

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

**Module Name: PROGRAMMING WITH PYTHON**

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

**Hand Out Date: 22ND APRIL 2021**

**Hand In Date: 18TH 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 3](#_Toc74666624)

[Pseudocode 4](#_Toc74666625)

[Utility Functions 4](#_Toc74666626)

[Console Input Utility Functions 4](#_Toc74666627)

[I/O Utility Functions 6](#_Toc74666628)

[Login System 8](#_Toc74666629)

[Admin 9](#_Toc74666630)

[All Customers 15](#_Toc74666631)

[Registered Customers 16](#_Toc74666632)

[Main Program 21](#_Toc74666633)

[Flowchart 22](#_Toc74666634)

[Program Source Code 23](#_Toc74666635)

[Annotations 23](#_Toc74666636)

[Naming Conventions 23](#_Toc74666637)

[File Structure 23](#_Toc74666638)

[Utility Functions 26](#_Toc74666639)

[Console Input Utility Functions 26](#_Toc74666640)

[Console Input Utility Functions Summary 28](#_Toc74666641)

[I/O Utility Functions 29](#_Toc74666642)

[I/O Utility Functions Summary 31](#_Toc74666643)

[Login System 31](#_Toc74666644)

[Admin 34](#_Toc74666645)

[All Customers 42](#_Toc74666646)

[Registered Customers 44](#_Toc74666647)

[Main Program 50](#_Toc74666648)

[Additional Features 52](#_Toc74666649)

[Return to main menu 52](#_Toc74666650)

[Delete Customer Account 52](#_Toc74666651)

[Available Car Remaining Counter 53](#_Toc74666652)

[Hourly and Daily Renting Price 53](#_Toc74666653)

[Sample Input and Output 54](#_Toc74666654)

[Main Menu 54](#_Toc74666655)

[Welcome Page 54](#_Toc74666656)

[Admin 55](#_Toc74666657)

[Admin Main Menu 55](#_Toc74666658)

[Add Car 56](#_Toc74666659)

[Modify Car Details 57](#_Toc74666660)

[Display Records 59](#_Toc74666661)

[Search Records 60](#_Toc74666662)

[Return Rented Car 61](#_Toc74666663)

[All Customers 63](#_Toc74666664)

[All Customers Main Menu 63](#_Toc74666665)

[View All Cars Available For Rent 63](#_Toc74666666)

[Create new account 64](#_Toc74666667)

[Registered Customers 65](#_Toc74666668)

[Registered Customer Main Menu 65](#_Toc74666669)

[Modify Personal Details 66](#_Toc74666670)

[View Rented Cars 67](#_Toc74666671)

[Cars To Be Rented 68](#_Toc74666672)

[Select And Book A Car For A Specific Duration 68](#_Toc74666673)

[Do Payment To Conform Booking 70](#_Toc74666674)

[Delete Account 70](#_Toc74666675)

[Exit Program 72](#_Toc74666676)

[Termination Confirmation 72](#_Toc74666677)

[Conclusion 73](#_Toc74666678)

[Abstraction 73](#_Toc74666679)

[Functions in a list 73](#_Toc74666680)

[Consistency and Readability 73](#_Toc74666681)

[Constant variables 74](#_Toc74666682)

[References 75](#_Toc74666683)

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 customer’ 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 **normal customer (all customer**) can only view cars available for rent, which is car name and not inclusive of car’s rental price.

Lastly, we assume that for the select and book a car for a specific duration is our “shopping cart”. While do payment to confirm booking includes our “shopping cart” and pay option.

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 "|"

  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)

  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

        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[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 "|"

  ENDLOOP

  LOOP i FROM 0 TO len(car\_details)

    car\_details[i] = SPLIT car\_details[i] BY "|"

  ENDLOOP

  LOOP i FROM 0 TO len(customer\_names)

    DISPLAY i+1, customer\_names[i]

  ENDLOOP

  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

    ENDLOOP

  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

        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]

        )

        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

    ENDLOOP

  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"

  ENDLOOP

  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]

    ENDLOOP

    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 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]

      )

      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 "|"

  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 "\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

      DISPLAY "Total Price: ", "RM", total\_price + "\n"

      ENDIF

    ENDIF

  ENDLOOP

  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 "|"

  ENDLOOP

  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

  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

  total\_price = 0

  DISPLAY "\nCars booked:\n"

  LOOP idx FROM 0 TO 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

  ENDLOOP

  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 unpaid\_car\_indices

      dates[curr\_user\_idx][idx] = current\_time

    ENDLOOP

    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]

    ENDLOOP

    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]

    ENDLOOP

    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

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

1. 1,2,D,1|3,10,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
4. 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 are 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

1. Nixon
2. 321
3. Andrew
4. 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

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|Serena S-Hybrid|Long drives out of the city while reducing carbon footprint.|2
6. 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

1. 1,2,D,1|3,10,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
4. 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 whileloop.

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 has 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 if 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:

1. 1,2,D,1|3,10,H,1|0,4,D,0 *# rent detail*
2. 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(" "\*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)")

  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.

1. 1,2,D,1|3,10,H,1|0,4,D,0 *# rent detail*
2. 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 remaning 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

1. Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
2. 5.9|59.0

This is a piece of information from the cars.txt. Notice that there is a number at then 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

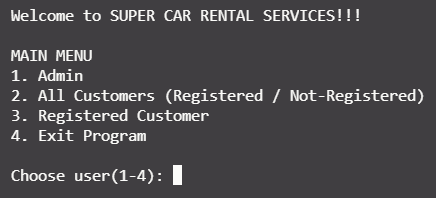
1. Nissan|Almera|City drives, running errands, road trips on a budget, and going meetings in town.|1
2. 5.9|59.0

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 more cheaper compared to renting daily. 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, the customer do not need to do any additional calculations.

Sample Input and Output

Main Menu

### Welcome Page

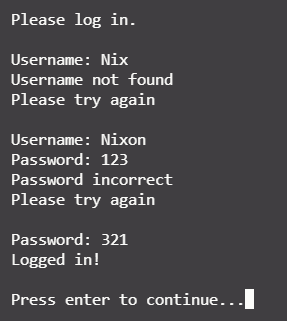


*Figure 1: main menu*

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

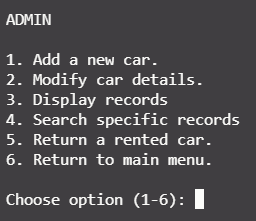
Admin

### Admin Main Menu



*Figure 2: admin login*

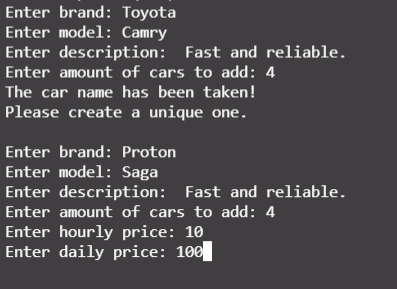
When entering the admin menu, we need to login before accessing the admin features. During login, we also check if the entered user name and password is correct or not. As you can see we prevent user from logging in if they type the wrong username or password.



*Figure 3: admin menu*

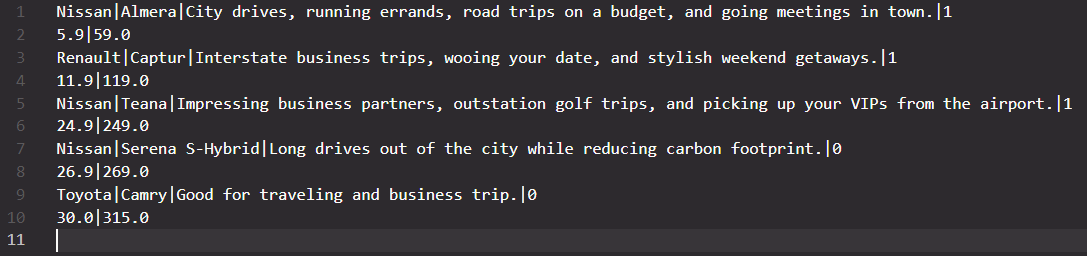
We displays all admin features in the admin menu.

### Add Car

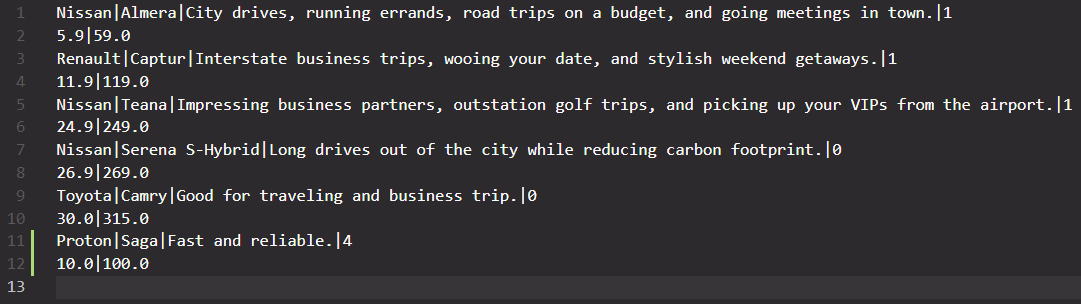


*Figure 3: add car*

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.

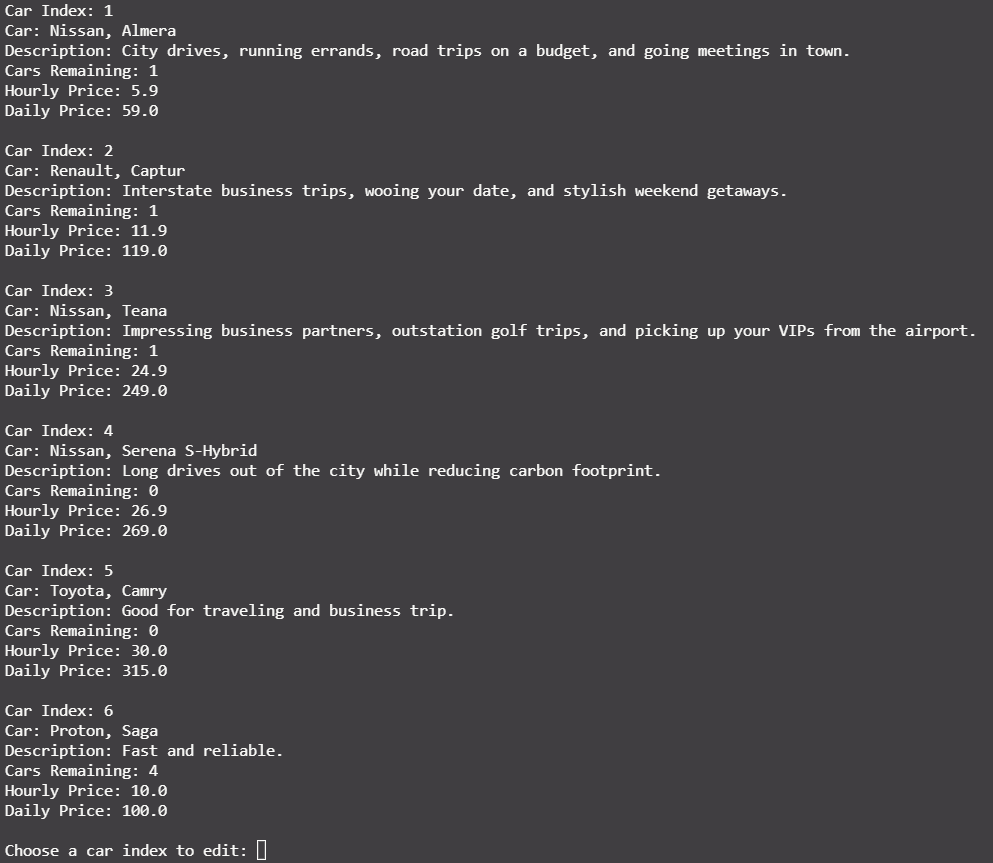


*Figure 4: database before adding car*



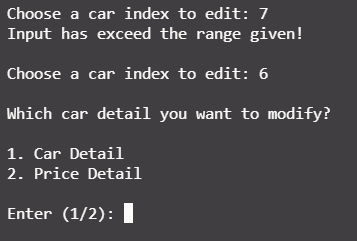
*Figure 5: database after adding car*

### Modify Car Details



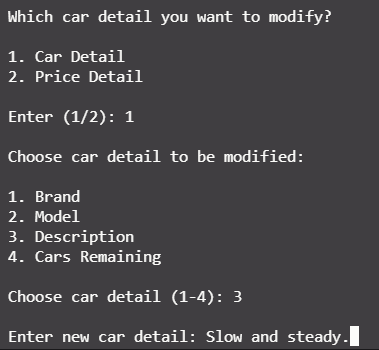
*Figure 6: modify car details (display)*

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

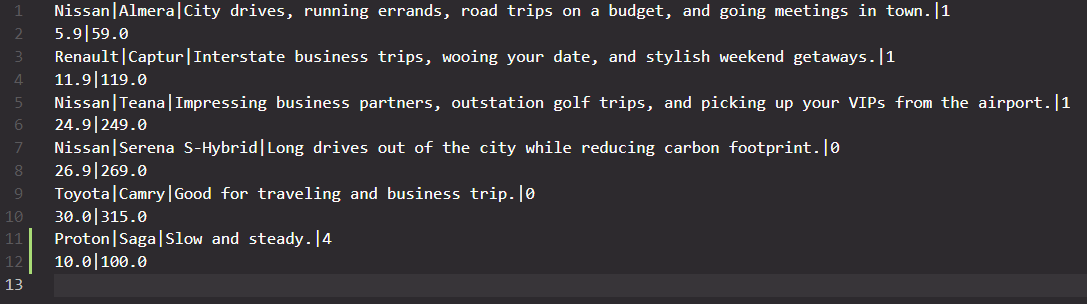


*Figure 7: modify car details (choosing a car)*

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.

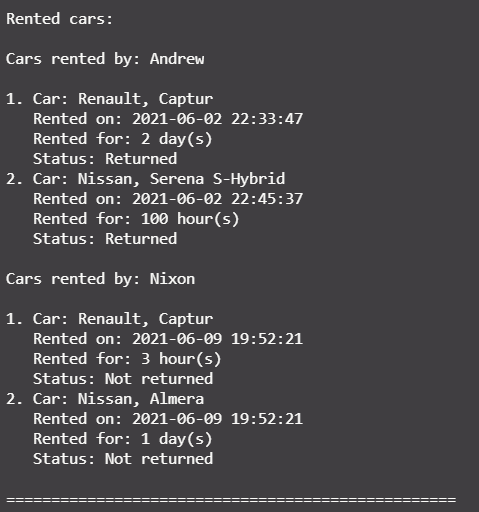


*Figure 8: modify car description*



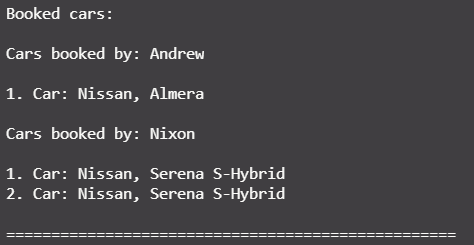
*Figure 9: database after modifying car*

### Display Records



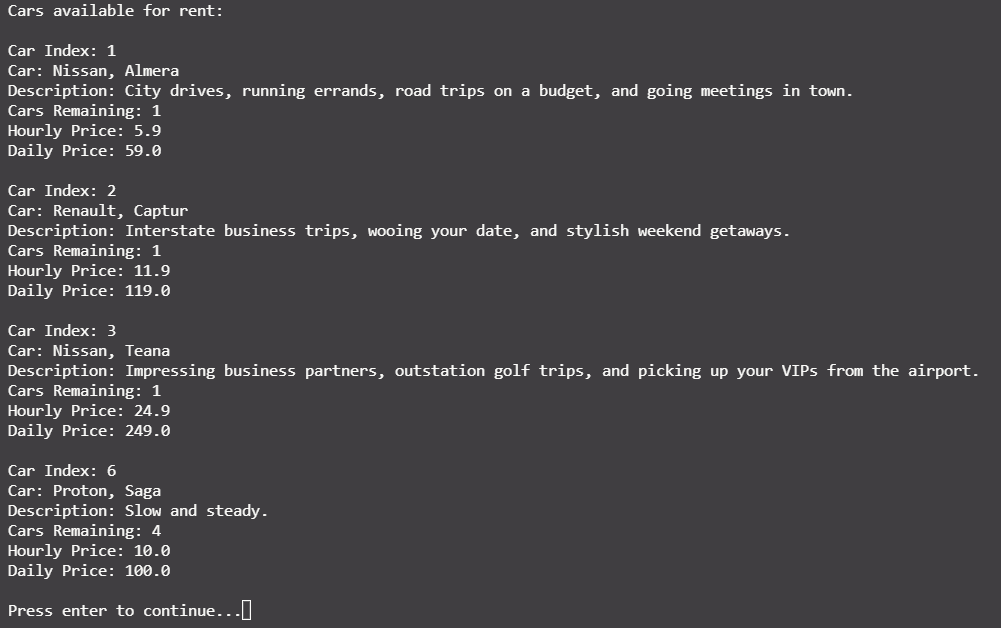
*Figure 10: display records (rented cars)*

We display all the rented cars.



*Figure 11: display records (booked cars)*

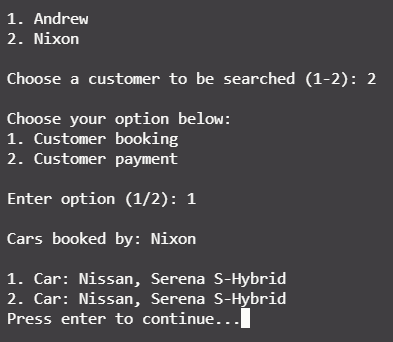
We display all the booked cars and who booked it.



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

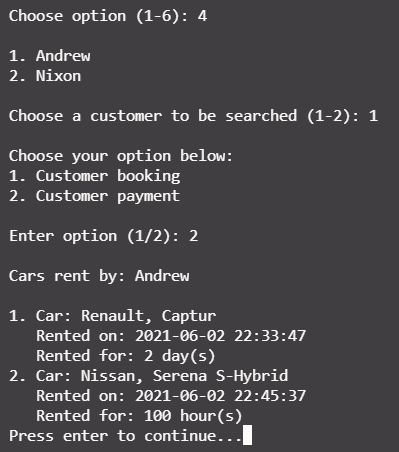
We display all the cars that are available for rental and booking.

### Search Records



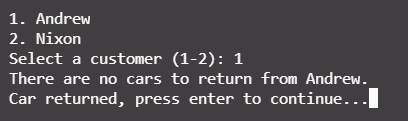
*Figure 13: search records (customer bookings)*

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



*Figure 14: search records (customer payments)*

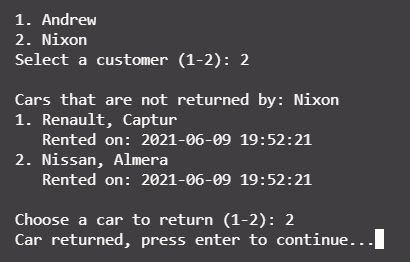
We allow admins to search a payment record for a specific customer.



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

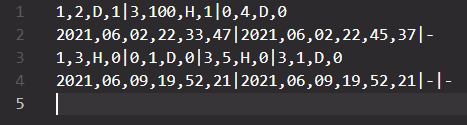
When there are no cars to be returned, we tell the admin.

### Return Rented Car

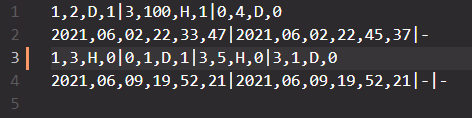


*Figure 16: returning cars (existing cars to return)*

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



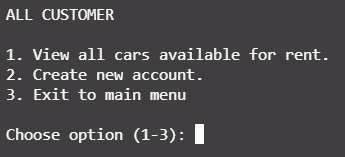
*Figure 17: database before return*



*Figure 18: database after returning*

All Customers

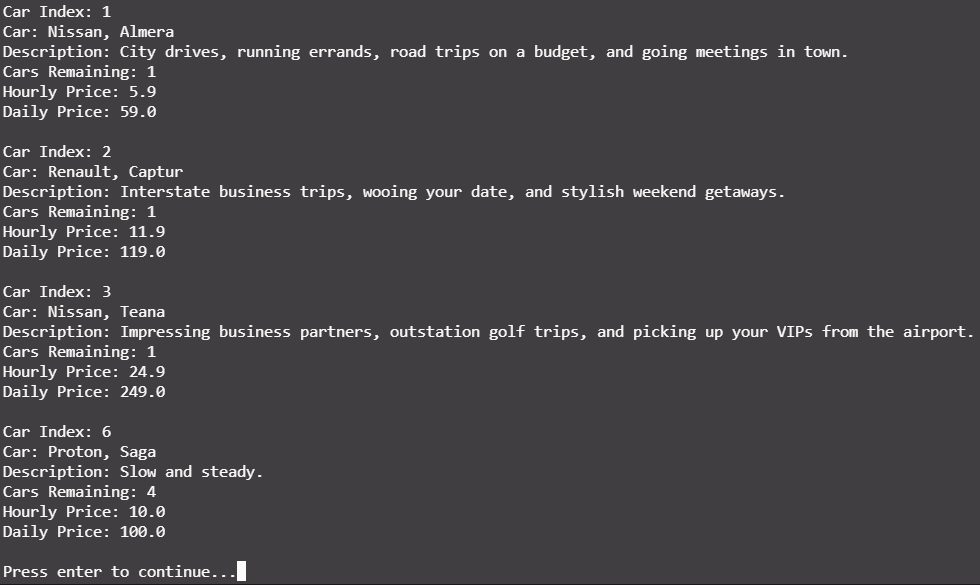
### All Customers Main Menu



*Figure 19: all customer menu*

We display all features available for all customers.

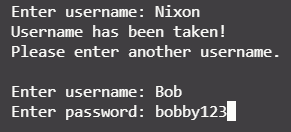
### View All Cars Available For Rent



*Figure 20: view all cars available for rent*

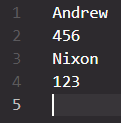
We display all cars that are available for rent.

### Create new account



*Figure 21: create new account*

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



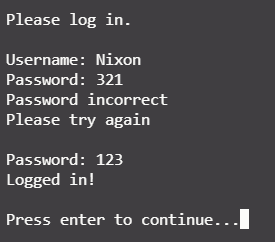
*Figure 22: database before creating new account*



*Figure 23: database after creating new account*

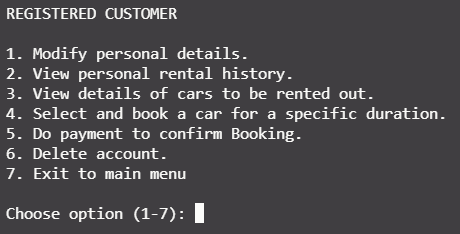
Registered Customers

### Registered Customer Main Menu



*Figure 24: logging in as registered customer*

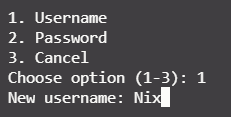
Similar to admin menu, we need to log in before entering the menu. We also make sure that the password and username is correct.



*Figure 25: registered customer menu*

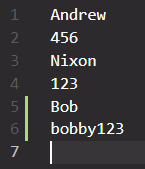
We display all the features included for the registered customer.

### Modify Personal Details

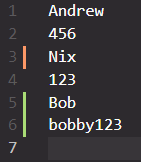


*Figure 26: modify personal details (username)*

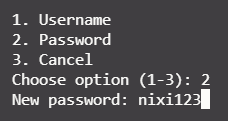
We allow customers to change their username.



*Figure 27: database before modifying username*

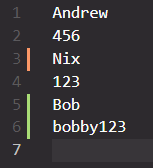


*Figure 28: database after modifying username*

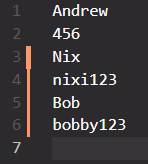


*Figure 29: modify personal details (password)*

We allow customers to change their password.

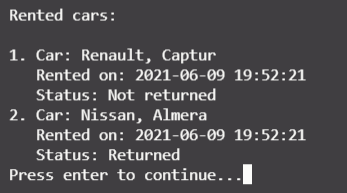


*Figure 30: database before modifying password*



*Figure 31: database after modifying password*

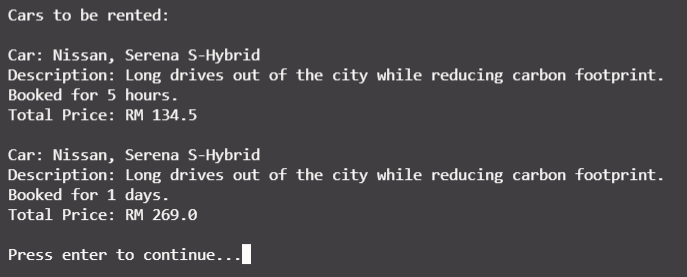
### View Rented Cars



*Figure 32: view rented cars*

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

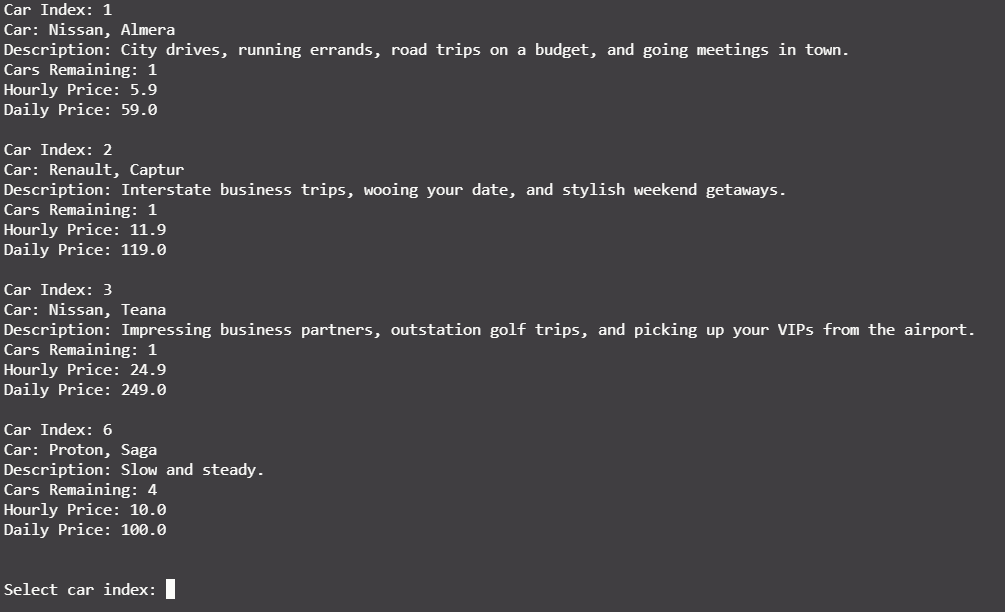
### Cars To Be Rented



*Figure 33: cars to be rented*

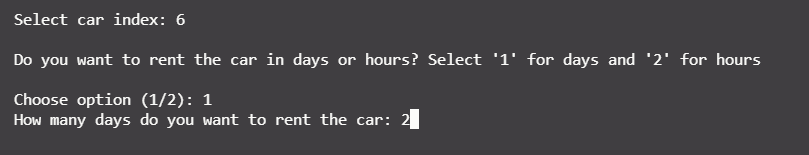
We display all cars that the customer has booked that is ready to be rented out.

### Select And Book A Car For A Specific Duration



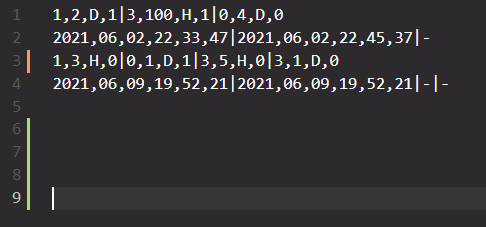
*Figure 34: selecting a car index*

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

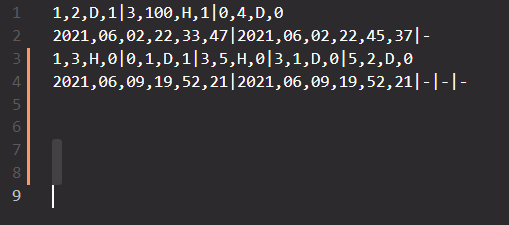


*Figure 35: rent option*

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

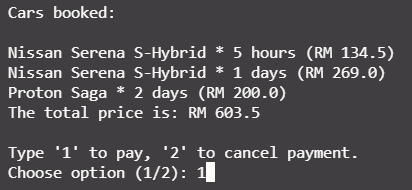


*Figure 36: database before rent*



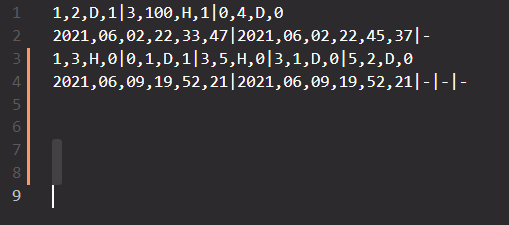
*Figure 37: database after rent*

### Do Payment To Conform Booking

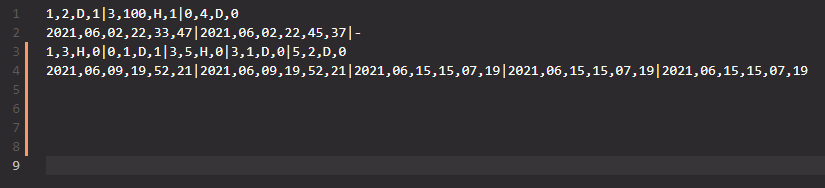


*Figure 38: display cars booked*

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.



*Figure 39: database before payment*



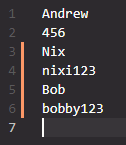
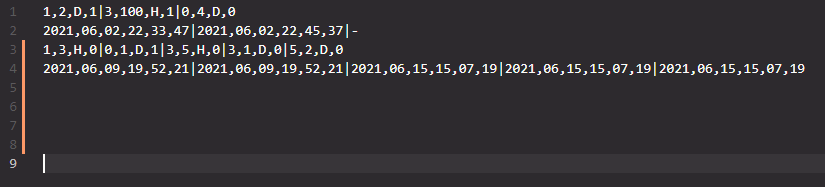
*Figure 40: database after payment*

### Delete Account

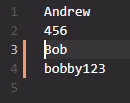
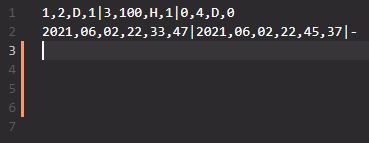


*Figure 41: delete account confirmation*

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



*Figure 42: database before account deletion*



*Figure 43: database after account deletion*

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

Exit Program

### Termination Confirmation



*Figure 44: termination confirmation*

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

# Conclusion

We have ensure 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. This 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 prevent 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 functions.

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.

References