Analysis

- **Introduction**
  - Student must design and make a program that is able to take an input as a list of integers and validate the input so that it can be used to calculate the total cost of the order by referencing the existing menu that has been created. The user should also be able to add, amend and delete menu items, and save menu changes that are saved to a file for later. The program must maintain a running total of order values, totals of quantity of each menu item ordered, loop the input for more orders and display order details for the user.

- **Key Requirements of the Program**
  - User should be able to input order details with table number and a string of numbers corresponding to menu items chosen
  - Program must be able to validate input data
  - Display the order details for printing
  - Loop for next order
  - Allow menu to be saved to a file
  - Allow for user to amend, add, delete menu items as well as save menu changes
  - Maintain running total of order values
  - Maintain running totals of the quantity
  - Save running totals to a file
  - Provide options to display the menu and running totals

- **What is decomposition?**
  - Decomposition is a general approach to solving a problem by breaking it up into smaller problems then solving it one problem at a time.
    - E.g. Separate function for validating input data
    - Separate function for calculating total
    - Separate function for writing to file and saving it

- **Sub-problems**
  - Sub-problem #1: Validating input data
    - The subprogram will take input data and then check the data to make sure that it fits the criteria of the input data. E.g. if input is wrong data type (e.g. string) the sub-program would not accept this input because then the whole program will not work with the data time.

  - Sub-problem #2: Calculating total cost of order
    - The subprogram will take the quantity of the menu items, then look at the menu items listed in the file and calculate the total cost by multiplying the amount of that item with the price listed in the menu item file.

  - Sub-problem #3: Saving and editing menu using files saved locally.
    - The sub-program will display a menu where the user view, edit and add to the menu items and add prices. The sub-program must be able to save changes to a file so that it can be accessed after the program has stopped running.

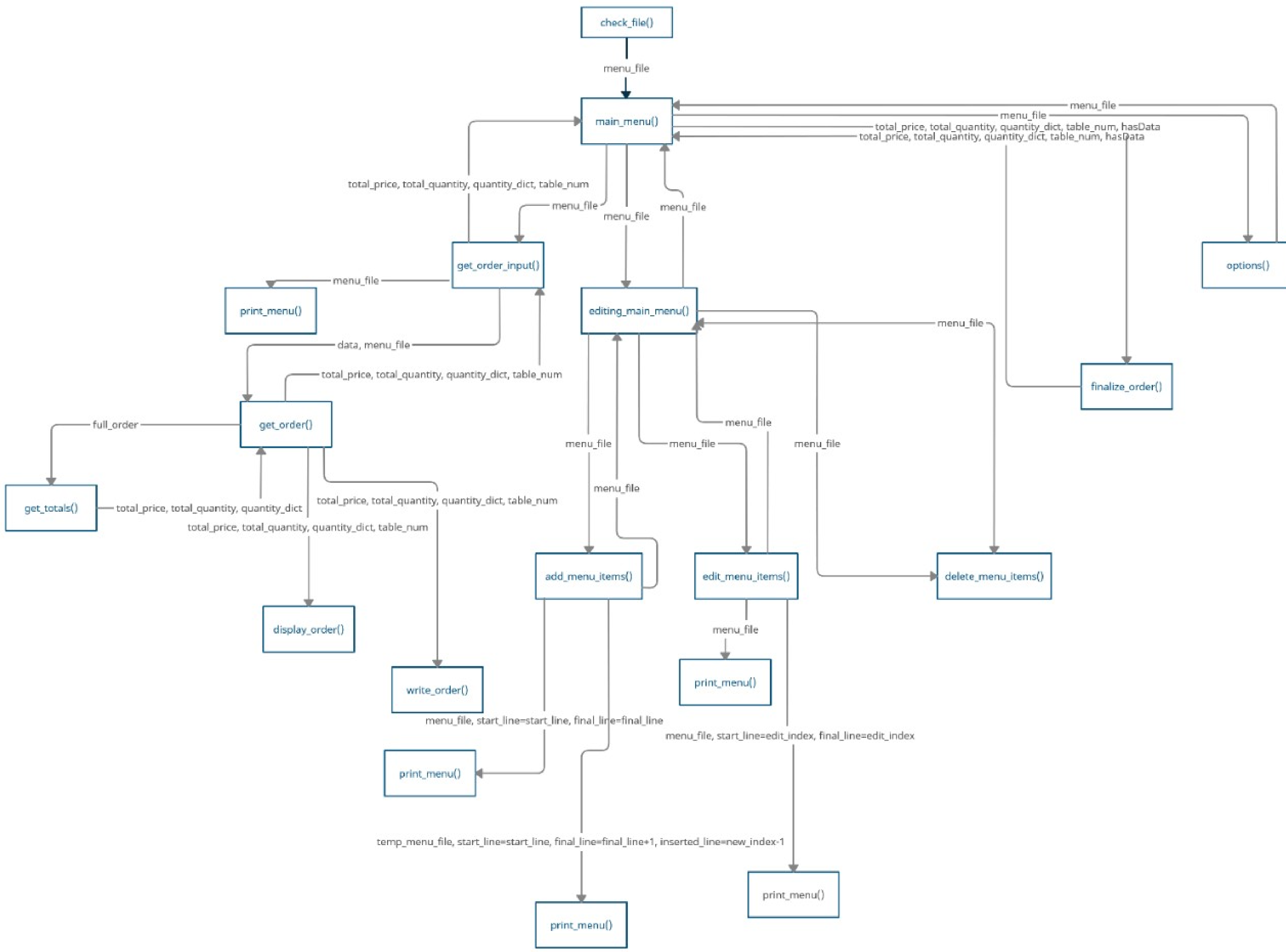
  - Sub-problem #4: Display final totals (e.g. **Balance due** and **Quantities** of menu items
    - The sub program will take the final totals from sub-problem #2 and it will display it in a generated table ready for printing.

  - I have broken down the problem into these specific sub-problems because each sub-problem represents a smaller problem that is crucial to the final program working. They are small and specific tasks that perform different and unique tasks from each other. Sub-problems should be of similar size, but of different processes/tasks than other sub-problems.

Design

- **Overall Plan**
  - This is a hierarchy chart displaying the whole program solution. When the program starts, check_file() starts after main_menu() is run and the user can choose either get_order_input(), editing_main_menu() finalize_order() or options(). The subprograms that the user can choose all return information that is used with the other subprograms to work properly. Without this information the program will not work and information will not be passed around. These subprograms run and when the user decides to exit those sub programs they will be brought back to the main menu unless they choose to exit. This is a basic abstraction of how the overall program's subprograms work together.

Pseudo Code on next page.


check_file()
menu_file
main_menu()
menu_file
menu_file
total_price, total_quantity, quantity_dict, table_num, hasData
total_price, total_quantity, quantity_dict, table_num, hasData
options()
finalize_order()
total_price, total_quantity, quantity_dict, table_num
menu_file
menu_file
menu_file
get_order_input()
menu_file
print_menu()
editing_main_menu()
menu_file
data, menu_file
total_price, total_quantity, quantity_dict, table_num
get_order()
full_order
get_totals()
total_price, total_quantity, quantity_dict
total_price, total_quantity, quantity_dict, table_num
total_price, total_quantity, quantity_dict, table_num
display_order()
write_order()
menu_file
menu_file
menu_file
menu_file
menu_file
add_menu_items()
edit_menu_items()
delete_menu_items()
menu_file
print_menu()
menu_file, start_line=start_line, final_line=final_line
menu_file, start_line=edit_index, final_line=edit_index
print_menu()
temp_menu_file, start_line=start_line, final_line=final_line+1, inserted_line=new_index-1
print_menu()
print_menu()

- **Individual subroutine plans**
  - Check “Pseudo Code.txt” for **<u>FULL</u>** Individual Subroutine plans in pseudo code

  - **Subroutine Descriptions:**
    - ***finalize_order()*** – takes all the totals and data from what the user has entered as their order and then displays it for printing.

    - ***options()*** – displays an options menu for the user so that they can choose if they want to manually reset the menu back to defaults

    - ***delete_menu_items()*** – allows the user to delete existing menu items that are stored in the menu that will be eventually written to: “menu.txt”

    - ***write_menu()*** – takes the menu then, creates menu.txt and writes to it from menu_file that is passed into it as an parameter

    - ***edit_menu_items()*** – allows the user to take an existing menu item and edit it’s index, name, price or category and saves changes to the menu

    - ***add_menu_items()*** – allows the user to add menu items to the menu. When adding an item, the user can decide it’s price, name, index number and category

    - ***editing_main_menu()*** – Provides the menu interface for the user to choose to Add, edit or delete menu items in the menu. Exiting this menu takes the user to the main_menu()

    - ***write*_order()** – Takes the user’s order totals such as total price and total quantity and writes it to “running_totals.txt”

    - ***display_order()*** – displays the current order with the quantities of each menu item ordered

    - ***get_quantity()*** – Generates a dictionary of how many of each item has been ordered in a simple dictionary format for display_order()

    - ***get_totals()*** – Calculates quantity and cost totals for display_order()

    - ***get_order()*** – Checks order with menu in “menu.txt” and generates the variable: “full_order”

    - ***print_menu()*** – Takes menu.txt from the “menu_file” variable and prints it to the user when so that the user can see a clear view of the current menu with index numbers, prices, names and the category that it is in

    - ***get_order_input()*** – Takes the user’s input for their order and validates it and formats it properly so that it can be used later in the program by other subroutines

- ***main_menu()*** – The Main Menu of the program, it’s the first menu that the user sees. The user can choose to Input order data, edit menu items, finalize order, change options or quit to desktop.

- ***check_file()*** – Checks for the menu.txt and creates menu.txt it if it doesn’t exist using the tuple: DEFAULT_MENU

- **Test table for carrying out tests in the program:**
  - Each color in the “Test No.” column represents a different module/subroutine
    - The module/subprogram is written in ***BOLD*** in the *“Purpose of Test”*
  - Each shade of that colour represents a single input with three different types of data: *Errorneous, Boundary and Normal*
  - There are 20 different input statements that the user can interact with in the program according to this table

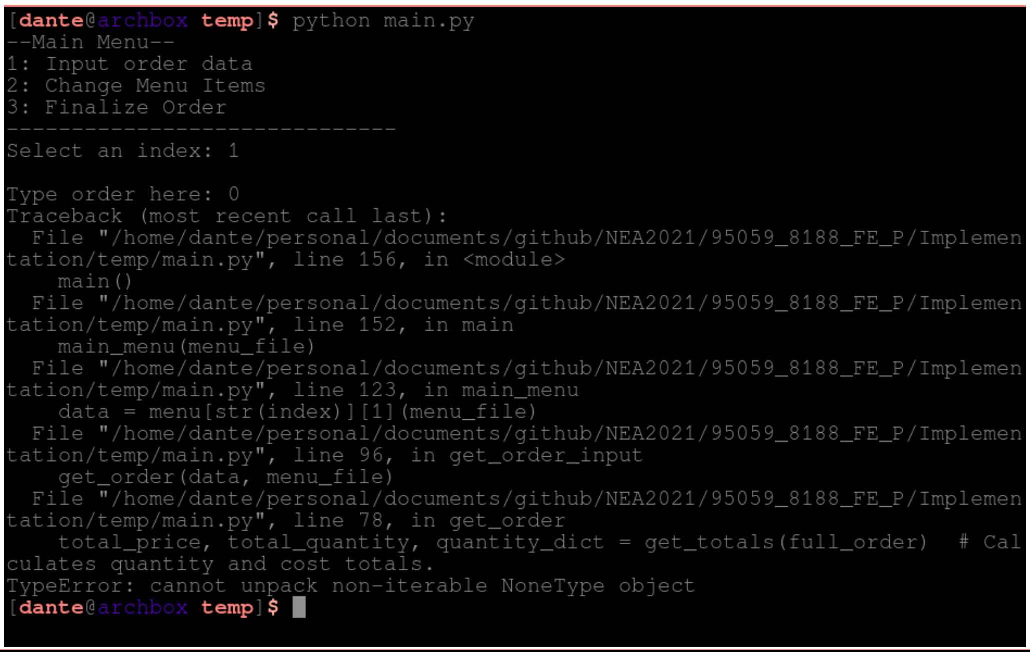
Stage 2 – Initial Test Plan			**Stage 3 and 4 – Test Table**		
Test No.	**Type of Data**	**Purpose of Test**	**Test Data**	**Expected Result**	**Actual Result**
1	Erroneous	Check that menu index is only an integer ***main_menu()***	abc123.!@#!!|})(*#&	“Invalid Index”	“Invalid Index”
2	Boundary	Check that menu index is in range ***main_menu()***	0	“Out of range try again!”	“Out of range try again!”
3	Normal	Check that menu index is in range ***main_menu()***	1	“E.g. \"6,4,4,7,8,10,10 Would be an order from Table 6 for 2 Burgers, 1 Fries, 1 Salad and 2 Soft Drinks”	“E.g. \"6,4,4,7,8,10,10 Would be an order from Table 6 for 2 Burgers, 1 Fries, 1 Salad and 2 Soft Drinks”
4	Normal	Check that input order formats correctly ***get_order_input()***	6, 1, 8, 5,4 ,2, 3, 6	*Shows correct output with items listed*	*Shows correct output with items listed*
5	Boundary	Check if user can’t input a table number or menu index number higher than what is already accepted ***get_order_input()***	100,1,1,1,1000,200	“We only have 10 tables! Table number must be lower than 10, please try again.”	“We only have 10 tables! Table number must be lower than 10, please try again.”
6	Erroneous	Check if user can’t input table order with characters that aren’t integers ***get_order_input()***	A, b ,c $, 1, _”:<AD>>{_+ “:<	“Your order has invalid characters, please try again.”	“Your order has invalid characters, please try again.”
7	Normal	Check that menu index is only an integer ***editing_main_menu()***	abc123.!@#!!|})(*#&	“Invalid index”	“Invalid index”
8	Boundary	Check that menu index is in range ***editing_main_menu()***	0	“Out of range try again!”	“Out of range try again!”
9	Erroneous	Check that menu index is in range ***editing_main_menu()***	1	*Shows correct output with items listed*	*Shows correct output with items listed*
10	Normal	Check that category in user input is a valid option ***add_menu_items()***	mains	“Please choose the name of your new menu item.”	“Please choose the name of your new menu item.”
11	Boundary	Check that category in user input is a valid option ***add_menu_items()***	deserts	“Please enter a valid category, try again”	“Please enter a valid category, try again”
12	Erroneous	Check that category in user input is a valid option ***add_menu_items()***	abc123.!@#!!|})(*#&	“Please enter a valid category, try again”	“Please enter a valid category, try again”
13	Normal	Check that name in user input is a valid option ***add_menu_items()***	Double Cheeseburger		
14	Boundary	Check that name in user input is a valid option ***add_menu_items()***	123 Burger 123	“Name must contain alphabetic characters only, try again.”	“Name must contain alphabetic characters only, try again.”
15	Erroneous	Check that name in user input is a valid option ***add_menu_items()***	abc123.!@#!!|})(*#&	“Name must contain alphabetic characters only, try again.”	“Name must contain alphabetic characters only, try again.”
16	Normal	Check if the price in the user input is a valid option ***add_menu_items()***	100	*Shows correct output with menu items listed in the specified category*	*Shows correct output with menu items listed in the specified category*
17	Boundary	Check if the price in the user input is a valid option ***add_menu_items()***	0.01	*Shows correct output with menu items listed in the specified category*	*Shows correct output with menu items listed in the specified category*
18	Erroneous	Check if the price in the user input is a valid option ***add_menu_items()***	abc123.!@#!!|})(*#&	Please input numeric characters only, try again.	Please input numeric characters only, try again.
19	Normal	Check if the new menu index entered is valid ***add_menu_items()***	4	*Shows correct output with the new menu item inserted into the menu with a arrow pointing at it.*	*Shows correct output with the new menu item inserted into the menu with a arrow pointing at it.*
20	Boundary	Check if the new menu index entered is valid ***add_menu_items()***	7	Please enter a value between 3 and 6.	Please enter a value between 3 and 6.
21	Erroneous	Check if the new menu index entered is valid ***add_menu_items()***	abc123.!@#!!|})(*#&	Your index must contain only numeric characters only, try again.	Your index must contain only numeric characters only, try again.
22	Normal	Check if the input for if deciding whether to save changes, retry, add another item or discard changes is valid ***add_menu_items()***	“S” ***OR*** “s” ***OR*** “R” ***OR*** “r” ***OR*** “A” ***OR*** “a” ***OR*** “D” ***OR*** “d”	*Returns user to Edit Menu Items menu and Shows correct output*	*Returns user to Edit Menu Items menu and Shows correct output*
23	Boundary	Check if the input for if deciding whether to save changes, retry, add another item or discard changes is valid ***add_menu_items()***	B	Please enter either:"S" to save, "R" to retry, "A" to add another item "D" to discard changes and exit	Please enter either:"S" to save, "R" to retry, "A" to add another item "D" to discard changes and exit
24	Erroneous	Check if the input for if deciding whether to save changes, retry, add another item or discard changes is valid ***add_menu_items()***	abc123.!@#!!|})(*#&	Please enter either:"S" to save, "R" to retry, "A" to add another item "D" to discard changes and exit	Please enter either:"S" to save, "R" to retry, "A" to add another item "D" to discard changes and exit
25	Normal	Check if menu index chosen by the user is valid ***edit_menu_items()***	4	Enter a letter from below to choose what to edit of Burger: (I)ndex # of item (N)ame of item (P)rice (C)ategory Enter "E" to (E)xit and choose another item edit or exit.	Enter a letter from below to choose what to edit of Burger: (I)ndex # of item (N)ame of item (P)rice (C)ategory Enter "E" to (E)xit and choose another item edit or exit.
26	Boundary	Check if menu index chosen by the user is valid ***edit_menu_items()***	13	Your index must be between: 1 and 12 Please try again...	Your index must be between: 1 and 12 Please try again...

27	Erroneous	Check if menu index chosen by the user is valid ***edit_menu_items()***	abc123.!@#!!|})(*#&	Please enter numeric characters only, try again.	Please enter numeric characters only, try again.
28	Normal	Check if the choice made by the user when selecting what to edit of the menu item is a valid selection ***edit_menu_items()***	“I” ***OR*** “i” ***OR*** “N” ***OR*** “n” ***OR*** “P” ***OR*** “p” ***OR*** “C” ***OR*** “c”	Choose the menu item index number that you want it to change to: Index Number:	Choose the menu item index number that you want it to change to: Index Number:
29	Boundary	Check if the choice made by the user when selecting what to edit of the menu item is a valid selection ***edit_menu_items()***	A	Your input must be either: "I", "N", "P" or "C". Try again!	Your input must be either: "I", "N", "P" or "C". Try again!
30	Erroneous	Check if the choice made by the user when selecting what to edit of the menu item is a valid selection ***edit_menu_items()***	abc123.!@#!!|})(*#&	Your input must be either: "I", "N", "P" or "C". Try again!	Your input must be either: "I", "N", "P" or "C". Try again!
31	Normal	Check if the menu item index number that the user wants to change the menu item to is valid ***edit_menu_items()***	10	*Shows the valid input with the menu item’s index number changed.*	*Shows the valid input with the menu item’s index number changed.*
32	Boundary	Check if the menu item index number that the user wants to change the menu item to is valid ***edit_menu_items()***	“0” ***OR*** “13”	Your index must be between: 1 and 12 Please try again...	Your index must be between: 1 and 12 Please try again...
33	Erroneous	Check if the menu item index number that the user wants to change the menu item to is valid ***edit_menu_items()***	abc123.!@#!!|})(*#&	Please enter numeric characters only, try again.	Please enter numeric characters only, try again.
34	Normal	Check if the name entered by the user when editing the name of the menu item is valid ***edit_menu_items()***	Double Mega Burger	*Shows the menu item with the new name*	*Shows the menu item with the new name*
35	Boundary	Check if the name entered by the user when editing the name of the menu item is valid ***edit_menu_items()***	Name123	Name must contain alphabetic characters only, try again.	Name must contain alphabetic characters only, try again.
36	Erroneous	Check if the name entered by the user when editing the name of the menu item is valid ***edit_menu_items()***	abc123.!@#!!|})(*#&	Name must contain alphabetic characters only, try again.	Name must contain alphabetic characters only, try again.
37	Normal	Check if the price entered by the user is valid ***edit_menu_items()***	40	*Shows the new menu item with the price entered by the user.*	*Shows the new menu item with the price entered by the user.*
38	Boundary	Check if the price entered by the user is valid ***edit_menu_items()***	10000	*Shows the new menu item with the price entered by the user.*	*Shows the new menu item with the price entered by the user.*
39	Erroneous	Check if the price entered by the user is valid ***edit_menu_items()***	abc123.!@#!!|})(*#&	Please input numeric characters only, try again.	Please input numeric characters only, try again.
40	Normal	Check if the category entered by the user when editing the category is a valid category ***edit_menu_items()***	Drinks	*Shows the menu with only the drinks showing and asks the user to input a menu index on where to insert the new menu item in the specified category*	*Shows the menu with only the drinks showing and asks the user to input a menu index on where to insert the new menu item in the specified category*
41	Boundary	Check if the category entered by the user when editing the category is a valid category ***edit_menu_items()***	Desserts	Please enter a valid category, try again.	Please enter a valid category, try again.
42	Erroneous	Check if the category entered by the user when editing the category is a valid category ***edit_menu_items()***	abc123.!@#!!|})(*#&	Please enter a valid category, try again.	Please enter a valid category, try again.
43	Normal	Check if the new menu index that the user wants to select when chanigng the menu item’s category is correct. ***edit_menu_items()***	12	*Shows the menu item with the menu item index that the user specified where choosing where to insert the menu item into the category.*	*Shows the menu item with the menu item index that the user specified where choosing where to insert the menu item into the category.*
44	Boundary	Check if the new menu index that the user wants to select when chanigng the menu item’s category is correct. ***edit_menu_items()***	13	Please enter a value between 9 and 12.	Please enter a value between 9 and 12.
45	Erroneous	Check if the new menu index that the user wants to select when chanigng the menu item’s category is correct. ***edit_menu_items()***	abc123.!@#!!|})(*#&	Your index must contain only numeric characters only, try again.	Your index must contain only numeric characters only, try again.
46	Normal	Check if the menu item index input is valid when the user is trying to delete a menu item. ***delete_menu_items()***	5	Are you sure you want to delete Sparkling water? - Enter "Y" to Confirm Menu Item Deletion - Enter "N" to Discard Changes and Exit to Edit Menu	Are you sure you want to delete Sparkling water? - Enter "Y" to Confirm Menu Item Deletion - Enter "N" to Discard Changes and Exit to Edit Menu
47	Boundary	Check if the menu item index input is valid when the user is trying to delete a menu item. ***delete_menu_items()***	14	Your delete index must be between: 1 and 13 Please try again...	Your delete index must be between: 1 and 13 Please try again...
48	Erroneous	Check if the menu item index input is valid when the user is trying to delete a menu item. ***delete_menu_items()***	abc123.!@#!!|})(*#&	Please enter numeric characters only, try again	Please enter numeric characters only, try again
49	Normal	Check if the user input when confirming to delete the menu item is valid ***delete_menu_items()***	“Y” ***OR*** “y” “0” ***OR*** “N” ***OR*** “n”	*Shows the new menu wth the selected menu item deleted and all the menu items in the correct order.*	*Shows the new menu wth the selected menu item deleted and all the menu items in the correct order.*
50	Boundary	Check if the user input when confirming to delete the menu item is valid ***delete_menu_items()***	abcdefg	Please enter "y" or "n" as your choice, try again.	Please enter "y" or "n" as your choice, try again.
51	Erroneous	Check if the user input when confirming to delete the menu item is valid ***delete_menu_items()***	abc123.!@#!!|})(*#&	Please enter "y" or "n" as your choice, try again.	Please enter "y" or "n" as your choice, try again.
52	Erroneous	Check if the user trying to finalize order before having entered an order gives out an error ***finalize_order()***	3	No current order! Go back and Input an order.	No current order! Go back and Input an order.
53	Normal	Check if the user’s input when choosing whether to “*Exit and enter another order*”, “*exit and keep current order*” or to “*quit the program*” is valid ***finalize_order()***	“Y” ***OR*** “y” “0” ***OR*** “N” ***OR*** “n” ***OR*** “Q” ***OR*** “q”	*- Choosing “y” would exit and reset the order made by the user - Choosing “n” would exit and keep the current order that the user has made - Choosing “q” would quit the program as expected*	*- Choosing “y” would exit and reset the order made by the user - Choosing “n” would exit and keep the current order that the user has made - Choosing “q” would quit the program as expected*
54	Boundary	Check if the user’s input when choosing whether to “*Exit and enter another order*”, “*exit and keep current order*” or to “*quit the program*” is valid ***finalize_order()***	abc	Please enter "y", "n" or "q"as your choice, try again. - Enter "Y" to Exit and enter another order - Enter "N" to Exit and keep current order - Enter "Q" to Quit the program	Please enter "y", "n" or "q"as your choice, try again. - Enter "Y" to Exit and enter another order - Enter "N" to Exit and keep current order - Enter "Q" to Quit the program
55	Erroneous	Check if the user’s input when choosing whether to “*Exit and enter another order*”, “*exit and keep current order*” or to “*quit the program*” is valid ***finalize_order()***	abc123.!@#!!|})(*#&	Please enter "y" or "n" as your choice, try again. - Enter "Y" to Exit and enter another order - Enter "N" to Exit and keep current order - Enter "Q" to Quit the program	Please enter "y", "n" or "q"as your choice, try again. - Enter "Y" to Exit and enter another order - Enter "N" to Exit and keep current order - Enter "Q" to Quit the program
56	Normal	Check if the choice index for choosing whether to Reset Menu Items File or to Exit to Main Menu is valid ***options()***	1	Are you sure you want to reset the Menu Items File? (menu.txt)? This will erase all saved changes to the file and reset it to defaults. - Enter "Y" to CONFIRM MENU RESET - Enter "N" to DISCARD CHANGES AND EXIT	Are you sure you want to reset the Menu Items File? (menu.txt)? This will erase all saved changes to the file and reset it to defaults. - Enter "Y" to CONFIRM MENU RESET - Enter "N" to DISCARD CHANGES AND EXIT
57	Boundary	Check if the choice index for choosing whether to Reset Menu Items File or to Exit to Main Menu is valid ***options()***	“3” ***OR*** “0”	Please enter 1 or 2 as your choice.	Please enter 1 or 2 as your choice.
68	Erroneous	Check if the choice index for choosing whether to Reset Menu Items File or to Exit to Main Menu is valid ***options()***	abc123.!@#!!|})(*#&	Please enter numeric characters only, try again.	Please enter numeric characters only, try again.
69	Normal	Check if the deletion confirmation message is valid ***options()***	“N” ***OR*** “n” ***OR*** “Y” ***OR*** “y”	*Selecting “Y” displays: “File reset!” then returns the user to the main menu. Selecting “N” displays: “Exiting...” then returns the user to the main menu.*	*Selecting “Y” displays: “File reset!” then returns the user to the main menu. Selecting “N” displays: “Exiting...” then returns the user to the main menu.*
70	Boundary	Check if the deletion confirmation message is valid ***options()***	a	Please enter "y" or "n" as your choice, try again.	Please enter "y" or "n" as your choice, try again.
71	Erroneous	Check if the deletion confirmation message is valid ***options()***	abc123.!@#!!|})(*#&	Please enter "y" or "n" as your choice, try again.	Please enter "y" or "n" as your choice, try again.

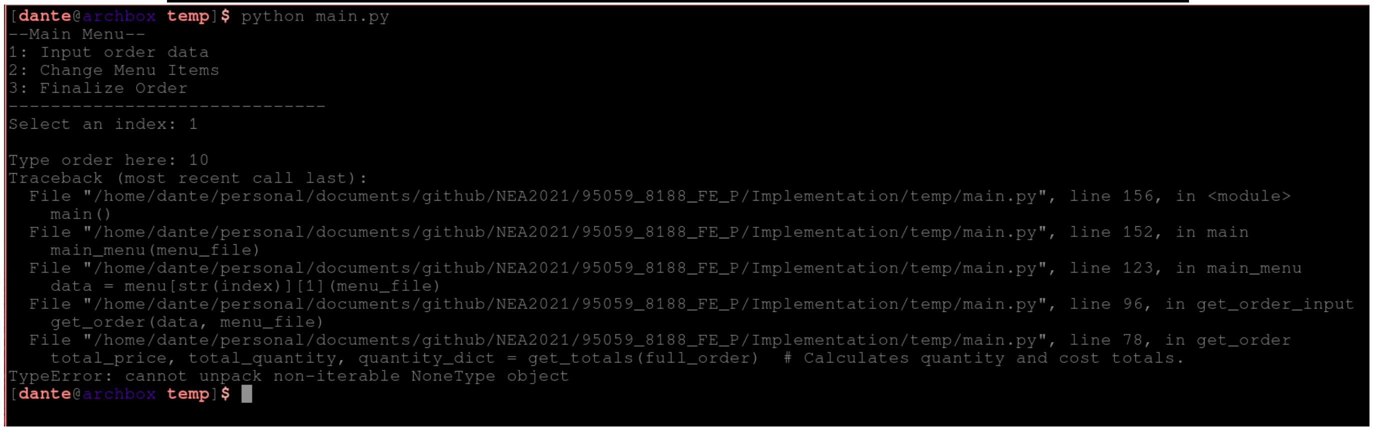
Debugging

- **Bug #1: Fixing Order Input Validation in get_order_input() *(Runtime Error)***
  - When entering a single value for the table number that is equal to or less than 10 the program would raise **TypeError**
    - Examples: “0” would raise **TypeError**, as well as entering “10”

Error Messages:

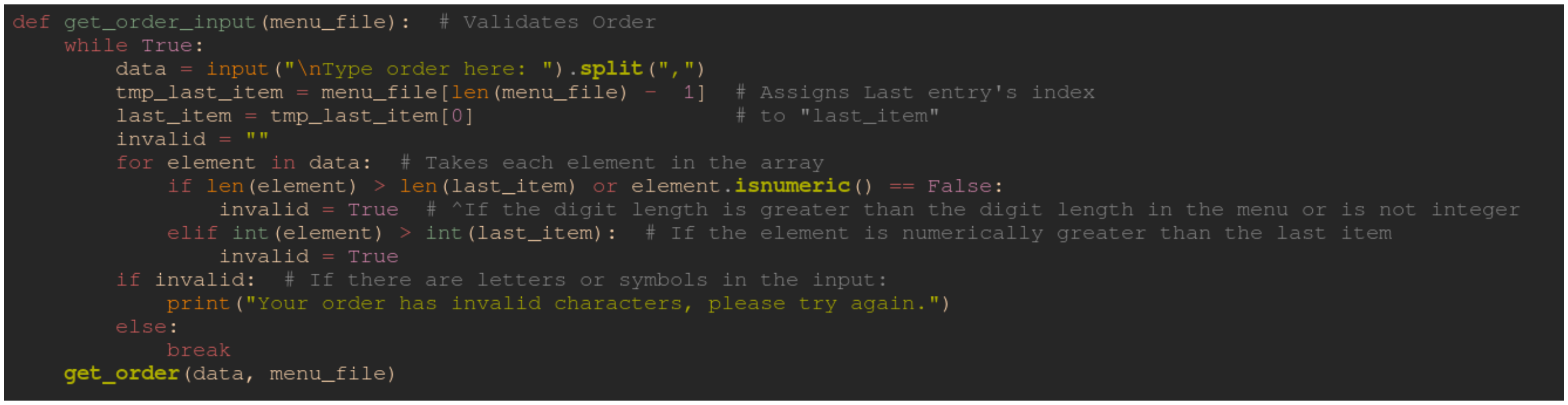


```
[dante@archbox temp]$ python main.py
--Main Menu--
1: Input order data
2: Change Menu Items
3: Finalize Order
------------------------------
Select an index: 1

Type order here: 0
Traceback (most recent call last):
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 156, in <module>
    main()
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 152, in main
    main_menu(menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 123, in main_menu
    data = menu[str(index)][1](menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 96, in get_order_input
    get_order(data, menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 78, in get_order
    total_price, total_quantity, quantity_dict = get_totals(full_order)  # Calculates quantity and cost totals.
TypeError: cannot unpack non-iterable NoneType object
[dante@archbox temp]$ 
```



```
[dante@archbox temp]$ python main.py
--Main Menu--
1: Input order data
2: Change Menu Items
3: Finalize Order
------------------------------
Select an index: 1

Type order here: 10
Traceback (most recent call last):
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 156, in <module>
    main()
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 152, in main
    main_menu(menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 123, in main_menu
    data = menu[str(index)][1](menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 96, in get_order_input
    get_order(data, menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/main.py", line 78, in get_order
    total_price, total_quantity, quantity_dict = get_totals(full_order)  # Calculates quantity and cost totals.
TypeError: cannot unpack non-iterable NoneType object
[dante@archbox temp]$ 
```

Original Code:



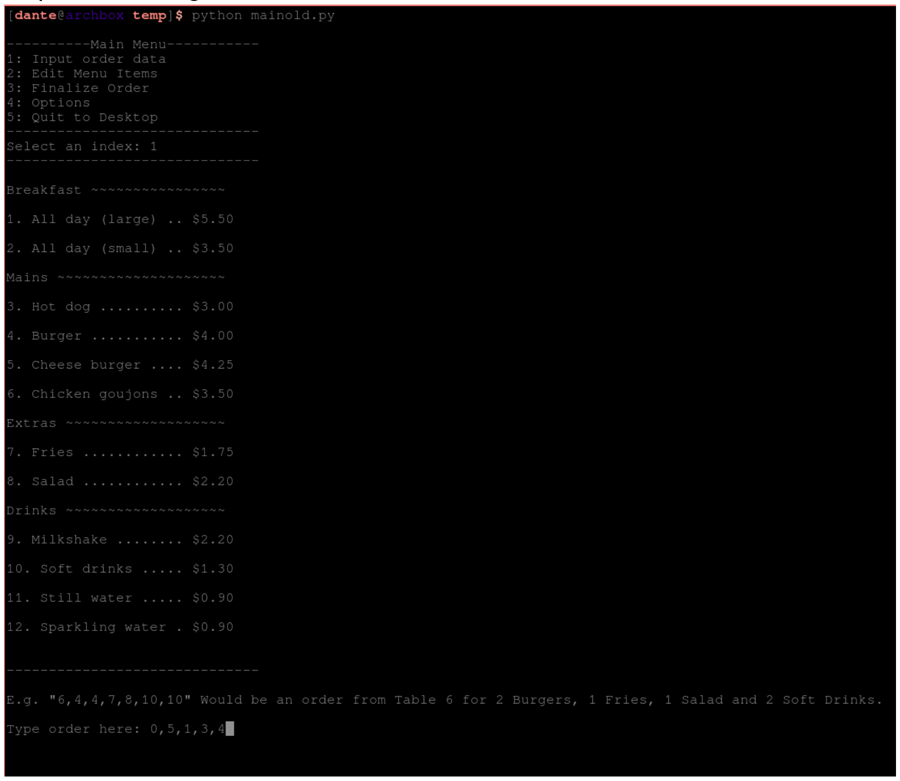
```python
def get_order_input(menu_file):  # Validates Order
    while True:
        data = input("\nType order here: ").split(",")
        tmp_last_item = menu_file[len(menu_file) - 1]  # Assigns Last entry's index
        last_item = tmp_last_item[0]                   # to "last_item"
        invalid = ""
        for element in data:  # Takes each element in the array
            if len(element) > len(last_item) or element.isnumeric() == False:
                invalid = True  # ^If the digit length is greater than the digit length in the menu or is not integer
            elif int(element) > int(last_item):  # If the element is numerically greater than the last item
                invalid = True
        if invalid:  # If there are letters or symbols in the input:
            print("Your order has invalid characters, please try again.")
        else:
            break
    get_order(data, menu_file)
```

Fixed Code:

```python
def get_order_input(menu_file):  # Validates Order
    while True:
        data = input("\nType order here: ").split(",")
        tmp_last_item = menu_file[len(menu_file) - 1]  # Assigns Last entry's index
        last_item = tmp_last_item[0]                    # to "last_item"
        invalid = ""
        invalidTable = ""
        for index, element in enumerate(data):  # Takes each element in the array
            if index == 0:  # If the element is first (table number doesn't apply for these rules)
                if element.isnumeric() == False:
                    invalid = True  # ^If the digit length is greater than 25 is not integer
                elif int(element) > 25:  # If the element is numerically greater than number of tables in (25)
                    invalidTable = True
            else:
                if len(element) > len(last_item) or element.isnumeric() == False:
                    invalid = True  # ^If the digit length is greater than the digit length in the menu or is not integer
                elif int(element) > int(last_item):  # If the element is numerically greater than the last item
                    invalid = True
        if invalidTable:
            print("We only have 25 tables! Table number must be lower than 25, please try again.")
        elif invalid:  # If there are letters or symbols in the input:
            print("Your order has invalid characters, please try again.")
        elif len(data) == 1:
            print("At least one order must be made per table, please try again.")
        else:
            break
    get_order(data, menu_file)
```

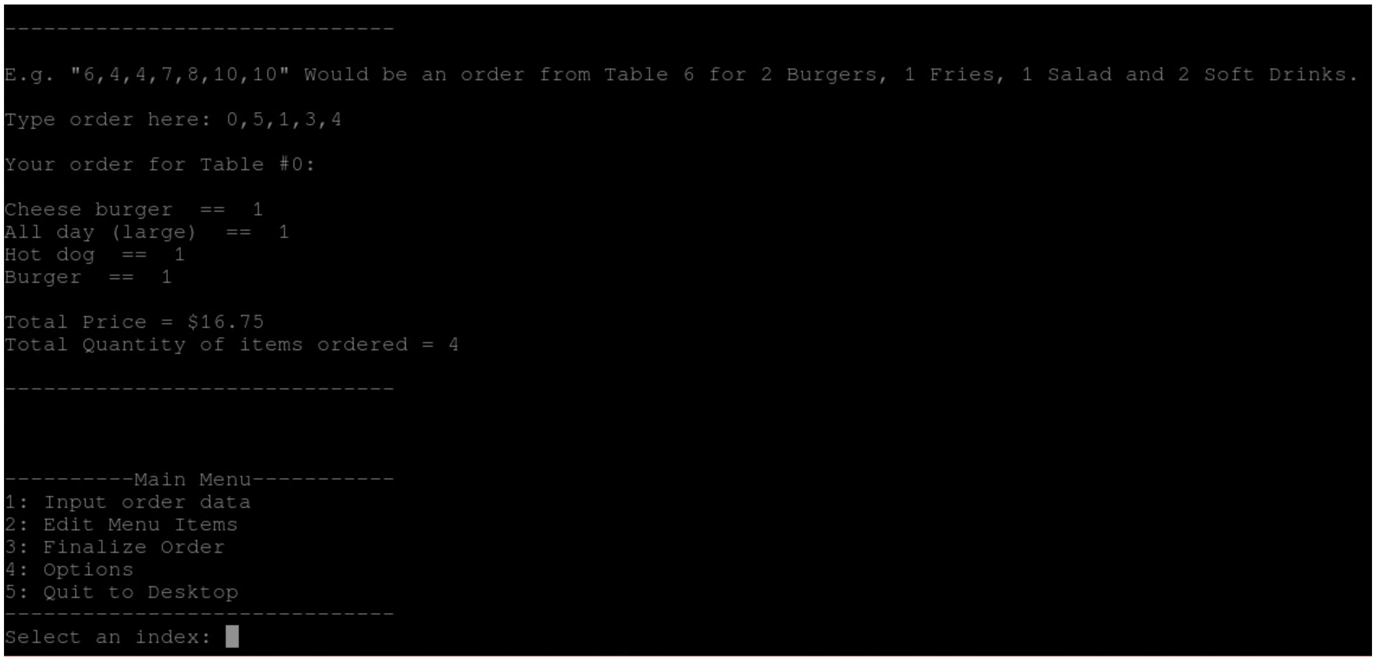
- How was the bug fixed?
  - The code will loop through each element in the user's input
  - The first element (the table number) will be checked to check to see:
    - IF: it is a numeric string
    - IF: The element is less than 25 *(at the time I was not aware that the maximum amount of tables was 10 and this was fixed later in the production of the program)*

- **Bug #2: Fixing Table number bug when entering "0" as a table number *(Logic Error)***
  - When entering the table number for the user's order as "0", the program would still accept it.
    - Example of entering"0" as the table number:



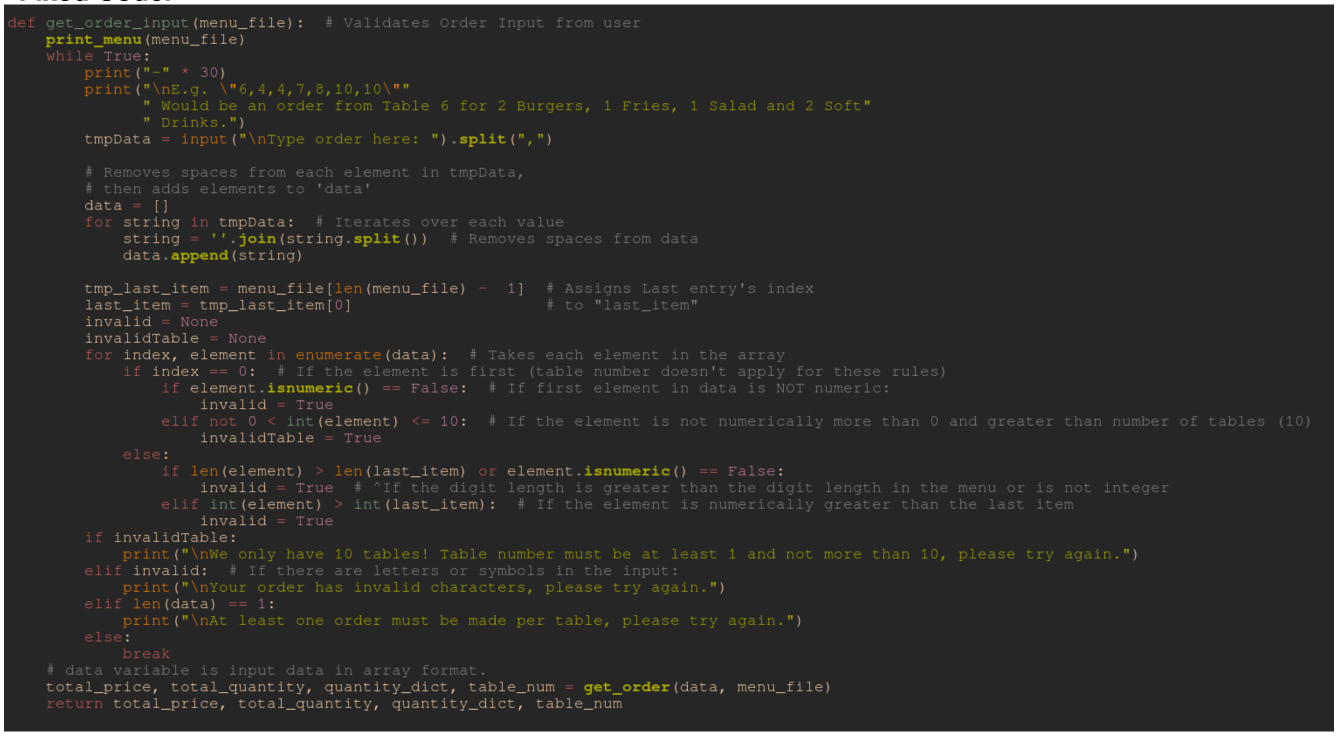
```
[dante@archbox temp]$ python mainold.py

----------Main Menu-----------
1: Input order data
2: Edit Menu Items
3: Finalize Order
4: Options
5: Quit to Desktop
------------------------------
Select an index: 1
------------------------------

Breakfast ~~~~~~~~~~~~~~~~

1. All day (large) .. $5.50

2. All day (small) .. $3.50

Mains ~~~~~~~~~~~~~~~~~~~~

3. Hot dog .......... $3.00

4. Burger ........... $4.00

5. Cheese burger .... $4.25

6. Chicken goujons .. $3.50

Extras ~~~~~~~~~~~~~~~~~~~

7. Fries ............ $1.75

8. Salad ............ $2.20

Drinks ~~~~~~~~~~~~~~~~~~~

9. Milkshake ........ $2.20

10. Soft drinks ..... $1.30

11. Still water ..... $0.90

12. Sparkling water . $0.90


------------------------------

E.g. "6,4,4,7,8,10,10" Would be an order from Table 6 for 2 Burgers, 1 Fries, 1 Salad and 2 Soft Drinks.

Type order here: 0,5,1,3,4
```

- The ouput shows order with the table number: “0”
- The program has accepted the table number when it should’ve rejected it instead because Table #0 doesn’t exist



```
------------------------------

E.g. "6,4,4,7,8,10,10" Would be an order from Table 6 for 2 Burgers, 1 Fries, 1 Salad and 2 Soft Drinks.

Type order here: 0,5,1,3,4

Your order for Table #0:

Cheese burger  ==  1
All day (large)  ==  1
Hot dog  ==  1
Burger  ==  1

Total Price = $16.75
Total Quantity of items ordered = 4

------------------------------



----------Main Menu-----------
1: Input order data
2: Edit Menu Items
3: Finalize Order
4: Options
5: Quit to Desktop
------------------------------
Select an index: 
```

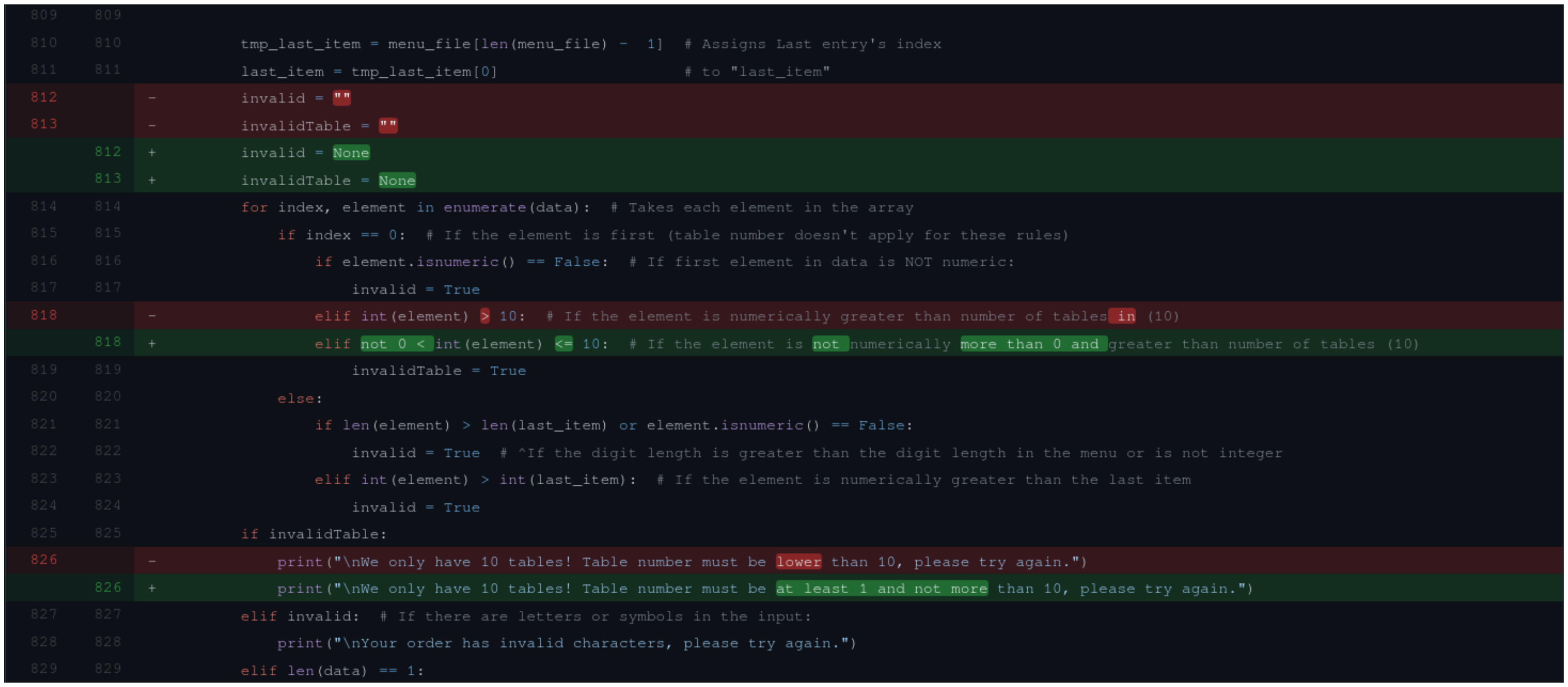
Fixed Code:



```python
def get_order_input(menu_file):  # Validates Order Input from user
    print_menu(menu_file)
    while True:
        print("-" * 30)
        print("\nE.g. \"6,4,4,7,8,10,10\""
              " Would be an order from Table 6 for 2 Burgers, 1 Fries, 1 Salad and 2 Soft"
              " Drinks.")
        tmpData = input("\nType order here: ").split(",")

        # Removes spaces from each element in tmpData,
        # then adds elements to 'data'
        data = []
        for string in tmpData:  # Iterates over each value
            string = ''.join(string.split())  # Removes spaces from data
            data.append(string)

        tmp_last_item = menu_file[len(menu_file) -  1]  # Assigns Last entry's index
        last_item = tmp_last_item[0]                    # to "last_item"
        invalid = None
        invalidTable = None
        for index, element in enumerate(data):  # Takes each element in the array
            if index == 0:  # If the element is first (table number doesn't apply for these rules)
                if element.isnumeric() == False:  # If first element in data is NOT numeric:
                    invalid = True
                elif not 0 < int(element) <= 10:  # If the element is not numerically more than 0 and greater than number of tables (10)
                    invalidTable = True
            else:
                if len(element) > len(last_item) or element.isnumeric() == False:
                    invalid = True  # ^If the digit length is greater than the digit length in the menu or is not integer
                elif int(element) > int(last_item):  # If the element is numerically greater than the last item
                    invalid = True
        if invalidTable:
            print("\nWe only have 10 tables! Table number must be at least 1 and not more than 10, please try again.")
        elif invalid:  # If there are letters or symbols in the input:
            print("\nYour order has invalid characters, please try again.")
        elif len(data) == 1:
            print("\nAt least one order must be made per table, please try again.")
        else:
            break
    # data variable is input data in array format.
    total_price, total_quantity, quantity_dict, table_num = get_order(data, menu_file)
    return total_price, total_quantity, quantity_dict, table_num
```

- How was the bug fixed?
  - The code will loop through each element in the user’s input
  - The first element (the table number) will be checked to check to see:
    - IF it’s greater than 0 and less than or equal to 10. (Tables)
  - Originally the code would just check if it was less than 10 which would allow “0” to go through

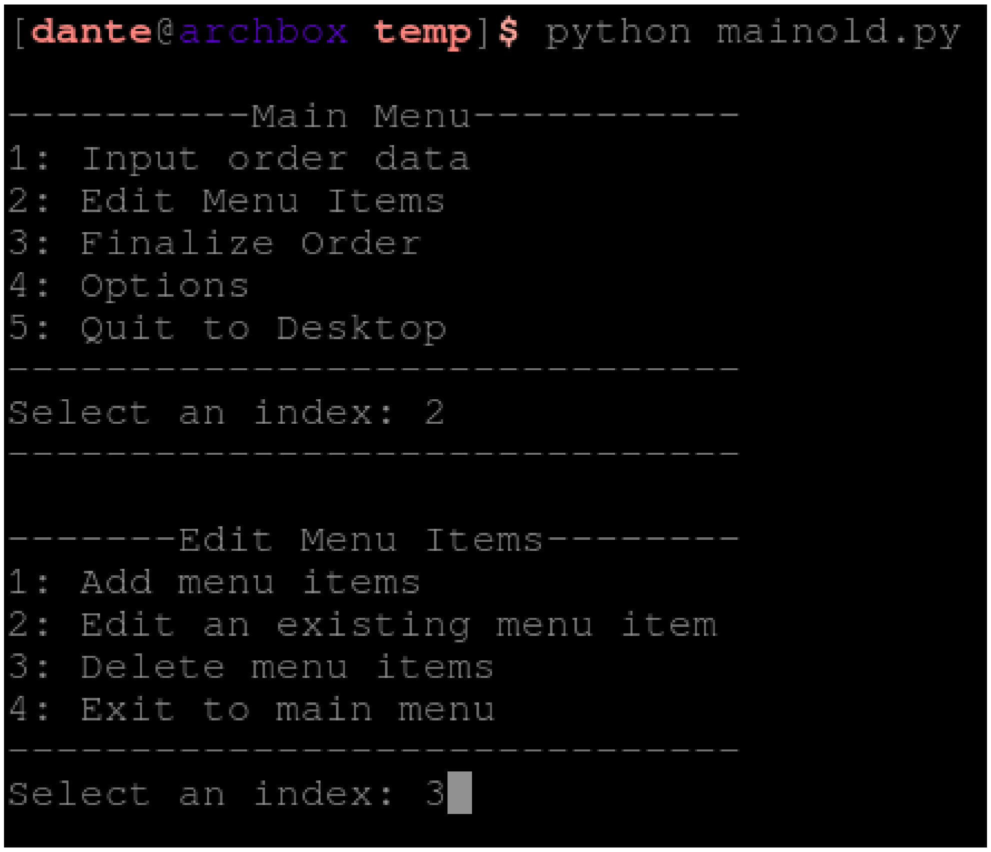
- **Key:**
  - ***Green** = Additions made in the program*
  - ***Red** = Deletions in the program*



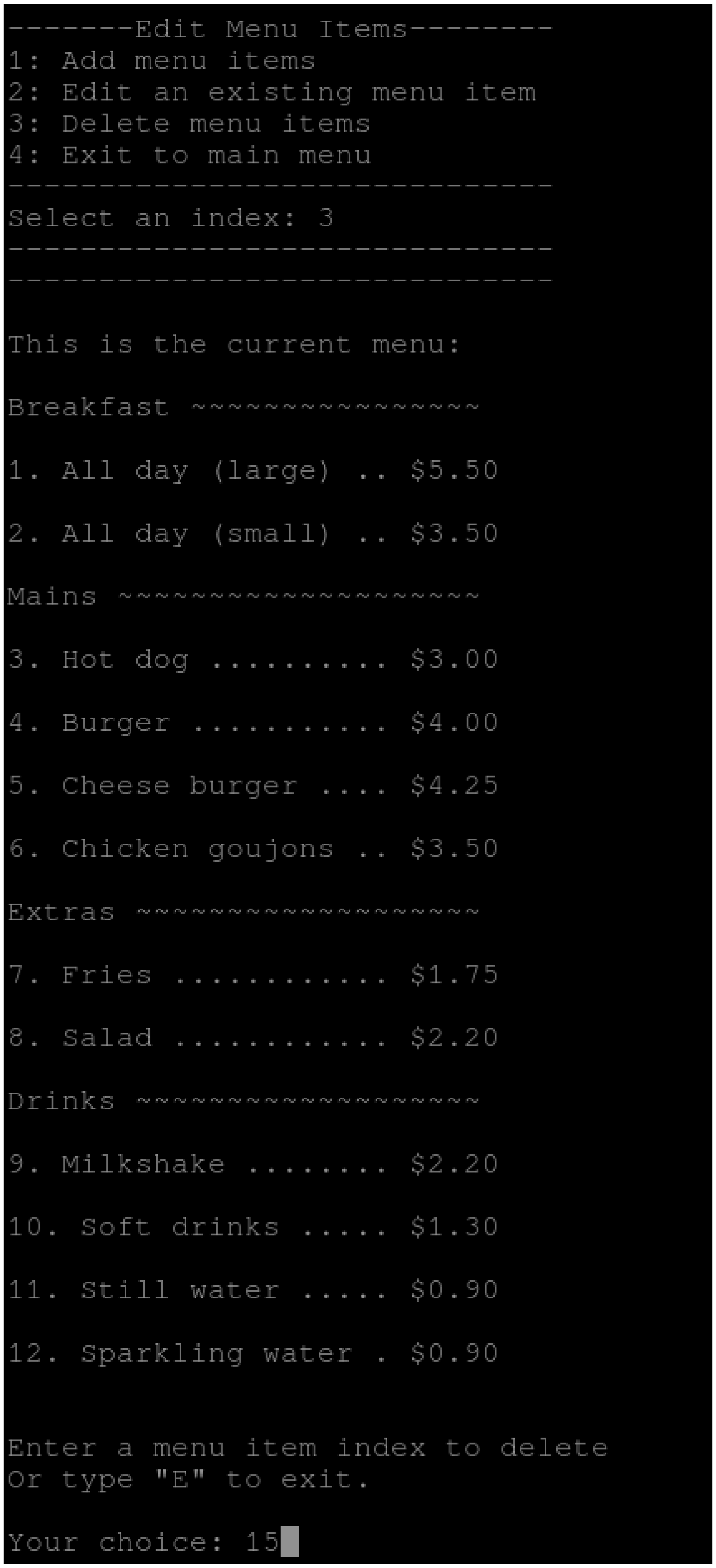
```diff
          tmp_last_item = menu_file[len(menu_file) - 1]  # Assigns Last entry's index
          last_item = tmp_last_item[0]                   # to "last_item"
-         invalid = ""
-         invalidTable = ""
+         invalid = None
+         invalidTable = None
          for index, element in enumerate(data):  # Takes each element in the array
              if index == 0:  # If the element is first (table number doesn't apply for these rules)
                  if element.isnumeric() == False:  # If first element in data is NOT numeric:
                      invalid = True
-                 elif int(element) > 10:  # If the element is numerically greater than number of tables in (10)
+                 elif not 0 < int(element) <= 10:  # If the element is not numerically more than 0 and greater than number of tables (10)
                      invalidTable = True
              else:
                  if len(element) > len(last_item) or element.isnumeric() == False:
                      invalid = True  # ^If the digit length is greater than the digit length in the menu or is not integer
                  elif int(element) > int(last_item):  # If the element is numerically greater than the last item
                      invalid = True
          if invalidTable:
-             print("\nWe only have 10 tables! Table number must be lower than 10, please try again.")
+             print("\nWe only have 10 tables! Table number must be at least 1 and not more than 10, please try again.")
          elif invalid:  # If there are letters or symbols in the input:
              print("\nYour order has invalid characters, please try again.")
          elif len(data) == 1:
```

- **Bug #3: Fixing Deletion of Menu Items Validation in delete_menu_items() *(Runtime Error)***
  - Upon entering a menu item index number that was greater than the maximum index number (out of range), the program would crash and raise **IndexError**
    - Examples: "" would raise **IndexError**, as well as entering "15"

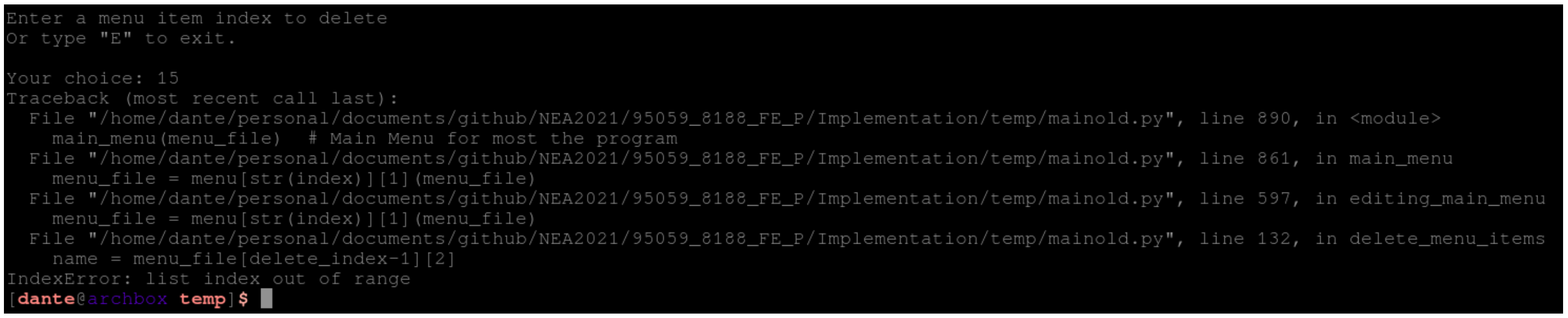
Program Running with Error Message:



```
[dante@archbox temp]$ python mainold.py

----------Main Menu-----------
1: Input order data
2: Edit Menu Items
3: Finalize Order
4: Options
5: Quit to Desktop
------------------------------
Select an index: 2
------------------------------


-------Edit Menu Items--------
1: Add menu items
2: Edit an existing menu item
3: Delete menu items
4: Exit to main menu
------------------------------
Select an index: 3
```

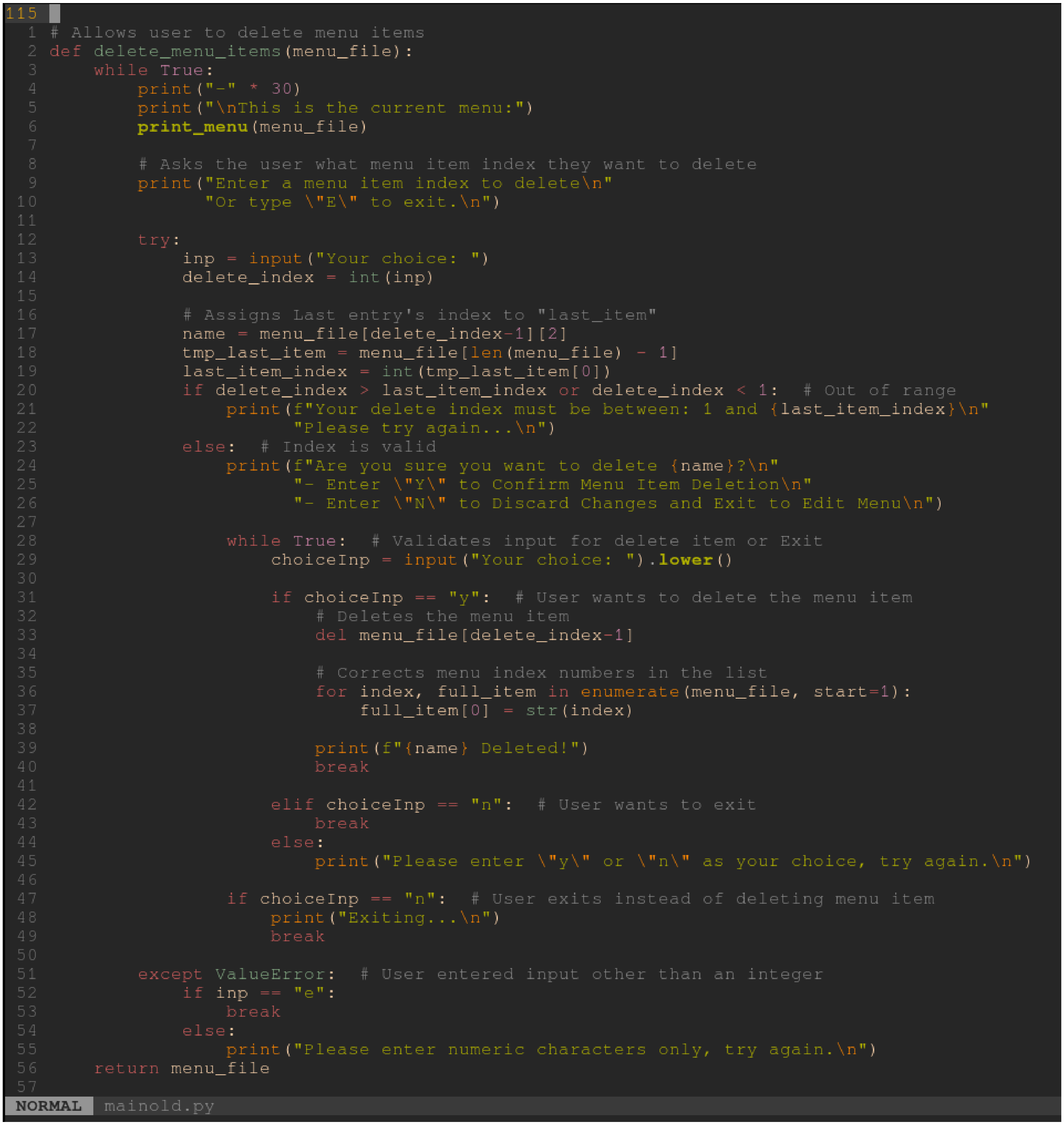


```
-------Edit Menu Items--------
1: Add menu items
2: Edit an existing menu item
3: Delete menu items
4: Exit to main menu
------------------------------
Select an index: 3
------------------------------
------------------------------

This is the current menu:

Breakfast ~~~~~~~~~~~~~~~~

1. All day (large) .. $5.50

2. All day (small) .. $3.50

Mains ~~~~~~~~~~~~~~~~~~~~

3. Hot dog .......... $3.00

4. Burger ........... $4.00

5. Cheese burger .... $4.25

6. Chicken goujons .. $3.50

Extras ~~~~~~~~~~~~~~~~~~~

7. Fries ............ $1.75

8. Salad ............ $2.20

Drinks ~~~~~~~~~~~~~~~~~~~

9. Milkshake ........ $2.20

10. Soft drinks ..... $1.30

11. Still water ..... $0.90

12. Sparkling water . $0.90


Enter a menu item index to delete
Or type "E" to exit.

Your choice: 15
```



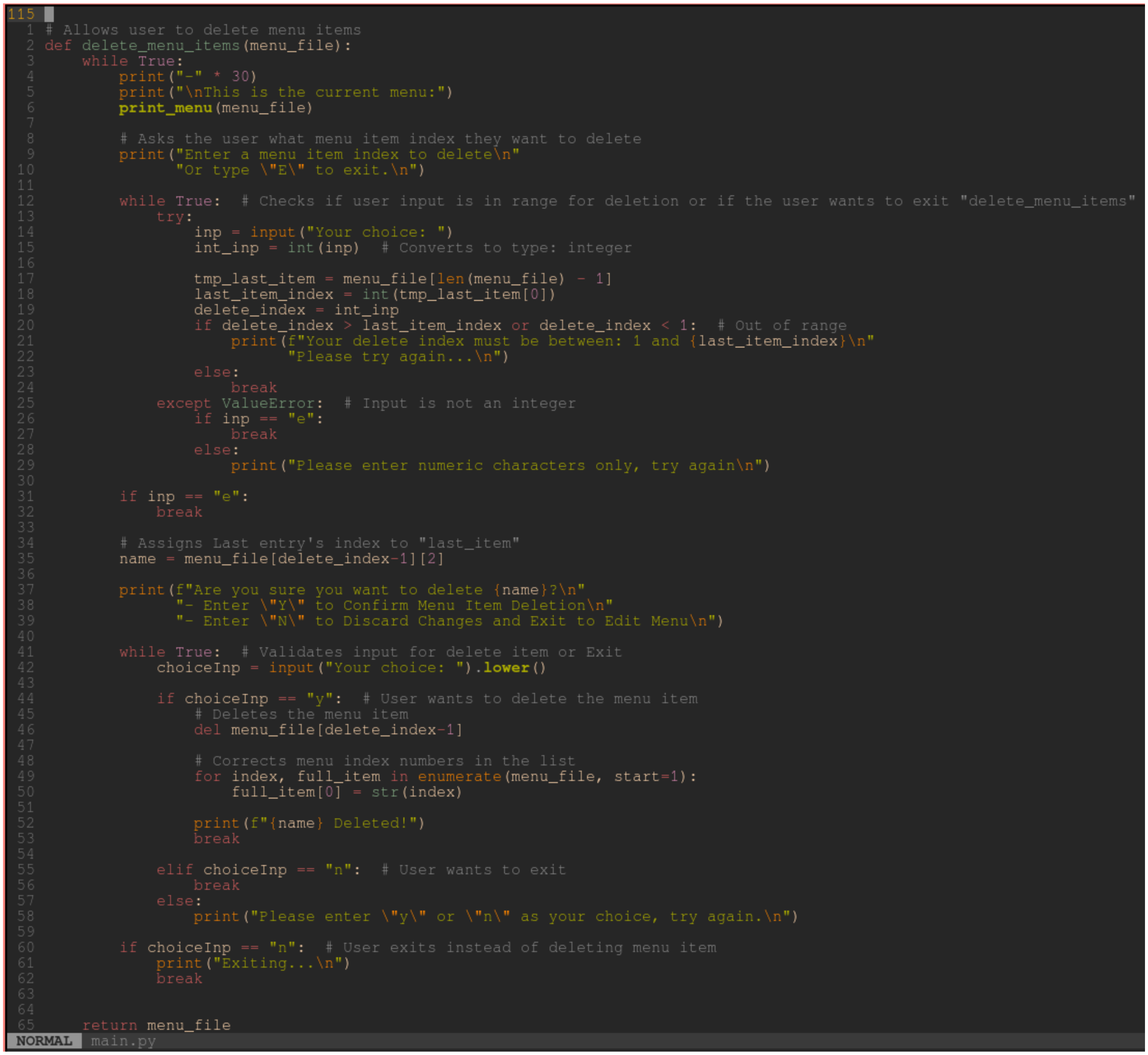
```
Enter a menu item index to delete
Or type "E" to exit.

Your choice: 15
Traceback (most recent call last):
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/mainold.py", line 890, in <module>
    main_menu(menu_file)  # Main Menu for most the program
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/mainold.py", line 861, in main_menu
    menu_file = menu[str(index)][1](menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/mainold.py", line 597, in editing_main_menu
    menu_file = menu[str(index)][1](menu_file)
  File "/home/dante/personal/documents/github/NEA2021/95059_8188_FE_P/Implementation/temp/mainold.py", line 132, in delete_menu_items
    name = menu_file[delete_index-1][2]
IndexError: list index out of range
[dante@archbox temp]$
```

Original Code:



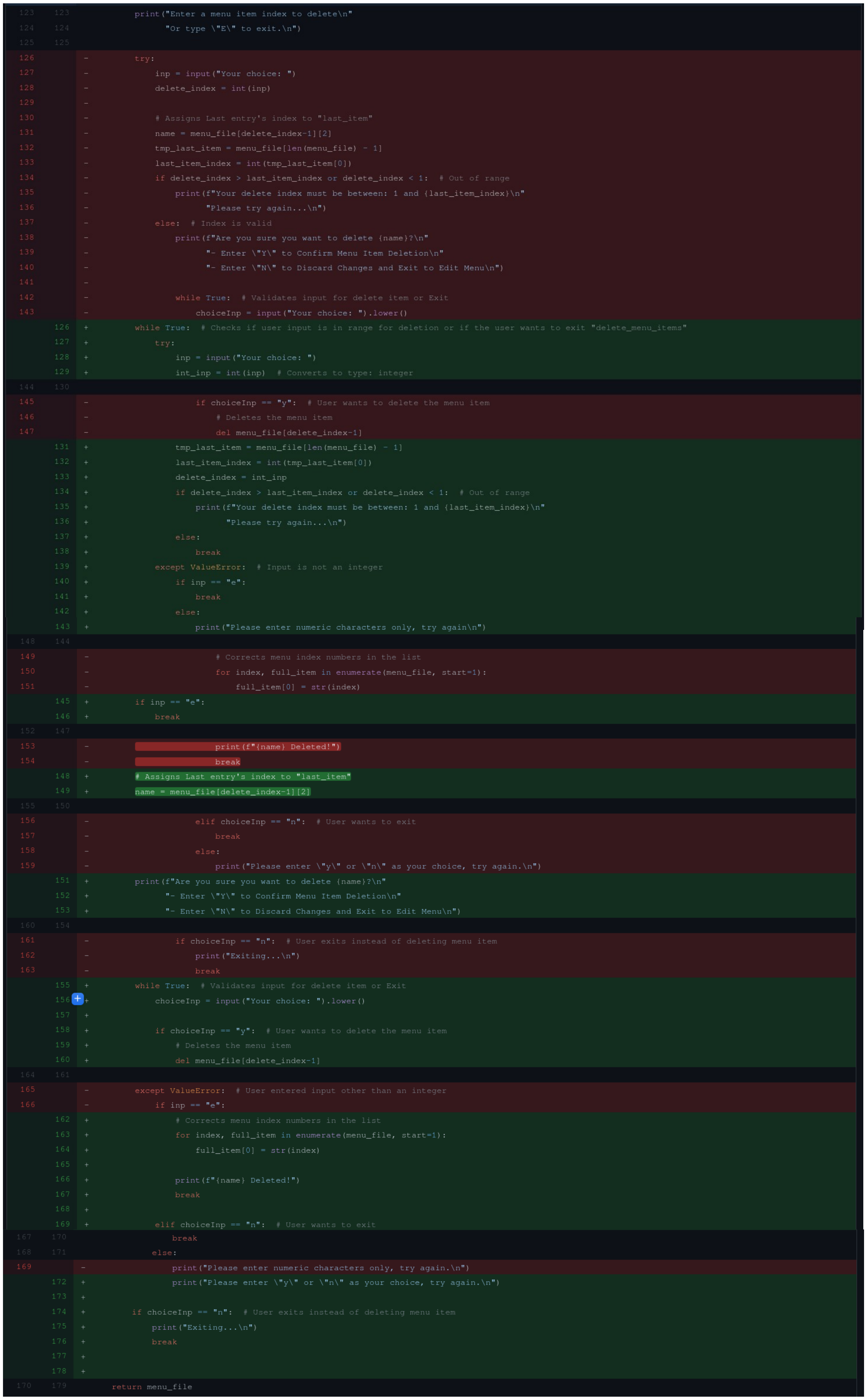
```python
115
# Allows user to delete menu items
def delete_menu_items(menu_file):
    while True:
        print("-" * 30)
        print("\nThis is the current menu:")
        print_menu(menu_file)

        # Asks the user what menu item index they want to delete
        print("Enter a menu item index to delete\n"
              "Or type \"E\" to exit.\n")

        try:
            inp = input("Your choice: ")
            delete_index = int(inp)

            # Assigns Last entry's index to "last_item"
            name = menu_file[delete_index-1][2]
            tmp_last_item = menu_file[len(menu_file) - 1]
            last_item_index = int(tmp_last_item[0])
            if delete_index > last_item_index or delete_index < 1:  # Out of range
                print(f"Your delete index must be between: 1 and {last_item_index}\n"
                      "Please try again...\n")
            else:  # Index is valid
                print(f"Are you sure you want to delete {name}?\n"
                      "- Enter \"Y\" to Confirm Menu Item Deletion\n"
                      "- Enter \"N\" to Discard Changes and Exit to Edit Menu\n")

                while True:  # Validates input for delete item or Exit
                    choiceInp = input("Your choice: ").lower()

                    if choiceInp == "y":  # User wants to delete the menu item
                        # Deletes the menu item
                        del menu_file[delete_index-1]

                        # Corrects menu index numbers in the list
                        for index, full_item in enumerate(menu_file, start=1):
                            full_item[0] = str(index)

                        print(f"{name} Deleted!")
                        break

                    elif choiceInp == "n":  # User wants to exit
                        break
                    else:
                        print("Please enter \"y\" or \"n\" as your choice, try again.\n")

                if choiceInp == "n":  # User exits instead of deleting menu item
                    print("Exiting...\n")
                    break

        except ValueError:  # User entered input other than an integer
            if inp == "e":
                break
            else:
                print("Please enter numeric characters only, try again.\n")
    return menu_file

NORMAL  mainold.py
```

Fixed Code:



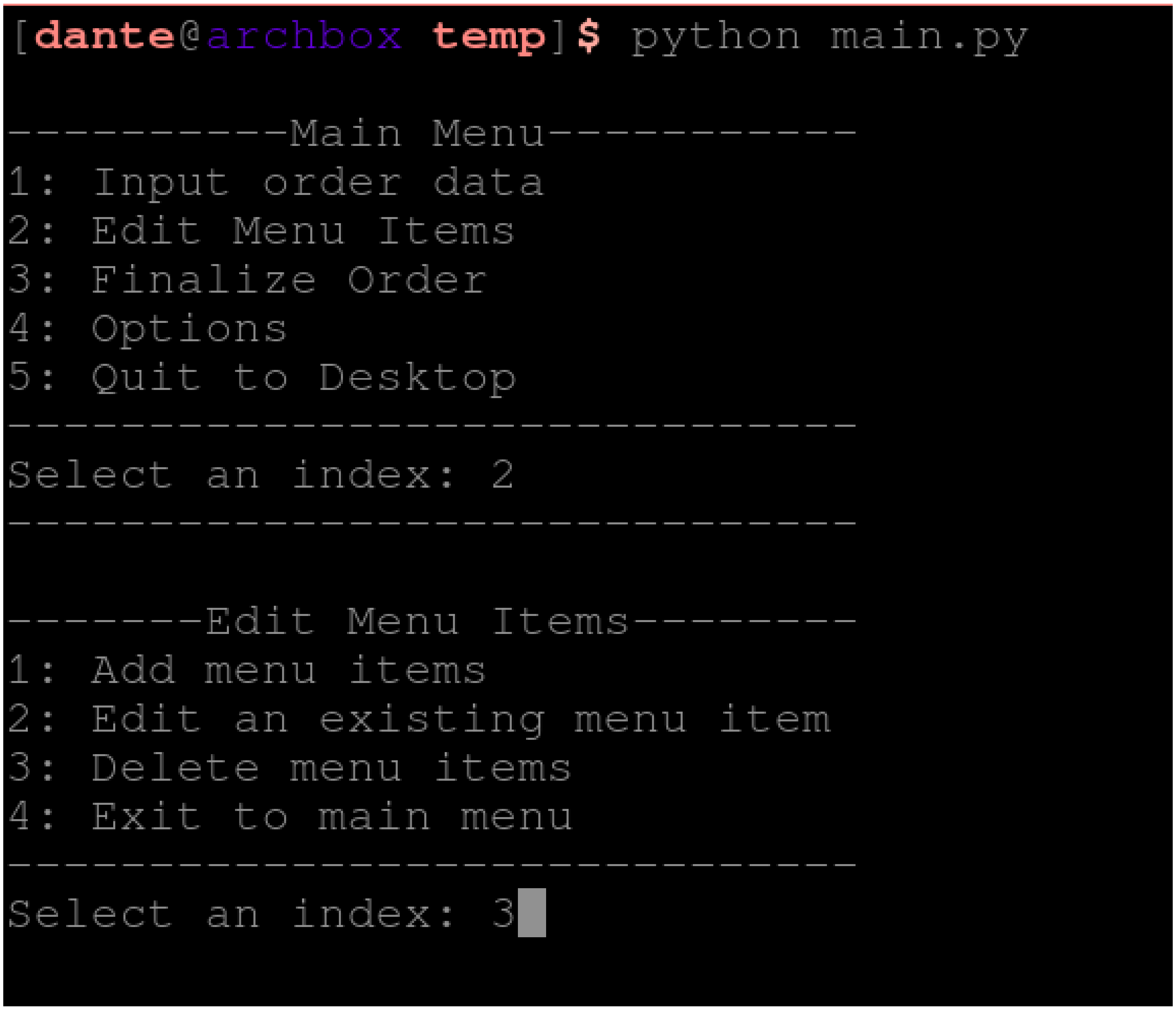
```python
# Allows user to delete menu items
def delete_menu_items(menu_file):
    while True:
        print("-" * 30)
        print("\nThis is the current menu:")
        print_menu(menu_file)

        # Asks the user what menu item index they want to delete
        print("Enter a menu item index to delete\n"
              "Or type \"E\" to exit.\n")

        while True:  # Checks if user input is in range for deletion or if the user wants to exit "delete_menu_items"
            try:
                inp = input("Your choice: ")
                int_inp = int(inp)  # Converts to type: integer

                tmp_last_item = menu_file[len(menu_file) - 1]
                last_item_index = int(tmp_last_item[0])
                delete_index = int_inp
                if delete_index > last_item_index or delete_index < 1:  # Out of range
                    print(f"Your delete index must be between: 1 and {last_item_index}\n"
                          "Please try again...\n")
                else:
                    break
            except ValueError:  # Input is not an integer
                if inp == "e":
                    break
                else:
                    print("Please enter numeric characters only, try again\n")

        if inp == "e":
            break

        # Assigns Last entry's index to "last_item"
        name = menu_file[delete_index-1][2]

        print(f"Are you sure you want to delete {name}?\n"
              "- Enter \"Y\" to Confirm Menu Item Deletion\n"
              "- Enter \"N\" to Discard Changes and Exit to Edit Menu\n")

        while True:  # Validates input for delete item or Exit
            choiceInp = input("Your choice: ").lower()

            if choiceInp == "y":  # User wants to delete the menu item
                # Deletes the menu item
                del menu_file[delete_index-1]

                # Corrects menu index numbers in the list
                for index, full_item in enumerate(menu_file, start=1):
                    full_item[0] = str(index)

                print(f"{name} Deleted!")
                break

            elif choiceInp == "n":  # User wants to exit
                break
            else:
                print("Please enter \"y\" or \"n\" as your choice, try again.\n")

        if choiceInp == "n":  # User exits instead of deleting menu item
            print("Exiting...\n")
            break

    return menu_file
```

- How was the bug fixed?
  - The program will initialize the variable *last_item_index* (the menu item in the menu with the greatest index number)
  - Then the program will evaluate if the user's input is greater than value of *last_item_index.*
    - IF the delete_index (deletion index that the user entered) is greater than the *last_item_index* less than 1 THEN: ***Ask to enter another value again.***
  - *Originally the code would raise* ***IndexError*** *upon the user entering an invalid index because the user specified index doesn't exist.*
  - **Key:**
    - ***Green*** *= Additions made in the program*
    - ***Red*** *= Deletions in the program*

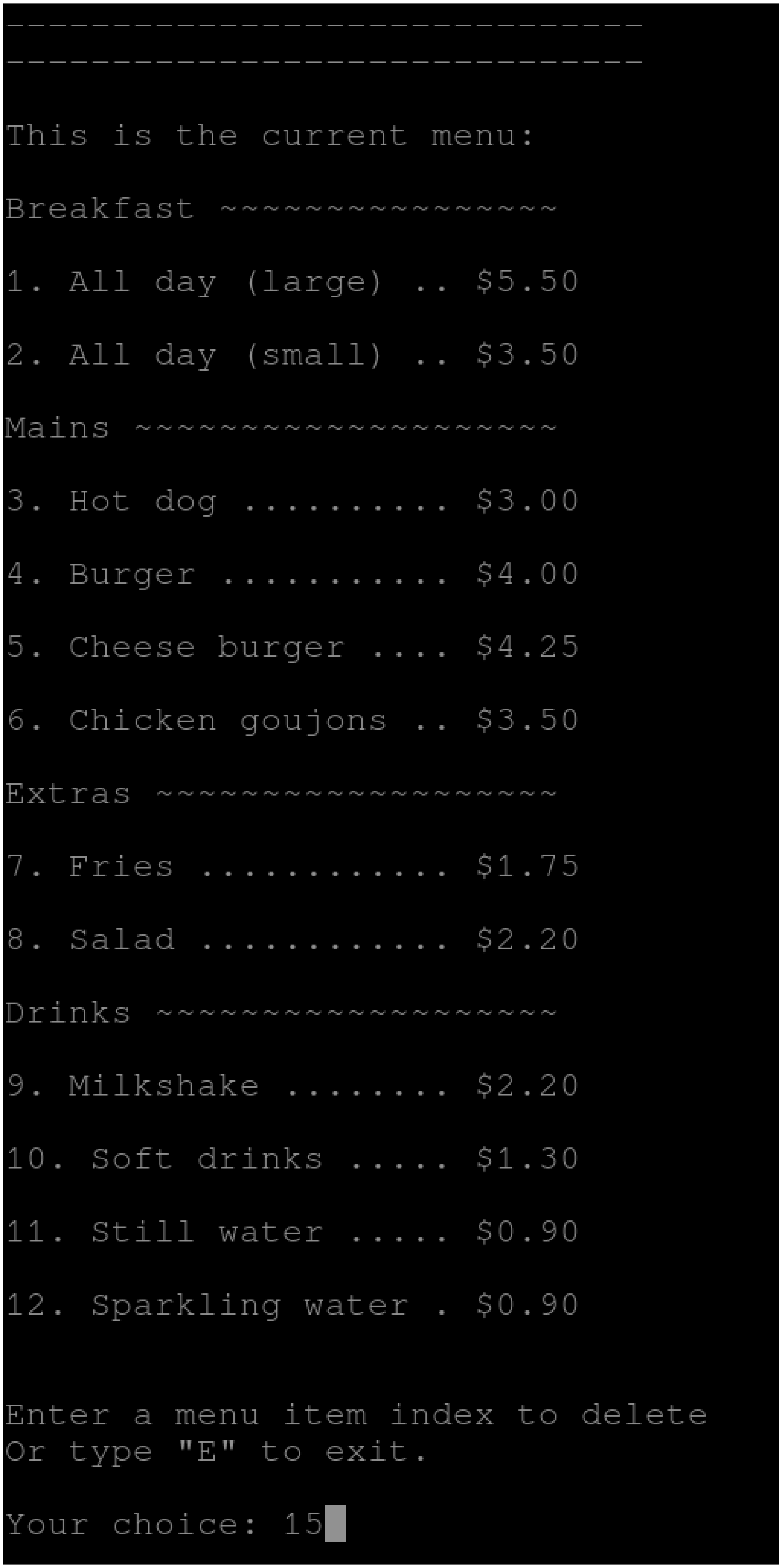


```diff
          print("Enter a menu item index to delete\n"
                "Or type \"E\" to exit.\n")
 
-         try:
-             inp = input("Your choice: ")
-             delete_index = int(inp)
-
-             # Assigns Last entry's index to "last_item"
-             name = menu_file[delete_index-1][2]
-             tmp_last_item = menu_file[len(menu_file) - 1]
-             last_item_index = int(tmp_last_item[0])
-             if delete_index > last_item_index or delete_index < 1:  # Out of range
-                 print(f"Your delete index must be between: 1 and {last_item_index}\n"
-                       "Please try again...\n")
-             else:  # Index is valid
-                 print(f"Are you sure you want to delete {name}?\n"
-                       "- Enter \"Y\" to Confirm Menu Item Deletion\n"
-                       "- Enter \"N\" to Discard Changes and Exit to Edit Menu\n")
-
-                 while True:  # Validates input for delete item or Exit
-                     choiceInp = input("Your choice: ").lower()
+         while True:  # Checks if user input is in range for deletion or if the user wants to exit "delete_menu_items"
+             try:
+                 inp = input("Your choice: ")
+                 int_inp = int(inp)  # Converts to type: integer
 
-                     if choiceInp == "y":  # User wants to delete the menu item
-                         # Deletes the menu item
-                         del menu_file[delete_index-1]
+                 tmp_last_item = menu_file[len(menu_file) - 1]
+                 last_item_index = int(tmp_last_item[0])
+                 delete_index = int_inp
+                 if delete_index > last_item_index or delete_index < 1:  # Out of range
+                     print(f"Your delete index must be between: 1 and {last_item_index}\n"
+                           "Please try again...\n")
+                 else:
+                     break
+             except ValueError:  # Input is not an integer
+                 if inp == "e":
+                     break
+                 else:
+                     print("Please enter numeric characters only, try again\n")
 
-                         # Corrects menu index numbers in the list
-                         for index, full_item in enumerate(menu_file, start=1):
-                             full_item[0] = str(index)
+         if inp == "e":
+             break
 
-                         print(f"{name} Deleted!")
-                         break
+         # Assigns Last entry's index to "last_item"
+         name = menu_file[delete_index-1][2]
 
-                     elif choiceInp == "n":  # User wants to exit
-                         break
-                     else:
-                         print("Please enter \"y\" or \"n\" as your choice, try again.\n")
+         print(f"Are you sure you want to delete {name}?\n"
+               "- Enter \"Y\" to Confirm Menu Item Deletion\n"
+               "- Enter \"N\" to Discard Changes and Exit to Edit Menu\n")
 
-                 if choiceInp == "n":  # User exits instead of deleting menu item
-                     print("Exiting...\n")
-                     break
+         while True:  # Validates input for delete item or Exit
+             choiceInp = input("Your choice: ").lower()
+
+             if choiceInp == "y":  # User wants to delete the menu item
+                 # Deletes the menu item
+                 del menu_file[delete_index-1]
 
-         except ValueError:  # User entered input other than an integer
-             if inp == "e":
+                 # Corrects menu index numbers in the list
+                 for index, full_item in enumerate(menu_file, start=1):
+                     full_item[0] = str(index)
+
+                 print(f"{name} Deleted!")
+                 break
+
+             elif choiceInp == "n":  # User wants to exit
                  break
              else:
-                 print("Please enter numeric characters only, try again.\n")
+                 print("Please enter \"y\" or \"n\" as your choice, try again.\n")
+
+         if choiceInp == "n":  # User exits instead of deleting menu item
+             print("Exiting...\n")
+             break
+
+
      return menu_file
```

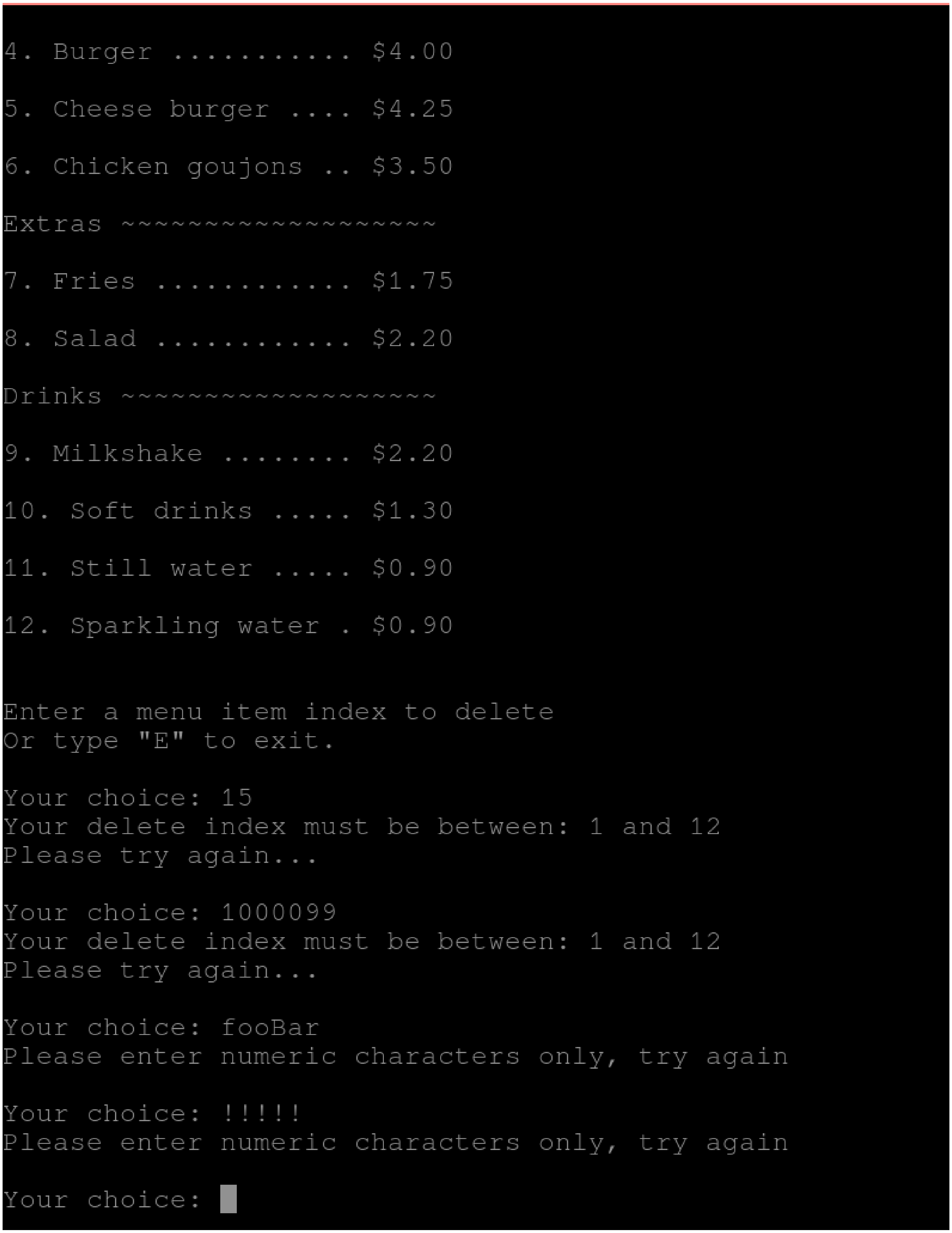
Fixed Code Running without Errors:



```
[dante@archbox temp]$ python main.py

----------Main Menu-----------
1: Input order data
2: Edit Menu Items
3: Finalize Order
4: Options
5: Quit to Desktop
------------------------------
Select an index: 2
------------------------------


-------Edit Menu Items--------
1: Add menu items
2: Edit an existing menu item
3: Delete menu items
4: Exit to main menu
------------------------------
Select an index: 3
```



```
-----------------------------
-----------------------------

This is the current menu:

Breakfast ~~~~~~~~~~~~~~~~

1. All day (large) .. $5.50

2. All day (small) .. $3.50

Mains ~~~~~~~~~~~~~~~~~~~~

3. Hot dog .......... $3.00

4. Burger ........... $4.00

5. Cheese burger .... $4.25

6. Chicken goujons .. $3.50

Extras ~~~~~~~~~~~~~~~~~~~

7. Fries ............ $1.75

8. Salad ............ $2.20

Drinks ~~~~~~~~~~~~~~~~~~~

9. Milkshake ........ $2.20

10. Soft drinks ..... $1.30

11. Still water ..... $0.90

12. Sparkling water . $0.90


Enter a menu item index to delete
Or type "E" to exit.

Your choice: 15
```


```
4. Burger ........... $4.00

5. Cheese burger .... $4.25

6. Chicken goujons .. $3.50

Extras ~~~~~~~~~~~~~~~~~~~

7. Fries ............ $1.75

8. Salad ............ $2.20

Drinks ~~~~~~~~~~~~~~~~~~~

9. Milkshake ........ $2.20

10. Soft drinks ..... $1.30

11. Still water ..... $0.90

12. Sparkling water . $0.90


Enter a menu item index to delete
Or type "E" to exit.

Your choice: 15
Your delete index must be between: 1 and 12
Please try again...

Your choice: 1000099
Your delete index must be between: 1 and 12
Please try again...

Your choice: fooBar
Please enter numeric characters only, try again

Your choice: !!!!!
Please enter numeric characters only, try again

Your choice: 
```

Sources Used

1. https://stackoverflow.com/questions/18667410/how-can-i-check-if-a-string-only-contains-letters-in-python
- 11th Feb 2020

2. https://stackoverflow.com/questions/22524635/can-you-define-a-variable-inside-an-if-statement
- 11th Feb 2020

https://www.w3schools.com/python/ref_string_strip.asp
- 11th Feb 2020

https://stackoverflow.com/questions/23240969/python-count-repeated-elements-in-the-list
- 12th Feb 2020

Evaluation

- **Evaluation of Key Requirements of the Program *(Previously stated in Analysis)***

  - User should be able to input order details with table number and a string of numbers corresponding to menu items chosen
    - ✓ When the user wants to enter an order, the program presents the user with a menu that is able to display a variety of menu items with their respective prices and what categories they are in
    - ✓ The user can then enter an input that contains a list of menu item index number that are separated by commas
    - ✓ The program smart enough to accept input with any amount of spaces between the commas and will reject any input that isn't valid that has invalid characters
    - ✓ The input is validated with a wide range of conditions that are built to reject any type of errors that the user can make when entering input
    - ✓ Allows for a single menu order to be entered and processed in a single line for efficiency
    - ✓ Takes the table number, name of items and how much of each item
    - ✓ This part of the program was built to have thorough validation checks so that the end user could not easily break and crash the program and thus not being user friendly
    - ✓ The program exceeds the expectations of the this requirement
    - ✘ In my opinion, this method of entering menu input is efficient, however unintuitive and could be hard to understand for the end user.

  - Program must be able to validate input data
    - ✓ For every place in the program where the user can enter an input, there has been thorough validation checks
    - ✓ This is so that the user cannot easily break the program and lose their progress and have to start over again
    - ✓ This makes it user friendly and easy to use
    - ✓ Debugging of the validation checks can be viewed in the Debugging file
    - ✓ Exceeds the expectations of the requirements
  - Display the order details for printing
    - ✓ If the user tries to finalize the order and print their order for printing without having ordered anything in the first place, the program will prevent them from doing so
      - ➔ This is so that the user cannot break the program and this ensures that proper information is printed out when finalizing their order
    - ✓ The program will print out a formatted version of the user's final order
    - ✓ It will display how many of each item has been ordered as well as the price totals and the quantity totals of menu items ordered
    - ✓ It will also display what Table ordered the items
    - ✓ At the bottom of the receipt, the program asks the user whether they would like to exit and enter another order, exit and keep current order, or just quit the program
      - ➔ This is so that the user can easily navigate around the program and doesn't have to quit program and execute it again if they want to enter another order
        - ➔ User friendly
    - ✓ Exceeds the expectations of this requirement
  - Loop for next order
    - ✓ After the user has finalized their order, the user can then choose to exit to the main menu and access all of the other features of the program including being able to loop for another order
    - ✓ Allowing the user to exit after finalizing the program and putting them back into the main menu for them to order another item is intuitive and user friendly
    - ✓ This makes it easier for the user to use the program and makes it less difficult to use
    - ✓ The program has accomplished this effectively

- Allow menu to be saved to a file
    - ✓ Everytime the user makes any changes to the menu whether it be editing an element of a menu item (e.g. price, name, or index number), deleting a menu item or adding a menu item, the program will always save the menu file to a file called menu.txt
    - ✓ The program saves it as a menu.txt file in the same directory as main.py
        - ➔ menu.txt contains information about each menu item on each line
        - ➔ Each line is a single menu item
        - ➔ The menu items are in the format of comma separated values
            1. **The first value [0] on a line is the menu index number**
            2. **The second value [1] is the price**
            3. **The third value [2] is the name of the item**
            4. **The fourth value [3] is the category for the item**
    - ✓ ***Upon running main.py for the first time, the program will automatically create menu.txt if it hasn't already been created, and the running_totals.txt as well***
        - ➔ This is so that the program is portable and can be used anywhere
        - ➔ If the user doesn't have the menu file it's not a problem because the program will just generate it itself
    - ✓ <u>The program has accomplished this effectively</u>
- Allow for user to amend, add, delete menu items as well as save menu changes
    - ✓ The user can go into the editing main menu and choose to Add, Edit or Delete menu items
    - ✓ The user can select these menu options where they will be displayed a menu with options based on their selection
        - ➔ Adding a menu item allows the user to add a category and is validated to make sure the input is valid
            - ➔ It checks to make sure the name only contains letters and the first letter of the menu item name is capitalized
            - ➔ Also checks to make sure that the price is a float only.
            - ➔ The price is then displayed to 2 decimal places and is rounded
            - ➔ If price's last digit is a zero, the menu will still display it to 2 decimal places like a real menu
                - ➔ **E.g. "$2.5" or "$2.50" would still display as $2.50**
        - ➔ Editing a menu item:
            - ➔ Allows the user to edit the Index, Name, Price and Category
            - ➔ If the user changes the Index the category will change as well (same applies for the category)
    - ✓ <u>Exceeds the expectations of the requirements</u>
- Maintain running total of order values
    - ✓ Once the program takes in the user's input, it stores it in a variable
    - ✓ <u>Meets the expectations of the requirements</u>
- Maintain running totals of the quantity
    - ✓ Stores it in a file as well like running total
    - ✓ <u>Meets the expectations of the requirements</u>
- Save running totals to a file
    - ✓ <u>Meets the expectations of the requirements</u>
- Provide options to display the menu and running totals
    - ✓ Whenever the use wants to order something, the menu is printed on the screen for the user to see
    - ✓ The menu items all have categories with prices and names displayed in order
    - ✓ The menu displayed with the categories makes it easy to view
        - ➔ I also progammed the menu myself so that based on how long the menu item with the most characters calculates the right amount of periods for each menu item
    - ✓ <u>Exceeds the expectations of the requirements</u>

```
# PSEUDO CODE FOR TIMS DINER PROGRAM
# github.com/dantefernando/NEA2021


# DEFAULT_MENU is used in check_file() upon running program for first
time.

SET DEFAULT_MENU TO ["1,5.50,All day (large),breakfast",
                     "2,3.50,All day (small),breakfast",
                     "3,3.00,Hot dog,mains",
                     "4,4.00,Burger,mains",
                     "5,4.25,Cheese burger,mains",
                     "6,3.50,Chicken goujons,mains",
                     "7,1.75,Fries,extras",
                     "8,2.20,Salad,extras",
                     "9,2.20,Milkshake,drinks",
                     "10,1.30,Soft drinks,drinks",
                     "11,0.90,Still water,drinks",
                     "12,0.90,Sparkling water,drinks"]


FUNCTION finalize_order (total_price, total_quantity, quantity_dict,
table_num, hasData)
BEGIN FUNCTION
    SEND "\n######## BEGIN PRINTING ########\n" TO DISPLAY
    SEND "TIMS DINER\n" TO DISPLAY
    SEND "\nFinal order:" TO DISPLAY
    display_order(total_price, total_quantity, quantity_dict, table_num)

    WHILE True DO
        SEND "- Enter \"Y\" to Exit and enter another order\n"
              "- Enter \"N\" to Exit and keep current order\n"
              "- Enter \"Q\" to Quit the program\n" TO DISPLAY
        SEND "Your choice: " TO DISPLAY
        RECEIVE inp (string) keyboard
        SET inp TO inp.lower()

        IF inp = "y" THEN  # user exits and enters another order

            # Resets all order values
            SET total_price TO None
            SET total_quantity TO None
            SET quantity_dict TO None
            SET table_num TO None
            SET hasData TO False

            RETURN total_price, total_quantity, quantity_dict, table_num,
hasData

        ELIF inp = "n" DO  # User exits and keeps current order
            RETURN total_price, total_quantity, quantity_dict, table_num,
hasData

        ELIF inp = "q" DO  # User quits the program
```

```
            quit()

        ELSE  # User input is invalid
            SEND "Please enter \"y\" or \"n\" as your choice, try
again.\n" TO DISPLAY
        ENDIF
    END WHILE
END FUNCTION


FUNCTION options (menu_file)  # Options menu for the whole program
BEGIN FUNCTION
    SEND "\n-------Options Menu--------" TO DISPLAY
    SEND "1: Reset Menu Items File\n"
         "2: Exit to Main Menu\n" TO DISPLAY
    WHILE True DO  # Validates user input choice
        TRY DO
            SEND "Your choice: " TO DISPLAY
            RECEIVE inp FROM keyboard (integer)
            IF inp = 1 THEN  # inp=1 | User attempts reset of file
                SEND "Are you sure you want to reset the Menu Items File?
(menu.txt)?\n"
                      "This will erase all saved changes to the file and
reset it to\n"
                      "defaults.\n\n"
                      "- Enter \"Y\" to CONFIRM MENU RESET\n"
                      "- Enter \"N\" to DISCARD CHANGES AND EXIT\n" TO
DISPLAY

                WHILE True DO  # Validates input for Reset or Exit
                    SEND "Your choice: " TO DISPLAY
                    RECEIVE choiceInp FROM (string) keyboard
                    SET choiceInp TO choiceInp.lower()

                    IF choiceInp = "y" THEN  # User wants to reset
                        WITH open("menu.txt", "w") AS file DO  # Creates
the file and writes default menu
                            FOR EACH line IN DEFAULT_MENU DO
                                file.write(f"{line}\n")
                            END FOR
                        END WITH
                        WITH open("menu.txt", "r") as file DO
                            SET whole_file TO file.readlines()  # Stores
the menu in the file as "whole_file"
                            SET menu_file TO []
                            FOR EACH element IN whole_file DO
                                SET element TO element.strip("\n")
                                SET element TO element.split(",")
                                menu_file.append(element)
                            END FOR
                        END WITH

                        SEND "File reset!\n" TO DISPLAY
                        RETURN menu_file
                    ELIF choiceInp = "n" DO  # User wants to exit
                        BREAK
                    ELSE
                        SEND "Please enter \"y\" or \"n\" as your choice,
try again.\n" TO DISPLAY
                    ENDIF
                END WHILE
                IF choiceInp = "n" THEN  # User exits instead of resetting
                    SEND "Exiting...\n" TO DISPLAY
                    BREAK
                ENDIF
            ELIF inp = 2 DO  # inp==2 | User Exits
                SEND "Exiting...\n" TO DISPLAY
                BREAK
            ELSE  # Not valid
                SEND "Please enter 1 or 2 as your choice.\n" TO DISPLAY
            ENDIF
        EXCEPT ValueError DO  # User doesn't input numeric characters
            SEND "Please enter numeric characters only, try again.\n" TO
DISPLAY
        END TRY
    RETURN menu_file
    END WHILE
END FUNCTION


# Allows user to delete menu items
FUNCTION delete_menu_items (menu_file)
BEGIN FUNCTION
    WHILE True DO
        SEND "-" * 30 TO DISPLAY
        SEND "\nThis is the current menu:" TO DISPLAY
        print_menu(menu_file)

        # Asks the user what menu item index they want to delete
        SEND ("Enter a menu item index to delete\n"
              "Or type \"E\" to exit.\n" TO DISPLAY

        TRY DO
            SEND "Your choice: " TO DISPLAY
            RECEIVE inp FROM (string) keyboard
            SET delete_index TO int(inp)

            # Assigns Last entry's index to "last_item"
            SET name TO menu_file[delete_index-1][2]
            SET tmp_last_item TO menu_file[len(menu_file) - 1]
            SET last_item_index TO int(tmp_last_item[0])
            IF delete_index > last_item_index OR delete_index < 1 DO  #
Out of range
                SEND f"Your delete index must be between: 1 and
{last_item_index}\n"
                      "Please try again...\n" TO DISPLAY
            ELSE  # Index is valid
```

```
                SEND f"Are you sure you want to delete {name}?\n"
                      "- Enter \"Y\" to Confirm Menu Item Deletion\n"
                      "- Enter \"N\" to Discard Changes and Exit to Edit
Menu\n" TO DISPLAY

                WHILE True DO  # Validates input for delete item or Exit
                    SEND "Your choice: " TO DISPLAY
                    RECEIVE choiceInp FROM (string) keyboard
                    SET choiceInp TO choiceInp.lower()

                    IF choiceInp = "y" THEN  # User wants to delete the
menu item
                        # Deletes the menu item
                        del menu_file[delete_index-1]

                        # Corrects menu index numbers in the list
                        FOR EACH index, full_item IN enumerate(menu_file,
start=1) DO
                            SET full_item[0] TO str(index)

                        SEND f"{name} Deleted!" TO DISPLAY
                        BREAK

                    ELIF choiceInp == "n" DO  # User wants to exit
                        BREAK
                    ELSE
                        SEND "Please enter \"y\" or \"n\" as your choice,
try again.\n" TO DISPLAY
                    ENDIF
                END WHILE

                IF choiceInp = "n" THEN  # User exits instead of deleting
menu item
                    SEND "Exiting...\n" TO DISPLAY
                    BREAK
                ENDIF
            ENDIF
        EXCEPT ValueError DO  # User entered input other than an integer
            IF inp = "e" THEN
                BREAK
            ELSE
                SEND "Please enter numeric characters only, try again.\n"
TO DISPLAY
            ENDIF
        END TRY
    RETURN menu_file
    END WHILE

    IF inp = "e" THEN  # Breaks the whole loop and exits to menu
        RETURN menu_file
END FUNCTION


PROCEDURE write_menu (menu_file)
```

```
BEGIN PROCEDURE
    WITH open("menu.txt", "w") as file:  # Creates the file and writes
menu_file
        FOR EACH line IN menu_file DO  # Iterates over each full item in
menu item
            SET index_num TO line[0]  # Assigns to index number
            SET price TO line[1]  # Assigns to price
            SET name TO line[2]  # Assigns to name
            SET category TO line[3]  # Assigns to category

            file.write(f"{index_num},{price},{name},{category}\n")
    END WITH
END PROCEDURE


# Allows user to edit menu items
FUNCTION edit_menu_items (menu_file)
BEGIN FUNCTION
    WHILE True DO
        SEND "-" * 30 TO DISPLAY
        SEND "\nThis is the current menu:" TO DISPLAY
        print_menu(menu_file)

        # Asks the user what menu item index they want to edit
        SEND ("Enter a menu item index to edit\n"
              "Or type \"e\" to exit.\n" TO DISPLAY
        WHILE True DO
            TRY DO
                SEND "Your choice: " TO DISPLAY
                RECEIVE inp FROM (string) keyboard
                SET edit_index TO int(inp)

                # Assigns Last entry's index to "last_item"
                SET tmp_last_item TO menu_file[len(menu_file) - 1]
                SET last_item_index TO int(tmp_last_item[0])
                IF edit_index > last_item_index OR edit_index < 1 DO  #
Out of range
                    SEND f"Your index must be between: 1 and
{last_item_index}\n"
                          "Please try again...\n" TO DISPLAY
                ELSE  # Index is valid
                    BREAK
                ENDIF
            EXCEPT ValueError DO  # User entered input other than an
integer
                IF inp = "e" THEN
                    BREAK
                ELSE
                    SEND "Please enter numeric characters only, try
again.\n" TO DISPLAY
                ENDIF
            END TRY
        END WHILE
        IF inp = "e" THEN  # Breaks the whole loop and exits to menu
            RETURN menu_file
        ENDIF

        # Loop for asking user what to edit of that item until they choose
to exit.
        WHILE True DO
            SEND "-" * 30 TO DISPLAY
            # Prints only that element in the menu with the catergory
            print_menu(menu_file, start_line=edit_index,
final_line=edit_index)

            SET current_name TO menu_file[edit_index-1][2]
            SEND f"Enter a letter from below to choose what to edit of
{current_name}:\n"
                  "(I)ndex # of item\n"
                  "(N)ame of item\n"
                  "(P)rice\n"
                  "(C)ategory\n"
                  "Enter \"E\" to (E)xit and choose another item edit or
exit.\n" TO DISPLAY
            WHILE True DO  # Validates input for choice
                SEND "Your choice: " TO DISPLAY
                RECEIVE inp FROM (string) keyboard
                SET inp to inp.lower()
                IF inp == "i" OR inp == "n" OR inp == "p" OR inp == "c" OR
inp == "e" DO # Valid input is received
                    BREAK  # Breaks the loop
                ELSE
                    SEND "Your input must be either: \"I\", \"N\", \"P\"
or \"C\". Try again!" TO DISPLAY
                ENDIF
            END WHILE

            IF inp = "i" THEN  # User wants to change index number
                print_menu(menu_file, start_line=edit_index,
final_line=edit_index)

                SEND "Choose the menu item index number that you want it
to change to: " TO DISPLAY
                WHILE True DO  # Validates input for inputting index to
change to
                    TRY DO
                        SEND "Index Number: " TO DISPLAY
                        RECEIVE inp FROM (integer) keyboard
                        # Assigns Last entry's index to "last_item"
                        SET tmp_last_item TO menu_file[len(menu_file) - 1]
                        SET last_item_index TO int(tmp_last_item[0])
                        IF inp > last_item_index or inp < 1 DO  # Out of
range
                            PRINT f"Your index must be between: 1 and
{last_item_index}\n"
                                  "Please try again...\n" TO DISPLAY
                        ELSE  # Index is valid
```

```
                            BREAK
                        ENDIF
                    EXCEPT ValueError DO  # User entered input other than
an integer
                        SEND "Please enter numeric characters only, try
again.\n" TO DISPLAY
                    END TRY
                END WHILE

                SET tmp_menu_item TO menu_file[edit_index-1]  # Sets tmp
to selected item
                SET tmp_menu_item[3] TO menu_file[inp-1][3]  # Sets tmp's
category to item to be inserted above

                del menu_file[edit_index-1]  # deletes the old item
                menu_file.insert(inp-1, tmp_menu_item)  # Inserts the tmp
menu item

                # Corrects the index numbers in the list
                FOR EACH index, full_item IN enumerate(menu_file, start=1)
DO
                    SET full_item[0] TO str(index)
                END FOR

                SEND f"Index changed to {inp}." TO DISPLAY

                SET edit_index TO inp

            ELIF inp = "n" DO  # User wants to change name
                SET original TO menu_file[edit_index-1][2]  # Sets
original to current name for item
                SEND f"Current name of item is: {original}" TO DISPLAY

                SEND "Please choose the name of your new menu item.\n" TO
DISPLAY
                WHILE True DO  # Validates Name
                    SEND "Name of new menu item: " TO DISPLAY
                    RECEIVE name_tmp FROM (string) KEYBOARD

                    # Checks if name contains only letters and spaces
                    IF name_tmp[-1] = " " THEN
                        SEND "Name must not contain spaces at the end!" TO
DISPLAY
                    ELIF NOT all(letter.isalpha() OR letter.isspace() FOR
EACH letter IN name_tmp) DO
                        SEND "Name must contain alphabetic characters
only, try again.\n" TO DISPLAY
                    ELSE
                        SET name TO ""
                        SET words TO name_tmp.split()  # Splits name_tmp
into list
                        SET len_words TO len(words)

                        # Takes each word in the string, formats
                        # and concatenates it to variable: "name"
                        FOR EACH index, word IN enumerate(words) DO
                            IF index = 0 THEN  # First word
                                SET word TO word.title()  # Makes first
letter capital
                                SET name TO name + f"{word} "
                            ELIF index+1 = len_words DO  # last word
                                SET name TO name + word
                            ELSE  # Other words
                                SET name TO name + f"{word} "
                            ENDIF
                        END FOR
                        BREAK
                    ENDIF
                END WHILE

                SET menu_file[edit_index-1][2] TO name
                SEND f"Name changed to {name}" TO DISPLAY

            ELIF inp = "p" DO  # User wants to change price
                SET original TO menu_file[edit_index-1][1]  # Sets to
current price of item
                SEND f"Current price of item is: {original}" TO DISPLAY

                SEND "Please choose the price of your new menu item.\n" TO
DISPLAY
                WHILE True DO
                    TRY DO
                        SET price_unrounded TO float(input("Price of new
menu item: $"))
                        SET price TO round(price_unrounded,2)  # Total
price stored as a float
                        BREAK
                    EXCEPT ValueError DO
                        SEND "Please input numeric characters only, try
again.\n" TO DISPLAY
                    END TRY
                END WHILE
                SET price TO str(price)
                IF price[-2] = "." THEN
                    SET price TO price + "0"
                ENDIF

                SET menu_file[edit_index-1][1] TO price  # Assigns the new
price to the menu item
                SEND f"Price changed to {price}" TO DISPLAY

            ELIF inp = "c" DO  # User inputs category for new item
                SET old_category TO menu_file[edit_index-1][3]  # Sets to
current/old category

                SET categories TO ["breakfast", "mains", "extras",
"drinks"]
                SEND "-" * 30 TO DISPLAY
                SEND "Please choose the menu category of your new menu
item:\n"
                      "Categories to choose from: Breakfast, Mains, Extras
and Drinks\n" TO DISPLAY

                WHILE True DO  # Validation of category
                    SEND "Your category: " TO DISPLAY
                    RECEIVE new_category FROM (string) keyboard
                    SET new_category TO new_category.lower()
                    IF new_category in categories = True DO  # Category is
valid
                        BREAK
                    ELSE
                        SEND "Please enter a valid category, try again.\n"
TO DISPLAY
                    ENDIF
                END WHILE

                IF new_category = old_category THEN  # Category hasn't
changed
                    SEND f"Category not changed. ({new_category} is the
same category as before)" TO DISPLAY
                ENDIF

                ELIF new_category != old_category DO  # Categeory has
changed

                    # Takes all categories in menu.txt and only
                    # stores the category in an array:
'categories_in_file'
                    SET categories_in_file TO []
                    FOR EACH element IN menu_file DO  # Iterates over each
element in menu.txt
                        categories_in_file.append(element[3])
                    END FOR

                    # Finds how many of 'category' is in
'categories_in_file'
                    # Along with how what their specific indexes are on
                    # the menu in the 'categories_present_index' variable
                    SET categories_present_index TO []
                    FOR EACH index, category_tmp IN
enumerate(categories_in_file) DO
                        IF category_tmp = new_category THEN
                            categories_present_index.append(int(index+1))
                        ENDIF

                    SET start_line TO min(categories_present_index)  #
Assigns min index value
                    SET final_line TO max(categories_present_index)  #
Assigns max index value

                    SEND "-" * 30 TO DISPLAY
                    print_menu(menu_file, start_line=start_line,
final_line=final_line)

                    # Get new item's index from user and insert it into
the menu file
                    SEND "You must assign a new index to your item since
it's in a different category than before.\n"
                          f"Please note that items shown above are for the
{new_category} category.\n"
                          f"Choose an index between {start_line} and
{final_line}.\n" TO DISPLAY

                    SET old_index TO menu_file[edit_index-1][0]  # Saves
old index for later

                    WHILE True DO  # Validation Check of menu item index
number
                        TRY DO
                            SEND "Your new menu item index: " TO DISPLAY
                            RECEIVE new_index FROM (integer) keyboard
                            IF new_index < start_line or new_index >
final_line DO   # Out of range
                                SEND f"Please enter a value between
{start_line} and {final_line}.\n" TO DISPLAY
                            ELSE
                                BREAK
                            ENDIF
                        EXCEPT ValueError DO
                            SEND "Your index must contain only numeric
characters only, try again.\n" TO DISPLAY
                        END TRY
                    END WHILE

                    SET tmp_new_item TO menu_file[edit_index-1]  # Saves
the new item to tmp_new_item
                    del menu_file[edit_index-1]  # Deletes the old item

                    SET tmp_new_item[0] TO str(new_index)  # Assigns the
new index to the menu item
                    SET tmp_new_item[3] TO new_category  # Assigns the new
category to the menu item

                    menu_file.insert(new_index-1, tmp_new_item)  # Inserts
the tmp menu item
                    SEND f"Index changed to {new_index}" TO DISPLAY

                    # Corrects the index numbers in the menu
                    FOR EACH index, full_item IN enumerate(menu_file,
start=1) DO
                        SET full_item[0] TO str(index)

                    SET edit_index TO new_index  # Saves edit_index for
next use
```

```
            ELSE  # User wants to exit editing loop of current item
                BREAK
            ENDIF
        END WHILE
    END WHILE
END FUNCTION


# Allows user to add menu items to the menu
FUNCTION add_menu_items (menu_file)
BEGIN FUNCTION

    # Prints the current menu for the user to see
    SEND "-" * 30 TO DISPLAY
    SEND "\nThis is the current menu:" TO DISPLAY
    print_menu(menu_file)

    # User inputs category for new item
    SET categories TO ["breakfast", "mains", "extras", "drinks"]
    SEND "-" * 30 TO DISPLAY
    WHILE True DO  # Loops adding item process

        SEND "Please choose the menu category of your new menu item:\n"
              "Categories to choose from: Breakfast, Mains, Extras and
drinks\n" TO DISPLAY
        WHILE True DO  # Validation of category
            SEND "Your category: " TO DISPLAY
            RECEIVE category FROM (string) keyboard
            SET category to category.lower()
            IF NOT category in categories DO
                SEND "Please enter a valid category, try again.\n" TO
DISPLAY
            ELSE
                BREAK
            ENDIF
        END WHILE

        # User inputs name for new item
        SEND "-" * 30 TO DISPLAY
        SEND "Please choose the name of your new menu item.\n" TO DISPLAY
        WHILE True DO  # Validates Name
            SEND "Name of new menu item: " TO DISPLAY
            RECEIVE name_tmp FROM (string) keyboard

            # Checks if name contains only letters and spaces
            IF name_tmp[-1] = " " THEN
                SEND "Name must not contain spaces at the end!" TO DISPLAY
            ELIF NOT all(letter.isalpha() OR letter.isspace() FOR EACH
letter IN name_tmp) DO
                SEND "Name must contain alphabetic characters only, try
again.\n" TO DISPLAY
            ELSE
                SET name TO ""
                SET words TO name_tmp.split()  # Splits name_tmp into list
                SET len_words TO len(words)

                # Takes each word in the string, formats
                # and concatenates it to variable: "name"
                FOR EACH index, word IN enumerate(words) DO
                    IF index = 0 THEN  # First word
                        SET word TO word.title()  # Makes first letter
capital
                        SET name TO name + f"{word} "
                    ELIF index+1 = len_words DO  # last word
                        SET name TO name + word
                    ELSE  # Other words
                        SET name TO name + f"{word} "
                    ENDIF
                END FOR
                BREAK
            ENDIF
        END WHILE

        # User inputs price for new item
        SEND "-" * 30 TO DISPLAY
        SEND "Please choose the price of your new menu item.\n" TO DISPLAY
        WHILE True DO
            TRY DO
                SET price_unrounded TO float(input("Price of new menu
item: $"))
                SET price TO round(price_unrounded,2)  # Total price
stored as a float
                BREAK
            EXCEPT ValueError DO
                SEND "Please input numeric characters only, try again.\n"
TO DISPLAY
            END TRY
        END WHILE
        SET price TO str(price)
        IF price[-2] = "." THEN
            SET price TO price + "O"
        ENDIF

        # Takes all categories in menu.txt and only
        # stores the category in an array: 'categories_in_file'
        SET categories_in_file TO []
        FOR element IN menu_file DO  # Iterates over each element in
menu.txt
            categories_in_file.append(element[3])
        END FOR

        # Finds how many of 'category' is in 'categories_in_file'
        # Along with how what their specific indexes are on
        # the menu in the 'categories_present_index' variable
        SET categories_present_index TO []
        FOR EACH index, category_tmp IN enumerate(categories_in_file) DO
            IF category_tmp = category THEN
                categories_present_index.append(int(index+1))
            ENDIF
        END FOR

        SET start_line TO min(categories_present_index)  # Assigns min
index value
        SET final_line TO max(categories_present_index)  # Assigns max
index value

        SEND "-" * 30 TO DISPLAY
        print_menu(menu_file, start_line=start_line,
final_line=final_line)

        # Get new item's index from user and insert it into the menu file
        SEND "What index would you like to assign to your new menu
item?\n"
              f"Please note that items shown above are for the {category}
category.\n"
              f"Choose an index between {start_line} and {final_line}.\n"
TO DISPLAY

        WHILE True DO  # Validation Check of menu item index number
            TRY DO
                SEND "your new menu item index: " TO DISPLAY
                RECEIVE new_index FROM (integer) keyboard
                IF new_index < start_line or new_index > final_line DO   #
Out of range
                    SEND f"Please enter a value between {start_line} and
{final_line}.\n" TO DISPLAY
                ELSE
                    BREAK
            EXCEPT ValueError DO
                SEND "Your index must contain only numeric characters
only, try again.\n" TO DISPLAY
            END TRY
        END WHILE

        # Formatting the new menu item for it to be written to menu.txt
        SET new_item TO [f'{new_index}', f"{price}", f"{name}",
f"{category}"]

        # Creates temp version as a preview for user
        SET temp_menu_file TO menu_file[:]

        # Adds 1 to the existing menu items' indexes
        FOR EACH index, el in enumerate(temp_menu_file, start=1) DO
            IF index > new_index THEN
                SET el_index TO int(el[0])
                SET el_index TO el_index + 1
                SET el[0] TO str(el_index)
        END FOR
        temp_menu_file.insert(new_index-1, new_item)  # Inserts the new
item

        SEND "-" * 30 TO DISPLAY
        print_menu(temp_menu_file, start_line=start_line,
final_line=final_line+1, inserted_line=new_index-1)

        SEND ("\n(S)ave changes and exit\n"
              "(R)etry and discard changes\n"
              "Save Changes and (A)dd another menu item\n"
              "(D)iscard changes and exit\n" TO DISPLAY

        WHILE True DO  # Validates input
            SEND "Your Choice: " TO DISPLAY
            RECEIVE choice FROM (string) keyboard
            SET choice TO choice.lower()
            IF choice == "s" OR choice == "r" OR choice == "a" OR choice
== "d" DO
                BREAK
            ELSE
                SEND ("Please enter either:"
                      "\"S\" to save,\n"
                      "\"R\" to retry,\n"
                      "\"A\" to add another item\n"
                      "\"D\" to discard changes and exit" TO DISPLAY
            ENDIF
        END WHILE

        IF choice = "s" THEN  # Save and Exit (S)
            SET menu_file TO temp_menu_file[:]  # Writes temp to menu_file
            BREAK  # Breaks the while
        ELIF choice = "r" DO  # Retry without saving changes (R)
            FOR EACH index, el IN enumerate(temp_menu_file, start=1) DO
                IF index > new_index DO
                    SET el_index TO int(el[0])
                    SET el_index TO el_index - 1
                    SET el[0] TO str(el_index)
                ENDIF
            END FOR
        ELIF choice = "a" DO  # Add another item (A)
            SET menu_file TO temp_menu_file[:]  # Saves changes to
menu_file
        ELSE  # Exits without saving changes (D)
            FOR EACH index, el in enumerate(temp_menu_file, start=1) DO
                IF index > new_index DO
                    SET el_index TO int(el[0])
                    SET el_index TO el_index - 1
                    SET el[0] TO str(el_index)
                ENDIF
            END FOR
            BREAK
        ENDIF
    END WHILE

    IF choice = "s" THEN  # Write changes to menu.txt
        WITH open("menu.txt", "w") AS file DO
            FOR EACH item IN menu_file DO  # writes from menu_file
                file.write(f"{item[0]},{item[1]},{item[2]},{item[3]}\n")
            END FOR
        END WITH
    ENDIF
    RETURN menu_file
END FUNCTION


# Provides menu interface for user to choose to Add, edit or delete menu
items
FUNCTION editing_main_menu (menu_file)  # Credits to github.com/RoyceLWC
for Menu.
BEGIN FUNCTION
    WHILE True DO
        SEND "\n-------Edit Menu Items--------" TO DISPLAY
        SET menu TO {
            "1": [": Add menu items", add_menu_items],
            "2": [": Edit an existing menu item", edit_menu_items],
            "3": [": Delete menu items", delete_menu_items],
            "4": [": Exit to main menu"]


        # Prints each menu index and its corresponding functions
description
        FOR EACH key IN sorted(menu.keys()) DO
            SEND key + menu[key][0] TO DISPLAY
        END FOR

        WHILE True DO  # Loop until a valid index is received
            SEND "-" * 30 TO DISPLAY
            SEND "Select an index: " TO DISPLAY
            RECEIVE index FROM (string) keyboard
            TRY DO  # Try to convert to an integer
                SET index TO int(index)  # Converts to an integer
                IF 1 <= index <= 4 DO  # In range
                    BREAK
                ELSE  # Out of range
                    SEND "Out of range try again!" TO DISPLAY
            EXCEPT ValueError DO  # If it can't be converted to an integer
                SEND "Invalid index" TO DISPLAY
            END TRY
        END WHILE
        SEND "-" * 30 TO DISPLAY

        IF index = 4 THEN  # User wants to exit the menu
            write_menu(menu_file)  # Saves the menu to menu.txt
            RETURN menu_file
        ELSE
            SET menu_file TO menu[str(index)][1](menu_file)
        ENDIF
    END WHILE
END FUNCTION


# Writes running totals to running_totals.txt
```

```
PROCEDURE write_order (total_price, total_quantity, quantity_dict,
table_num)
BEGIN PROCEDURE
    WITH open("running_totals.txt", "w") as file DO  # Creates the file
and writes default menu
        file.write(f"{total_price}\n")
        file.write(f"{total_quantity}\n")
    END WITH
END PROCEDURE


# Displays the current order with the quantities of each menu item.
PROCEDURE display_order (total_price, total_quantity, quantity_dict,
table_num)
BEGIN PROCEDURE

    # By default the price doesn't display all 2 decimal points.
    # E.g. a price of "$2.50" would only display "$2.5"
    # These 3 lines below fix this issue.
    SET total_price TO str(total_price)
    IF total_price[-2] = "." THEN
        SET total_price TO total_price + "0"
    ENDIF

    SEND f"\nYour order for {table_num}: \n" TO DISPLAY
    FOR EACH key, value IN quantity_dict.items() DO
        SEND key , ' == ', value TO DISPLAY
    SEND f"\nTotal Price = ${total_price}" TO DISPLAY
    SEND f"Total Quantity of items ordered = {total_quantity}\n" TO
DISPLAY
    SEND "-" * 30 TO DISPLAY
    SEND "\n" TO DISPLAY
END PROCEDURE



# Generates a dictionary of how many of each item has been ordered in a
simple dict format.
FUNCTION get_quantity (names)
BEGIN FUNCTION
    SET elements_dict TO dict()
    FOR EACH elem IN names DO  # Iterate over each element in list
        IF elem in elements_dict DO  # If element exists add 1 to value
else stay at one
            elements_dict[elem] += 1
        else:
            elements_dict[elem] = 1
        ENDIF
    END FOR
    SET elements_dict TO { key:value for key, value in
elements_dict.items()}
    RETURN elements_dict
END FUNCTION
```

```
# Calculates quantity and cost totals
FUNCTION get_totals (full_order)
BEGIN FUNCTION
    SET prices TO []
    FOR EACH index IN range(1, len(full_order)) DO
        SET item TO full_order[index]  # Gets the Full item information
        SET price TO item[1]  # Gets the price only
        prices.append(float(price))  # Converts the string price into a
float and adds to prices.
    END FOR
    SET total_price_tmp TO 0
    FOR EACH price IN prices DO  # Adds up the prices together
        SET total_price_tmp TO total_price_tmp + price  # **total price
currently**
    END FOR
    SET total_price TO round(total_price_tmp,2)  # Total price stored as a
float

    SET names TO []
    FOR EACH index IN range(1, len(full_order)) DO
        SET item TO full_order[index]  # Gets the Full item information
        SET name TO item[2]  # Gets name only of full item
        names.append(name)
    END FOR
    SET total_quantity TO 0
    SET quantity_dict TO get_quantity(names)
    FOR EACH key, value IN quantity_dict.items() DO
        SET total_quantity TO total_quantity + value  # **total quantity
currently**
    END FOR
    RETURN total_price, total_quantity, quantity_dict
END FUNCTION



# Checks order with menu in "menu.txt" and generates 'full_order' var
FUNCTION get_order (data, menu_file)
BEGIN FUNCTION
    SET full_order TO []
    SET table_num TO f"Table #{data[0]}"
    full_order.append(table_num)
    SET tmp_order TO ""
    FOR i FROM 1 TO LENGTH(data) DO  # iterates over order input (data)
except for table num.
        SET menu_num TO data[i]  # set the menu_num to i (any number
greater than index: 0)
        FOR EACH menu_item IN menu_file DO  # Searches for index num in
menu_file
            IF menu_item[0] = menu_num THEN
                SET tmp_order TO menu_item
                full_order.append(tmp_order)
                BREAK
            ENDIF
        END FOR
    END FOR
    # The 'full_order' var consists of the table num at 0 and each element
    # contains the each order's line in menu.txt one by one.

    SET total_price, total_quantity, quantity_dict TO
get_totals(full_order)  # Calculates quantity and cost totals.
    display_order(total_price, total_quantity, quantity_dict, table_num)
    write_order(total_price, total_quantity, quantity_dict, table_num)
    return total_price, total_quantity, quantity_dict, table_num
END FUNCTION



# Takes menu.txt from the "menu_file" var and prints it to user
FUNCTION print_menu (menu_file, **kwargs)
BEGIN FUNCTION
    SET items TO []
    FOR EACH element IN menu_file DO  # Takes menu_file items and strips
them of the category e.g. "breakfast"
        SET tmp TO []
        FOR EACH index, el IN enumerate(element) DO
            IF not index = 3 THEN
                tmp.append(el)
            ENDIF
        END FOR
        items.append(tmp)
    END FOR

    SET tmp2 TO []  # Finds which element in the array has the greatest
amount of chars.
    FOR EACH element IN items DO
        SET string TO ""
        FOR EACH el IN element DO
            SET string TO string + el
        END FOR
        tmp2.append(string)
    END FOR
    SET max_len_el TO max([len(i) for i in tmp2])
    SET max_len TO max_len_el + 5

    SET real_output TO []
    FOR EACH element IN menu_file DO  # Formats each menu item with the
right amount of periods "."
        SET number TO element[0]
        SET price TO element[1]
        SET name TO element[2]
        SET full_item_tmp TO f"{number}. {name} ${price}"
        SET length_of_full TO len(full_item_tmp)
        SET period_num TO max_len - length_of_full
        SET periods TO period_num * "."
        SET full_item TO f"{number}. {name} {periods} ${price}"
        real_output.append(full_item)
    END FOR
    # Sets values using kwargs to determine
    # which specfic lines to iterate over
    SET start_line TO kwargs.get("start_line")
    SET final_line TO kwargs.get("final_line")
    SET inserted_line TO kwargs.get("inserted_line")
    SET single_line TO kwargs.get("single_line")

    # Takes all categories in menu.txt and only
    # stores the category in an array: 'categories_in_file'

    SET categories_in_file TO []
    FOR EACH element IN menu_file DO  # Iterates over each element in
menu.txt
        categories_in_file.append(element[3])
    END FOR

    # Finds how many of each category is in the file
    SET category_dict TO get_quantity(categories_in_file)

    # Formats categories with separators for menu
    SET all_keys TO []
    SET separator TO "~"  # Define the separator for printing
    IF start_line = None THEN  # No **kwargs provided
        FOR EACH key IN category_dict.keys() DO  # Iterates over each
category
                SET full_item_tmp TO f"{key} "
                SET length_of_full TO len(full_item_tmp)
                SET separator_num TO max_len - length_of_full
                SET separators TO separator_num * separator
                SET full_item TO f"\n{key.title()} {separators}"
                all_keys.append(full_item)
    ELSE  # **kwargs provided
        SET category TO menu_file[start_line-1][3]
        SET full_item_tmp TO f"{category} "
        SET length_of_full TO len(full_item_tmp)
        SET separator_num TO max_len - length_of_full
        SET separators TO separator_num * separator
        SET full_item TO f"\n{category.title()} {separators}"
    ENDIF

    # Print if no **kwargs are provided
    IF start_line = None AND final_line = None AND inserted_line = None DO
        SET current_index TO 0  # Current index of real_output
        FOR EACH key IN all_keys DO
            SEND key TO DISPLAY
        END FOR

            # Loops for the amount of items there are for that
key/category
            FOR i FROM 1 TO category_dict[key.strip(f"{separator}
\n").lower()] DO
                SEND f"\n{real_output[current_index]}" TO DISPLAY
                SET current_index TO current_index + 1  # Changes to next
key
            END FOR

    # start_line and final_line are provided but inserted_line isn't as
**kwargs
    ELIF (start_line != None and final_line != None) and inserted_line ==
None DO
        SEND full_item TO DISPLAY
        FOR EACH index, item IN enumerate(real_output) DO
            IF start_line <= index+1 <= final_line DO  # Index is in
correct printing range
                SEND f"\n{item}") TO DISPLAY # Prints all menu items
formatted with perfect amount of periods
            ENDIF
        END FOR

    # start_line, final_line and inserted_line are all provided as
**kwargs
    ELIF start_line != None or final_line != None or inserted_line != None
DO
        SEND full_item TO DISPLAY
        FOR EACH index, item IN enumerate(real_output) DO
            IF start_line <= index+1 <= final_line DO  # Index is in
correct printing range
                IF index+1 = inserted_line+1 DO
                    # Prints menu items formatted with perfect amount of
periods.
                    SEND f"\n{item} <--- YOUR NEW ITEM" TO DISPLAY
                ELSE
                    SEND f"\n{item}" TO DISPLAY  # Prints all menu items
formatted with perfect amount of periods.
                ENDIF
            ENDIF
        END FOR
    ENDIF
    SEND "\n" TO DISPLAY
END FUNCTION



FUNCTION get_order_input (menu_file)  # Validates Order
BEGIN FUNCTION
    print_menu(menu_file)
    WHILE True DO
        SEND "-" * 30 TO DISPLAY
        SEND ("\nE.g. \"6,4,4,7,8,10,10\""
              " Would be an order from Table 6 for 2 Burgers, 1 Fries, 1
Salad and 2 Soft"
              " Drinks." TO DISPLAY
        SEND "\nType order here: " TO DISPLAY
        RECEIVE tmpData FROM (string) keyboard
        SET tmpData TO tmpData.split(",')
        # Removes spaces from each element in tmpData,
        # then adds elements to 'data'
        SEND = [ TO DISPLAY
        FOR EACH string IN tmpData DO  # Iterates over each value
            SET string TO ''.join(string.split())  # Removes spaces from
data
            data.append(string)
        END FOR

        SET tmp_last_item TO menu_file[len(menu_file) -  1]  # Assigns
Last entry's index
        SET last_item TO tmp_last_item[0]                    # to
"last_item"
        SET invalid TO ""
        SET invalidTable TO ""
        FOR EACH index, element IN enumerate(data) DO  # Takes each
element in the array
            IF index = 0 THEN  # If the element is first (table number
doesn't apply for these rules)
                IF element.isnumeric() = False DO  # If first element in
data is NOT numeric:
                    SET invalid TO True
                ELIF int(element) > 10 DO  # If the element is numerically
greater than number of tables in (10)
                    SET invalidTable TO True
                ENDIF
            ELSE IF
                IF len(element) > len(last_item) or element.isnumeric() ==
False DO
                    SET invalid TO True  # ^If the digit length is greater
than the digit length in the menu or is not integer
                ELIF int(element) > int(last_item) DO  # If the element is
numerically greater than the last item
                    SET invalid TO True
                ENDIF
            ENDIF
        END FOR
        IF invalidTable DO
            SEND "\nWe only have 10 tables! Table number must be lower
than 10, please try again." TO DISPLAY
        ELIF invalid DO  # If there are letters or symbols in the input:
            SEND "\nYour order has invalid characters, please try again."
TO DISPLAY
        ELIF len(data) == 1 DO
            SEND "\nAt least one order must be made per table, please try
again." TO DISPLAY
        ELSE
            BREAK
        ENDIF
    END WHILE
    # data variable is input data in array format.
    SET total_price, total_quantity, quantity_dict, table_num TO
get_order(data, menu_file)
    RETURN total_price, total_quantity, quantity_dict, table_num
```

```
END FUNCTION



# Main Menu, first menu that the user sees.
PROCEDURE main_menu (menu_file)   # Credits to github.com/RoyceLWC for
Menu.
BEGIN PROCEDURE
    SET hasData TO False
    WHILE True DO
        SEND "\n----------Main Menu-----------" TO DISPLAY
        SET menu TO {
            "1": [": Input order data", get_order_input],
            "2": [": Edit Menu Items", editing_main_menu],
            "3": [": Finalize Order", finalize_order],
            "4": [": Options", options],
            "5": [": Quit to Desktop"]
            }

        # Prints each menu index and its corresponding functions
description
        FOR EACH key IN sorted(menu.keys()) DO
            SEND key + menu[key][0] TO DISPLAY
        END FOR

        WHILE True DO  # Loop until a valid index is received
            SEND "-" * 30 TO DISPLAY
            SEND "Select an index: " TO DISPLAY
            RECEIVE index FROM (string) keyboard
            TRY DO  # Try to convert to an integer
                SET index TO int(index)  # Converts to an integer
                IF 1 <= index <= 5 DO  # In range
                    BREAK
                ELSE  # Out of range
                    SEND "Out of range try again!" TO DISPLAY
                ENDIF
            EXCEPT ValueError DO  # If it can't be converted to an integer
                SEND "Invalid index" TO DISPLAY
            END TRY
        END WHILE
        SET data TO ""
        SEND "-" * 30 TO DISPLAY
        IF index == 1 DO  # get_order_input() | Get all data about the
order, e.g. price, quantity and table num.
            SET total_price, total_quantity, quantity_dict, table_num TO
menu[str(index)][1](menu_file)
            SET hasData TO True
        ELIF index == 2 or index == 4 DO  # editing_main_menu or options()
            SET menu_file TO menu[str(index)][1](menu_file)
        ELIF index == 3 DO  # finalize_order()
            IF NOT hasData DO  # No order data
                print("No current order! Go back and Input an order.")
            ELSE
```

```
                SET total_price, total_quantity, quantity_dict, table_num,
hasData TO menu[str(index)][1](total_price, total_quantity, quantity_dict,
table_num, hasData)
            ENDIF
        ELSE
            quit()
        ENDIF
    END WHILE
END PROCEDURE



FUNCTION check_file (DEFAULT_MENU)  # Checks for menu and Creates
DEFAULT_MENU if doesn't exist.
BEGIN FUNCTION
    FOR i FROM 0 TO 2 DO  # Quick fix for the file not being read on first
try idk.
        TRY DO  # Check for existing file by trying to read the file
            WITH open("menu.txt", "r") AS file DO
                SET whole_file TO file.readlines()  # Stores the menu in
the file as "whole_file"
                SET menu_file TO []
                FOR EACH element IN whole_file DO
                    SET element TO element.strip("\n")
                    SET element TO element.split(",")
                    menu_file.append(element)
                END FOR
                RETURN menu_file
            END WITH
        EXCEPT IOError DO  # If menu.txt is not found, make a new file
            WITH open("menu.txt", "w") AS file DO  # Creates the file and
writes default menu
                FOR EACH line IN DEFAULT_MENU DO
                    file.write(f"{line}\n")
            END WITH
            check_file(DEFAULT_MENU)
        END TRY
END FUNCTION


SET menu_file TO check_file(DEFAULT_MENU)  # Checks for menu and Creates
DEFAULT_MENU if doesn't exist.
main_menu(menu_file)  # Main Menu for most the program
```