Dear Piggy, My ayaan on your python skill upgrade

Project Title: Online Car Rental Platform

Project Objective:

Build an online car rental platform using Object-Oriented Programming in Python.

→ Try this when your to sykt ! learn Slytherin

Problem Statement:

A car rental company has requested you to build an online car rental platform where customers should be able to view the available cars that can be rented on an hourly, daily, or weekly basis. The company can display the available inventory and confirm requests by checking the available stock. Customers will receive an auto-generated bill when they return the car.

For simplicity, let's assume that:

1. Customers can rent cars from any one of the following options—hourly, daily, or weekly rental.
2. Customers are free to choose any number of cars they want, provided the number of available cars is more than the number of requested cars.

I →

Understand this → Flowchart it → this is ! Bustacks

You must use the following tools:

Jupyter Notebook: To create the module and main project files

II →

Should confirm to flowchart

Instructions to Perform:

1. Create a module (.py file) for car rental and import the built-in module DateTime to handle the rental time and bill.
2. Create a class for renting the cars and define a constructor in it.
3. Define a method for displaying the available cars. Also, define methods for renting cars on an hourly, daily and weekly basis, respectively.
4. Inside these methods, make sure that the number of requested cars is positive and lesser than the total available cars.
5. Store the time of renting a car in a variable, which can later be used in the bill while returning the car.
6. Define a method to return the cars using rental time, rental mode (hourly, daily, or weekly), and the number of cars rented.
7. Inside the return method; update the inventory stock, calculate the rental period, and

generate the final bill.

8. Create a class for customers and define a constructor in it.
9. Define methods for requesting the cars and returning them.
10. Next, create the main project (.ipynb) file and import the car rental module in it.
11. Define the main method and create objects for both car rental and customer classes.
12. Inside the main method, take the customer's input as a choice for displaying car availability, rental modes, or returning the cars.
13. Use the relevant method for the customer's input and print relevant messages.
14. Run the main method to start your project.

Breaking down the buisness logic:

Treat all these programs like a start of game → Data dies on closing the app.

Set game timer → carrent

I] We have a pool of cars

→ Store it in a set

{ type: nos, }

→ store it in list. Assume all cars are same

II] Business logic. {cars} {mycost per hour/day/week}

- if car available rent else refuse
- When rented set timer, rental option weekly/daily/hourly options. Update inventory.
- On return: (return time − rent time) × weekly/daily/hourly cost

Update inventory
Generate bill

→ Don't get overwhelmed & Zoom out if you are caught in minor details.

Visualising solution in OOP

This is like start of game. Moment you create it game starts

Class: Car Renting

attributes: Instance att {available cars}

methods:

Renting method → either have 1 or 3 (as per prob statement)

Return method →

Class:

Customer Name

Instance attr:

Name

Variable: Rent time

Methods:

→ also store no of cars hired by customer

→ Mode of hire for cars hired

Main Method

1 ⇒ Create class object type carrental
This starts the game

2. Create one customer obj from class customer
Either make a list of objs from name list or a single one will do.

3. Take customer input on screen
   ⇒ check availability
   ⇒ Rent
   ⇒ Return
   ⇒ End game / Exit

① → Call available() function
print list

② Rent → input no of cars
if available: check availability
method
↓
if true
↓
call rent car method
in user object

③ Return → call return fn
from user obj

Project Title: Online Car Rental Platform

Project Objective: Build an online car rental platform using Object-Oriented Programming in Python.

Problem Statement: A car rental company has requested you to build an online car rental platform where customers should be able to view the available cars that can be rented on an hourly, daily, or weekly basis. The company can display the available inventory and confirm requests by checking the available stock. Customers will receive an auto-generated bill when they return the car.

For simplicity, let's assume that:

1. Customers can rent cars from any one of the following options—hourly, daily, or weekly rental.
2. Customers are free to choose any number of cars they want, provided the number of available cars is more than the number of requested cars.

You must use the following tools: Jupyter Notebook: To create the module and main project files

Instructions to Perform:

1. Create a module (.py file) for car rental and import the built-in module DateTime to handle the rental time and bill.
2. Create a class for renting the cars and define a constructor in it.
3. Define a method for displaying the available cars. Also, define methods for renting cars on an hourly, daily and weekly basis, respectively.
4. Inside these methods, make sure that the number of requested cars is positive and lesser than the total available cars.
5. Store the time of renting a car in a variable, which can later be used in the bill while returning the car.
6. Define a method to return the cars using rental time, rental mode (hourly, daily, or weekly), and the number of cars rented.
7. Inside the return method; update the inventory stock, calculate the rental period, and generate the final bill.
8. Create a class for customers and define a constructor in it.
9. Define methods for requesting the cars and returning them.
10. Next, create the main project (.ipynb) file and import the car rental module in it.
11. Define the main method and create objects for both car rental and customer classes.
12. Inside the main method, take the customer's input as a choice for displaying car availability, rental modes, or returning the cars.
13. Use the relevant method for the customer's input and print relevant messages.
14. Run the main method to start your project.

Documentation: I have made two classes

1. CarRental solution
2. Customer The methods thers are documented in the commented portion. Ideally this solution should have an Class car with car inventory created in it. I have left the type variable unused to keep the solution simple. If type of vehicles have to be added the code will need modification A dictonary stores the inventoty{key-type: value number} I have used only one type of car the user is forced to enter alto.

Known buisness logic problem. Though you can make repeted hire. last time of hire will be stored in the customer object variable. This is limitation of the model. Expand to Class: Cars to capture all this data. the type of hire stored in the customer object is the last car hired. This implies while returning the car the logic checks what was your last hire mode and time and makes a bill accordingly. If statement ensure you cannot return more cars than hired or change the mode of hiring.

This is basic bare bones solution meeting the brief.

```
# Module for car rental operations
#Till the main methid you can create a seperate file and revoke it as a module
#Else just run it in same program

import datetime # This library has all the time function

# Class for managing car rentals
class CarRental:
    def __init__(self, available_cars):
        self.available_cars = available_cars
        # This is start of the game time counter
        self.rental_time = datetime.datetime.now()

#Method to check car availability Ser 1 of input list
    def display_available_cars(self):
        print("Available cars:")
        print(self.rental_time)
```

```
        for car_type, quantity in self.available_cars.items():
            print('We have only the following types of cars available: choose from them')
            print(f"{car_type}: {quantity}")

#Invoked on booking and checks if booking can go ahead
    # Define a function is available to show inventory
    def is_available(self,car_type, car_requested):
        if self.available_cars[car_type]>=car_requested and self.available_cars[car_type]!=0:
            return True
        else: return False
    # Define a rent car method in the class
    def rent_cars(self,car_type, rental_mode, number_of_cars ):
        #Update the inventory and print confirmation
        self.available_cars["alto"] -= number_of_cars
        print(f"You have rented {number_of_cars} of {car_type} cars on {rental_mode} basis.")



    def return_cars(self, car_type,rental_mode, number_of_cars, rental_start):
        #With a while loop I no longer need this check
        if rental_mode not in ["hourly", "daily", "weekly"]:
            print("Invalid rental mode.")
            return

        # Update inventory stock
        if rental_mode == "hourly":
            rental_period = (datetime.datetime.now() - rental_start).total_seconds() / 3600
        elif rental_mode == "daily":
            rental_period = (datetime.datetime.now() - rental_start).days
        else:
            rental_period = (datetime.datetime.now() - rental_start).days * 7

        # Enter data of charges in rupees for hire
        if rental_mode == "hourly":
            rental_cost = number_of_cars * rental_period * 400
        elif rental_mode == "daily":
            rental_cost = number_of_cars * rental_period * 3500
        else:
            rental_cost = number_of_cars * rental_period * 12000

        self.available_cars[car_type] += number_of_cars

        # Generate bill when car is returned.
        print(f"Bill:")
        print(f"Rental mode: {rental_mode}")
        print(f"Type of car: {car_type}")
        print(f"Number of cars: {number_of_cars}")
        print(f"Rental period: {rental_period} {rental_mode.upper()}")
        print(f"Rental cost: {rental_cost}")

# Class for representing customers
class Customer:
    def __init__(self, name):
        self.name = name
        # Initialising start time of the rental with a time variable
        self.rental_start = datetime.datetime.now()
        self.number_of_cars=0 # Stores how many cars customer has hired
        self.rental_mode=rental_mode # Stores mode of hire of customer
#Defining a method to book cars
    def request_cars(self, car_rental, car_type ,rental_mode1, number_of_cars1):
        if not car_rental.is_available(car_type, number_of_cars1):
            print(f"Not enough cars available for rental.")
            return

        car_rental.rent_cars(car_type,rental_mode1, number_of_cars1)
        # Update the instance variable denoting number of cars hired by customer
        self.number_of_cars += number_of_cars1
        #Update rental Mode
        self.rental_mode=rental_mode
        print(f"{self.name} has successfully rented {number_of_cars1} cars on{rental_mode1} basis")
        print(f' A total of {self.number_of_cars} have been rented')

#Defining a method to return cars
    def return_cars(self, car_rental,car_type, number_of_cars1, rental_mode1):
```

Sor items in dic

Can delete I also added input check.

Business logic

This variable is stored in obj: customer.

This is the Key

Called from method.

Calls rent car in obj: Game

Limitation is you have only one value. If you create obj car you should store there ideally

Called from method

```
        if number_of_cars1> self.number_of_cars or rental_mode1 != self.rental_mode:
            print(f'you have hired only {self.number_of_cars} of cars on {self.rental_mode} basis ')
            print('cars to be returned to be less than or equal to alrady hired')
            print('mode of hire at the time of return cannot change')
            print('reneter the number of cars to be returned with rental mode')
            return
        car_rental.return_cars(car_type,rental_mode1, number_of_cars1, self.rental_start)
        print(f"{self.name} has successfully returned {number_of_cars1} cars.")
#Main Method
# This method helps you to run the main program without importing a module
#Will work for chota problem. Import mudule is the better way.

if __name__ == "__main__":
    # Available cars stored in a dictonary with key as type of car
    #I have kept only one type in the dictonary
    available_cars = {"alto": 21}

    # Create car rental object
    car_rental = CarRental(available_cars)
# Welcome message for portal

    print( 'Welcome to car rental app')
    # Create customer object
    customer = Customer(input('Enter your Name :'))

    # Main menu
    while True:
        print(f'hi!{customer.name} please choose your options')
        print("1. Check car availability")
        print("2. Rent cars")
        print("3. Return cars")
        print("4. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            # Check car availability
            car_rental.display_available_cars()
        elif choice == 2:
            # Rent cars
            while True:
                car_type = input('Choose Car type(only alto available): ').strip().lower()
                if car_type=='alto':
                    break
            while True:
                rental_mode = input("Choose rental mode (hourly, daily, or weekly): ").strip().lower()
                if rental_mode=='hourly' or rental_mode=='daily' or rental_mode=='weekly':
                    break
            number_of_cars = int(input("Enter number of cars to rent: "))
            # call the car rent method from customer object
            customer.request_cars(car_rental, car_type ,rental_mode, number_of_cars)
            # Update the number of cars customer has hired and reset rental mode
            #customer.number_of_cars+=number_of_cars
            #this is a known logic problem. The last hire mode will reset the attribute
            #In the customer object
            #customer.rental_mode=rental_mode
            # debugging line print(f'line 135 {customer.number_of_cars} total cars requested by customer on {customer.rental_mo
            #Customer hire time. This is stored as a variable in the Customer obj
            #Logic problem the latest hire will reset the time of hire
            #This is limitation of the data model.
            customer.rental_time = datetime.datetime.now()
        elif choice == 3:
            # Return cars
            #Typical while loop looping to get correct entry
            while True:
                car_type = input('Choose Car type(only alto available): ').strip().lower()
                if car_type=='alto':
                    break
            while True:
                rental_mode = input("Choose rental mode (hourly, daily, or weekly): ").strip().lower()
                if rental_mode=='hourly' or rental_mode=='daily' or rental_mode=='weekly':
                    break

            number_of_cars = int(input("Enter number of cars to return: "))
```

```
165                 number_of_cars = int(input( Enter number of cars to return.  ))
166                 customer.return_cars(car_rental,car_type, number_of_cars, rental_mode)
167             elif choice == 4:
168                 print("Thank you for using our car rental service.")
169                 break
170             else:
171                 print("Invalid choice. Please enter a valid number.")
```

```
ra has successfully rented 1 cars onhaily basis
 A total of 1 have been rented
hi!ra please choose your options
1. Check car availability
2. Rent cars
3. Return cars
4. Exit
Enter your choice: 2
Choose Car type(only alto available): alto
Choose rental mode (hourly, daily, or weekly): hourly
Enter number of cars to rent: 1
You have rented 1 of alto cars on hourly basis.
ra has successfully rented 1 cars onhourly basis
 A total of 2 have been rented
hi!ra please choose your options
1. Check car availability
2. Rent cars
3. Return cars
4. Exit
Enter your choice: 3
Choose Car type(only alto available): alto
Choose rental mode (hourly, daily, or weekly): daily
Enter number of cars to return: 1
you have hired only 2 of cars on hourly basis
cars to be returned to be less than or equal to alrady hired
mode of hire at the time of return cannot change
reneter the number of cars to be returned with rental mode
hi!ra please choose your options
1. Check car availability
2. Rent cars
3. Return cars
4. Exit
Enter your choice: 1
Available cars:
2024-01-14 21:15:31.067356
We have only the following types of cars available: choose from them
alto: 19
hi!ra please choose your options
1. Check car availability
2. Rent cars
3. Return cars
4. Exit
Enter your choice: 3
Choose Car type(only alto available): alto
Choose rental mode (hourly, daily, or weekly): hourly
Enter number of cars to return: 1
Bill:
Rental mode: hourly
Type of car: alto
Number of cars: 1
Rental period: 0.036044865833333335 HOURLY
Rental cost: 14.417946333333335
ra has successfully returned 1 cars.
hi!ra please choose your options
1. Check car availability
2. Rent cars
3. Return cars
4. Exit
Enter your choice:
```

Message if you try to return more than you hire.

as we return in a jiffy

Variables even in this tiny prog became huge. Imp to know ⇒ Less the static all variables are destroyed in the class

⇒ Must recreate them

⇒ I have unnecessarily created new names as understood this concept after debugging

Known Logic Flaws

→ The time of hire is stored in customer obj. Hence you can store only one.

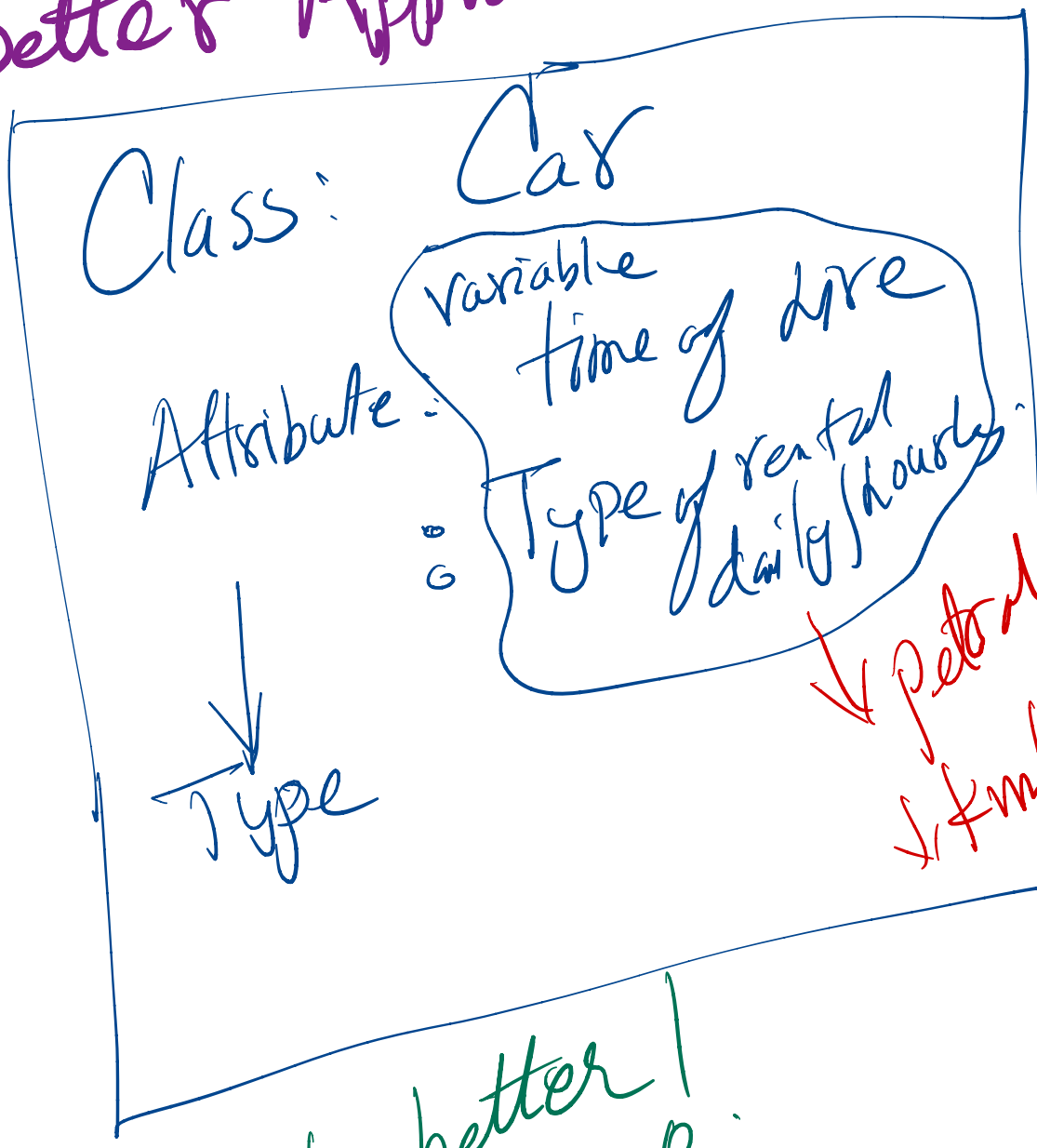
→ Repeated hire this value is overwritten.

→ Also type of hire is again stored in customer ⇒ single value over written.

→ On return the last values of car ~~hire~~ hired, time & mode are taken to find cost.

Obvious limitation

Better Approach →



Would be better!

Next proj

Changeability logic to add petrol&map

(corresponding store petrol status on hire for in customer &fn on hire

Better still :

Logically you should have a Transaction Class :

Parameter (customer, car)

Attrib : Time of hire, mode of hire ?

Maybe in cardobj

Now each customer can hire multiple car multiple times & each is treated as is seperate tran.

Each transaction is a obj in this

The bills should be generated here

→ Encapsulation : Concept

If you wanna implement this create child class & iden data you want to protect.

Backend:

All data evaporates when solution restarts.
SQL must come into play here. Next module

Import a file library to access file fns.

Write data to the csv file on each
run (using Colab? store it in google drive)

Retrieve it when solution starts.

Play with this data using matplot

Multiple Customers

Admin console to update data.
Enter multiple sta customers.

Current UI is bad. Import some
libs to make slick dropdown & slide
selector

Keep improving your projs as you learn

or

Should you learn !

Mmm

I am really not sure about the trajectory of tech.

- For artsy kinds like you:
  - → focus on driving the car not opening the hood
  - → Yeah write 1000 lines algos to render your image in photoshop
  - → you <human> decide what looks good
- Learning python or any other language help you think logically & creatively
- By the time you are out of college writing code will be done by machine <Replacable skills>
- Learning to how to unambiguously communicate with machines is core

- Try this if you don't believe:
  - → Take any real world problem.
  - → Write it in plain english step by step (your flowchart also)
    Consider all conditionals
  - → Get it generated on Bing.
  - → Troubleshoot and execute
    - ← the troubleshooting is getting better so now this will be a replicable skill in future)
- Don't waste more than two weeks in this. If you don't crack it, you're not paying for any of your courses use skillshare subscription.
- All my files are in our G drive
- I have completed the best I could. After 2 months I will be 'initialised' to ∅.
- After paying 160K feel like in Barbaque nation' force feeding mtech. Vacation in Goa was better spent 😊 Bye