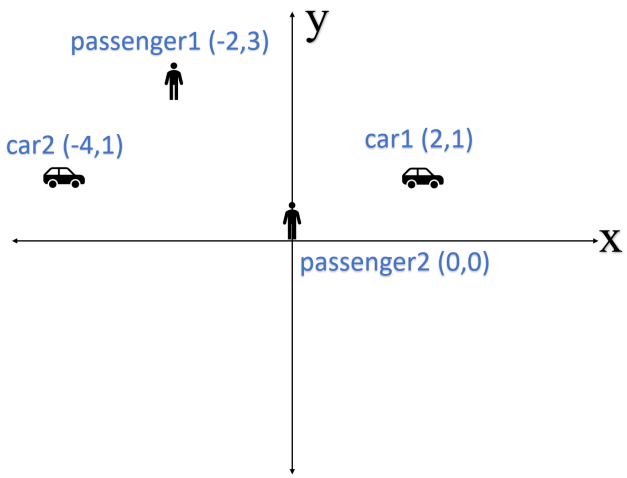
A Simple Ride-Sharing System



Let's consider a two-dimensional city where positions are defined using x and y coordinates. Within this city, multiple cars and passengers are situated at various locations. As illustrated in the image above, you can see the positions of two cars and two passengers. The cars, passengers, and their respective positions can be represented by the 'Car,' 'Passenger,' and 'Location' classes respectively. The UML diagrams depicting these classes are provided below. This city operates a straightforward ride-sharing system, which is represented by the 'RideSharingApp' class in the UML diagram below.

Location
+ x : int + y : int
+ __init__() + __str__()

Car
+ car_name : String + location : Location + cost_per_mile : float
+__init__() +__str__() +move_to()

Passenger
+ passenger_name : String + location : Location
+ __init__() + __str__() + move_to()

RideSharingApp
+ cars : Car [] + passengers : Passenger []
+ __init__() + add_car() + add_passenger() + remove_car() + remove_passenger() + find_cheapest_car() + find_nearest_car()

UML Class Diagrams

Implement these four classes using Python. Below is a brief description of those classes:

1. **Location**

This class has 2 data attributes and 2 methods.

Data attributes

x - An integer value representing the x-coordinate

y - An integer value representing the y-coordinate

Methods

__init__(self, x, y): It is the constructor of the class. It should assign the given parameters to the corresponding data attributes.

__str__(self): This method should return a string representation of the location object. The format of the string should be similar to - '(x,y)'

2. **Car**

This class has 3 data attributes and 3 methods.

Data attributes

car_name - A string representing the name of the car.

location – An object of the 'Location' class representing the position of the car within the 2-dimensional space.

cost_per_mile – A floating point number representing the car's fare per mile.

Methods

__init__(self, name, location, cost): It is the constructor of the class. It should assign the given parameters to the corresponding data attributes.

__str__(self): This method should return a string representation of a car object. The format of the string should be similar to - '[car_name, location, cost_per_mile]'

move_to(self, new_x, new_y): this method should update the x and y coordinates of the 'location' attribute to new_x and new_y

3. Passenger

This class has 2 data attributes and 3 methods.

Data attributes

passenger_name - A string representing the name of the passenger.

location – An object of the 'Location' class representing the position of the passenger within the 2-dimensional space.

Methods

__init__(self, name, location): It is the constructor of the class. It should assign the given parameters to the corresponding data attributes.

__str__(self): This method should return a string representation of a passenger object. The format of the string should be similar to – '[passenger_name, location]'

move_to(self, new_x, new_y): this method should update the x and y coordinates of the 'location' attribute to new_x and new_y.

4. RideSharingApp

This class has 2 data attributes and 7 instance methods.

Data attributes

cars – A list of 'Car' objects

passengers – A list of 'Passenger' objects

Methods

__init__(self): It is the constructor of the class. It should set the 'cars' and 'passengers' attributes to empty lists.

add_car(self, *car*) - adds the given *car* object to the 'cars' attribute

add_passenger(self, *passenger*) – adds the given *passenger* object to the 'passengers' attribute

remove_car(self, *car*) – removes the given *car* object from the 'cars' attribute

remove_passenger(self, *passenger*) - removes the given *passenger* object from the 'passengers' attribute

find_cheapest_car(self) – This method finds the cheapest car and prints the string representation of that car. The cheapest car is determined based on the 'cost_per_mile' attribute of the car objects.

find_nearest_car(self, *passenger*) – This method finds the nearest car for the given '*passenger'* object and prints the string representation of that car. The nearest car

is determined based on the distance between the *passenger* and the available cars. For example, if a passenger's location is (x_1, y_1) and a car's location is (x_2, y_2), their distance can be calculated according to:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Submission Instructions

(Please follow the instructions carefully and submit accordingly.)

- Name your source code file as "FULL_NAME_HW1.py"
- Submit this file in iCollege folder '**Homework1**'
- Due date: Fri, 9/19/2025 11:59 PM
- Late submission will be accepted until: Mon, 9/22/2025 11:59 PM

The late submissions penalty will be determined based on the following formula:

PENALTY = 0.4 * NUMBER_OF_HOURS_LATE

Examples:

If your submission is 2 hours late, PENALTY = 0.8%
If your submission is 24 hours late, PENALTY = 9.6%
If your submission is 72 hours late, PENALTY = 28.8%

Note:
-All submissions must be made through iCollege. No email submission will be accepted.

Grading Breakdown:

Task	Points
#1: Proper Implementation of the 'Location' class	10
#2: Proper Implementation of the 'Car' class	20
#3: Proper Implementation of the 'Passenger' class	20
#4: Proper Implementation of the 'RideSharingApp' class	40
#5: Comments	10

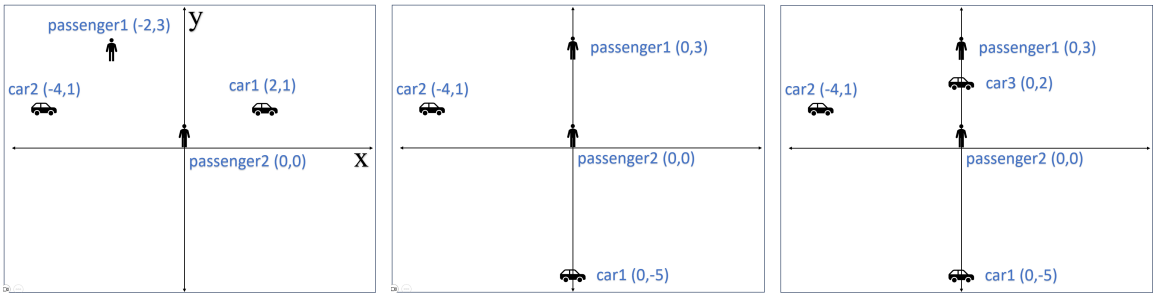
Note: Add around 10-12 comments in your script to explain how your code works. Comments should be meaningful and concise.

Example code:

A sample source code file has been attached (hw1.py), please start writing your solution using this template. You need to complete the #TODO sections only.

Test Case Scenarios and Sample Outputs:

Initially, the city has two cars and two passengers as depicted in Scenario 1 below. Next, the 'car1' and 'passenger1' are moved to new positions (Scenario 2). Finally, a new car (car3) is added to the city (Scenario 3).



Scenario 1 **Scenario 2** **Scenario 3**

If your class implementations are correct, the program produce output similar to:

----------Object creation----------

Car object 1 created: [car1, (2,1), 0.61]

Car object 2 created: [car2, (-4,1), 0.5]

Passenger object 1 created: [passenger1, (-2,3)]

Passenger object 2 created: [passenger2, (0,0)]

----------Scenario 1----------

Cheapest car available: car2, Cost per mile: 0.5

Nearest car for passenger1: car2, Distance: 2.83

Nearest car for passenger2: car1, Distance: 2.24

----------Scenario 2----------

car1's location has been updated: [car1, (0,-5), 0.61]

passenger1's location has been updated: [passenger1, (0,3)]

Cheapest car available: car2, Cost per mile: 0.5

Nearest car for passenger1: car2, Distance: 4.47

Nearest car for passenger2: car2, Distance: 4.12

----------Scenario 3----------

New car added: [car3, (0,2), 0.3]

Cheapest car available: car3, Cost per mile: 0.3

Nearest car for passenger1: car3, Distance: 1.00

Nearest car for passenger2: car3, Distance: 2.00

Notes:

- You submission must be your own work. A student who submits an assignment that copies the work of another student, in whole or in part, will be assigned a grade of zero.
- Use of generative AI tools like ChatGPT is prohibited for this homework assignment. Prohibited AI usage includes idea/code generation, revising your work, and so on.
- Five students will be randomly selected to explain their solution to the instructor/TAs. Failure to do so may result in a deduction from your homework score.