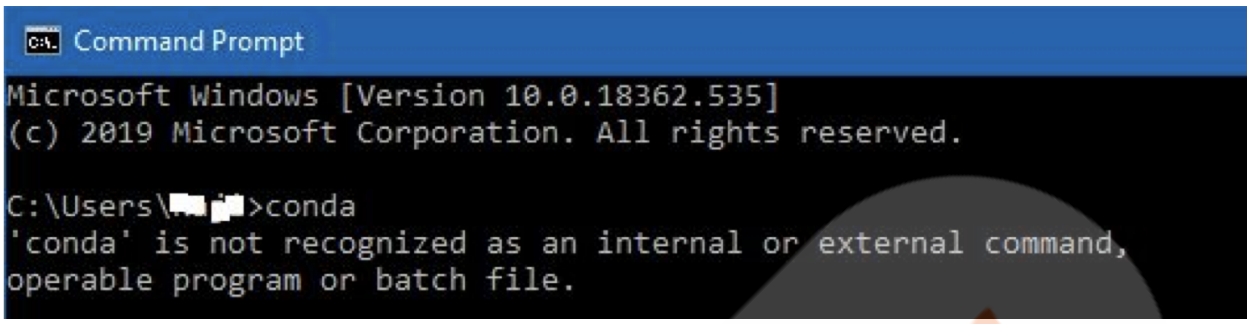
Topic	COLLISION DETECTION	
Class Description	**Taking the most effective data-driven decision is the primary logic behind self-driving cars. Kids learn to inject basic logic into the car to take start and stop decisions to begin with.**	
Class	**ADV-C210**	
Class Time	**60 mins**	
Goals 	• Gather the distance information of the other cars. • Get the names of the other cars.	

Resources Required	• Teacher Resources:
	  ○ Use Gmail login credentials
	  ○ Earphone with mic
	  ○ Notepad and Pen
	• Student Resources:
	  ○ Use Gmail login credentials
	  ○ Earphone with mic (optional)
	  ○ Notepad and Pen	

Class Structure	**Warm Up** **Teacher-Led Activity** **Student-Led Activity** **Project Pointers and Cues** **Wrap Up**	**5 Mins** **15 Mins** **30 Mins** **5 Mins** **5 Mins**
NOTE - Before starting please check whether conda is running properly or its showing error something like this.		



It means you have not set the path of the anaconda.
To resolve this please refer to the error 'Conda not found error' from -Teacher-Reference-Activity-2

NOTE:
Some Instructions are mentioned in Student Activity regarding demoing collision sensor don't forget to perform them when student activity starts.
(Please close all applications for better performance of CARLA simulator)
For WINDOWS:

1. **Open Task manager.**
2. **Stop all the processes that are not required and are taking more Memory.**
3. **Run only 1 CARLA application at a time. Opening multiple CARLA applications will slow down the system.**

The flow for the student to follow when he/she starts the activity is mentioned in the Student-Activity section. Don't forget to follow the same flow.

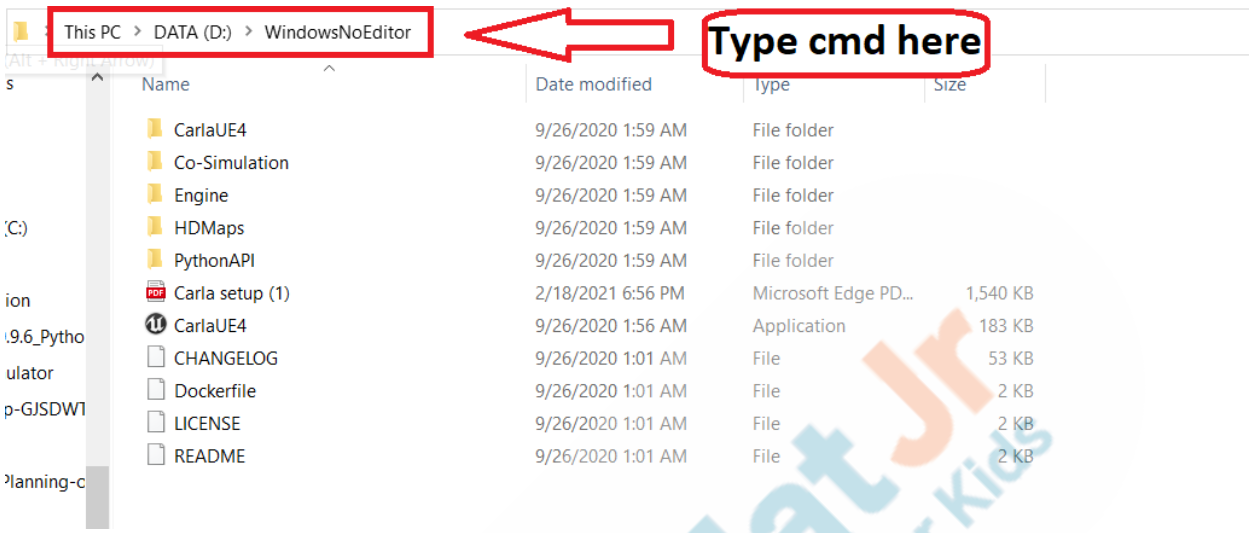
Class Steps	Say	Do
Step 1: Warm up (5 mins)	In the previous class C209, we learned about how to use a lambda function to drive the car. We also learnt how to detect a collision, display the collider name and collision	

	intensity, and take the desired action on collision. Q Which mathematical function did we use to get the intensity of collision? A Using math.sqrt(), we can get the intensity of a collision.	Please click on the  button on the bottom right corner of your screen to start the In-Class Quiz. A quiz will be visible to both you and the student. Encourage the student to answer the quiz questions. The student may choose the wrong option. In that case, help the student to think correctly about the question and then let the student answer again. After the student selects the correct option, the  button will appear on your screen. Click **End quiz** to close the quiz pop-up and continue the class. **Please follow the flow of the class.** 1. **Demo what we will be building today.** 2. **Explain the code.** 3. **Ask the student to**

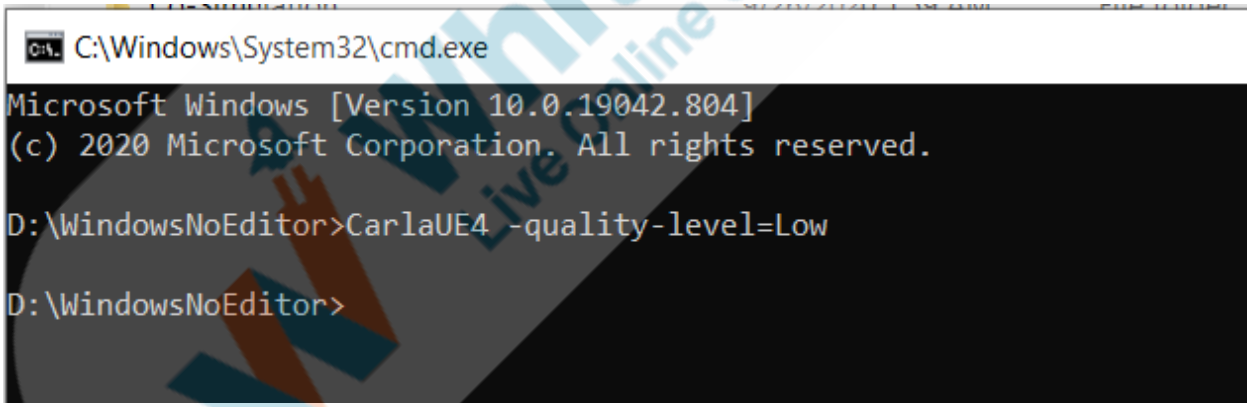
<table>
<tr><td></td><td>Now that you have answered the question and are clear with the topic which we learnt in the last class, we will move to our next topic.</td><td>download prewritten code from Student-Activity-2 and continue coding in it.
4. Explain predefined code.</td></tr>
<tr><td colspan="3">Teacher Initiates Screen Share</td></tr>
<tr><td>Step 2: Teacher-Led Activity (15 mins)</td><td>In the previous class C209, we used a collision sensor and drove the car from source to destination with traffic on the road. When our car collided with an obstacle, we displayed where the collision took place and at what intensity.</td><td></td></tr>
<tr><td colspan="3">In this class, we are going to collect data from the environment and take actions on it. So we can stop the car before it collides by sensing the possibility of a collision.

Let me demo first -

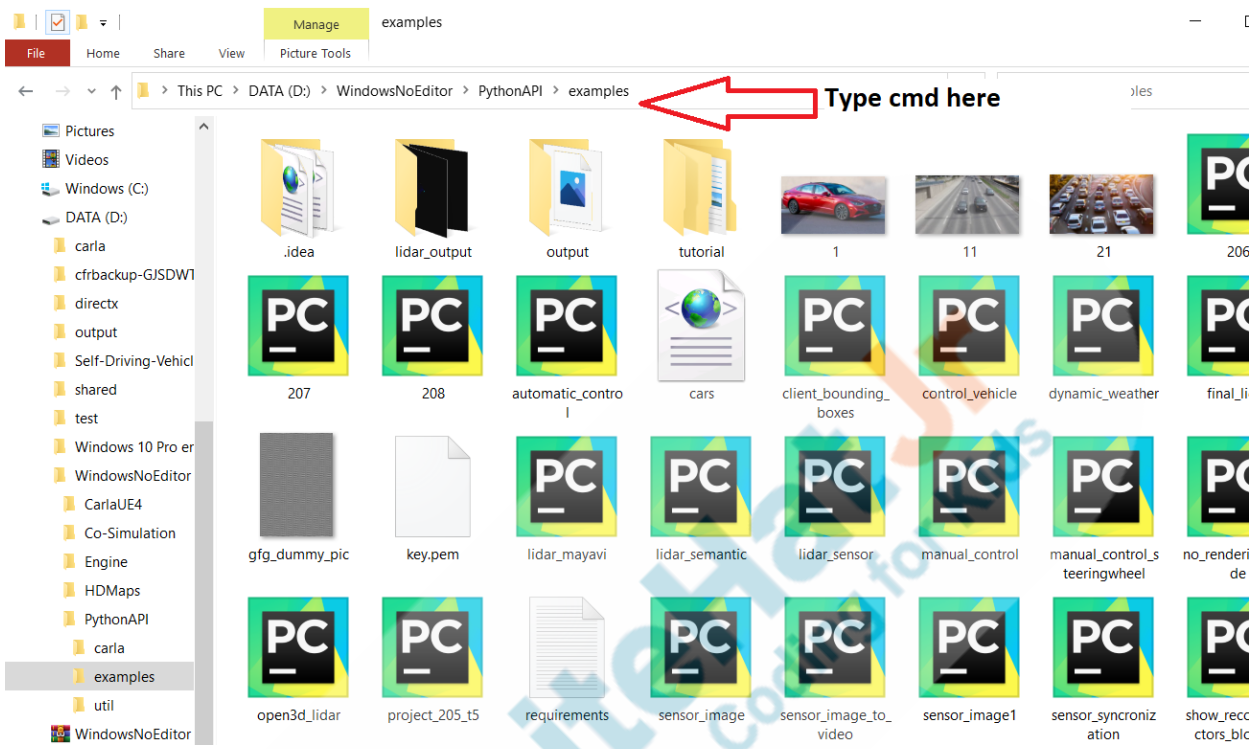
NOTE - Download complete code from Teacher-Activity-2 for demoing and put it in the example folder and run it.
Run CARLA simulator in command prompt like this in the WindowsNoEditor folder:</td></tr>
</table>



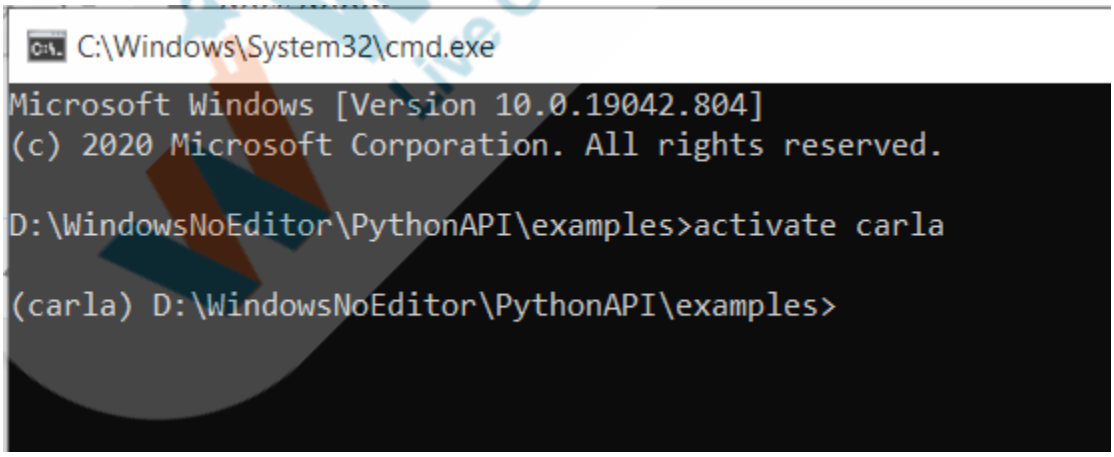
Now, run this command in cmd: **CarlaUE4 -quality-level=Low**
By running the above command, you are running CARLA in low resolution which helps you move in CARLA faster. It won't slow down your system.



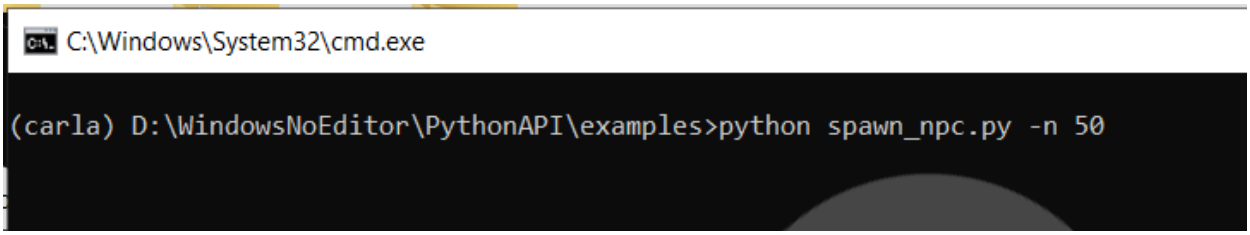
Open the **examples** folder from **WindowsNoEditor > PythonAPI > examples**.
Select the full path from the address bar and type **cmd** like this:



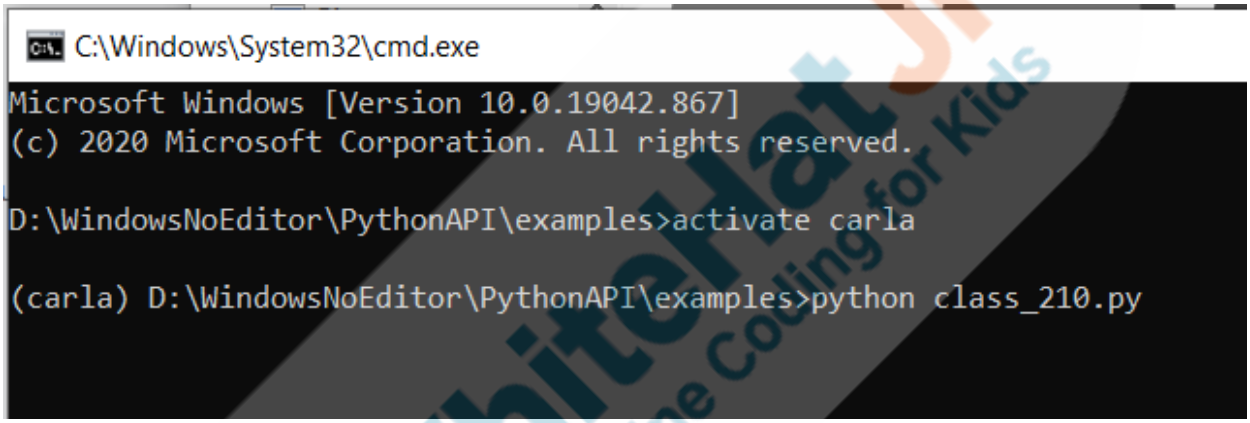
Open **two** command prompts in the **example** folder and activate your CARLA environment.



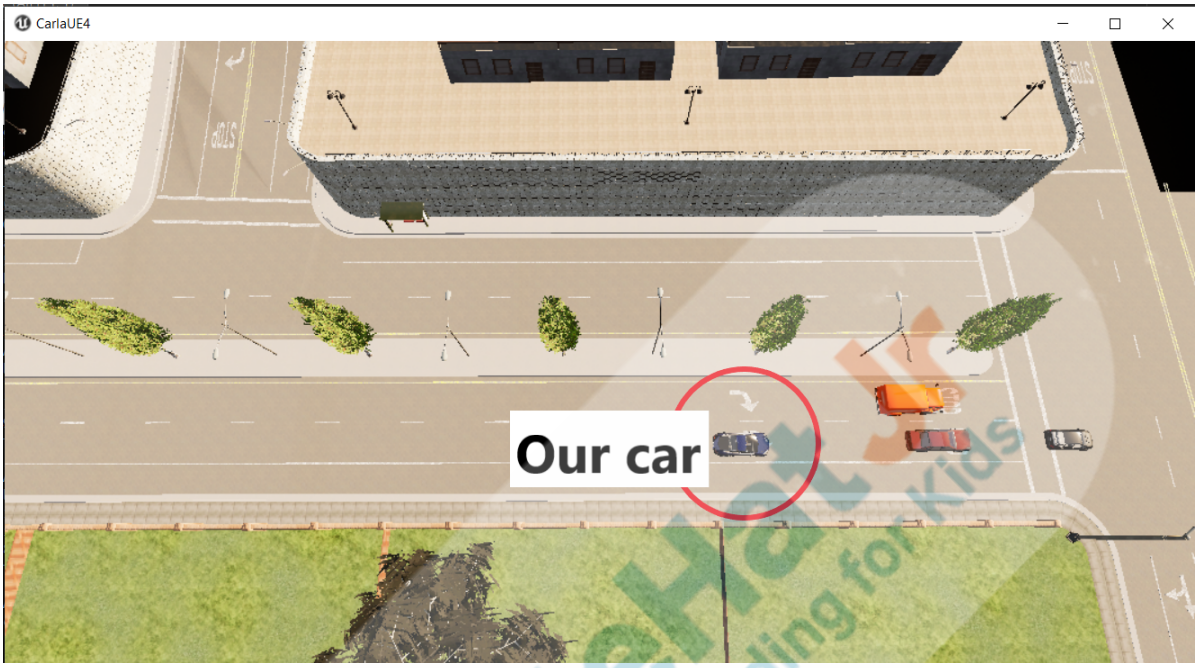
In first command prompt, run this file: **python spawn_npc.py -n 50**



In the second command prompt, run this file: **python class_210.py**



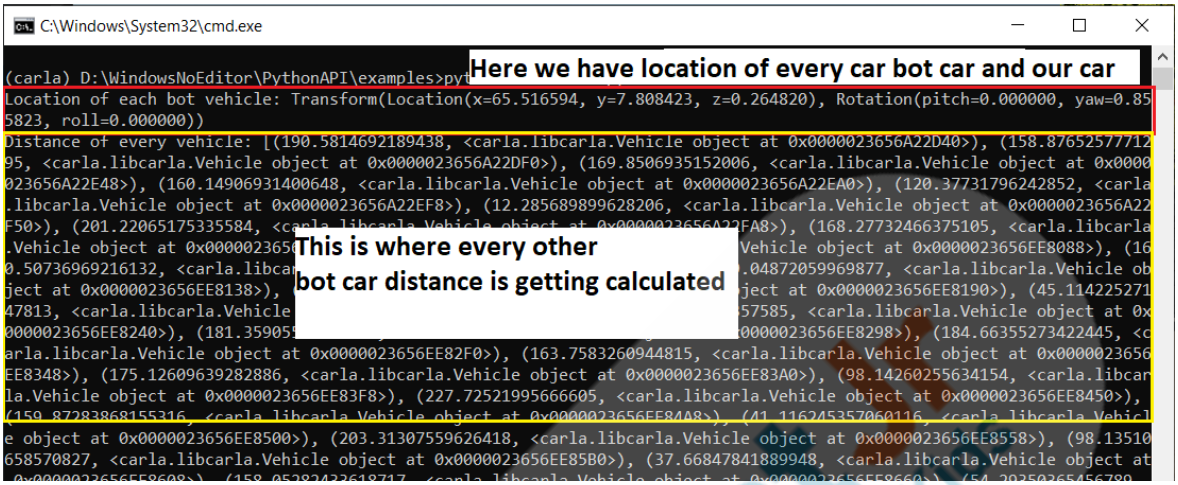
Output:



So the following activities can be observed -

- Cars and walkers are moving around the city because of the **spawn_npc.py** file.
- Then our car follows the path from the spawn location and goes straight.
- The car gathers data from the surrounding to get the location information of each bot car or we can say all the robo cars of the CARLA environment.

On cmd, you can see:



- The **location of each bot vehicle**
- The **distance between the car and every other vehicle**

In the next class, we will also learn how to use this data to take actions like stopping the car,

In today's class, we will gather data from the environment where the bot cars are moving around the city. And we want to get their distance from our car.

Let's have a look at the code:

```python
import glob
import os
import sys
import time
import math
import threading

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla


def get_actor_display_name(actor):
    name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
    return name


actor_list = []
```

```python
def number_of_vehicle():
    all_vehicles = world.get_actors().filter('vehicle.*')
    threading.Timer(0.1, number_of_vehicle).start()
    transform_location = dropped_vehicle.get_transform()
    print("Location of each bot vehicle:",transform_location)

    if len(all_vehicles) > 1:
        distance = lambda data: math.sqrt(
            (data.x - transform_location.location.x) ** 2 + (data.y - transform_location.location.y) ** 2 + (
                    data.z - transform_location.location.z) ** 2)

        get_distance_of_bot_vehicles = []
        for each_car in all_vehicles:
            if each_car.id != world.id:
                get_distance_of_bot_vehicles.append((distance(each_car.get_location()), each_car))

        print("Distance of every vehicle:", get_distance_of_bot_vehicles)


def car_control():
    dropped_vehicle.apply_control(carla.VehicleControl(throttle=0.5))
    time.sleep(10)

```

```python
try:
    client = carla.Client('127.0.0.1', 2000)
    client.set_timeout(10.0)
    world = client.get_world()

    get_blueprint_of_world = world.get_blueprint_library()
    car_model = get_blueprint_of_world.filter('model3')[0]
    spawn_point = (world.get_map().get_spawn_points()[1])
    dropped_vehicle = world.spawn_actor(car_model, spawn_point)
    simulator_camera_location_rotation = carla.Transform(spawn_point.location, spawn_point.rotation)
    simulator_camera_location_rotation.location += spawn_point.get_forward_vector() * 30
    simulator_camera_location_rotation.rotation.yaw += 180
    simulator_camera_view = world.get_spectator()
    simulator_camera_view.set_transform(simulator_camera_location_rotation)
    actor_list.append(dropped_vehicle)

    collision_sensor = get_blueprint_of_world.find('sensor.other.collision')
    sensor_collision_spawn_point = carla.Transform(carla.Location(x=2.5, z=0.7))
    sensor = world.spawn_actor(collision_sensor, sensor_collision_spawn_point, attach_to=dropped_vehicle)

    sensor.listen(lambda data: _on_collision(data))

    actor_list.append(sensor)
```

```python
    def _on_collision(data):
        print("Collision is there")
        actor_type = get_actor_display_name(data.other_actor)
        print("Collision with", actor_type)
        Collision_event_record = data.normal_impulse
        intensity_of_collision = math.sqrt(
            Collision_event_record.x ** 2 + Collision_event_record.y ** 2 + Collision_event_record.z ** 2)
        print("Intensity of collision", intensity_of_collision)
        dropped_vehicle.apply_control(carla.VehicleControl(hand_brake=True))
        time.sleep(5)


    number_of_vehicle()
    car_control()
    time.sleep(1000)
finally:
    print('destroying actors')
    for actor in actor_list:
        actor.destroy()
    print('done.')
```

If you take a closer look at the code, you would see that a major portion of the code is already done by us many times in our previous classes. We have made small modifications to it. I will quickly take you through the code and then explain the new code which we will be writing today.
Let's understand the code:

1. First, we have imported our libraries and set the path for importing CARLA files.

```
import glob
import os
import sys
import time
import math
import threading

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla
```

2. Then we have defined the **get_actor_display_name()** function which returns the name of the collider.

```
Get the name of the collider
def get_actor_display_name(actor):
    name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
    return name
```

Q: What is the use of the split () function?

A: Using a **split(',')** function, we can split sentences in separate words. For example, **" Hi i'm whitehat jr student "** will be **["Hi", "i'm", "whitehat", "jr", "student"].**

3. Then we have the **car_control()** function which will drive the car straight.

```
def car_control():
    dropped_vehicle.apply_control(carla.VehicleControl(throttle=0.5))
    time.sleep(10)
```

4. Then we have a block of code for initialising the car in the **try** block where we can call and define all the libraries we are going to need for spawning and selecting a car.

```
50  try:
51      client = carla.Client('127.0.0.1', 2000)
52      client.set_timeout(10.0)
53      world = client.get_world()
54
55      get_blueprint_of_world = world.get_blueprint_library()
56      car_model = get_blueprint_of_world.filter('model3')[0]
57      spawn_point = (world.get_map().get_spawn_points()[1])
58      dropped_vehicle = world.spawn_actor(car_model, spawn_point)
59      simulator_camera_location_rotation = carla.Transform(spawn_point.location, spawn_point.rotation)
60      simulator_camera_location_rotation.location += spawn_point.get_forward_vector() * 30
61      simulator_camera_location_rotation.rotation.yaw += 180
62      simulator_camera_view = world.get_spectator()
63      simulator_camera_view.set_transform(simulator_camera_location_rotation)
64      actor_list.append(dropped_vehicle)
```

5. Then in line 66-68, we have defined a collision sensor. In case of a collision, this sensor will take action.

```
65
66      collision_sensor = get_blueprint_of_world.find('sensor.other.collision')
67      sensor_collision_spawn_point = carla.Transform(carla.Location(x=2.5, z=0.7))
68      sensor = world.spawn_actor(collision_sensor, sensor_collision_spawn_point, attach_to=dropped_vehicle)
69
70      sensor.listen(lambda data: _on_collision(data))
71
72      actor_list.append(sensor)
73
74
```

Define collision sensor and spawn the sensor to car

6. Then we have called the **sensor.listen()** method to pass the environment data to the **_on_collision()** function, get the collision data, and append this sensor to **actor_list**.

```
65
66      collision_sensor = get_blueprint_of_world.find('sensor.other.collision')
67      sensor_collision_spawn_point = carla.Transform(carla.Location(x=2.5, z=0.7))
68      sensor = world.spawn_actor(collision_sensor, sensor_collision_spawn_point, attach_to=dropped_vehicle)
69
70      sensor.listen(lambda data: _on_collision(data))
71
72      actor_list.append(sensor)
73
74
```

Receive data from sensor. And perform _on_collision function

sensor is the variable that holds the collision sensor and collision point of the vehicle.

In the previous class, we had the **_on_collision()** function for displaying collision.

```
74
75  def _on_collision(data):
76      print("Collision is there")
77      actor_type = get_actor_display_name(data.other_actor)
78      print("Collision with", actor_type)
79      Collision_event_record = data.normal_impulse
80      intensity_of_collision = math.sqrt(
81          Collision_event_record.x ** 2 + Collision_event_record.y ** 2 + Collision_event_record.z ** 2)
82      print("Intensity of collision", intensity_of_collision)
83      dropped_vehicle.apply_control(carla.VehicleControl(hand_brake=True))
84      time.sleep(5)
85
```

Perform the actions such as
1.get the name of collider
2. Show the intensity of collision
3. And apply brake.

Q: What is the role of the **_on_collision()** function?
A: _on_collision(data)- is the function which takes action on the collision detection.

Q: Explain this line of code:
math.sqrt(Collision_event_record.x ** 2 + Collision_event_record.y ** 2 + Collision_event_record.z ** 2)
A: We have a variable called **collision_event_record** which contains X , Y, and Z axis values. We are multiplying each axis value with 2 and adding those values together for calculating the intensity. Then we are applying **math.sqrt** on the result value to get the accurate intensity of collision.

Now let's talk about our objective for this class and also the upcoming classes. The objective is to stop the car before a collision occurs. This sounds quite advanced and interesting, right? We are actually going to make our car smarter than before.

And for this, in this class, we are going to gather data from the surrounding and sense a collision before it happens. This is going to help us make our car intelligent.

To achieve this, we are going to work on a very powerful technique called **Data gathering**.

Data gathering:
Data gathering basically means collecting, measuring, and analyzing accurate data for getting the useful insights of a research.

Let's take one simple example: If your best friend's birthday is around the corner, you will look for a gift to give your friend. For that, you need to find out what he/she likes. You can browse the internet or go to the market looking for the best and interesting gift for your friend.

After your research is done, you will make a choice and get the best item for your friend. Because, after all, he/she is your best friend and you want to give the best thing to your best friend, right?
So, it means that you need to gather some information from surrounding and decide to take action, which is taking the best gift for your best friend.

Similarly in CARLA, we can gather data from the virtual world. As you can see, we have multiple entities in the CARLA world such as cars, walkers, street poles, walls, buildings etc.

Street poles, walls, and buildings are not moveable objects, right? So we cannot receive any real-time data from them as they remain at the same place.
But running cars and walkers are moveable entities here, right? So it can deliver real-time data such as location. In other words, when a car moves from one location to another, the location of the car changes. This means that it travels from point A to point B.

So our logic for data gathering will be:

- Get the information about all the cars in the world. Information such as Name, ID, and location of each car.
- Get the location of our car.

Let's understand the code which you will be writing today -

```
26 def number_of_vehicle():
27     all_vehicles = world.get_actors().filter('vehicle.*')
28     threading.Timer(0.1, number_of_vehicle).start()
29     transform_location = dropped_vehicle.get_transform()
30     print("Location of each bot vehicle:",transform_location)
31
```

Here, you can see that we have created one function called **number_of_vehicle()** in which we are performing the below task:

1. First, get the actors from `world.get_actors().` and apply the **filter** method to select all vehicles `.filter('vehicle.*')` and store it in the `all_vehicles` variable.

2. **Get the data continuously from all the moving cars:** Since the cars keep moving and we want to get the data continuously, we will be using one library called **threading**.

   Threading()
   Threading is a process where we perform certain operations continuously after a set interval of time, and each of the operations performed can be called as a thread.
   There are 3 things we can do:
   - Stop a thread.
   - Start a thread.
   - Sleep a thread.

   Whatever we want to continuously perform certain operations, we can do it using the **threading()** library.
   For example: Remember in **p5.js** in class C113**,** we had used the **draw()** function which was called continuously. Similarly, you can consider this as an example of threading to perform an operation or process continuously.

   Example:
   import threading #import threading library

   def newFunction(): #Define function

print("Hello, Whitehat coders!",) #Print a line
threading.Timer(0.1, newFunction).start()

- **threading** is the library which we have imported.
- **Timer()** is a function that receives two parameters. One is Time in **millisecond** and the other one is the name of the function **newFunction()** which you want to run in **threads**.
- Thread means calling the function continuously which is mentioned with an interval of 0.1 millisecond.
- **start()** function starts the threading.

So in our case,

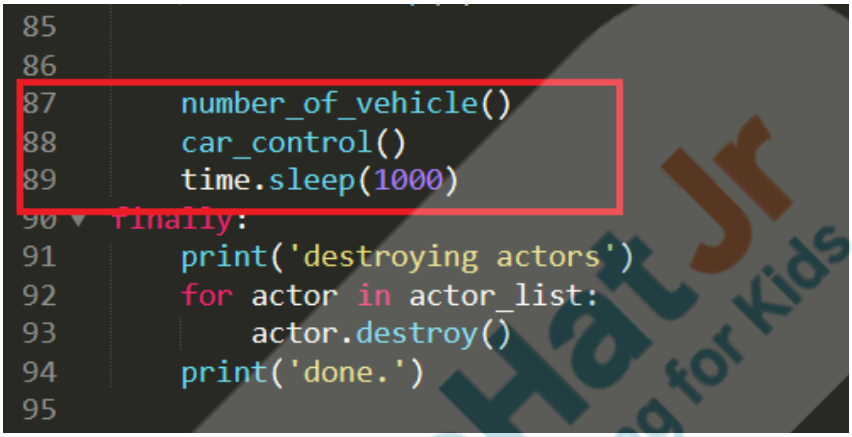
- `threading.` calls a threading library.
- `Timer(0.1, number_of_vehicle).` receives 0.1 millisecond time and runs the **number_of_vehicle** function.
- `.start()` This function starts the threading process by running the **number_of_vehicle** function multiple times with an interval of 0.1 millisecond, which will result in getting the car's location after every 0.1 millisecond.

3. Now, we need the location of our car. We will get the location of our car by - `dropped_vehicle.get_transform()` and store this location in the variable `transform_location`. Here **dropped_vehicle** holds information about the spawn car. So using the **get_transform()** function, we can get the location of our spawn_car.

4. Now print the **transform_location** variable where we can see the updated location of all the cars in the environment.

```
print("Location of each bot vehicle:",transform_location)
```

5. At the end of the code, we are calling `number_of_vehicle()` and `car_control()` functions before the **finally** code block.



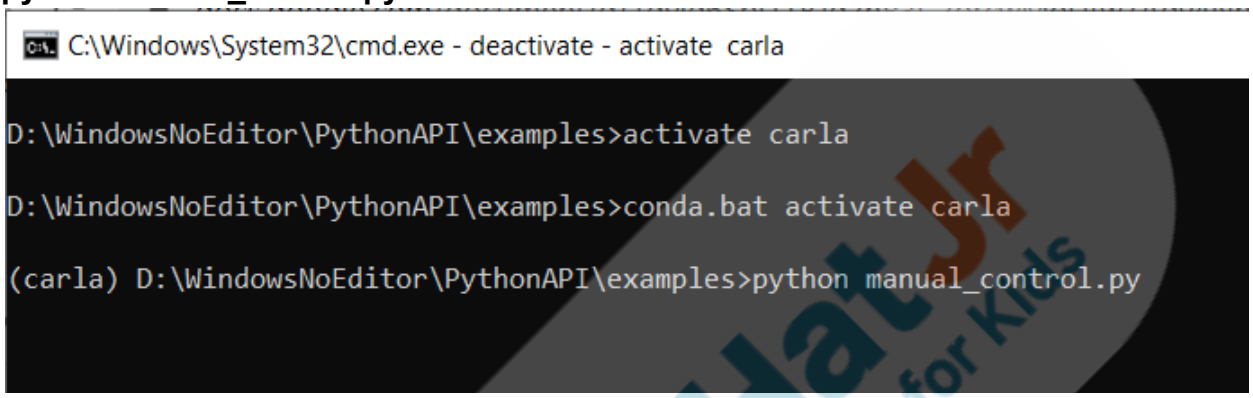
```
85
86
87          number_of_vehicle()
88          car_control()
89          time.sleep(1000)
90      finally:
91          print('destroying actors')
92          for actor in actor_list:
93              actor.destroy()
94          print('done.')
95
```

	Teacher Stops Screen Share	
	Now it is your turn.	
	• **Ask the Student to press the ESC key to come back to the panel.** • **Guide the Student to start Screen Share.** • **Teacher gets into Fullscreen.**	
Step 3: Student-Led Activity (30 mins)	Lets first run our car in a carla environment and demo collision sensor.	**Student-Activity-1- CODE DIAGRAM** **Student-Activity-2- PREWRITTEN CODE**

Open the cmd and activate carla environment, enter below command

NOTE - Don't put more then 5 minutes in this demo activity

python manual_control.py



After opening the pygame windows.

You can drive car using W, S, A, D keys
If you collide somewhere you can see a graph in orange over in collision

Then you will be able to see gathered data in number of vehicle column

Download the pre-written code from Student-Activity-2.

Move the downloaded file in WindowsNoEditor > PythonAPI > examples folder.
Then launch **Sublime** and open this downloaded file, and start coding in it.

You need to do the following tasks -

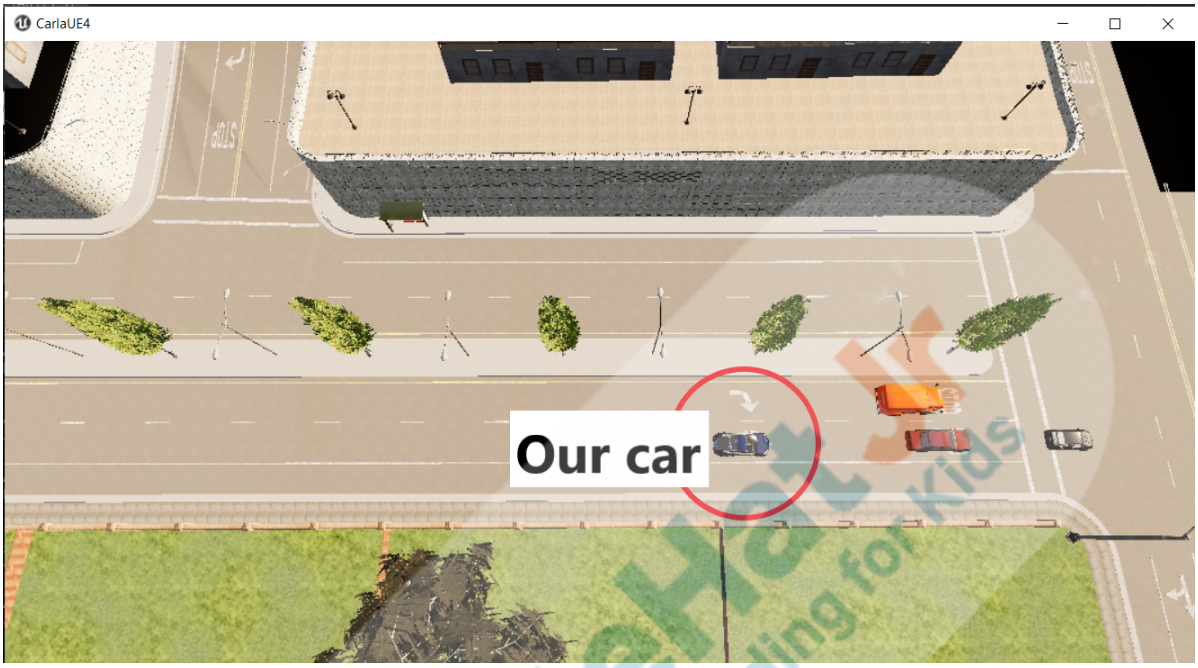
Define the **number_of_vehicle()** function and write code for -

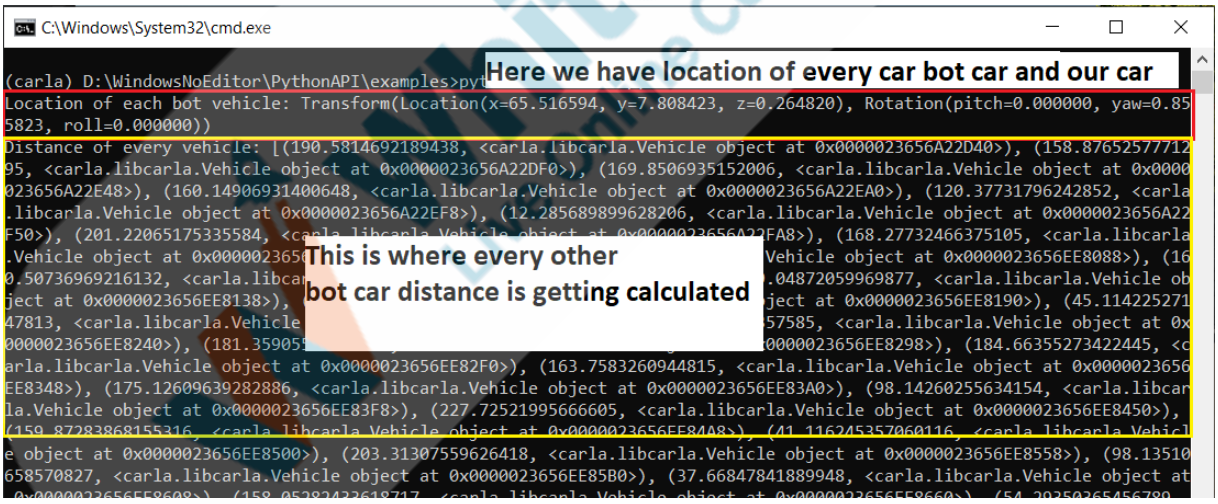
1. the **get_actor_()** function to get the car actor name.
2. Define threading and set 0.1 millisecond timer in the **number_of_vehicle()** function
3. Get the location of all cars.
4. Print the location of all the cars.

NOTE -
Once student is done with the code
- Run CARLA in **low quality**.
- Run **python spawn_npc.py -n 50** in one command prompt.
- Then in another command prompt, run the **student-activity210.py** file.

The following screenshot is a reference code that the student needs to add.

```python
import glob
import os
import sys
import time
import math
import threading

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla


def get_actor_display_name(actor):
    name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
    return name


actor_list = []

def number_of_vehicle():
    all_vehicles = world.get_actors().filter('vehicle.*')
    threading.Timer(0.1, number_of_vehicle).start()
    transform_location = dropped_vehicle.get_transform()
    print("Location of each bot vehicle:",transform_location)

```





Once the program is done, please close the CARLA simulator to avoid slowing down the system.

Teacher Initiates Screen Share

Now we have the number of bot cars running in the city and also the updated location of our car.

Using this information, now we can get the distance of each bot car from our car. In other words, we are going to get the distance of each car and check it with our car location.

NOTE - Explain the following code to the student. The student doesn't have to code this as it is pre-written and given to the student.

For this, we are going to follow the below steps:

- Check the condition if the number of both cars is more than 1.
- Subtract the location of our car with the location of every other bot car.
- Apply square root on the result.
- Display the distance of each car from our car.

```
32      if len(all_vehicles) > 1:
33          distance = lambda data: math.sqrt(
34              (data.x - transform_location.location.x) ** 2 + (data.y - transform_location.location.y) ** 2 + (
35                      data.z - transform_location.location.z) ** 2)
36
37          get_distance_of_bot_vehicles = []
38          for each_car in all_vehicles:
39              if each_car.id != world.id:
40                  get_distance_of_bot_vehicles.append((distance(each_car.get_location()), each_car))
41
42          print("Distance of every vehicle:", get_distance_of_bot_vehicles)
```

Here,

1. First, we are checking the condition with **all_vehicle** `(all_vehicles` where we have data of all the bot cars. And check the length of **bot_vehicle**.

   `if len(all_vehicles)` must be greater than > 1

   `if len(all_vehicles) > 1:`

   - Now inside this 'if-condition', we have a function with lambda `lambda data:` and perform math square root operation `lambda data: math.sqrt(` on two entities. Which are the locations of bot cars and our car.

This lambda function will be used to calculate the distance between bot cars and our car.

Let's understand this with an example -
We have two cars. One is a BMW and the other one is an Audi. We have the location of BMW in X, Y, and Z format like this,

```
Vector3D(x=0.000000, y=77.120049, z=0.000000)
```

Just like the BMW, we also have similar location data of the Audi. But values are different since both cars are at different locations.
As a result, we can see something like this:
BMW = (X=0.0000, Y=77.120049, Z=0.0000)
Audi = (X=21.1400, Y=47.120049, Z=0.0000)

Now we want to get the distance between these two cars. We can get it by subtracting the values of X,Y, and Z axis of BMW from X,Y, and Z axis of AUDI respectively and add these results together like this:
(X=0.0000 - X=21.1400) + (Y=77.120049 - Y=47.120049) + (Z=0.0000 - Z=0.0000)
And we can get some value, right?

But this value won't be accurate. So we will get the approximate value. For this, we will multiply each axis with 2.
((X=0.0000 - X=21.1400) ** 2 + (Y=77.120049 - Y=47.120049) **2 + (Z=0.0000 - Z=0.0000) ** 2)

And then apply the **math.sqrt** function. So that we can have an exact number in **meters**.

Now lets see this in code -

- `(data.x - transform_location.location.x) ** 2` subtract the location of our car from the location of every other bot cars given by x axis.

`(data.y - transform_location.location.y) ** 2` subtract the location of our car from the location of every bot cars given by y axis.

- `data.z - transform_location.location.z) ** 2` subtract the location of our car from the location of every bot cars given by z axis..

- Now add this result of all the three axes:

```
(data.x - transform_location.location.x) ** 2 + (data.y - transform_location.location.y) ** 2 + (
        data.z - transform_location.location.z) ** 2)
```

- And apply the **math.sqrt** function for square root and store in the **distance** variable.

```
distance = lambda data: math.sqrt(
    (data.x - transform_location.location.x) ** 2 + (data.y - transform_location.location.y) ** 2 + (
            data.z - transform_location.location.z) ** 2)
```

2. Now let's loop the **all_vehicles** variable and use the lambda function which we created above, to get the distance between our car and the bot cars.

- Create a list variable `get_distance_of_bot_vehicles = []` where we will store the distance of location of all the bot vehicles.
- Write a **for** loop `for each_car in all_vehicles:` where the **all_vehicles** variable contains a number of vehicles and we are fetching the information of each vehicle one by one and storing it in each_car variable.
- Now we have data of each vehicle in **each_car** variable which contains the ID of the vehicle and the name of the vehicle.
- Our **car's id** is stored in the **world.id** variable `world.id:` and all the bot or all vehicle's ids are stored in the **each_car variable, and we are getting those id's by accessing the id value using the each_car variable like this** `each_car.id`.
- So we will check **if** all the vehicle ids are not equal to **world.id**.
  `if each_car.id != world.id:` performs the following actions which return the distance of the locations of all cars.

- Then inside 'if-condition' we, will append the distance of location of each vehicle and store it in **get_distance_of_bot_vehicles** list which we had created. For this we will use the distance() function, which is the lambda function `distance()`
  This **distance()** will calculate the distance between our car and the bot cars location. For this, we need the location of the bot cars and the location of our car which are present in the all_vehicles variable.
  All car's information is stored in the **each_car** variable, and using **get_location()**, we get the location of each car, hence we will write:
  `(distance(each_car.get_location()),`
- Now, this each car variable consists of data of all the cars, that is the bot car and our car as well, so assuming the information of our car is present in each_car variable itself, so we will again pass the each_car variable, like this in the distance() function. `distance(each_car.get_location()), each_car)`
- For each bot vehicle distance we are going to receive, we will be appending it to the in the list variable which we have created.
  `.append((distance(each_car.get_location()), each_car))`

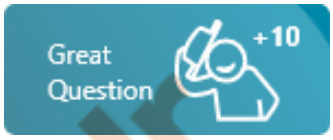
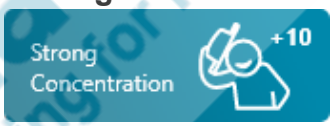
```
get_distance_of_bot_vehicles.append((distance(each_car.get_location()), each_
```

Now, we have stored all the distance values in the **get_disatnce_of_bot_vehicle** variable.
To see the result, in line 42, we have provided a print statement to print the result.

```
42          # print("Distance of every vehicle:", get_distance_of_bot_vehicles)
```

Please uncomment this statement by removing '**#**' from the starting of the print statement and observe the output.

Teacher Initiates Screen Share		
Step 4: Project Pointers and Cues (5 mins)	**Goals of the Project:** Today, we learned how to gather data from bot cars such as location information. **Story:** Many car companies have been working on self-driving vehicle technology for understanding how self-driving cars can be stopped before a collision takes place. They got to know that you are learning the same thing. They want you to perform the below task which may help them complete their research. **GATHERING AUDI DATA:** In this project, we want you to perform a set of tasks to improve the function of gathering data. Complete the code, get the information of vehicle Audi, and keep increasing the thread time by 0.4 milliseconds. Good Luck!	**Run the Project Solution from the activity section.**
Teacher Guides Student to Stop Screen Share		
Step 4: Wrap-Up (5 mins)	You did great today as well. Great! You have two hats off.	(Give at least 2 hats off) Press the **Hats Off** Icon for **Creatively Solving Activities**.

	Q What is the use of threading function? A Using **threading()**, we can continuously call a function after a set interval time. Q What is the use of append() in python? A The **append()** method in python adds an item to an existing list.	 Press the **Hats Off** Icon for **Great Question**.  Press the **Hats Off** Icon for "**Strong Concentration**".  **If you don't have time to perform additional activities, ask the student to perform all the additional activities after the class. Additional activities are VERY important for kids, so that they are ready for the next module. And some challenging concepts are coming ahead.**

		Also remind the student to refer to the Student Reference activity for increasing his/her knowledge. This should also be done.

For the solution of all the Additional Activity, open Teacher-Activity-3 and navigate to class number C210.

Additional Activity 1 -
Run Student-Activity-3 from the panel.
The TASK and HINTS are mentioned on the website itself.

Additional Activity 2 -
Run Student-Activity-4 from the panel.
The TASK and HINTS are mentioned on the website itself.

Additional Activity 3 -
Run Student-Activity-5 from the panel.
The TASK and HINTS are mentioned on the website itself.

Additional Activity 4 -
Run Student-Activity-6 from the panel.
The TASK and HINTS are mentioned on the website itself.

Additional Activity 5 -
Run Student-Activity-7 from the panel.
The TASK and HINTS are mentioned on the website itself.

Teacher Clicks ✖ End Class

Activity	Activity Name	Links

Teacher Activity 1	CODE DIAGRAM	https://docs.google.com/document/d/e/2PACX-1vRz2wlBG_UYmgnUdg-7Lie4eItiRZWGnK6B4_v1UyqFvKsCSzBCcVxHa5z2pyToWjZ5rDqJleLcfpgA/pub
Teacher Activity 2	SOURCE CODE	https://drive.google.com/file/d/1pjfpPNLuz7M6fvBpzloCwhphvJLS4XzN/view?usp=sharing On top right of the screen there is a download button use it  for downloading this .py file.
Teacher Reference Activity 1	RECORD SCREEN REFERENCE	https://curriculum.whitehatjr.com/ADV+Asset/C145-+Guide+to+loom+recording.pdf
Teacher Reference Activity 2	Carla Troubleshoot	https://docs.google.com/document/d/e/2PACX-1vT5ftSaMcd7tJsOGgHdYboHqU26eC6ppmClVd0o5oaIkwEpmjs6xlA7Dc7AYJwE_gq0pVdnLy4XXbLr/pub Note: Follow this document for carla error fixing
Student Activity 1	CODE DIAGRAM	https://docs.google.com/document/d/e/2PACX-1vRz2wlBG_UYmgnUdg-7Lie4eItiRZWGnK6B4_v1UyqFvKsCSzBCcVxHa5z2pyToWjZ5rDqJleLcfpgA/pub
Student Activity 2	PREWRITTEN CODE	https://drive.google.com/file/d/1b0_fl1qk7XptvItbfhfgTgYbX7NWWZUc/view?usp=sharing On top right of the screen there is a download button use it  for downloading this .py file.

Student Reference Activity 1	Carla vehicle control	https://carla.readthedocs.io/en/latest/python_api/
Student Reference Activity 2	RECORD SCREEN REFERENCE	https://curriculum.whitehatjr.com/ADV+Asset/C145-+Guide+to+loom+recording.pdf
Project Solution	GATHERING AUDI DATA	https://drive.google.com/file/d/10D537BpezqDxmvyQ11LMum6vbIu_w7v3/view?usp=sharing