**CISC 487 / 680 SU 2019**

**Applied Artificial Intelligence International Boot Camp**

**Lab Booklet #04**

**Lab Title: Object-Oriented Programming**

### Instructors: Prof. Shalaan PhD, Prof. Grey M.S., Prof. Chu M.S.

### TA's: Aditya Syal, Bao Pham, Zach Pelkey

# Objectives

Objects are a type of structured data type that provides, among other things, an independent state space for each object. While objects do inherently introduce a processing overhead and over-reliance on object-oriented principles can cause their own problems, the paradigm of object-oriented has taken a hold in the marketplace due to the abstraction and encapsulation of behavior, the segregation of state, and the concrete conception of objects it provides.

In order to demonstrate our knowledge and abilities with object-oriented programming, we will be creating a simulation of a four-way intersection with a red light. For this simulation, you will need to create classes for the various vehicles on the road. Each class will track the current state of the vehicle and leave the interconnection of vehicles and roads to the over-seeing simulation application.

After completing this lab, you will be able to:

* Understand basic object-oriented programming principles
* Create Python objects and methods
* Understand how Python encapsulates state
* Understand the importance of information hiding and the best practices surrounding information hiding in Python
* Apply inheritance to the design and implementation of an object hierarchy
* How to construct and test object which you have defined

# Resources

While the lab will explain everything that you need to know, you may want to consult the following sources for additional knowledge:

<https://en.wikipedia.org/wiki/Object-oriented_programming>

<https://www.freecodecamp.org/news/how-to-explain-object-oriented-programming-concepts-to-a-6-year-old-21bb035f7260/>

# Deliverables

For this lab, you will need to submit:

* Python code files (.py), named and coded as instructed. In this lab, you will need to submit:
  + Vehicle.py
  + Bicycle.py
  + MotorVehicle.py

# Instructions

# Design Narrative

In order to create our simulation, we will need you to create different types of vehicles to populate the road. All Vehicles will have some of the same properties: a length, a top speed, a current speed, a location represented by an (x, y) coordinate, and a direction (“NORTH”, “SOUTH”, “EAST”, or “WEST). All vehicles are able to turn left, turn right, accelerate, decelerate, and move. Turning changes a Vehicle’s direction by 90°, accelerating or decelerating changes the Vehicle’s current speed in an unspecified way, and moving changes the Vehicle’s position by the same amount as the Vehicle’s speed in the direction it’s facing.

For example, presume that there’s a Vehicle that is positioned at point (0, 0), is facing NORTH, and is moving 15 units per turn. If it turns left and then moves, the vehicle will be facing WEST at point (-15, 0).

In addition to the general behavior of a Vehicle, there will be two broad sub-groupings of Vehicles: Bicycles and MotorVehicles. Bicycles are 5 feet long, are able to turn at top speed, and can instantly get from 0 to top speed or from top speed to 0.

MotorVehicles can have any length but have additional features: a license plate number, which can be a string of letters and numbers, and an acceleration factor greater than 0 and less than or equal to 1. Given the physics of inertia, changing speed or direction in a MotorVehicle is more complicated. Accelerating or decelerating done by changing the current speed of the MotorVehicle by the MotorVehicle’s top speed × its acceleration factor. Also, whenever a vehicle turns, it decelerates.

For example, if a MotorVehicle has a top speed of 10 and an acceleration factor of 0.2, then the MotorVehicle will accelerate or decelerate by 2 (or 10 × 0.2) every time it accelerated or decelerates, including while turning.

Each class should be in its own code file. These objects are to be called Vehicle, Bicycle, and MotorVehicle. Vehicles and MotorVehicles will have their length, top speed, current location, and direction declared at instantiation. Bicycles will have their top speed, current location, and direction declared.

**All objects will have accessor methods for ALL state variables. MotorVehicles will have a mutator method that allows you to change the vehicle’s license plate number**.

Remember: These requirements are not all encompassing. Use your brain, your knowledge of the system, and the names and descriptions of certain features to make logical leaps about feature behaviors. For instance, what is the minimum speed of a Vehicle?

**Design Artifacts**

Below, we have included some incomplete class diagrams for your reference. Be sure to reference the narrative above to understand how these methods should behave and design your applications using proper object-oriented design principles.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: | Vehicle | | |
| Features: | Up to you as long as you follow the conventions | | |
| Construct: | \_\_init\_\_(length, top speed, location, direction) | | |
| Methods: | * get\_length() | * get\_top\_speed() | * get\_current\_speed() |
| * get\_location() | * get\_direction() | * turn\_left() |
| * turn\_right() | * accelerate() | * decelerate() |
| * move() |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Class: | Bicycle | | |
| Features: | Up to you as long as you follow the conventions | | |
| Construct: | \_\_init\_\_(top speed, location, direction) | | |
| Methods: | * get\_length() | * get\_top\_speed() | * get\_current\_speed() |
| * get\_location() | * get\_direction() | * turn\_left() |
| * turn\_right() | * accelerate() | * decelerate() |
| * move() |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Class: | MotorVehicle | | |
| Features: | Up to you as long as you follow the conventions | | |
| Construct: | \_\_init\_\_( length ,top speed, location, direction, license plate, acceleration factor) | | |
| Methods: | * get\_length() | * get\_top\_speed() | * get\_current\_speed() |
| * get\_location() | * get\_direction() | * get\_acceleration() |
| * get\_license\_plate() | * set\_license\_plate(str) | * turn\_left() |
| * turn\_right() | * accelerate() | * decelerate() |
| * move() |  |  |