# *Introduction to Programming Concepts (ITAS185)*

# *Assignment 3 – Summative*

Date assigned: November 27, 2023

Date due: **December 13, 2023**

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

1. Create Class Diagrams for Python classes with Inheritance
2. Implement a Python Application using Inheritance and everything else we have learned this term.

**To be uploaded to the portal:**

1. A zipped folder called ***username*\_185A03\_Summative** that contains the following files:
   1. The ***username\_*185A03\_ClassDiagram.docx** Word document containing the Class diagram for your project.
   2. All your source files
   3. A Microsoft Word document called ***username\_*185A03\_SelfAssessment.docx**.

**To Start:**

1. Create a folder called ***username*\_185A03\_Summative** in your **Assignments** folder and download the input files to the folder.
2. Download **Self Assessment.docx** from **the portal** to your **Assignments\*username*\_185A03\_Summative** folder.

**To Do:**

We will be building a program to simulate a few cars having a race. All classes and methods other than basic set and get methods need to be commented. Single line comments should be used to help lines of code that are not obvious to someone else reading through your program. Marks will be deducted for missing comments, and if your code is not indented properly.

You must start by completing the class diagram for the entire project (Vehicle, Motorcycle, Truck, Tesla and RaceTrack classes). I would use Miro (free online software) or MS Visio (free with your Microsoft Software Account) to draw the class diagram, but you can choose any tool you choose as long as you can take a copy of it and put it in the document ***username*\_185A03\_ClassDiagram.docx.** This document also goes in the **Assignments\*username*\_185A03\_Summative** folder.

The Vehicle class and all of its subclasses and the RaceTrack class should be in a subfolder of *username*\_185A03\_Summative called classes. The main program (race\_run.py) should be in the folder *username*\_185A03\_Summative

# Vehicle class

1. Create an **abstract** Vehicle class (use the abc standard library) that has attributes for:
   1. model (str)
   2. colour (str)
   3. speed (float)
   4. position (float)
2. The Vehicle class must have a constructor that requires **ONLY model and colour**. The other two fields (e.g. speed, position) should be set to zero. All class fields/attributes should be **private**.

# Vehicle methods

1. Write getter methods to ‘get’ or retrieve the values for all the class fields/attributes (get\_model(self), get\_colour(self), get\_speed(self), get\_position(self)). Make sure your getter methods specifies the types of parameters being sent to and returned from the methods ( -> str) ( -> float) ( -> None) etc.
2. You do not need to write setter methods to set all the values, the only ‘set’ methods you should write is one to set the position, and one to set the speed (set\_position(self, value) and set\_speed(self, value)). Make sure your getter methods specifies the types of parameters being sent to and returned from the methods.
3. Write a method that returns nothing named **move()** that changes to the value of the position to be the current position plus the current speed.
4. **Write an abstract method named accelerate()** – that does nothing since it is abstract!
5. **Write an abstract method named get\_icon()** – this method does nothing because it is abstract.
6. Write a **get\_position\_int()** method that returns the position as an **int**! (it is a float in the class, and must be returned as an int).
7. Write the \_\_repr\_\_**()** method that returns a nicely formatted string that includes all give attributes for each vehicle – for example the String could look something like: (Model: KawasakiNinja, Colour: Green, Speed: 10.0, Position: 2).
8. Write the \_\_str\_\_() method to return a string representation of the vehicle. This will work as part of the \_\_str\_\_() method of the child classes.
9. Use any other dunder methods you wish for the vehicle class.

# Motorcycle class

1. Create a Motorcycle class that Inherits from (is a child of) Vehicle.
2. The Motorcycle class should have a constructor that takes fields for model and colour. The Motorcycle class should pass the model and colour parameters to the parent class.
3. Add the **accelerate()** method which takes no parameters and returns no value. Inside this method, you should **add the current speed with the accel variable to update the speed of the motorcycle by a value of 0.6.** For example, if a motorcycle had a speed of 10.0, and the value of accel was 0.6, when you call accelerate() the car should now have a speed of 10.6.
4. Add the **get\_icon()** method to return the String “M” or a single character value of your choice.
5. Add any other dunder methods you think you need. If a dunder method is used the same way in multiple child classes, add it to the parent class instead.
6. \_\_str\_\_() method that displays the motorcycle information nicely formatted and uses the parent \_\_str\_\_ class to help.

# Truck class

1. Create a Truck class that Inherits from (is a child of) Vehicle, and has the following extra attribute:
   * + Boolean is\_diesel (make this private)
2. The Truck class has a constructor that takes fields for model and colour, and whether or not it is a diesel (a third Boolean parameter). The Truck class should pass the model and colour parameters to the parent Constructor.
3. Override the accelerate() method which takes no parameters and returns no value. Inside this method, you should **add the current speed with the accel variable to update the speed of the truck by 0.4 if it is a diesel, and 0.5 if it is not diesel.**
4. Override the **get**\_i**con()** method to return the String “T” or an appropriate single character value that can be displayed.
5. Add any other dunder methods you think you need. If a dunder method is used the same way in multiple child classes, add it to the parent class instead.
6. \_\_str\_\_() method that displays the truck information nicely formatted and uses the parent \_\_str\_\_ class to help.
7. **Tesla class**
8. Create a Tesla class that Inherits from (is a child of) Vehicle, and has the following extras:
   * + boolean is\_two\_motor
9. The Tesla class has a constructor that takes fields for model, colour and whether or not it is a performance two motor version (the extra boolean parameter). The Tesla class should pass the model and colour parameters to the parent constructor.
10. Override the **accelerate()** method which takes no parameters and returns no value. Inside this method, you should **add the current speed with the accel variable to update the speed of the Tesla by** a value of 0.7 if it is a two motor Tesla, and 0.6 if it is not.
11. Override the **get\_icon()** method to return “E” (for Electric, or Elon Musk, just something other than T or an appropriate single character value that can be displayed).
12. Add any other dunder methods you think you need. If a dunder method is used the same way in multiple child classes, add it to the parent class instead.
13. \_\_str\_\_() method that displays the tesla information nicely formatted and uses the parent \_\_str\_\_ class to help.
14. **Setting up the Race Track**

Create a new class called RaceTrack.py. This class is responsible for drawing the racetrack that the vehicles will race on.

1. The RaceTrack class has attributes for:
   1. name (str) – Default “ITAS Motor Speedway”
   2. length (int) – Default 20
2. The RaceTrack class must have a constructor that requires both the attributes. Both attributes should be **private**.
3. **RaceTrack methods**
4. Write getter methods to ‘get’ or retrieve the values for all the class fields/attributes (get\_name(self), get\_length(self)). Make sure your getter methods specifies the types of parameters being sent to and returned from the methods.
5. You do not need to write setter methods as the values can only be set at instantiation.
6. \_\_str\_\_() method that is passed a list of vehicles and their positions and returns a string of the racetrack to be printed.
7. champion which is passed the winning object and displays the information about the winning object and a congratulatory message.

# File of Vehicles

I am providing you with three file which contain the vehicle information in from a text file. I’ve provided a sample file below. The file contents should look like:

Motorcycle,Kawasaki Ninja,Green,

Truck,Ford F150,Red,False,

Truck,Ford 150,Purple, True,

Tesla,S90,White,True,

Tesla,Model 3,Black,False,

Motorcycle,Yamaha Scooter,Blue,

1. **race\_run.py**
2. At the start of the program, prompt the user for the name of the file that contains the vehicle data for the race. Use exceptions to make sure that the requested file exists. Prompt until the user enters a valid filename or chooses to exit.
3. When a valid filename has been entered, open the file and read it. Each line of the file contains information on a specific vehicle. For each line in the file, create an object of the appropriate type of vehicle and append the object to a list called race\_vehicles.
   1. The delimiter parameter in the file is a comma (,) (can use split method for this).
   2. The first value of each line determines the type of vehicle you need to create. For Motorcycle the line has three ‘words’ which are the values to create a Motorcycle object – the name of the class, the model and the colour.
   3. For each line starting with Truck or Tesla data there should be four words in a line – the name of the class, the model, the colour and the fourth boolean parameter.
   4. For each line, create the appropriate Vehicle object – you’ll need to have a set of if/elif statements to check if the first word is a Motorcycle, create a new Motorcycle object and append this to the race\_vehicles list.
4. I recommend using the following algorithm:
   1. Read the line from the file.
   2. Split the line at the comma (,) to create a list of the attributes.
   3. Check the first item in the list.
   4. Call the appropriate constructor to create an object of that type.
   5. Append the object to the race\_vehicles list.
   6. Repeat steps i to v until no more vehicles.
5. Construct an object of type RaceTrack for the current race (optionally prompt the user for the track name and length).
6. Use a (while) loop that will run until some vehicle has won the race. You need to keep track of the rounds; use a descriptive variable such as ‘round’. Declare this variable and set it to 1 before starting your while loop. The algorithm about how to simulate the race is:

Within your while loop:

* 1. Call the \_\_str\_\_ method of the race track with the current race\_vehicles list. The algorithm to display the racetrack is below.
  2. At the end of the outer while loop, if no vehicle has reached position 20 or more, loop through all the race\_vehicles list and call each vehicle’s accelerate() method, then that vehicle’s move() method.
  3. Increment the round counter!

1. **Displaying the racetrack (text-based)**
2. A picture containing timeline

   Description automatically generatedThis is an algorithm you could use to display the racetrack (\_\_str\_\_ method of RaceTrack object)
   1. Print out which round of the race it is.
   2. Print out the racetrack by printing out a row to the screen for up to the length of the track. Use the ‘|’ character to show lane markers between the vehicles.
   3. Write a for loop that goes from 0 to the length of the track.
   4. For this loop position print out a | character to show the left side of the lane
   5. Write another loop that goes through the race\_vehicle list.
      * 1. if the INTEGER position of the next vehicle is equal to the position counter variable, use the vehicle’s get\_icon() method to retrieve the icon and draw the icon for the vehicle, else draw a space(s)
3. Make sure you use the appropriate methods like get\_position\_int in order to determine where to print out the vehicle.
4. Make sure you use the number of vehicles in the file read in to display a separate lane for each vehicle in the race. As per the image.
5. **Add Randomness**

After you get the entire program running as above (the race should finish the same way each time it is run). Modify the code so that each time the accelerate method of a class is called, there is a random factor added. This is done by changing the accelerate method of the parent class (Vehicle) to determine the acceleration variance. This variance is determined using the choices method of the random library. The algorithm for the accelerate method of the Vehicle class changes to:

1. Declare a list called values with the values -.2, -.1, 0, .1, .2 (you can use your own values if you want, but this is what I recommend). Call the list acceleration\_change.
2. Create a weight\_values tuple with a value for each value in the acceleration\_change list. This list will be used to determine how likely it is that a given number in the acceleration\_change list is chosen. I recommend starting with (1, 2, 6, 2, 1) so that it is most likely that no acceleration change will occur.
3. Call random.choices (you will have to import random at the top of the file) with the acceleration\_change list and the weight\_values list. This will return one of the numbers from the acceleration\_change list which will determine how much the acceleration is changed by. For example,

variance = random.choices(acceleration\_change, weights=weight\_values)

1. Update the speed of the Vehicle (using set\_speed) by the amount of the variance.
2. For each subclass the accelerate method then calls the parent (super()) accelerate method AFTER it performs its own changes to the speed.

Options:

* Prompt for racetrack name and length.
* Use turtle graphics to display the racetrack and vehicles. If you do this, make sure that you use functions effectively.

**Assignment Marking Scheme**

The following marking scheme will be used in marking the assignment.

|  | Out of |
| --- | --- |
| Class Diagram (all classes and links between them) | 10 |
| Vehicle Class | 6 |
| Vehicle Methods and Attributes | 8 |
| Motorcycle Class | 4 |
| Truck Class | 7 |
| Tesla Class | 7 |
| RaceTrack Class | 10 |
| race\_run file | 8 |
| Running code | 15 |
| Adding randomness | 5 |
| Display Track each round | 10 |
| Coding Style  - PEP 8 followed  - Good use of functions | 5 |
| Organization  - assignment folder and project named according to standards  - assignment folder zipped and uploaded to Moodle  - all required files included  - classes formatted  - self assessment completed | 5 |
| Total | 100 |