**Validation Testing**

I have performed a full suite of validation tests for each major function in the program. This includes input validation testing. Every potential form of invalid input was checked, to determine if invalid input was properly blocked from the program. After I had performed input validation testing, I tested the functionality of each button. Each button has, at least, three separate tests. Due to the exhaustive nature of input validation within my program, each form of invalid input was only tested once.

**New File Validation**

This form of input validation covered the creation of new files with the main file entry field. There were 16 different tests for this. Main file names are invalid if the main file entry field is left empty, if the main file entry field contains a space, and if the following characters are in the main file entry field: / \ ? % \* : | “ < > . , ; =

My program is intended for the Windows operating system, and the above characters are normally reserved for file management purposes in Windows. The correct response of my program to invalid input in the main file entry field is the “Invalid main file name” error message. To see if this message was consistently output, I tested each form of invalid main file name independently. To make sure that the message was consistently output, I reset the output window by using the “Open File” button to open a file that actually exists. That replaced the error message with an inventory list. This ensured that the error message was consistently displayed for each form of invalid main file name. I also consistently checked the terminal window for IDLE, to see if I was avoiding error messages for the program itself. I did this to check if any invalid input for main file names was accidentally slipping through.

By the time that I implemented the main file name input validation, I had already tested the module thoroughly in a separate script. When I was creating the module, my main issue was with testing for the list of invalid characters. I wanted to avoid using a series of elif statements for each individual character, but I was not able to directly check if anything in a list or a tuple was present in the main file entry field through a “if <list or tuple> in <main file entry field>” statement. I used a for loop with a tuple of the invalid characters to get around this. Once I placed the validity check module in my entire program, it functioned properly.

**Copy File Creation**

The module for copy file entry field input validation is identical to the module for main file entry field input validation, so the test suite is the exact same. The main difference is that a valid filename for a file that actually exists was left in the main file entry field for each of the copy file input validation tests, and that the correct form of output in my program is the “Invalid copy file name” message. To create the module for checking the validity of the copy file, I just copied the code for the main file entry field input validation module. I could have just used the same module for both forms of input validation, because both are identical. I just wanted to directly obtain the copy file name from the copy file entry field for each individual input validation test.

The main issue that I faced when testing the module in the final program was that I had forgotten to complete the “Invalid copy file name” error message. There was only one line of output, which was “Invalid copy file name.” I copied the rest of the output from the “Invalid main file name” message and placed it into the “Invalid copy file name” message. I altered the message so that it fit copy file errors.

**File in Directory**

For the functions that are tied to the “Copy File” button, the “Open File” button, the “Add To File” button, and the “Delete From Inventory” button, my program needs to open a file that actually exists. I test for the existence of a file with the filename in the main file entry field, to ensure this. If the file does not exist, the correct program output is the “File not in directory for main file name” message. For the four buttons for which an existing file is relevant, I tested with a valid filename in the main file entry field, but the filename is for a file that does not exist. All of the relevant entry fields for each relevant button were filled with valid values. To ensure that the correct error message was output to the user consistently, I reset the output window by using the “Open File” button for a file that actually exists. I also checked the IDLE terminal window to see if any invalid input was slipping through.

As with the previous validity check modules, I tested the module for this with a separate script before I implemented the module in the entire program. Once I placed the validity check module in my entire program, it functioned properly.

**Inventory Item Entry Fields**

Input validation for the inventory item entry fields was the most demanding. Luckily, all of the inventory item entry fields are only relevant for the “Add To File” button. The “Delete From Inventory” button only requires the SKU entry field. Each inventory item entry field has its own standards for valid input, and any failure from a single inventory entry field results in the “Invalid input” error message. Each form of invalid input for each inventory item entry field was tested while the remaining inventory entry fields were filled with valid values, and while the main file entry field was a valid filename for a file that actually exists. As with the previous tests, I constantly restarted the output window by opening a file to output its inventory. This ensured that the “Invalid input” error message is output to the user each time. I also checked the IDLE terminal window for error messages that were relevant for the program itself, to check if any invalid input for the inventory item entry fields was slipping through.

I checked for the correct error message if any inventory item entry field was left empty and if there was a space in any inventory item entry field. The text files that contain the inventory lists separate the individual inventory item values with spaces, so if a space is contained in any of these values, it interferes with obtaining the inventory dictionary from a file. The inventory value with a space is interpreted as two separate inventory item values. The index for each inventory item value is important in my program, so the correct inventory item values must be in the correct locations.

The SKU entry field must contain four characters, so I tested the SKU entry field with a number of characters that was one fewer and one more than the strict character count. When the sorting function is performed on the inventory dictionary, the inventory dictionary is only properly sorted if the SKU for each inventory item has the same number of characters. The name and department values must be ten characters or fewer, so I tested for values that were one greater than this character limit. To make it easier to tell if the inventory item entry fields at the correct number of characters for each test, the characters that were added to a valid value were numbers that represented the number of characters. The number 0 represents the tenth character, and the repeated number 0 represents the eleventh character. All of the inventory items are output to the user as a formatted table. The character limit ensures the proper formatting of the table.

The quantity and price values have the same ten character limit, so they were tested in the same fashion as the name and department values. The quantity and price values must also be digits only. I tested both values with non-digit entries. The character limit for the quantity and price values exists for the same reason as the character limit for the name and department values: ensure the proper formatting for the output table. This is also why the quantity and price values must be digits only. Numbers with digits that are past the decimal point make the proper formatting more complicated and less reliable. Mathematical functions are performed on the quantity and price values. Both values also need to be numerical.

As with the previous validity check modules, I tested the module for this with a separate script before I implemented the module in the entire program. Once I placed the validity check module in my entire program, it functioned properly.

**SKU Entry Field**

This was only relevant for the “Delete From Inventory” button. The input validation for the SKU entry field is the exact same as it is for inventory item value input validation. The same tests were performed in the same fashion, but only for the SKU entry field. The “Invalid input” message is the exact same, but it only covers the SKU. For the “Delete From Inventory” button to function properly, the SKU value in the SKU entry field needs to actually exist in the inventory dictionary. There is a validation check for the existence of an SKU value. If the SKU value does not exist in the inventory dictionary, then the “SKU not found in inventory” message is the correct output of the program. To test for this message, I opened a file with inventory items. I then attempted to delete an inventory item using a valid SKU value that does not exist in the file.

When I first incorporated the SKU validity check module into my full program, the “Invalid input” message was skipped over. My program went straight to the “SKU not found in inventory message,” which is connected to a separate module. This was not a major bug, because an invalid SKU value would not exist in an inventory file. I tested the SKU validity check module in a separate script, and it worked properly. This indicated that the issue was with the decision structure that contains the callback functions for the input check modules. I discovered that I had accidentally used double equal signs (i.e., ==) while initializing the flag variable for SKU validity. The validity test was basically skipped over.

As with the previous validity check modules, I tested the modules for this with separate scripts before I implemented the module in the entire program. Once I placed the validity check modules in my entire program, they functioned properly.

**Error Message Precedence**

The “Copy File” button, “Open File” button, “Add To File” button, and “Delete From Inventory” button can each produce multiple error messages. There is an order of precedence for the error messages, in that some error messages are output before other error messages, even when multiple types of invalid input are present. The order matches the level of significance for each type of invalid input. I tested the order of precedence for each type of invalid input for each button that can have multiple forms of invalid input. I tested the order of precedence by filling every relevant entry field with invalid values, to see which error message is output. I then corrected the particular value that produced the error message, to see which error message is output next. I repeated this process for each type of invalid input for each relevant entry field.

For the “Copy File” button, the validity of the main file entry field is tested first, the existence of the filename for the main file entry field is tested second, and the validity of the copy file entry field is tested last. Error messages are displayed for the test that occurs first, if everything is invalid. For the “Open File” button, the validity of the main file entry field is tested first, and the existence of the filename for the main file entry field is tested last. For the “Add To File” button, the validity of the main file entry field is tested first, the validity of the inventory item entry fields is tested next, and the existence of the filename for the main file entry field is tested last. For the “Delete From File” button, the validity of the main file entry field is tested first, the existence of the filename for the main file entry field is tested second, the validity of the SKU entry field is tested third, and the existence of the SKU value in the file is tested last.

**New File**

Compared to testing for error messages, the main functions of the program were much easier to test. To test the “New File” button, I entered valid filenames for the main file entry field and checked for the existence of the files in the file directory of the program. I did this on three separate occasions. I decided to add an output message that occurs after the creation of a new file, to indicate that a new file has been created. Correct output is when a new file is created with the chosen file name in the inventory file directory folder.

As with the modules that detected invalid user input, the module that is relevant to the creation of new files was tested in a separate script before being implemented in the entire program.

**Copy File**

To test the “Copy File” button, I entered a valid filename for the main file entry field and checked for the existence of the copies in the file directory of the program. I used four separate filenames for the copy file entry field. I decided to add an output message that occurs after the creation of a copy file, to indicate that a copy file has been created. Correct output is when the main file and the copy file both have the same timestamp, since the inventory of the main file is automatically sorted and saved to the main file during the creation of the copy file.

After establishing the creation of copy files, I checked the contents of copy files, to see if they matched the contents of the original files. I created three separate inventory files, then I created a copy of each inventory file plus an extra copy of a single inventory file. I checked if the inventory files were identical by outputting their inventory lists with the “Open File” button. Correct output is when the copy files are identical to the original main files.

The modules that were relevant to the creation copy files were also tested in separate scripts before being implemented into the entire program. This is the first major program function that involves the use of multiple sub-functions that can function interchangeably, which made testing each sub-function separately a necessity.

**Open File**

To test the “Open File” button, I used valid filenames for three separate files that actually exist in the file directory. I created these files beforehand, and I gave each file a different yet simple inventory list, to differentiate them in a way that was easy to notice. This ensured that each separate file was actually being opened. The test files were filled with inventory items before testing the “Open File” button. This was done to ensure that the inventory lists were properly output and that the calculated values were accurate. Correct output is when the inventory list is output in the output window, when the inventory list is properly sorted by SKU, when the output is properly formatted as an organized table, and when the derived value of total inventory value is properly calculated.

Outputting the inventory list of a file used some of the same interchangeable sub-functions as copying a file, with the addition of a few other functions. These sub-functions were independently tested in separate scripts before being implemented into the entire program.

**Add To File**

To test the “Add To File” button, I used valid filenames for files that actually exist in the file directory. I then used valid values for each of the inventory item entry fields. I made sure that the SKU’s for each inventory item were not entered in sorted order (i.e., from least to greatest or greatest to least). I did this to test the functionality of the inventory dictionary sorting function. I used a combination of numbers and letters for the SKU, to see how numbers and letters are sorted when both are used. In Python, numbers come before letters. I used four separate inventory items to test everything. The inventory list of a file is automatically sorted, saved to the file, and output to the user after an inventory item is added to a file. I checked if the added inventory items matched the values of the inventory item entry fields, and if the inventory items were sorted properly by SKU. Correct output is when the altered inventory list is output in the output window with the new inventory item, when all of the inventory items are properly sorted by SKU, when the output is organized in a properly formatted table, when the derived value of inventory item value has been properly calculated for the new inventory item, and when the derived value of total inventory value has been properly calculated.

This main function uses all of the same sub-functions as opening a file to output its dictionary, with several extra sub-functions that obtain and add inventory items to the main file. These sub-functions were independently tested in separate scripts before being implemented into the entire program.

**Delete From Inventory**

To test the “Delete From Inventory” button, I used valid filenames for files that actually exist in the file directory. I created a basic inventory file that only contained four inventory items. I opened the file so that I could know the SKU’s. I then entered each SKU to see if the inventory items would actually be removed from the file. The inventory list from a file is automatically saved to the file and output to the user after an inventory item is removed. I just made sure that each inventory item was actually being removed by checking the inventory list output. Correct output is when the altered inventory list is output in the output window, when this inventory list is missing the deleted inventory item, when the output is in a properly formatted table, and when the derive value of total inventory value is properly calculated.

This main function uses the same sub-functions as opening a file with some extra functions that are relevant to deleting an inventory item from a file. These functions were independently tested in separate scripts before being implemented into the entire program.

**Close Program**

Every single time that I have closed my program, I have used the “Close Program” button. Correct output is when both the input window and the output window are simultaneously closed. This is the simplest of all the main functions. It was tested in a separate script before being implemented into the entire program.

**Window Resizable**

I checked if I could resize the input window and the output window. Only the y-axis of the output window can be expanded. Both the output textbox and the scrollbar expand with the output window.

**Scrollbar**

I created an inventory file with a large number of inventory items, to see if the scrollbar works properly. I then expanded and shrank the output window, to see if the scrollbar matched the vertical size of the output window.

**Datasets For Testing**

**Order of Functions**

To test the proper functionality of the main program functions fully, the datasets for the main functions must be followed in the precise order in which they are presented. New files need to be created for the sake of containing inventory lists, inventory items must be added to the files so that the files can be copied and opened in a meaningful way, and so that the inventory of a single file can be gradually deleted. At least one copy file must be created so that when the inventory is gradually deleted from a file the original source file still exists. The datasets for each function set up all of the tests for the primary functionality of each button.

**Pre-Existing Files**

There are six pre-existing inventory text files in the Inventory Files folder of the program. The files “inventory01”, “inventory02”, and “inventory03” are identical to the files “inventory1”, “inventory2”, and “inventory3” that can be created by using the main functions and their datasets in the proper order. The file “inventory04” is an experiment to see how Python sorts string values other than letters and digits. Python appears to have its own standards of precedence. The file “inventory05” is used to test the scroll bar functionality of the program. The file “inventory06” is an experiment with how Python handles exceptionally large numerical values. It appears that Python automatically resorts to using scientific notation. Numbers of exceptional magnitude can be processed, but some accuracy is lost.

**New File**

Main File Name: inventory1

Main File Name: inventory2

Main File Name: inventory3

**Add To File**

Main File Name: inventory1

SKU: 0001

Name: Diapers

Department: Infant

Quantity: 200

Price: 30

Main File Name: inventory1

SKU: 0000

Name: Boots

Department: Footwear

Quantity: 20

Price: 5

Main File Name: inventory1

SKU: AA00

Name: Shoes

Department: Footwear

Quantity: 20

Price: 3

Main File Name: inventory1

SKU: A000

Name: Licorice

Department: Candy

Quantity: 100

Price: 2

Main File Name: inventory2

SKU: 0025

Name: Bubblegum

Department: Candy

Quantity: 100

Price: 1

Main File Name: inventory2

SKU: 0050

Name: Licorice

Department: Candy

Quantity: 100

Price: 2

Main File Name: inventory2

SKU: 0005

Name: Red-Rope

Department: Candy

Quantity: 100

Price: 2

Main File Name: inventory3

SKU: 0010

Name: Boots

Department: Footwear

Quantity: 100

Price: 15

Main File Name: inventory3

SKU: 0005

Name: High-Heels

Department: Footwear

Quantity: 100

Price: 20

Main File Name: inventory3

SKU: 0000

Name: Sandals

Department: Footwear

Quantity: 100

Price: 5

Main File Name: inventory3

SKU: 0025

Name: Sneakers

Department: Footwear

Quantity: 100

Price: 10

**Copy File**

Main File Name: inventory1

Copy File Name: inventory5

Main File Name: inventory2

Copy File Name: inventory6

Main File Name: inventory3

Copy File Name: inventory7

Main File Name: inventory3

Copy File Name: inventory4

**Open File**

Main File Name: inventory1

Main File Name: inventory2

Main File Name: inventory3

Main File Name: inventory5

Main File Name: inventory6

Main File Name: inventory7

Main File Name: inventory4

**Delete From Inventory**

Main File Name: inventory4

SKU: 0025

Main File Name: inventory4

SKU: 0010

Main File Name: inventory4

SKU: 0005