Button "Sign In" clicked
                connect to user.db
                if Username==CurrentUser
                        if Password==CurrentPassword
                                Remove all Widgets
                                Open Frame "Menu"
                        else
                        print "Your UserName/Password was wrong"
                else
                print "Your UserName/Password was wrong"

        if Button "Exit" clicked
                End All Processes
```

Pseudo Code for the Concept of Sign In



Final Iteration of Sign In

```
def creatSignIn():
    def removeSign():
        SignLabel.grid_forget()
        SignIn.grid_forget()
        UserLabel.grid_forget()
        PwordLabel.grid_forget()
        User.grid_forget()
        Pword.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        mainMenu()

    def findUN():
        conn = sqlite3.connect('signin.db')
        UN=userInput.get()
        output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %UN)
        for row in output:
            Name=row[1]
            if UN==Name:
                global disUN
                disUN=Name
                print ("UserName correct")
                findPW()
                conn.close()
                return disUN
            else:
                print ("UserName incorrect")
                print ("Your Username or Password is incorrect")
                conn.close()
                return
        print ("UserName incorrect")
        print ("Your Username or Password is incorrect")
        conn.close()

    def findPW():
        conn = sqlite3.connect('signin.db')
        PW=pwordInput.get()
        output = conn.execute("SELECT*from SignIn WHERE Password='%s'" %PW
        for row in output:
            Pass=row[2]
            if PW==Pass:
                print ("Password correct")
                removeSign()
                conn.close()
                return PW
            else:
                print ("Password incorrect")
                print ("Your Username or Password is incorrect")
                conn.close()
                return
        print ("Password incorrect")
        print ("Your Username or Password is incorrect")
        conn.close()
    global ExitASC
    ExitASC = Button(self, width=5, fg="red", text="Exit")
    ExitASC["command"] =  self.quit
    ExitASC.grid (sticky=W,row=10, column=1)

    SignLabel= Label(self, text="Sign In", font="calibri, 16", pady="20")
    SignLabel.grid (row=1, column=2)

    UserLabel = Label(self, text="Account Name:", font="calibri")
    UserLabel.grid (sticky=W,row=2,column=2)

    PwordLabel = Label(self, text="Password:", font="calibri")
    PwordLabel.grid (sticky=W,row=3,column=2)

    userInput=StringVar()

    pwordInput=StringVar()

    User = Entry(self,textvariable=userInput)
    User.grid (row=2, column=2)

    Pword= Entry(self,textvariable=pwordInput)
    Pword.grid (row=3, column=2)

    SignIn=Button(self, text="Sign In",  bg="light blue", font="calibri", command=findUN)
    SignIn.grid (row=4, column=2)

    title1=Label(self, text="Bentley and Skinner", font="calibri, 30")
    title1.grid (row=6, column=2)

    title2=Label(self, text="A.S.C", font="calibri, 30")
    title2.grid (row=7, column=2)

    space=Label(self,width=3)
    space.grid (row=8, column=3)
```

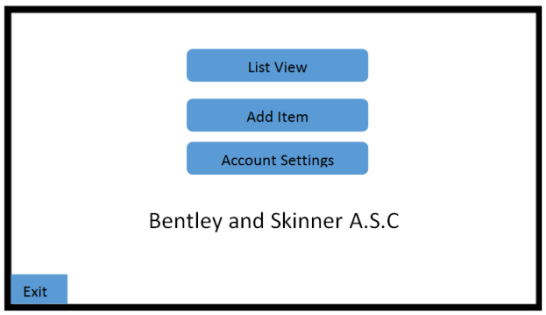
Code for Final Iteration of Sign In

Evaluation of the Iteration of Sign In

Through the iteration of the Sign In page very little changed from the concept. The main differences are mainly graphical such as the size of the font used, the colour of the buttons and the position of the ASC text. These changes were made to the client's specification as the original layout and colours once implemented were sub-standard. The font size was increased as on a larger screen the Title, Account Name, Password and the program name were all the same size. This lead to the load out being very confusing at first though now with the larger fonts in use it is easier to see where you have to go to sign in without having to read the text. This is also why the exit button's font colour was changed to red as it allowed exiting to become more instinctive. The sign in buttons size was also reduced as any large and it would fill most of the screen as well as having text the same size or larger than the title.

In terms of the code the final implementation has received many upgrades to the original pseudo as many factors were not included or addresses originally. For example the final code has had to have many lines of code implemented for future screens to be able to retrieve information. This can be seen in the function "findUN" where a global variable is declared in order for the current username to be easily retrieved.

Menu

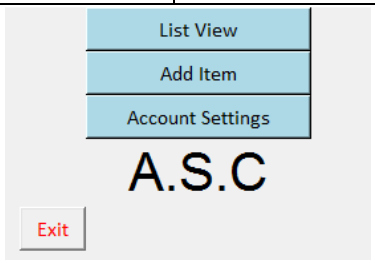


Main Menu

Concept of Menu

```
Create Menu Screen
        Button"List View"
        Button "Add Item"
        Button "Edit Accounts"
        Label "Bentley and Skinner A.S.C"
        Button "Exit"

        if Button "List View" clicked
                Remove all widgets
                Open Frame "List View"

        if Button "Add Item" clicked
                Remove all widgets
                Open Frame "Add Item"

        if Button "Edit Accounts" clicked
                Remove all widgets
                Open Frame "Account Settings"

        if Button "Exit" clicked
                End All Processes
```

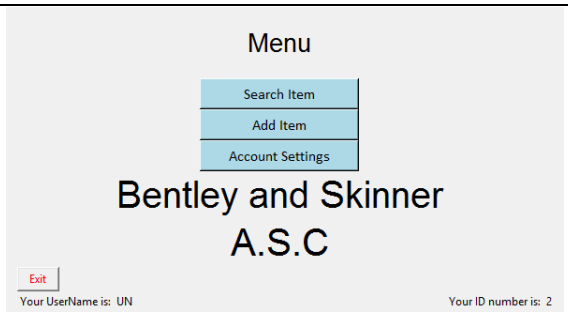
Pseudo Code from Menu Concept



First Iteration of Menu

```
class Menu(Frame):
    def createWidgets(self):

        Exit = Button(self, width=5, fg="red", text="Exit", font="calibri")
        Exit["command"] =  self.quit
        Exit.grid (sticky=W, row=10, column=2)

        Menu = Label(self,text="Menu", pady=20, font="calibri, 20")
        Menu.grid (row=1, column=2)

        List = Button(self, width=20, bg="light blue", text="List View", font="calibri")
        List.grid (row=2, column=2)

        Add = Button(self, width=20, bg="light blue", text="Add Item", font="calibri")
        Add.grid (row=3, column=2)

        Account = Button(self, width=20, bg="light blue", text="Account Settings", font="calibri")
        Account.grid (row=4, column=2)

        title1=Label(self, text="A.S.C", font="calibri, 30")
        title1.grid (row=7, column=2)
```

Code For First Iteration of the Menu



Final Iteration of Menu

```
def mainMenu():
    def removeAccountMenu():
        Back.grid_forget()
        Title.grid_forget()
        account.grid_forget()
        priveleges.grid_forget()
        pword.grid_forget()
        RemoveRemoveButton()

    def accountToMenu():
        account.grid_forget()
        priveleges.grid_forget()
        pword.grid_forget()
        RemoveRemoveButton()
        showMenu()

    def removeAccount():
        Back.grid_forget()
        ID.grid_forget()
        idLabel.grid_forget()
        account.grid_forget()
        accountLabel.grid_forget()
        privileges.grid_forget()
        privilegesLabel.grid_forget()
        pword.grid_forget()
        pwordLabel.grid_forget()
        Edit.grid_forget()
        Remove.grid_forget()
        Inst.grid_forget()
        showMenu()

    def removeMenu():
        ExitASC.grid_forget()
        List.grid_forget()
        Add.grid_forget()
        Account.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        Menu.grid_forget()

    def showMenu():
        mainMenu()

    def getID():
        conn = sqlite3.connect('signin.db')
        output = conn.execute("SELECT ID from SignIn WHERE Name='%s'" %disUN)
        for row in output:
            ID=row[0]
        return ID

    ExitASC.grid_forget()
    ExitMenu = Button(self, width=5, fg="red", text="Exit")
    ExitMenu["command"] =  self.quit
    ExitMenu.grid (sticky=W,row=10, column=1)
    Menu = Label(self,text="Menu", pady=20, font="calibri, 20")
    Menu.grid (row=1, column=2)

    List = Button(self, width=20, bg="light blue", text="Search Item", font="calibri", command=makeList)
    List.grid (row=2, column=2)

    Add = Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=makeAdd)
    Add.grid (row=3, column=2)

    Account = Button(self, width=20, bg="light blue", text="Account Settings", font="calibri", command=makeAccountMenu)
    Account.grid (row=4, column=2)

    ID1=Label (self, text="Your ID number is:")
    ID1.grid (row=11, column=3, sticky=E)

    ID2=Label (self, text=getID())
    ID2.grid (row=11, column=4, sticky=W)

    UN1=Label (self, text="Your UserName is:")
    UN1.grid (row=11, column=1, sticky=E)

    UN2=Label (self, text=disUN)
    UN2.grid (row=11, column=2, sticky=W)

    title1=Label(self, text="Bentley and Skinner", font="calibri, 30")
    title1.grid (row=6, column=2)

    title2=Label(self, text="A.S.C", font="calibri, 30")
    title2.grid (row=7, column=2)
```

Code For Final Iteration of the Menu

Evaluation of the Iteration of Menu

The main menu of the ASC program was the easiest to implement as it has the simplest functions and interface. The main interface has mainly over gone small graphical changes to better display information to the user. For example once signed in the user now has their User name as well as there as well as their ID number. Both of these are collected from the sign in database and the displayed to the user. Another change that has occurred has been the implementation of the Bentley and Skinner name as well as the title of Menu. This is in order to make the program to seem more personalised for the client. The implementation of the Menu title is so that when the user is flicking through the different screens they always know where they are. This is in order to make the

program easier to navigate for people who are less common with computers. The Exit Button has also been reduced in size as original it was the same thickness as the buttons which made it seem as if it was part of the original list of options. Therefore by reducing its size and putting it further into the corner it seems less in the way and obvious though still easily navigable.

However creating the means to change what was being shown was a challenge but also very important as this function was then replicated at least once on every other screen. The function started off by trying to put every single screen into a frame which would then be called by their parent page which could also be returned to if need be. The problem with this came with implementing new pages as well as sending information between the two pages. Therefore this prototype was abandoned. The second iteration of page switching involved having each screen open in a separate window. Though after long use this lead to a cluttered work space and one which was moving across the screen as each one opened. Extended use also lead to a high use in Ram if multiple windows were open. This made it also harder to see what page you were on and to navigate through them. This therefore contradicted part of my specification which was to make it easy to navigate due to the limited computer experience of some of the users. The third and final iteration of screen switching involves each screen being its own function which can then be called by a press of a button. This method allows easy implementation of new pages as they are placed within their parent function and then called by the intended button. Also due them being within one main ASC function it is a lot easier to send information across them. The main issue with this process is that all UI widgets have to be removed individually if one needs to be retained e.g. the Exit button, User Name information and ID number. This is done with the variables name followed by .grid_forget(). Due to the UI being made with tkinter this made the whole process a lot easier of creating a screen more logical and simpler. Due to this simple functions could be created at the start of each screen to remove all of the widgets from the screen.

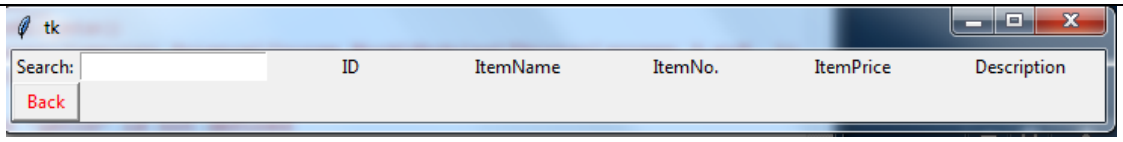
List View

List View

Concept of Stock Checker

```
Create List view Screen
        Entry "Search"
        Label "Stock"
        Label "Item No."
        Label "Price"
        Button "Back"

        NewSearch<-Entry "Search"

        if Button "Search" clicked
                connect to stock.db
                cursor<-Select Stock, Price, Description where ItemNo or Stock = Search
                for rows in cursor =n
                x=0
                if x=<n
                        label "Stock[n]"
                        label "Price[n]"
                        label "Description[n]"
                        button "Edit"
                        x+1
                else:
                        return

        if Button "Edit" clicked
                Remove all Widgets
                Open Frame "Add Item"

        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

Pseudo Code for Stock Checker



First Iteration of Stock Checker

```
def createWidgets(self):
    def getSearch():
        def getID():
            SI=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%SI)
            for row in cursor:
               ID=row[0]


        def getName():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
            for row in cursor:
               Name = row[0]

        def getNo():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
            for row in cursor:
               NO = row [2]


        def getPrice():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
            for row in cursor:
               Price = row[3]


        def getDesc():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
            for row in cursor:
               Desc = row[4]

        getID()
        getNo()
        getName()
        getPrice()
        getDesc()

    Back = Button(self, width=5, fg="red", text="Back")
    Back.grid (sticky=W,row=100, column=0)

    searchInput=StringVar()

         searchTitle=Label (self, text="Item Search", font="calibri, 16")
         searchTitle.grid (column=1,row=1)

         search=Entry(self,textvariable=searchInput,command=getSearch)
         search.grid (sticky=W,column=1,row=2)

         searchLabel=Label(self, text="Search:")
         searchLabel.grid (sticky=W,column=0, row=2)

         ID=Label(self,width=15,text="ID")
         ID.grid (sticky=W,column=2,row=2)

         Name= Label(self,width=15,text="ItemName")
         Name.grid(sticky=W,column=3,row=2)

         NO=Label(self,width=15,text="ItemNo.")
         NO.grid(sticky=W,column=4,row=2)

         Price=Label(self,width=15,text="ItemPrice")
         Price.grid(sticky=W,column=5,row=2)

         Desc=Label(self,width=30,text="Description")
         Desc.grid(sticky=W,column=6,row=2)
```

Code for First Iteration of Stock Checker

Final Iteration of Stock Checker Before Search

Final Iteration of Stock Checker After Search

```
def makeList():
    List.grid_forget()
    Add.grid_forget()
    Account.grid_forget()
    title1.grid_forget()
    title2.grid_forget()
    ExitMenu.grid_forget()
    Menu.grid_forget()

    def MakeSubEdit()......|

    def removeList():
        Back.grid_forget()
        search.grid_forget()
        searchLabel.grid_forget()
        Name.grid_forget()
        NO.grid_forget()
        Price.grid_forget()
        Desc.grid_forget()
        RestockL.grid_forget()
        Edit.grid_forget()
        searchTitle.grid_forget()
        Clear.grid_forget()
        showMenu()

    def notify():
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login("AutomaticStockChecker@gmail.com", "ASC12345678")
        entry=searchInput.get()
        conn = sqlite3.connect('ASC.db')
        cursor= conn.cursor()
        cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
        row = cursor.fetchone()
        print (row)
        f=0
        if row==None:
            print("Search is empty")
            f+1
            return
        else:
            while f==0:
               Name = row[1]
            msg = "Notification From A.S.C", Name, "Needs to be restocked as it is running low or is out of stock"
            server.sendmail("AutomaticStockChecker@gmail.com", "AutomaticStockChecker@gmail.com", msg)
            server.quit()
            print ("Email sent")

    def getSearch():
        def clear():
            aRestock.grid_forget()
            aEdit.grid_forget()
            aName.grid_forget()
            aNO.grid_forget()
            aPrice.grid_forget()
            aDesc.grid_forget()
            aClear.grid_forget()

        def toSubEdit():
            Back.grid_forget()
            searchTitle.grid_forget()
            search.grid_forget()
            searchLabel.grid_forget()
            Name.grid_forget()
            NO.grid_forget()
            Price.grid_forget()
            Desc.grid_forget()
            Clear.grid_forget()
            RestockL.grid_forget()
            Edit.grid_forget()
            clear()
            searchTitle.grid_forget()
            makeSubEdit()

        def getName():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor= conn.cursor()
            cursor = conn.execute("SELECT ItemName from Stock WHERE ItemName LIKE '%s'"%entry)
            if cursor==None:
                Name="No Such Item"
                return Name
            else:
                for row in cursor:
                    Name=row[0]
                    return Name


        def getNo():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor= conn.cursor()
            cursor = conn.execute("SELECT ItemNo from Stock WHERE ItemName LIKE '%s'"%entry)
            if cursor==None:
                No="No Such Item"
                return No
            else:
                for row in cursor:
                    No=row[0]
                    return No



        def getPrice():
            entry=searchInput.get()
            conn = sqlite3.connect('ASC.db')
            cursor= conn.cursor()
            cursor = conn.execute("SELECT Price from Stock WHERE ItemName LIKE '%s'"%entry)
            if cursor==None:
                Price="No Such Item"
                return Price
```

```
        else:
            for row in cursor:
                Price=row[0]
                return Price

    def getDesc():
        entry=searchInput.get()
        conn = sqlite3.connect('ASC.db')
        cursor= conn.cursor()
        cursor = conn.execute("SELECT Description from Stock WHERE ItemName LIKE'%s'"%entry)
        if cursor==None:
            Desc="No Such Item"
            return Desc
        else:
            for row in cursor:
                Desc=row[0]
                return Desc

    getNo()
    getName()
    getPrice()
    getDesc()

    aName= Label(self,width=15,text=getName())
    aName.grid(sticky=W,column=2,row=3)

    aNO=Label(self,width=15,text=getNo())
    aNO.grid(sticky=W,column=3,row=3)

    aPrice=Label(self,width=15,text=getPrice())
    aPrice.grid(sticky=W,column=4,row=3)

    aDesc=Label(self,width=30,text=getDesc())
    aDesc.grid(sticky=W,column=5,row=3)

    aRestock=Button(self,width=14,text="Restock", command=notify)
    def getDesc():
        entry=searchInput.get()
        conn = sqlite3.connect('ASC.db')
        cursor= conn.cursor()
        cursor = conn.execute("SELECT Description from Stock WHERE ItemName LIKE'%s'"%entry)
        if cursor==None:
            Desc="No Such Item"
            return Desc
        else:
            for row in cursor:
                Desc=row[0]
                return Desc

    getNo()
    getName()
    getPrice()
    getDesc()

    aName= Label(self,width=15,text=getName())
    aName.grid(sticky=W,column=2,row=3)

    aNO=Label(self,width=15,text=getNo())
    aNO.grid(sticky=W,column=3,row=3)

    aPrice=Label(self,width=15,text=getPrice())
    aPrice.grid(sticky=W,column=4,row=3)

    aDesc=Label(self,width=30,text=getDesc())
    aDesc.grid(sticky=W,column=5,row=3)

    aRestock=Button(self,width=14,text="Restock", command=notify)
    aRestock.grid(sticky=W,column=6,row=3)

    aEdit=Button(self,width=14,text="Edit", command=toSubEdit)
    aEdit.grid(sticky=W,column=7,row=3)

    aClear=Button(self,width=14,text="Clear Search", command=clear)
    aClear.grid(sticky=W,column=8,row=3)


Back = Button(self, width=5, fg="red", text="Back", command=removeList)
Back.grid (sticky=W,row=100, column=0)

searchInput=StringVar()

searchTitle=Label (self, text="Item Search", font="calibri, 16")
searchTitle.grid (column=1,row=1)

search=Entry(self,textvariable=searchInput)
search.grid (sticky=W,column=1,row=2)

searchLabel=Button(self, text="Search:",command=getSearch)
searchLabel.grid (sticky=W,column=0, row=2)

Name= Label(self,width=15,text="ItemName",bg="grey")
Name.grid(sticky=W,column=2,row=2)

NO=Label(self,width=15,text="ItemNo.",bg="grey")
NO.grid(sticky=W,column=3,row=2)

Price=Label(self,width=15,text="ItemPrice",bg="grey")
Price.grid(sticky=W,column=4,row=2)
```

```
Desc=Label(self,width=30,text="Description",bg="grey")
Desc.grid(sticky=W,column=5,row=2)

RestockL=Label(self,width=15,text="Restock",bg="grey")
RestockL.grid(sticky=W,column=6,row=2)

Edit=Label(self,width=15,text="Edit",bg="grey")
Edit.grid(sticky=W,column=7,row=2)

Clear=Label(self,width=15,text="Clear",bg="grey")
Clear.grid(sticky=W,column=8,row=2)
```

Code for Final Iteration of Stock Checker

Evaluation of the Iteration of List View

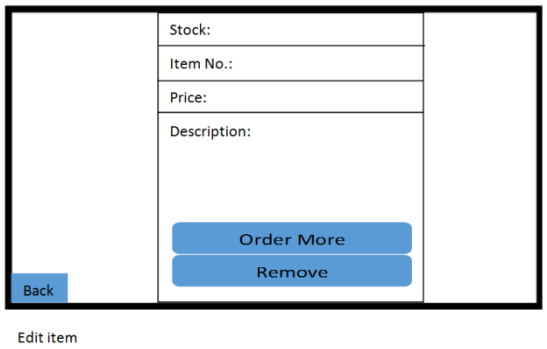
This screen has received the second biggest change out of all of the screen implementations. This as in concept it was meant to be a list view of all items within the data base. However from my research it was clear that this would be a difficult task to achieve. This is as the database would need to include upwards of 2000 entities which displaying would lead to large amount of RAM use and making it difficult to navigate as well as displaying even in a table with a scroll bar. If this was to be done a search bar would need to be implemented. However this would then lead to all 2000 plus entities being loaded and then a large scroll bar having to be implemented. This would make my program increasingly hard to navigate through. This lead to the biggest change occurring being the removal of the table and the implementation of the search function. This means the user has to search up what they wish to view before anything loads. Therefore this will decrease the RAM used, as well as increase the ease of navigating the program.

However in the first iteration there was no way to edit the item that had been searched this is as the screen simply only retrieved the items ID, Name, No, Price and Description. This is as at this current state in time the screens were all separated. This lead to there being no way to edit an entry that you didn't want in the stock database as well as no way to notify yourself of an entry being low and clearing the search. However in the final iteration all of these features have been added. This is as three new columns have been added where, once searched, the ability to restock an item, edit it or clear it will appear. This can be seen in "Final Iteration of Stock Checker After Search". Clicking restock will run the function "Notify" which will send an automated email to the ASC Notification address allowing a log to be built up. The ability to do this is one of the specification points mentioned in my design. When the edit button is clicked "makeSubEdit" is run, this function creates the Edit Item page, the code and screenshots of which can be viewed in the Add Item section below. The Clear button is used to remove the current search displayed and allow you to search for another item.

Another change that occurred was the removal of the ID column. The main reason for this was the restructuring of the data base. This is as the main reason for its existence was for it to serve as the primary key of the database. However ID and Item No could both be used for the same purpose and created a redundancy. This is the client would then have to lots of ID's for a product. Therefore ID was removed and replaced with Item code. This uses the stores Item Codes so that a second system doesn't have to be integrated. This is important as according to the specification the program must make the business run better. If they had to integrate another numbering system into their stock then this would go against this specification point.

Another improvement is the change in the colour of the columns. The reason for this is to better distinguish the headers form the rest of the table. This is as a piece of feedback I received was that it was difficult to see the individual rows. This is why it lead to me changing the background colour to a dark grey. Evidence for this feedback can be viewed in the feedback section. As for the other pages another graphical update was adding in a title displaying "Item Search" in order to ease navigation and for the user to see where they are at.

Add Item

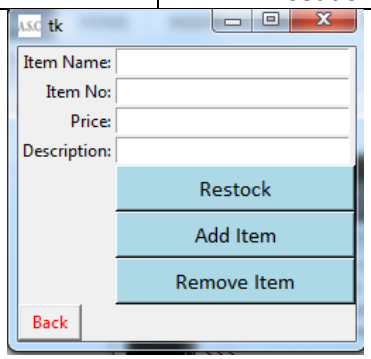


Edit item

Concept of Add Item

```
Create Add Item Screen
        Label "Stock"
        Entry "Stock"
        Label "Item No."
        Entry "Item No."
        Label "Price"
        Entry "Price"
        Label "Description"
        Entry "Description"
        Button "Order More"
        Button "Remove"
        Button "Back"

        NewName<-Entry "Stock"
        No<-Entry "Item No"
        NewPrice<-Entry "Price"
        NewDescription<-Entry "Description

        if Button "OrderMore" clicked
                Open "Notifications.docx"
                Write NewName,"Is running low or is out of stock"
                Close file


        if Button "Remove" clicked
                connect to stock.db
                Remove Stock, ItemNo, Price, Description where ItemNo = NO


        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

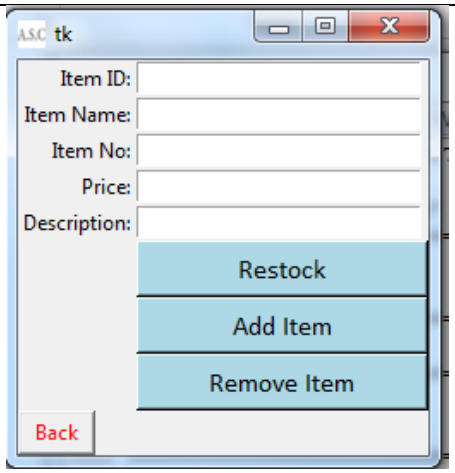
Pseudo code for Add Item Concept



First Iteration of Add Item

```
def createWidgets(self):
        def addItem():
            Name=NameInput.get()
            No=NoInput.get()
            Price=PriceInput.get()
            Desc=DescInput.get()
            conn = sqlite3.connect('ASC.db')
            conn.execute("INSERT INTO Stock (ID,ItemName,ItemNo,Prices,Description) \
                  VALUES (ItemName,ItemNo,Prices,Desc)");
            conn.commit()


        NameInput=StringVar()
        NoInput=StringVar()
        PriceInput=StringVar()
        DescInput=StringVar()

        Back = Button(self, width=5, fg="red", text="Back")
        Back["command"] =  self.quit
        Back.grid (sticky=W,row=10, column=1)

        item= Entry(self, width=27,textvariable=NameInput)
        item.grid (sticky=W,column=2,row=1)

        itemLabel= Label(self, text="Item Name:")
        itemLabel.grid (sticky=E,column=1, row=1)

        itemNo= Entry(self, width=27,textvariable=NoInput)
        itemNo.grid (sticky=W,column=2,row=2)

        itemNoLabel= Label(self, text="Item No:")
        itemNoLabel.grid (sticky=E,column=1, row=2)

        price= Entry(self, width=27,textvariable=PriceInput)
        price.grid (sticky=W,column=2,row=3)

        priceLabel= Label(self, text="Price code:")
        priceLabel.grid (sticky=E,column=1, row=3)

        description= Entry(self, width=27,textvariable=DescInput)
        description.grid (sticky=W,column=2,row=4)

        descriptionLabel= Label(self, text="Description:")
        descriptionLabel.grid (sticky=E,column=1, row=4)


        Restock= Button(self, width=20, bg="light blue", text="Restock", font="calibri")
        Restock.grid (sticky=E,column=2,row=6)

        Add= Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=addItem)
        Add.grid (sticky=E,column=2,row=7)

        Remove= Button(self, width=20, bg="light blue", text="Remove Item", font="calibri")
        Remove.grid (sticky=E,column=2,row=8)
```

Code of First Iteration of Add Item



Second Iteration of Add Item

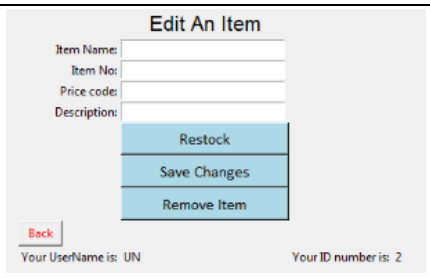
```
def createWidgets(self):
    def addItem():
        Name=NameInput.get()
        No=NoInput.get()
        Price=PriceInput.get()
        Desc=DescInput.get()
        conn = sqlite3.connect('ASC.db')
        conn.execute("INSERT INTO Stock (ID,ItemName,ItemNo,Prices,Description) \
              VALUES (ItemName,ItemNo,Prices,Desc)");
        conn.commit()


    IDInput=StringVar()
    NameInput=StringVar()
    NoInput=StringVar()
    PriceInput=StringVar()
    DescInput=StringVar()

    Back = Button(self, width=5, fg="red", text="Back")
    Back["command"] =  self.quit
    Back.grid (sticky=W,row=10, column=1)

    itemID= Entry(self, width=27,textvariable=IDInput)
    itemID.grid (sticky=W,column=2,row=1)

    itemIDLabel= Label(self, text="Item ID:")
    itemIDLabel.grid (sticky=E,column=1, row=1)

    itemName= Entry(self, width=27,textvariable=NoInput)
    itemName.grid (sticky=W,column=2,row=2)

    itemNameLabel= Label(self, text="Item Name:")
    itemNameLabel.grid (sticky=E,column=1, row=2)

    itemNo= Entry(self, width=27,textvariable=NoInput)
    itemNo.grid (sticky=W,column=2,row=3)

    itemNoLabel= Label(self, text="Item No:")
    itemNoLabel.grid (sticky=E,column=1, row=3)

    price= Entry(self, width=27,textvariable=PriceInput)
    price.grid (sticky=W,column=2,row=4)

    priceLabel= Label(self, text="Price:")
    priceLabel.grid (sticky=E,column=1, row=4)

    description= Entry(self, width=27,textvariable=DescInput)
    description.grid (sticky=W,column=2,row=5)

    descriptionLabel= Label(self, text="Description:")
    descriptionLabel.grid (sticky=E,column=1, row=5)


    Restock= Button(self, width=20, bg="light blue", text="Restock", font="calibri")
    Restock.grid (sticky=E,column=2,row=6)

    Add= Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=addItem)
    Add.grid (sticky=E,column=2,row=7)

    Remove= Button(self, width=20, bg="light blue", text="Remove Item", font="calibri")
    Remove.grid (sticky=E,column=2,row=8)
```

Code of Second Iteration of Add Item

Final Iteration of Add Item

```
def makeAdd():
    List.grid_forget()
    Add.grid_forget()
    Account.grid_forget()
    title1.grid_forget()
    title2.grid_forget()
    Menu.grid_forget()
    def removeAdd():
        BackAdd.grid_forget()
        item.grid_forget()
        itemLabel.grid_forget()
        itemNo.grid_forget()
        itemNoLabel.grid_forget()
        price.grid_forget()
        priceLabel.grid_forget()
        description.grid_forget()
        descriptionLabel.grid_forget()
        Addbtn.grid_forget()
        AddTitle.grid_forget()
        showMenu()

    def addItem():
        ID=IDInput.get()
        Name=NameInput.get()
        No=NoInput.get()
        Price=PriceInput.get()
        Desc=DescInput.get()
        conn = sqlite3.connect('ASC.db')
        cursor=conn.execute("SELECT*from Stock")
        row=cursor.fetchall()
        print (row)
        conn.execute ("""INSERT INTO Stock VALUES ('%s', '%s', '%s', '%s') """, (Name, No, Price, Desc))
        conn.commit()

    def getName():
        item=NameInput.get()
        print (NameInput.get())
        return item

    NameInput=StringVar()
    NoInput=StringVar()
    PriceInput=StringVar()
    DescInput=StringVar()
    IDInput=StringVar()

    BackAdd= Button(self, width=5, fg="red", text="Back", command=removeAdd)
    BackAdd.grid (sticky=W,row=10, column=1)

    AddTitle= Label(self, text="Add Item", font="calibri, 16", pady="20")
    AddTitle.grid (row=1,column=2)

    item= Entry(self, width=27,textvariable=NameInput)
    item.grid (sticky=W,column=2,row=2)
    itemLabel= Label(self, text="Item Name:")
    itemLabel.grid (sticky=E,column=1, row=2)

    itemNo= Entry(self, width=27,textvariable=NoInput)
    itemNo.grid (sticky=W,column=2,row=3)

    itemNoLabel= Label(self, text="Item Code:")
    itemNoLabel.grid (sticky=E,column=1, row=3)

    price= Entry(self, width=27,textvariable=PriceInput)
    price.grid (sticky=W,column=2,row=4)

    priceLabel= Label(self, text="Price:")
    priceLabel.grid (sticky=E,column=1, row=4)

    description= Entry(self, width=27,textvariable=DescInput)
    description.grid (sticky=W,column=2,row=5)

    descriptionLabel= Label(self, text="Description:")
    descriptionLabel.grid (sticky=E,column=1, row=5)

    Addbtn= Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=addItem)
    Addbtn.grid (sticky=E,column=2,row=6)
```

Code for Final Iteration of Add Item



Final Iteration of Edit Item

```
def makeSubEdit():
    print("Welcome to Sub Edit")
    def removeEdit():
        BackEdit.grid_forget()
        SubTitle.grid_forget()
        item.grid_forget()
        itemLabel.grid_forget()
        itemNo.grid_forget()
        itemNoLabel.grid_forget()
        price.grid_forget()
        priceLabel.grid_forget()
        description.grid_forget()
        descriptionLabel.grid_forget()
        Restock.grid_forget()
        Addbtn.grid_forget()
        Remove.grid_forget()
        makeList()

    def notify():
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login("AutomaticStockChecker@gmail.com", "ASC12345678")
        getName()
        msg = "Notification From A.S.C", item, "Needs to be restocked as it is running low or is out fo stock"
        server.sendmail("AutomaticStockChecker@gmail.com", "AutomaticStockChecker@gmail.com", msg)
        server.quit()
        print ("Email sent")

    def addItem():
        getName()
        getNo()
        getDesc()
        getPrice()
        conn = sqlite3.connect('ASC.db')
        conn.execute("INSERT INTO Stock (ItemName,ItemNo,Prices,Description) \
        VALUES (Name,No,Prices,Desc)");
        conn.commit()

    def Remove():
        getName()
        getNo()
        getDesc()
        getPrice()
        conn = sqlite3.connect('ASC.db')
        conn.execute("REMOVE FROM Stock WHERE ID='%s'" %ID)
        conn.commit()

    def getName():
        entry=NameInput.get()
        if entry==None:
            Name="No Such Item"
            return Name
        else:
            Name=entry
```

```
        print (Name)
        return Name


def getNo():
    entry=NoInput.get()
    if entry==None:
        No="No Such Item"
        return No
    else:
        No=entry
        print (No)
        return No



def getPrice():
    entry=PriceInput.get()
    if entry==None:
        Price="No Such Item"
        return Price
    else:
        Price=entry
        print (Price)
        return Price

def getDesc():
    entry=DescInput.get()
    if entry==None:
        Desc="No Such Item"
        return Desc
    else:
        Desc=entry
        print (Desc)
        return Desc

 NameInput=StringVar()
 NoInput=StringVar()
 PriceInput=StringVar()
 DescInput=StringVar()

 BackEdit= Button(self, width=5, fg="red", text="Back", command=removeEdit)
 BackEdit.grid (sticky=W,row=10, column=1)

 SubTitle=Label (self, text="Edit An Item", font="calibri, 16")
 SubTitle.grid (column=2,row=1)

 item= Entry(self, width=27,textvariable=NameInput)
 item.grid (sticky=W,column=2,row=2)

 itemLabel= Label(self, text="Item Name:")
 itemLabel.grid (sticky=E,column=1, row=2)

 itemNo= Entry(self, width=27,textvariable=NoInput)
 itemNo.grid (sticky=W,column=2,row=3)

 itemNoLabel= Label(self, text="Item No:")
 itemNoLabel.grid (sticky=E,column=1, row=3)

 price= Entry(self, width=27,textvariable=PriceInput)
 price.grid (sticky=W,column=2,row=4)

 priceLabel= Label(self, text="Price code:")
 priceLabel.grid (sticky=E,column=1, row=4)

 description= Entry(self, width=27,textvariable=DescInput)
 description.grid (sticky=W,column=2,row=5)

 descriptionLabel= Label(self, text="Description:")
 descriptionLabel.grid (sticky=E,column=1, row=5)

 Restock= Button(self, width=20, bg="light blue", text="Restock", font="calibri", command=notify)
 Restock.grid (sticky=E,column=2,row=6)

 Addbtn= Button(self, width=20, bg="light blue", text="Save Changes", font="calibri", command=addItem)
 Addbtn.grid (sticky=E,column=2,row=7)

 Remove= Button(self, width=20, bg="light blue", text="Remove Item", font="calibri")
 Remove.grid (sticky=E,column=2,row=8)
```

Code for Final Iteration Of Edit Item

Evaluation of the Iteration of Add Item

The Add item screen has had one of the strangest iteration processes due to the creation of the Edit Item screen where some of the original functions were moved to. The concept for this screen was for it to be used to add to an items content, edit them, remove that item as well as be able to order more. Editing and adding would have occurred through changing information displayed in the text boxes. However within the first iteration this was changed to having to click a button to send the request of alteration. However the layout and naming of the labels, text boxes and buttons was pretty much the same.

From the first iteration to the second there was only one change. Though within this time the original database for the stock was created. As explained above in the Stock Checker screen the original database used an Item ID as primary key in order to better organise the structure of it and reduce redundancies. Because of this the Item ID field was added. Though this soon raised the problem of how will the client know the ID number for an item that hasn't yet been added. This lead to the changes that can be viewed above in the Stock Checker Screen.

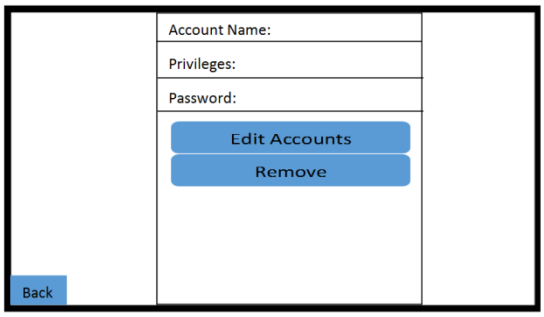
However on the third iteration of this screen it was as simple as removing the Label and Field of the Item ID. The third iteration received the biggest overhaul as it lead to the creation of a second screen and the migration of 60% of the functions to the new separated screen, Edit Item. This Edit item screen is referred to in the code as the "SubEdit" screen under the function "makeSubEdit" in the "Stock Checker" screens code. There were two main reasons for this change. One was that the previous "Stock Checker" screen needed the ability for the user to edit the current item but not add a new one. The second was that when adding an item some of the functions would be redundant. For example removing an item before it has been created is impossible and therefore makes that feature redundant. Therefore the only two functions remaining were the ability to add the item and return to the previous page. Also a title displaying "Add Item" inserted in order to help the client navigate their program.

The Edit Item page went through only one iteration. This is as it simply had its features transferred from the previous page. In this final version it shares many similarities to its predecessor the Add Item screen. For example the layout of the buttons is exactly the same as the first and second iteration although the "Add Item" button has been renamed to "Save Changes". This is feedback from my user showed this as confusing. This is possibly due to it being the same text used in the "Menu" screen. This is similar to why I decided to change the "Order More" button from the concept to "Restock" throughout all of the iterations. The Add Item and Edit Item screens also had the title added in the final iterations as with the other screens to help ease navigation.

Throughout all iterations the code has stayed with the same fundamental functions. One of the larger changes has been the introduction of the function dubbed "Notify" within the "Edit Item" screen. This function connects to the Gmail server in order to send a restock notification to the user. This is done by sending an email to the email address "AutomaticStockChecker@gmail.com". This has iterated from the original idea where the message was written to a text file. Now with it being sent to an email a log is able to be built up as well as the client being able to access these notifications from multiple devices. This therefore increases the productivity of the business as they won't have to be on a desktop to see the items out of stock. This also applies to the "Notify" function within the "Stock Checker" as this is where this code was first implemented. However by

having it on both screens it allows the user to not have to go back to the previous page to send the notification and can do it straight from the edit menu.

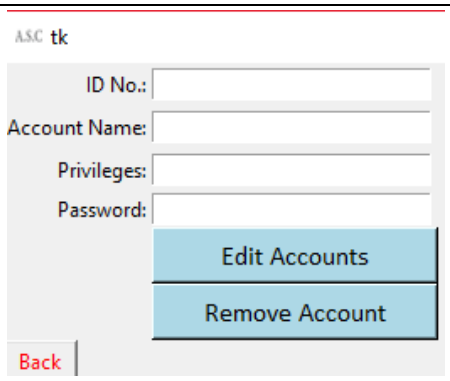
Account Settings



Concept of Account Settings

```
Create Account Edit Screen
        Label "Account Name"
        Entry "Account Name"
        Label "Privileges"
        Entry "Privileges"
        Label "Password"
        Entry "Password"
        Button "Edit Accounts"
        Button "Remove"
        Button "Back"

        NewName<-Entry "Account Name"
        NewPrivileges<-Entry "Privileges"
        NewPassword<-Entry "Password"

        if Button "Edit Accounts" clicked
                connect to user.db
                Select Username, Password, Privileges where Username = current user
                Alter values to NewName, NewPassword, NEwPassword

        if Button "Remove" clicked
                connect to user.db
                Remove Username, Password, Privileges where Username = current user


        if Button "Back" clicked
                Remove all Widgets
                Open Frame "Menu"
```

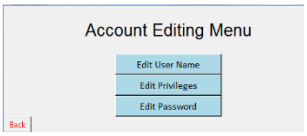
Pseudo Code for the Concept of Account Settings



First Iteration Of Account Edit

```
class Account(Frame):
    def Next(self):
        self.lift()

    def createWidgets(self):

        def getName():
            ID=idInput.get()
            NW=nameInput.get()
            print (NW)
            conn = sqlite3.connect('signin.db')
            output = conn.execute("SELECT*from SignIn WHERE ID='%s'" %ID)
            print (output)
            for row in output:
                Name=row[1]
                if NW==Name:
                    print ("Users Name found")

                    conn.close()
                    return
                else:
                    conn.close()
                    return
            conn.close()

        def getPriv():
            ID=idInput.get()
            PR=privInput.get()
            print (PR)
            conn = sqlite3.connect('signin.db')
            output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %PR)
            for row in output:
                Priv=row[1]
                if PR==Priv:
                    print ("Users Privellages found")

                    conn.close()
                    return
                else:
                    conn.close()
                    return
            conn.close()

        def getPass():
            ID=idInput.get()
            PW=pwordInput.get()
            print (PW)
            conn = sqlite3.connect('signin.db')
            output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %PW)
            for row in output:
                Pass=row[1]
                if PW==Priv:
                    print ("Users Password found")

                    conn.close()
                    return
                else:
                    conn.close()
                    return
        def edit():
            getName()
            getPriv()
            getPass()

        idInput=StringVar()

        pwordInput=StringVar()

        nameInput=StringVar()

        privInput=StringVar()

        Back = Button(self, width=5, fg="red", text="Back")
        Back["command"] = self.quit
        Back.grid (sticky=W,row=10, column=1)

        ID= Entry(self, width=27,textvariable=idInput)
        ID.grid (sticky=W,column=2,row=1)

        idLabel= Label(self, text="ID No.:")
        idLabel.grid (sticky=E,column=1, row=1)

        account= Entry(self, width=27, textvariable=nameInput)
        account.grid (sticky=W,column=2,row=2)

        accountLabel= Label(self, text="Account Name:")
        accountLabel.grid (sticky=E,column=1, row=2)

        privileges= Entry(self, width=27, textvariable=privInput)
        privileges.grid (sticky=W,column=2,row=3)

        privilegesLabel= Label(self, text="Privileges:")
        privilegesLabel.grid (sticky=E,column=1, row=3)

        pword= Entry(self, width=27,textvariable=pwordInput)
        pword.grid (sticky=W,column=2,row=4)

        pwordLabel= Label(self, text="Password:")
        pwordLabel.grid (sticky=E,column=1, row=4)

        Edit= Button(self, width=20, bg="light blue", text="Edit Accounts", font="calibri", command=edit)
        Edit.grid (sticky=E,column=2,row=5)

        Remove= Button(self, width=20, bg="light blue", text="Remove Account", font="calibri", command=warning)
        Remove.grid (sticky=E,column=2,row=6)
```

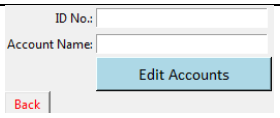
Code Of First Iteration Of Account Edit



Second Iteration Of Account Menu :Menu

```
def makeAccountMenu():

    def showAccountMenu():
        ToMenu.grid_forget()
        Title.grid_forget()
        accountMenu.grid_forget()
        privilegesMenu.grid_forget()
        pwordMenu.grid_forget()

    ToMenu= Button(self, width=5, fg="red", text="Back")
    ToMenu.grid (sticky=W,row=10, column=1)

    Title= Label(self, pady=20, font="calibri, 20", width=27, text="Account Editing Menu")
    Title.grid (column=2,row=1)

    accountMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit User Name", command=NameEdit)
    accountMenu.grid (column=2,row=2)

    privilegesMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit Privileges",command=PrivEdit)
    privilegesMenu.grid (column=2,row=4)

    pwordMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit Password",command=PassEdit)
    pwordMenu.grid (column=2,row=5)

makeAccountMenu()
```

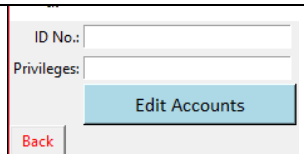
Code Of Second Iteration Of Account Menu :Menu



Second Iteration Of Account Menu :User

```
def createWidgets(self):

    def getName():
        ID=idInput.get()
        NM=nameInput.get()
        print (NM)
        conn = sqlite3.connect('signin.db')
        output = conn.execute("SELECT*from SignIn WHERE ID='%s'" %ID)
        for row in output:
            Name=row[1]
            if NM==Name:
                print ("Users Name found")

                conn.close()
                return
            else:
                conn.close()
                return
        conn.close()

    def edit():
        getName()


    idInput=StringVar()

    nameInput=StringVar()

    Back = Button(self, width=5, fg="red", text="Back")
    Back["command"] =  self.quit
    Back.grid (sticky=W,row=10, column=1)

    ID= Entry(self, width=27,textvariable=idInput)
    ID.grid (sticky=W,column=2,row=1)

    idLabel= Label(self, text="ID No.:")
    idLabel.grid (sticky=E,column=1, row=1)

    account= Entry(self, width=27, textvariable=nameInput)
    account.grid (sticky=W,column=2,row=2)

    accountLabel= Label(self, text="Account Name:")
    accountLabel.grid (sticky=E,column=1, row=2)

    Edit= Button(self, width=20, bg="light blue", text="Edit Accounts", font="calibri", command=edit)
    Edit.grid (sticky=E,column=2,row=3)
```

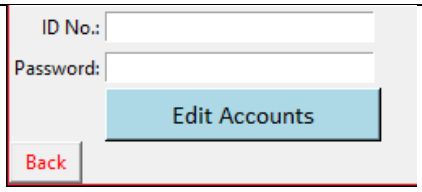
Code of Second Iteration Of Account Menu :User



Second Iteration Of Account Menu :Privileges Edit

```
def createWidgets(self):

    def getPriv():
        ID=idInput.get()
        PR=privInput.get()
        print (PR)
        conn = sqlite3.connect('signin.db')
        output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %PR)
        for row in output:
            Priv=row[1]
            if PR==Priv:
                print ("Users Privellages found")

                conn.close()
                return
            else:
                conn.close()
                return
        conn.close()

    def edit():
        getPriv()


    idInput=StringVar()

    privInput=StringVar()

    Back = Button(self, width=5, fg="red", text="Back")
    Back["command"] =  self.quit
    Back.grid (sticky=W,row=10, column=1)

    ID= Entry(self, width=27,textvariable=idInput)
    ID.grid (sticky=W,column=2,row=1)

    idLabel= Label(self, text="ID No.:")
    idLabel.grid (sticky=E,column=1, row=1)

    privileges= Entry(self, width=27, textvariable=privInput)
    privileges.grid (sticky=W,column=2,row=2)

    privilegesLabel= Label(self, text="Privileges:")
    privilegesLabel.grid (sticky=E,column=1, row=2)


    Edit= Button(self, width=20, bg="light blue", text="Edit Accounts", font="calibri", command=edit)
    Edit.grid (sticky=E,column=2,row=3)
```

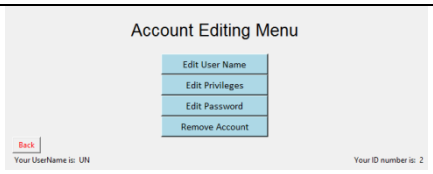
Code of Second Iteration Of Account Menu :Privileges Edit



Second Iteration Of Account Menu :Password Edit

```
    def createWidgets(self):

        def getPass():
            ID=idInput.get()
            PW=pwordInput.get()
            print (PW)
            conn = sqlite3.connect('signin.db')
            output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %PW)
            for row in output:
                Pass=row[1]
                if PW==Priv:
                    print ("Users Password found")

                    conn.close()
                    return
                else:
                    conn.close()
                    return
        def edit():
            getPass()



        idInput=StringVar()

        pwordInput=StringVar()

        Back = Button(self, width=5, fg="red", text="Back")
        Back["command"] =  self.quit
        Back.grid (sticky=W,row=10, column=1)

        ID= Entry(self, width=27,textvariable=idInput)
        ID.grid (sticky=W,column=2,row=1)

        idLabel= Label(self, text="ID No.:")
        idLabel.grid (sticky=E,column=1, row=1)

        pword= Entry(self, width=27,textvariable=pwordInput)
        pword.grid (sticky=W,column=2,row=2)

        pwordLabel= Label(self, text="Password:")
        pwordLabel.grid (sticky=E,column=1, row=2)

        Edit= Button(self, width=20, bg="light blue", text="Edit Accounts", font="calibri", command=edit)
        Edit.grid (sticky=E,column=2,row=3)
```

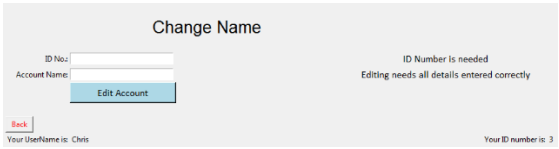
Code of Second Iteration Of Account Menu :Password Edit



Final Iteration Of Account Menu :Menu

```
def makeAccountMenu():
        List.grid_forget()
        Add.grid_forget()
        Account.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        Menu.grid_forget()
        ExitAdd.grid_forget()

        def showAccountMenu():
            toMenu.grid_forget()
            Title.grid_forget()
            accountMenu.grid_forget()
            privilegesMenu.grid_forget()
            pwordMenu.grid_forget()
            conn = sqlite3.connect('signin.db')
            cursor= conn.cursor()
            cursor.execute("SELECT PRIVELLAGES from SignIn WHERE Name='%s'" %disUN)
            row = cursor.fetchone()
            for cursor in row:
                Priv=row[0]
            if Priv=="Admin":
                RemoveMenu.grid_forget()
            else:
                return

        def toMainMenu():
            toMenu.grid_forget()
            Title.grid_forget()
            accountMenu.grid_forget()
            privilegesMenu.grid_forget()
            pwordMenu.grid_forget()
            mainMenu()

        def getRemove():
            conn = sqlite3.connect('signin.db')
            cursor= conn.cursor()
            cursor.execute("SELECT PRIVELLAGES from SignIn WHERE Name='%s'" %disUN)
            row = cursor.fetchone()
            for cursor in row:
                Priv=row[0]
            if Priv=="Admin":
                ButtonRemove()
            else:
                return
        def ButtonRemove():
            global RemoveMenu
            RemoveMenu= Button(self, font="calibri", width=20, bg="light blue", text="Remove Account",command=RemoveAccount)
            RemoveMenu.grid (column=2,row=6)

        def RemoveRemoveButton():
            RemoveMenu.grid_forget()

        getRemove()

        toMenu= Button(self, width=5, fg="red", text="Back",command=toMainMenu)
        toMenu.grid (sticky=W,row=10, column=1)

        Title= Label(self, pady=20, font="calibri, 20", width=27, text="Account Editing Menu")
        Title.grid (column=2,row=1)

        accountMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit User Name", command=NameEdit)
        accountMenu.grid (column=2,row=2)

        privilegesMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit Privileges",command=PrivEdit)
        privilegesMenu.grid (column=2,row=4)

        pwordMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit Password",command=PassEdit)
        pwordMenu.grid (column=2,row=5)
```

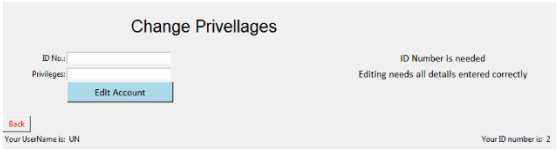
Code Of Final Iteration Of Account Menu :Menu



Final Iteration Of Account Menu : User Name Edit

```
def NameEdit():

    def RemoveNameEdit():
        ToAMenu.grid_forget()
        ID.grid_forget()
        idLabel.grid_forget()
        account.grid_forget()
        accountLabel.grid_forget()
        Edit.grid_forget()
        Inst1.grid_forget()
        Inst2.grid_forget()
        NameTitle.grid_forget()
        showAccountMenu()
        makeAccountMenu()

    def getName():
        ID=idInput.get()
        NM=nameInput.get()
        print (NM)
         conn = sqlite3.connect('signin.db')
         output = conn.execute("UPDATE Privileges from SignIn WHERE ID='%s'" %ID)

    showAccountMenu()

    idInput=StringVar()

    nameInput=StringVar()

    ToAMenu = Button(self, width=5, fg="red", text="Back", command=RemoveNameEdit)
    ToAMenu.grid (sticky=W,row=10, column=1)

    NameTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change Name")
    NameTitle.grid (sticky=W,column=2,row=1)

    ID= Entry(self, width=27,textvariable=idInput)
    ID.grid (sticky=W,column=2,row=2)

    idLabel= Label(self, text="ID No.:")
    idLabel.grid (sticky=E,column=1, row=2)

    account= Entry(self, width=27, textvariable=nameInput)
    account.grid (sticky=W,column=2,row=3)

    accountLabel= Label(self, text="Account Name:")
    accountLabel.grid (sticky=E,column=1, row=3)

    Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri", command=edit)
    Edit.grid (sticky=W,column=2,row=4)

    Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")
    Inst1.grid (sticky=W,row=2, column=3)

    Inst2= Label(self, width=40, text="Editing needs all details entered correctly", font="calibri")
    Inst2.grid (sticky=W,row=3, column=3)
```

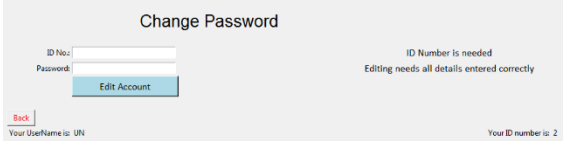
Code OF Final Iteration Of Account Menu : User Name Edit



Final Iteration Of Account Menu :Privileges Edit

```
def PrivEdit():

    def RemovePrivEdit():
        ToAMenu.grid_forget()
        ID.grid_forget()
        idLabel.grid_forget()
        privileges.grid_forget()
        privilegesLabel.grid_forget()
        Edit.grid_forget()
        Inst1.grid_forget()
        Inst2.grid_forget()
        PrivTitle.grid_forget()
        showAccountMenu()
        makeAccountMenu()


    def getPriv():
            ID=idInput.get()
            print (PR)
            conn = sqlite3.connect('signin.db')
            output = conn.execute("UPDATE Privileges from SignIn WHERE ID='%s'" %ID)

    def edit():
        getPriv()

    showAccountMenu()

    idInput=StringVar()

    privInput=StringVar()

    ToAMenu = Button(self, width=5, fg="red", text="Back", command=RemovePrivEdit)
    ToAMenu.grid (sticky=W,row=10, column=1)

    PrivTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change Privellages")
    PrivTitle.grid (sticky=W,column=2,row=1)

    ID= Entry(self, width=27,textvariable=idInput)
    ID.grid (sticky=W,column=2,row=2)

    idLabel= Label(self, text="ID No.:")
    idLabel.grid (sticky=E,column=1, row=2)

    privileges= Entry(self, width=27, textvariable=privInput)
    privileges.grid (sticky=W,column=2,row=3)

    privilegesLabel= Label(self, text="Privileges:")
    privilegesLabel.grid (sticky=E,column=1, row=3)


    Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri", command=edit)
    Edit.grid (sticky=W,column=2,row=4)

    Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")
    Inst1.grid (sticky=W,row=2, column=3)

    Inst2= Label(self, width=40, text="Editing needs all details entered correctly", font="calibri")
    Inst2.grid (sticky=W,row=3, column=3)
```

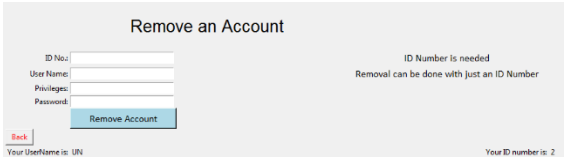
Code Of Final Iteration Of Account Menu :Privileges Edit



Final Iteration Of Account Menu :Password Edit

```
def PassEdit():
    def RemovePassEdit():
        ToAMenu.grid_forget()
        ID.grid_forget()
        idLabel.grid_forget()
        pword.grid_forget()
        pwordLabel.grid_forget()
        Edit.grid_forget()
        Inst1.grid_forget()
        Inst2.grid_forget()
        PassTitle.grid_forget()
        showAccountMenu()
        makeAccountMenu()

    def getPass():
        ID=idInput.get()
        PW=pwordInput.get()
        print (PW)
         conn = sqlite3.connect('signin.db')
         output = conn.execute(UPDATE Password from SignIn WHERE ID='%s'" %ID)

       showAccountMenu()

       idInput=StringVar()

       pwordInput=StringVar()

       ToAMenu = Button(self, width=5, fg="red", text="Back", command=RemovePassEdit)
       ToAMenu.grid (sticky=W,row=10, column=1)

       PassTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change Password")
       PassTitle.grid (sticky=W,column=2,row=1)

       ID= Entry(self, width=27,textvariable=idInput)
       ID.grid (sticky=W,column=2,row=2)

       idLabel= Label(self, text="ID No.:")
       idLabel.grid (sticky=E,column=1, row=2)

       pword= Entry(self, width=27,textvariable=pwordInput)
       pword.grid (sticky=W,column=2,row=3)

       pwordLabel= Label(self, text="Password:")
       pwordLabel.grid (sticky=E,column=1, row=3)

       Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri", command=edit)
       Edit.grid (sticky=W,column=2,row=4)

       Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")
       Inst1.grid (sticky=W,row=2, column=3)

       Inst2= Label(self, width=40, text="Editing needs all details entered correctly", font="calibri")
       Inst2.grid (sticky=W,row=3, column=3)
```

Code Of Final Iteration Of Account Menu :Password Edit



Final Iteration :Remove Account

```
def RemoveAccount():
                    def RemoveRemove():
                        ToAMenu.grid_forget()
                        ID.grid_forget()
                        idLabel.grid_forget()
                        accountMenu.grid_forget()
                        accountLabel.grid_forget()
                        privileges.grid_forget()
                        privilegesLabel.grid_forget()
                        pword.grid_forget()
                        pwordLabel.grid_forget()
                        Remove.grid_forget()
                        Inst1.grid_forget()
                        Inst2.grid_forget()
                        RemTitle.grid_forget()
                        showAccountMenu()
                        makeAccountMenu()

                    def remove():
                        pop = Toplevel()
                        pop.title("REMOVED")

                        msg = Label(pop, text="THIS ACCOUNT HAS BEEN REMOVED")
                        msg.pack()

                        ok = Button(pop, text="Dismiss", command=pop.destroy)
                        ok.pack()

                        ID=idInput.get()

                        conn = sqlite3.connect('signin.db')
                        conn.execute("DELETE from SignIn WHERE ID='%s'" %ID)
                        conn.close()
                        print("REMOVED USER")

                    def warning():
                        pop = Toplevel()
                        pop.title("WARNING")

                        msg = Label(pop, text="THIS WILL REMOVE ALL ACCOUNT DETAILS, ARE YOU SURE?")
                        msg.pack()

                        yes = Button(pop, text="I Am Sure",command=remove, bg="light blue")
                        yes.pack()

                        no = Button(pop, text="Dismiss", command=pop.destroy)
                        no.pack()

                    showAccountMenu()

                    idInput=StringVar()

                    pwordInput=StringVar()

                    nameInput=StringVar()

                    privInput=StringVar()

                    ToAMenu = Button(self, width=5, fg="red", text="Back",command=RemoveRemove)
                    ToAMenu.grid (sticky=W,row=10, column=1)

                    RemTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Remove an Account")
                    RemTitle.grid (sticky=W,column=2,row=1)

                    ID= Entry(self, width=27,textvariable=idInput)
                    ID.grid (sticky=W,column=2,row=2)

                    idLabel= Label(self, text="ID No.:")
                    idLabel.grid (sticky=E,column=1, row=2)

                    accountMenu= Entry(self, width=27, textvariable=nameInput)
                    accountMenu.grid (sticky=W,column=2,row=3)

                    accountLabel= Label(self, text="User Name:")
                    accountLabel.grid (sticky=E,column=1, row=3)
                    privileges= Entry(self, width=27, textvariable=privInput)
                    privileges.grid (sticky=W,column=2,row=4)

                    privilegesLabel= Label(self, text="Privileges:")
                    privilegesLabel.grid (sticky=E,column=1, row=4)

                    pword= Entry(self, width=27,textvariable=pwordInput)
                    pword.grid (sticky=W,column=2,row=5)

                    pwordLabel= Label(self, text="Password:")
                    pwordLabel.grid (sticky=E,column=1, row=5)

                    Remove= Button(self, width=20, bg="light blue", text="Remove Account", font="calibri", command=warning)
                    Remove.grid (sticky=W,column=2,row=6)

                    Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")
                    Inst1.grid (row=2, column=3)

                    Inst2= Label(self, width=40, text="Removal can be done with just an ID Number", font="calibri")
                    Inst2.grid (row=3, column=3)
```

Code Of Final Iteration :Remove Account

Evaluation of the Iteration of Account Settings

The main concept for the “Account Edit” screen allowed the Name, Privileges and Password of the account to be changed. The account could also be removed by the “Remove” button. This concept was very simple for the screen that came to be one of the most complicated and changed screen out of all of them.

In the first iteration very little changed other than the inclusion of an ID number for the user. This inclusion was made during the designing of the User database where there needed to be a primary key. Therefore the User’s Id number became this primary key as it allowed for easy organisation and removal of users. The only other changes from concept to first iteration were graphical based. These were changes such as the “Remove” button got changed to “Remove Accounts” in order for its purpose to become more easily understood. The entry fields for the information had their widths increased to the same as the buttons located below them. This is a change which was carried forward onto all of the other pages as well. This is due to having them different sizes meant that the page didn’t seem very well organised.

During the transit from first iteration to second iteration it became clear that having everything on a single page would not function efficiently. This is as certain functions could only be available to those that were authorised. These are things such as changing privileges and removing their account. Therefore by creating a menu screens could be assigned to each of the buttons and if a user did not have the correct privileges then that button would not be created for them. However this code was not implemented till the final iteration of the program. A prominent function that is not included within the second iteration is the ability to remove the User form the database. This is as the remove page was not created until the final iteration as in creating the remove page many of the improvements between the final and second iterations came to be. Each of the individual pages after the menu have the same layout but with an alternate different function. The standard layout is entry for ID number, entry for information to be changed followed by a button to save the change made. This simple layout was carried out across all of the secondary screens. However an error soon appeared, this being that users of the program would be expected to remember their ID number through out there use of the program. Also at the time the only way to find this out was to access the database itself and view the information located within it. This lead to the changes that can be seen within the final iteration of the “Account Edit” screen.

In the final iteration of the “Account Edit” screen the largest change was the introduction of the ability to see the current users ID number. This can be seen in the bottom right hand corner of all final iteration screens. The code for this to be generated however is located in the sign in screen under the function “getName” this retrieves the User Name from the sign in database. Once this is done it then checks that username against the associated ID number within in the database. This is then sent to the Main Menu’s ID labels. These ID labels stay prevalent throughout the rest of the program and only change if the information is edited if the information in the database changes.

In terms of more graphical changes the titles have also been added to again help with the navigation of the program. Also for this effect instructions on how to use each of the screens have been added to each of the screens. This is to reduce the confusion when a user tries to use a screen for the first time.

The code behind the screen has had to change drastically due to the change in functions and design since concept. The original pseudo code for the concept of this screen was very simplified when it is compared to the final results. Although a large portion of the final code is labels and buttons these were not included in the original pseudo code. From the pseudo code to the first iteration the code

only became more fleshed out. This is as the functions for the individual buttons became more descriptive and then their pseudo counter parts. Also the general structure of the code was altered as the widgets are now called after the functions. This is a change that had to occur as if put the other way round then the functions would be undefined when they're called. Other than this it was simply retrieving the Usernames, Passwords and Privileges that has changed.

The biggest difference in code is the jump from first iteration to second iteration as this is the moment when the single screen gets split into four different screens. Other than the menu all the screens share the same layout. The only real difference is what the main function that is defined does. This functions operation depends on which page the function is located in.

Testing

Testing Criteria

1.Easily search through all items in stock	2.Easily edit any items details	3.All information is secure and can only be accessed by authorised personal	4.Automatic alert if stock reaches zero
5.Account details can be changed	6.New stock can be inputted by appropriate users	7.The screens can be navigated easily	9.The automatic restock button orders more stock from the retailer

Appropriate Testing methods

Criteria	*Type of method*	*Method of testing*
Easily search through all items in stock	Quantitative, Qualitative	Will gather five people from the workplace and ask them to try and find certain objects. During this process screen shots and recordings will be taken to see if the process is simple to do. A questionnaire can also be taken to ask for the member's experience of navigating the program
Easily edit any items details	Quantitative, Qualitative	Take five different people from the intend place of use and let them edit specific items details. Whilst this occurs pictures and evidence will be taken to prove it is legitimate. A survey can be taken by the people from the experiment to ask how easy it was to edit the items details.
All information is secure and can only be accessed by authorised personal	Quantitative	Take three people with sign in details and three without and let them all try and sign in. The same steps shall be repeated but this time with access to the Database. Screenshots will be taken at each stage
Automatic alert if stock reaches zero	Quantitative	Five random items in the stock list will be set to zero items in stock. The appropriate response should be sent to the intended members of staff.
Account details can be changed	Quantitative	Three users with administration privileges will be asked to sign in to the program and edit

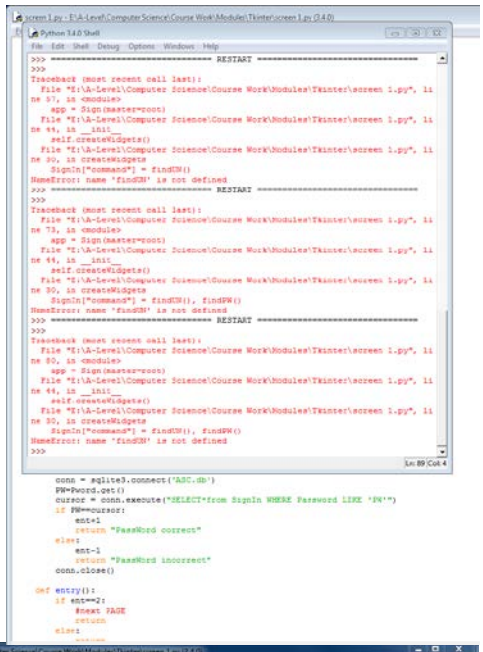
		three random users' data.
New stock can be inputted by appropriate users	Quantitative	Five users will be asked to add three random items of stock to the database. The Database will then be closed and reopened to check if the items persist. This can be recorded via screenshots and captures.
The screens can be navigated to easily	Quantitative, Qualitative	Take three members of staff and ask them to navigate to specific parts of the program. A timer shall be run to see if this is indeed easy to do along with screenshots of the process and the Pc's clock. A survey can also be taken to ask for the member's experience of the program.
The restock button orders more stock from the retailer	Quantitative, Qualitative	Three authorised members of staff will be asked to restock random items. A test email account can then be checked to see if the test emails have been received and if not the problem can be looked into.

White Box and Black Box Testing Examples

Definitions

White Box: This type of testing focuses on the structure and syntax of the written code. This means that code shall be run to make sure the program runs not worrying if the functions work themselves.

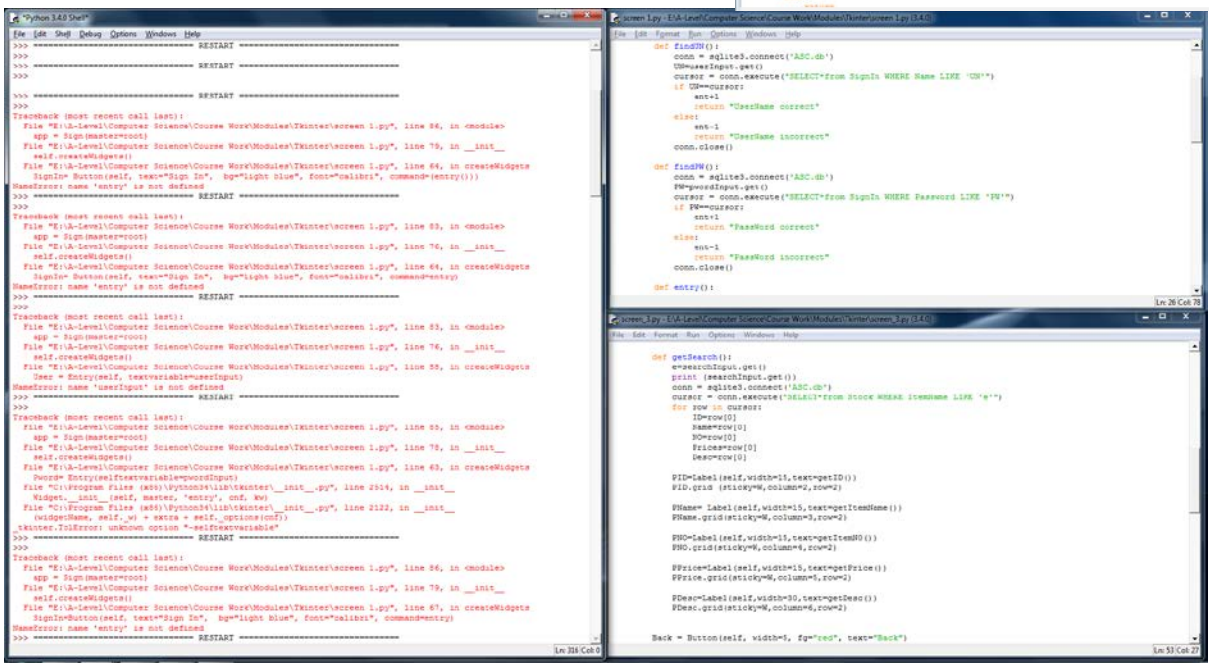
Black Box: This type of testing is all about receiving the correct outputs from the system when the correct inputs are entered into the system.



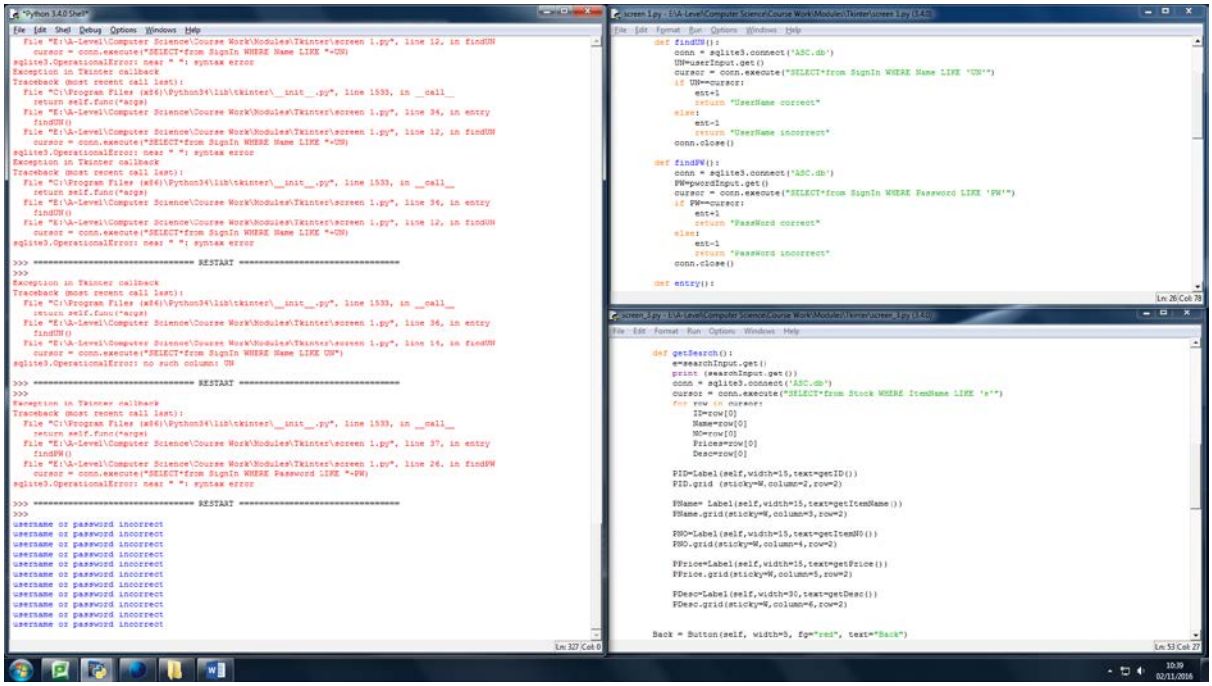
Sign In Testing

In this first example we can see the White Box testing of the first iteration of the sign in function. In this first image it can be seen that the function “findUN()” has not been successfully defined. Therefore the function can not be executed and the sign in process can’t be carried out. If this is the case then the whole system falls apart as without the ability to sign in the rest of the functions can’t be accessed. In order to solve this issue the function “findUN()” was moved above the button that hjad this command attached to it. This is due to the code reading the command to call the function before the fucntion was given it’s definition.

In the second image White Box testing of the same



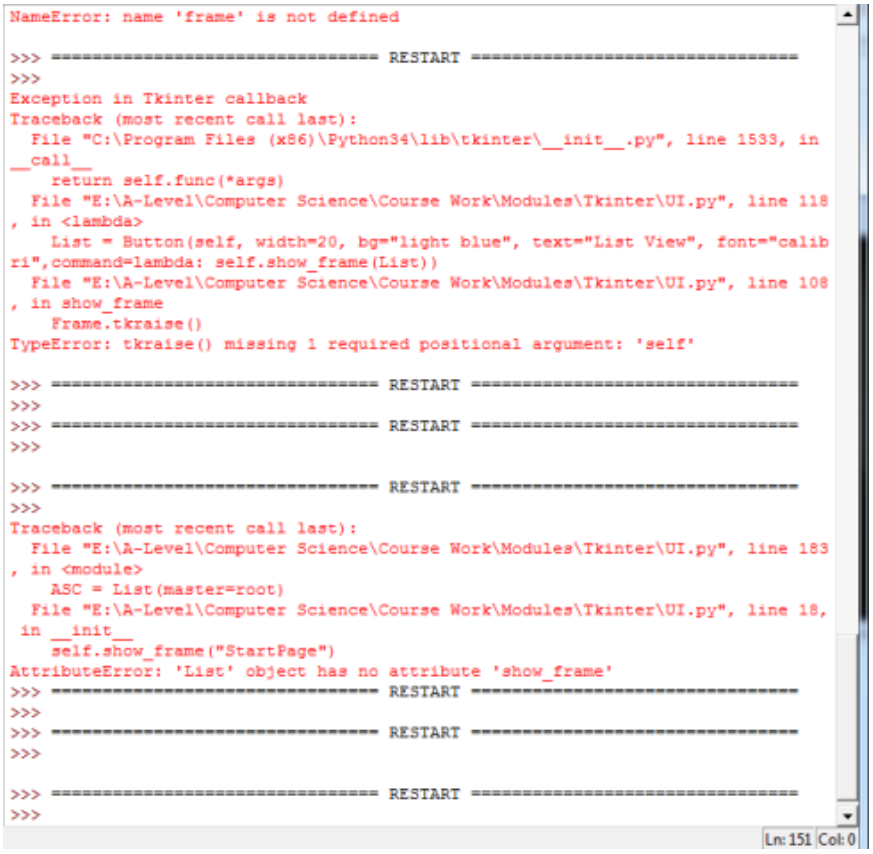
function unvieled that several other funtions were not being defined properly. However on the right hand side of the image these functions can be clearly nseenwithin the given “def” commands. For example one of the errors that is repeated the most is that ““entry” is not defined”. However when looking in the screen 1.py file “def entry()” can clearly be seen. Again this error was the matter of structure where the define commands were given to much indentation and so the order was read wrong.



Here is the first example of a Black Box test where the inputs and outputs of the system are tested to their full extent. In the example given above the correct information is inputted into the system but access is not given as the system returns that these are incorrect submissions. This information is given due to print functions being entered into the code to help debug the system. This therefore shows that there is an error in the system that is non-syntax based. In this case the error was due to what information was being taken from the database to be checked against. This can be seen in both the “findUM()” and “findPW” functions where it says cursor=. This is as these commands were simply retriveing items that were like UN or PW. Due to this it was wsimply retriebvong “None” which when cdomapered to the inforamton entered was not the same. Therefore the function had to be changed to “output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %UN)” where %UN is a variable given before hand.

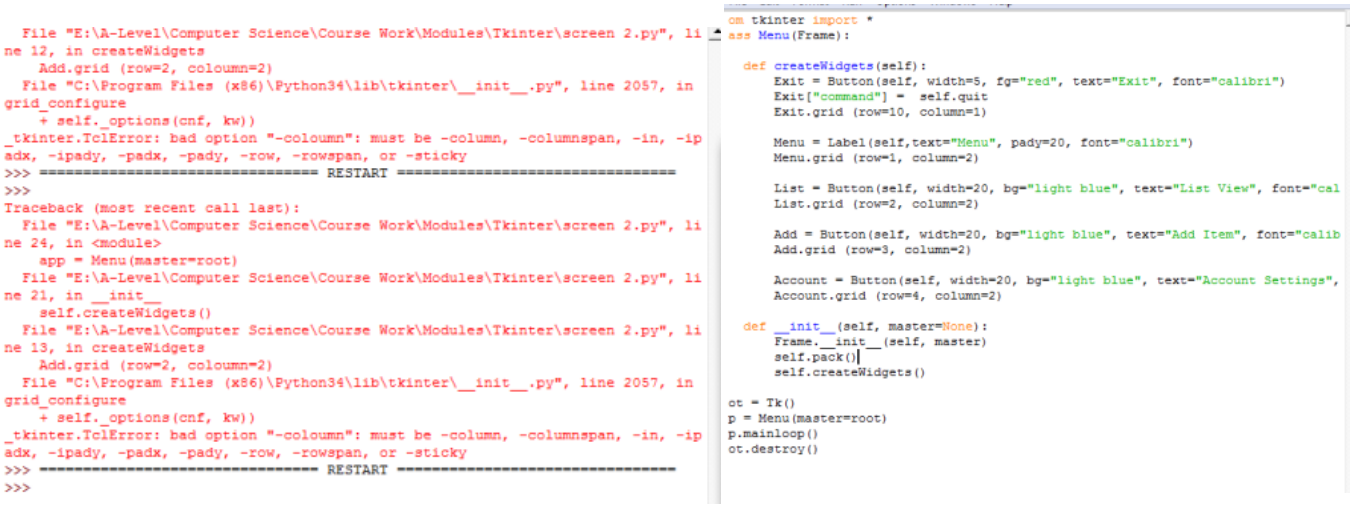
Screen Switch Testing

This is another example of a white box test that has been carried out where I am trying to get the two screens to switch between each other. In the given screen shot this is being done via the use of frames. However in theory the main problem that arose from this method is that every time a new frame was opened it opened in a new window which reduced the performance and usability of the program. The particular error shown below though shows how this particular iteration of the program has trouble raising the frames or showing them. This is the main reason why this method was abandoned as it refused to work with the set of screens as each one needed a designated parent. Therefore this error was never fixed and instead lead to the incorporation of the forgetting and replacing of the widgets. Although this new method lead to longer functions having to be developed it did allow for easier selection of what widgets were to be shown.



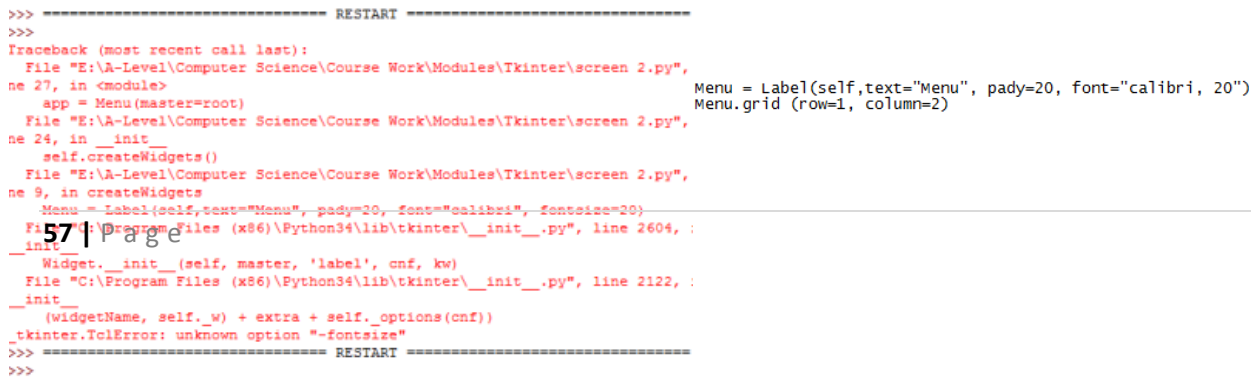
Menu Generation

This is yet another white box test, due to this uncovering a more code related issue, where I am testing to see whether the widgets for the main menu will load correctly. However as can be seen from the debugger the attributes for certain widgets were incorrect. This is as problem that persisted throughout the generation of many other screens although in almost all cases it was a form of syntax error. The example given below can be seen to have had this happen in the Add button widgets attributes where “column” was mistyped with an extra o after the “l”. This lead to a failure in the generation of the screen as it was given an unknown attribute. However this was easily fixed and the menu continued to generate as can be seen at the end of the debugger and also in the



PY file itself.

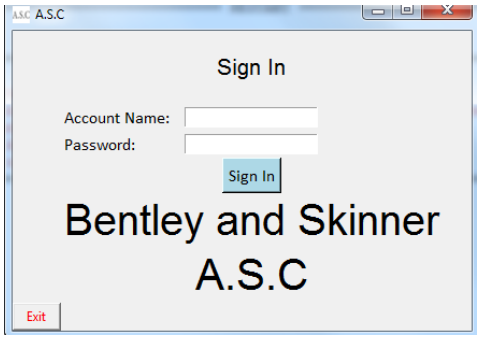
Shown below is a similar example of the error given above occurring in the attributes of another widget of the Main Menu screen in a separate black box test. This time however it takes place within a label widget which acts as a title for the Main Menu screen. Though this time it wasn’t a syntax error that caused the menu generation not to occur it was a wrong command all together. This is as I enter “fontsize” into the attributes list to try to apply the attribute separately. However the solution to this error was simply adding the font size to the font attribute instead of separately. This can be seen in the second screen shot from a partial iteration of the Main Menu.



Logo Generation

This is a White box test with the intended result being the generation of the companies logo in the middle of the Sign In screen. The error that occurred on this test is that the logo file couldn't be found by the program and therefore it could't be displayed. This is another synatx error of sorts as all that had to be done to rectify this mistake was to re type the address to the file and to incorporate the nessecary file destinations aswell. This is as the file wasn't stored in the same file as the code and therefore the directory to find it was different. Therefore the address to this file had to be redefined within the code. After this ocurred the logo successfully genrated itself onto the sign in screen however no evidence can be given for this. This is as the logo on the screen was soon replaced with the title labels which display "Bentley and Skinner" and "ASC" as can be seen in the second screenshot.

```
>>> ================================ RESTART ================================
>>>
Traceback (most recent call last):
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen 1.py", li
ne 49, in <module>
    app = Sign(master=root)
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen 1.py", li
ne 40, in __init__
    self.createWidgets()
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen 1.py", li
ne 32, in createWidgets
    logo=ImageTk.PhotoImage(Image.open("logo.gif"))
  File "C:\Program Files (x86)\Python34\lib\site-packages\PIL\Image.py", line 22
74, in open
    % (filename if filename else fp))
OSError: cannot identify image file 'logo.gif'
>>>
```



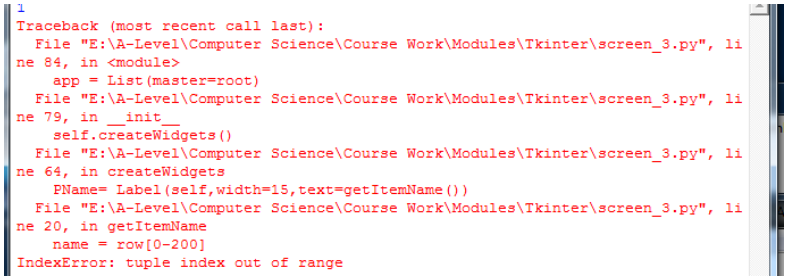
Search Function

The search function was a difficult one to implement into the system even though the idea of it was very basic in its final step by step design. Below are the screen shots showing the Black Box testing of the search function. The reason this test is a black box test is as it is testing whether the required outputs are being given when the information is given. Unfortunately this is not the case in this given example as an error has appeared in the debugger. This error states that the function get ID cannot be defined and therefore cannot be run by the program. Due to this the intended output, being the search criteria, has not been given. The answer to this problem was simple being that the get ID function was moved above the button in the code. This is so as the program read itself it could find the getID function before it reached the button where it asked to call it. The mistake here was just that the code had been order correctly.

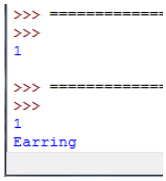
```
>>> ================================ RESTART ================================
>>>
Traceback (most recent call last):
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen_3.py", li
ne 84, in <module>
    app = List(master=root)
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen_3.py", li
ne 79, in __init__
    self.createWidgets()
  File "E:\A-Level\Computer Science\Course Work\Modules\Tkinter\screen_3.py", li
ne 61, in createWidgets
    PID=Label(self,width=15,text=getID())
NameError: name 'getID' is not defined
```

```
def getID():
    SI=searchInput.get()
    conn = sqlite3.connect('ASC.db')
    cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%S
    for row in cursor:
        ID=row[0]
```

Here is another few problems that arose in the black box testing of the list view screen also in its search function. Here the error is nothing to do with the order of the code itself but the inability to retrieve the information from the database as the error “tuple index out of range” occurs. This error means that the amount of information I have tried to access cannot be retrieved as there is not enough to cover the intended range. The reason for this occurring is that I said that there were 201 rows as can be seen in the debugger where it says “name = row [0-200]”. This is as I was asking for it to retrieve 201 rows when there were only 6 in the database. The simple fix to this was to set it to row [0] where it would select only 1 row. Once this was carried out I was able to retrieve an entity from the database multiple times.



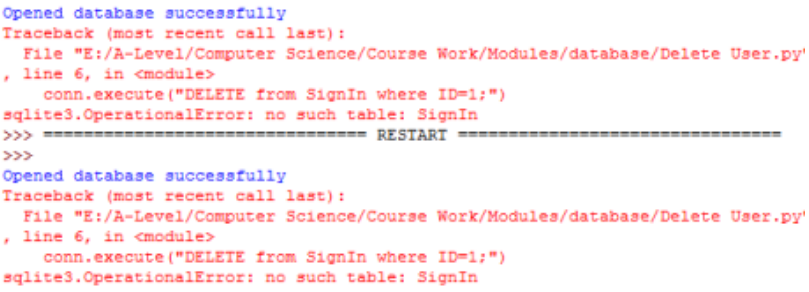

```
    for row in cursor:
        name = row[0]
def getItemN0():
```



Database

Below is the White Box testing of me inputting information to the database with SQL commands in python, the first screenshot shows the code I used to complete this task. The main error that occurred from this inputting of data via SQL commands is that the sign in table couldn’t be found by Python. If it couldn’t find the table it couldn’t insert the information into that table and so therefore the command fails. The fix to this error was not via this set of code. Instead I had to delve on to the code for the database. The problem here was that the Sign In table had a syntax error in its name and therefore the correct table couldn’t be loaded by python when it was asked to.

```
import sqlite3

conn = sqlite3.connect('ASC.db')
print ("Opened ASC database successfully");

conn.execute("INSERT INTO SignIn (ID,Name,Password,Privellages) \
      VALUES (1, 'Paul', 'Password', 'Admin' )");

conn.commit()
print ("Records created successfully");
conn.close()
```



Log of Final Testing and Evaluation of Functions

In order to prove that continuous testing has led to the program becoming more robust and efficient program I have created a log of final testing. This will a show a log of every function being tested to its full extent. The test will be run according to the previously given Testing Criteria though the main criteria points are given below. Each function is equipped with an evaluation of its contents.

Main Criteria Points

1.Easily search through all items in stock	2.Easily edit any items details	3.All information is secure and can only be accessed by authorised personal	4.Automatic alert if stock reaches zero
5.Account details can be changed	6.New stock can be inputted by appropriate users	7.The screens can be navigated easily	8.The restock button orders more stock from the retailer

Throughout the ASC's code "print" functions have been entered in the executed functions. This means that a message will be displayed in the debugger so it is proved that it has been completed. An example of one of these functions has been provided:

```
    searchTitle.grid_forget()
    Clear.grid_forget()
    print("Welcome to the Main Menu")
    showMenu()

f notify():
```

```
Type "copyright", "credits"
>>> ======================
>>>

>>> ======================
>>>
Welcome to the Main Menu
```

Some of the screenshots have been cropped in order for the information to be viewed more easily.

The more simplified functions such as moving to another screen only have a single outcome. However the more advanced functions such as retrieving functions will have multiple outcomes and multiple tests. This is to ensure that all situations that could occur have been taken into account.

Sign In Screen

This a small screen with two functions but is the first screen that users will see and determines multiple future factors.

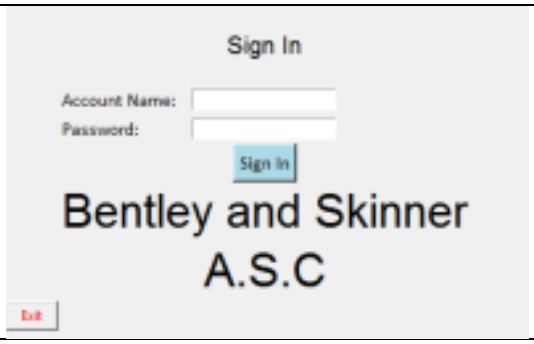
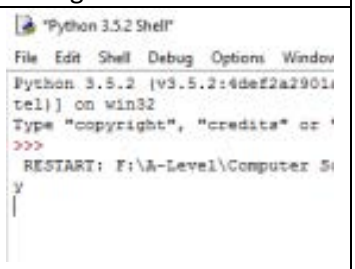
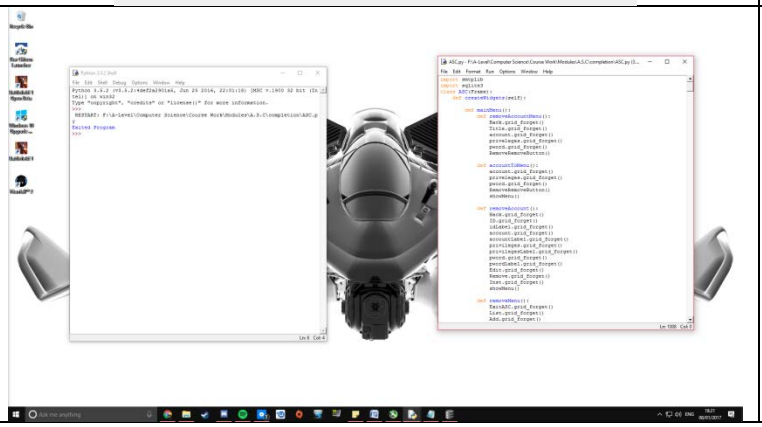
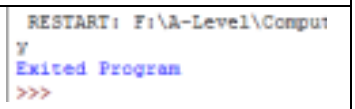
Function	Function Description	Possible Outcomes	Criteria Point
Exit	When run will cause the program to cease all functions and shut down	1)Program Window should close	7. Screens can be navigated easily
Sign In	Once activated should check the appropriate data base to see if that by the given details if the user should be able to gain access to the rest of the program	1)User gains access opening the menu 2)The user is refused access due to wrong password 3)The user is refused access due to wrong username	3.All information is secure and can only be accessed by authorised personal 7. Screens can be navigated easily

Exit Function Testing

Testing the exit button is very simple as there is only one correct outcome. This is that the program stops all functions and closes, a message will then be displayed in the debugger saying the program had been closed. This will help prove that criteria point 7 has been carried out, this being “7.The screens can be navigated easily”.

Outcome 1:

The sign in screen is loaded and it’s “Exit” function is tested by pressing the “Exit” widget. This shall prompt a message in the debugger to be called.

Stage	Screenshot Of Process	Debug Of Process
1 Open ASC		
2 Exited		

Evaluation of Exit Function

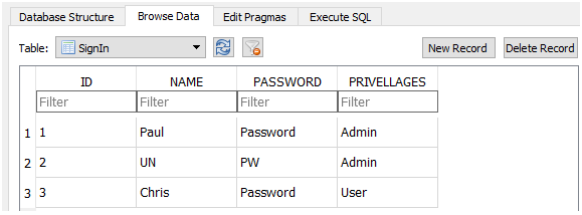
The exit function of the sign in page performs as expected. This can clearly be seen in the screens shots given. This is as when the exit widget is pressed the window closes and everything is returned to the desktop as can be seen in the second screen shot. The debugger shows this function has been executed as “Print (“Exited Program”)” was added at the end of the function to be done when the program has executed. When the data form this test is checked next to the fundamentals that it is meant to achieve being criteria 7 it is clear why it is a successful function.

Sign In Testing

To test the first outcome three methods will be executed, method one sees the correct information entered into the fields in order for Outcome 1 to be achieved. The second method sees the Password entered incorrectly to create Outcome 2. The third method will see the username entered incorrectly so the user is denied access due to a wrong username, Outcome 3. Each of these shall be tested at least twice with two different UN or passwords.

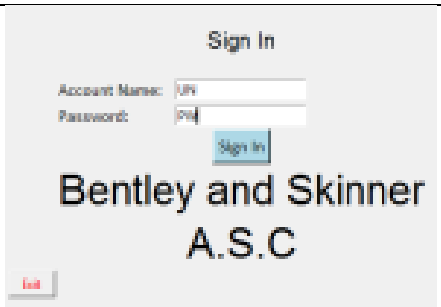
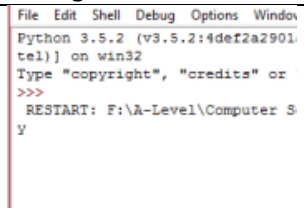
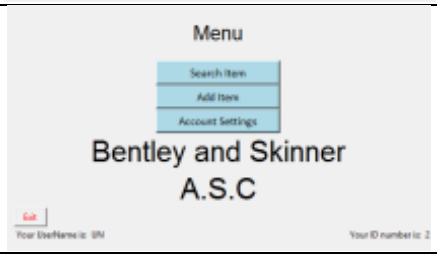
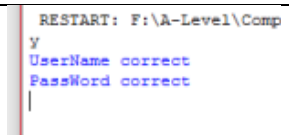
Testing the sign in feature allows me to prove that the criteria point 3, "All information is secure and can only be accessed by authorised personal", has been assessed. This is as without the appropriate sign in information it is impossible for the data to be accessed and retrieved.

The accepted login information from the database:



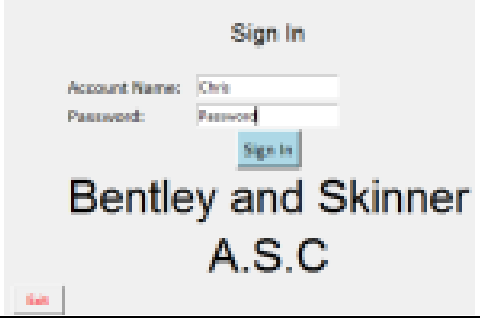
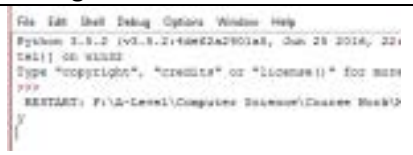
Outcome 1: Test 1:

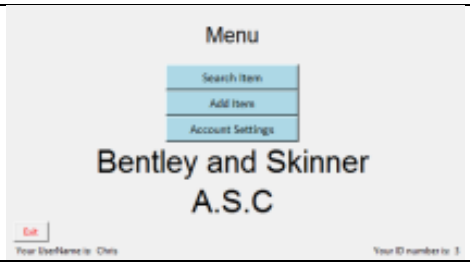
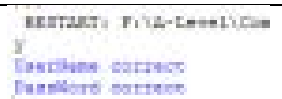
For this test I am using the log in details of the username being "UN" and the password being "PW"

Stage	Screenshot Of Process	Debug Of Process
1 Open ASC + Enter Info.		
2 Click "Sign In" Button		

Outcome 1: Test 2:

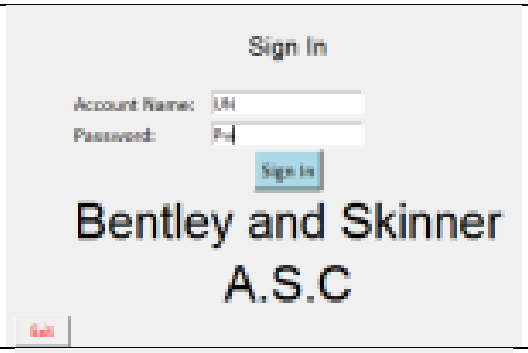
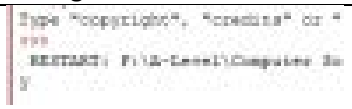
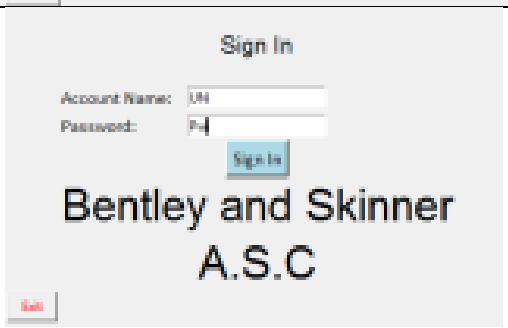
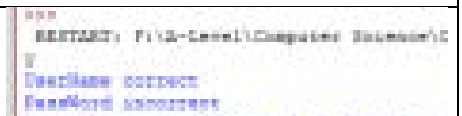
For this test I am using the log in details of the username being "Chris" and the password being "Password". This is to show that multiple usernames work and not just UN.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Opened + Info Entered		

2 Sign In Button Clicked		

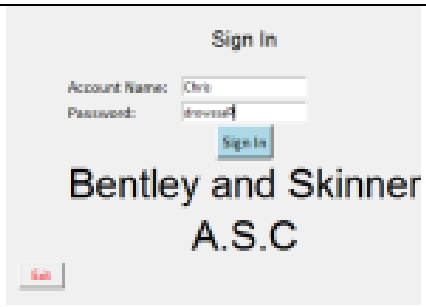
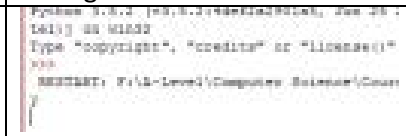
Outcome 2: Test 1:

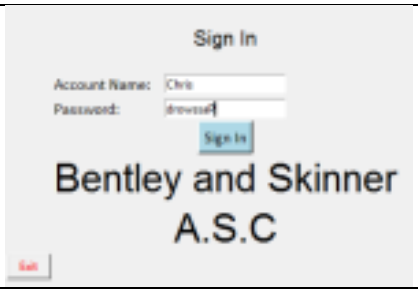
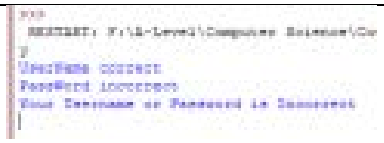
For this test I am using the log in details of the username being “UN” and the password being “PW”. This will show that due to the “W” being in a different case then the user will be denied entry.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Program Run + Details Entered		
2 Sign In Button Deployed		

Outcome 2: Test 2:

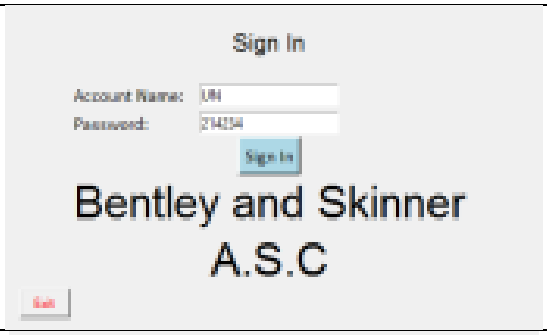
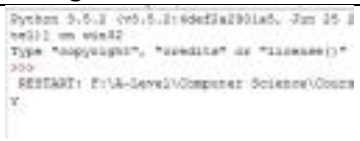
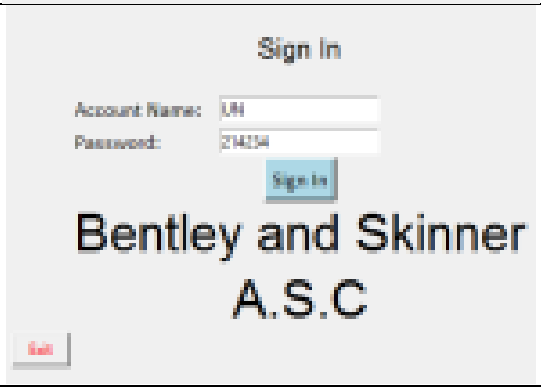
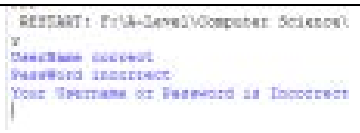
For this test I am using the log in details of the username being “Chris” and the password being “drowssaP”. This is to show that it denies other users and even if all characters are there but in the incorrect order.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Run + Credentials Entered		

2 Sign In Widget Clicked		

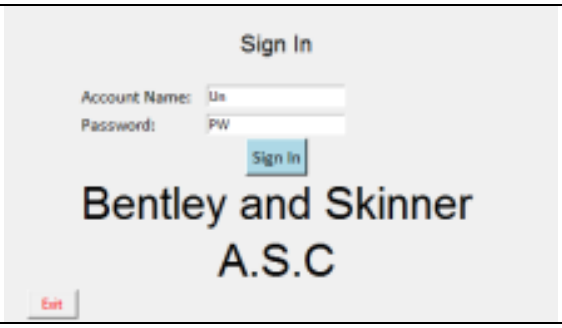
Outcome 2: Test 3:

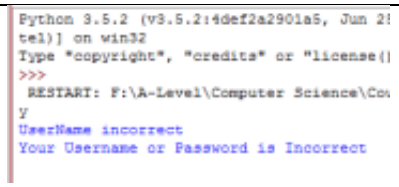
For this test I am using the log in details of the username being “UN” and the password being “214254”. This will ensure that the functions also register and check numbers as well as text.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Executed + Entry Details Entered		
2 Sign In Is Pressed		

Outcome 3: Test 1:

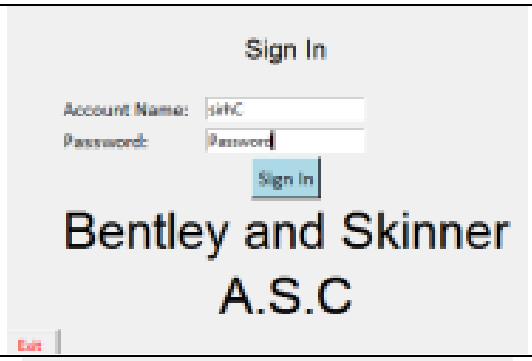
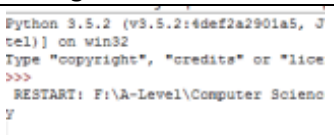
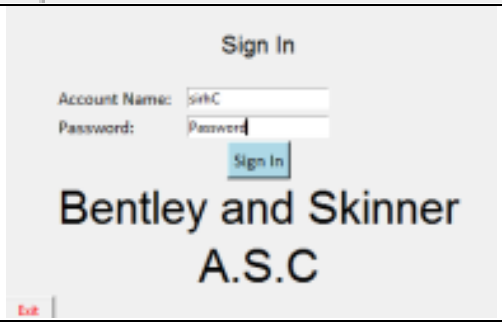
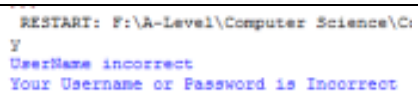
For this test I am using the log in details of the username being “Un” and the password being “PW”. This will show that due to the “N” being in a different case then the user will be denied entry.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Opened + Details In Fields		

2 Sign In Run		

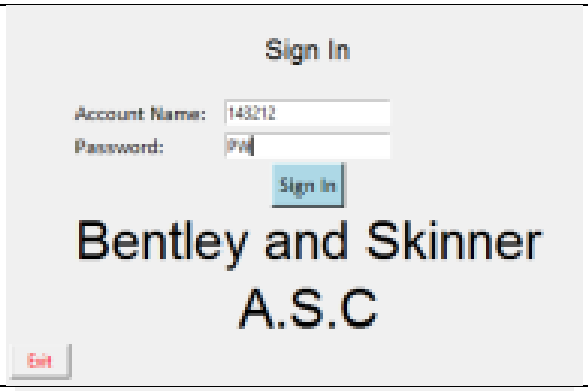
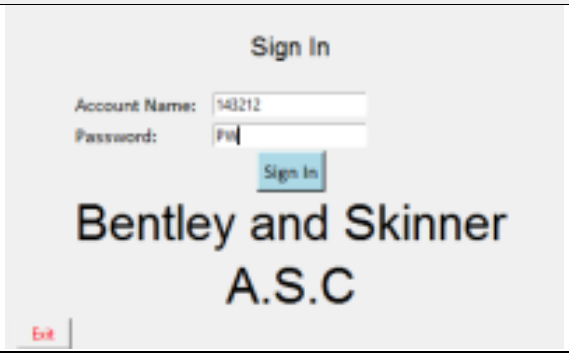
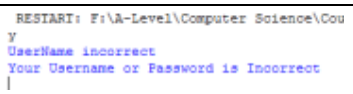
Outcome 3: Test 2:

For this test I am using the log in details of the username being “sirhC” and the password being “Password”. This is to show that it denies other users and even if all characters are there but in the incorrect order.

Stage	Screenshot Of Process	Debug Of Process
1 Sign In Screen Opened + Details Entered		
2 Sign In Function Executed		

Outcome 3: Test 3:

For this test I am using the log in details of the username being “143212” and the password being “PW”. This will ensure that the function also registers and check numbers as well as text.

Stage	Screenshot Of Process	Debug Of Process
1 Sign In Screen Opened + Test Details Put In Entries		
2 Widget labelled Sign In Activated		

Evaluation of Sign In test

From the test of the “Sign In” function it is clear that this function is a success. This is as throughout the test only when entry details were 100% correct was access given. The Outcome 1 tests showed that if the login details were correct the user would be taken to the Main Menu which fulfils criteria point 7 of ease of navigation. This is as this is where the user would be expected to be taken when they signed in to the program. Also it gives credit towards point 3 as the users sign in details can only be on the system when verified and inserted by the appropriate officials.

From the Outcome 2 and Outcome 3 tests more credit can be given towards criteria 3. This is as they show incorrect information given when gaining access and due to this they’re not given entry to the program as they’re seen as non-authorised. This can also be seen to be true when the correct information is given in the incorrect sequence. This shows that the system takes the entered information and checks it completely and not just if all information is present.

From this it is clear to see why I think this function exceeds at keeping the data away from unauthorised individuals as well as improve the ease of navigation for the user.

Main Menu Screen

Although this is not the first screen that is seen or used it is the one that users will spend most of their time on. It consists of four functions all of which have been tested to the same degree.

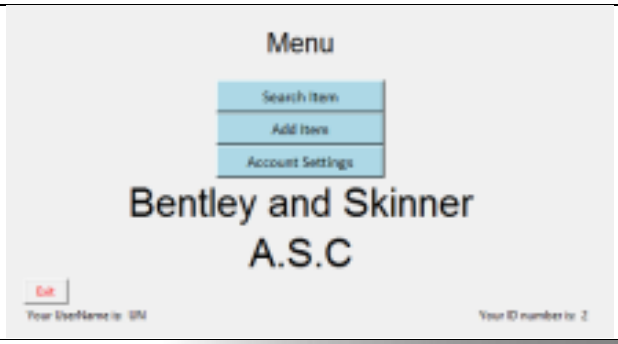
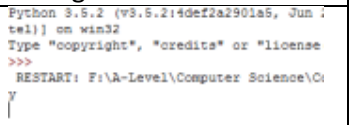
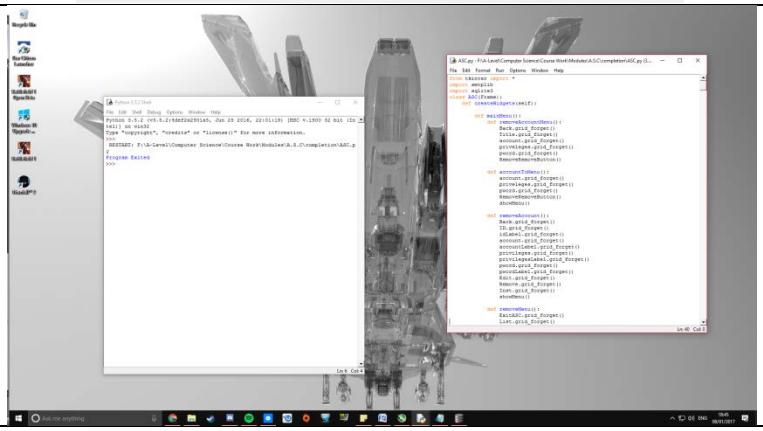
Function	Function Description	Possible Outcomes	Criteria Point
Exit	When run will cause the program to cease all functions and shut down	1)Program Window should close	7.The screens can be navigated easily
Go to Search	When button is clicked all widgets are removed and widgets for the Search screen are loaded	1)Program goes to the Search screen	7.The screens can be navigated easily
Go to Add Item	When button is clicked all widgets are removed and widgets for the Add Item screen are loaded	1) Program goes to the Add Item screen	7.The screens can be navigated easily
Go to Account Menu	When button is clicked all widgets are removed and widgets for the Account Menu screen are loaded	1) Program goes to the Account Menu	7.The screens can be navigated easily
Generate UN and ID information	When user signs in depending on privileges given to them depends on what buttons are generated	1)Information is retrieved and given to the user	3.All information is secure and can only be accessed by authorised personal

Exit Function Testing

The same as the previous test of the exit button this is very simple and allows me to test for criteria point 7. This is that “The screens can be navigated easily”.

Outcome 1:

All that has to be done is the exit button to be clicked allowing the window to close. Once this is done a message will be displayed in the debug window saying that it has been closed.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Opened on Main Menu		
2 Exit Widget Clicked		

Evaluation of Exit Function

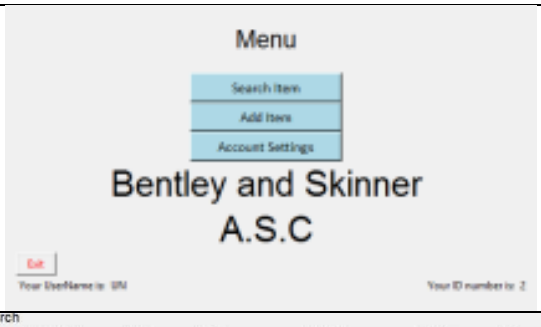
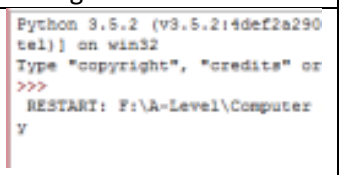
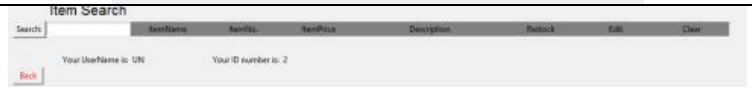
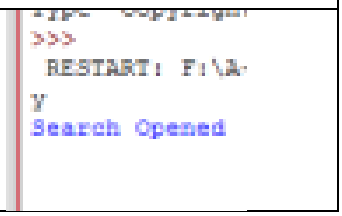
As with the “Exit” function seen within the sign in screen this one also excels at its primary purpose. This is of course being criteria 7, the ease of navigation. This is as soon as the assigned widget is pressed the exit function is run returning the user to their desktop and displaying the text ”Program Exited” in the debugger. Due to the ease of this transition is why I believe it can be said that criteria 7 has been completed in this instant.

Go to Search Testing

This is to test the functionality of the system when attributing to criteria point 7 being that “The screens can be navigated easily”.

Outcome 1:

Due to the simplicity of this function there can only be one intended outcome. This being that the Search screen opens and a message is displayed into the debugger notifying that it has opened the search screen.

Stage	Screenshot Of Process	Debug Of Process
1 Main Menu Run		
2 Search Widget Clicked		

Evaluation of Go To Search

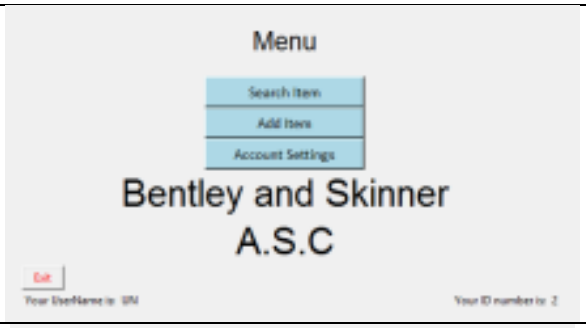
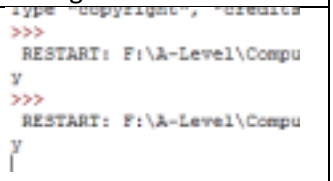
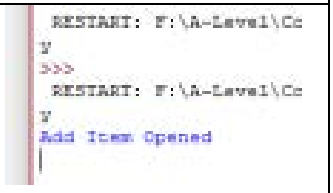
The results of the “Go To Search” testing shows it being very effective at completing its aim of criteria 7. This is as when run through its given widget the user is taken exactly where they wish to go. Evidence of this is given in the debugger where the message “Search Opened” is displayed. This message is given at the end of the function by a “print” function that could only be run when the rest of the function was successful. This is why this function is to be seen as complete and adhering to criteria 7.

Go to Add Item Testing

This function allows the testing of criteria 7, “The screens can be navigated easily” as the ability to switch screens is included within this point.

Outcome 1:

The intended output of this function is for the Main Menu screen to change to the Add Item screen by removing all the Main Menu’s widgets and then creating all of the Add Item’s widgets. This shall produce a message in the debugger when done.

Stage	Screenshot Of Process	Debug Of Process
1 Main Menu Opened		
2 Add Item Button Pressed		

Evaluation of Go To Add Item

This function indeed works to its full extent as when executed by the assigned button it takes the user to the intended destination of the Add Item screen. The given criteria for this function was number 7 which is the ease of access. Due to its successfulness of taking the user to the intended destination I can say that it has been completed. This is furthermore proved by the text printed into the debugger after the function was run.

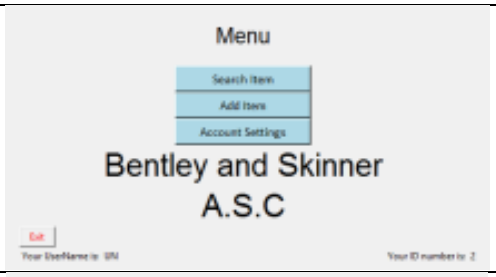
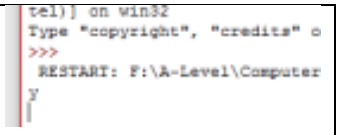
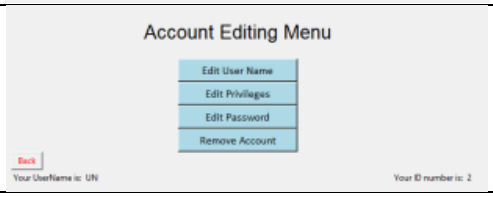
Go to Account Menu Testing

This function is essential for criteria point 7, “The screens can be navigated easily”, as without this function you would be unable to access the “Account Menu”.

Outcome 1:

For this function to operate effectively it must clear the Menu screen and then generate the Account Menu screen. A message will be displayed in the debugger when done so.

Stage	Screenshot Of Process	Debug Of Process

1 ASC's Main Menu Opened			
2 Account Settings Widget Is Activated			

Evaluation of Go To Account Menu Test

As with previous criteria 7 functions I believe this one has been successful. This is due to it successfully meeting criteria 7 of easing navigation. This is as it is very simplistic and takes the user where they wish to go and therefore doesn't confuse them. This execution can be seen in Outcome 1 where after the function has been fully executed a message is printed in the debugger window. This gives evidence towards the function being successful at its intended purpose.

Search Screen

This is the screen that is the most important of all of the rest as it gives the ASC its name

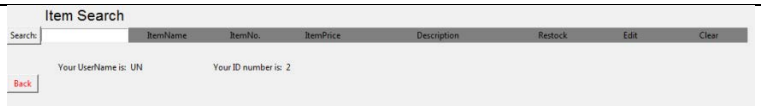
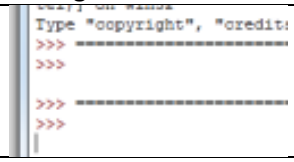
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Main Menu screen	Should remove all of the search screens widgets and create the widgets for the main menu	1)Program goes to Main menu	7.The screens can be navigated easily
Search	When search button is clicked items are generated according to the given criteria	1)Program loads an item which matches the given criteria 2)Program loads Item not Found as one does not much the given criteria	1.Easily search through all items in stock
Go to Edit Item Screen	When button is clicked all Search screen widgets are removed and widgets for the Edit Item screen are created	1) Program goes to the Edit Item screen	7. The screens can be navigated easily
Send Email Notification	When restock button is clicked an email is sent to the associated email address notifying them of a need to restock	1) Program sends email to an assigned Gmail account	4. Automatic alert if stock reaches zero 8.The restock button orders more stock from the retailer
Clear Search Selection	When Clear Search button Is clicked the search entries are removed allowing a for second to be performed	1)Search entry is removed	N/A

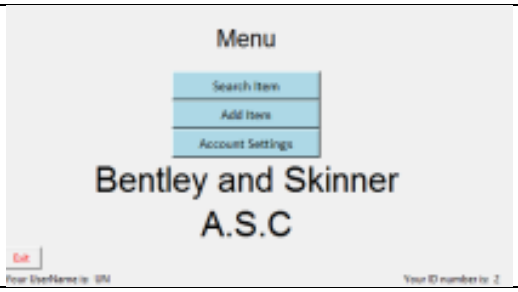
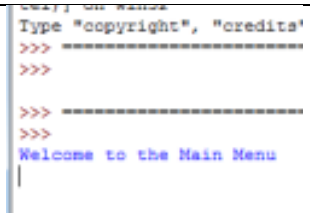
To Main Menu Testing

As with other previous screen switching functions this is a necessary function for the program to successfully meet the given criteria point 7, “The screens can be navigated easily”.

Outcome 1:

For this test the back button will be pressed which will force the program to move back to the previous screen of the program. The message “Welcome to the main menu” will then be displayed in the debugger. This is in order to meet the criteria point 7 of, “The screens can be navigated easily”.

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Run		

2 Back Button Deployed			

Evaluation of To Main Menu

The return to Main Menu, registered as the "Back" function, helps ease the navigation for the user as they have a way to return to their previous page. This is the hoped for outcome from the criteria 7 that was assigned to this function. Criteria 7 is the ease of navigation and is something that this function definitely attributes to and for this reason is why I say that I think for the evidence shown by the testing it succeeds at its intended purpose.

Search Function Testing

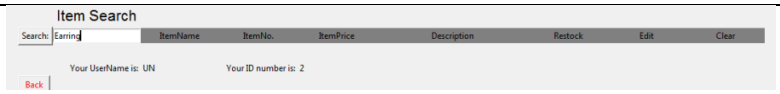
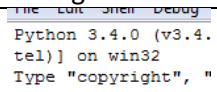
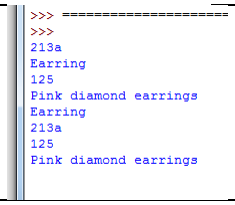
In order to test the search function these are the entries included into the database that it can possibly retrieve:

	ItemName	ItemNo	Price	Description
	Filter	Filter	Filter	Filter
1	Ring	231d	220	Silver Ring Plain
2	Bracelet	e12e	123	Plain gold band
3	Necklace	124e	400	Necklace with...
4	Pendant	we12	500	Emerald gree...
5	Earring	213a	125	Pink diamond ...

This test is one of the most important as it has to achieve criteria point 1, "Easily search through all items in stock". This criteria point is the original solution to the problem given by the client.

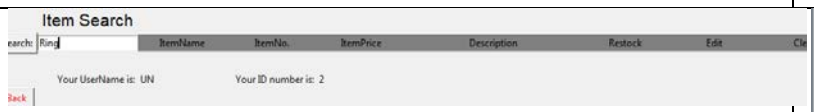
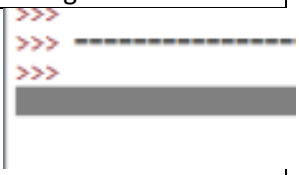
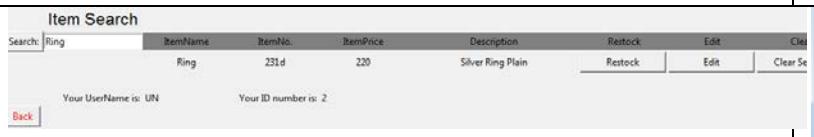
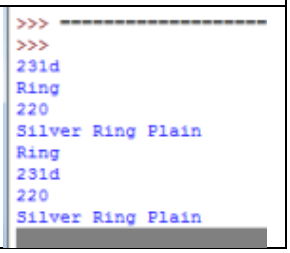
Outcome 1: Test 1:

"Earring" shall be entered into the search bar to retrieve the information associated with this item. This information shall be displayed in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Deployed		
2 Search Result For Earring		

Outcome 1: Test 2:

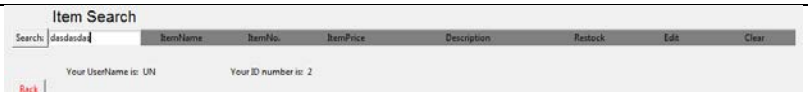
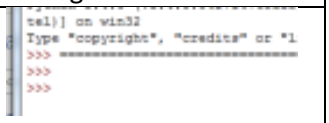
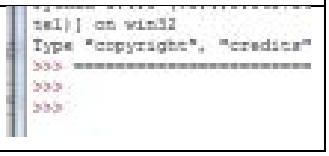
"Ring" shall be entered into the search bar to retrieve information associated with this item and to prove other items can be retrieved from the table

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Activated		
2 Search Result for Ring		

The next lot of tests will test to see if the search failure part of the system operates fully.

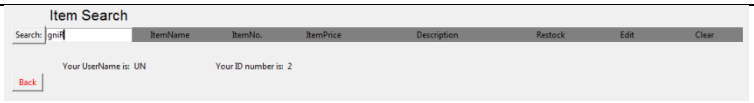
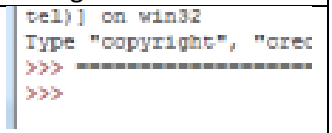
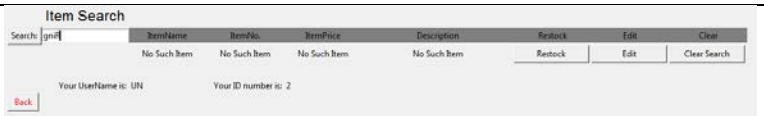
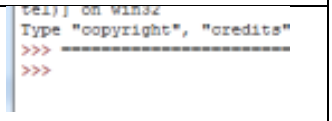
Outcome 2: Test 1:

Random characters will be entered into the search bar in order to obtain the “No Such Item” response.

Stage	Screenshot Of Process	Debug Of Process
1 ASC Search Screen opened + Test Data Entered		
2 “No Such Item” Retrieved		

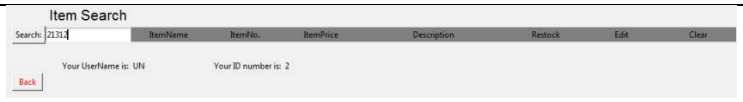
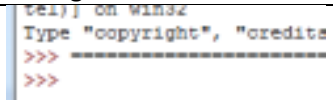
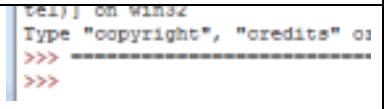
Outcome 2: Test 2:

“gniR” shall be entered to get the “No Such Item” response to prove that even if all the correct characters are given the item is not retrieved

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Executed + gniR entered		
2 Search Entry for “gniR” gained		

Outcome 2: Test 3:

A random string of numbers will entered into the system to show that the system can register number as well as letters. This shall prompt the “No Such Item” response.

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Executed + random number string entered		
2 Search Widget Activated		

Evaluation of Search Function Test

This test of the Search Function is one of the most important of the whole program this is as this function is the one that the rest of the program was designed around. From the multiple tests that were carried out on this function it is clear that it operates at its full effectiveness.

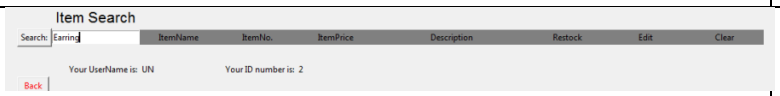
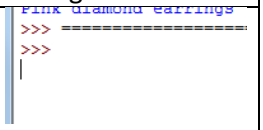
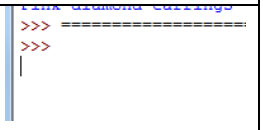
In the Outcome 1 tests all of the test data showed that the program was able to retrieve an items data from the database and display it to the user. This can be seen in the retrieval of both Earring and Ring as all of their information is taken and correctly displayed.

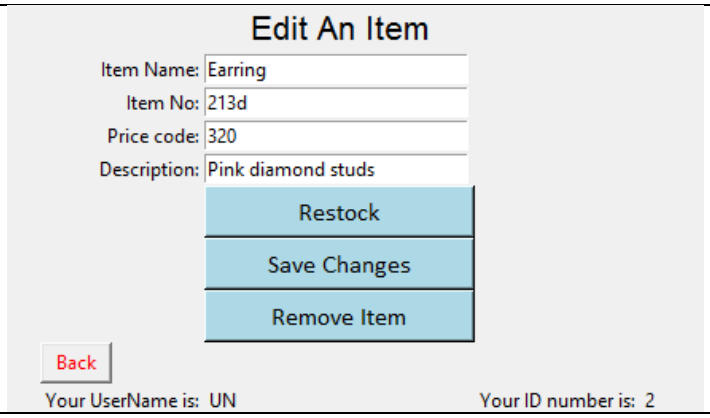
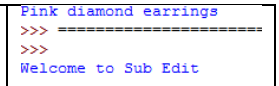
Go to Edit Item Testing

For the same purpose as the other functions whose aim it is to switch screens this must be tested against criteria 7 for “The screens can be navigated easily”.

Outcome 1:

The “Edit” button shall be pressed taking the user to the “Edit Item” screen using the phrase “Earring”. A message will be displayed in the debugger on its completion.

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Loaded + Search Term Entered		
2 Search Result for “Earring” Retrieved		

3 Edit Widget pressed Displaying information		

Evaluation of Go to Edit Item testing

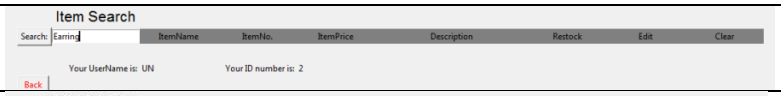
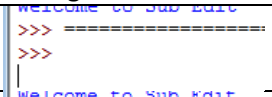
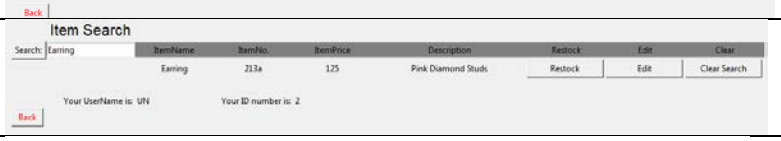
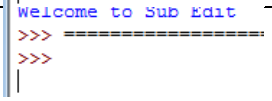
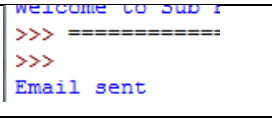
This go to function has been incorporated from the original Main Menu screen in order to keep the results the same. The main idea of this function is for user to be able to edit an item straight from the search screen without having to return to the Main Menu first. This is important as this function is meant to withhold criteria 7 which is to ease the navigation of the program. This is done in this case by reducing the amount of pages and functions that have to be activated to reach the intended destination. This function allows only one activation and screen transition where as previously it would have been double that.

Send Email Notification Testing

This test is in order to see if an email notification can be sent to the email address that is entered into the system. This therefore can be tested against both points 4 and 8. These being “Automatic alert if stock reaches zero” and “The restock button orders more stock from the retailer”.

Outcome 1:

This test involves the clicking of the “Restock” button in order to send the email notification that the selected search item needs to be restocked.

Stage	Screenshot Of Process	Debug Of Process
1		
2		
3		

Evaluation of Email Notification Test

From the evidence gathered it is clear that the Email Notification function does indeed work however the function is meant to fulfil both criteria 8 and 4. Criteria 8 is the ability to send a restock email to the supplier and ask for more stock of an item. This can easily be established by changing the destination email address to the suppliers. This therefore means that this function has met that Criteria. Criteria 4 on the other hand has not been met as fully as criteria 8 has. This is as criteria 4 asks for the program to send an automatic restock notification when an objects stock reaches zero. One problem that arose from this criteria was the process of making the program’s notification automatic. This is as the client’s store sells many product that cannot be restocked as they’re one off items. Therefore there would be no reason for a notification to be sent saying they need to be

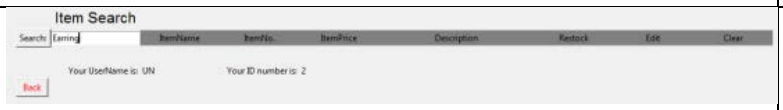
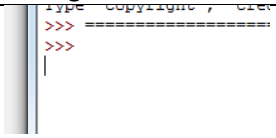
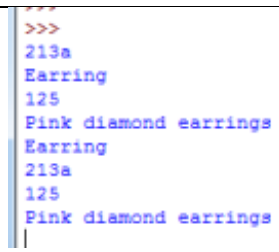
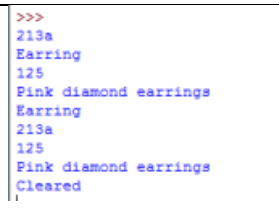
restocked. In any case for the stock to be set to zero the user would need to interact with the system. That is why the stock counter was never implemented and the restock button was put in its place as instead of setting the counter to zero the user can just request more. This is why I believe that this function has just met this criteria as although the process isn't automatic a notification can still be sent when there is none of the product remaining.

Clear Search Selection Testing

Clear Search function testing doesn't have strict criteria to adhere to as it was added to help keep the Search screen working.

Outcome 1: Test 1:

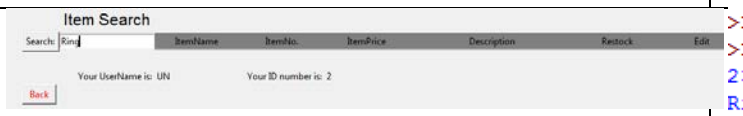
This test involves searching for "Earring" to generate the result before the search is then cleared.

Stage	Screenshot Of Process	Debug Of Process
1 Search Screen Generated + Test Data in Field		
2 Search Screen With Result		
3 Clear Widget Activated		

Outcome 1: Test 2:

This test involves searching for "Ring" to generate the result before the search is then cleared.

Stage	Screenshot Of Process	Debug Of Process

1 Search Screen Generated + Test Data in Field		
2 Search Screen With Result		>>> ========== >>> 231d Ring 220 Silver Ring Plain Ring 231d 220 Silver Ring Plain
3 Clear Widget Activated		>>> ========== >>> 231d Ring 220 Silver Ring Plain Ring 231d 220 Silver Ring Plain

Evaluation of Clear Search Test

The clear search function testing in my opinion was a success as the gathered data proves that the function can work to its full capabilities. However to this end there is no criteria made for this function to be tested against as it was only included to ease the use of the search function. This is as once an item has been searched for the item has to be cleared for another one to be searched. Because of this I believe that the function still is a success as it allows a second more important function to operate fully.

Add Item Screen

This screen as the simple task of adding new items to the data base because of this it only has two functions to be tested. However multiple outcomes will be available when adding an item due to the huge uncertainty of what could be added into the database

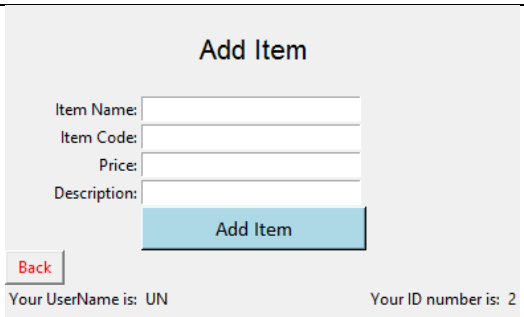
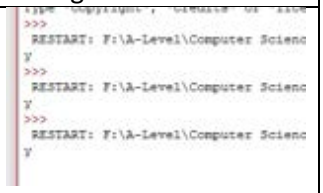
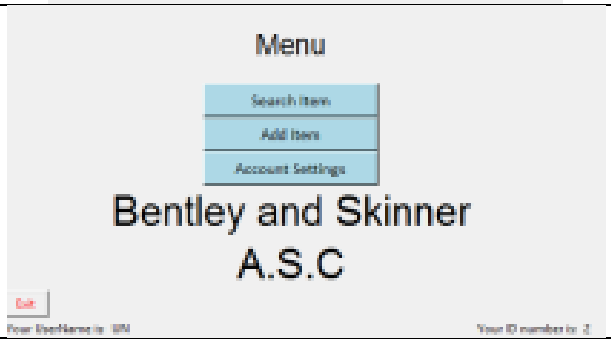
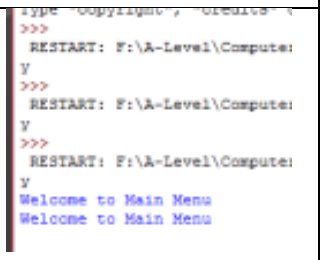
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Main Menu screen	Should remove all of the Add Item screen's widgets and create the widgets for the main menu	1)Program goes to Main menu	7.The screens can be navigated easily
Add Item	Should add all of the items entered details into the database	1)Item gets added to the data base 2) Item doesn't get added due to wrong character types entered. 3)Item isn't added due to empty fields	6.New stock can be inputted by appropriate users

Back to the Main Menu screen

This is another switch screen function that also is to be tested against criteria point 7, "The screens can be navigated easily".

Outcome 1:

When the "Back" button is pressed the screens widgets should be removed and replaced with the Main Menu's

Stage	Screenshot Of Process	Debug Of Process
1 Add Item Screen Created		
2 Back Button Pressed		

Evaluation of Back to Main Menu Test

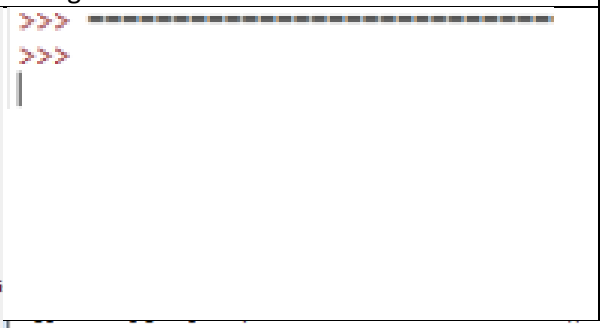
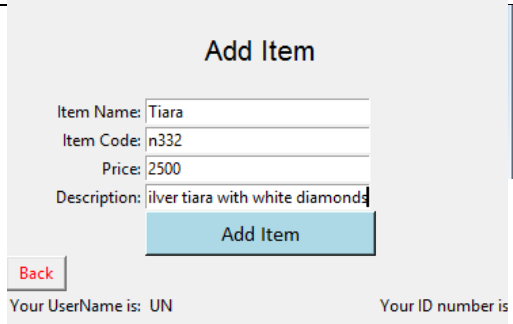
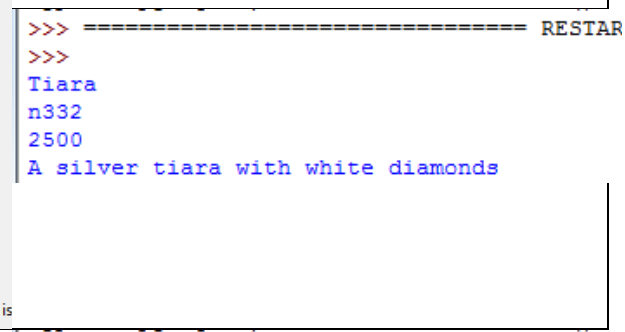
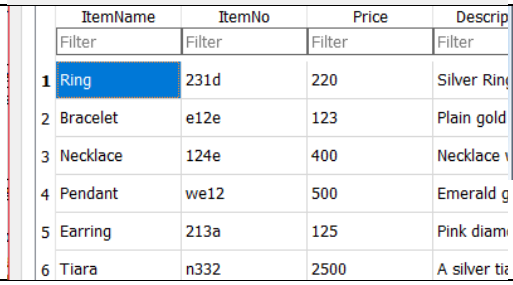
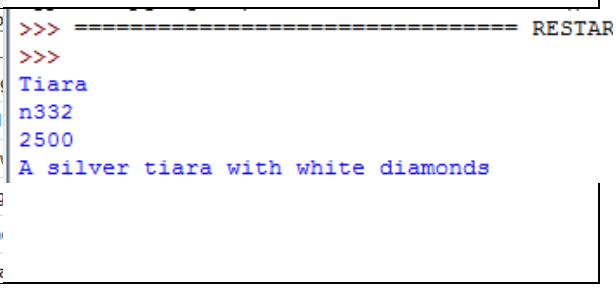
As with other return to main menu functions this one has had success in all of its criteria. The main criteria it was checked against was criteria 7 being to ease the navigation of the system. By adding this back function and its assigned widget it means that any user will not get stuck on the Add item screen. It also gives them the ability to move to other pages without restarting the program. Due to this I believe its inclusion has increased the efficiency of the program making it a great addition to the program.

Add Item

The Add Item function is an important function as it is the only that can be checked against criteria 6, this being "New stock can be inputted by appropriate users".

Outcome 1: Test 1:

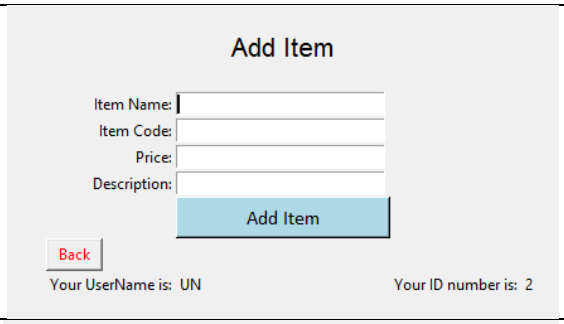
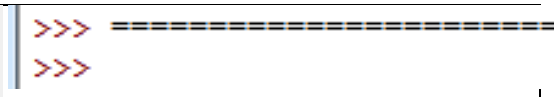
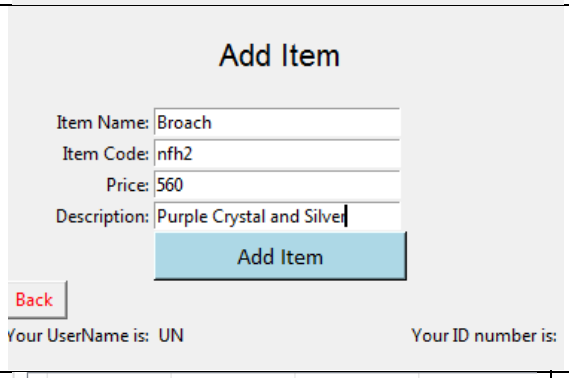
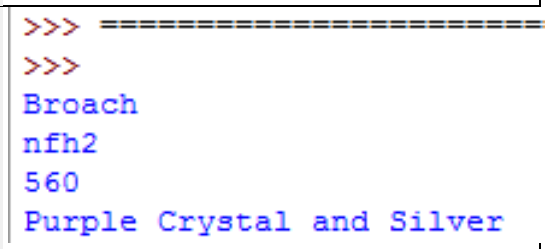
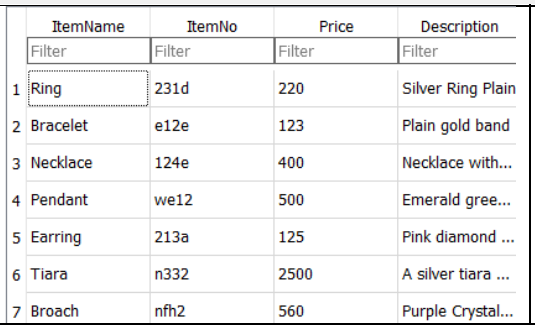
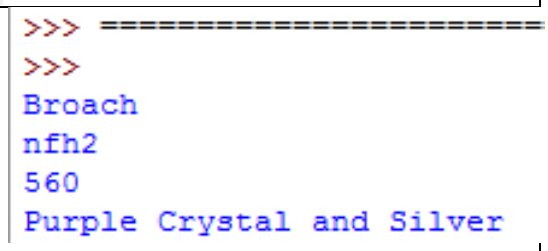
For this test the item "Tiara" will be added. It will have the details "n332", "2500" and "A silver tiara with white diamonds".

Stage	Screenshot Of Process	Debug Of Process
1 Add Item Screen Loaded		
2 Entry Fields Filled		
3 Evidence For Adding In SQL Editor		

Outcome 1: Test 2:

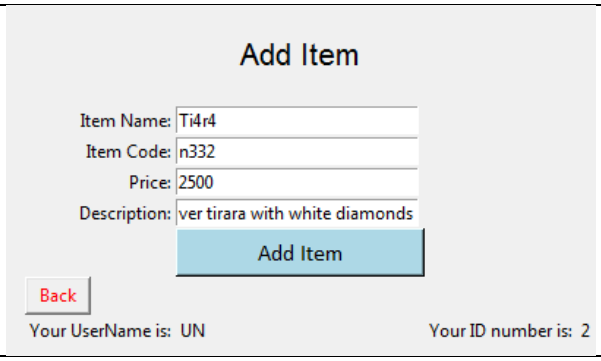
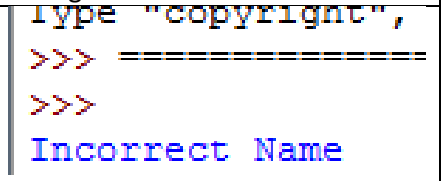
For this I shall add the item "Broach", it will have the following details; "nfh2", "560", "Purple Crystal and Silver"

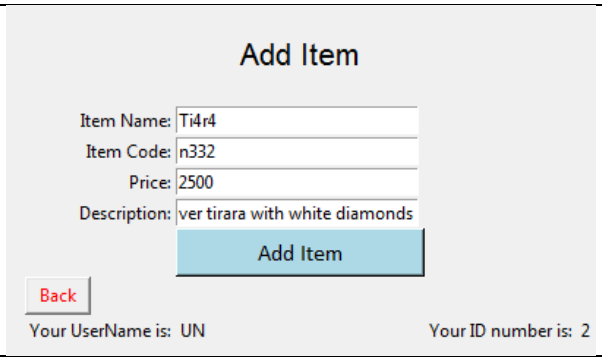
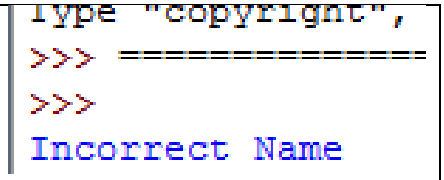
Stage	Screenshot Of Process	Debug Of Process

1 Add Item Opened		
2 Info Entered		
3 Evidence For Add In SQL Editor		

Outcome 2: Test 1:

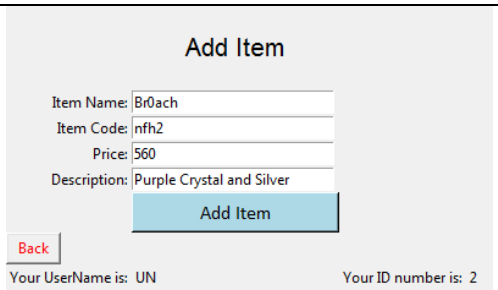
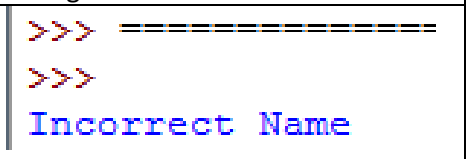
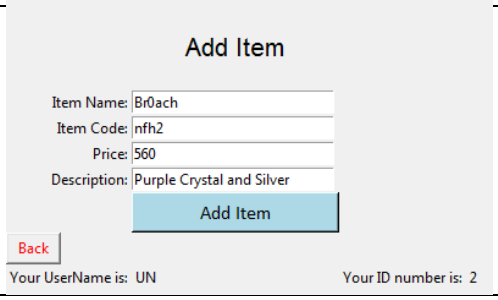
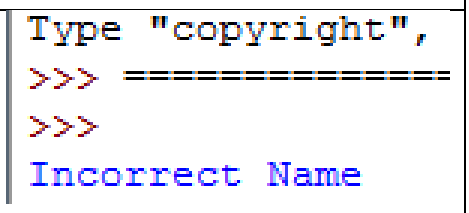
In order to show that trying to add items that contain incorrect characters for those fields results in the failure of the process I have tried adding “Tiara” again but replaced the a’s with 4’s.

Stage	Screenshot Of Process	Debug Of Process
1 Test Data Entered		

2 Evidence of Failure in Debugger		

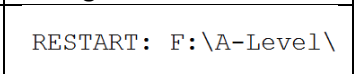
Outcome 2: Test 2:

To show putting incorrect characters in field's results in an addition failure I have tried adding "Broach" with the 0's of the price replaced with o's.

Stage	Screenshot Of Process	Debug Of Process
1 Test Data Inserted into Fields		
2 Add Item Pressed + Evidence for Failure		

Outcome 3: Test 1:

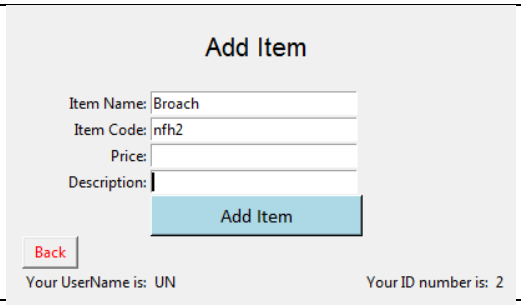
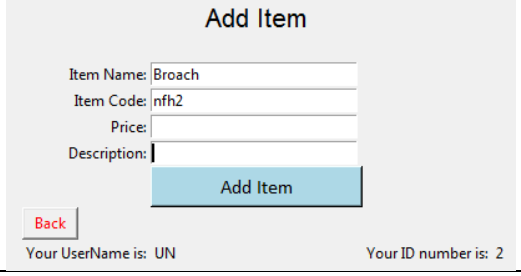
To show failure is also caused from empty fields I have filled in none of the fields to show failure.

Stage	Screenshot Of Process	Debug Of Process
1 Add Item Screen Loaded + Entries Left Blank		

2 Add Item Widget Pressed + Error Shown In Debugger		RESTART: F:\A-Leve Blank Field Blank Field Blank Field Blank Field

Outcome 3: Test 2:

To show failure in all degrees half the fields have been filled in the others have been left blank.

Stage	Screenshot Of Process	Debug Of Process
1 Add Item Loaded + Half of Fields Filled		RESTART: F:\A-Level\
2 Add Item Button Activated + Error Evidence		RESTART: F:\A-Leve Blank Field Blank Field Blank Field Blank Field

Evaluation of Add Item Test

From the data supplied from the tests I can conclusively check this function against its available criteria being Criteria 6. This point is unique only to this program as it states “New stock can be inputted by appropriate users”. This criteria for a function is not used anywhere else within the system which is why it is important that this function operates at 100% to this Criteria.

From the Outcome 1 tests and the evidence given in both the screen shots and the debugger it is clear that the function can be used to add item to the database. This is as in both instances given all information given ends up within the database it has been assigned to with no problems. This is as the information can be seen to have been extracted and added to the database in the debugger.

In the Outcome 2 tests the screen shots show how the information can’t be added if it is not added in a readable format. This is as the database has been set up with the Name field being a text only field which doesn’t include numerical characters. This is why when the numerical characters are added to these fields an error occurs although this has been simplified in screenshots.

In the Outcome 3 tests the failure is caused by empty fields as a simple “if” statement has been added to check if the fields are blank or not. If they are then the function is ended due to the error.

From this the Criteria 6 must have been met as items can indeed be added to the database. As for items being added only by authorised personnel that is the responsibility of the sign in function and the administrators of the system. As both of these have the ability to decide the privileges of the user and if they can enter. This is why criteria 6 has been met by the inclusion of this function.

Edit Item Screen

The edit item screen is a counter part of the search screen as well as a modified version of the Add Item screen.

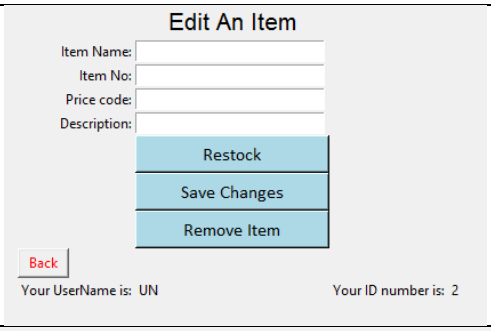
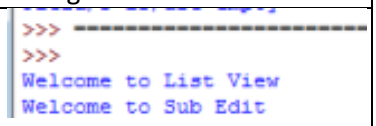
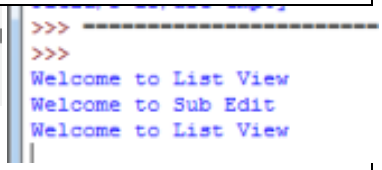
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Search screen	Should remove all of the Edit Item's widgets and create the widgets for the main menu	1)Program goes to Main menu	7.The screens can be navigated easily
Save Changes	Should take the information from the entry fields and insert it into the database	1)Changes to Item's information gets saved to appropriate database 2)Information isn't saved to the database	2.Easily edit any items details
Remove Item	Should remove the item with the same Item No from the database	1)Information gets removed from the database	2.Easily edit any items details

Back to the Search screen

This back navigation is essential for it to be able to complete criteria 7, "The screens can be navigated easily". This is as if a page cannot return the whole system breaks down.

Outcome 1:

This is another simple function that's job is to remove all of the widgets and replace them with the ones for the next screen.

Stage	Screenshot Of Process	Debug Of Process
1 Edit Item Screen Loaded		
2 Back Button Clicked + Search Screen Loaded		

Evaluation of Back to Search Screen

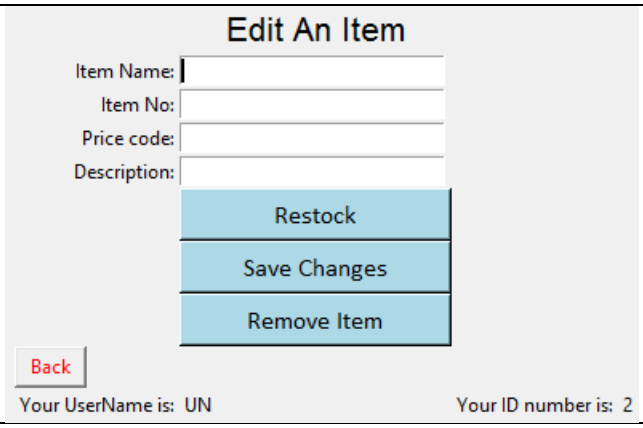
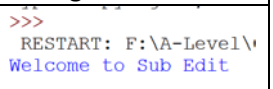
This function is an iteration upon the back to main menu function has it serves the same purpose by returning the given user to their previous screen so they can continue to use that screens functions. As with the other functions this one can also be checked against criteria 7 being to ease the navigation of this program. In this criteria I believe this function succeeds as with its inclusion the user can much more freely operate and as well as it increasing their productivity. This is as they can get back to the search page quicker and can therefore move on to the next item more quickly.

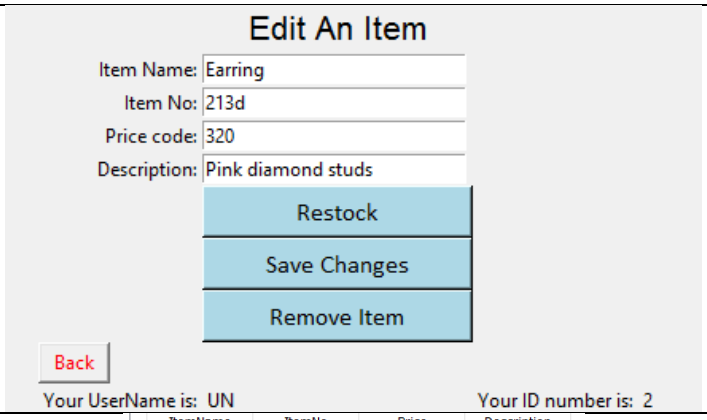
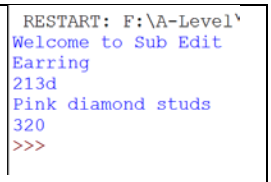
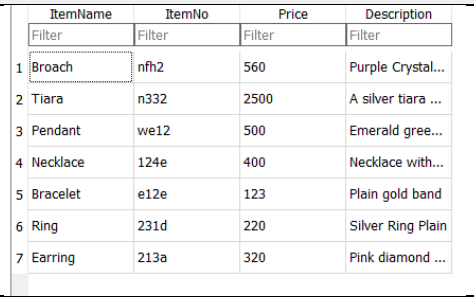
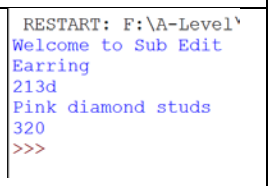
Save Changes

As with other functions whose aim it is to save entered data to the database this function adheres to the criteria point 2. This being that it must “Easily edit any items details”.

Outcome 1:

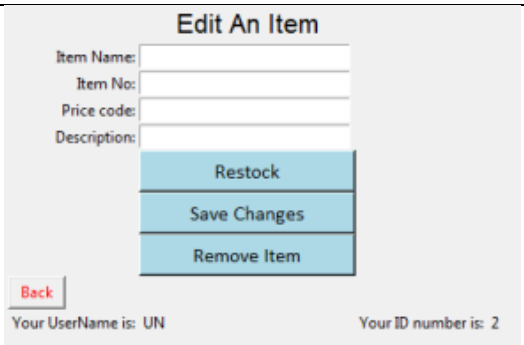
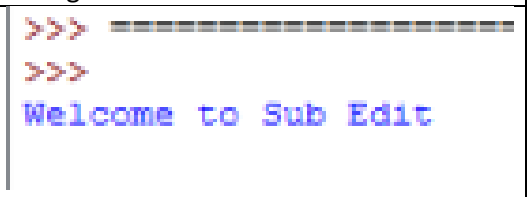
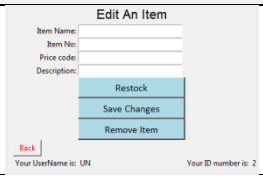
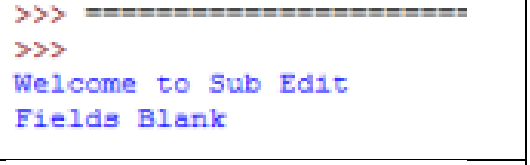
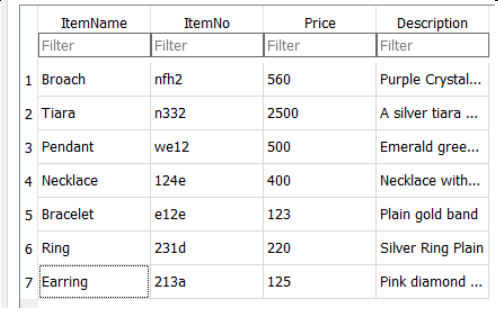
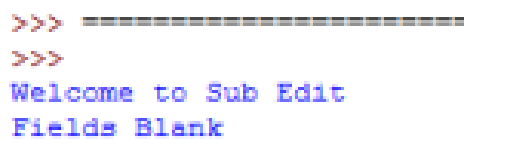
This function should upload all of the edited data entered to the database. The edited information shall be that the price is no longer “220” but “320”.

Stage	Screenshot Of Process	Debug Of Process
1 Edit Item Screen Loaded		

2 Price Changed + Save Changes Pressed		
3 Evidence For Change in the SQL Editor		

Outcome 2:

Fields are left empty to show that no data is taken and nothing I s uploaded.

Stage	Screenshot Of Process	Debug Of Process
1 Edit Item Screen Loaded		
2 Save Changes Pressed		
3 Evidence For Failure Of Change in the SQL Editor		

Evaluation of Save Changes Test

The Save Changes function is very similar to that of the Add item function although instead of just inserting the information as a new record it must replace an existing items information. Since working on the account menu I believe this could be improved to edit specific parts of an items information instead of replacing it all. However in terms of completing its assigned criteria of Criteria 2 it excels. This point is that this function must “Easily edit items details”.

By the results shown in the screenshots and the debugger of Outcome 1 it is clear to see that items can indeed be edited as is asked for within Criteria 2. It is worth noting that the code calls for the information in a strange order though it still adds them to the database correctly. This is all due to the order in which functions are called.

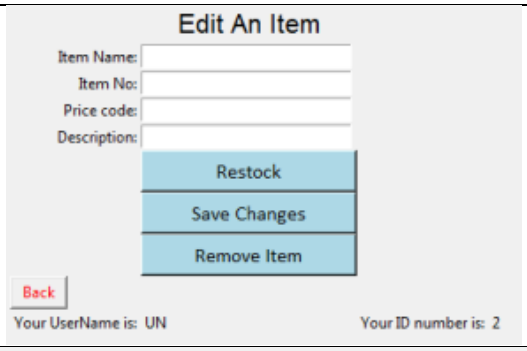
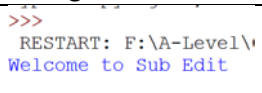
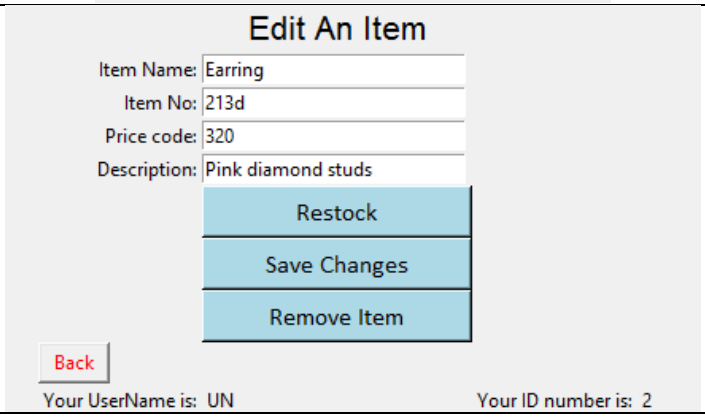
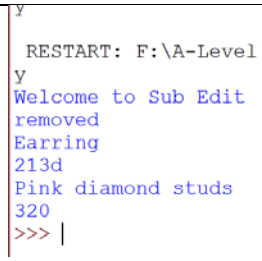
In Outcome 2 the fields have been empty to show that if the fields were left empty nothing was added as there was no way for the program to find the item with data that needed to be edited. This can be seen in the debugger at stage 2 where the included print function says that the fields are blank. From the analysis of all of this data it can be determined that this function has met its criteria.

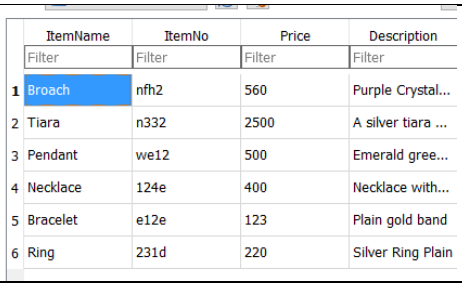
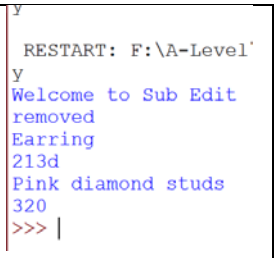
Remove Item

The remove item function’s aim is to remove any unnecessary data from the database. For this reason it doesn’t have a strict criteria to be checked against however it can vaguely be referred to criteria 2 being “Easily edit any items details”. This is as it is somewhat responsible for editing an item’s details.

Outcome 1:

This should remove all of the entered information from the database. This requires multiple screen shots to show the stages of the database as well as the stages of the program.

Stage	Screenshot Of Process	Debug Of Process
1 Edit Item Screen Opened		
2 Remove Item Button Pressed		

3 Evidence For Removal in SQL Editor		

	ItemName	ItemNo	Price	Description
	Filter	Filter	Filter	Filter
1	Broach	nfh2	560	Purple Crystal...
2	Tiara	n332	2500	A silver tiara ...
3	Pendant	we12	500	Emerald gree...
4	Necklace	124e	400	Necklace with...
5	Bracelet	e12e	123	Plain gold band
6	Ring	231d	220	Silver Ring Plain

```
y
 RESTART: F:\A-Level
y
Welcome to Sub Edit
removed
Earring
213d
Pink diamond studs
320
>>>
```

Evaluation of Remove Item Test

From the test of removing an item from the data base by loosely checking it against Criteria 2 it is clear this function operates as it should. This is as in the test of Outcome 1 “Earring” was removed from the database when information was included. However due to the way the function operates there is only so much testing that can be done. This is a sit only takes the Item’s number which is a unique code used as the primary key of the stock’s database. This means there is very little that can go wrong as long as the function is very simple. However I checked it against criteria 2 anyway as it is the closest thing to what the function is able to do, this being to edit an Item’s contents. In a way it does do this via the way the code is written. This is why this function meets its suspected standards.

Account Edit Menu Screen

This screen is the secondary Menu that is used to run all of the functions attributing to opening the account editing screens of the ASC. This is why it has 4 main functions and two sub-functions.

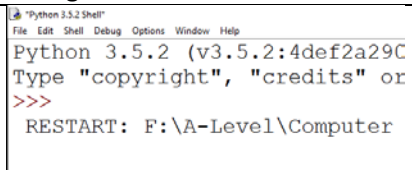
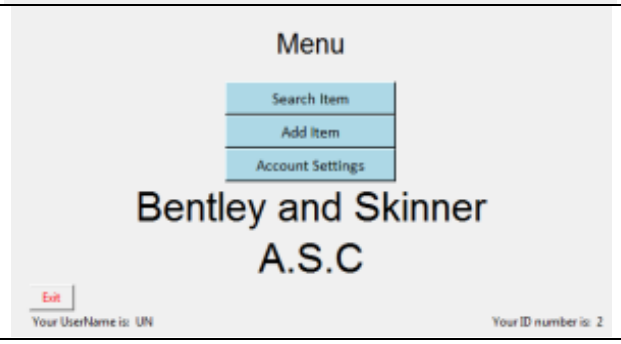
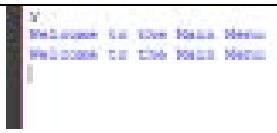
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Main Menu screen	Should remove all of the Account Edit Menu's widgets and create the widgets for the main menu	1)Program goes to Main menu	7.The screens can be navigated easily
Go to Username Edit screen	Should remove all of the Account Edit Menu's widgets and create the widgets for the User Name edit screen	1) Program goes to Username Edit screen	7.The screens can be navigated easily
Go to Password Edit screen	Should remove all of the Account Edit Menu's widgets and create the widgets for the Password edit	1) Program goes to Password Edit screen	7.The screens can be navigated easily
Go to Privileges Edit screen	Should remove all of the Account Edit Menu's widgets and create the widgets for the Privileges edit screen	1) Program goes to Privileges Edit screen	7.The screens can be navigated easily
Generate Privileges and Remove button if Admin	Should check the privileges of the signed in user and if authorised load the specific buttons	1)Program generates remove and privileges buttons 2)Program doesn't generate remove and privileges buttons	3.All information is secure and can only be accessed by authorised personal
Go to Remove Account screen	Should remove all of the Account Edit Menu's widgets and create the widgets for the main menu	1) Program goes to Remove Account screen	7.The screens can be navigated easily

Back to the Main Menu screen

This functions purpose is the same as all of the other back functions being that it must "ease the navigation of the program". This is more commonly associated with criteria 7.

Outcome 1:

When the "Back" button is pressed the screens widgets should be removed and replaced with the Main Menu's widgets

Stage	Screenshot Of Process	Debug Of Process
1 Account Edit Menu Loaded		
2 Back Button Pressed + Menu Loaded		

Evaluation of Back to Main Menu Test

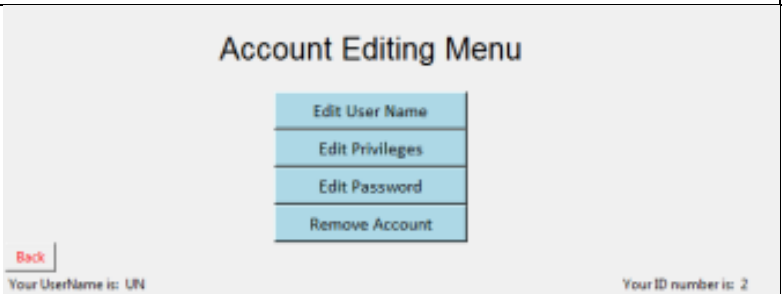
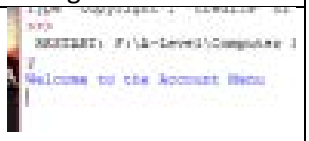
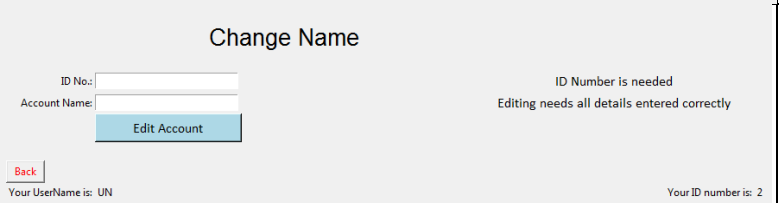
As with previous back to main menu functions this one has been checked against criteria 7 in order to make sure it helps the user move through the system. From the test it is clear this is the response that was gained form this function. This is as it allowed the two menus to be switched between depending on the given task. This therefore makes the life of the user to be much easier and therefore completes the given criteria.

Go to Username Edit screen

This “go to” functions purpose is to help the user go from one page to another. This is easily a function built for criteria 7 being “The screens can be navigated easily”.

Outcome 1:

When clicking the “Edit Username” button widgets of the “Account Menu” are removed and replaced with the ones for the “Edit Username” screen.

Stage	Screenshot Of Process	Debug Of Process
1 Account Edit Menu Pressed		
2 Edit User Name Button		

Pressed + Change Name Screen Loaded		

Evaluation of Go To Username Edit Test

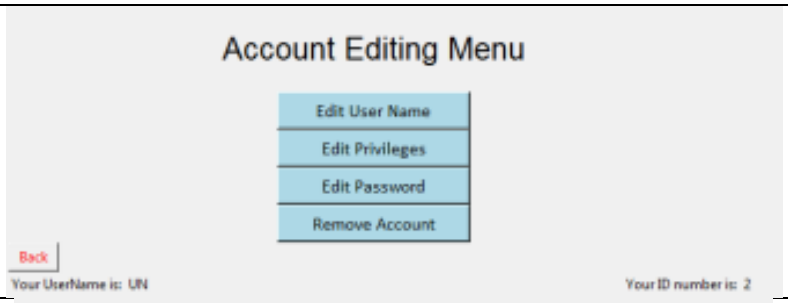
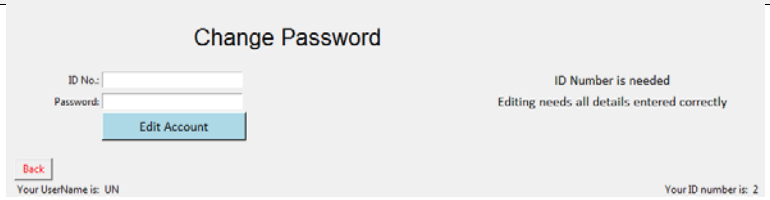
This function is taken directly from the main menu as it is important for the user to be able to switch between the included screens. This is the main point of criteria 7 which is to allow the user to move between screens as easily as possible and so by adding this I believe this has been completed.

Go to Password Edit screen

Criteria 7 is "The screens can be navigated easily" and this is "go to" function has specifically been built around this purpose.

Outcome 1:

When clicking the "Edit Password" button widgets of the "Account Menu" are withdrawn and replaced with the ones for the "Edit Password" screen.

Stage	Screenshot Of Process	Debug Of Process
1 Account Edit Screen Opened		
2 Edit Password Clicked + Change Password Screen Made		

Evaluation Go To Password Edit Test

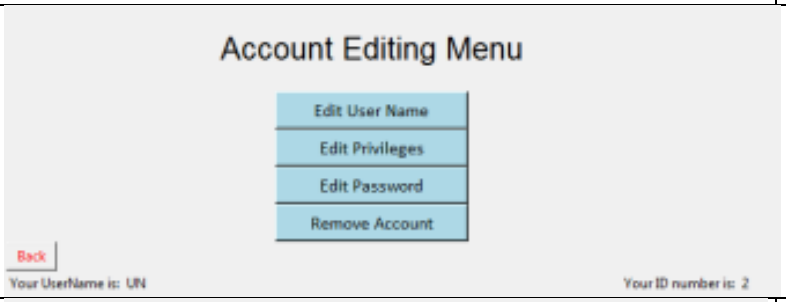
The creation of this function was for the intended purpose of fulfilling Criteria 7 which was stated at the beginning of this screen test. In hindsight to this I believe there were very little improvements that had to be made due to simplicity of the function. This is as the function does meet the standards portrayed by the criteria.

Go to Privileges Edit screen

"The screens can be navigated easily" is what criteria point 7 is and portrays why it is so important for this function to be included. This is as without it the program would cease to function at 100%.

Outcome 1:

When clicking the "Edit Privileges" button widgets of the "Account Menu" are destroyed and replaced with the ones for the "Edit Privileges" screen.

Stage	Screenshot Of Process	Debug Of Process
1 Account Edit Menu Deployed		
2 Edit Privileges Activated		

Evaluation Go To Privileges Test

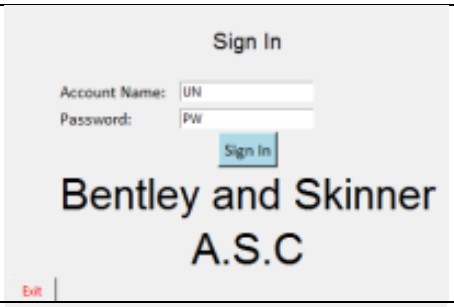
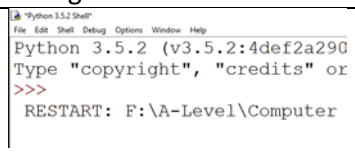
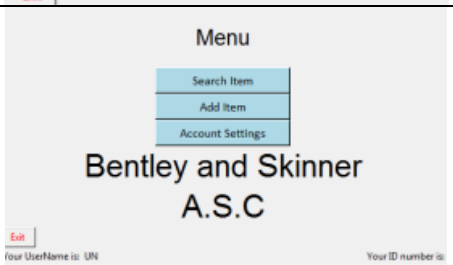
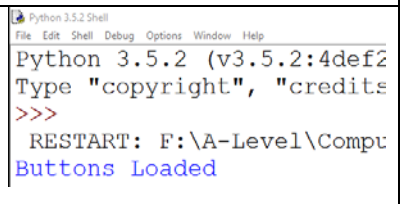
This function successfully meets the criteria 7 that it was made for as can be seen in the given screenshots in the table above. This is as there is full evidence of the function operating to the standards that was given to it in criteria 7.

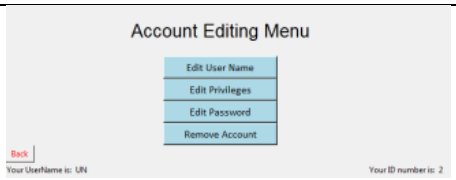
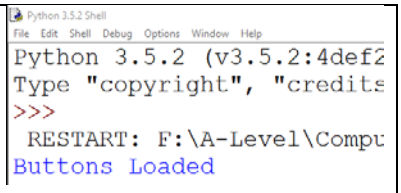
Generate Privileges and Remove button if Admin

This a function that was added towards the end of the iteration process due to concerns of the program being unsecure. Therefore to address that this function was made so that criteria 3 “All information is secure and can only be accessed by authorised personal” was met.

Outcome 1:

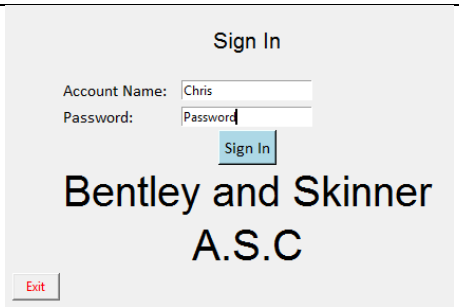
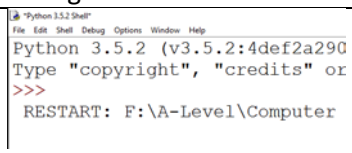
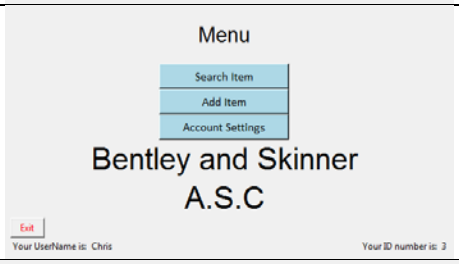
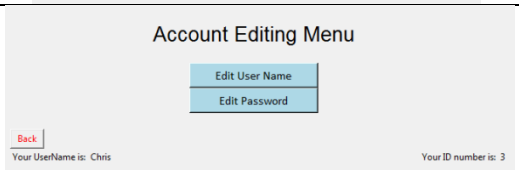
User signs in with an account with “Admin” privileges and goes to the “Account Edit Menu” by pressing the button on the “Main Menu”. Proof of this shall be registered in the Debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Sign In Screen Opened + Test User Entered		
2 Sign In Pressed + Admin Privileges Given		

3 Account Settings Pressed + Widgets Loaded		

Outcome 2:

User signs in with an account with “User” privileges before navigating to the “Account Edit Menu”. A message will be displayed in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Sign In Screen Made + Fields Filled In		
2 Sign In Clicked + User Privileges Given		>>> RESTART: F:\A-Level\Compute Buttons not Loaded
3 Account Settings Pressed + Account Edit Menu Widgets Loaded		>>> RESTART: F:\A-Level\Compute Buttons not Loaded

Evaluation of Generating Privileges and Remove Buttons Test

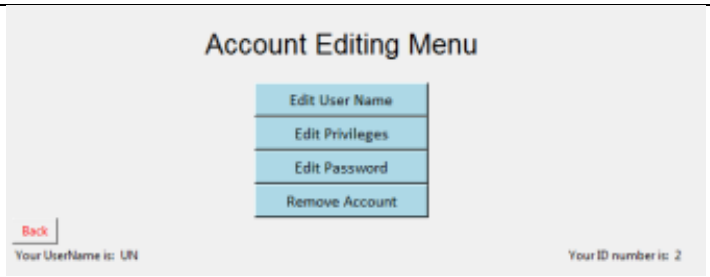
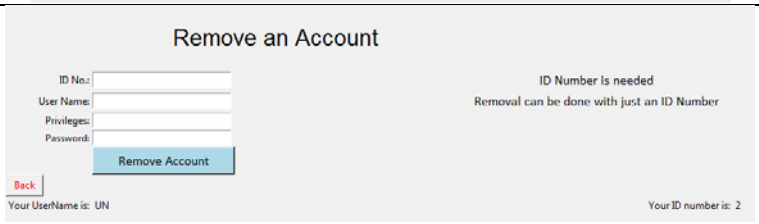
In order to raise the security of the program this function was implemented in to the system as the ability to change privileges and remove users should be reserved to administrators’ only. This is with these functions the whole system could be brought down. This is important when this function is compared against criteria 3 which is all to do with the security of the program and not for it to be accessed by unauthorised individuals. In terms of this and the success of the function I can say that it is operating at 100%. This is as the evidence provided shows anyone who logs on with “user” privileges e.g. the “Chris” test user finds themselves to have revoked access to these widgets.

Go to Remove Account screen

For the system to operate effectively it must meet criteria point 7 “The screens can be navigated easily”, this is as without the less technically operable would struggle to move around the system.

Outcome 1:

By clicking the “Remove Account” button the widgets for the menu will be deleted and replaced with the ones for the “Remove Account” screen.

Stage	Screenshot Of Process	Debug Of Process
1 Account Edit Menu Loaded		
2 Remove Account Pressed + Remove Account Screen Loaded		

Evaluation of Got To Remove Account Test

The ability to switch from a menu to a side screen is very important as without it the menu is flawed. This is as the main purpose of it is to move between different screens and hold multiple functions that do this. Therefore if one of these held functions no longer operates then the whole page falls apart.

Username Edit Screen

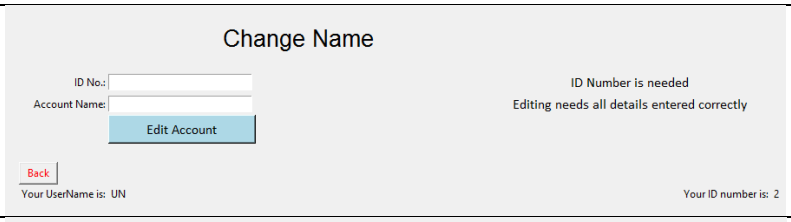
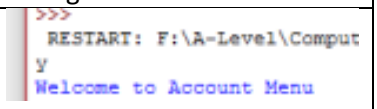
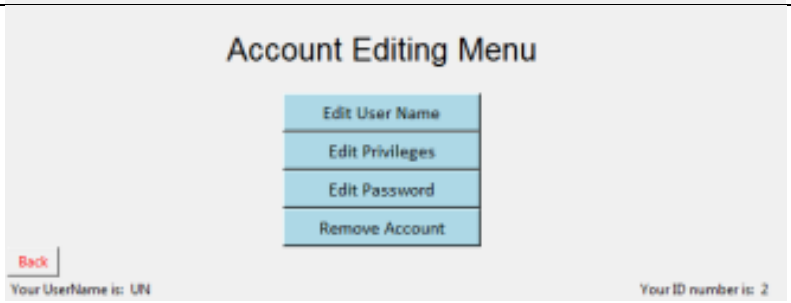
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Account Menu screen	Should remove all of the Username Edit screen's widgets and create the widgets for the Account menu	1)Program goes to Account menu	7.The screens can be navigated easily
Save Changes	Takes all data from the entry fields and saves it to the appropriate place in the user database	1)Data gets saved to the user database 2)Data is not saved to the user database due to blank fields	2.Easily edit any items details

Back to the Account Menu screen

"The screens can be navigated easily" is the point of criteria 7 that must be made in order for the ASC to succeed. Without this critical point being met then the system would surely fall apart.

Outcome 1:

The "Back" button is pressed in order to remove all of the widgets from the "Change Name" screen and reload the widgets from the "Account Edit Menu".

Stage	Screenshot Of Process	Debug Of Process
1 Change Name Screen Deployed		
2 Back Button Pressed + Account Edit Menu Loaded		

Evaluation of Back to Account Menu Test

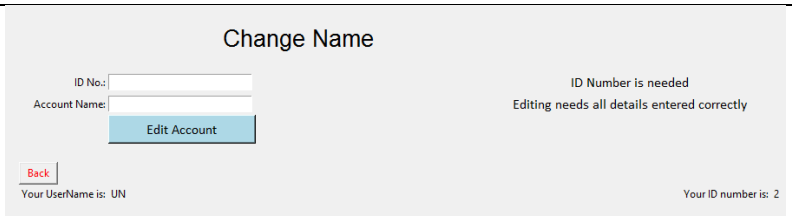
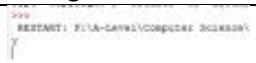
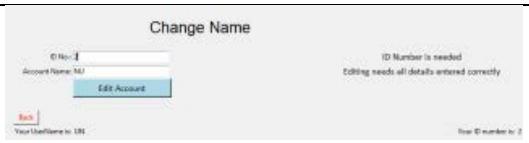
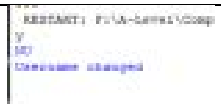
As stated in the previous evaluation the main purpose for a menu is for screens to be viewed via it. Therefore if you can't remove yourself from that screen after you have entered it what is the point of having it in the first place as it would merely becoming a one use function. By having this button it allows that menu to have multiple uses as it can continually be returned to. Due to this the ease of navigation increases which is the basics of criteria 7.

Edit Account

In order for the solution of the problem to be fully functioning it must meet all criteria points. The main one of these being that it must "Easily edit any items details" also known as point 2.

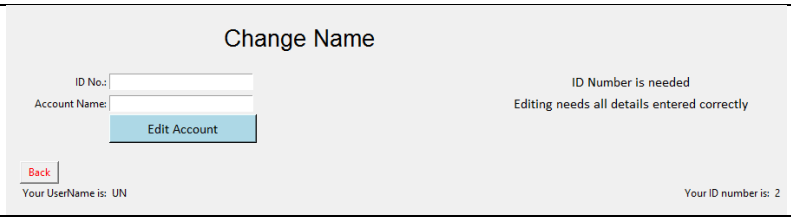
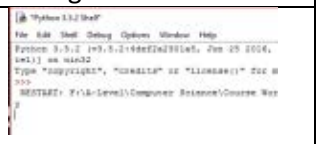
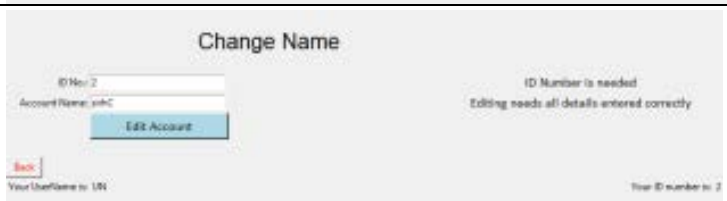
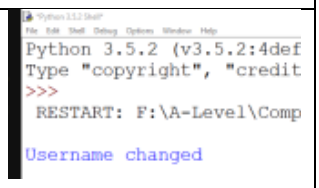
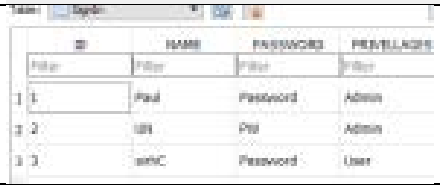
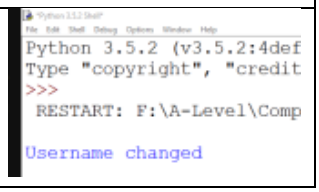
Outcome 1: Test 1:

Here I have entered some test data in order to prove that it has been saved to the appropriate area of the database. This data being that "UN" is being changed to "NU".

Stage	Screenshot Of Process	Debug Of Process
1 Change Name Screen Loaded + Information Entered		
2 Edit Account Pressed		
3 Test Data Shown In SQL Editor		

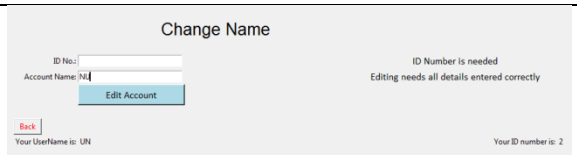
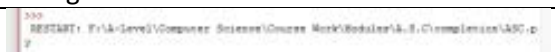
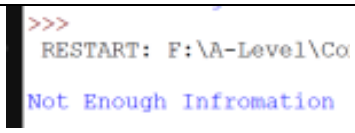
Outcome 1: Test 2:

Here I have entered a second lot of test data in order to prove that it can be saved to the correct row of the database. This data being that “Chris” is changed to “sirhC”.

Stage	Screenshot Of Process	Debug Of Process
1		
2		
3		

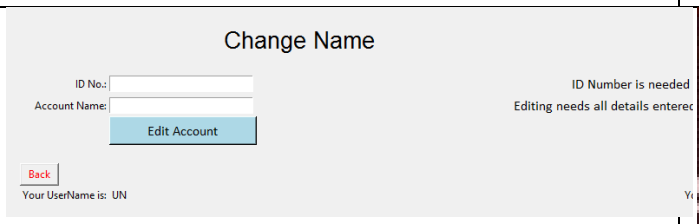
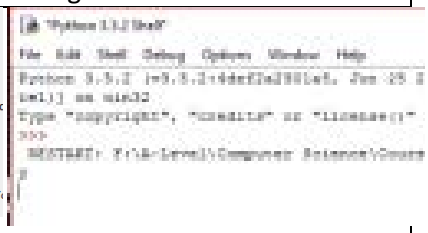
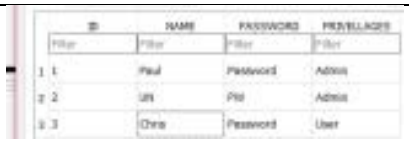
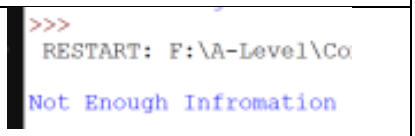
Outcome 2: Test 1:

Here I have left the ID field blank in order to prove that it can’t be saved without the appropriate information. This generates a string in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Change Name Screen Loaded + One Field Filled		
2 Edit Account Pressed + Evidence Of Failure		

Outcome 2: Test 2:

Here I have left all entry fields blank in order to prove that it can't be saved to the database. This will lead to some text being created in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Change Name Screen Opened + Fields Left Blank		
2 Edit Account Pressed + Evidence Of Failure		

Evaluation of Edit Account Test

The edit account function is very similar to that of the save changes function although it doesn't apply to stock items and instead to the actual user's information. This is as the code of both of the functions is almost identical apart from small alterations in the intended destination. For this reason they also have the same criteria this being criteria number 2. Which is the ability to easily edit a given items details although in this case that item is a user.

From Outcome 1 we can see how the username can easily be changed to something else which in these cases was the reverse of the original. However in Outcome 2 we can see how the function can fail when not all fields are filled in. The error response in the debugger was done by inserting a print function into a given if statement. This is as the function works by using the ID number to find the appropriate user to edit. This does mean that the user can set their name to be blank. Other than this small error I say that the function has been a full success.

Password Edit Screen

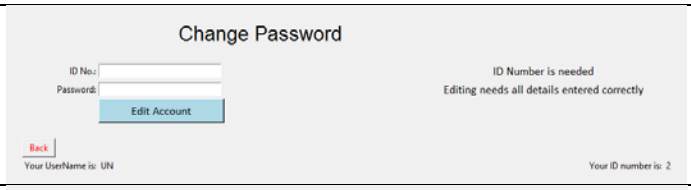
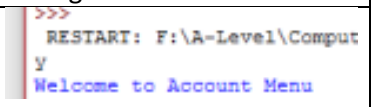
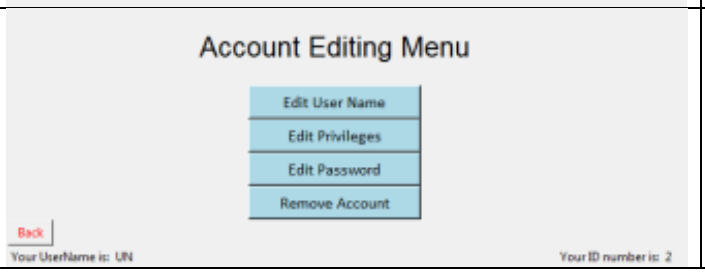
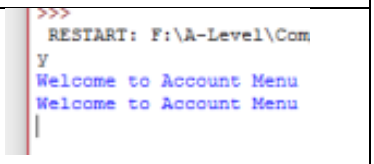
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Account Menu screen	Should remove all of the Password Edit screen's widgets and create the widgets for the Account menu	1)Program goes to Account menu	7.The screens can be navigated easily
Edit Account	Takes all data from the entry fields and saves it to the appropriate place in the user database	1)Data gets saved to the user database 2)Data is not saved to the user database due to blank fields	2.Easily edit any items details

Back to the Account Menu screen

The same with all previous back functions this one must be run against criteria 7 "The screens can be navigated easily".

Outcome 1:

This test involves the pressing of the "Back" button in order to return to the menu and load all of its widgets whilst removing the "Change Password's" widgets.

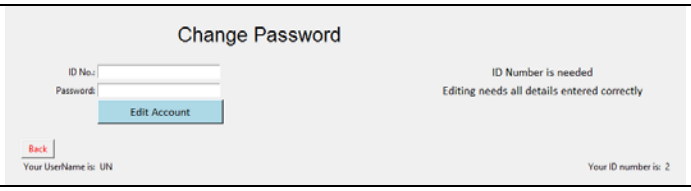
Stage	Screenshot Of Process	Debug Of Process
1 Change Password Screen Loaded		
2 Back Button Clicked		

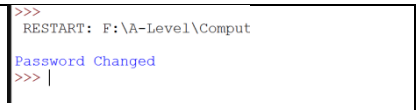
Edit Account

In order for this screen to be successful it must make the user easily be able to edit any items details which is the fundamentals of criteria 2. This is why this function was included here on this screen.

Outcome 1: Test 1:

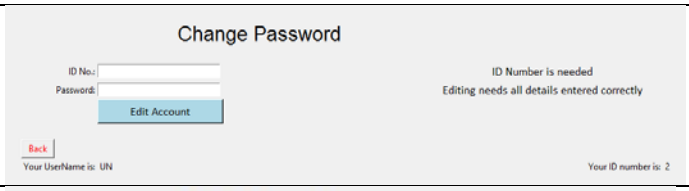
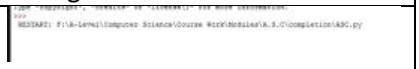
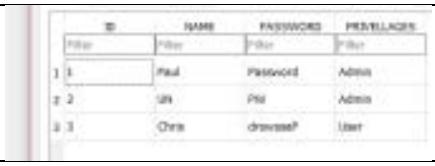
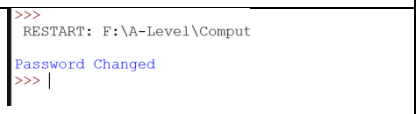
Here I have entered some test data in order to prove that it has been saved to the appropriate area of the database. This data being that 2pw2 has changed to "WP".

Stage	Screenshot Of Process	Debug Of Process
1 Change Password Screen Loaded		

2 Information Entered and Edit Account Pressed		
3 Evidence for Completion		

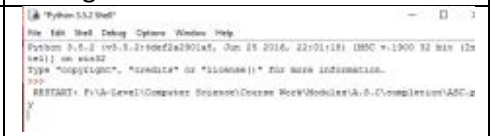
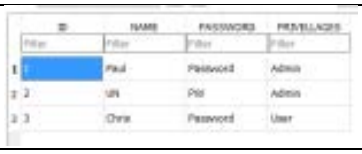
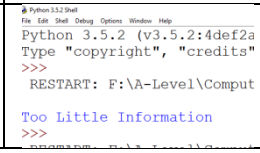
Outcome 1: Test 2:

Here I have entered a second lot of test data in order to prove that it can be saved to the correct row of the database. This data being:

Stage	Screenshot Of Process	Debug Of Process
1 Change Password Screen Loaded		
2 Information Entered and Edit Account Pressed		
3 Evidence for Completion		

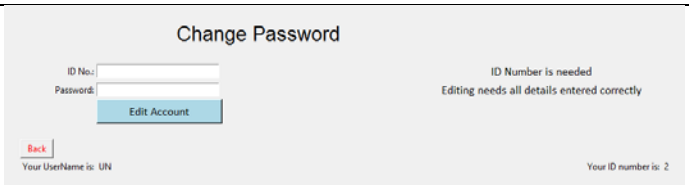
Outcome 2: Test 1:

Here I have left the ID field blank in order to prove that it can't be saved without the appropriate information.

Stage	Screenshot Of Process	Debug Of Process
1 Change Password Screen Loaded and Password entered		
2 Evidence for Failure		

Outcome 2: Test 2:

Here I have left all entry fields blank in order to prove that it can't be saved to the database.

Stage	Screenshot Of Process	Debug Of Process
1 Change Password Screen Loaded and left blank		
2 Evidence for Failure		

Evaluation of Edit Account Test

The edit account function across all of the account editing screens is very simplistic and similar in their designs. This is as they all use the same coded solution just with different database locations and information entering it.

From the Outcome 1 tests we can see that the users password can easily be changed as the information in the SQL database has been changed as well as the print functions in the debugger. In outcome 2 the flaws of the function can be seen. This being that information can't be added without the ID number. The only reason that this occurs is the way in which the coded solution has been generated. Due to this I still believe that this meets it's given criteria of 2 which involves how easily it is to edit an items details. Seeing at the simplicity of how this is executed in Outcome 1 I have reason to say this has been completed.

Privileges Edit Screen

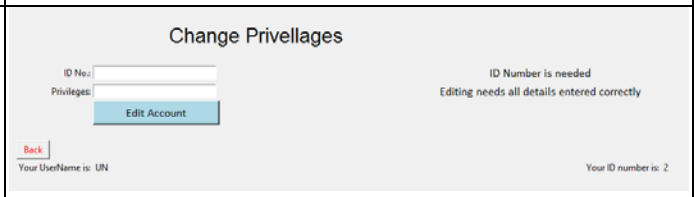
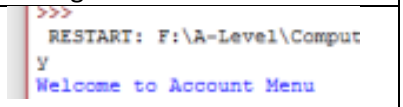
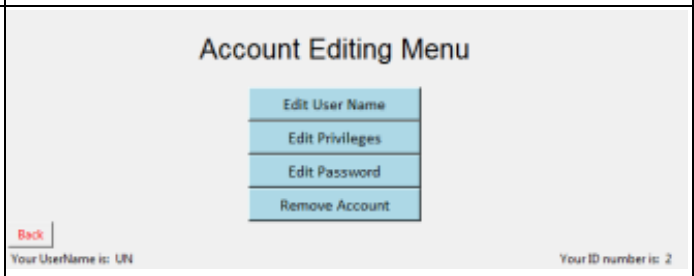
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Account Menu screen	Should remove all of the Privileges Edit screen's widgets and create the widgets for the Account menu	1)Program goes to Account menu	7.The screens can be navigated easily
Edit Account	Takes all data from the entry fields and saves it to the appropriate place in the user database	1)Data gets saved to the user database 2)Data is not saved to the user database due to blank fields	2.Easily edit any items details 3.All information is secure and can only be accessed by authorised personal

Back to the Account Menu screen

The main existence for this function for this function is the criteria point 7, "The screens can be navigated easily", therefore by adding this function and widget this can be shown to be true.

Outcome 1:

This test has the simple process of me clicking the "Back" button to clear all widgets before loading the "Account Edit Menu's" widgets.

Stage	Screenshot Of Process	Debug Of Process
1 Change Privileges Screen Opened		
2 Back Widget Clicked + Account Edit Menu Loaded		

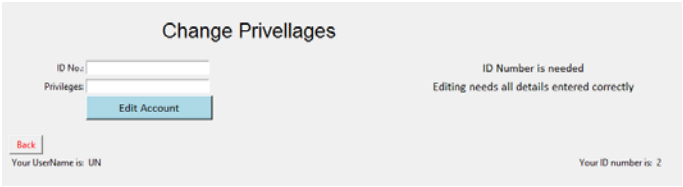
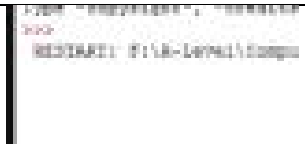
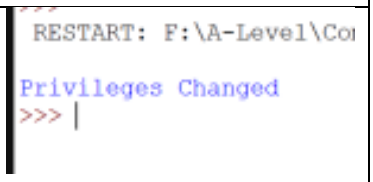
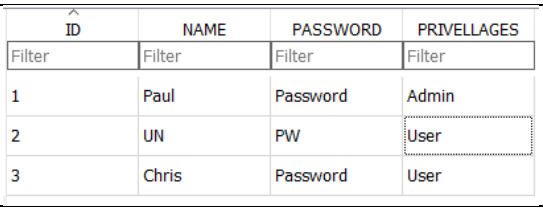
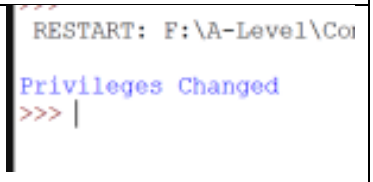
Edit Account

Criteria 7 "Easily edit any items details" is the main function of this screen and therefore is the reason why it must be checked against it.

Outcome 1: Test 1:

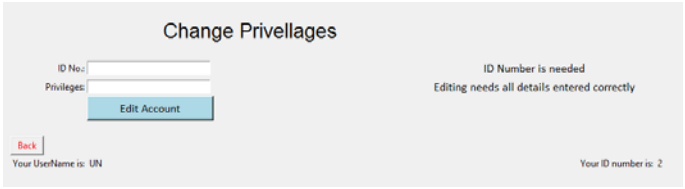
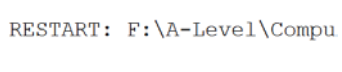
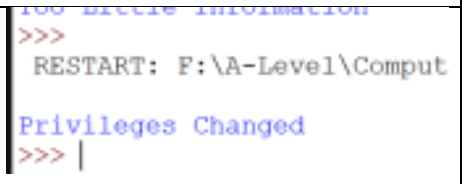
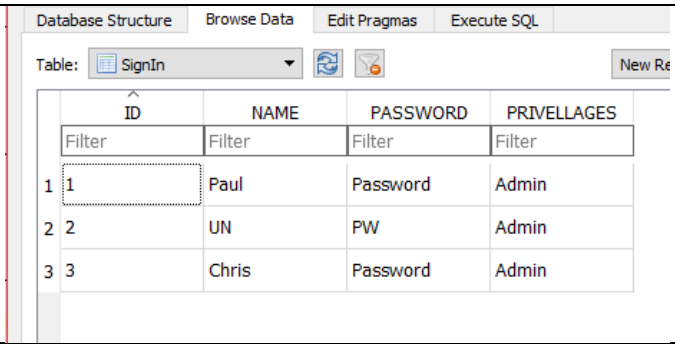
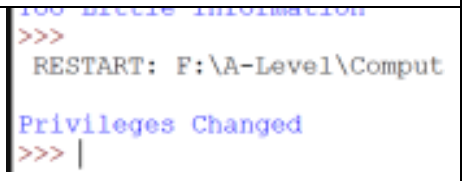
Here I have entered some test data in order to prove that it has been saved to the appropriate database. This data being "Admin" changing to "User".

Stage	Screenshot Of Process	Debug Of Process

1 Change Privileges Screen Loaded		
2 Info entered and function run		
3 Evidence of Success		

Outcome 1: Test 2:

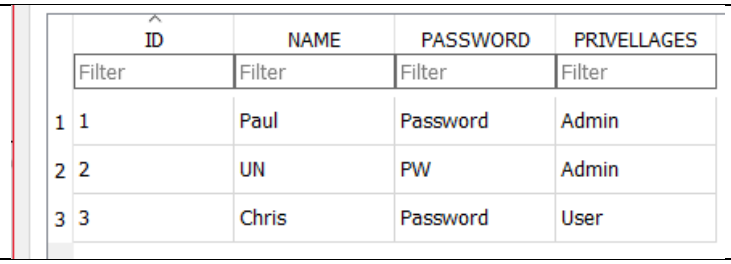
Here I have entered a second lot of test data in order to prove that it can be saved to the appropriate database. This data being “User” changing to “Admin”.

Stage	Screenshot Of Process	Debug Of Process
1 Change Privileges Screen Loaded		
2 Info entered and function run		
3 Evidence of Success		

Outcome 2: Test 1:

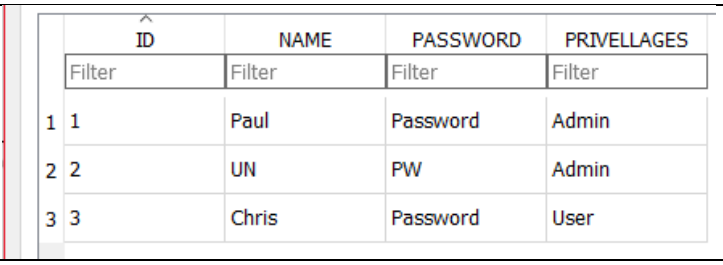
Here I have left the ID field blank in order to prove that it can’t be saved to the appropriate database.

Stage	Screenshot Of Process	Debug Of Process

1 Change Privileges Screen Loaded and info entered		RESTART: F:\A-Level\Compu
2 Evidence of Failure		RESTART: F:\A-Level\Comp Too Little Information

Outcome 2: Test 2:

Here I have left all entry fields blank in order to prove that it can't be saved to a database.

Stage	Screenshot Of Process	Debug Of Process
1 Change Privileges Screen Loaded		
2 Evidence of Failure		RESTART: F:\A-Level\Comp Too Little Information >>>

Evaluation of Edit Account Test

This function allows for the editing of a user's privileges which is an important factor when it comes to the ability of editing an item as well as the security of the program. For example if a user was abusing their rights then they could be set to user. This is why this function runs against criteria 2 and 3 as 2 is about the ease of which these details can be changed whilst 3 is about the security of the program.

Throughout both Outcomes 1 and 2 both of these can be seen to have been met by this function. This is as an admin can change the privileges to of a user as long as they have the id as with other edit account functions. However unless they have administrator rights they would not have access to this function has the button would not load as shown in the account menu screens test.

Remove Account Screen

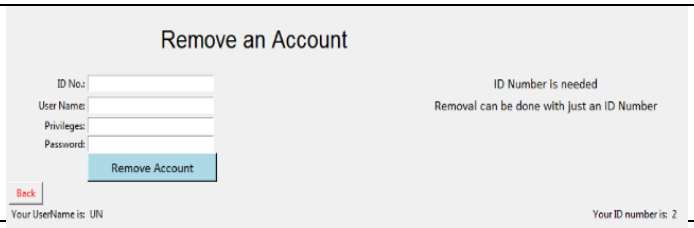
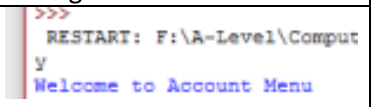
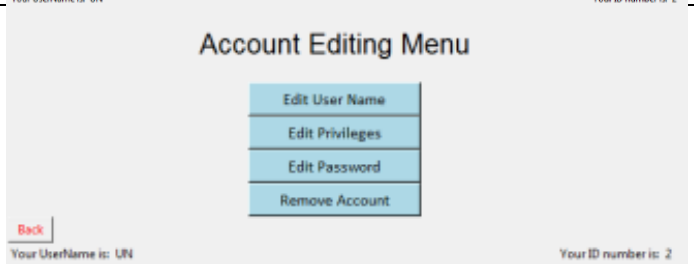
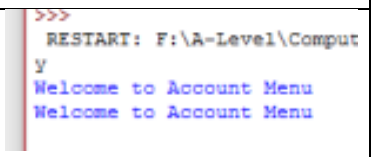
Function	Function Description	Possible Outcomes	Criteria Points
Back to the Account Menu screen	Should remove all of the Remove Account screen's widgets and create the widgets for the Account menu	1)Program goes to Account menu	7.The screens can be navigated easily
Remove Account	Should take all data from the entered fields and remove it into the database	1)Data is removed from database 2) Data isn't removed due to fields being empty	2.Easily edit any items details

Back to the Account Menu screen

For this back function to be successful as with all of the others it must be checked against criteria point 7 "The screens can be navigated easily".

Outcome 1:

For this test I simply click the "Back" button to remove all of the "Remove" screen's widgets and restore "Account Menu's" widgets. A message will be displayed in the debugger to prove that this has been done.

Stage	Screenshot Of Process	Debug Of Process
1 Remove Account Screen Loaded		
2 Back Widget Activated + Account Edit Menu's Widget's Loaded		

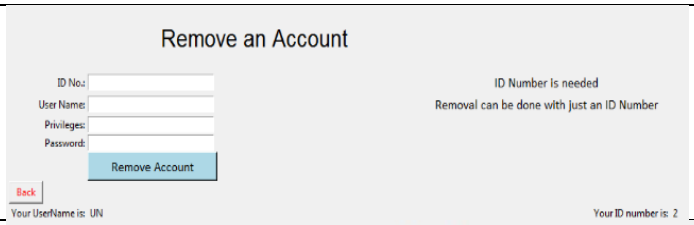
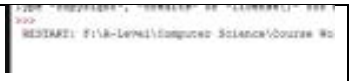
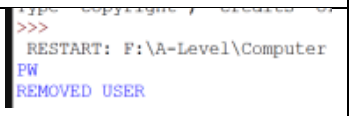
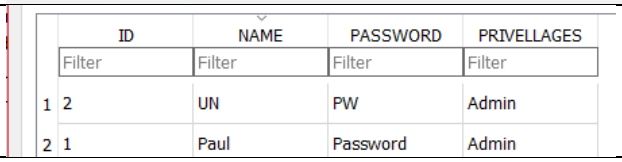
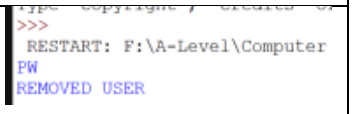
Remove Account

As with other "Remove" functions there is not really any criteria that it was made for. Due this reason the closest it can be run against is point 2 "Easily edit any items details".

Outcome 1: Test 1:

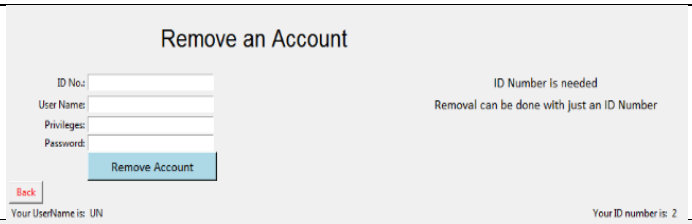
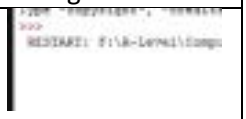
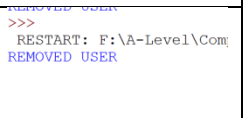
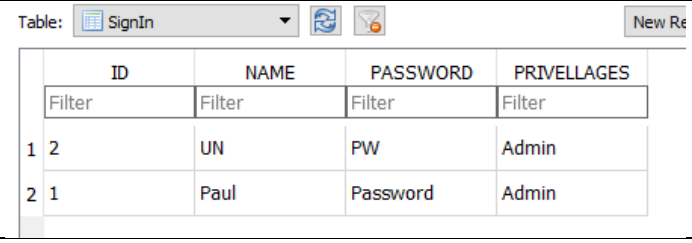
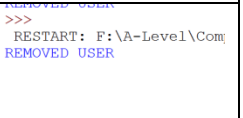
Some test data has been entered into the fields to be removed from the database in order to prove the function works. This shall prompt a response in the debugger. The following data is to be removed:

Stage	Screenshot Of Process	Debug Of Process

1 Remove Account Screen Loaded		
2		
3		

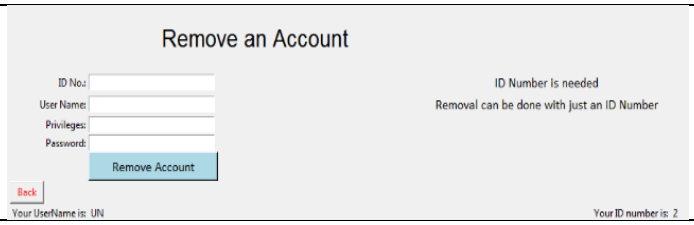
Outcome 1: Test 2:

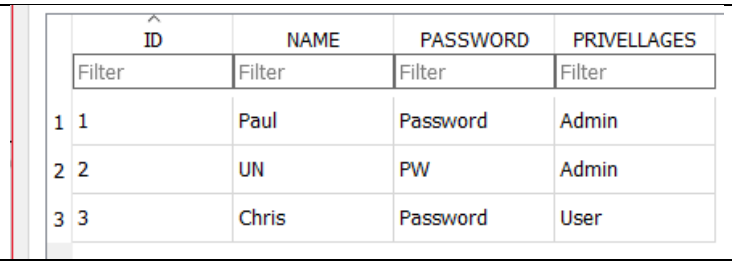
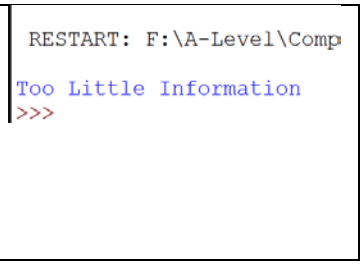
Test data has been entered into the fields to be removed from the database in order to prove the function works. This shall prompt a response in the debugger. The following data is to be removed:

Stage	Screenshot Of Process	Debug Of Process
1 Remove Account Screen Loaded		
2 Information entered and function deployed		
3 Evidence of Success		

Outcome 2: Test 1:

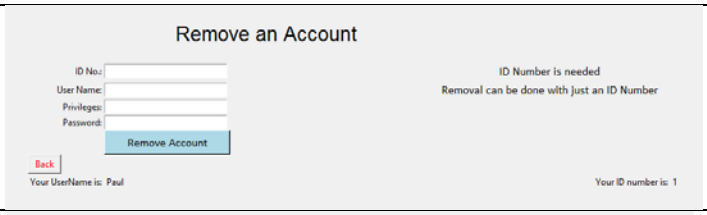
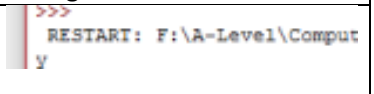
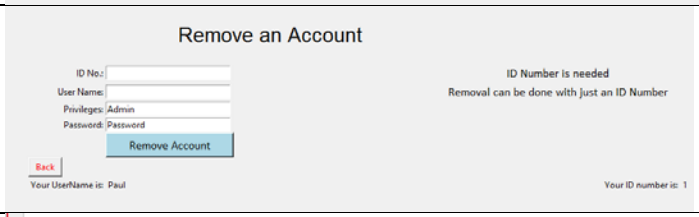
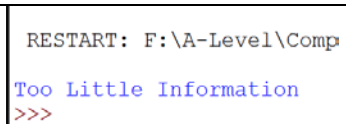
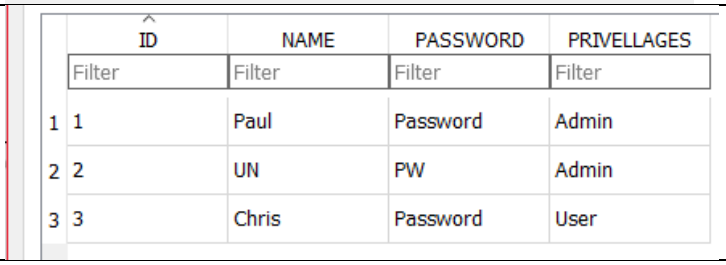
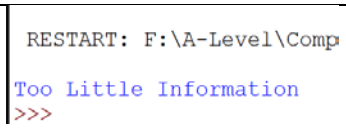
To show that the account cannot be removed with empty fields all of the fields have been left blank. This shall give an automated response in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Remove Account Screen Loaded		

2 Evidence of failure	 	

Outcome 2: Test 2:

To show that the account cannot be removed with empty fields half of the fields have been left blank. This shall give an automated response in the debugger.

Stage	Screenshot Of Process	Debug Of Process
1 Remove Account Screen Loaded	 	
2 Information edited and function activated	 	
3 Evidence of failure	 	

Evaluation of Remove Account Test

In terms of removing a user form the system this program excels. This allows it to complete its assigned criteria of criteria 2 as the process is made simple. On top of this the steps to remove has been extended to stop accidental removal of a user. This can be seen in the outcome 1 test where extra pop up boxes appear to make sure that the user who is removing is sure they wish to remove them.

This function shares traits to that of the edit account functions as they all cannot be activated without the input of the users ID. This is due to them all sharing the same base code. This does mean that the user can remove someone if only their ID is entered although this could be improved in the future. However aside from this the function still meets criteria 2 as it is easy to edit the items contents.

Full Code Annotation

Listed below is all of the code from the ASC program complete with full annotations. Main screens of the code have been given headings so you may find them in the contents page and navigate to them quickly. The majority of indentation has been preserved within the relative modules although due to lines being too long they may have dropped to the line below.

```
from tkinter import *

import smtplib

import sqlite3

class ASC(Frame):

   def createWidgets(self):

#Start of the Main Menu Screen

      def mainMenu(): #Creation of Main Menu screen

         def removeMenu(): #Function to call the removal of the main menus widgets

            ExitASC.grid_forget()#Functions to remove the main menus widgets

            List.grid_forget()

            Add.grid_forget()

            Account.grid_forget()

            title1.grid_forget()

            title2.grid_forget()

            Menu.grid_forget()

         def showMenu(): #Function to call for the creation of the Main Menu

            mainMenu()

#Add Item Screen Start

         def makeAdd(): #Creation of the Add Item Screen

            List.grid_forget()#Back up functions for the removal of the Main Menu in case the remove menu function fails

            Add.grid_forget()

            Account.grid_forget()

            title1.grid_forget()

            title2.grid_forget()

            Menu.grid_forget()

            def removeAdd(): #Function to call for the removal of the Add Item Screens widgets
```

```
    BackAdd.grid_forget()#Functions to remove each of the Add Item Screens widgets
    item.grid_forget()
    itemLabel.grid_forget()
    itemNo.grid_forget()
    itemNoLabel.grid_forget()
    price.grid_forget()
    priceLabel.grid_forget()
    description.grid_forget()
    descriptionLabel.grid_forget()
    Addbtn.grid_forget()
    AddTitle.grid_forget()
    showMenu()#Function to call for the creation of the Main Menu
def getName():#Function to retrieve the name
    Name=NameInput.get()#Retrieves name from the entry field
    if Name==(""):
        print ("Blank Field")#If field is blank process returns nothing
        return
    else:
        global AddName
        AddName=NameInput.get()
        getNo()#Call for the function to retrieve the items no from the entry field
        return AddName#Returns the name from the entered field


def getNo():#Function to retrieve the Items No
    No=NoInput.get()#Retrieves the items no from the entry field
    if No==(""):
        print ("Blank Field")#If field is blank process returns nothing
        return
    else:
        global AddNo
```

```
            AddNo=NoInput.get()
            getPrice()#Call for the function to retrieve the Price of the item from the entry field
            return AddNo#Returns the items no from the entered field


        def getPrice():#Function to retrieve the Items Price
            Price=PriceInput.get()#Retrieves the price from the entry field
            if Price==(""):
                print ("Blank Field")#If field is blank process returns nothing
                return
            else:
                global AddPrice
                AddPrice=PriceInput.get()
                getDesc()#Call for the function to retrieve the Description of the item  from the entry
field
                return AddPrice#Returns the items price from the entered field

        def getDesc():#Function to retrieve the Items Description
            Desc=DescInput.get()#Retrieves the description of the item from the entry field
            if Desc==(""):
                print ("Blank Field")#If field is blank process returns nothing
                return
            else:
                global AddDesc
                AddDesc=DescInput.get()
                return AddDesc#Returns the items description from the entered field

        def addItem():#Function to add details to the database
            getName()#Call for the function to retrieve the name froim the entry field
            print(AddName)
            print(AddNo)
```

```
            print(AddPrice)

            print(AddDesc)

            conn = sqlite3.connect('ASC.db')#Command to connect to the stock data base of name
ASC.db

            conn.execute ("INSERT INTO Stock VALUES (?,?,?,?)",
([AddName],[AddNo],[AddPrice],[AddDesc]))#Command to execute the SQL to add four variables
into the database

            conn.commit()

        NameInput=StringVar()#These four set the inputs to be string variables

        NoInput=StringVar()

        PriceInput=StringVar()

        DescInput=StringVar()

        BackAdd= Button(self, width=5, fg="red", text="Back", command=removeAdd)#Creation of
the back button widget and its attributes

        BackAdd.grid (sticky=W,row=10, column=1)

        AddTitle= Label(self, text="Add Item", font="calibri, 16", pady="20")#Creation of the Add
Items title label and its attributes

        AddTitle.grid (row=1,column=2)

        item= Entry(self, width=27,textvariable=NameInput)#Creation of the items name entry
widget and its attributes

        item.grid (sticky=W,column=2,row=2)

        itemLabel= Label(self, text="Item Name:")#Creation of the Item Names entry label and its
attributes

        itemLabel.grid (sticky=E,column=1, row=2)

        itemNo= Entry(self, width=27,textvariable=NoInput)#Creation of the items No entry widget
and its attributes

        itemNo.grid (sticky=W,column=2,row=3)

        itemNoLabel= Label(self, text="Item Code:")#Creation of the Items No entry label and its
attributes

        itemNoLabel.grid (sticky=E,column=1, row=3)

        price= Entry(self, width=27,textvariable=PriceInput)#Creation of the items price entry
widget and its attributes

        price.grid (sticky=W,column=2,row=4)

        priceLabel= Label(self, text="Price:")#Creation of the Item Price entry label and its
attributes
```

priceLabel.grid (sticky=E,column=1, row=4)

description= Entry(self, width=27,textvariable=DescInput)#Creation of the items description entry widget and its attributes

description.grid (sticky=W,column=2,row=5)

descriptionLabel= Label(self, text="Description:")#Creation of the Items Description entry label and its attributes

descriptionLabel.grid (sticky=E,column=1, row=5)

Addbtn= Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=addItem)#Creation of the Add Items add button widget

Addbtn.grid (sticky=E,column=2,row=6)

End of Add Item Screen

#Start of Account Menu Screen

def makeAccountMenu():#Creation of the account menu

List.grid_forget()#Back up functions for the removal of the Main Menu in case the remove menu function fails

Add.grid_forget()

Account.grid_forget()

title1.grid_forget()

title2.grid_forget()

Menu.grid_forget()

ExitASC.grid_forget()

#Start of Remove Account Screen

def RemoveAccount():#Creation of Remove Account screen

def RemoveRemove():#Functiom to remove the remove account screens widgets

ToAMenu.grid_forget()#Functions that will remove the remove account screens widgets

ID.grid_forget()

idLabel.grid_forget()

accountMenu.grid_forget()

accountLabel.grid_forget()

privileges.grid_forget()

privilegesLabel.grid_forget()

```
            pword.grid_forget()
            pwordLabel.grid_forget()
            Remove.grid_forget()
            Inst1.grid_forget()
            Inst2.grid_forget()
            RemTitle.grid_forget()
            showAccountMenu()#Calling of function to remove any widgets that may be left over from previous jump
            makeAccountMenu()#Calling of function to reload the account menus widgets
        def remove():#Function to create pop-up box to tell user the account has been removed
            pop = Toplevel()#Creation of pop up box
            pop.title("REMOVED")
            msg = Label(pop, text="THIS ACCOUNT HAS BEEN REMOVED")#Creation of message for pop up box
            msg.pack()
            ok = Button(pop, text="Dismiss", command=pop.destroy)#Creation of button to close the pop-up box
            ok.pack()
            ID=idInput.get()#commands to remove the account from the database
            conn = sqlite3.connect('signin.db')
            conn.execute("DELETE from SignIn WHERE ID='%s'" %ID)
            conn.close()
            print("REMOVED USER")
        def warning():#Function to create a pop-up box to warn against removal
            pop = Toplevel()#Creation of the pop-up box
            pop.title("WARNING")
            msg = Label(pop, text="THIS WILL REMOVE ALL ACCOUNT DETAILS, ARE YOU SURE?")#Creation of message to appear in the pop-up box
            msg.pack()
            yes = Button(pop, text="I Am Sure",command=remove, bg="light blue")#Creation of yes button to continue with removal
            yes.pack()
```

```
            no = Button(pop, text="Dismiss", command=pop.destroy)#Creation of no button to not continue with removing the account

            no.pack()

        showAccountMenu()#Calling of function to remove account menu

        idInput=StringVar()#Declaration of entry fields being string variables

        pwordInput=StringVar()

        nameInput=StringVar()

        privInput=StringVar()

        ToAMenu = Button(self, width=5, fg="red", text="Back",command=RemoveRemove)#Creation of back button for the Remove account screen

        ToAMenu.grid (sticky=W,row=10, column=1)

        RemTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Remove an Account")#Creation of title label for the remove account screen

        RemTitle.grid (sticky=W,column=2,row=1)

        ID= Entry(self, width=27,textvariable=idInput)#Creation of ID entry field

        ID.grid (sticky=W,column=2,row=2)

        idLabel= Label(self, text="ID No.:")#Creation of the label for the ID entry field

        idLabel.grid (sticky=E,column=1, row=2)

        accountMenu= Entry(self, width=27, textvariable=nameInput)#Creation of the Account Name entry field

        accountMenu.grid (sticky=W,column=2,row=3)

        accountLabel= Label(self, text="User Name:")#Creation of the label for the Account Name entry field

        accountLabel.grid (sticky=E,column=1, row=3)

        privileges= Entry(self, width=27, textvariable=privInput)#Creation of the Privileges entry field

        privileges.grid (sticky=W,column=2,row=4)

        privilegesLabel= Label(self, text="Privileges:")#Creation of the label for the Privilleges entry field

        privilegesLabel.grid (sticky=E,column=1, row=4)

        pword= Entry(self, width=27,textvariable=pwordInput)#Creation of the password entry field

        pword.grid (sticky=W,column=2,row=5)
```

```
        pwordLabel= Label(self, text="Password:")#Creation of a label for the Password Entry
field

        pwordLabel.grid (sticky=E,column=1, row=5)

        Remove= Button(self, width=20, bg="light blue", text="Remove Account", font="calibri",
command=warning)#Creation of the Remove account screens remove account button

        Remove.grid (sticky=W,column=2,row=6)

        Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")#Creation of a
label to show part of the instrctions needed to operate the page

        Inst1.grid (row=2, column=3)

        Inst2= Label(self, width=40, text="Removal can be done with just an ID Number",
font="calibri")#Creation of a label to show part of the instrctions needed to operate the page

        Inst2.grid (row=3, column=3)

#End of Remove Account

#Start of Privileges Edit Screen

    def PrivEdit():#Function to create the Privileges edit screen

        def RemovePrivEdit():#Function to call removal of all of this screens widgets

            ToAMenu.grid_forget()#Functions to remove all the widgets

            ID.grid_forget()

            idLabel.grid_forget()

            privileges.grid_forget()

            privilegesLabel.grid_forget()

            Edit.grid_forget()

            Inst1.grid_forget()

            Inst2.grid_forget()

            PrivTitle.grid_forget()

            showAccountMenu()#Calling of function to remove all of the account menus widget
incase of previous error in doing so

            makeAccountMenu()#Calling of function to create Account Menu

        def getPriv():#Function to be called to retrieve privileges from the database

            ID=idInput.get()#Retrival of Id and Privileges from the entry fields

            PR=PrivInput.get()

            print (PR)
```

```
            conn = sqlite3.connect('signin.db')#SQL commands to update the database with the given information

            output = conn.execute("UPDATE SignIn WHERE No='%' VALUES ('%s') ",(%ID,%PR)

        showAccountMenu()#Calling of function to remove all of Account menus widgets

        idInput=StringVar()#Setting of all entry fields to be string variables

        privInput=StringVar()

        ToAMenu = Button(self, width=5, fg="red", text="Back", command=RemovePrivEdit)#Creation of a back button to return to the Account Menu

        ToAMenu.grid (sticky=W,row=10, column=1)

        PrivTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change Privellages")#Creation of Label to title the Privileges edit page

        PrivTitle.grid (sticky=W,column=2,row=1)

        ID= Entry(self, width=27,textvariable=idInput)#Creation of the entry field for the users ID

        ID.grid (sticky=W,column=2,row=2)

        idLabel= Label(self, text="ID No.:")#Creation of the label for the entry field

        idLabel.grid (sticky=E,column=1, row=2)

        privileges= Entry(self, width=27, textvariable=privInput)#Creation of the Privileges entry field

        privileges.grid (sticky=W,column=2,row=3)

        privilegesLabel= Label(self, text="Privileges:")#Creation of the label for the Privileges entry field

        privilegesLabel.grid (sticky=E,column=1, row=3)

        Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri", command=getPriv)#Creation of the edit button widget

        Edit.grid (sticky=W,column=2,row=4)

        Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")#Creation of one of the instruction labels to aid with the use of this screen

        Inst1.grid (sticky=W,row=2, column=3)

        Inst2= Label(self, width=40, text="Editing needs all details entered correctly", font="calibri")#Creation of one of the other instruction label to aid with the use of this screen

        Inst2.grid (sticky=W,row=3, column=3)

#End of Privileges Edit Screen
```

#Start of Password edit screen

```
    def PassEdit():#Function to create the Password edit screen
```

```
        def RemovePassEdit():#Function to call all functions to remove the password edit screen
            ToAMenu.grid_forget()#Functions to remove all of this screens widgets
            ID.grid_forget()
            idLabel.grid_forget()
            pword.grid_forget()
            pwordLabel.grid_forget()
            Edit.grid_forget()
            Inst1.grid_forget()
            Inst2.grid_forget()
            PassTitle.grid_forget()
            showAccountMenu()#Back up function to remove any widgets that weren't removed in
the last transition
            makeAccountMenu()#Function to call for the creation of the Account Menu
        def getPass():#function to add password to database
            ID=idInput.get()#retrieval of information added to the fields
            PW=pwordInput.get()
            print (PW)
            conn = sqlite3.connect('signin.db')#SQL commands to add the information to the
database
            output = conn.execute("UPDATE SignIn WHERE No='%' VALUES ('%s') ",(%ID,%PW)


        showAccountMenu()#Function to remove the Account Menus widgets
        idInput=StringVar()#Declares entry fields to be string variables
        pwordInput=StringVar()
        ToAMenu = Button(self, width=5, fg="red", text="Back",
command=RemovePassEdit)#Creation of back to Account Menu button
        ToAMenu.grid (sticky=W,row=10, column=1)
        PassTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change
Password")#Label for the title of the Password edit screen
        PassTitle.grid (sticky=W,column=2,row=1)
        ID= Entry(self, width=27,textvariable=idInput)#Creation of the ID entry field
        ID.grid (sticky=W,column=2,row=2)
```

```
            idLabel= Label(self, text="ID No.:")#Creation of the label for the ID entry field
            idLabel.grid (sticky=E,column=1, row=2)
            pword= Entry(self, width=27,textvariable=pwordInput)#Creation of the password entry
field
            pword.grid (sticky=W,column=2,row=3)
            pwordLabel= Label(self, text="Password:")#Creation of the label for the Password entry
field
            pwordLabel.grid (sticky=E,column=1, row=3)
            Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri",
command=getPass)#Creation of the edit button for the Password Edit screen
            Edit.grid (sticky=W,column=2,row=4)
            Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")#Creation of an
instruction label to help with use of this screen
            Inst1.grid (sticky=W,row=2, column=3)
            Inst2= Label(self, width=40, text="Editing needs all details entered correctly",
font="calibri")#Creation of another instruction label to help with use of this screen
            Inst2.grid (sticky=W,row=3, column=3)
#End of the Password Edit Screen
```

#Name Edit Screen Start

```
        def NameEdit():#Creation of the User Name Edit screen
            def RemoveNameEdit():#Function to remove the User Name Edit Screen
                ToAMenu.grid_forget()#Functions to remove the widgets of this screen
                ID.grid_forget()
                idLabel.grid_forget()
                account.grid_forget()
                accountLabel.grid_forget()
                Edit.grid_forget()
                Inst1.grid_forget()
                Inst2.grid_forget()
                NameTitle.grid_forget()
                showAccountMenu()#Back up removal of Account Screens widgets in case any remain
                makeAccountMenu()#Calling of function to create the Account Menu Screen
            def getName():#Function to change the user name in the database
```

```
            ID=idInput.get()#Functions to retrieve the information from the entry fields
            NM=nameInput.get()
            print (NM)
            conn = sqlite3.connect('signin.db')#SQL commands to alter the information within the
database
            output = conn.execute("UPDATE SignIn WHERE No='%' VALUES ('%s') ",(%ID,%NM)
        showAccountMenu()#Calling of function to remove account Menus widgets
        idInput=StringVar()#Declaration of entry fields being string variables
        nameInput=StringVar()
        ToAMenu = Button(self, width=5, fg="red", text="Back",
command=RemoveNameEdit)#Creation of back button to return to the Account Menu
        ToAMenu.grid (sticky=W,row=10, column=1)
        NameTitle= Label(self, pady=20, font="calibri, 20", width=27, text="Change
Name")#Creation of the User Name Edit screens Title Label
        NameTitle.grid (sticky=W,column=2,row=1)
        ID= Entry(self, width=27,textvariable=idInput)#Creation of the Id Entry field
        ID.grid (sticky=W,column=2,row=2)
        idLabel= Label(self, text="ID No.:")#Creation of the Id entry's label
        idLabel.grid (sticky=E,column=1, row=2)
        account= Entry(self, width=27, textvariable=nameInput)#Creation of the User Names
entry field
        account.grid (sticky=W,column=2,row=3)
        accountLabel= Label(self, text="Account Name:")#Creation of the label for the user entry
field
        accountLabel.grid (sticky=E,column=1, row=3)
        Edit= Button(self, width=20, bg="light blue", text="Edit Account", font="calibri",
command=getName)#Creation of the Edit button for the User Name Edit screen
        Edit.grid (sticky=W,column=2,row=4)
        Inst1= Label(self, width=40, text="ID Number is needed", font="calibri")#Loading of one
of two of the instructions to aid with the use of this screen
        Inst1.grid (sticky=W,row=2, column=3)
        Inst2= Label(self, width=40, text="Editing needs all details entered correctly",
font="calibri")#Loading of one of two of the instructions to aid with the use of this screen
        Inst2.grid (sticky=W,row=3, column=3)
```

```
#End of User Name Edit Screen
```

#Continuation of Account Menu Screen

```
        def toMainMenu():#Function to go to the main menu
            showAccountMenu()#Calling of function to remove the Account Menu
            mainMenu()#Calling of the function to create the main menus widgets


        def getRemove():#Function to decide whether certain widgets can be loaded
            conn = sqlite3.connect('signin.db')#SQL commands to check whether the privileges of the
user
            cursor= conn.cursor()
            cursor.execute("SELECT PRIVELLAGES from SignIn WHERE Name='%s'" %disUN)
            row = cursor.fetchone()
            for cursor in row:
                Priv=row[0]
            if Priv=="Admin":#if admin certain buttons are loaded
                ButtonRemove()
            else:#if not admin these buttons are not loaded
                return
        def ButtonRemove():#Function to be called if admin to generate "Special" button widgets
            global RemoveAMenu#Declaration of this button being global
            RemoveAMenu= Button(self, font="calibri", width=20, bg="light blue", text="Remove
Account",command=RemoveAccount)#Creation of the go to Remove account screen button
            RemoveAMenu.grid (column=2,row=6)
            global privilegesAMenu#Declaration of this button being global
            privilegesAMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit
Privileges",command=PrivEdit)#Creation of the go to Change Privileges Screen button
            privilegesAMenu.grid (column=2,row=4)
        def RemoveRemoveButton():#A function to remove the "Special Buttons"
            RemoveAMenu.grid_forget()
            privilegesAMenu.grid_forget()
                                    def showAccountMenu():#Function that will remove the account
menus widgets
```

```
                                        ToAMenu.grid_forget()#Functions to remove the widgets
                                        TitleA.grid_forget()
                                        accountAMenu.grid_forget()
                                        privilegesAMenu.grid_forget()
                                        pwordAMenu.grid_forget()
                                        conn = sqlite3.connect('signin.db')#Following lines are to
remove certain widgets that will only appear if Admin
                                        cursor= conn.cursor()
                                        cursor.execute("SELECT PRIVELLAGES from SignIn WHERE
Name='%s'" %disUN)
                                        row = cursor.fetchone()
                                        for cursor in row:
                                                Priv=row[0]
                                        if Priv=="Admin":
                                                RemoveRemoveButton()
                                        else:
                                                return
                                getRemove()
        ToAMenu= Button(self, width=5, fg="red", text="Back",command=toMainMenu)#Creation
of back to Main Menu button
        ToAMenu.grid (sticky=W,row=10, column=1)
        TitleA= Label(self, pady=20, font="calibri, 20", width=27, text="Account Editing
Menu")#Creation this screens Title Label
        TitleA.grid (column=2,row=1)
        accountAMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit User
Name", command=NameEdit)#Creation of the to edit user name screen button
        accountAMenu.grid (column=2,row=2)
        pwordAMenu= Button(self, font="calibri", width=20, bg="light blue", text="Edit
Password",command=PassEdit)#Creation of the to Password Edit Button
        pwordAMenu.grid (column=2,row=5)
#End of Account Menu Screen
```

#Start of Edit Item Screen

```
    def makeEdit():#Creation of Edit Item Screen Button
```

```
        List.grid_forget()#Back up functions for the removal of the Main Menu in case the remove menu function fails
        Add.grid_forget()
        Account.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        Menu.grid_forget()
        def removeAdd():#Function to remove the Edit item screen
            BackAdd.grid_forget()#Functions to remove the Edit item screens widgets
            item.grid_forget()
            itemLabel.grid_forget()
            itemNo.grid_forget()
            itemNoLabel.grid_forget()
            price.grid_forget()
            priceLabel.grid_forget()
            description.grid_forget()
            descriptionLabel.grid_forget()
            Restock.grid_forget()
            Addbtn.grid_forget()
            Remove.grid_forget()
            showMenu()#Function to go to loading all of the main menus buttons
        def notify():#Function to send a notification to the ASC Email
            server = smtplib.SMTP('smtp.gmail.com', 587)#Connection to server information
            server.starttls()
            server.login("AutomaticStockChecker@gmail.com", "ASC12345678")
            getName()#Calling of function to get the information
            msg = "Notification From A.S.C", item, "Needs to be restocked as it is running low or is out of stock"#Message to be included in  notification
            server.sendmail("AutomaticStockChecker@gmail.com", "AutomaticStockChecker@gmail.com", msg)#Command to send the email
            server.quit()#End of connection to email server
            print ("Email sent")#Text to be displayed in debugger
```

```
        def addItem():#Function to Edit Item in the database
            Name=NameInput.get()#Functions to get information from the entry fields
            No=NoInput.get()
            Price=PriceInput.get()
            Desc=DescInput.get()
            print (Name)#Information to be entered into debugger
            print (No)
            print (Price)
            print (Desc)
            conn = sqlite3.connect('ASC.db')#SQL commands to edit item in the database
            conn.execute("SELECT*from Stock")
            conn.execute ("UPDATE Stock WHERE ItemNo='%' VALUES ('%s','%s','%s') ",
(No,Name,Price,Desc))
            row = conn.fetchall()
        def getName():#Function to get the information from the entry fields
            item=NameInput.get()#retrieving information and strong it in a variable
            print (NameInput.get())
            return item#Returning the declared variable
        NameInput=StringVar()#Setting of the entry fields to be string variables
        NoInput=StringVar()
        PriceInput=StringVar()
        DescInput=StringVar()
        BackAdd= Button(self, width=5, fg="red", text="Back", command=removeAdd)#Creation of
back button widget to return to the main menu
        BackAdd.grid (sticky=W,row=10, column=1)
        item= Entry(self, width=27,textvariable=NameInput#Creation of items name entry field
        item.grid (sticky=W,column=2,row=1)

        itemLabel= Label(self, text="Item Name:")#Creation of the label for the items name entry
field
        itemLabel.grid (sticky=E,column=1, row=1)
        itemNo= Entry(self, width=27,textvariable=NoInput)#Creation of the Item No entry field
```

```
        itemNo.grid (sticky=W,column=2,row=2)
        itemNoLabel= Label(self, text="Item No:")#Creation of the label for the Item No entry field
        itemNoLabel.grid (sticky=E,column=1, row=2)
        price= Entry(self, width=27,textvariable=PriceInput)#Creation of the Items price entry field
        price.grid (sticky=W,column=2,row=3)
        priceLabel= Label(self, text="Price code:")#Creation of the label for the price entry field
        priceLabel.grid (sticky=E,column=1, row=3)
        description= Entry(self, width=27,textvariable=DescInput)#Creation of the description entry
field
        description.grid (sticky=W,column=2,row=4)
        descriptionLabel= Label(self, text="Description:")#Creation of a label for the description
entry field
        descriptionLabel.grid (sticky=E,column=1, row=4)
        Restock= Button(self, width=20, bg="light blue", text="Restock", font="calibri",
command=notify)#Creation of the restock button
        Restock.grid (sticky=E,column=2,row=6)
        Addbtn= Button(self, width=20, bg="light blue", text="Add Item", font="calibri",
command=addItem)#Creation of the add item button
        Addbtn.grid (sticky=E,column=2,row=7)
        Remove= Button(self, width=20, bg="light blue", text="Remove Item",
font="calibri")#Creation of the Remove Item  button
        Remove.grid (sticky=E,column=2,row=8)
#Edit Item Screen End
#Start List Screen
    def makeList():#Creation of the List screen
        List.grid_forget()#Loading of all of the screens widgets
        Add.grid_forget()
        Account.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        ExitMenu.grid_forget()
        Menu.grid_forget()
```

#Start sub-Edit screen

```
        def makeSubEdit():#Creation of the subEdit Screen
            def removeEdit():#Function to remove all of the subEdit screens widgets
                BackEdit.grid_forget()#Functions to remove the selected widgets
                SubTitle.grid_forget()
                item.grid_forget()
                itemLabel.grid_forget()
                itemNo.grid_forget()
                itemNoLabel.grid_forget()
                price.grid_forget()
                priceLabel.grid_forget()
                description.grid_forget()
                descriptionLabel.grid_forget()
                Restock.grid_forget()
                Addbtn.grid_forget()
                Remove.grid_forget()
                makeList()#Function to load the List screen
            def notify():#Function to send a notification to the ASC Email
                server = smtplib.SMTP('smtp.gmail.com', 587)#Connection to server information
                server.starttls()
                server.login("AutomaticStockChecker@gmail.com", "ASC12345678")
                getName()#Calling of function to get the information
                msg = "Notification From A.S.C", item, "Needs to be restocked as it is running low or is out fo stock"#Message to be included in  notification
                server.sendmail("AutomaticStockChecker@gmail.com", "AutomaticStockChecker@gmail.com", msg)#Command to send the email
                server.quit()#End of connection to email server
                print ("Email sent")#Text to be displayed in debugger
            def addItem():#Funtion to edit item
                getName()#Calling of Functions to retrieve details
                getNo()
                getDesc()
```

```
            getPrice()
            conn = sqlite3.connect('ASC.db')#SQL commands to insert the new information
            conn.execute("UPDATE Stock WHERE No='%s' VALUES ('%s','%s','%s','%s')"
(No,Name,No,Prices,Desc))
        def Remove():#function to Remove Item
            getName()#Calling of Functions to retrieve details
            getNo()
            getDesc()
            getPrice()
            conn = sqlite3.connect('ASC.db')#SQL commands to insert the new information
            conn.execute("REMOVE FROM Stock WHERE ID='%s'" %ID)
        def getName():#Function to retrieve information from the name entry field
            entry=NameInput.get()
            if entry==None:
                Name="No Such Item"#If blank then function No such item prints
                return Name
            else:
                Name=entry
                print (Name)
                return Name
        def getNo():#Function to retrieve information from the No entry field
            entry=NoInput.get()
            if entry==None:
                No="No Such Item"#If blank then function No such item prints
                return No
            else:
                No=entry
                print (No)
                return No
        def getPrice():#Function to retrieve information from the Price entry field
            entry=PriceInput.get()
```

```
            if entry==None:
                Price="No Such Item"#If blank then function No such item prints
                return Price
            else:
                Price=entry
                print (Price)
                return Price
        def getDesc():#Function to retrieve information from the Description entry field
            entry=DescInput.get()
            if entry==None:
                Desc="No Such Item"#If blank then function No such item prints
                return Desc
            else:
                Desc=entry
                print (Desc)
                return Desc
        NameInput=StringVar()#declaration of inputs being string variables
        NoInput=StringVar()
        PriceInput=StringVar()
        DescInput=StringVar()
        BackEdit= Button(self, width=5, fg="red", text="Back", command=removeEdit#Creation
of back button widget
        BackEdit.grid (sticky=W,row=10, column=1)
        SubTitle=Label (self, text="Edit An Item", font="calibri, 16")#Creation of title label for the
page
        SubTitle.grid (column=2,row=1)
        item= Entry(self, width=27,textvariable=NameInput)#Creation of item name entry
        item.grid (sticky=W,column=2,row=2)
        itemLabel= Label(self, text="Item Name:")#Creation of label for item name entry
        itemLabel.grid (sticky=E,column=1, row=2)
        itemNo= Entry(self, width=27,textvariable=NoInput)#Creation of item no entry field
        itemNo.grid (sticky=W,column=2,row=3)
```

```
        itemNoLabel= Label(self, text="Item No:")#Creation of a label for the item no entry field
        itemNoLabel.grid (sticky=E,column=1, row=3)
        price= Entry(self, width=27,textvariable=PriceInput)#Creation of item price entry field
        price.grid (sticky=W,column=2,row=4)
        priceLabel= Label(self, text="Price code:")#Creation of a label for the item price entry
field
        priceLabel.grid (sticky=E,column=1, row=4)
        description= Entry(self, width=27,textvariable=DescInput)#Creation of a description entry
field
        description.grid (sticky=W,column=2,row=5)
        descriptionLabel= Label(self, text="Description:")#Creation of a label for the description
entry field
        descriptionLabel.grid (sticky=E,column=1, row=5)
        Restock= Button(self, width=20, bg="light blue", text="Restock", font="calibri",
command=notify)#Creation of a button to run the "Notify" function
        Restock.grid (sticky=E,column=2,row=6)
        Addbtn= Button(self, width=20, bg="light blue", text="Save Changes", font="calibri",
command=addItem)#Creation of a button to edit the item
        Addbtn.grid (sticky=E,column=2,row=7)
        Remove= Button(self, width=20, bg="light blue", text="Remove Item", font="calibri",
command=Remove)#Creation of a button to remove the item
        Remove.grid (sticky=E,column=2,row=8)
#End of the subEdit screen
```

#Continuation of List Screen

```
    def removeList():#Function to remove all of the List Screens widgets
        Back.grid_forget()#Functions to forget the List screens widgets
        search.grid_forget()
        searchLabel.grid_forget()
        Name.grid_forget()
        NO.grid_forget()
        Price.grid_forget()
        Desc.grid_forget()
        RestockL.grid_forget()
```

```
            Edit.grid_forget()
            searchTitle.grid_forget()
            Clear.grid_forget()
            showMenu()#Function to load the main menu
        def notify():#Function to send a notification to the ASC Email
            server = smtplib.SMTP('smtp.gmail.com', 587)#Connection to server information
            server.starttls()
            server.login("AutomaticStockChecker@gmail.com", "ASC12345678")
            entry=searchInput.get()#Calling of function to get the information
            conn = sqlite3.connect('ASC.db')#SQL commands to load gather information form the
database
            cursor= conn.cursor()
            cursor = conn.execute("SELECT*from Stock WHERE ItemName='%s'"%entry)
            if cursor==None:#If statement to not send email if the search is empty
                print("Search is empty")
                return
            else:
                for row in cursor:
                    Name = row[0]
                msg = "Notification From A.S.C", Name, "Needs to be restocked as it is running low or is
out of stock"#Message to be included in  notification
                server.sendmail("AutomaticStockChecker@gmail.com",
"AutomaticStockChecker@gmail.com", msg)#Command to send the email
                server.quit()#End of connection to email server
                print ("Email sent")#Text to be displayed in debugger
        def getSearch():#Funtion to retrieve the search information
            def clear():#Function to clear the search
                aRestock.grid_forget()#Functions to forget parts of the search
                aEdit.grid_forget()
                aName.grid_forget()
                aNO.grid_forget()
                aPrice.grid_forget()
```

```
            aDesc.grid_forget()
            aClear.grid_forget()
        def toSubEdit():#Function to remove the list screens widgets and load the subEdit screen
            Back.grid_forget()#functions to forget all of sub edit screens widgets
            searchTitle.grid_forget()
            search.grid_forget()
            searchLabel.grid_forget()
            Name.grid_forget()
            NO.grid_forget()
            Price.grid_forget()
            Desc.grid_forget()
            Clear.grid_forget()
            RestockL.grid_forget()
            Edit.grid_forget()
            clear()
            searchTitle.grid_forget()
            makeSubEdit()#Function to load subEdit's widgets
        def getName():#Function to retrieve the name from the database
            entry=searchInput.get()#Function to retrieve information from an entry field
            conn = sqlite3.connect('ASC.db')#SQL commands to select information from the
database
            cursor= conn.cursor()
            cursor = conn.execute("SELECT ItemName from Stock WHERE ItemName LIKE
'%s'"%entry)
            if cursor==None:
                Name="No Such Item"#If blank No Such item is returned
                return Name
            else:
                for row in cursor:
                    Name=row[0]
                    return Name
        def getNo():#Function to retrieve the No from the database
```

```
            entry=searchInput.get()#Function to retrieve information from an entry field
            conn = sqlite3.connect('ASC.db')#SQL commands to select information from the
database
            cursor= conn.cursor()
            cursor = conn.execute("SELECT ItemNo from Stock WHERE ItemName LIKE '%s'"%entry)
            if cursor==None:
               No="No Such Item"#If blank No Such item is returned
               return No
            else:
               for row in cursor:
                  No=row[0]
                  return No
         def getPrice():#Function to retrieve the price from the database
            entry=searchInput.get()#Function to retrieve information from an entry field
            conn = sqlite3.connect('ASC.db')#SQL commands to select information from the
database
            cursor= conn.cursor()
            cursor = conn.execute("SELECT Price from Stock WHERE ItemName LIKE '%s'"%entry)
            if cursor==None:
               Price="No Such Item" #If blank No Such item is returned
               return Price
            else:
               for row in cursor:
                  Price=row[0]
                  return Price
         def getDesc():#Function to retrieve the Description from the database
            entry=searchInput.get()#Function to retrieve information from an entry field
            conn = sqlite3.connect('ASC.db')#SQL commands to select information from the
database
            cursor= conn.cursor()
            cursor = conn.execute("SELECT Description from Stock WHERE ItemName
LIKE'%s'"%entry)
```

```
            if cursor==None:
                Desc="No Such Item"#If blank No Such item is returned
                return Desc
            else:
                for row in cursor:
                    Desc=row[0]
                    return Desc
        getNo()#Calling of functions to retrieve the information
        getName()
        getPrice()
        getDesc()
        aName= Label(self,width=15,text=getName())#Creation of label to give information on
the name
        aName.grid(sticky=W,column=2,row=3)
        aNO=Label(self,width=15,text=getNo())#Creation of label to give information on the No
        aNO.grid(sticky=W,column=3,row=3)
        aPrice=Label(self,width=15,text=getPrice())#Creation of label to give information on the
Price
        aPrice.grid(sticky=W,column=4,row=3)
        aDesc=Label(self,width=30,text=getDesc())#Creation of label to give information on the
Description
        aDesc.grid(sticky=W,column=5,row=3)


        aRestock=Button(self,width=14,text="Restock", command=notify)#Creation of the button
to run the restock notification function
        aRestock.grid(sticky=W,column=6,row=3)
        aEdit=Button(self,width=14,text="Edit", command=toSubEdit)#Creation of a button to
make the sub edit screen
        aEdit.grid(sticky=W,column=7,row=3)
        aClear=Button(self,width=14,text="Clear Search", command=clear#Creation of a button
to clear the search
        aClear.grid(sticky=W,column=8,row=3)
```

Back = Button(self, width=5, fg="red", text="Back", command=removeList)#Creation of a button to return to the list screen

Back.grid (sticky=W,row=100, column=0)

searchInput=StringVar()#Declaration of an input as a string variable

searchTitle=Label (self, text="Item Search", font="calibri, 16")#Creation of a label to act as a title for the page

searchTitle.grid (column=1,row=1)

search=Entry(self,textvariable=searchInput)#Creation of an entry field to act as a search bar

search.grid (sticky=W,column=1,row=2)

searchLabel=Button(self, text="Search:",command=getSearch)#creation of a label to act as a title for the search bar

searchLabel.grid (sticky=W,column=0, row=2)

Name= Label(self,width=15,text="ItemName",bg="grey")#creation of a label for the name

Name.grid(sticky=W,column=2,row=2)

NO=Label(self,width=15,text="ItemNo.",bg="grey")#creation of a label for the No

NO.grid(sticky=W,column=3,row=2)

Price=Label(self,width=15,text="ItemPrice",bg="grey")#creation of a label for the Price

Price.grid(sticky=W,column=4,row=2)

Desc=Label(self,width=30,text="Description",bg="grey")#creation of a label for the description

Desc.grid(sticky=W,column=5,row=2)

RestockL=Label(self,width=15,text="Restock",bg="grey")#A label that reads Restock

RestockL.grid(sticky=W,column=6,row=2)

Edit=Label(self,width=15,text="Edit",bg="grey")#A label for the edit heading

Edit.grid(sticky=W,column=7,row=2)

Clear=Label(self,width=15,text="Clear",bg="grey")#A label that reads clear as a table heading

Clear.grid(sticky=W,column=8,row=2)

#End of List screen

#Continuation of Main Menu

def getID():# funtion to retrieve the current users ID

conn = sqlite3.connect('signin.db')#SQL command to retrieve the users ID

```
        output = conn.execute("SELECT ID from SignIn WHERE Name='%s'" %disUN)
        for row in output:
            ID=row[0]
        return ID
    ExitASC.grid_forget()#creation of a button to exit the program
    ExitMenu = Button(self, width=5, fg="red", text="Exit")
    ExitMenu["command"] =  self.quit
    ExitMenu.grid (sticky=W,row=10, column=1)
    Menu = Label(self,text="Menu", pady=20, font="calibri, 20")#Label that reads menu and acts as a title for this screen
    Menu.grid (row=1, column=2)
    List = Button(self, width=20, bg="light blue", text="Search Item", font="calibri", command=makeList)#A button to take the user to list screen
    List.grid (row=2, column=2)
    Add = Button(self, width=20, bg="light blue", text="Add Item", font="calibri", command=makeAdd)#A button to take the user to the Add Item screen
    Add.grid (row=3, column=2)
    Account = Button(self, width=20, bg="light blue", text="Account Settings", font="calibri", command=makeAccountMenu)#A button to take the user to the Account menu screen
    Account.grid (row=4, column=2)
    ID1=Label (self, text="Your ID number is:")#A label to display Id information
    ID1.grid (row=11, column=3, sticky=E)
    ID2=Label (self, text=getID())#A label to display the users ID
    ID2.grid (row=11, column=4, sticky=W)
    UN1=Label (self, text="Your UserName is:")#A label to display the user name
    UN1.grid (row=11, column=1, sticky=E)
    UN2=Label (self, text=disUN)#A label to display the users user name
    UN2.grid (row=11, column=2, sticky=W)
    title1=Label(self, text="Bentley and Skinner", font="calibri, 30")#A label to display the companies name
    title1.grid (row=6, column=2)
    title2=Label(self, text="A.S.C", font="calibri, 30")#a label to display the systems name
    title2.grid (row=7, column=2)
```

```
#End of Main Menu screen
#Start of Sign In screen
    def removeSign():#Function to remove the sign in screen
        SignLabel.grid_forget()#functions to forget the sign in screens widgets
        SignIn.grid_forget()
        UserLabel.grid_forget()
        PwordLabel.grid_forget()
        User.grid_forget()
        Pword.grid_forget()
        title1.grid_forget()
        title2.grid_forget()
        mainMenu()
    def findUN():#function to check the validity of the user name
        conn = sqlite3.connect('signin.db')
        UN=userInput.get()#the collection of the username from the entry fields
        output = conn.execute("SELECT*from SignIn WHERE Name='%s'" %UN)#SQL command to
retrieve a user name similar to the one entered
        for row in output:
            Name=row[1]
            if UN==Name:
                global disUN#Declaration of a global variable for the main menus display of information
                disUN=Name
                findPW()#the given username is equal to one in the database and therefore the program
moves on to the password validation
                conn.close()
                return disUN#Declaration of a global variable for the main menus display of information
            else:#If username isn't equal to one in the database the function ends and entry is not
granted
                conn.close()
                return
        conn.close()
    def findPW():#function to validate the entered password
```

```
        conn = sqlite3.connect('signin.db')
        PW=pwordInput.get()#collection of the password from the entry fields
        output = conn.execute("SELECT*from SignIn WHERE Password='%s'" %PW)#SQL command to retrieve a password similar to the one entered
        for row in output:
            Pass=row[2]
            if PW==Pass:
                removeSign()#the given password is equal to the one in the database and therefore the program moves on to the main menu screen
                conn.close()
                return PW
            else:#If password is not equal then the function ends and entry is not granted
                conn.close()
                return
        conn.close()
    global ExitASC#variable declared as gloabal for easy removal
    ExitASC = Button(self, width=5, fg="red", text="Exit")#Creation of an exit button to exit from the program
    ExitASC["command"] =  self.quit
    ExitASC.grid (sticky=W,row=10, column=1)
    SignLabel= Label(self, text="Sign In", font="calibri, 16", pady="20")#creation of a label for the sign in screen
    SignLabel.grid (row=1, column=2)
    UserLabel = Label(self, text="Account Name:", font="calibri")#creation of a label to partner with the username field
    UserLabel.grid (sticky=W,row=2,column=2)
    PwordLabel = Label(self, text="Password:", font="calibri")#creation of a label rt partner with password entry field
    PwordLabel.grid (sticky=W,row=3,column=2)
    userInput=StringVar()#Declaration of a variable having a string type
    pwordInput=StringVar()
    User = Entry(self,textvariable=userInput)#an entry field for the username
    User.grid (row=2, column=2)
```

```
        Pword= Entry(self,textvariable=pwordInput)#an entry field for the password
        Pword.grid (row=3, column=2)
        SignIn=Button(self, text="Sign In",  bg="light blue", font="calibri", command=findUN)# a button to begin the sign in process
        SignIn.grid (row=4, column=2)
        title1=Label(self, text="Bentley and Skinner", font="calibri, 30")#a label to act as a title displaying the compnay name
        title1.grid (row=6, column=2)
        title2=Label(self, text="A.S.C", font="calibri, 30")#a label to display the programs name
        title2.grid (row=7, column=2)
        space=Label(self,width=3)# a blank label to fill in space for the program
        space.grid (row=6, column=3)
#End of the sign in screen
    def __init__(self, master=None):#Creation of the frame in which the program occurs
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()
root = Tk()
root.iconbitmap(r'Icon\favicon.ico')#Creation the icon of the program
app = ASC(master=root)
root.title("A.S.C")#Name of program given
root.attributes("-fullscreen",True)#Program made full screen
app.mainloop()
root.destroy()
```

END OF CODE

Evaluation

Final Product Description

In order to solve the listed problems the final program will need to be able to run off desktops as the information can be run locally through the network to the PC's. The main data base however will be stored on a Raspberry Pi connected to the network and accessed via the user program.

Once signed in the users are then taken to a menu screen which is the main hub of the system as it allows the user to navigate to all the main screens. The list view screen shows all of the stock that has been attributed to the data base next to these should be the information that would be required the quickest, these being the price and amount of the item in stock. The price would be encoded with the shops own system. At the top of this list view is a search bar that will allow the user to find a specific item of stock a lot quicker. In order for this to be fully efficient the search bar is locked in position and does not scroll with the rest of the list. If an item runs low the font colour will change in the stock column and an automatic alert is sent via email. When a user enters that stock items information edit screen, IES, the user is able to order more of that item via the restock button.

The information used in the program is stored in a separate data base which only administrators have access to. This is as different users are only authorised to access different bits of information depending on who they are. This is known as compartmentalisation and is another way to make the system more secure as only two types of users have access to the more in depth information, these being the shop managers and the administrators.

Requirements of the system

As has been already seen the 8 things listed below are the requirements for the ASC system for it to fulfil its task of solving the management problem of the Business. In the following evaluation I shall refer back to these points in order to prove that the system is a success.

1.Easily search through all items in stock	2.Easily edit any items details	3.All information is secure and can only be accessed by authorised personal	4.Automatic alert if stock reaches zero
5.Account details can be changed	6.New stock can be inputted by appropriate users	7.The screens can be navigated easily	8.The restock button orders more stock from the retailer

Cost

The cost of the final program came to a total of £0 as there was no software or hardware that was needed that hadn't already been bought or was free for use. For example the large amount of code used to run this system was written in python IDLE. This software can be downloaded straight from the python website and is open source meaning that no licence fee has to be paid unless a large amount of money is made from the ASC program. Therefore no charge came from this as the finished ASC product is free and no monetary value is gained from its creation. The full implementation of the code was written on computers provided by the school or myself and therefore no extra hardware for myself had to be purchased. The computers needed to run the ASC

program already exist in the work space of the business. Also the PI computer where the database is stored was provided by myself to the company so there was no extra charge. Therefore no extra hardware or software has had to be bought for the users or myself bringing the total cost of the project to £0.

Robustness, Usability and Performance

The final implementation of the program is very robust as can be seen throughout the final testing of the product. This is as no errors occurred throughout its final testing and after long periods of it being used no degradation occurred. The program has now been integrated into the company and the full stock list is being added to the database. This means that the system is going to be working to its full extent of 2000 items and 20 users. During this addition of the stock and users to the database the system has been seen to working at its full extent and no reports of it slowing down have been reported. However one problem that has arose is that when 1000s of entities are in the database the search function does seem to become flawed. This is as it asks for you to be very specific with your search criteria. For example just searching the word “Ring” would bring up 100s of entities. Due to this the search criteria must be more specific such as “Ring Silver Emerald” which would select only a couple of entities one of these being the one you were looking for. A possible alternative for this is to have an item type so the word ring doesn’t have to be included in the search bar and instead a type of item would be selected from a drop down menu. However even with the search having to be more specific the first requirement is still met as users can still search for an item.

The usability of the ASC program incorporates a large amount of the system requirements as many of them ask for it to be highly usable. For example requirement 1, 2 and 7 all require the program to accomplish a specific task with ease whether it is navigating, editing an item or searching through multiple items. In terms of navigating between screens I believe that the system achieves this very well as it is very simple to use the on screen widgets to move between the different screens. This is as all buttons have been enlarged and the appropriate text added to the face of them. Also title labels have been added to each screen so that the user always knows where they’re and can navigate themselves back. This is due to in early testing it was very easy to get lost and lose track of where you were as can be seen in the Developing the Solution section. Editing items is also very simplistic as once you have searched for the specific item you want there is a large button displayed allowing you to edit that item. However the business did say that it is a shame that there is no direct way to go to an edit page and that you have to search for the item first. Though in them saying this it would be hard to jump straight to the edit page unless some kind of details about the item were already known. Throughout my testing I tested many different functions against criteria 7 as there are multiple functions whose main purpose is to navigate between pages. A few examples of these are functions such as Go to Main Menu, Go to Edit Item Screen and Go to Add Item Screen. All of the test on functions relating to criteria 7 have been successfully completed and proven to function appropriately. Other criteria’s such as 1 also relate to usability as it states that you must be able to search through all of the items with ease. For the Search function, which this criteria applies to, the function completed this criteria completely. The other criteria that affects usability is criteria 2 being that details if an item can be easily edited which applies to the Save Changes function in the Edit screen. This function also manged to function as intended. Due to the in depth testing I performed and with the success of the functions I am confident in saying that the ASC program is successful in terms of its usability.

Early reports and interviews from the business since the handover has lead me to believe that the Performance of the system is good. This is as the business believes that “the ASC program is an

effective solution". Although this may be the case when items reach zero amounts in stock they are not removed from the data base in case they are put in stock again. Due to this the program may exceed its intended capacity of about 2000 items. When this occurs memory may become limited as more items will have to be stored and accessed and one time. If this occurs then the system may begin to slow down and the performance of the system may drop dramatically. For this reason I am hesitant to say that the ASC meets the performance requirements needed although the chance of the performance decreasing enough to have a large impact is very low. Other than this the only other criticism is the fact of notifications not being automatic although this has been discussed before. This is making the notifications automatic requires the program to be running all the time and checking g to see if any of the entities have had their stock changed to zero. However why this should be done if a user can trigger a restock order when they look at the item. Also what if a user wishes to restock an item before the stock counter is at zero, if this process was automatic this couldn't be done.

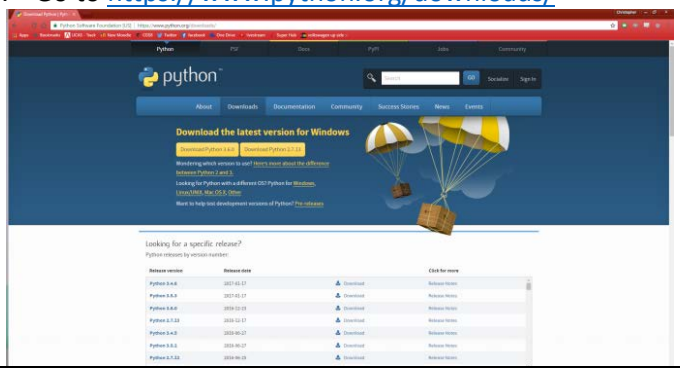
In conclusion I believe that the ASC program that has been created is a success as it meets all given criteria bar one being the notifications being automatic. I believe this is acceptable as the replacement for that gives many more opportunities than if it had been automatic. Thus all that is to be done is for the integration and hand over of the program to the business who are already starting to use this system.

Documentation

Installation Guide

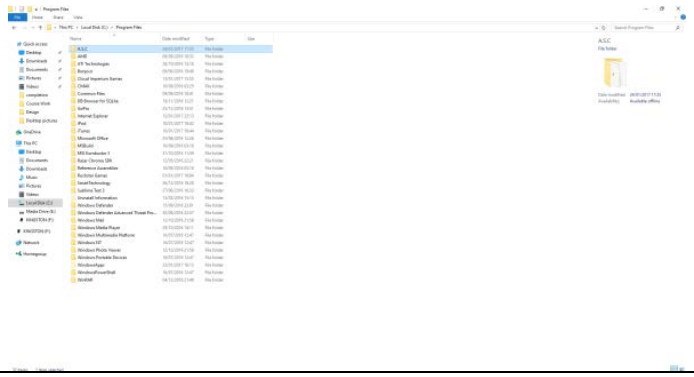
To install the program the following steps need to be followed:

1. Go to https://www.python.org/downloads/

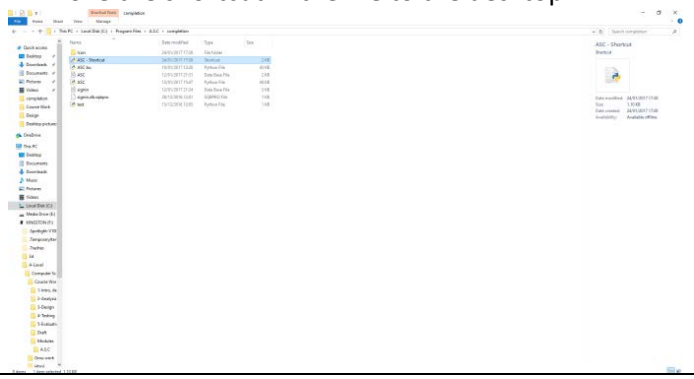


2. Download the latest version of Python

3. Take the ASC folder from the disc



4. Move the shortcut in the file to the desktop



5. Click the shortcut to begin

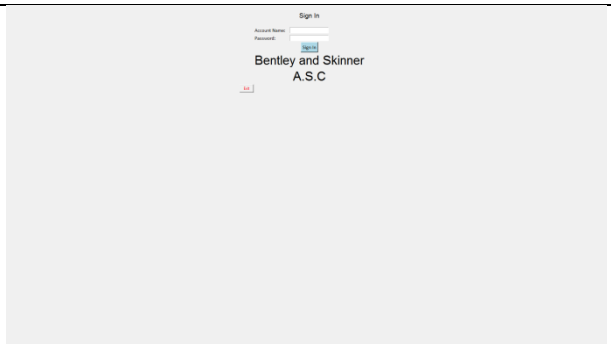
How to Use the ASC Program

The following information displays how a User can use the program to store stock as well as check the databases contents:

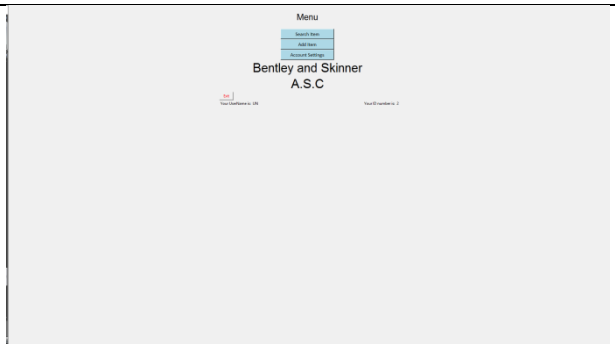
Sign In Screen

Signing In

First of all run the ASC program from your desktop



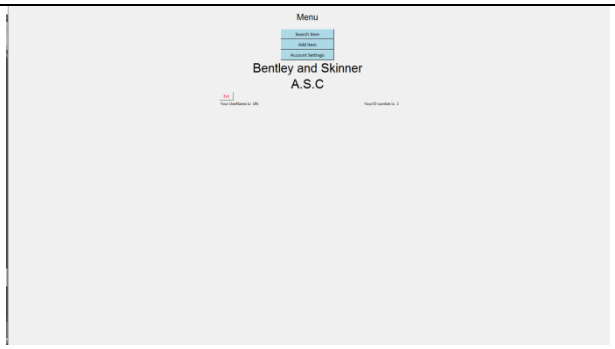
The program should open to the main sign in screen where the appropriate details can be entered these being your assigned Username and Password which should have been given to you previously.



Once the details have been entered click the sign in button can be clicked which should take you to the main menu. From here you can access multiple functions

Search Item Screen

Navigating to



From the main Menu screen press the Search Item Button

This should run the command to take you to the Search Item screen

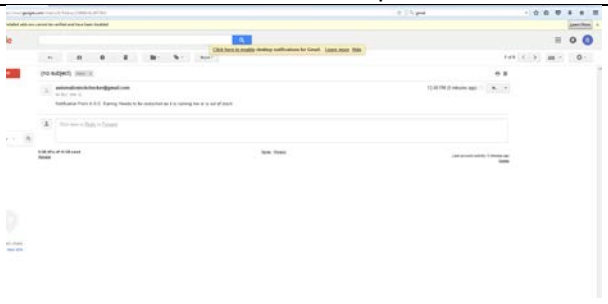
Search for an Item

First of all enter your search criteria into the search bar located at the top left

Then press the search button to generate your search result

Restocking An Item

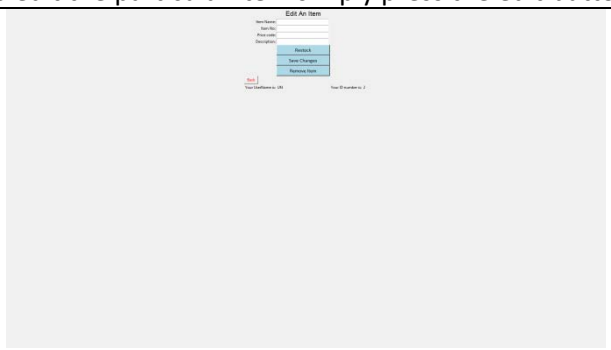
Once an item has been searched for press the restock button



This shall send the notification email to the given address of the system

Editing the Item

To edit the particular item simply press the edit button



This shall take you to the Item Edit Page

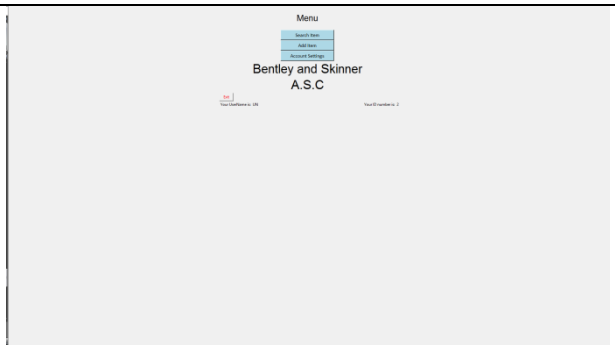
Clearing Search

To clear the search simply press the Clear Search button

This will remove all search entries from the page

Add Item Screen

Navigating to



To navigate to the Add item screen simply press the Add Item button

This will then take you to the Add Item screen

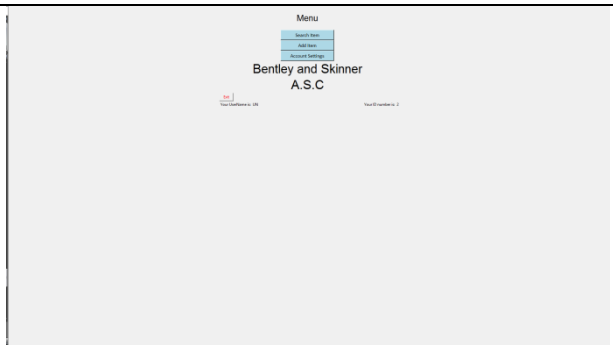
Add Item

First of all fill in all details of the item

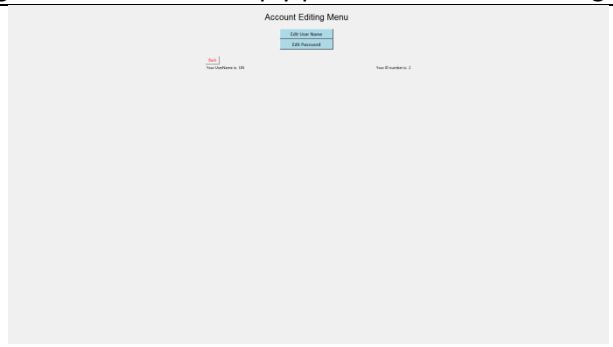
Then press the Add Item Button

Account Menu

Navigating to



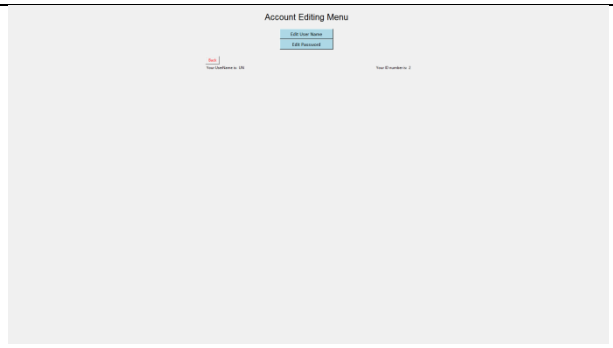
To navigate to the screen simply press the Account Settings button



This shall to take you to a basic version of the menu due to your privileges being set to “User” you have only been given access to two functions

User Name Edit

Navigating to



To navigate to this page simply press the Edit Username Button

This will the take you to the Edit User Name screen

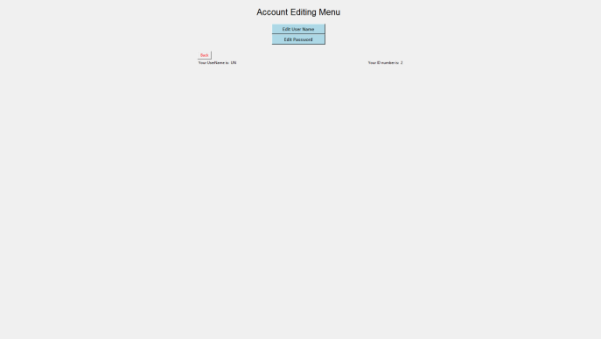
Changing Username

Enter your ID number, which is displayed on the right, and your new account name

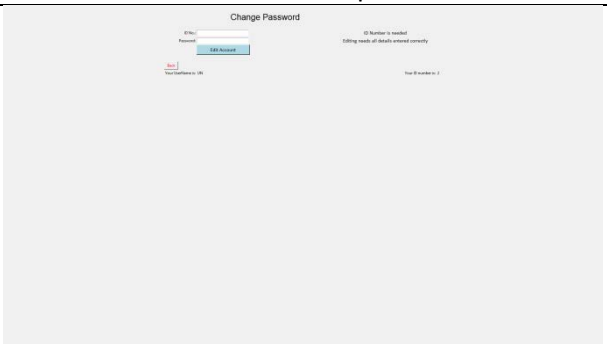
Then press the Edit Account button to change your user name

Edit Password Screen

Navigating to



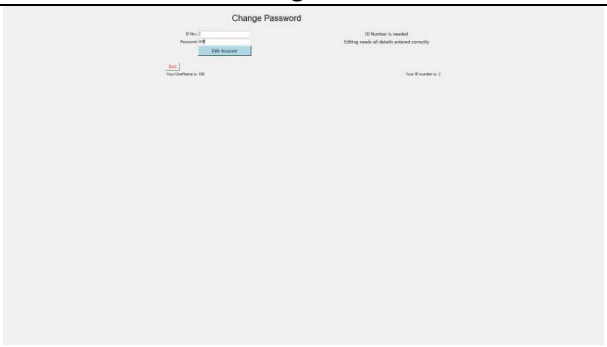
To navigate to the Edit Password screen press the Edit Password Button



The Edit Password screen should then open in the window

Editing Password

To edit your password enter your details into the fields, your id number is located on the bottom right

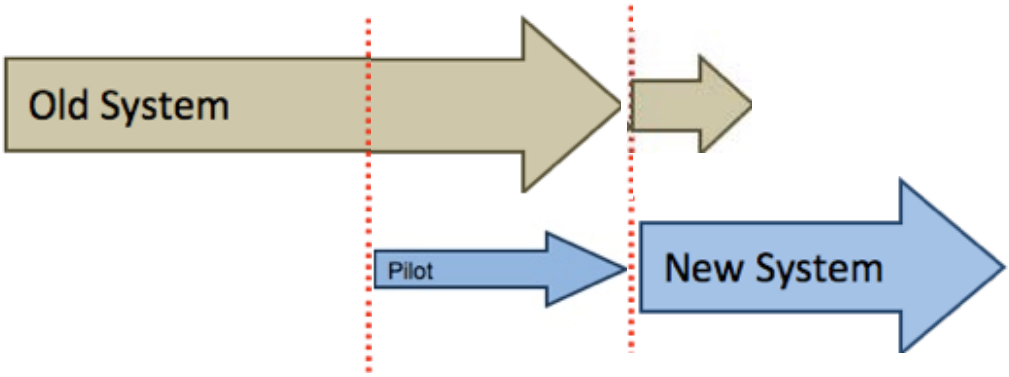


Then press the Edit Account button to save these changes to the database

Handover

Diagram

Parallel-Pilot Change Over



For my project I have decided to use the Direct-Pilot changeover method. This method takes attributes from both the Parallel changeover and Pilot changeover methods. This is as before the changeover takes place the old system is in place. In the second phase a pilot system is implemented to test that it does indeed work with business. This is pilot will act as a test phase of sorts. In the end both systems will work side by side as the old numbering method will still be in use within the new program. This is as well as the old stock storing method will also still be in place. The new system will simply be replacing the cataloguing method of the old system.

Maintenance

Throughout the use of the ASC system some essential maintenance needs to be carried out to ensure it continues to function at its full capacity.

Preventive Maintenance

One of the methods is the removal of unique items that are out of stock. By this I mean if a one of a kind item has been in stock but then sold to a customer it must be removed. This is as over time these items which are not in stock may lead to the program slowing down as they still have to be sorted through

This is similar to the removal of vacant users. This involves the removing of users who no longer work at the store. This is as in the log in process and user editing functions there information still needs to be checked against all of the others.

Corrective Maintenance

Some corrective maintenance is that if the sign in database ever fails that a backup of the database is stored in the ASC files. This database can be dragged and dropped into the main ASC folder to replace the sign in database that was in there. The downside of this is that users will need to renter all of their information.

Perfective

Certain parts of the system can be altered with some small changes in the code. For example the font and colours of buttons can be changed in the attributes of each of the widgets. The location of each of the widgets has been annotated in the full code annotation so that they can easily be found. This maintenance will allow the user to define the program and perfect it to their needs.