```
# 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
```