



**Module Code: AAPP010-4-2-PWP**

**Module Name: PROGRAMMING WITH PYTHON**

**Intake Code: UCDF2005(1) ICT(SE)**

**Hand Out Date: 22<sup>ND</sup> APRIL 2021**

**Hand In Date: 18<sup>TH</sup> JUNE 2021**

**Weightage: 100%**

## **PYTHON ASSIGNMENT**

**Team Members Name & TP:** 1. Tan Chun Hung TP058968

2. Cheng Yi Heng TP058994

**Lecturer:** Mr. Usman Hashmi

## Table of Contents

Introduction and Assumptions.....	4
Pseudocode.....	5
Utility Functions .....	5
Console Input Utility Functions.....	5
I/O Utility Functions .....	7
Login System .....	9
Admin.....	10
All Customers .....	16
Registered Customers.....	17
Main Program .....	22
Flowchart .....	23
Utility Functions .....	23
Console Input Utility Functions.....	24
I/O Utility Functions .....	29
Login System .....	32
Admin.....	34
All Customers .....	40
Registered Customers.....	43
Main Program .....	49
Program Source Code .....	52
Annotations.....	52
Naming Conventions.....	52
File Structure.....	52
Utility Functions .....	55
Console Input Utility Functions.....	55
Console Input Utility Functions Summary .....	57
I/O Utility Functions .....	58
I/O Utility Functions Summary.....	60
Login System .....	60
Admin.....	63
All Customers .....	71

Registered Customers .....	73
Main Program .....	79
Additional Features.....	81
Return to main menu.....	81
Delete Customer Account.....	81
Available Car Remaining Counter .....	82
Hourly and Daily Renting Price .....	82
Sample Input and Output.....	83
Main Menu.....	83
Welcome Page .....	83
Admin.....	84
Admin Main Menu .....	84
Add Car.....	85
Modify Car Details.....	86
Display Records.....	88
Search Records.....	89
Return Rented Car.....	90
All Customers .....	92
All Customers Main Menu .....	92
View All Cars Available For Rent .....	92
Create new account .....	93
Registered Customers.....	94
Registered Customer Main Menu.....	94
Modify Personal Details .....	95
View Rented Cars .....	96
Cars To Be Rented .....	97
Select And Book A Car For A Specific Duration.....	97
Do Payment To Confirm Booking.....	99
Delete Account.....	99
Exit Program.....	101
Termination Confirmation .....	101
Conclusion .....	102
Abstraction.....	102

Functions in a list .....	102
Consistency and Readability .....	102
Constant variables.....	103
Workload Matrix.....	104
Subtask Workload .....	104
References .....	105

## Introduction and Assumptions

Due to large number of customers demanding ease to use and fast rental service of cars, SUPER CAR RENTAL SERVICES (SCRS) quoted us to develop python program for online car rental system to enhance effectiveness and efficiency of customers ordering and booking cars for rent.

Our program has 3 types of users as requested by SCRS, which are **admin**, **all customer**, and **registered customers**. We assume that “**all customers**” must use username and password as per global standardization to create an account as “**registered customer**”. We also assume that registered customers’ personal details are username and password, they can modify their personal details which are username and password. We also assume that “**registered customer**” can view detail of cars to be rented out, which are car rental price and car name (brand & model) while “**all customer**” can only view cars available for rent.

Next, for the **admins**, we assume that they can modify the details of each cars (brand, name, description, hourly price, daily price, and remaining count). They can also return a specific car that is rented by a customer. Admins can also look at records of a specific customer, for example, their rental and booking history.

Lastly, we assume that the “**select and book a car for a specific duration**” section is our “**shopping cart**”. Customers can perform payment to confirm booking that is being included in the “**shopping cart**” and we let customers confirm their payment via a binary question (yes/no).

## Pseudocode

### Utility Functions

```
FUNCTION join_util(join_list, join_str)
BEGIN
    IF join_list[0] != "" THEN
        RETURN JOIN join_list with join_str
    ELSE
        RETURN join_list[1]
    ENDIF
END
```

*Pseudo 1: join\_util*

### Console Input Utility Functions

```
FUNCTION get_user_int(prompt_msg)
BEGIN
    usr_input = ""
    DOWHILE NOT usr_input IS DIGIT
        DISPLAY prompt_msg
        READ usr_input
        IF NOT usr_input IS DIGIT THEN
            DISPLAY "Input is not an integer\nPlease try again\n"
        ENDIF
    ENDWHILE
    RETURN usr_input AS INTEGER
END
```

*Pseudo 2: get\_user\_int*

```
FUNCTION get_user_float(prompt_msg)
BEGIN
    usr_input = ""
    DOWHILE NOT (REPLACE FIRST "." WITH "" IN usr_input) IS DIGIT
        DISPLAY prompt_msg
        READ usr_input
        IF NOT REPLACE FIRST "." WITH "" IN usr_input IS DIGIT THEN
            DISPLAY "Input is not a number\nPlease try again\n"
        ENDIF
    ENDWHILE
    RETURN usr_input AS FLOAT
END
```

*Pseudo 3: get\_user\_float*

```

FUNCTION get_user_not_empty(prompt_msg, error_msg="Input cannot be empty\n")
BEGIN
    usr_input = ""
    DOWHILE usr_input == ""
        DISPLAY prompt_msg
        READ usr_input
        IF usr_input == "" THEN
            DISPLAY error_msg
        ENDIF
    ENDWHILE
    RETURN usr_input
END

```

*Pseudo 4: get\_user\_not\_empty*

```

FUNCTION get_user_selection(prompt_msg, selections,
    error_msg="Input is not within selections given\n")
BEGIN
    usr_input = ""
    DOWHILE usr_input NOT IN selections
        DISPLAY prompt_msg
        READ usr_input
        IF usr_input NOT IN selections THEN
            DISPLAY error_msg
        ENDIF
    ENDWHILE
    RETURN usr_input
END

```

*Pseudo 5: get\_user\_selection*

```

FUNCTION get_user_int_range(prompt_msg, range_min, range_max,
    exceed_range_error_msg="Input has exceed the range given!\n")
BEGIN
    usr_int = range_min - 1
    DOWHILE usr_int < range_min OR usr_int > range_max
        usr_int = get_user_int(prompt_msg)
        DISPLAY prompt_msg
        READ usr_int
        IF usr_int < range_min OR usr_int > range_max THEN
            DISPLAY exceed_range_error_msg
        ENDIF
    ENDWHILE
    RETURN usr_int
END

```

*Pseudo 6: get\_user\_int\_range*

## I/O Utility Functions

```
FUNCTION get_file_info(filename)
BEGIN
    READ file_content FROM filename
    lines = SPLIT file_content BY "\n"

    content1 = []
    content2 = []

    LOOP line_idx FROM 0 TO len(lines) - 1
        IF line_idx % 2 == 0 THEN
            APPEND lines[line_idx] INTO content1
        ELSE
            APPEND lines[line_idx] INTO content2
        ENDIF
    ENDLOOP
    RETURN content1, content2
END
```

*Pseudo 7: get\_file\_info*

```
FUNCTION modify_file_info(filename, location, is_content1, info)
BEGIN
    content1, content2 = get_file_info(filename)

    IF is_content1 THEN content1[location] = info
    ELSE content2[location] = info
    ENDIF

    final_string = ""
    LOOP content_idx FROM 0 TO len(content1)
        final_string += content1[content_idx] + "\n"
        final_string += content2[content_idx] + "\n"
    ENDLOOP

    WRITE final_string INTO filename
END
```

*Pseudo 8: modify\_file\_info*



```

FUNCTION delete_file_info(filename, location)
BEGIN
    content1, content2 = get_file_info(filename)

    final_string = ""

    LOOP content_idx FROM 0 TO len(content1)
        IF (content_idx != location) THEN
            final_string += content1[content_idx] + "\n"
            final_string += content2[content_idx] + "\n"
        ENDIF
    ENDLOOP

    WRITE final_string INTO filename
END

```

*Pseudo 9: delete\_file\_info*

```

FUNCTION add_file_info(filename, new_content1, new_content2)
BEGIN
    APPEND new_content1 INTO filename
    APPEND new_content2 INTO filename
END

```

*Pseudo 10: add\_file\_info*

## Login System

```
FUNCTION register_new_user(filename, usernames)
BEGIN
    username = ""
    password = ""

    DOWHILE username == "" OR username IN usernames
        username = get_user_not_empty("Enter username: ")
        IF username == "" THEN
            DISPLAY "Username cannot be empty!\n"
        ELSE IF username IN usernames THEN
            DISPLAY "Username has been taken!\nPlease enter another username.\n"
        ENDIF
    ENDWHILE

    DOWHILE password == ""
        password = get_user_not_empty("Enter password: ")
        IF password == "" THEN
            DISPLAY "Password cannot be empty!\n"
        ENDIF
    ENDWHILE

    DISPLAY "Your username and password is " + username + "," + password

    APPEND (username + "\n" + password + "\n") INTO filename
END
```

*Pseudo 11: register\_new\_user*

```
FUNCTION login(usernames, passwords)
BEGIN
    DISPLAY "Please log in.\n"

    username = ""
    username_idx = 0

    DOWHILE username NOT IN usernames
        username = get_user_not_empty("Username: ")
        IF username NOT IN usernames THEN
            DISPLAY "Username not found\nPlease try again\n"
        ELSE
            username_idx = INDEX username FROM usernames
        ENDIF
    ENDWHILE

    password = ""

    DOWHILE password != passwords[username_idx]
        password = get_user_not_empty("Password: ")
        IF password != passwords[username_idx] THEN
            DISPLAY "Password incorrect\nPlease try again\n"
        ENDIF
    ENDWHILE

    DISPLAY "Logged in!\n"
    DISPLAY "Press enter to continue..."
    READ
    os.system("cls")
    RETURN username_idx
END
```

*Pseudo 12: login*

## Admin

```
FUNCTION add_new_car()
BEGIN
    car_details, _ = get_file_info(CARS_FILE)

    used_car_names = []
    LOOP i FROM 0 to len(car_details)
        car_details[i] = car_details[i].split("|")
        used_car_names[i] = LOWERCASE car_details[i][0] + LOWERCASE car_details[i][1]
    ENDLOOP

    car_detail = ""
    car_name = ""

    brand = ""
    model = ""
    description = ""

    DOWHILE car_detail == "" OR car_name IN used_car_names

        brand = get_user_not_empty("Enter brand: ", "Brand cannot be empty!\n")
        model = get_user_not_empty("Enter model: ", "Model cannot be empty!\n")
        description = get_user_not_empty("Enter description: ", "Description cannot be empty!\n")
        amount = get_user_int("Enter amount of cars to add: ")

        car_detail = brand + "|" + model + "|" + description + "|" + amount + "\n"
        car_name = LOWERCASE brand + LOWERCASE model

        IF car_name IN used_car_names THEN
            DISPLAY "The car name has been taken!\nPlease create a unique one.\n"
        ENDIF
    ENDWHILE

    hourly_price = get_user_float("Enter hourly price: ")
    daily_price = get_user_float("Enter daily price: ")
    price_detail = hourly_price + "|" + daily_price + "\n"

    DISPLAY "Your car name is: ", brand, model
    DISPLAY "Your car description is: ", description
    DISPLAY "The hourly price is: ", hourly_price + "\n" + "The daily price is: ", daily_price

    add_file_info(CARS_FILE, car_detail, price_detail)
END
```

*Pseudo 13: add\_new\_car*

```

FUNCTION modify_car_details()
BEGIN
    cars, prices = get_file_info(CARS_FILE)

    car_details = []
    price_details = []

    LOOP i FROM 0 TO len(cars)
        APPEND (SPLIT cars[i] BY "|") INTO car_details
        APPEND (SPLIT prices[i] BY "|") INTO price_details
    ENDLOOP

    view_cars(False, False)
    car_idx = get_user_int_range("Choose a car index to edit: ", 1, len(car_details)) - 1

    DISPLAY "\nWhich car detail you want to modify?"
    DISPLAY("\n1. Car Detail\n2. Price Detail")
    choose_detail = get_user_int_range("\nEnter (1/2): ", 1, 2)

    IF choose_detail == 1 THEN

        DISPLAY "\nChoose car detail to be modified:"
        DISPLAY"\n1. Brand\n2. Model\n3. Description\n4. Cars Remaining"
        detail_idx = get_user_int_range("\nChoose car detail (1-4): ", 1, 4) - 1

        IF detail_idx != 3 THEN
            new_info = get_user_not_empty("\nEnter new car detail: ", "Nothing is entered. Please try again!\n")
        ELSE
            new_info = get_user_int("\nEnter new car remaining: ")
        ENDIF

        car_details[car_idx][detail_idx] = new_info AS STRING
        modification = car_details[car_idx]
        modification = join_util(modification, "|")

        modify_file_info(CARS_FILE, car_idx, True, modification)

    ELSE
        DISPLAY "\nWhich price detail you want to modify?"
        DISPLAY "\n1. Hourly Price\n2. Daily Price"
        detail_idx = get_user_int_range("\nChoose price detail(1/2): ", 1, 2) - 1
        new_info = get_user_float("\nEnter new car rent price: ")

        price_details[car_idx][detail_idx] = new_info AS STRING
        modification = price_details[car_idx]
        modification = join_util(modification, "|")

        modify_file_info(CARS_FILE, car_idx, False, modification)
    ENDIF
END

```

*Pseudo 14: modify\_car\_details*

```

FUNCTION display_records()
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDOLOOP

    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDOLOOP

    usernames, _ = get_file_info(CUSTOMERS_FILE)

    DISPLAY "\nRented cars:"

    LOOP customer_idx FROM 0 TO len(car_indices)
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]
        IF len(indices_details) <= 0 THEN CONTINUE
        ENDIF

        DISPLAY "\nCars rented by: ", usernames[customer_idx] + "\n"
        rent_idx = 0

        LOOP history_idx FROM 0 TO len(indices_details)
            IF date_details[history_idx] != "-" THEN
                rent_idx += 1
                rent_idx_str = rent_idx + "."
                rent_details = SPLIT indices_details[history_idx] BY ","
                rent_car_date = SPLIT date_details[history_idx] BY ","

                LOOP i FROM 0 TO len(rent_car_date)
                    CONVERT rent_car_date[i] INTO INTEGER
                ENDOLOOP

                rent_car_date = datetime.datetime(

                    rent_car_date[0], rent_car_date[1],
                    rent_car_date[2], rent_car_date[3],
                    rent_car_date[4], rent_car_date[5]
                )
                rent_car_details = car_details[rent_details[0] AS INTEGER]
                DISPLAY rent_idx_str + "Car: ", rent_car_details[0], rent_car_details[1]
                DISPLAY "Rented on:", rent_car_date
                IF rent_details[2] == "D" then
                    DISPLAY "Rented for: ", rent_details[1], "day(s)"
                ELSE
                    DISPLAY "Rented for: ", rent_details[1], "hour(s)"
                ENDIF

                IF rent_details[3] AS INTEGER == 1 THEN
                    DISPLAY "Status: Returned"
                ELSE
                    DISPLAY "Status: Not returned"
                ENDIF
            ENDIF
        ENDLOOP
    ENDLOOP

    DISPLAY "Booked cars:"
    LOOP customer_idx FROM 0 TO len(car_indices)
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]

        DISPLAY "\nCars booked by: ", usernames[customer_idx] + "\n"
        rent_idx = 0
        LOOP history_idx FROM 0 TO len(indices_details)
            IF date_details[history_idx] == "-" THEN
                rent_idx += 1
                rent_idx_str = rent_idx + ". "
                rent_details = SPLIT indices_details[history_idx] BY ","
                rent_car_details = car_details[rent_details[0] AS INTEGER]
                DISPLAY rent_idx_str + "Car: ", rent_car_details[0], rent_car_details[1]
            ENDIF
        ENDLOOP
    ENDLOOP
    DISPLAY "Cars available for rent:\n"
    view_cars(True, True)
END

```

*Pseudo 15: display\_records*

```

FUNCTION search_records()
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    customer_names, _ = get_file_info(CUSTOMERS_FILE)
    car_details, _ = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDOLOOP

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
    ENDOLOOP

    LOOP i FROM 0 TO len(customer_names)
        DISPLAY i+1, customer_names[i]
    ENDOLOOP
    customer_idx = get_user_int_range("\nChoose a customer to be searched (1- " + len(customer_names) + "): ", 1, len(customer_names)) - 1

    DISPLAY "\nChoose your option below:"
    DISPLAY "1. Customer booking\n2. Customer payment"
    option = get_user_int_range("\nEnter option (1/2): ", 1, 2)

    IF option == 1 THEN
        DISPLAY "\nCars booked by: ", customer_names[customer_idx] + "\n"
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]

        book_idx = 0
        LOOP history_idx FROM 0 TO len(indices_details)
            IF date_details[history_idx] == "-" THEN
                book_idx += 1
                rent_idx_str = book_idx + ". "
                rent_details = SPLIT indices_details[history_idx] BY ","
                rent_car_details = car_details[rent_details[0] AS INTEGER]
                DISPLAY rent_idx_str + "Car: ", rent_car_details[0], rent_car_details[1]
            ENDIF
        ENDOLOOP
    ELSE
        DISPLAY "\nCars rent by: " + customer_names[customer_idx] + "\n"
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]

        book_idx = 0
        LOOP history_idx FROM 0 TO len(indices_details)
            IF date_details[history_idx] != "-" THEN
                book_idx += 1
                rent_idx_str = book_idx + ". "
                rent_details = SPLIT indices_details[history_idx] BY ","
                rent_car_details = car_details[rent_details[0] AS INTEGER]
                rent_car_date = SPLIT date_details[history_idx] BY ","

                LOOP i FROM 0 TO len(rent_car_date)
                    CONVERT rent_car_date[i] INTO INTEGER
                ENDOLOOP

                rent_car_date = datetime.datetime(
                    rent_car_date[0], rent_car_date[1],
                    rent_car_date[2], rent_car_date[3],
                    rent_car_date[4], rent_car_date[5]
                )
                DISPLAY rent_idx_str + "Car: ", rent_car_details[0], rent_car_details[1]
                DISPLAY rent_idx_str + "Rented on: ", rent_car_date
                IF rent_details[2] == "D" THEN
                    DISPLAY "Rented for: ", rent_details[1], "day(s)"
                ELSE
                    DISPLAY "Rented for: ", rent_details[1], "hour(s)"
                ENDIF
            ENDIF
        ENDOLOOP
    ENDIF
END

```

*Pseudo 16: search\_records*

```

FUNCTION return_rented_cars()
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDLOOP

    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDLOOP

    usernames, _ = get_file_info(CUSTOMERS_FILE)

    LOOP i FROM 0 TO len(usernames)
        DISPLAY i+1, usernames[i]
    ENDLOOP

    customer_idx = get_user_int_range("Select a customer (1-" + len(usernames) + "): ", 1, len(usernames)) - 1

    date_details = dates[customer_idx]
    indices_details = car_indices[customer_idx]
    unreturned_history_indices = []
    unreturned_car_indices = []
    unreturned_car_dates = []
    LOOP history_idx FROM 0 TO len(indices_details)
        IF date_details[history_idx] != "-" THEN
            rent_details = SPLIT indices_details[history_idx] BY ","
            rent_car_date = SPLIT date_details[history_idx] BY ","

            LOOP i FROM 0 TO len(rent_car_date)
                CONVERT rent_car_date[i] INTO INTEGER
            ENDLOOP

            rent_car_date = datetime.datetime(
                rent_car_date[0], rent_car_date[1],
                rent_car_date[2], rent_car_date[3],
                rent_car_date[4], rent_car_date[5]
            )
            IF rent_details[3] != "1" THEN
                APPEND history_idx INTO unreturned_history_indices
                APPEND (rent_details[0] AS INTEGER) INTO unreturned_car_indices
                APPEND (rent_car_date AS STRING) INTO unreturned_car_dates
            ENDIF
        ENDIF
    ENDLOOP
    IF len(unreturned_history_indices) > 0 THEN
        DISPLAY "\nCars that are not returned by: ", usernames[customer_idx]
        LOOP i FROM 0 TO len(unreturned_car_indices)
            unreturned_car_detail = car_details[unreturned_car_indices[i]]
            DISPLAY i+1, unreturned_car_detail[0], unreturned_car_detail[1]
            DISPLAY "Rented on: ", unreturned_car_dates[i]
        ENDLOOP
        return_car_idx = get_user_int_range("\nChoose a car to return (1-" + len(unreturned_history_indices) + "): ", 1, len(unreturned_history_indices)) - 1
        rent_details = SPLIT indices_details[unreturned_history_indices[return_car_idx]] BY ","
        rent_details[3] = "1"
        indices_details[unreturned_history_indices[return_car_idx]] = join_util(rent_details, ",")

        modify_file_info(CUSTOMER_RENTS_FILE, customer_idx, True, join_util(indices_details, "|"))
    ELSE
        DISPLAY "There are no cars to return from ", usernames[customer_idx]
    ENDIF
    DISPLAY "Car returned, press enter to continue..."
    READ
END

```

*Pseudo 17: return\_rented\_cars*

```

FUNCTION admin()
BEGIN
    os.system("cls")
    admin_list = [
        "ADMIN",
        "\n1. Add a new car.",
        "2. Modify car details.",
        "3. Display records",
        "4. Search specific records",
        "5. Return a rented car.",
        "6. Return to main menu.\n"
    ]
    admin_func = [add_new_car, modify_car_details, display_records, search_records, return_rented_cars]

    usernames, passwords = get_file_info(ADMINS_FILE)
    login(usernames, passwords)
    DOWHILE True
        os.system("cls")
        LOOP i FROM 0 TO len(admin_list)
            DISPLAY admin_list[i]
        no = get_user_int_range("Choose option (1-6): ", 1, 6)
        IF no == 6 THEN RETURN
        ENDIF
        admin_func[no-1]()
    ENDWHILE
END

```

*Pseudo 18: admin*



## All Customers

```
FUNCTION create_acc()
BEGIN
    usernames, _ = get_file_info(CUSTOMERS_FILE)
    register_new_user(CUSTOMERS_FILE, usernames)
    add_file_info(CUSTOMER_RENTS_FILE, "\n", "\n")
END
```

*Pseudo 19: create\_acc*

```
FUNCTION view_cars(view_available=True, wait=True)
BEGIN
    cars, price = get_file_info(CARS_FILE)
    available_car_indices = []
    LOOP i FROM 0 TO len(cars)
        car_detail = cars[i] SPLIT("|")
        price_detail = price[i] SPLIT("|")
        IF view_available AND car_detail[3] AS INTEGER <= 0 THEN CONTINUE
        ENDIF
        APPEND (i) INTO available_car_indices
        DISPLAY "Car Index: ", i+1
        DISPLAY "Car: ", car_detail[0], car_detail[1]
        DISPLAY "Description: ", car_detail[2]
        DISPLAY "Cars Remaining: ", car_detail[3]
        DISPLAY "Hourly Price: ", price_detail[0]
        DISPLAY "Daily Price: ", price_detail[1] + "\n"
    ENDLLOOP

    IF wait THEN
        DISPLAY "Press enter to continue..."
        READ
    ENDIF
    RETURN available_car_indices
END
```

*Pseudo 20: view\_cars*

```
FUNCTION all_customer()
BEGIN
    os.system("cls")
    customer_list = [
        "ALL CUSTOMER",
        "\n1. View all cars available for rent.",
        "2. Create new account.",
        "3. Exit to main menu\n"
    ]
    customer_func = [view_cars, create_acc]

    DOWHILE True
        os.system("cls")
        LOOP i FROM 0 TO len(customer_list)
            DISPLAY customer_list[i]
        ENDLLOOP
        no = get_user_int_range("Choose option (1-3): ", 1, 3)
        IF no == 3 THEN RETURN
        ENDIF
        customer_func[no-1]()
    ENDWHILE
END
```

*Pseudo 21: all\_customer*

## Registered Customers

```
FUNCTION modify_personal_details(curr_user_idx)
BEGIN
    DISPLAY "\n1. Username\n2. Password\n3. Cancel"
    selection = get_user_int_range("Choose option (1-3): ", 1, 3, "Select 1 or 2 only!\n")
    IF selection == 3 THEN RETURN
    ENDIF

    IF selection == 1 THEN
        new_username = get_user_not_empty("New username: ")
        modify_file_info(CUSTOMERS_FILE, curr_user_idx, True, new_username)
    ELSE
        new_password = get_user_not_empty("New password: ")
        modify_file_info(CUSTOMERS_FILE, curr_user_idx, False, new_password)
    ENDIF
END
```

*Pseudo 22: modify\_personal\_details*

```
FUNCTION view_history(curr_user_idx)
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDLOOP

    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDLOOP

    DISPLAY "\nRented cars:\n"
    rented_cars = []

    date_details = dates[curr_user_idx]
    indices_details = car_indices[curr_user_idx]

    rent_idx = 0
    LOOP i FROM 0 TO len(date_details)
        IF date_details[i] != "-" THEN
            rent_idx += 1
            rent_idx_str = rent_idx + ". "
            rent_details = SPLIT indices_details[i] BY ","
            rent_car_date = SPLIT date_details[i] BY ","

            LOOP j FROM 0 TO len(rent_car_date)
                CONVERT rent_car_date[j] INTO INTEGER
            ENDLOOP

            rent_car_date = datetime.datetime(
                rent_car_date[0], rent_car_date[1],
                rent_car_date[2], rent_car_date[3],
                rent_car_date[4], rent_car_date[5]
            )
            rent_car_details = car_details[rent_details[0] AS INTEGER]
            DISPLAY rent_idx_str + "Car: ", rent_car_details[0] + ", ", rent_car_details[1]
            DISPLAY "Rented on: ", rent_car_date
            IF rent_details[3] AS INTEGER == 1 THEN
                DISPLAY "Status: Returned"
            ELSE
                DISPLAY "Status: Not returned"
            ENDIF
        ENDIF
    ENDLOOP
    DISPLAY "Press enter to continue..."
    READ
END
```

*Pseudo 23: view\_history*

```

FUNCTION view_booked_cars(curr_user_idx)
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDLLOOP

    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDLLOOP

    DISPLAY "\nCars to be rented:\n"
    rented_cars = []

    date_details = dates[curr_user_idx]
    indices_details = car_indices[curr_user_idx]

    LOOP i FROM 0 TO len(date_details)
        IF date_details[i] == "-" THEN
            rent_details = SPLIT indices_details[i] BY ","
            rent_car_details = car_details[rent_details[0] AS INTEGER]
            rent_car_price = price_details[rent_details[0] AS INTEGER]
            DISPLAY "Car: ", rent_car_details[0], rent_car_details[1]
            DISPLAY "Description: ", rent_car_details[2]
            IF rent_details[2] == "H" THEN
                DISPLAY "Booked for ", rent_details[1], "hours."
                total_price = rent_car_price[0] AS FLOAT*rent_details[1] AS INTEGER
            ELSE
                DISPLAY "Booked for ", rent_details[1], "days."
                total_price = rent_car_price[1] AS FLOAT*rent_details[1] AS INTEGER
            ENDIF
            DISPLAY "Total Price: ", "RM", total_price + "\n"
        ENDIF
    ENDLLOOP

    DISPLAY "Press enter to continue..."
    READ
END

```

*Pseudo 24: view\_booked\_cars*

```

FUNCTION book_cars(curr_user_idx)
BEGIN
    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDLOOP

    available_car_indices = view_cars(True, False)

    LOOP i FROM 0 TO len(available_car_indices)
        available_car_indices[i] += 1
        CONVERT available_car_indices[i] INTO STRING
    ENDLOOP

    car_idx = get_user_selection("\nSelect car index: ", available_car_indices, "Car index is not available for rent!\n")
    car_idx = car_idx AS INTEGER - 1

    car_details[car_idx][3] = ((car_details[car_idx][3]) - 1 AS INTEGER) AS STRING
    modify_file_info(CARS_FILE, car_idx, True, join_util(car_details[car_idx], "|"))

    DISPLAY "\nDo you want to rent the car IN days or hours? Select '1' for days and '2' for hours"
    selection = get_user_int_range("\nChoose option (1/2): ", 1, 2)
    booking_result = ""
    IF selection == 1 THEN
        duration = get_user_int("How many days do you want to rent the car: ")
        booking_result = car_idx + "," + duration + ",D,0"
    ELSE
        duration = get_user_int("How many hours do you want to rent the car: ")
        booking_result = car_idx + "," + duration + ",H,0"
    ENDIF

    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDLOOP

    APPEND booking_result INTO car_indices[curr_user_idx]
    APPEND "-" INTO dates[curr_user_idx]

    new_car_indices = join_util(car_indices[curr_user_idx], "|")
    new_dates = join_util(dates[curr_user_idx], "|")

    modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, True, new_car_indices)
    modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, False, new_dates)
END

```

*Pseudo 25: book\_cars*

```

FUNCTION payment(curr_user_idx)
BEGIN
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)

    LOOP i FROM 0 TO len(car_indices)
        car_indices[i] = SPLIT car_indices[i] BY "|"
        dates[i] = SPLIT dates[i] BY "|"
    ENDLLOOP

    unpaid_car_indices = []
    LOOP i FROM 0 TO len(car_indices[curr_user_idx])
        IF dates[curr_user_idx][i] == "-" THEN
            APPEND i INTO unpaid_car_indices
        ENDIF
    ENDLLOOP
    car_details, price_details = get_file_info(CARS_FILE)

    LOOP i FROM 0 TO len(car_details)
        car_details[i] = SPLIT car_details[i] BY "|"
        price_details[i] = SPLIT price_details[i] BY "|"
    ENDLLOOP

    total_price = 0
    DISPLAY "\nCars booked:\n"
    LOOP idx FROM 0 TO len(unpaid_car_indices)
        rent_details = SPLIT car_indices[curr_user_idx][idx] BY ","
        rent_car_details = car_details[rent_details[0] AS INTEGER]
        rent_car_price = price_details[rent_details[0] AS INTEGER]
        IF rent_details[2] == "H" THEN
            rent_price = rent_car_price[0] AS FLOAT*rent_details[1] AS INTEGER
            DISPLAY rent_car_details[0], rent_car_details[1]
            DISPLAY rent_details[1], "hours", "RM", rent_price
            total_price += rent_price
        ELSE
            rent_price = rent_car_price[1] AS FLOAT*rent_details[1] AS INTEGER
            DISPLAY rent_car_details[0], rent_car_details[1]
            DISPLAY rent_details[1], "days", "RM", rent_price
            total_price += rent_price
        ENDIF
    ENDLLOOP
    DISPLAY "The total price is: ", "RM", total_price + "\n"

    DISPLAY "Type '1' to pay, '2' to cancel payment."
    pay = get_user_int_range("Choose option (1/2): ", 1, 2)
    IF pay == 1 THEN
        current_time = datetime.datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
        LOOP idx FROM 0 TO len(unpaid_car_indices)
            dates[curr_user_idx][idx] = current_time
        ENDLLOOP
        modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, False, join_util(dates[curr_user_idx], "|"))
    ENDIF
END

```

*Pseudo 26: payment*

```

FUNCTION delete_account(curr_user_idx)
BEGIN
    decision = ""
    decision = get_user_selection("Are you sure? y/n: ", ["y", "n"])

    IF decision == "y" THEN
        delete_file_info(CUSTOMERS_FILE, curr_user_idx)
        delete_file_info(CUSTOMER_RENTS_FILE, curr_user_idx)
        RETURN True
    ENDIF
    RETURN False
END

```

*Pseudo 27: delete\_account*

```

FUNCTION registered_customer()
BEGIN
    os.system("cls")
    registered_customer_list = [
        "REGISTERED CUSTOMER",
        "\n1. Modify personal details.",
        "2. View personal rental history.",
        "3. View details of cars to be rented out.",
        "4. Select and book a car for a specific duration.",
        "5. Do payment to confirm Booking.",
        "6. Delete account.",
        "7. Exit to main menu\n"
    ]
    registered_customer_func = [modify_personal_details, view_history, view_booked_cars, book_cars, payment, delete_account]

    usernames, passwords = get_file_info(CUSTOMERS_FILE)
    curr_user_idx = login(usernames, passwords)
    in_loop = True
    DOWHILE in_loop
        os.system("cls")
        LOOP i FROM 0 TO len(registered_customer_list)
            DISPLAY registered_customer_list[i]
        ENDLLOOP
        no = get_user_int("Choose option (1-7): ")
        IF no == 7 THEN RETURN
        ENDIF
        in_loop = NOT registered_customer_func[no-1](curr_user_idx)
    ENDWHILE
END

```

*Pseudo 28: registered\_customer*

## Main Program

```
FUNCTION exit_program()
BEGIN
    DISPLAY "\nDo you want to continue? To exit to the Main Menu type '1', To Terminate Program type '2': "
    no = get_user_int_range("Choose option (1/2): ", 1, 2)
    IF no == 2: exit()
ENDIF
END
```

*Pseudo 29: exit\_program*

```
FUNCTION main()
BEGIN
    user_func = [admin, all_customer, registered_customer]

    user_type_list = [
        "MAIN MENU",
        "1. Admin",
        "2. All Customers (Registered / Not-Registered)",
        "3. Registered Customer",
        "4. Exit Program\n"
    ]

    DOWHILE True
        os.system("cls")
        DISPLAY "Welcome to SUPER CAR RENTAL SERVICES!!!\n"
        LOOP i FROM 0 TO len(user_type_list)
            DISPLAY user_type_list[i]
        ENDLLOOP

        no = get_user_int_range("Choose user(1-4): ", 1, 4)
        IF no == 4 THEN exit_program()
        ELSE user_func[no-1]()
        ENDIF
    ENDWHILE
END
```

*Pseudo 30: main*

```
IMPORT datetime
IMPORT os

CUSTOMERS_FILE = "./customers.txt"
CUSTOMER_RENTS_FILE = "./customer_rents.txt"
CARS_FILE = "./cars.txt"
ADMINS_FILE = "./admins.txt"

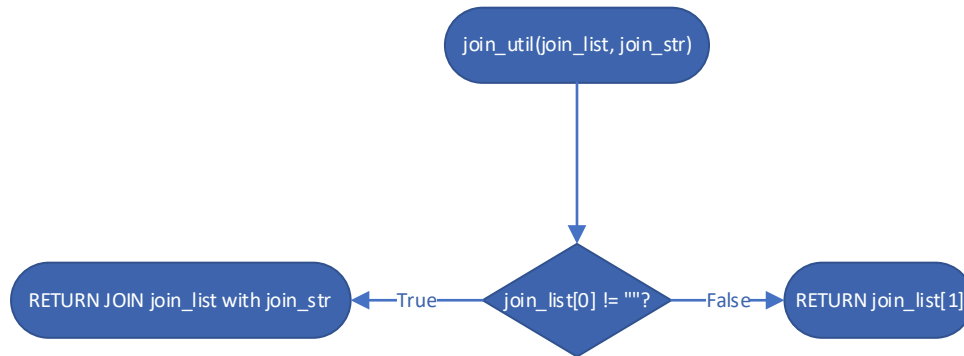
main()
```

*Pseudo 31: running the program*

## Flowchart

*Disclaimer: Some flow charts are too large to fit in, therefore it may not be in its actual size, please feel free to zoom in if needed, the resolution will not be reduced as it is in vector space.*

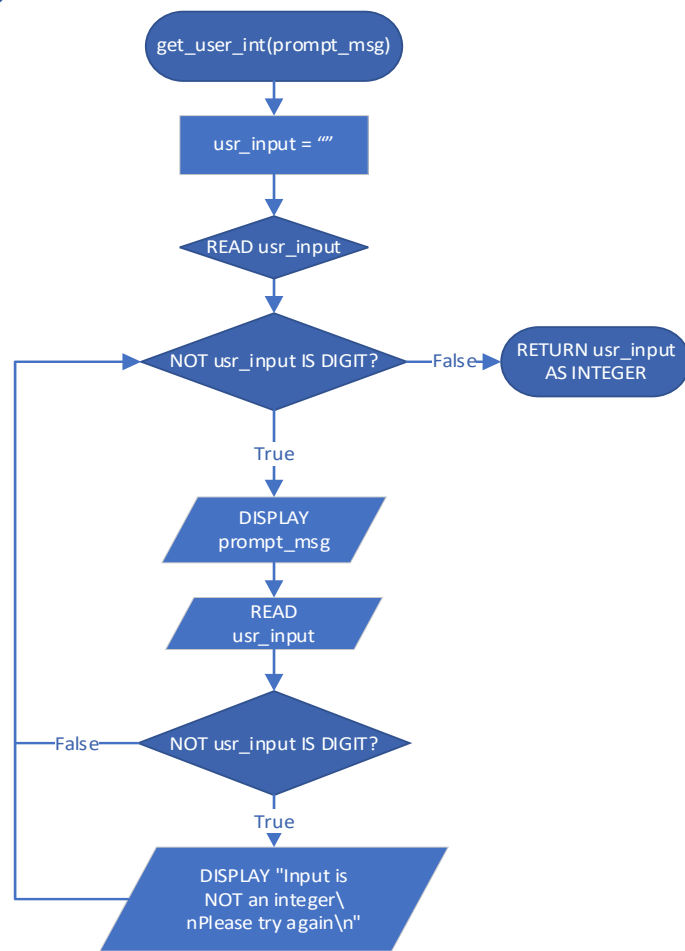
### Utility Functions



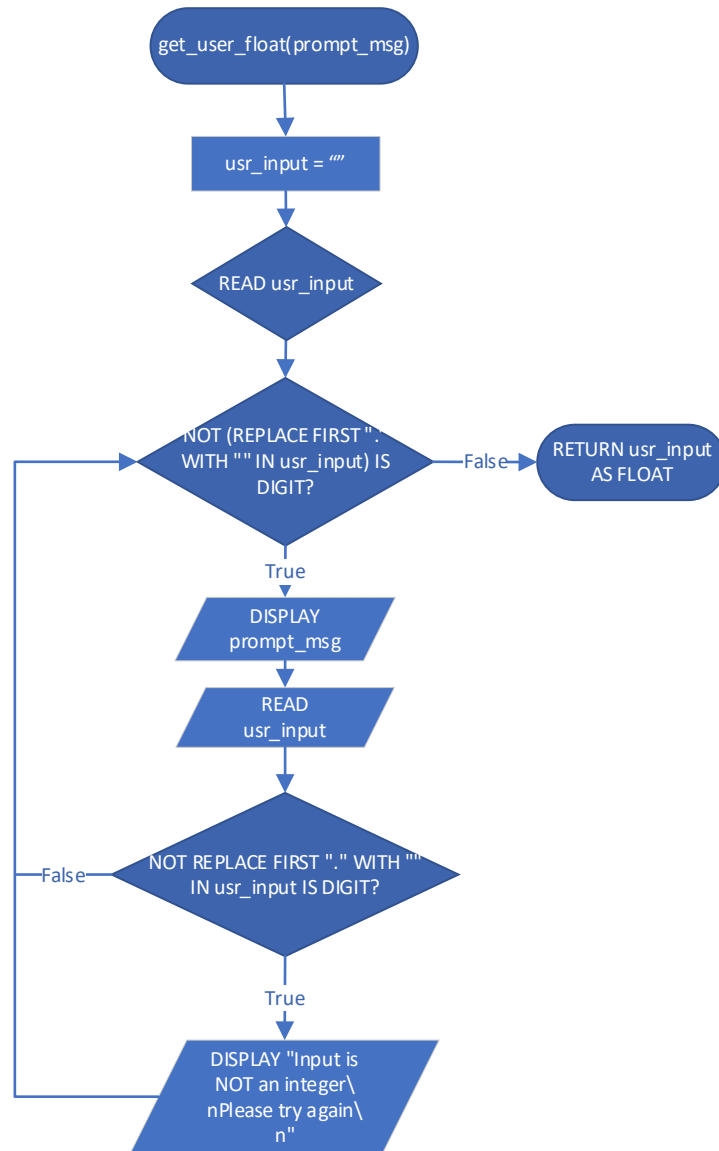
*Flowchart 1: join\_util*



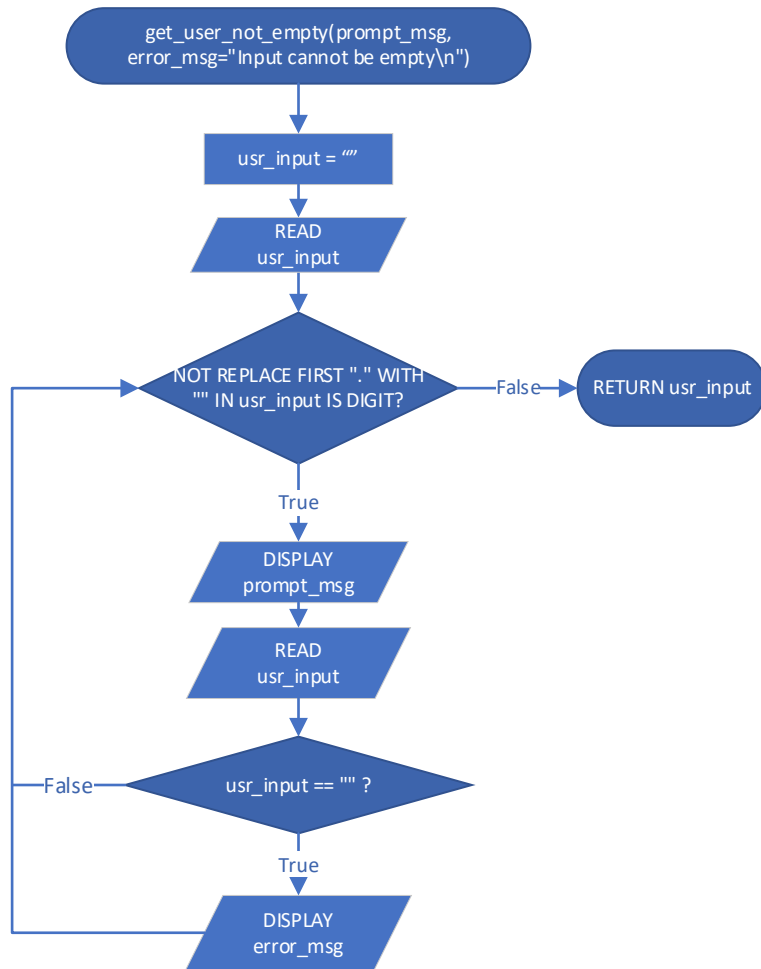
## Console Input Utility Functions



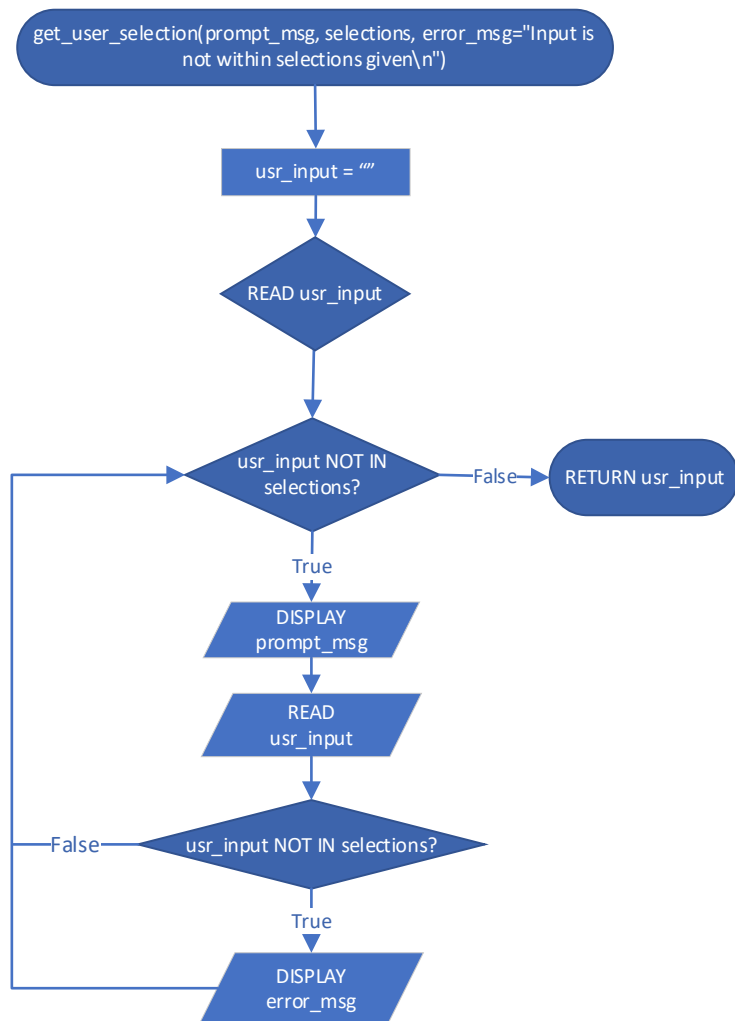
*Flowchart 2: get\_user\_int*



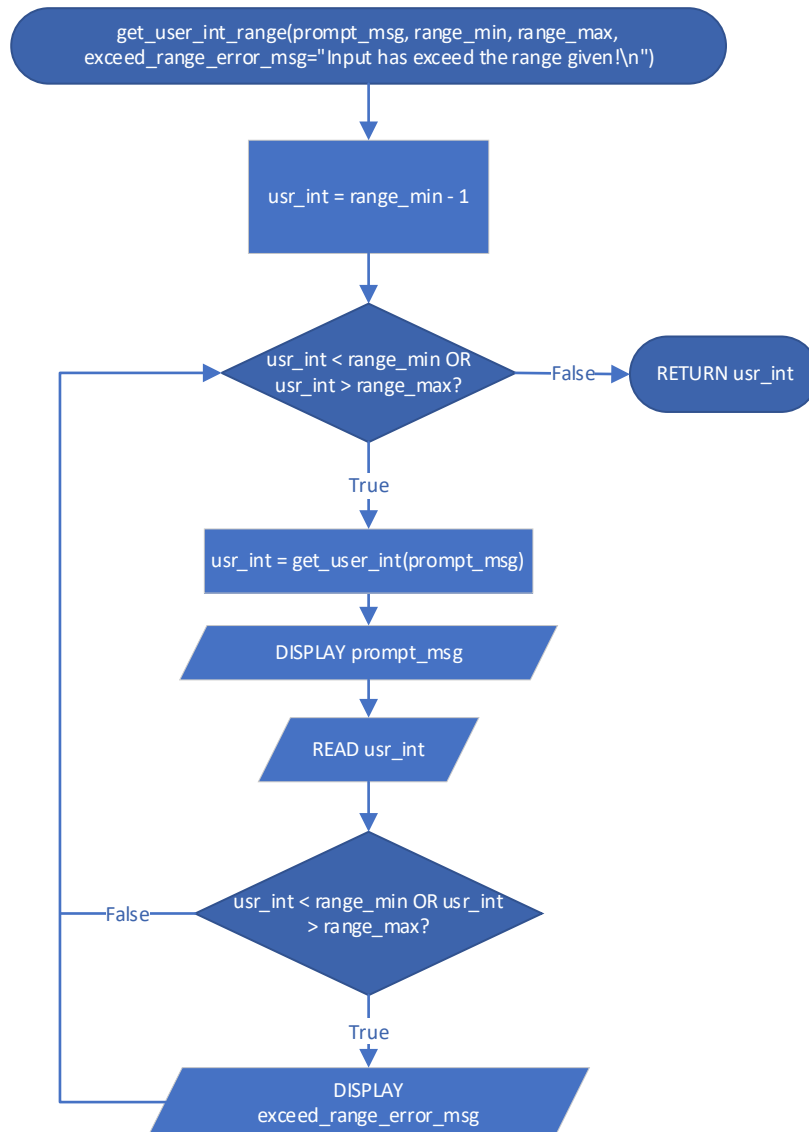
*Flowchart 3: get\_user\_float*



Flowchart 4: *get\_user\_not\_empty*

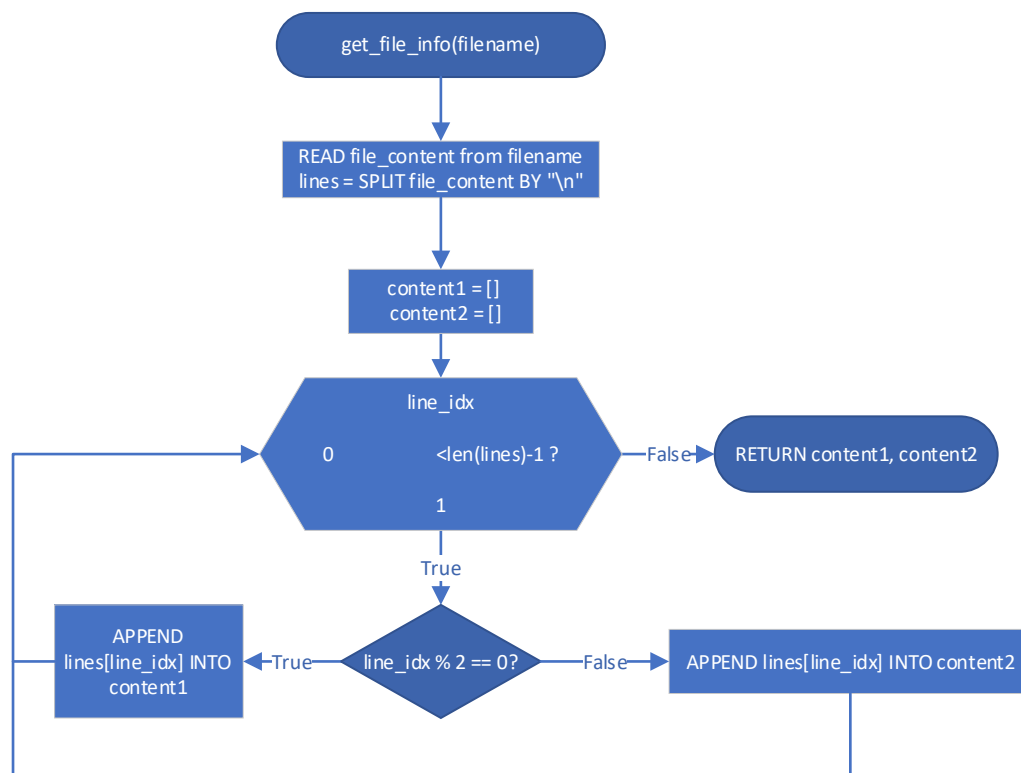


*Flowchart 5: get\_user\_selection*

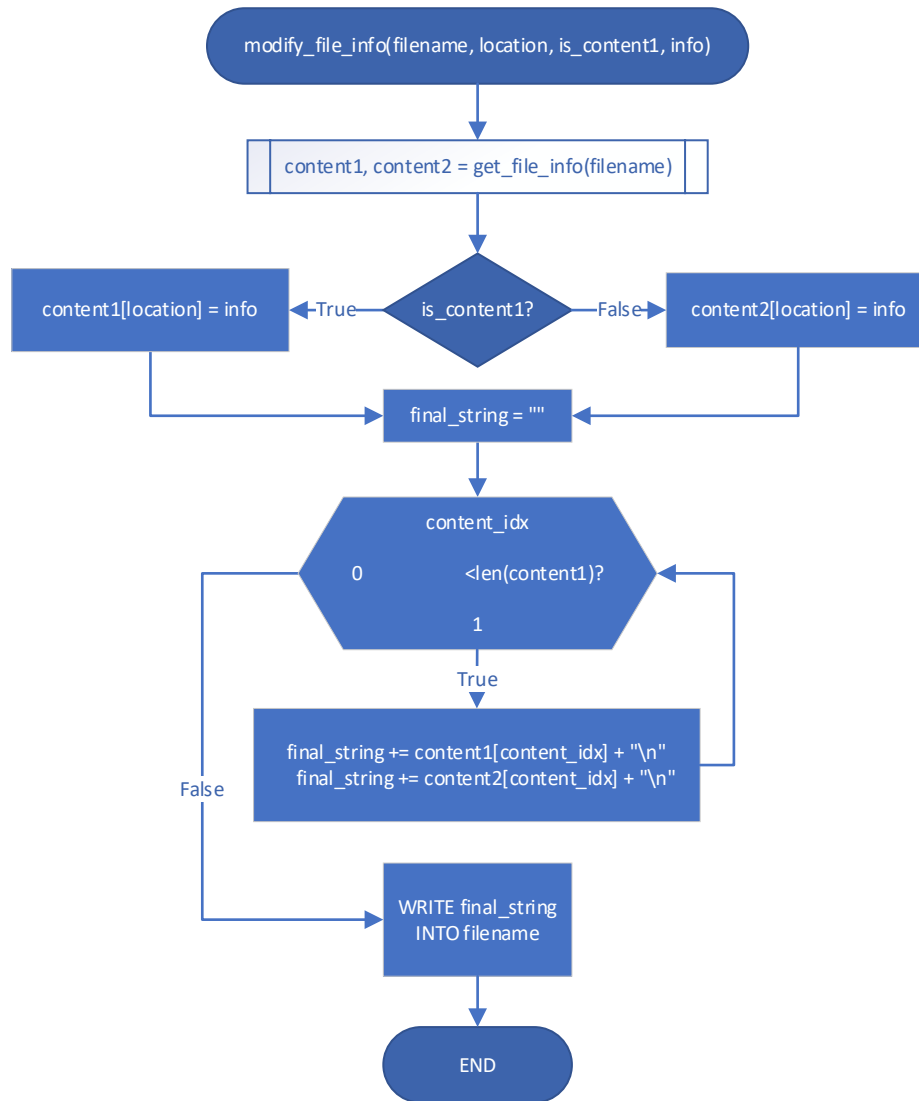


*Flowchart 6: get\_user\_int\_range*

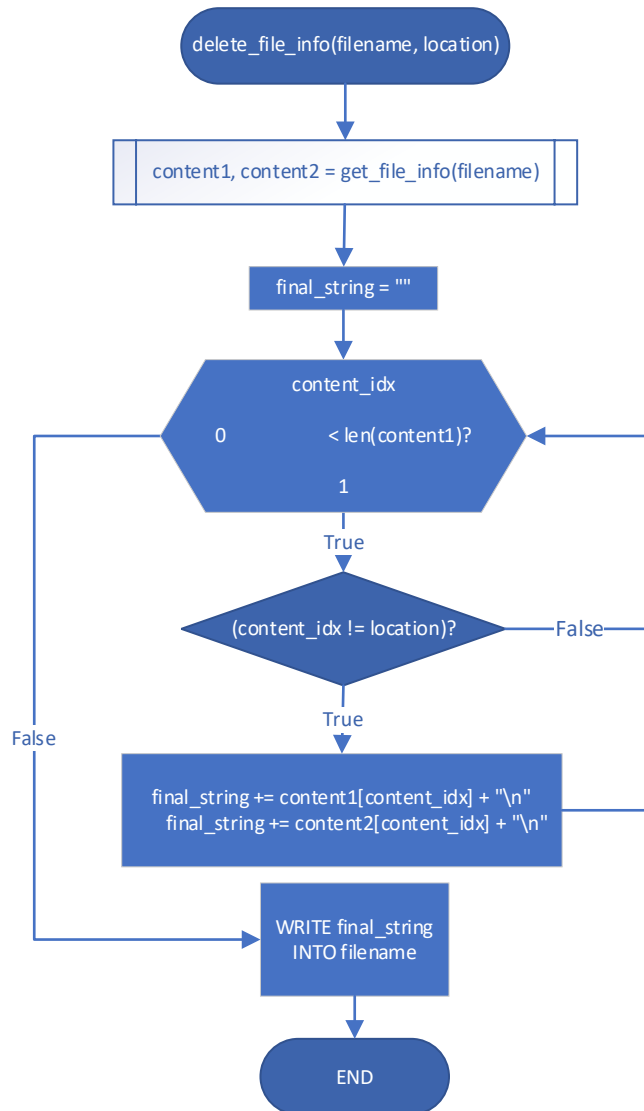
## I/O Utility Functions



Flowchart 7: *get\_file\_info*

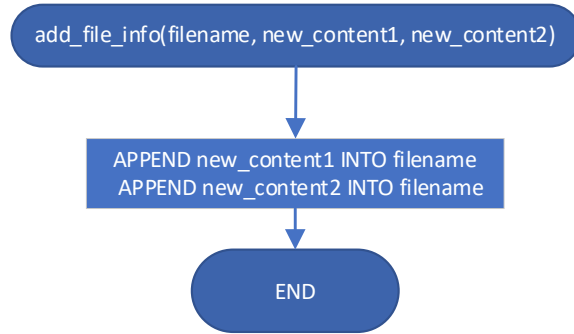


*Flowchart 8: modify\_file\_info*



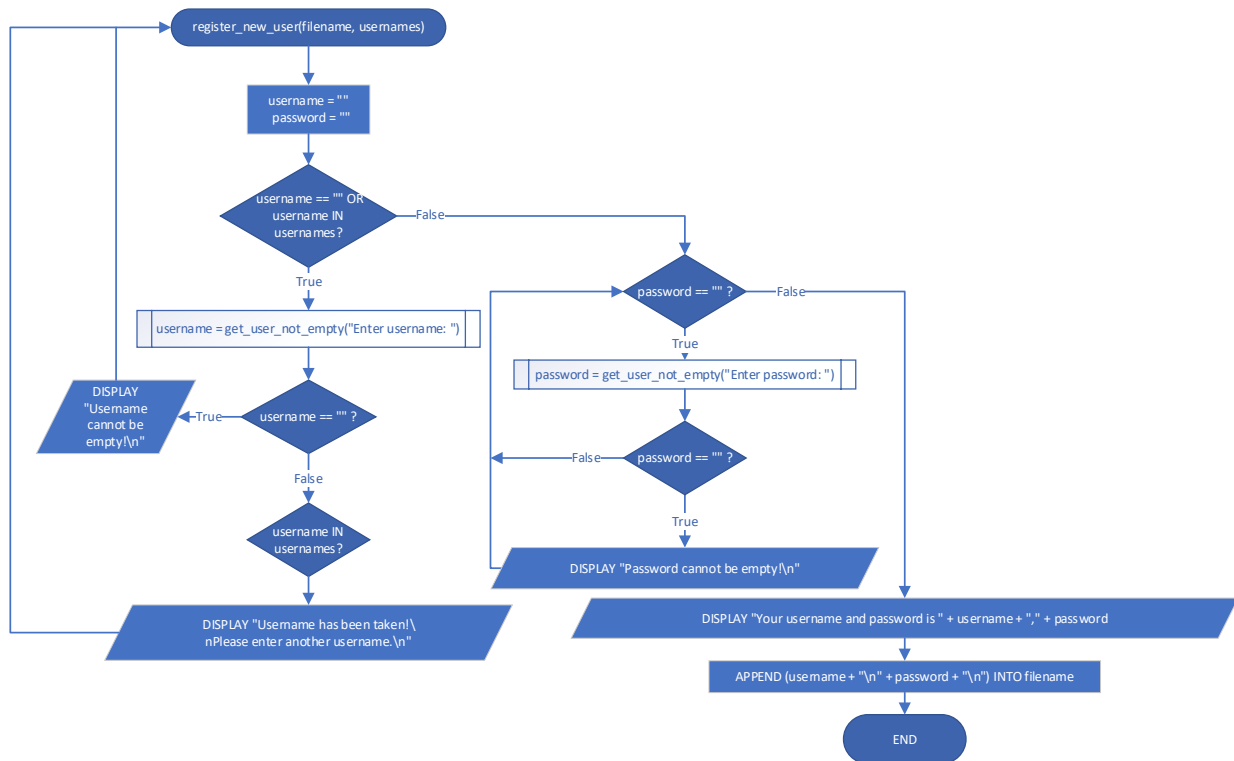
*Flowchart 9: delete\_file\_info*



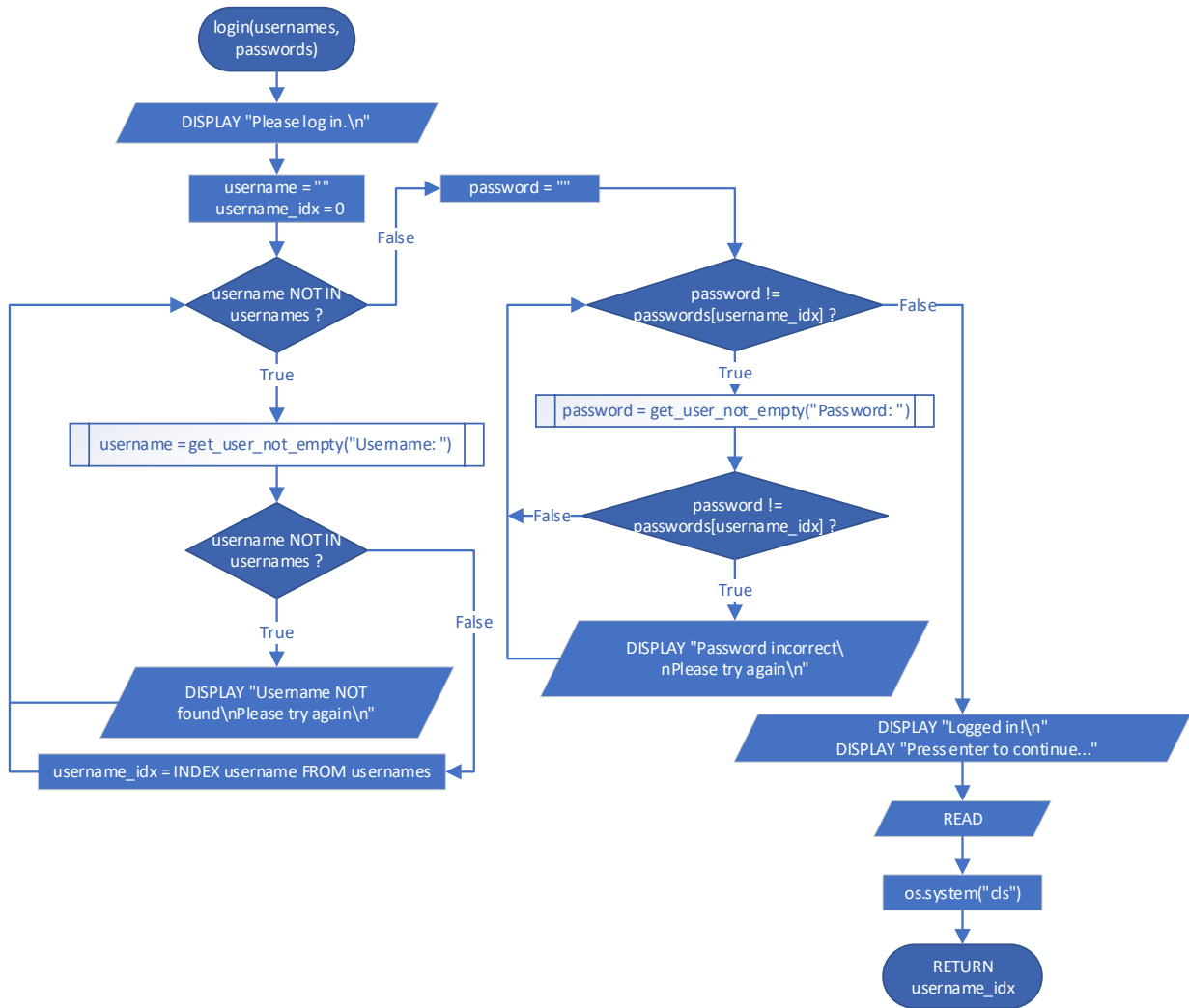


Flowchart 10: add\_file\_info

## Login System

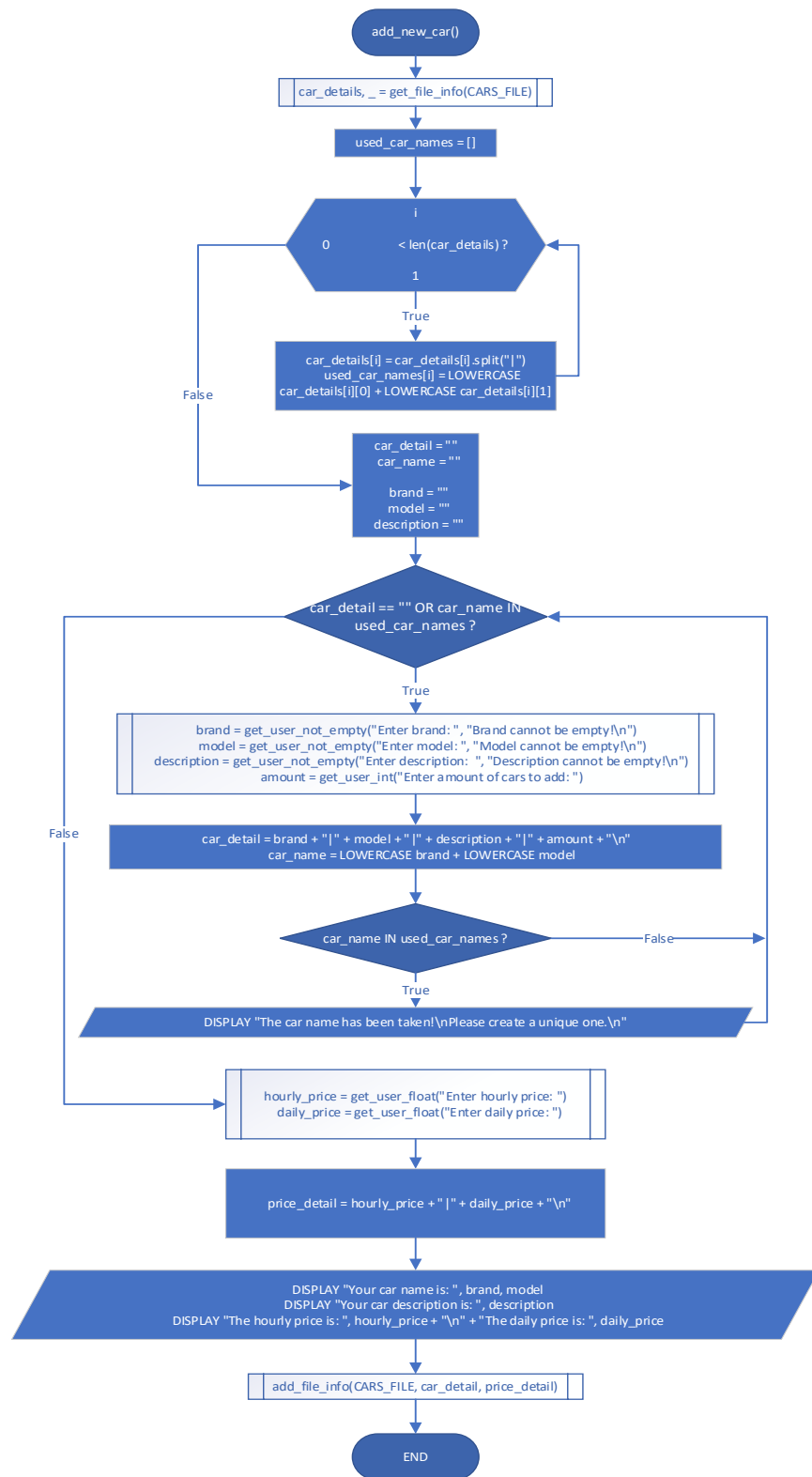


Flowchart 11: register\_new\_user

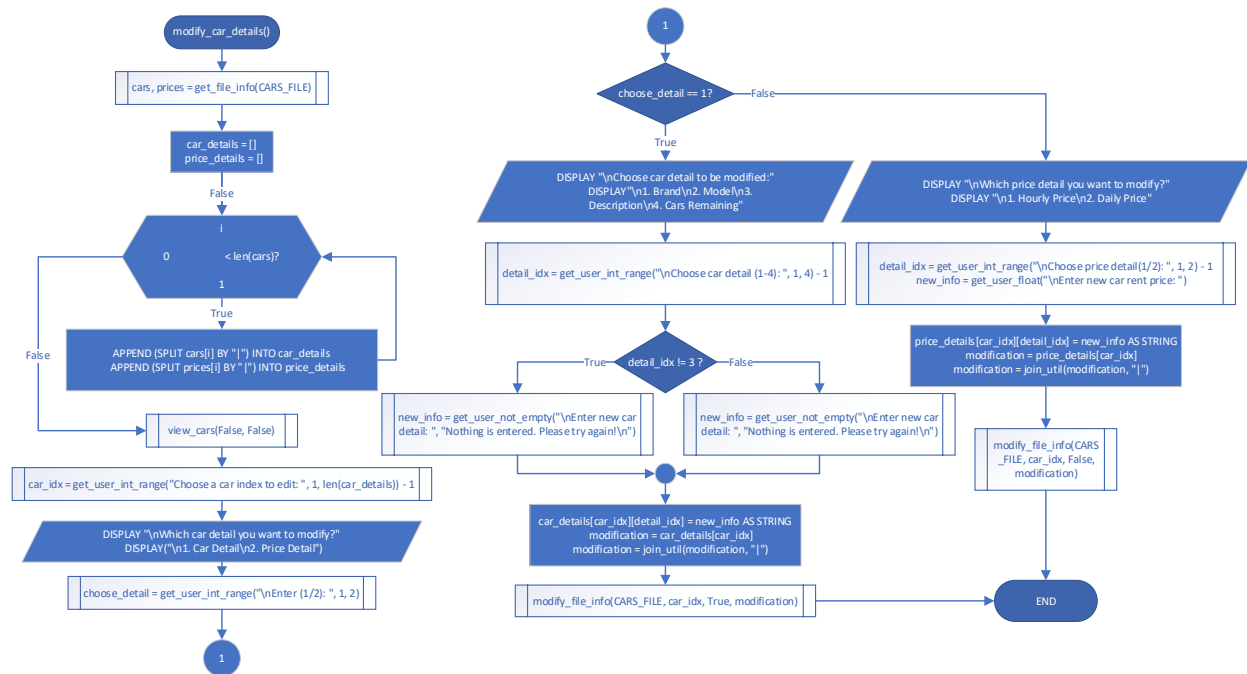


Flowchart 12: login

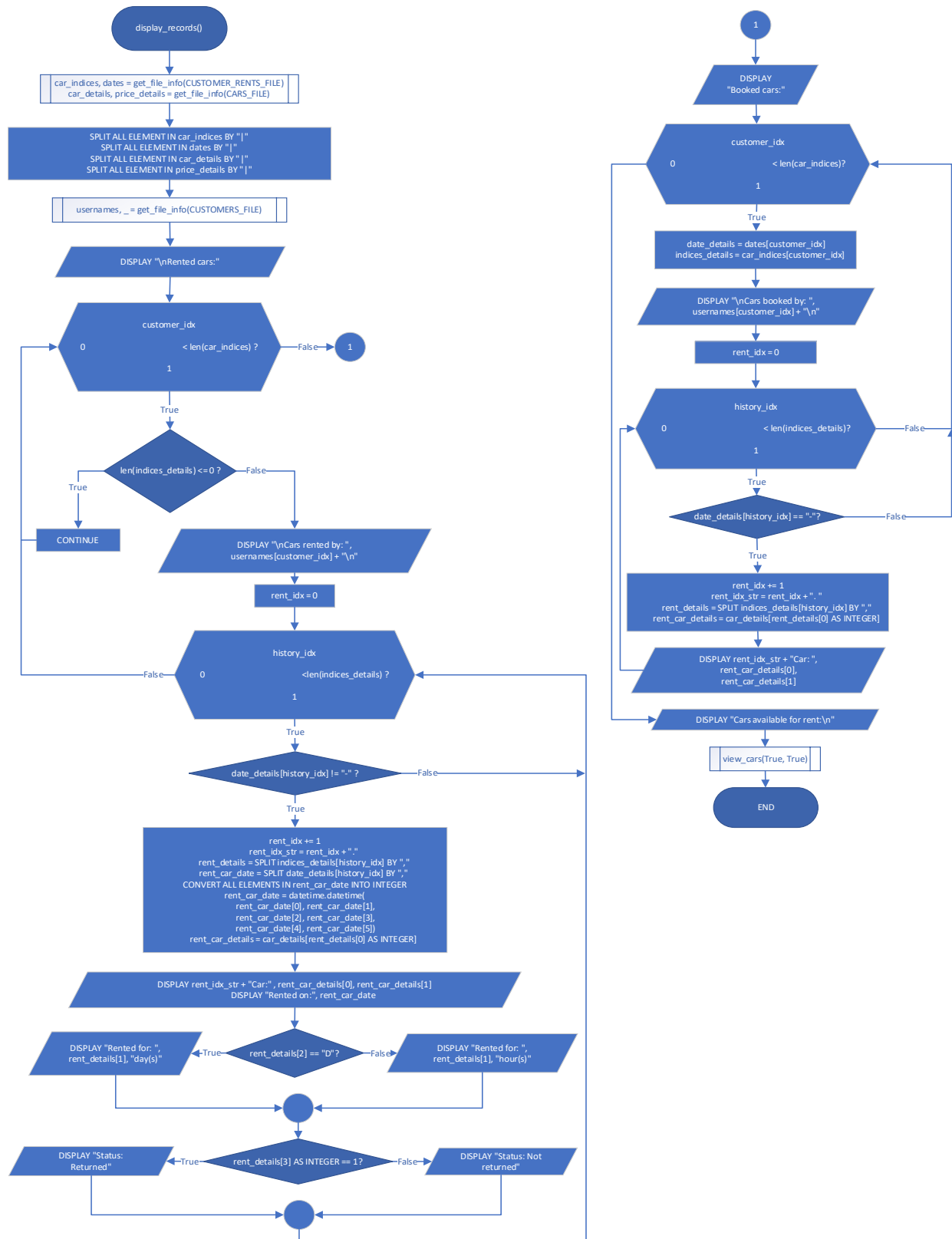
## Admin



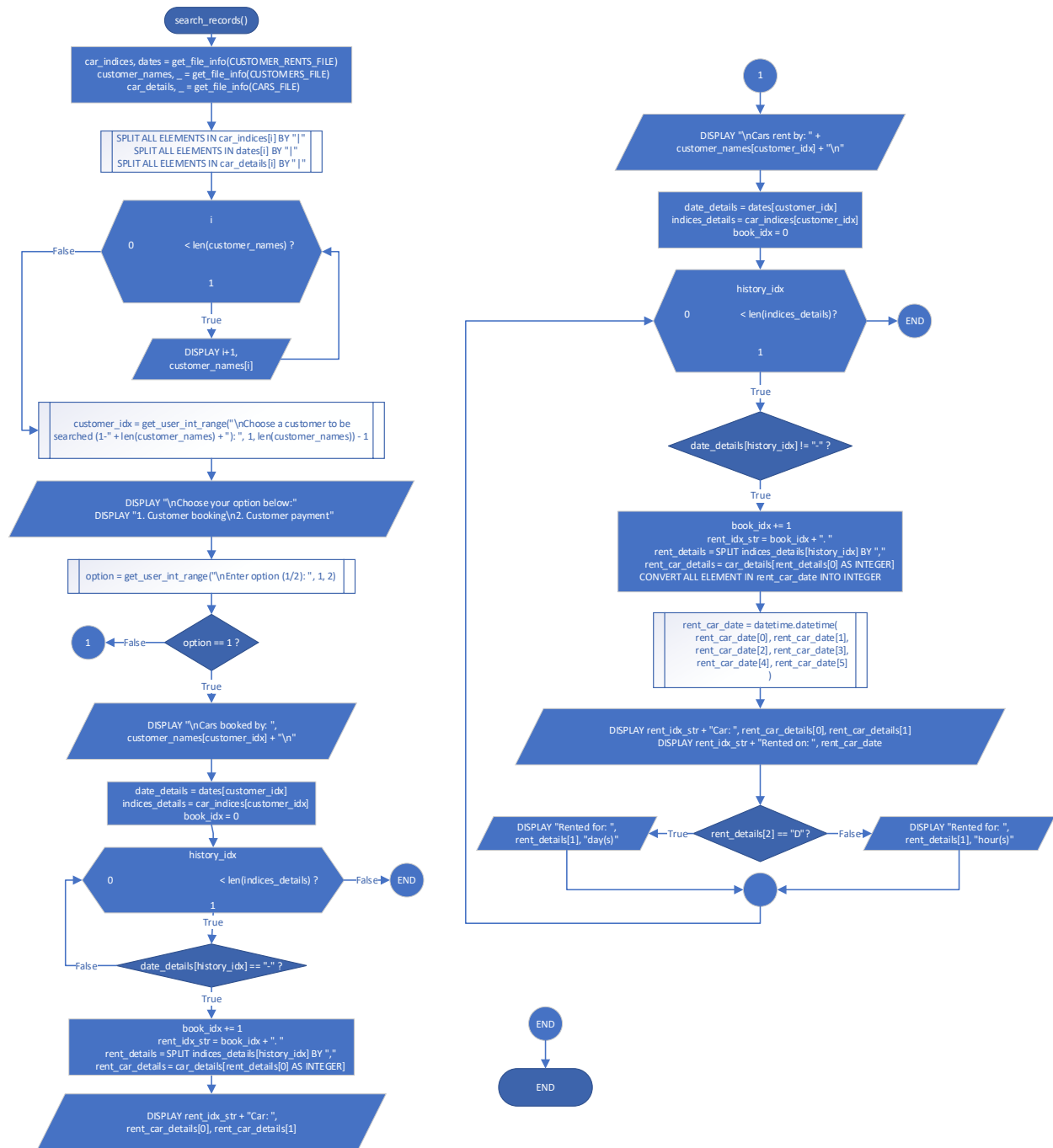
Flowchart 13: add\_new\_car



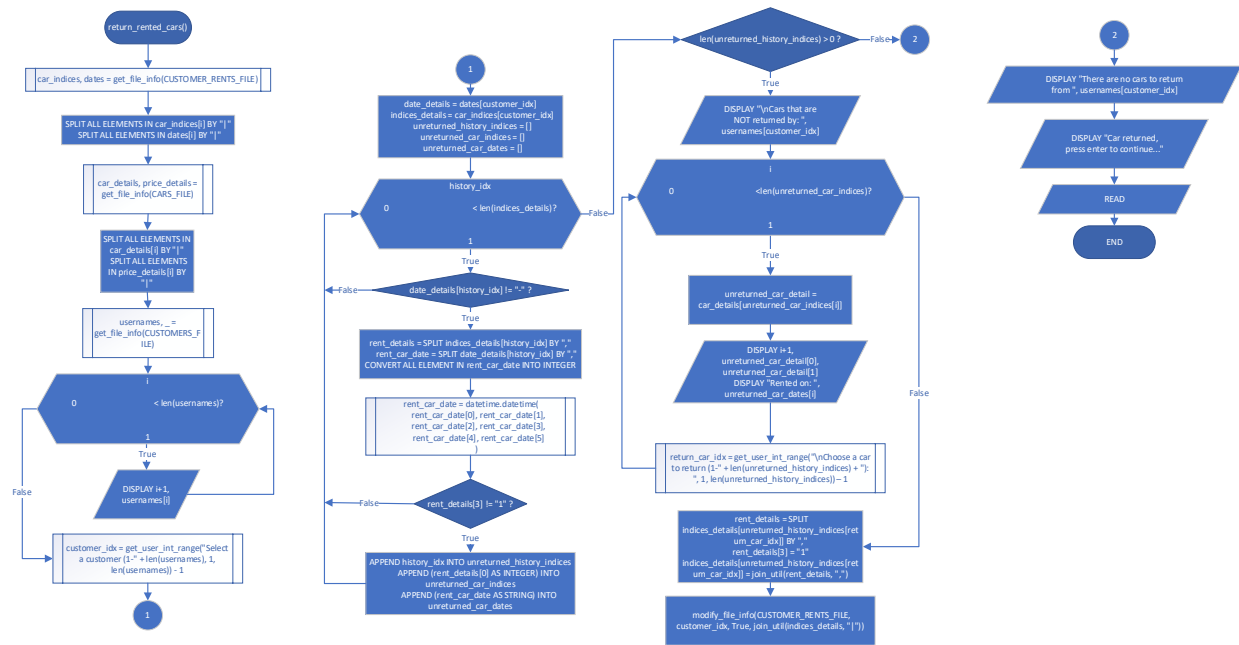
Flowchart 14: *modify\_car\_details*



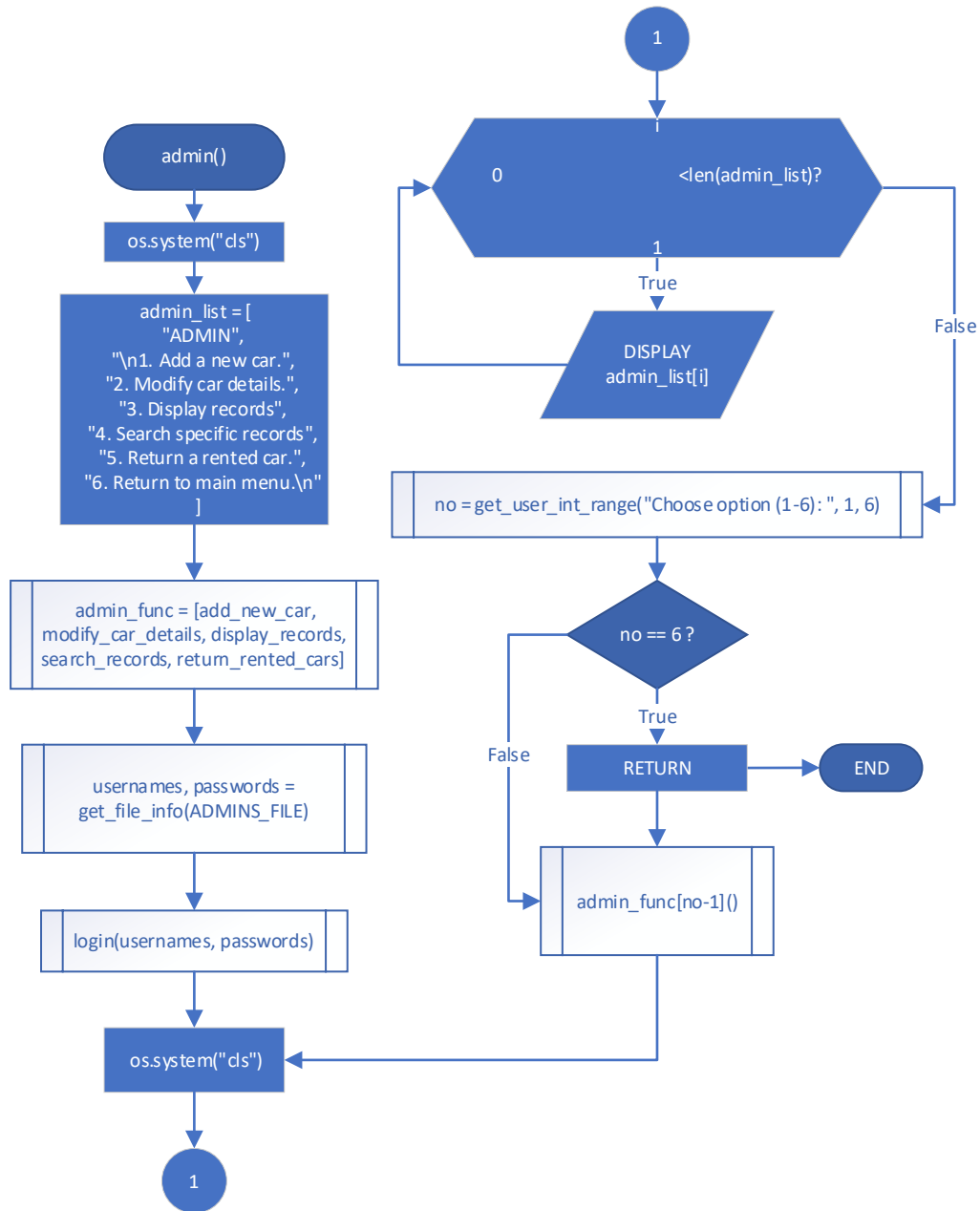
Flowchart 15: display\_records



Flowchart 16: search\_records



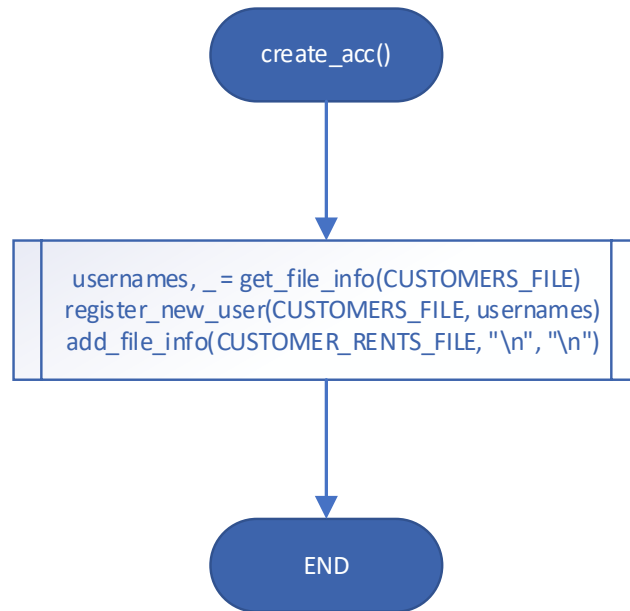
Flowchart 17: return\_rented\_cars



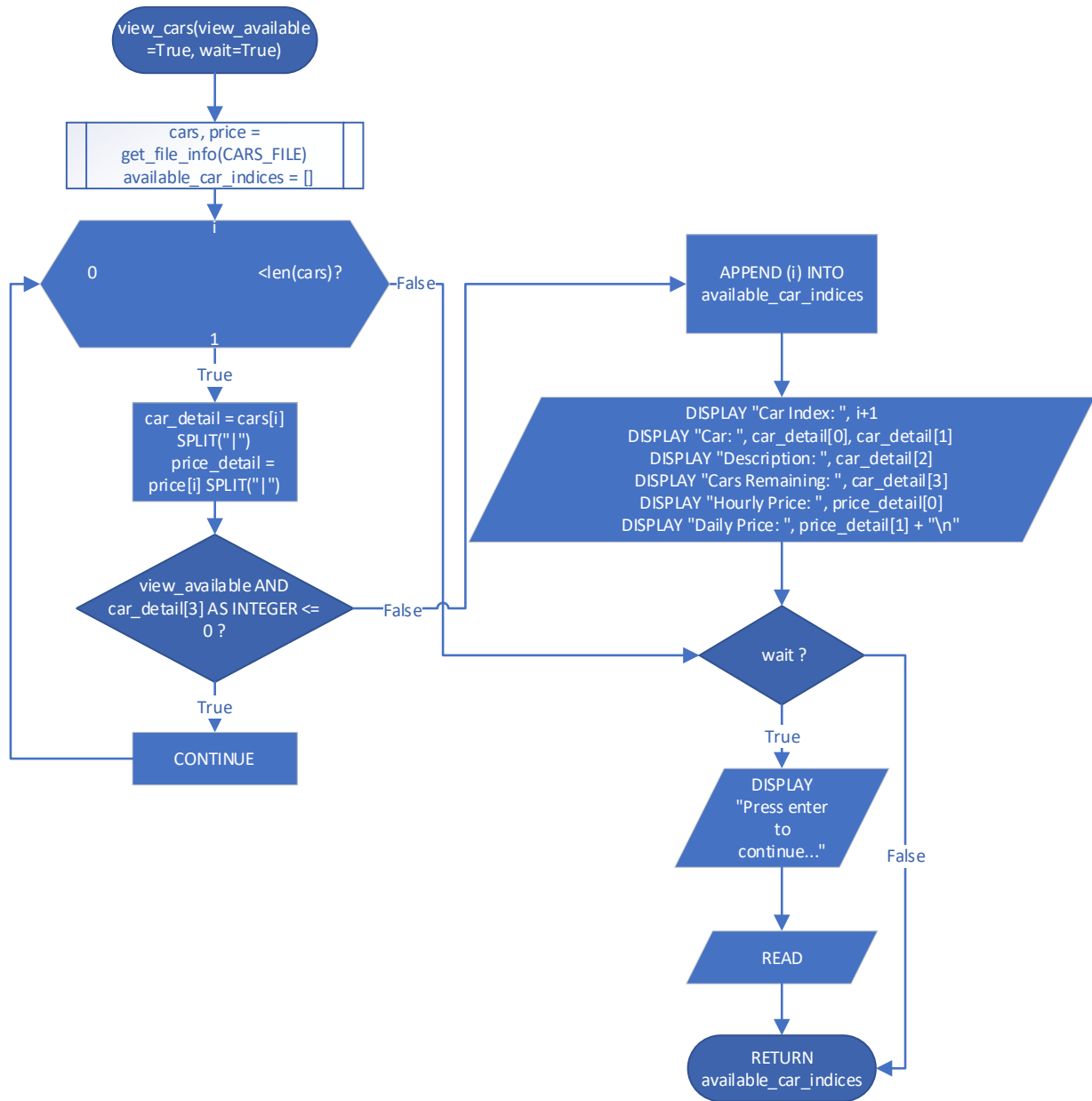
Flowchart 18: admin



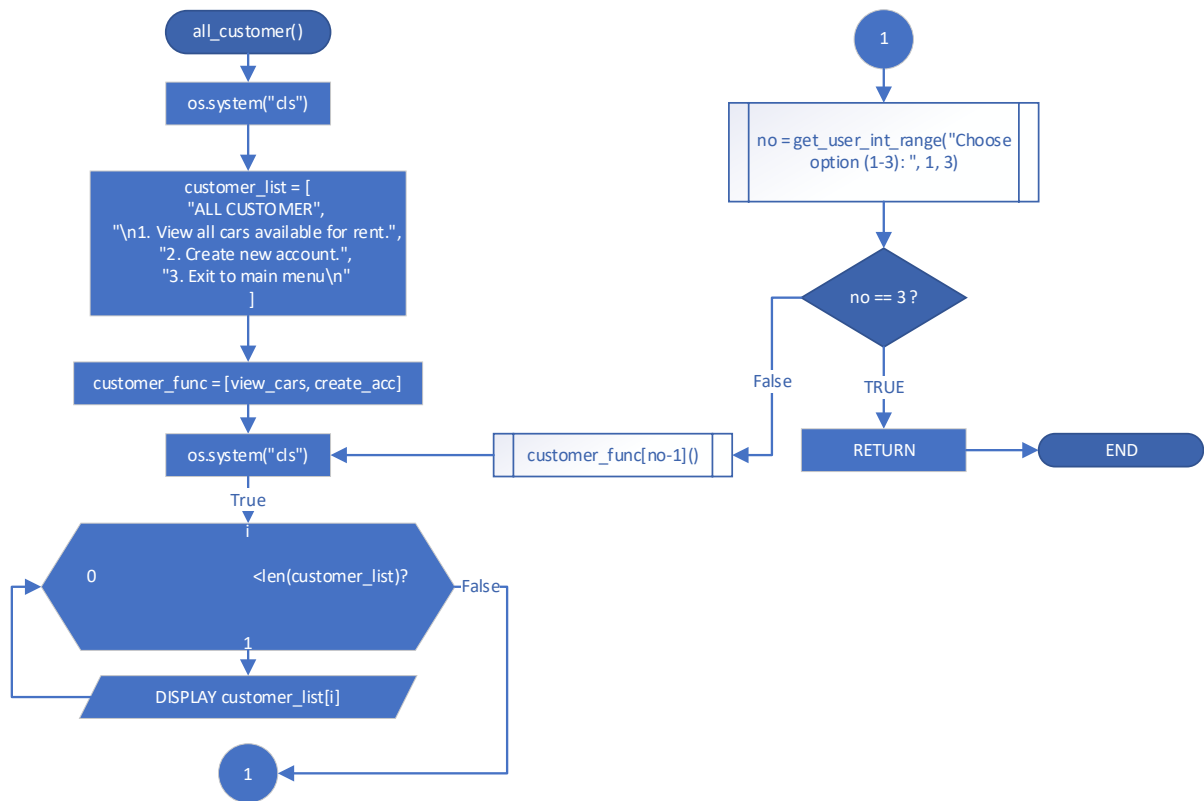
## All Customers



*Flowchart 19: create\_acc*

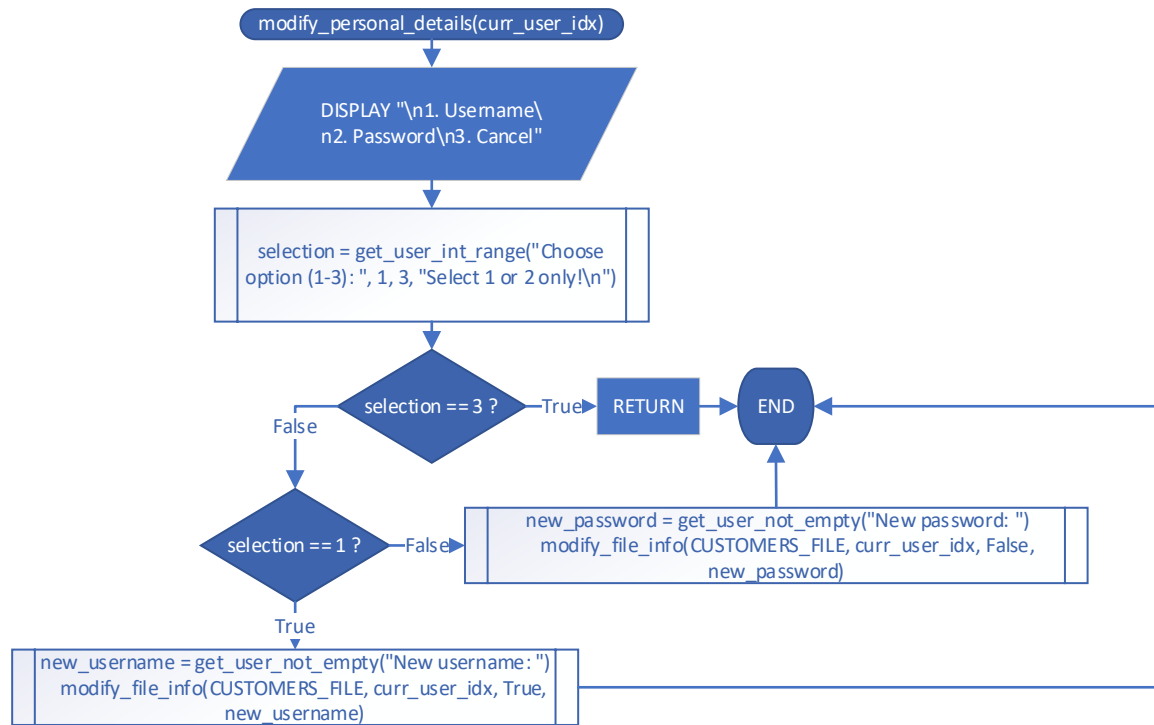


*Function 20: view\_cars*

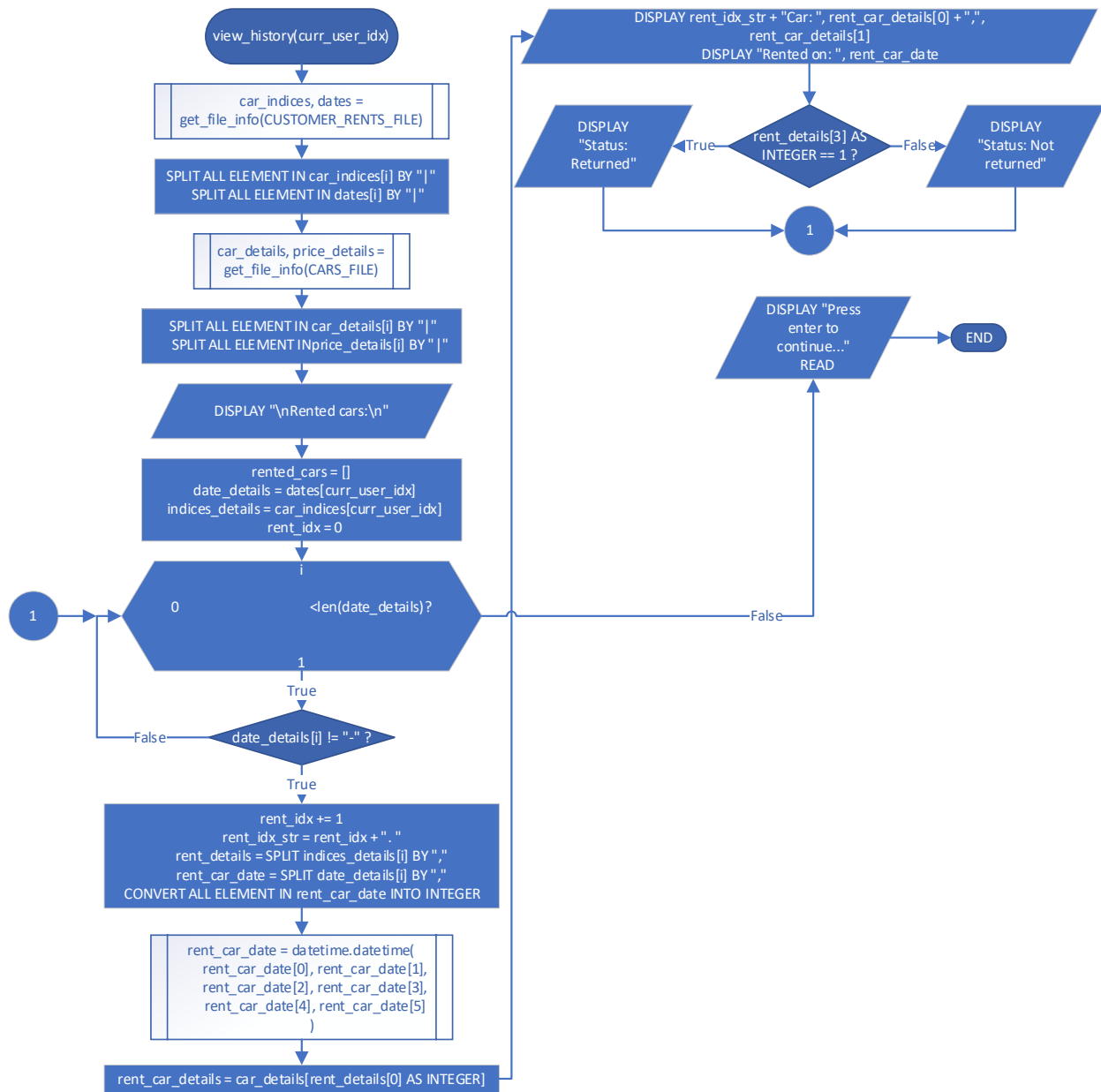


*Flowchart 21: all\_customer*

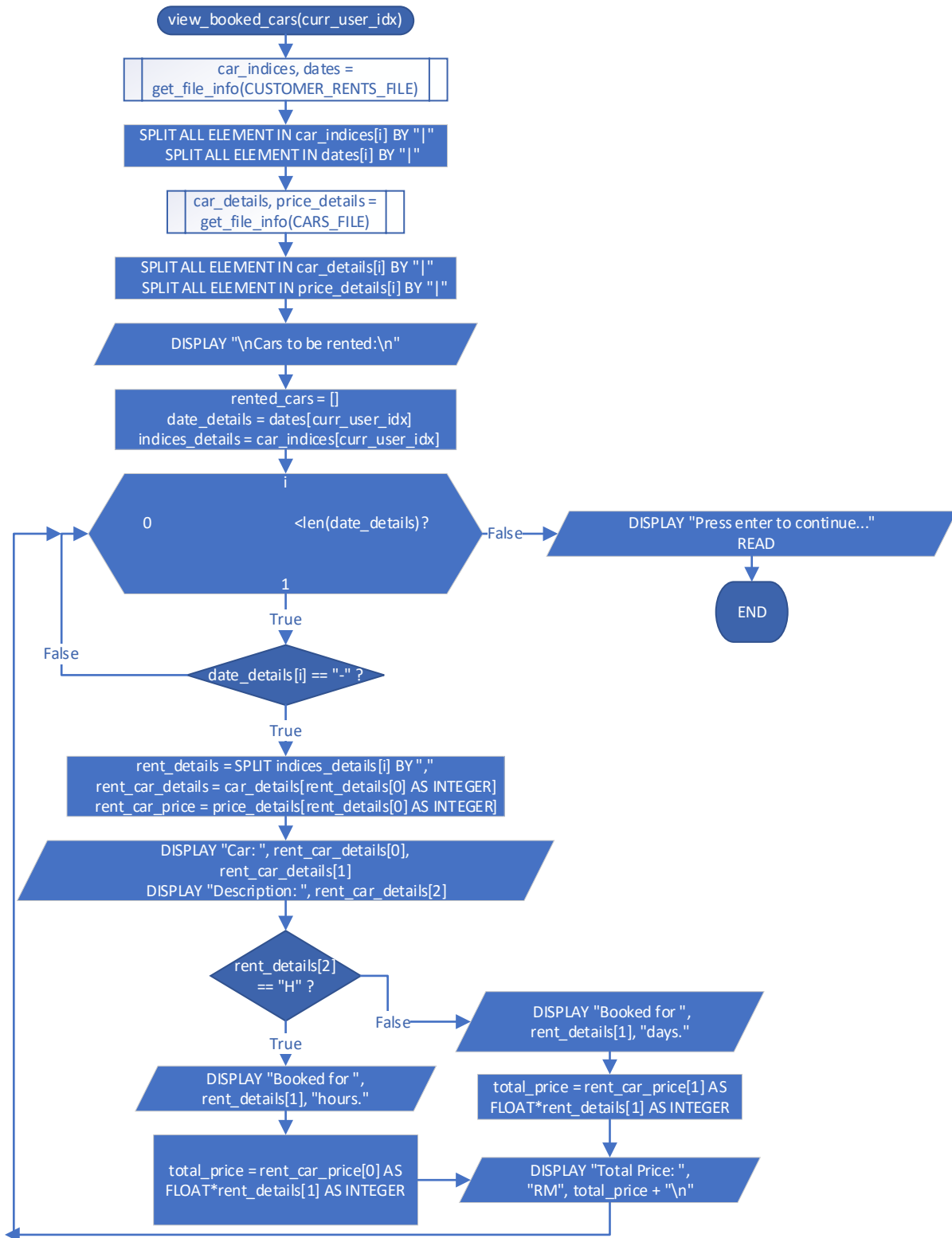
## Registered Customers



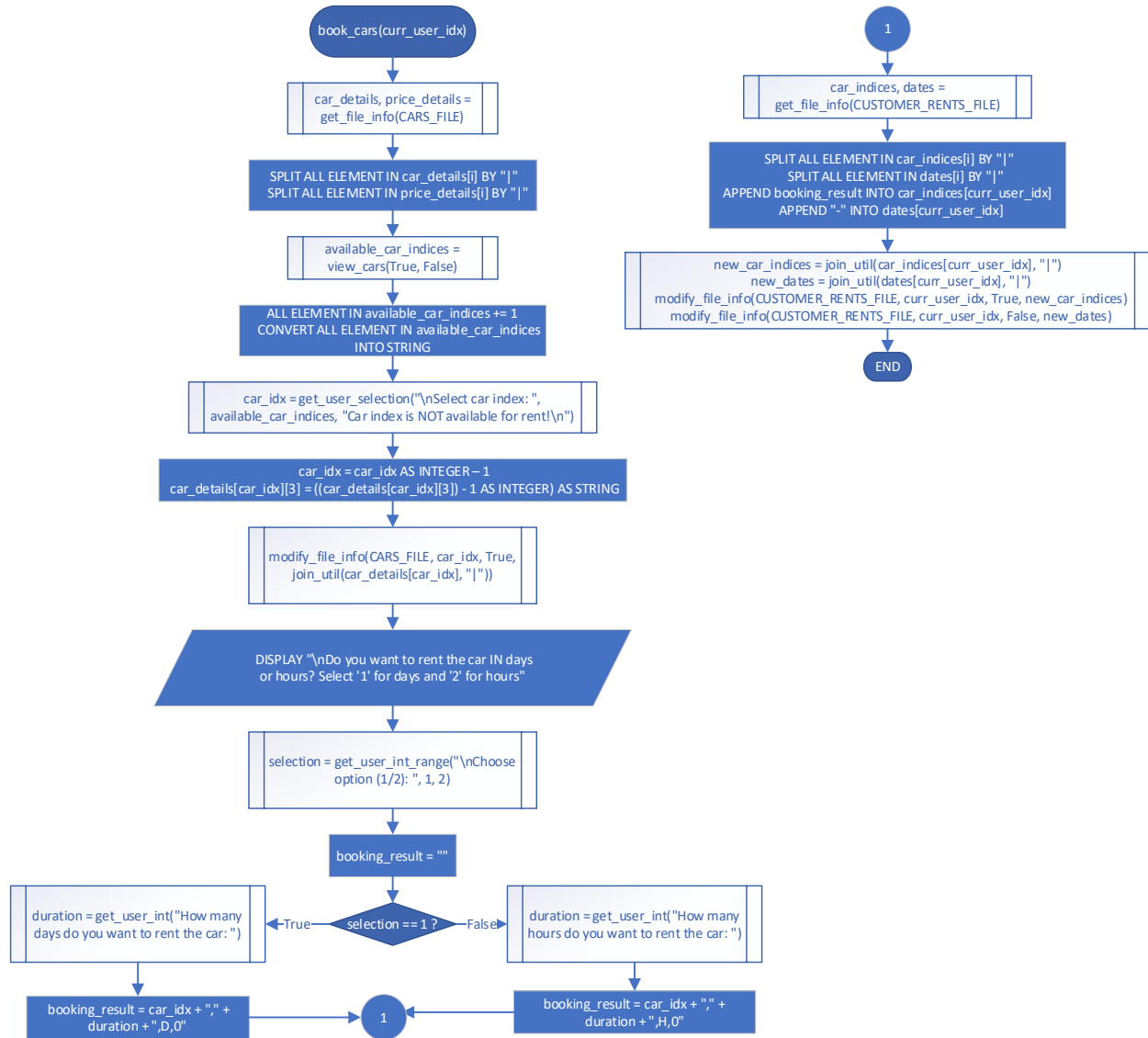
*Flowchart 22: modify\_personal\_details*



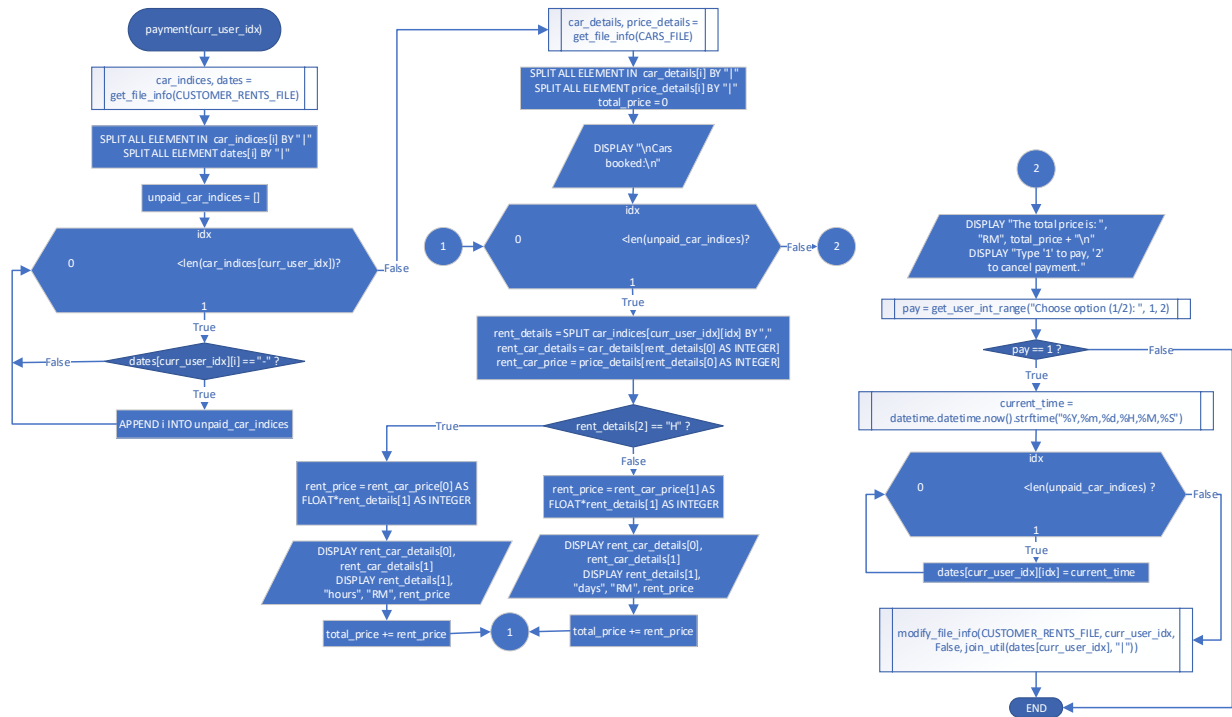
Flowchart 23: view\_history



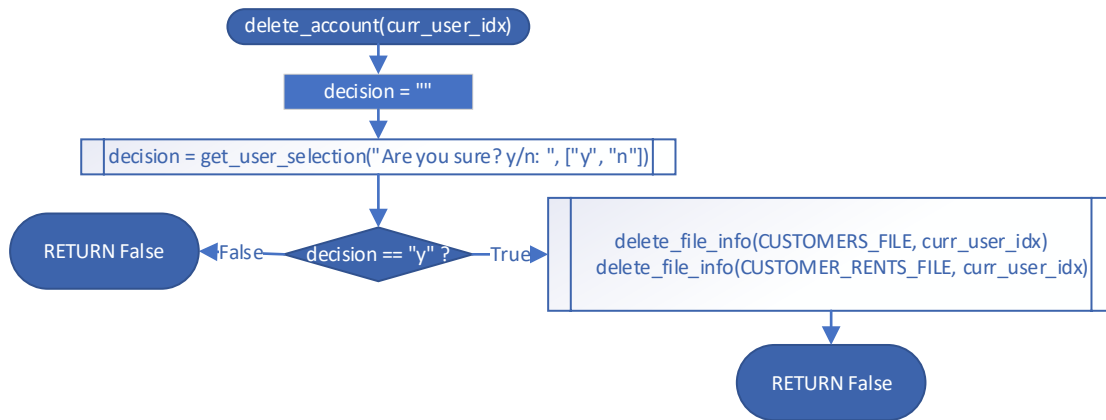
Flowchart 24: view\_booked\_cars



Flowchart 25: book\_cars

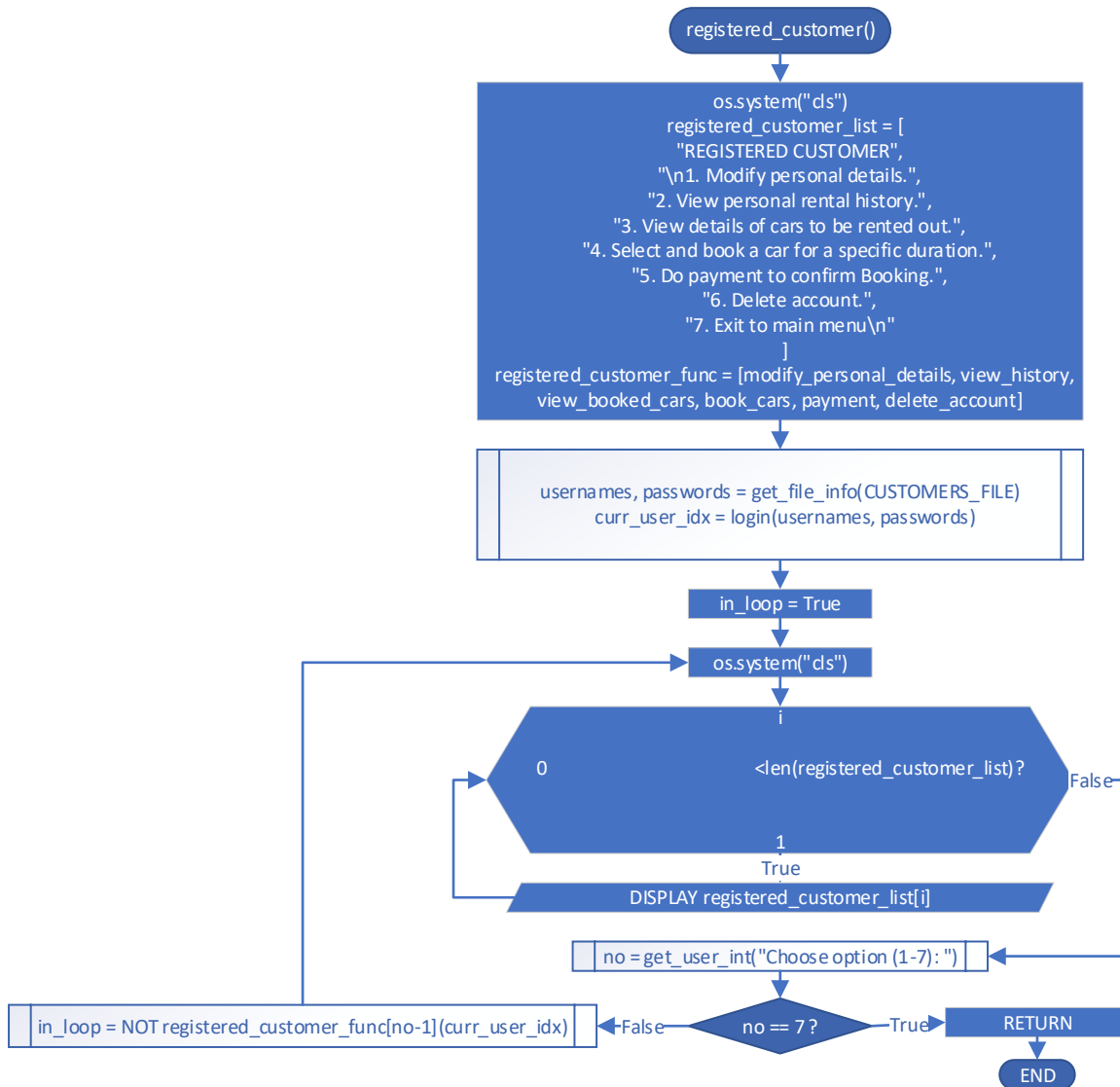


Flowchart 26: payment



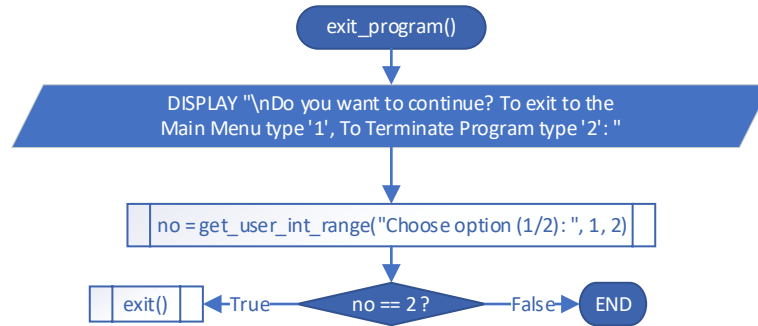
Flowchart 27: delete\_account



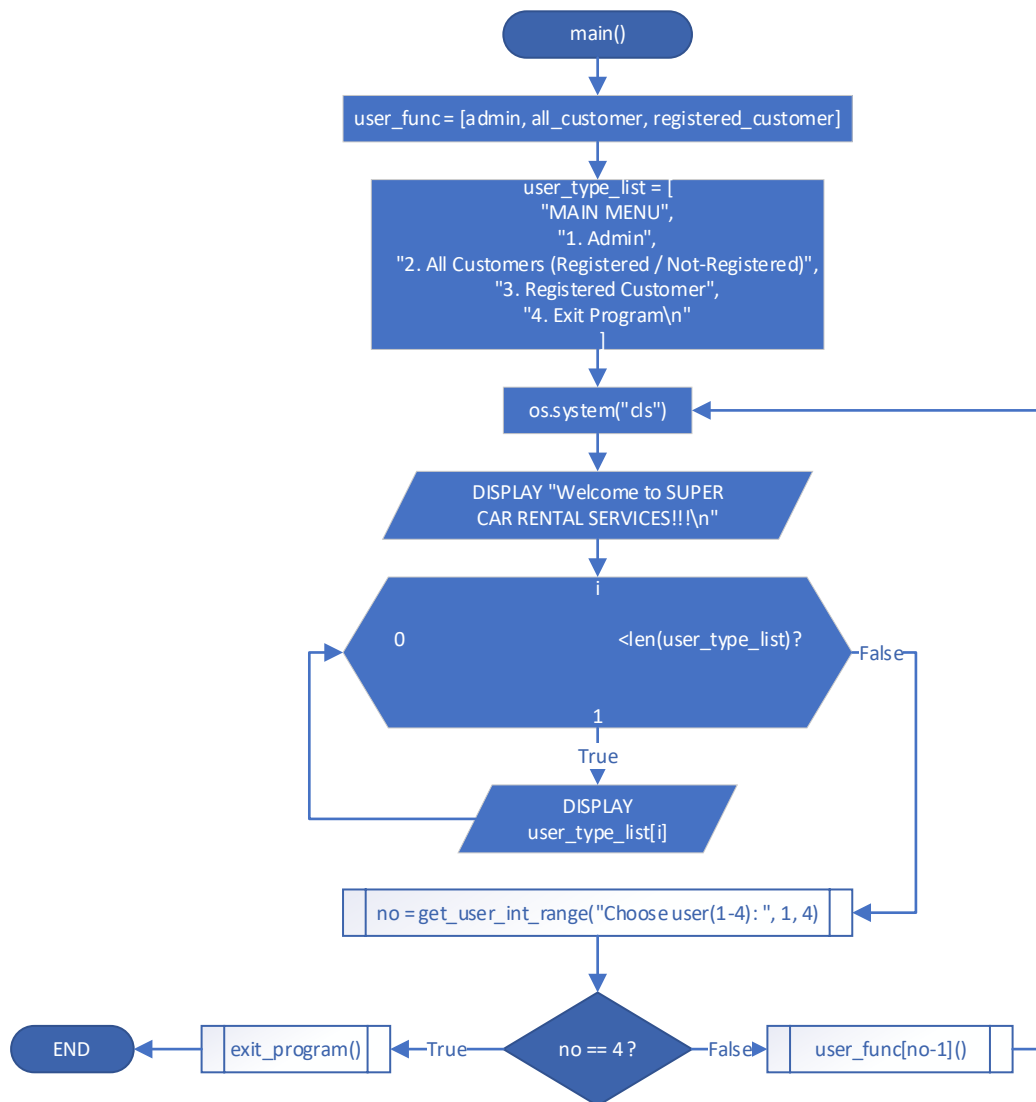


Flowchart 28: *registered\_customer*

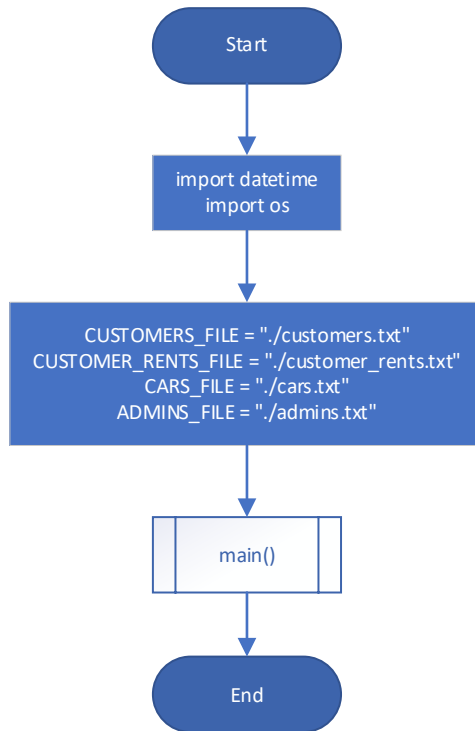
## Main Program



*Flowchart 29: exit\_program*



Flowchart 30: main



*Flowchart 31: running the program*

## Program Source Code

### Annotations

Python annotations are used in the Python code, therefore, you will be able to see snippets like `:str` after a variable to indicate that the variable should be a string or signs like `-> int` which shows that the function returns an integer.

### Naming Conventions

```
def function_name() -> int:
    pass

CONSTANT_VARIABLE = "./customers.txt"
variable_name = 10
```

*Source 1: naming conventions*

These are the naming conventions that we use throughout our python assignment to ensure readability and consistency.

1. All function names uses lowercase. Separate words are being separated by an underscore “\_”.
2. Constant variables uses uppercase on all letters. Separate words are also being separated by an underscore “\_”.
3. Variable names uses the same format as function names.

### File Structure

```
0. 1,2,D,1|3,10,H,1|0,4,D,0
1. 2021,06,02,22,33,47|2021,06,02,22,45,37|-
2. 1,3,H,0|0,1,D,0
3. 2021,06,09,19,52,21|2021,06,09,19,52,21
```

*Source 2: customer\_rents.txt (example of file structure)*

Our file structure is organized so that each data occupies 2 lines. The first line often represents the key and the second line often represents the value. When there is more than one item to store in a line, we use “|” as a separator. When each item has more than one attribute, we use “,” as a separator.

admins.txt, customers.txt

```
0. Nixon
1. 321
2. Andrew
3. 123
```

*Source 3: admins.txt*

In admins.txt and customers.txt, all even number lines are the usernames and odd number lines are the passwords.

[cars.txt](#)

```
0. Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
1. 5.9|59.0
2. Renault|Captur|Interstate business trips, wooing your date, and stylish weekend getaways.|1
3. 11.9|119.0
4. Nissan|Serena S-Hybrid|Long drives out of the city while reducing carbon footprint.|2
5. 26.9|269.0
```

*Source 4: cars.txt*

```
# format
Brand|Model|Description|Car Remaining
Hourly Price|Daily Price
```

*Source 5: cars.txt format*

As shown in the source above, the car details and prices are stored in the cars.txt file where even number lines stores the details and odd number lines store the prices. The details and prices are separated by the "|" separator.

[customer\\_rents.txt](#)

```
0. 1,2,D,1|3,10,H,1|0,4,D,0
1. 2021,06,02,22,33,47|2021,06,02,22,45,37|-
2. 1,3,H,0|0,1,D,0
3. 2021,06,09,19,52,21|2021,06,09,19,52,21
```

*Source 6: customer\_rents.txt*

```
# format
Car Index,Rent Duration,Day/Hour,Return Status|(next record with same format)
Year,Month,Day,Hour,Minute,Second|(next record with same format)
```

*Source 7: customer\_rents.txt format*

In customer\_rents.txt, we store rent details on the even number lines and rent date on the odd number lines. Each record is being separated by " | " and each detail in the record is again separated by " , " to differentiate the details in the record which can be referred from Source 7.

## Utility Functions

```
def join_util(join_list:list, join_str:str) -> str:
    if join_list[0] != "":
        return join_str.join(join_list)
    else: return join_list[1]
```

*Source 8: join\_util function*

This is a simple utility function to join a list of strings together with a unique character to separate each element in the list. This function mainly prevents joining empty elements. We do this by checking if the first element of the list is an empty string or not. If it is, we simply just return the second element. If it is not, we can safely join all strings together.

## Console Input Utility Functions

```
# forcefully get integer input from user
def get_user_int(prompt_msg:str) -> int:
    usr_input = ""
    # check if usr_input is digit or not
    while not usr_input.isdigit():
        # get usr_input from user
        usr_input = input(prompt_msg)

    # check if usr_input is digit or not
    if not usr_input.isdigit():
        # print error message
        print("Input is not an integer\nPlease try again\n")

    return int(usr_input)
```

*Source 9: get\_user\_int function*

This function forcefully gets integer input from the user. We achieve it by using a **while** loop that goes on forever unless the user inputs an integer number. In order to check if the use input is a digit or not, we use the **str** object built in function **.isdigit()**. Whenever the user inputs a non-digit input, we will prompt an error message to indicate the user to try again. When a digit number is obtained, we convert the **usr\_input** into **int** and return it out of this function.



```
# forcefully get float input from user
def get_user_float(prompt_msg:str) -> float:
    usr_input = ""
    # check if usr_input is a float or not
    while not usr_input.replace(".", "", 1).isdigit():
        # get usr_input from user
        usr_input = input(prompt_msg)
        # if usr_input is not a float, print error msg
        if not usr_input.replace(".", "", 1).isdigit():
            print("Input is not a number\nPlease try again\n")
    return float(usr_input)
```

*Source 10: get\_user\_float function*

This function forcefully gets float input from the user. We store it in a function so that we can reuse it. Similar to Source 1, we also achieve this by using a **while** loop.

This time, we cannot check just by using `.isdigit()` because it will return false when there is a character inside the string, which in this case is the ‘.’ decimal point in any float number.

Therefore, in order to check if given input is a *float*, we first remove the decimal point by using the `.replace(".", "", 1)` function to remove the first encountered decimal point. We remove only one decimal point to prevent users from providing a float number with 2 or more decimal point.

When all check is done, we convert the `usr_input` into a *float* and return it out of this function.

```
# forcefully get a non-empty string from user
def get_user_not_empty(prompt_msg:str, error_msg:str) -> str:
    usr_input = ""
    while usr_input == "":
        usr_input = input(prompt_msg)
        if usr_input == "":
            print(error_msg + "\n")
    return usr_input
```

*Source 11: get\_user\_not\_empty function*

Similar to Source 9 and Source 10, we also use a **while** loop to forcefully get a non-empty string input from the user. This function is fairly simple as it only checks if the `usr_input` is empty or

not by doing a simple `usr_input == ""` comparison. When a non-empty input is obtained, we return the raw value out of this function.

```
# forcefully get a string input that is inside the given selection from user
def get_user_selection(prompt_msg:str, selections:list,
    error_msg:str="Input is not within selection given\n") -> str:
    usr_input = ""
    while usr_input not in selections:
        usr_input = input(prompt_msg)
        if usr_input not in selections:
            print(error_msg)

    return usr_input
```

*Source 12: get\_user\_selection function*

This function forcefully gets a string input from user within a selection list. We use the `not in` keyword to check if the input we get from the user is acceptable or not. We will only return the value when the user inputs an acceptable answer.

```
# forcefully get integer input within a range from user
def get_user_int_range(prompt_msg:str, range_min:int, range_max:int,
    exceed_range_error_msg:str="Input has exceed the range given!\n") -> int:
    """
    get user int and only accept a range of int, anything outside the range will be rejected

    ex: range_min: 0, range_max: 5 only accepts (0, 1, 2, 3, 4, 5)
    """
    usr_int = range_min - 1
    while usr_int < range_min or usr_int > range_max:
        usr_int = get_user_int(prompt_msg)
        if usr_int < range_min or usr_int > range_max:
            print(exceed_range_error_msg)

    return usr_int
```

*Source 13: get\_user\_int\_range function*

This function forcefully gets an integer from the user within a given range. The range is define by the `range_min` and `range_max` parameter.

### Console Input Utility Functions Summary

All functions gets a `prompt_msg` as an input argument. The `prompt_msg` is being displayed before the user input. This parameter makes sures that we can alter the prompt message for different use cases.

Some utility functions have a second argument called the `error_msg`. This argument allows us to display different error messages on different scenarios. For example, when we are getting user input for customer name, we could change the error message to "Customer name cannot be empty". This makes the entire program significantly user friendly as the program understands the context of the current situation.

These utility functions are important as they will be used throughout the entire program so that we as the programmers do not need to retype these logic again and again.

## I/O Utility Functions

```
# get usernames and passwords from a text file
def get_file_info(filename:str) -> tuple:
    with open(filename, "r", encoding="utf-8") as file:
        file_content = file.read()
        lines = file_content.split("\n")

    content1 = []
    content2 = []

    # Loop through the entire list lines in the text file and exclude the last line
    for line_idx in range(len(lines) - 1):
        # check if line_idx is an even or odd number
        if line_idx % 2 == 0:
            content1.append(lines[line_idx])
        else:
            content2.append(lines[line_idx])

    return content1, content2
```

*Source 14: get\_file\_info function*

This function reads the content of file based on the given `filename` using the `open` keyword and then splits it into 2 major contents (`content1` or `content2`). The contents in the file should be organized such that all `content1` is in the even number lines and all `content2` is in the odd number lines of the file (see [File Structure](#) section).

Using the `with` keyword, we are able to open the file and read its contents without explicitly closing it with the `file.close()` method. We then split the contents of the file by newline `file_content.split("\n")`. To check if a given `line_idx` is even or odd, we use the modulo `%`

operator to check if the number is divisible by 2 or not. Based on the outcome, we either append it into content1 or content2. These 2 contents will then be returned as a *tuple*.

```
# modify file lines from a text file
def modify_file_info(filename:str, location:int, is_content1:bool, info:str) -> None:
    content1, content2 = get_file_info(filename)
    # edit content1[location]/content2[location] based on is_content1
    if is_content1: content1[location] = info
    else: content2[location] = info

    final_string = ""
    for content_idx in range(len(content1)):
        # content_idx is an integer which will increase until the range ends
        final_string += content1[content_idx] + "\n"
        final_string += content2[content_idx] + "\n"

    with open(filename, "w") as file:
        file.write(final_string)
```

*Source 15: modify\_file\_info function*

This function overwrites a particular line in a file based on a given *filename* and *location*. A *location* consists of 2 lines, therefore, we also need to specify if it is content1 or content2 via the *is\_content1* argument. This function also takes in an *info* argument that specifies what data are we going to replace that line to.

```
# delete file lines from a text file
def delete_file_info(filename:str, location:int) -> None:
    content1, content2 = get_file_info(filename)

    final_string = ""
    for content_idx in range(len(content1)):
        if (content_idx != location):
            final_string += content1[content_idx] + "\n"
            final_string += content2[content_idx] + "\n"

    with open(filename, "w") as file:
        file.write(final_string)
```

*Source 16: delete\_file\_info function*

This function finds the *location* of the data in the file (based on its *filename*) and remove it from the file. As shown above, the method we use to remove the data is by excluding it from content1 and content2.

```
# add file lines to a text file
def add_file_info(filename:str, new_content1:str, new_content2:str) -> None:
    with open(filename, "a") as file:
        file.write(new_content1)
        file.write(new_content2)
```

*Source 17: add\_file\_info function*

As the function name suggests, this function adds a new content into the file (based on *filename*). Contents are based on arguments *new\_content1* and *new\_content2*.

## I/O Utility Functions Summary

With these 4 functions built, we are now capable of manipulating any data in any file. These functions play a major role in storing data, retrieving data, changing data, and deleting data.

## Login System

```
# register new username and password
def register_new_user(filename:str, usernames:list) -> None:
    username = ""
    password = ""

    while username == "" or username in usernames:
        username = get_user_not_empty("Enter username: ")
        if username == "":
            print("Username cannot be empty!\n")
        elif username in usernames:
            print("Username has been taken!\nPlease enter another username.\n")

    while password == "":
        password = get_user_not_empty("Enter password: ")
        if password == "":
            print("Password cannot be empty!\n")

    print(f"Your username and password is: {username}, {password}")

    with open(filename, "a", encoding="utf-8") as customer_file:
        customer_file.write(f"{username}\n{password}\n")
```

*Source 18: register\_new\_user function*

This function lets unregistered customers to register new account. Because our user accounts are stored in a text file, we need to take in the target *filename* as an argument. To prevent users from creating a username that exists in the database itself, we also need to take in all *usernames* that has been created before.

With *usernames* in hand, we can simply check if then given username is taken or not via the username *in* *usernames* evaluation. This evaluation will return *True* as long as the given username is taken. Using a *while* loop, we can keep asking the user to retry as long as the given username is not unique.

```
def login(usernames:list, passwords:list) -> int:
    """
    login a user and return the index of the user
    """
    print("Please log in.\n")
    username = ""
    username_idx = 0
    while username not in usernames:
        # get input from user
        username = get_user_not_empty("Username: ")
        if username not in usernames:
            print("Username not found\nPlease try again\n")
        else:
            username_idx = usernames.index(username)

    password = ""
    # check if password is correct corresponding to the username_idx
    while password != passwords[username_idx]:
        # get input from user
        password = get_user_not_empty("Password: ")
        if password != passwords[username_idx]:
            print("Password incorrect\nPlease try again\n")

    print("Logged in!\n")
    input("Press enter to continue...")
    os.system("cls")
    return username_idx
```

*Source 19: login function*

This function's goal is to let users to login based on a list of available *usernames* and their corresponding *passwords*. This function takes in *usernames* as the list of registered usernames and *passwords* as the list of passwords that corresponds to the *usernames* list.

Firstly, we get the username from the user and check if it exists inside the *usernames* list or not. This is to prevent from accepting username that is not yet registered. Once a valid username is obtained, we get the *username\_idx* of the username from the *usernames* list. This *username\_idx*

is used to identify the actual password for this `username`. That way when we are getting the password from the user, we can easily check if the given password is correct or not by checking if the given password is equal to `passwords[username_idx]` or not. Lastly when a valid password is obtained, a "Logged in!" string is being printed to indicate that the user has successfully logged in and the function ends.

## Admin

```
def add_new_car() -> None:
    car_details, _ = get_file_info(CARS_FILE)
    car_details = [c.split("|") for c in car_details]
    # lower down all cases to perform name checking
    used_car_names = [c[0].lower()+c[1].lower() for c in car_details]

    car_detail = ""
    car_name = ""

    brand = ""
    model = ""
    description = ""

    while car_detail == "" or car_name in used_car_names:
        brand = get_user_not_empty("Enter brand: ", "Brand cannot be empty!\n")
        model = get_user_not_empty("Enter model: ", "Model cannot be empty!\n")
        description = get_user_not_empty("Enter description: ", "Description cannot be empty!\n")
        amount = get_user_int("Enter amount of cars to add: ")

        car_detail = f"{brand}|{model}|{description}|{amount}\n"
        car_name = brand.lower() + model.lower()

        if car_name in used_car_names:
            print("The car name has been taken!\nPlease create a unique one.\n")

    hourly_price = get_user_float("Enter hourly price: ")
    daily_price = get_user_float("Enter daily price: ")
    price_detail = f"{hourly_price}|{daily_price}\n"

    print(f"Your car name is: {brand}, {model}")
    print(f"Your car description is: {description}")
    print(f"The hourly price is: {hourly_price}\nThe daily price is: {daily_price}")

    add_file_info(CARS_FILE, car_detail, price_detail)
```

*Source 20: add\_new\_car function*

This function's goal is to add a new car into the car list. A car contains a few details:

1. Brand
2. Model
3. Description
4. Amount
5. Hourly price
6. Daily price



In this function, we also prevent users from creating a car that has been created before. We achieve this by combining all the `brand` and `model` of existing cars in lower case and store it in a list called `used_car_names`. When user creates a new car, we also combine the `brand` and `model` of the new car in lower case and store it in a variable called `car_name`. Using a simple `car_name in used_car_names` evaluation, we are able to check if the new car has been created before or not.

Once all the details are retrieved, we simply add it into the `CARS_FILE`.

```
def modify_car_details():
    cars, prices = get_file_info(CARS_FILE)
    car_details = []
    price_details = []
    for i in range(len(cars)):
        # split makes strings into list in list
        car_details.append(cars[i].split("|"))
        price_details.append(prices[i].split("|"))

    view_cars(False, False)
    car_idx = get_user_int_range("Choose a car index to edit: ", 1, len(car_details)) - 1

    print("\nWhich car detail you want to modify?")
    print("\n1. Car Detail\n2. Price Detail")
    choose_detail = get_user_int_range("\nEnter (1/2): ", 1, 2)

    if choose_detail == 1:
        print("\nChoose car detail to be modified:")
        print("\n1. Brand\n2. Model\n3. Description\n4. Cars Remaining")
        detail_idx = get_user_int_range("\nChoose car detail (1-4): ", 1, 4) - 1
        if detail_idx != 3:
            new_info = get_user_not_empty("\nEnter new car detail: ", "Nothing is entered. Please try again!\n")
        else:
            new_info = get_user_int("\nEnter new car remaining: ")

        car_details[car_idx][detail_idx] = str(new_info)
        modification = car_details[car_idx]
        modification = join_util(modification, "|")

        modify_file_info(CARS_FILE, car_idx, True, modification)
    else:
        print("\nWhich price detail you want to modify?")
        print("\n1. Hourly Price\n2. Daily Price")
        detail_idx = get_user_int_range("\nChoose price detail(1/2): ", 1, 2) - 1
        new_info = get_user_float("\nEnter new car rent price: ")

        price_details[car_idx][detail_idx] = str(new_info)
        modification = price_details[car_idx]
        modification = join_util(modification, "|")

        modify_file_info(CARS_FILE, car_idx, False, modification)
```

*Source 21: modify\_car\_details function*

This function's goal is to modify the detail of a specific car. We allow admins to choose a car by getting the index of a car. The admins will be able to obtain the index of the car from the `view_cars`

function that we will be defined later in the program below (see [All Customers](#) section). Once we know which car we are going to modify, we ask the admin whether he/she wants to edit the car details or the car price. Once a detail type is selected (car detail / car price), we ask which `detail_idx` the admin wants to change. We then get the user input and modify it based on the selected `detail_idx`.

```
def display_records():
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    car_indices = [idx.split("|") for idx in car_indices]
    dates = [date.split("|") for date in dates]

    car_details, price_details = get_file_info(CARS_FILE)
    car_details = [detail.split("|") for detail in car_details]
    price_details = [detail.split("|") for detail in price_details]

    usernames, _ = get_file_info(CUSTOMERS_FILE)

    # show rented cars START
    print("\nRented cars:")
    for customer_idx in range(len(car_indices)):
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]
        if len(indices_details) <= 0: continue

        print(f"\nCars rented by: {usernames[customer_idx]}\n")
        rent_idx = 0
        for history_idx in range(len(indices_details)):
            if date_details[history_idx] != "-":
                rent_idx += 1
                rent_idx_str = f"{rent_idx}. "
                rent_details = indices_details[history_idx].split(",")
                rent_car_date = date_details[history_idx].split(",")
                # convert all string elements from string to integer
                rent_car_date = [int(d) for d in rent_car_date]
                # create readable datetime format
                rent_car_date = datetime.datetime(
                    rent_car_date[0], rent_car_date[1],
                    rent_car_date[2], rent_car_date[3],
                    rent_car_date[4], rent_car_date[5]
                )
                rent_car_details = car_details[int(rent_details[0])]
                print(f"{rent_idx_str}Car: {rent_car_details[0]}, {rent_car_details[1]}")
                print(" "*len(rent_idx_str) + f"Rented on: {rent_car_date}")
                if rent_details[2] == "D":
                    print(" "*len(rent_idx_str) + f"Rented for: {rent_details[1]} day(s)")
                else:
                    print(" "*len(rent_idx_str) + f"Rented for: {rent_details[1]} hour(s)")
                print(" "*len(rent_idx_str) + f"Status: {'Returned' if int(rent_details[3]) else 'Not returned'}")
    # show rented cars END

    # show cars that are booked START
    print("\n" + "="*50)
    print("Booked cars:")
    for customer_idx in range(len(car_indices)):
        date_details = dates[customer_idx]
```

```

indices_details = car_indices[customer_idx]

print(f"\nCars booked by: {usernames[customer_idx]}\n")
rent_idx = 0
for history_idx in range(len(indices_details)):
    if date_details[history_idx] == "-":
        rent_idx += 1
        rent_idx_str = f"{rent_idx}. "
        rent_details = indices_details[history_idx].split(",")
        rent_car_details = car_details[int(rent_details[0])]
        print(f"{rent_idx_str}Car: {rent_car_details[0]}, {rent_car_details[1]}")
# show cars that are booked END

# show cars available for rent START
print("\n" + "="*50)
print("Cars available for rent:\n")
view_cars(True, True)
# show cars available for rent END

```

*Source 22: display\_records function*

This function's main goal is to just display all the records for the admin. Things that are being displayed include:

1. Cars rented out
2. Customer bookings
3. Customer payment for a specific time duration
4. Cars available for rent

#### [Cars rented out, Customer bookings](#)

This is done by looping through the rent history of each customers. The rent history can be obtained from the CUSTOMER\_RENTS\_FILE which looks like this:

```

0. 1,2,D,1|3,10,H,1|0,4,D,0 # rent detail
1. 2021,06,02,22,33,47|2021,06,02,22,45,37|- # rent date

```

We determine a history as rented when there is a date at that record. If there is only a "-", then we say that it is only booked.

#### [Customer payment for a specific time duration](#)

In the CUSTOMER\_RENTS\_FILE, we are able to obtain the duration of the rent by looking the second and third element of the rent detail record. The second element being the duration and third element being a specification of either it is day or hour. (ex: 1,2,D,1 in this case, it is 2 days)

#### [Cars available for rent](#)

We use the view\_cars function to accomplish this (see [All Customers](#) section).

```

def search_records():
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    customer_names, _ = get_file_info(CUSTOMERS_FILE)
    car_details, _ = get_file_info(CARS_FILE)

    car_indices = [c.split("|") for c in car_indices]
    dates = [d.split("|") for d in dates]
    car_details = [c.split("|") for c in car_details]

    print("")
    for i in range(len(customer_names)):
        print(f"{i+1}. {customer_names[i]}")
        customer_idx = get_user_int_range(f"\nChoose a customer to be searched (1-
{len(customer_names)}): ", 1, len(customer_names)) - 1

    print("\nChoose your option below:")
    print("1. Customer booking\n2. Customer payment")
    option = get_user_int_range("\nEnter option (1/2): ", 1, 2)

    if option == 1:
        # do customer booking
        print(f"\nCars booked by: {customer_names[customer_idx]}\n")
        # all date history in that line
        date_details = dates[customer_idx]
        # all index history in that line
        indices_details = car_indices[customer_idx]

        book_idx = 0
        for history_idx in range(len(indices_details)):
            if date_details[history_idx] == "-":
                book_idx += 1
                rent_idx_str = f"{book_idx}. "
                rent_details = indices_details[history_idx].split(",")
                rent_car_details = car_details[int(rent_details[0])]
                print(f"{rent_idx_str}Car: {rent_car_details[0]}, {rent_car_details[1]}")

    else:
        # do customer payment
        # name in customer_names[i] == car_info[i]
        print(f"\nCars rent by: {customer_names[customer_idx]}\n")
        date_details = dates[customer_idx]
        indices_details = car_indices[customer_idx]

        book_idx = 0
        for history_idx in range(len(indices_details)):
            if date_details[history_idx] != "-":
                book_idx += 1
                rent_idx_str = f"{book_idx}. "
                rent_details = indices_details[history_idx].split(",")
                rent_car_details = car_details[int(rent_details[0])]
                rent_car_date = date_details[history_idx].split(",")
                # convert all string elements from string to integer
                rent_car_date = [int(d) for d in rent_car_date]
                # create readable datetime format
                rent_car_date = datetime.datetime(
                    rent_car_date[0], rent_car_date[1],
                    rent_car_date[2], rent_car_date[3],
                    rent_car_date[4], rent_car_date[5]
                )
                print(f"{rent_idx_str}Car: {rent_car_details[0]}, {rent_car_details[1]}")
                print(f"*{len(rent_idx_str)} + Rented on: {rent_car_date}")
                if rent_details[2] == "D":
                    print(f"*{len(rent_idx_str)} + Rented for: {rent_details[1]} day(s)")
                else:
                    print(f"*{len(rent_idx_str)} + Rented for: {rent_details[1]} hour(s)")

    input("Press enter to continue...")

```

Source 23: search\_records function

This function allows admins to search a specific history on a specific customer. To achieve this, we prompt the admin to enter a customer index according to the customer menu that we printed out. We also ask which detail that the admin wants to search for (bookings or payment/rent). Based on these 2 results, we are able to obtain the related records.

### Obtaining the result

To obtain the result requested by the admin, we first locate the history via the `customer_idx` variable obtained from the admin user. Then, based on the option that the admin choose, we gather different information.

### Customer bookings

Customer bookings is determined by the format of the date details. As mentioned above, we gather all information where the date shows only a dash ("-") and then display it.

### Customer payment

Customer payment is also determined by the format of the details. We gather all information where the date is being updated to an actual date (is not a dash) and then display it.

```
def return_rented_cars():
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    car_indices = [idx.split("|") for idx in car_indices]
    dates = [date.split("|") for date in dates]

    car_details, price_details = get_file_info(CARS_FILE)
    car_details = [detail.split("|") for detail in car_details]
    price_details = [detail.split("|") for detail in price_details]

    usernames, _ = get_file_info(CUSTOMERS_FILE)

    print("")
    for i in range(len(usernames)):
        print(f"{i+1}. {usernames[i]}")

    customer_idx = get_user_int_range(f"Select a customer (1-{len(usernames)}): ", 1, len(usernames)) - 1

    # show unreturned cars START
    date_details = dates[customer_idx]
    indices_details = car_indices[customer_idx]
    unreturned_history_indices = []
    unreturned_car_indices = []
    unreturned_car_dates = []
    for history_idx in range(len(indices_details)):
        if date_details[history_idx] != "-":
            rent_details = indices_details[history_idx].split(",")
            rent_car_date = date_details[history_idx].split(",")
            # convert all string elements from string to integer
            rent_car_date = [int(d) for d in rent_car_date]
            # create readable datetime format
            rent_car_date = datetime.datetime(
                rent_car_date[0], rent_car_date[1],
```

```

        rent_car_date[2], rent_car_date[3],
        rent_car_date[4], rent_car_date[5]
    )
    if rent_details[3] != "1":
        # store history index, car index and rent date
        unreturned_history_indices.append(history_idx)
        unreturned_car_indices.append(int(rent_details[0]))
        unreturned_car_dates.append(str(rent_car_date))

    if len(unreturned_history_indices) > 0:
        print(f"\nCars that are not returned by: {usernames[customer_idx]}")
        for i in range(len(unreturned_car_indices)):
            unreturned_car_detail = car_details[unreturned_car_indices[i]]
            print(f"{i+1}. {unreturned_car_detail[0]}, {unreturned_car_detail[1]}")
            print(" " * len(f"{i+1}. ") + f"Rented on: {unreturned_car_dates[i]}")

        return_car_idx = get_user_int_range(f"\nChoose a car to return (1-
{len(unreturned_history_indices)}): ", 1, len(unreturned_history_indices) - 1)
        # return the car by setting the last value in rent_details to "1"
        rent_details = indices_details[unreturned_history_indices[return_car_idx]].split(",")
        rent_details[3] = "1"
        indices_details[unreturned_history_indices[return_car_idx]] = join_util(rent_details, ",")

        modify_file_info(CUSTOMER_RENTS_FILE, customer_idx, True, join_util(indices_details, "|"))

    else:
        print(f"There are no cars to return from {usernames[customer_idx]}.")

    input("Car returned, press enter to continue...")

```

*Source 24: return\_rented\_cars function*

This function allows admins to return a car that is rented from the customer and restore the car into the available car list. For example, if Vios, Toyota only has 1 remaining car available and customer 1 rented 1 of it, the admin is able to go to customer 1's rent history and return the car to restore the remaining number of Vios, Toyota from 1 to 2.

We first obtain the `customer_idx` from the admin (similar to `search_records` function). Then we display all the rented but not returned cars related to this `customer_idx`. The admin can then choose which car to return.

```

0. 1,2,D,1|3,10,H,1|0,4,D,0 # rent detail
1. 2021,06,02,22,33,47|2021,06,02,22,45,37|- # rent date

```

In this example, 1,2,D,1 means that it has been returned while 0,4,D,0 means that it has not yet been returned. We also avoid displaying cars that has not been rented. In this example, 0,4,D,0 will not be displayed as the rent date is still not being updated to mark it as rented.

```

def admin():
    os.system("cls")
    admin_list = [
        "ADMIN",
        "\n1. Add a new car.",
        "2. Modify car details.",
        "3. Display records",
        "4. Search specific records",
    ]

```

```

    "5. Return a rented car.",
    "6. Return to main menu.\n"
]
admin_func = [add_new_car, modify_car_details, display_records, search_records, return_rented_cars]

usernames, passwords = get_file_info(ADMINS_FILE)
login(usernames, passwords)
while True:
    os.system("cls")
    for i in range(len(admin_list)):
        print(admin_list[i])
    no = get_user_int_range("Choose option (1-6): ", 1, 6)
    if no == 6: return
    admin_func[no-1]()

```

*Source 25: admin function*

This is the main function for our admin menu. It displays all the features that the admins can use. The admin can then select an option to do different task. Once a task is done, it will automatically return to this menu until the admin decides to leave, which can be done by choosing option 6.

There is something unique in this function, which is the `admin_func` list. This is not an ordinary list as it stores functions instead of data. This way, we can prevent ourselves from using multiple if else statement to check the option that the admin chooses. It both enhance the performance of the code and also ensure that the code is always clean.

It is also robust as programmers can easily add new functionalities by extending the list (adding a new function into the list) and adding a new selection to the `admin_list`.

## All Customers

```
def create_acc():
    usernames, _ = get_file_info(CUSTOMERS_FILE)
    register_new_user(CUSTOMERS_FILE, usernames)
    add_file_info(CUSTOMER_RENTS_FILE, "\n", "\n")
```

*Source 26: create\_acc function*

We allow new customers to create a new account so that they can be a registered customer, allowing them to book and rent cars. This function is quite simple as we have already predefined all the abstraction of the needed functionality, which is the `register_new_user` function. After creating a new customer username and password, we also create a new entry at the `CUSTOMER_RENTS_FILE` to reserve a spot for that customer's bookings and rentals.

```
def view_cars(view_available:bool=True, wait=True) -> List:
    cars, price = get_file_info(CARS_FILE)
    available_car_indices = []
    for i in range(len(cars)):
        car_detail = cars[i].split("|")
        price_detail = price[i].split("|")
        if view_available and int(car_detail[3]) <= 0: continue
        available_car_indices.append(i)
        print(f"Car Index: {i+1}")
        print(f"Car: {car_detail[0]}, {car_detail[1]}")
        print(f"Description: {car_detail[2]}")
        print(f"Cars Remaining: {car_detail[3]}")
        print(f"Hourly Price: {price_detail[0]}")
        print(f"Daily Price: {price_detail[1]}\n")

    if wait: input("Press enter to continue...")
    return available_car_indices
```

*Source 27: view\_cars function*

We also allow all customers (registered/not registered) to view all the cars that are available to be rented. If `view_available` is set to `True`, this function simply loops through all the cars inside the `CARS_FILE` and print out all the cars that has a remaining number that is greater than 0. If `view_available` is set to `False`, all cars will be printed out along with its details regardless of how much remaining it has. Once all cars is being printed out, we prompt the user to press enter to continue if `wait` is set to `True` and vice versa. We also return out indices of cars that has a remaining that is greater than 0. Note that this remaining indices will only be correct if `view_available` is



set to `True`. This `view_cars` function as you have noticed has also been used by many other functions above (and below).

```
def all_customer():
    os.system("cls")
    customer_list = [
        "ALL CUSTOMER",
        "\n1. View all cars available for rent.",
        "2. Create new account.",
        "3. Exit to main menu\n"
    ]
    customer_func = [view_cars, create_acc]

    while True:
        os.system("cls")
        for i in range(len(customer_list)):
            print(customer_list[i])
        no = get_user_int_range("Choose option (1-3): ", 1, 3)
        if no == 3: return
        customer_func[no-1]()
```

*Source 28: all\_customer function*

And that's it, we have completed all our features for **All Customers**. This function encapsulates everything just like how the `admin` function does, with option 3 to exit to main menu.

## Registered Customers

```
def modify_personal_details(curr_user_idx:int) -> None:
    print("\n1. Username\n2. Password\n3. Cancel")
    selection = get_user_int_range("Choose option (1-3): ", 1, 3, "Select 1 or 2 only!\n")
    if selection == 3: return

    if selection == 1:
        new_username = get_user_not_empty("New username: ")
        modify_file_info(CUSTOMERS_FILE, curr_user_idx, True, new_username)
    else:
        new_password = get_user_not_empty("New password: ")
        modify_file_info(CUSTOMERS_FILE, curr_user_idx, False, new_password)
```

*Source 29: modify\_personal\_details function*

This function allows registered customers to modify their username and password. We first ask them which detail the customer wants to change, based on that, we prompt them to enter the new detail and then store it back into the CUSTOMERS\_FILE. It is also important to note that a `curr_user_idx` is taken in as an argument so that we know which username and password to overwrite.

```
def view_history(curr_user_idx:int) -> None:
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    car_indices = [idx.split("|") for idx in car_indices]
    dates = [date.split("|") for date in dates]

    car_details, price_details = get_file_info(CARS_FILE)
    car_details = [detail.split("|") for detail in car_details]
    price_details = [detail.split("|") for detail in price_details]

    print("\nRented cars:\n")
    rented_cars = []

    date_details = dates[curr_user_idx]
    indices_details = car_indices[curr_user_idx]

    rent_idx = 0
    for i in range(len(date_details)):
        if date_details[i] != "-":
            rent_idx += 1
            rent_idx_str = f"{rent_idx}. "
            rent_details = indices_details[i].split(",")
            rent_car_date = date_details[i].split(",")
            # convert all string elements from string to integer
            rent_car_date = [int(d) for d in rent_car_date]
            # create readable datetime format
            rent_car_date = datetime.datetime(
                rent_car_date[0], rent_car_date[1],
                rent_car_date[2], rent_car_date[3],
                rent_car_date[4], rent_car_date[5]
            )
            rent_car_details = car_details[int(rent_details[0])]
            print(f"{rent_idx_str}Car: {rent_car_details[0]}, {rent_car_details[1]}")
            print("  *len(rent_idx_str) + f"Rented on: {rent_car_date}")
            print("  *len(rent_idx_str) + f>Status: {'Returned' if int(rent_details[3]) else 'Not returned'}")

    input("Press enter to continue...")
```

*Source 30: view\_history function*

This function allows a registered customers to view their rental history only. We do this by looping over all the details in the `CUSTOMER_RENTS_FILE` using a `for` loop. We obtain the detail of the car that the customer rent by referencing it from the `CARS_FILE`. Similar to `modify_personal_details`, we also takes in the `curr_user_idx` as an argument so that we know which customer's rent history to look at. Again, as mentioned above, we determine a car is rented when the date is updated to actual date instead of a "-".

```
def view_booked_cars(curr_user_idx:int) -> None:
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    car_indices = [idx.split("|") for idx in car_indices]
    dates = [date.split("|") for date in dates]

    car_details, price_details = get_file_info(CARS_FILE)
    car_details = [detail.split("|") for detail in car_details]
    price_details = [detail.split("|") for detail in price_details]

    print("\nCars to be rented:\n")
    rented_cars = []

    date_details = dates[curr_user_idx]
    indices_details = car_indices[curr_user_idx]

    for i in range(len(date_details)):
        if date_details[i] == "-":
            rent_details = indices_details[i].split(",")
            rent_car_details = car_details[int(rent_details[0])]
            rent_car_price = price_details[int(rent_details[0])]
            print(f"Car: {rent_car_details[0]}, {rent_car_details[1]}")
            print(f"Description: {rent_car_details[2]}")
            if rent_details[2] == "H":
                print(f"Booked for {rent_details[1]} hours.")
                total_price = float(rent_car_price[0])*int(rent_details[1])
            else:
                print(f"Booked for {rent_details[1]} days.")
                total_price = float(rent_car_price[1])*int(rent_details[1])
            print(f"Total Price: RM {total_price}\n")

    input("Press enter to continue...")
```

*Source 31: view\_booked\_cars function*

This function allows user to view their booking history. It works just like the `view_history` function but just in reverse. Instead of avoiding "-", we print out all cars with dates are not updated yet.

```
def book_cars(curr_user_idx:int) -> None:
    # decrease cars remaining by 1 START
    car_details, price_details = get_file_info(CARS_FILE)
    car_details = [detail.split("|") for detail in car_details]
```

```

price_details = [detail.split("|") for detail in price_details]

available_car_indices = view_cars(True, False)
available_car_indices = [str(idx+1) for idx in available_car_indices]
car_idx = get_user_selection("\nSelect car index: ", available_car_indices, "Car index is not available for rent!\n")
car_idx = int(car_idx) - 1

car_details[car_idx][3] = str(int(car_details[car_idx][3]) - 1)
modify_file_info(CARS_FILE, car_idx, True, join_util(car_details[car_idx], "|"))
# decrease car remaining by 1 END

# car_indices: car_idx,duration,D/H|car_idx,duration,D/H|car_idx,duration,D/H
# dates      : date|date|date

# ask how long does the customer wants to rent the car START
print("\nDo you want to rent the car in days or hours? Select '1' for days and '2' for hours")
selection = get_user_int_range("\nChoose option (1/2): ", 1, 2)
booking_result = ""
if selection == 1:
    duration = get_user_int("How many days do you want to rent the car: ")
    booking_result = f"{car_idx},{duration},D,0"
else:
    duration = get_user_int("How many hours do you want to rent the car: ")
    booking_result = f"{car_idx},{duration},H,0"
# ask how long does the customer wants to rent the car END

# append new car index and allocate a new date in customer_rents.txt START
car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
car_indices = [idx.split("|") for idx in car_indices]
dates = [date.split("|") for date in dates]

car_indices[curr_user_idx].append(booking_result)
dates[curr_user_idx].append("-")

new_car_indices = join_util(car_indices[curr_user_idx], "|")
new_dates = join_util(dates[curr_user_idx], "|")

modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, True, new_car_indices)
modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, False, new_dates)
# append new car index and allocate a new date in customer_rents.txt END

```

*Source 32: book\_cars function*

This function allows customers to book a specific cars that has a remaining that is greater than 0. This prevents the remaining number to go negative. We use the `view_cars` function that we defined above (see [All Customers](#) section) to show all cars that are available to be booked. Noticed that we pass in `True` for the `view_available` argument to prevent showing cars that has no remaining and `False` for the wait argument to prevent the menu to pause. From the `view_cars` function, we obtain all available indices and then ask for the `car_idx` that the customer wants to book. Once a car is selected, we ask for the duration (Day/Hour). We then save it back into the `CUSTOMER_RENTS_FILE` based on the customer's answer. We take `curr_user_idx` as an argument so that we know which part of the file to modify.

```

def payment(curr_user_idx:int) -> None:
    # check which booked cars are not payed yet START
    car_indices, dates = get_file_info(CUSTOMER_RENTS_FILE)
    car_indices = [idx.split("|") for idx in car_indices]

```

```

dates = [date.split("|") for date in dates]

unpaid_car_indices = []
for i in range(len(car_indices[curr_user_idx])):
    if dates[curr_user_idx][i] == "-":
        unpaid_car_indices.append(i)
# check which booked cars are not payed yet END

car_details, price_details = get_file_info(CARS_FILE)
car_details = [detail.split("|") for detail in car_details]
price_details = [detail.split("|") for detail in price_details]

# print out details and show cars that the customer booked START
total_price = 0
print("\nCars booked:\n")
for idx in unpaid_car_indices:
    rent_details = car_indices[curr_user_idx][idx].split(",")
    rent_car_details = car_details[int(rent_details[0])]
    rent_car_price = price_details[int(rent_details[0])]
    if rent_details[2] == "H":
        rent_price = float(rent_car_price[0])*int(rent_details[1])
        print(rent_car_details[0], rent_car_details[1], end=" ")
        print(f"* {rent_details[1]} hours (RM {rent_price})")
        total_price += rent_price
    else:
        rent_price = float(rent_car_price[1])*int(rent_details[1])
        print(rent_car_details[0], rent_car_details[1], end=" ")
        print(f"* {rent_details[1]} days (RM {rent_price})")
        total_price += rent_price

print(f"The total price is: RM {total_price}\n")
# print out details and show cars that the customer booked END

print("Type '1' to pay, '2' to cancel payment.")
pay = get_user_int_range("Choose option (1/2): ", 1, 2)
if pay == 1:
    # replace "-" in dates with actual dates to mark it as paid START
    # get current time
    current_time = datetime.datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
    for idx in unpaid_car_indices:
        dates[curr_user_idx][idx] = current_time
    # replace "-" in dates with actual dates to mark it as paid END

# write car_indices and dates back into the file START
modify_file_info(CUSTOMER_RENTS_FILE, curr_user_idx, False, join_util(dates[curr_user_idx], "|"))
# write car_indices and dates back into the file END

```

Source 33: payment function

This function allows customer to pay for their bookings and then mark them as rent. As explained in the above sections, we mark the car as rented by updating the date to the date of payment. Therefore, in this function, we will ask the customer to confirm if he/she wants to pay for the booked cars. If the customer agrees, we replace all the "-" with the exact date that the car is being paid. Before asking for payment, we also calculate the price by multiplying the duration with the

hourly/monthly cost (depending on what the user choose when booking). We take `curr_user_idx` as an argument so that we know which part of the `CUSTOMER_RENTS_FILE` to modify.

```
def delete_account(curr_user_idx:int) -> bool:
    decision = ""
    decision = get_user_selection("Are you sure? y/n: ", ["y", "n"])

    if decision == "y":
        delete_file_info(CUSTOMERS_FILE, curr_user_idx)
        delete_file_info(CUSTOMER_RENTS_FILE, curr_user_idx)
        return True

    return False
```

*Source 34: delete\_account function*

Last but not least, we have our newly added feature. This function allows a registered customer to delete their account and therefore become an unregistered customer. It is a simple function that asks a yes or no question before deleting the account. If the customer confirms to delete their account, we return a `True` and vice versa. This is to signal our main menu system to exit out of registered customer menu when the customer's account is being deleted.

```
def registered_customer() -> None:
    os.system("cls")
    registered_customer_list = [
        "REGISTERED CUSTOMER",
        "\n1. Modify personal details.",
        "2. View personal rental history.",
        "3. View details of cars to be rented out.",
        "4. Select and book a car for a specific duration.",
        "5. Do payment to confirm Booking.",
        "6. Delete account.",
        "7. Exit to main menu\n"
    ]
    registered_customer_func = [modify_personal_details, view_history, view_booked_cars, book_cars, payment, delete_account]

    usernames, passwords = get_file_info(CUSTOMERS_FILE)
    curr_user_idx = login(usernames, passwords)
    in_loop = True
    while in_loop:
        os.system("cls")
        for i in range(len(registered_customer_list)):
            print(registered_customer_list[i])
        no = get_user_int("Choose option (1-7): ")
        if no == 7: return
        in_loop = not registered_customer_func[no-1](curr_user_idx)
```

*Source 35: registered\_customer function*

This function is similar to the `admin` and `all_customer` function, however, this time we also expect an output from our sub functions. This is because we have a `delete_account` function that returns a `True` when an account is delete. When we get this signal, we immediately terminates this function and return to the main menu.

## Main Program

```
def exit_program() -> None:
    print("\nDo you want to continue? To exit to the Main Menu type '1', To Terminate Program type '2': ")
    no = get_user_int_range("Choose option (1/2): ", 1, 2)
    if no == 2: exit()
```

*Source 36: exit\_program function*

This function simply asks the user to confirm if he/she wants to exit the program. If so, we exit the entire python program via the `exit` keyword.

```
def main() -> None:
    user_func = [admin, all_customer, registered_customer]

    user_type_list = [
        "MAIN MENU",
        "1. Admin",
        "2. All Customers (Registered / Not-Registered)",
        "3. Registered Customer",
        "4. Exit Program\n"
    ]

    while True:
        os.system("cls")
        print("Welcome to SUPER CAR RENTAL SERVICES!!!\n")
        for i in range(len(user_type_list)):
            print(user_type_list[i])

        no = get_user_int_range("Choose user(1-4): ", 1, 4)
        if no == 4: exit_program()
        else: user_func[no-1]()
```

*Source 37: main function*

And there we have it! Our main function that encapsulates every features in the program. This is essentially the main menu of the entire program. We let users select the type of users and bring them to their respective menu.

```
import datetime
import os

CUSTOMERS_FILE = "./customers.txt"
CUSTOMER_RENTS_FILE = "./customer_rents.txt"
CARS_FILE = "./cars.txt"
ADMINS_FILE = "./admins.txt"

main()
```

*Source 38: running the program*



This is how our actual program outside those functions looks like (excluding all those functions of course). We first import the only 2 libraries that we can use which is the `datetime` and `os` library. We then define 4 constants, assigning them as the name of the files that we store our data at. Last but not least, we call our `main` function!

## Additional Features

These are the additional features that we add because we think they are necessary to the car rental system:

1. Return to main menu and stay at a menu as long as user decides not to return to main menu.
2. Delete customer account.
3. Available car remainder counter.
4. Splitting renting price into hourly and daily.

### Return to main menu

```
def all_customer():
    os.system("cls")
    customer_list = [
        "ALL CUSTOMER",
        "\n1. View all cars available for rent.",
        "2. Create new account.",
        "3. Exit to main menu\n"
    ]
    customer_func = [view_cars, create_acc]

    while True:
        os.system("cls")
        for i in range(len(customer_list)):
            print(customer_list[i])
        no = get_user_int_range("Choose option (1-3): ", 1, 3)
        if no == 3: return
        customer_func[no-1]()
```

*Source 39: return to main menu demonstration*

This feature can be found in all of the main function of each user. This prevents customers and admins to re-login every time they complete an action.

### Delete Customer Account

```
def delete_account(curr_user_idx:int) -> bool:
    decision = ""
    decision = get_user_selection("Are you sure? y/n: ", ["y", "n"])

    if decision == "y":
        delete_file_info(CUSTOMERS_FILE, curr_user_idx)
        delete_file_info(CUSTOMER_RENTS_FILE, curr_user_idx)
        return True

    return False
```

*Source 40: delete customer account demonstration*

This function allows customers to be unregistered. All renting and booking history will be removed alongside with it when the account is being removed.

### Available Car Remaining Counter

```
0. Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
1. 5.9|59.0
```

This is a piece of information from the [cars.txt](#). Notice that there is a number at the end of line 0. This is the counter for how many remaining cars there are. With this, we can prevent duplicate data as we don't need to store a new entry when we add a new similar car. In Source 21, ([modify\\_car\\_details](#) function), we also allow admins to edit the remaining counter of a car.

### Hourly and Daily Renting Price

```
0. Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
1. 5.9|59.0
```

In line 1, there are 2 items, the first one being the hourly price and the second being the daily price. This encourages customers to rent daily as it is far cheaper compared to renting hourly. Splitting it to daily and hourly also prevents customers from calculating so much. This is because, normally a customer would need to give a number that is a multiple of 24 when he/she decides to rent in a daily basis. However, due to our newly added daily price, customers do not need to do any additional calculations.

## Sample Input and Output

### Main Menu

Welcome Page

```
Welcome to SUPER CAR RENTAL SERVICES!!!

MAIN MENU
1. Admin
2. All Customers (Registered / Not-Registered)
3. Registered Customer
4. Exit Program

Choose user(1-4): █
```

*Figure 1: main menu*

This is the main menu for our program. We display a welcome message and options available.

## Admin

### Admin Main Menu

```
Please log in.  
  
Username: Nix  
Username not found  
Please try again  
  
Username: Nixon  
Password: 123  
Password incorrect  
Please try again  
  
Password: 321  
Logged in!  
  
Press enter to continue...█
```

*Figure 2: admin login*

When entering the admin menu, we need to login before accessing the admin features. During login, we also check if the entered username and password is correct or not. As you can see we prevent user from logging in if they type the wrong username or password.

```
ADMIN  
  
1. Add a new car.  
2. Modify car details.  
3. Display records  
4. Search specific records  
5. Return a rented car.  
6. Return to main menu.  
  
Choose option (1-6): █
```

*Figure 3: admin menu*

We display all admin features in the admin menu.

## Add Car

```
Enter brand: Toyota
Enter model: Camry
Enter description: Fast and reliable.
Enter amount of cars to add: 4
The car name has been taken!
Please create a unique one.

Enter brand: Proton
Enter model: Saga
Enter description: Fast and reliable.
Enter amount of cars to add: 4
Enter hourly price: 10
Enter daily price: 100
```

*Figure 3: add car*

When adding car we also checks if the car has been added or not. We only add it in if it is a unique new car.

```
1 Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
2 5.9|59.0
3 Renault|Captur|Interstate business trips, wooing your date, and stylish weekend getaways.|1
4 11.9|119.0
5 Nissan|Teana|Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.|1
6 24.9|249.0
7 Nissan|Serena S-Hybrid|Long drives out of the city while reducing carbon footprint.|0
8 26.9|269.0
9 Toyota|Camry|Good for traveling and business trip.|0
10 30.0|315.0
11 |
```

*Figure 4: database before adding car*

```
1 Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
2 5.9|59.0
3 Renault|Captur|Interstate business trips, wooing your date, and stylish weekend getaways.|1
4 11.9|119.0
5 Nissan|Teana|Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.|1
6 24.9|249.0
7 Nissan|Serena S-Hybrid|Long drives out of the city while reducing carbon footprint.|0
8 26.9|269.0
9 Toyota|Camry|Good for traveling and business trip.|0
10 30.0|315.0
11 Proton|Saga|Fast and reliable.|4
12 10.0|100.0
13
```

*Figure 5: database after adding car*

## Modify Car Details

```
Car Index: 1
Car: Nissan, Almera
Description: City drives, running errands, road trips on a budget, and going meetings in town.
Cars Remaining: 1
Hourly Price: 5.9
Daily Price: 59.0

Car Index: 2
Car: Renault, Captur
Description: Interstate business trips, wooing your date, and stylish weekend getaways.
Cars Remaining: 1
Hourly Price: 11.9
Daily Price: 119.0

Car Index: 3
Car: Nissan, Teana
Description: Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.
Cars Remaining: 1
Hourly Price: 24.9
Daily Price: 249.0

Car Index: 4
Car: Nissan, Serena S-Hybrid
Description: Long drives out of the city while reducing carbon footprint.
Cars Remaining: 0
Hourly Price: 26.9
Daily Price: 269.0

Car Index: 5
Car: Toyota, Camry
Description: Good for traveling and business trip.
Cars Remaining: 0
Hourly Price: 30.0
Daily Price: 315.0

Car Index: 6
Car: Proton, Saga
Description: Fast and reliable.
Cars Remaining: 4
Hourly Price: 10.0
Daily Price: 100.0

Choose a car index to edit: 
```

*Figure 6: modify car details (display)*

Display all car index and details to let admin choose which car to modify.

```
Choose a car index to edit: 7
Input has exceed the range given!

Choose a car index to edit: 6

Which car detail you want to modify?

1. Car Detail
2. Price Detail

Enter (1/2): 
```

*Figure 7: modify car details (choosing a car)*

When choosing a car to edit we prevent admins from choosing a car index that is not available. Once a valid index is chosen, we let them choose the details that they want to edit.

```
Which car detail you want to modify?

1. Car Detail
2. Price Detail

Enter (1/2): 1

Choose car detail to be modified:

1. Brand
2. Model
3. Description
4. Cars Remaining

Choose car detail (1-4): 3

Enter new car detail: Slow and steady.
```

*Figure 8: modify car description*

```
1 Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
2 5.9|59.0
3 Renault|Captur|Interstate business trips, wooing your date, and stylish weekend getaways.|1
4 11.9|119.0
5 Nissan|Teana|Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.|1
6 24.9|249.0
7 Nissan|Serena S-Hybrid|Long drives out of the city while reducing carbon footprint.|0
8 26.9|269.0
9 Toyota|Camry|Good for traveling and business trip.|0
10 30.0|315.0
11 Proton|Saga|Slow and steady.|4
12 10.0|100.0
13
```

*Figure 9: database after modifying car*



## Display Records

```
Rented cars:

Cars rented by: Andrew

1. Car: Renault, Captur
   Rented on: 2021-06-02 22:33:47
   Rented for: 2 day(s)
   Status: Returned
2. Car: Nissan, Serena S-Hybrid
   Rented on: 2021-06-02 22:45:37
   Rented for: 100 hour(s)
   Status: Returned

Cars rented by: Nixon

1. Car: Renault, Captur
   Rented on: 2021-06-09 19:52:21
   Rented for: 3 hour(s)
   Status: Not returned
2. Car: Nissan, Almera
   Rented on: 2021-06-09 19:52:21
   Rented for: 1 day(s)
   Status: Not returned

=====
```

*Figure 10: display records (rented cars)*

We display all the rented cars.

```
Booked cars:

Cars booked by: Andrew

1. Car: Nissan, Almera

Cars booked by: Nixon

1. Car: Nissan, Serena S-Hybrid
2. Car: Nissan, Serena S-Hybrid

=====
```

*Figure 11: display records (booked cars)*

We display all the booked cars and who booked it.

```
Cars available for rent:

Car Index: 1
Car: Nissan, Almera
Description: City drives, running errands, road trips on a budget, and going meetings in town.
Cars Remaining: 1
Hourly Price: 5.9
Daily Price: 59.0

Car Index: 2
Car: Renault, Captur
Description: Interstate business trips, wooing your date, and stylish weekend getaways.
Cars Remaining: 1
Hourly Price: 11.9
Daily Price: 119.0

Car Index: 3
Car: Nissan, Teana
Description: Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.
Cars Remaining: 1
Hourly Price: 24.9
Daily Price: 249.0

Car Index: 6
Car: Proton, Saga
Description: Slow and steady.
Cars Remaining: 4
Hourly Price: 10.0
Daily Price: 100.0

Press enter to continue...█
```

*Figure 12: display records (cars available for rent)*

We display all the cars that are available for rental and booking.

#### Search Records

```
1. Andrew
2. Nixon

Choose a customer to be searched (1-2): 2

Choose your option below:
1. Customer booking
2. Customer payment

Enter option (1/2): 1

Cars booked by: Nixon

1. Car: Nissan, Serena S-Hybrid
2. Car: Nissan, Serena S-Hybrid
Press enter to continue...█
```

*Figure 13: search records (customer bookings)*

We allow admins to search a booking record for a specific customer.

```

1. Andrew
2. Nixon

Choose a customer to be searched (1-2): 1

Choose your option below:
1. Customer booking
2. Customer payment

Enter option (1/2): 2

Cars rent by: Andrew

1. Car: Renault, Captur
   Rented on: 2021-06-02 22:33:47
   Rented for: 2 day(s)
2. Car: Nissan, Serena S-Hybrid
   Rented on: 2021-06-02 22:45:37
   Rented for: 100 hour(s)
Press enter to continue...

```

*Figure 14: search records (customer payments)*

We allow admins to search a payment record for a specific customer.

```

1. Andrew
2. Nixon
Select a customer (1-2): 1
There are no cars to return from Andrew.
Car returned, press enter to continue...

```

*Figure 15: returning cars (no cars to return)*

When there are no cars to be returned, we tell the admin.

Return Rented Car

```

1. Andrew
2. Nixon
Select a customer (1-2): 2

Cars that are not returned by: Nixon
1. Renault, Captur
   Rented on: 2021-06-09 19:52:21
2. Nissan, Almera
   Rented on: 2021-06-09 19:52:21

Choose a car to return (1-2): 2
Car returned, press enter to continue...

```

*Figure 16: returning cars (existing cars to return)*

When there is a car to be returned, we display them and let admin to choose a car to return.

```

1  1,2,D,1|3,100,H,1|0,4,D,0
2  2021,06,02,22,33,47|2021,06,02,22,45,37|-
3  1,3,H,0|0,1,D,0|3,5,H,0|3,1,D,0
4  2021,06,09,19,52,21|2021,06,09,19,52,21|-|-
5  |

```

*Figure 17: database before return*

```

1  1,2,D,1|3,100,H,1|0,4,D,0
2  2021,06,02,22,33,47|2021,06,02,22,45,37|-
3  | 1,3,H,0|0,1,D,1|3,5,H,0|3,1,D,0
4  2021,06,09,19,52,21|2021,06,09,19,52,21|-|-
5

```

*Figure 18: database after returning*

## All Customers

### All Customers Main Menu

```
ALL CUSTOMER

1. View all cars available for rent.
2. Create new account.
3. Exit to main menu

Choose option (1-3): █
```

*Figure 19: all customer menu*

We display all features available for all customers.

### View All Cars Available For Rent

```
Car Index: 1
Car: Nissan, Almera
Description: City drives, running errands, road trips on a budget, and going meetings in town.
Cars Remaining: 1
Hourly Price: 5.9
Daily Price: 59.0

Car Index: 2
Car: Renault, Captur
Description: Interstate business trips, wooing your date, and stylish weekend getaways.
Cars Remaining: 1
Hourly Price: 11.9
Daily Price: 119.0

Car Index: 3
Car: Nissan, Teana
Description: Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.
Cars Remaining: 1
Hourly Price: 24.9
Daily Price: 249.0

Car Index: 6
Car: Proton, Saga
Description: Slow and steady.
Cars Remaining: 4
Hourly Price: 10.0
Daily Price: 100.0

Press enter to continue...█
```

*Figure 20: view all cars available for rent*

We display all cars that are available for rent.

Create new account

```
Enter username: Nixon
Username has been taken!
Please enter another username.

Enter username: Bob
Enter password: bobby123
```

*Figure 21: create new account*

We let users create a new account by taking in username and password. We also make sure no duplicated account is created by checking the username.

```
1  Andrew
2  456
3  Nixon
4  123
5  |
```

*Figure 22: database before creating new account*

```
1  Andrew
2  456
3  Nixon
4  123
5  Bob
6  bobby123
7  |
```

*Figure 23: database after creating new account*

## Registered Customers

### Registered Customer Main Menu

```
Please log in.  
  
Username: Nixon  
Password: 321  
Password incorrect  
Please try again  
  
Password: 123  
Logged in!  
  
Press enter to continue...█
```

*Figure 24: logging in as registered customer*

Similar to admin menu, we need to log in before entering the menu. We also make sure that the password and username is correct.

```
REGISTERED CUSTOMER  
  
1. Modify personal details.  
2. View personal rental history.  
3. View details of cars to be rented out.  
4. Select and book a car for a specific duration.  
5. Do payment to confirm Booking.  
6. Delete account.  
7. Exit to main menu  
  
Choose option (1-7): █
```

*Figure 25: registered customer menu*

We display all the features included for the registered customer.

## Modify Personal Details

```
1. Username
2. Password
3. Cancel
Choose option (1-3): 1
New username: Nix
```

*Figure 26: modify personal details (username)*

We allow customers to change their username.

1	Andrew
2	456
3	Nixon
4	123
5	Bob
6	bobby123
7	

*Figure 27: database before modifying username*

1	Andrew
2	456
3	Nix
4	123
5	Bob
6	bobby123
7	

*Figure 28: database after modifying username*

```
1. Username
2. Password
3. Cancel
Choose option (1-3): 2
New password: nixi123
```

*Figure 29: modify personal details (password)*

We allow customers to change their password.



```
1 Andrew
2 456
3 Nix
4 123
5 Bob
6 bobby123
7
```

*Figure 30: database before modifying password*

```
1 Andrew
2 456
3 Nix
4 nixi123
5 Bob
6 bobby123
7
```

*Figure 31: database after modifying password*

View Rented Cars

```
Rented cars:

1. Car: Renault, Captur
   Rented on: 2021-06-09 19:52:21
   Status: Not returned
2. Car: Nissan, Almera
   Rented on: 2021-06-09 19:52:21
   Status: Returned
Press enter to continue...
```

*Figure 32: view rented cars*

We display all cars that has been rented along with when it is rented and return status.

## Cars To Be Rented

```
Cars to be rented:

Car: Nissan, Serena S-Hybrid
Description: Long drives out of the city while reducing carbon footprint.
Booked for 5 hours.
Total Price: RM 134.5

Car: Nissan, Serena S-Hybrid
Description: Long drives out of the city while reducing carbon footprint.
Booked for 1 days.
Total Price: RM 269.0

Press enter to continue...|
```

*Figure 33: cars to be rented*

We display all cars that the customer has booked that is ready to be rented out.

## Select And Book A Car For A Specific Duration

```
Car Index: 1
Car: Nissan, Almera
Description: City drives, running errands, road trips on a budget, and going meetings in town.
Cars Remaining: 1
Hourly Price: 5.9
Daily Price: 59.0

Car Index: 2
Car: Renault, Captur
Description: Interstate business trips, wooing your date, and stylish weekend getaways.
Cars Remaining: 1
Hourly Price: 11.9
Daily Price: 119.0

Car Index: 3
Car: Nissan, Teana
Description: Impressing business partners, outstation golf trips, and picking up your VIPs from the airport.
Cars Remaining: 1
Hourly Price: 24.9
Daily Price: 249.0

Car Index: 6
Car: Proton, Saga
Description: Slow and steady.
Cars Remaining: 4
Hourly Price: 10.0
Daily Price: 100.0

Select car index: |
```

*Figure 34: selecting a car index*

We let user select a car index from the menu which only shows cars that are available (remaining count that is greater than 0).

Select car index: 6

Do you want to rent the car in days or hours? Select '1' for days and '2' for hours

Choose option (1/2): 1

How many days do you want to rent the car: 2

*Figure 35: rent option*

We let customers choose if they want to rent in days or hours.

```
1 1,2,D,1|3,100,H,1|0,4,D,0
2 2021,06,02,22,33,47|2021,06,02,22,45,37|-
3 | 1,3,H,0|0,1,D,1|3,5,H,0|3,1,D,0
4 | 2021,06,09,19,52,21|2021,06,09,19,52,21|-|-
5
6
7
8
9 |
```

*Figure 36: database before rent*

```
1 1,2,D,1|3,100,H,1|0,4,D,0
2 2021,06,02,22,33,47|2021,06,02,22,45,37|-
3 | 1,3,H,0|0,1,D,1|3,5,H,0|3,1,D,0|5,2,D,0
4 | 2021,06,09,19,52,21|2021,06,09,19,52,21|-|-|-
5
6
7
8
9 |
```

*Figure 37: database after rent*

## Do Payment To Confirm Booking

```
Cars booked:

Nissan Serena S-Hybrid * 5 hours (RM 134.5)
Nissan Serena S-Hybrid * 1 days (RM 269.0)
Proton Saga * 2 days (RM 200.0)
The total price is: RM 603.5

Type '1' to pay, '2' to cancel payment.
Choose option (1/2): 1
```

Figure 38: display cars booked

We display all the cars that has been booked and also the total price calculated. We also allow customers to cancel payment if they want to.

```
1 1,2,D,1|3,100,H,1|0,4,D,0
2 2021,06,02,22,33,47|2021,06,02,22,45,37|-
3 1,3,H,0|0,1,D,1|3,5,H,0|3,1,D,0|5,2,D,0
4 2021,06,09,19,52,21|2021,06,09,19,52,21|-|-
5
6
7
8
9
```

Figure 39: database before payment

```
1 1,2,D,1|3,100,H,1|0,4,D,0
2 2021,06,02,22,33,47|2021,06,02,22,45,37|-
3 1,3,H,0|0,1,D,1|3,5,H,0|3,1,D,0|5,2,D,0
4 2021,06,09,19,52,21|2021,06,09,19,52,21|2021,06,15,15,07,19|2021,06,15,15,07,19|2021,06,15,15,07,19
5
6
7
8
9
```

Figure 40: database after payment

## Delete Account

```
Are you sure? y/n: y
```

Figure 41: delete account confirmation

We also allow customers to delete their account if they choose not to be a registered customer anymore.

1	1,2,D,1 3,100,H,1 0,4,D,0	1	Andrew
2	2021,06,02,22,33,47 2021,06,02,22,45,37 -	2	456
3	1,3,H,0 0,1,D,1 3,5,H,0 3,1,D,0 5,2,D,0	3	Nix
4	2021,06,09,19,52,21 2021,06,09,19,52,21 2021,06,15,15,07,19 2021,06,15,15,07,19 2021,06,15,15,07,19	4	nixi123
5		5	Bob
6		6	bobby123
7		7	
8			
9			

*Figure 42: database before account deletion*

1	1,2,D,1 3,100,H,1 0,4,D,0	1	Andrew
2	2021,06,02,22,33,47 2021,06,02,22,45,37 -	2	456
3		3	Bob
4		4	bobby123
5		5	
6			
7			

*Figure 43: database after account deletion*

We remove both the records and also the username and password.

## Exit Program

### Termination Confirmation

```
Do you want to continue? To exit to the Main Menu type '1', To Terminate Program type '2':  
Choose option (1/2): 1
```

*Figure 44: termination confirmation*

We ask the user to confirm if he/she wants to exit the program or not.

## Conclusion

We have ensured that the program that we wrote is robust, consistent, readable, performant, and future proof. This can be shown with a few examples:

### Abstraction

If you look back at Source 8 to Source 19 (all of our utility functions), you will notice that these functions are used again and again throughout the entire program. These functions are used to abstract away repetitive task so that we do not need to rewrite the same logic again and again.

### Functions in a list

```
def main() -> None:
    user_func = [admin, all_customer, registered_customer]

    user_type_list = [
        "MAIN MENU",
        "1. Admin",
        "2. All Customers (Registered / Not-Registered)",
        "3. Registered Customer",
        "4. Exit Program\n"
    ]

    while True:
        os.system("cls")
        print("Welcome to SUPER CAR RENTAL SERVICES!!!\n")
        for i in range(len(user_type_list)):
            print(user_type_list[i])

        no = get_user_int_range("Choose user(1-4): ", 1, 4)
        if no == 4: exit_program()
        else: user_func[no-1]()
```

*Source 41: main function*

Looking at our main function, what you will notice is that we do not use if else statement to check what numbers our user type. Instead, what we use is list indexing. We call the function that we store in a list earlier (`user_func`) based on the index that the user types. This prevents branching in our program.

### Consistency and Readability

In the first section of our [Source Code Explanation](#), we discuss about naming conventions and annotations. We have ensured that our code is consistent by following the naming conventions and insert appropriate annotations to explicitly tell other programmers what type of datatype that we expect in each function.

### Constant variables

To prevent our program from failing when the name of a certain database changes, we use constant variables to store the name of each database so that we can easily change the content of the variables when the name of our database changes. Note that these constant variables are not meant to be changed (although they can be changed technically) during the runtime of the entire program.



## Workload Matrix

### Subtask Workload

Note, different task contains different amount of work. Totaling up the workload is not equivalent to the person's contribution.

Task	Cheng Yi Heng	Tan Chun Hung
Python Programming	80%	20%
Introduction	10%	90%
Pseudocode	0%	100%
Flowchart	15%	85%
Program Source Code Explanation	85%	15%
Additional Features	100%	0%
Sample Input and Output	100%	0%
Conclusion	100%	0%

## References

Kinsley, H., 2018. Python Programming Tutorials. [online] Pythonprogramming.net. Available at: <<https://pythonprogramming.net/introduction-learn-python-3-tutorials/>> [Accessed 17 June 2021].