Python Code Challenges: Classes

Python Code Challenges Involving Classes

This article will help you review Python classes by providing some interesting code challenges.

Some of these challenges are difficult! Take some time to think about them before starting to code.

You might not get the solution correct on your first try — look at your output, try to find where you're going wrong, and iterate on your solution.

Finally, if you get stuck, use our solution code! If you "Check Answer" twice with an incorrect solution, you should see an option to get our solution code. However, truly investigate that solution — experiment and play with the solution code until you have a good grasp of how it is working. Good luck!

Class Syntax

As a refresher, class syntax looks like this:

```
class MyClass:
    # ... class variables ...

    def __init__(self):
        # ... instance variables ...
```

For example, a class which defines a rectangle using a class variable, instance variables, and a method looks like this:

```
class Rectangle:
    sides = 4

    def __init__(self, width=0, height=0):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height


rectangle_1 = Rectangle(5, 10)
rect_area = rectangle_1.calculate_area()
```

The last two lines in the above example show how to initialize an object of the class as well as calling one of the methods.

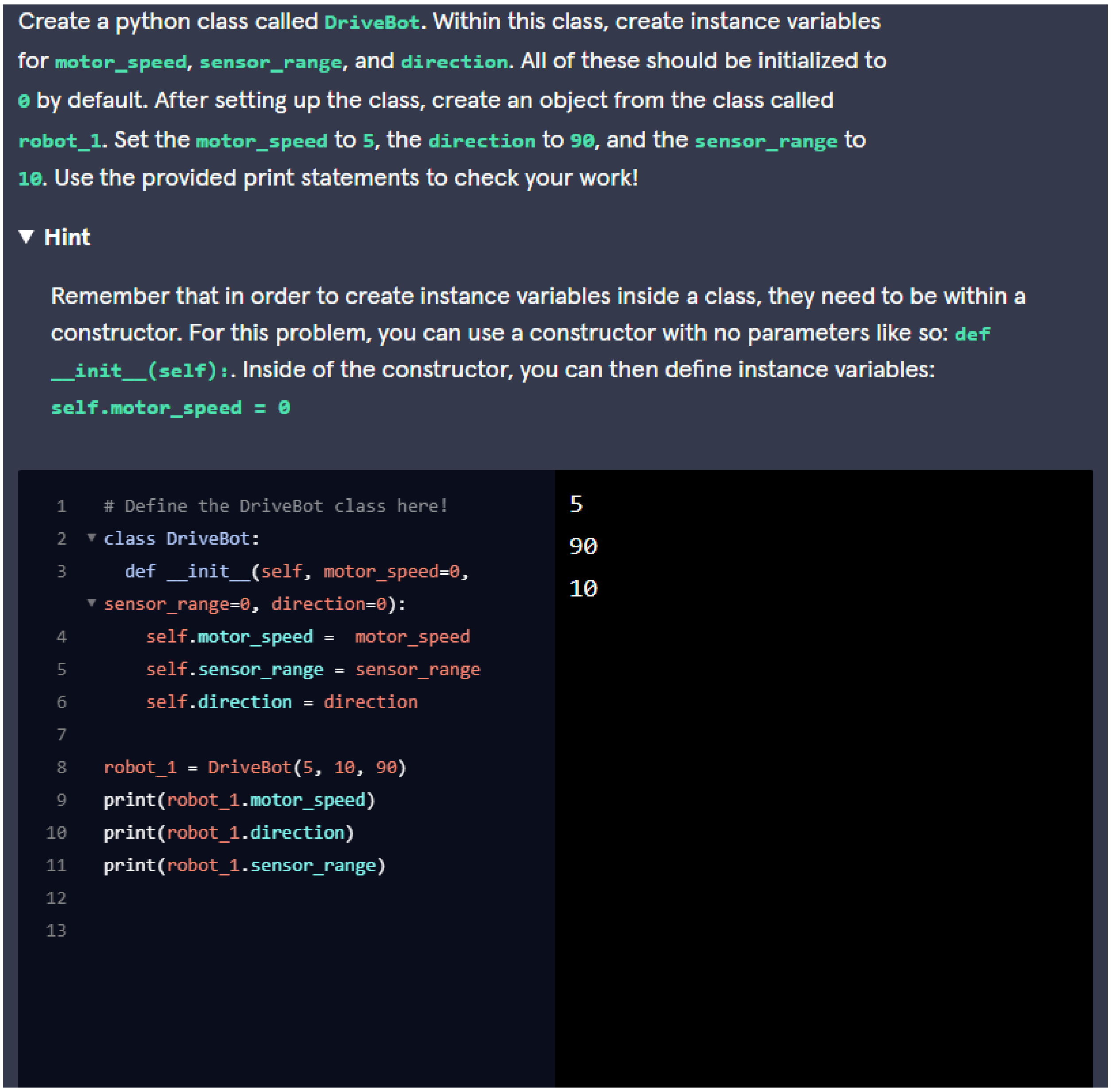
Challenges

You have decided to use your programming knowledge to create a new robotics company. Your idea for micro driving robots which are able to pick up and deliver objects was promising and now you want to start programming the logic. These code challenges will use your knowledge of Classes to solve some example scenarios. Try solving the five challenge problems below!

1. Setting Up Our Robot

Let’s begin by creating the class for our robot. We will begin by setting up the instance variables to keep track of important data related to the robot. Here are the steps we need to do:

1. Create a new class called `DriveBot`
2. Set up a basic constructor (no parameters needed)
3. Initialize three instance variables within our constructor which all default to 0: `motor_speed`, `direction`, and `sensor_range`



Create a python class called `DriveBot`. Within this class, create instance variables for `motor_speed`, `sensor_range`, and `direction`. All of these should be initialized to `0` by default. After setting up the class, create an object from the class called `robot_1`. Set the `motor_speed` to `5`, the `direction` to `90`, and the `sensor_range` to `10`. Use the provided print statements to check your work!

▼ **Hint**

Remember that in order to create instance variables inside a class, they need to be within a constructor. For this problem, you can use a constructor with no parameters like so: `def __init__(self):`. Inside of the constructor, you can then define instance variables: `self.motor_speed = 0`

```python
# Define the DriveBot class here!
class DriveBot:
  def __init__(self, motor_speed=0, sensor_range=0, direction=0):
    self.motor_speed =  motor_speed
    self.sensor_range = sensor_range
    self.direction = direction

robot_1 = DriveBot(5, 10, 90)
print(robot_1.motor_speed)
print(robot_1.direction)
print(robot_1.sensor_range)
```

```
5
90
10
```

Here is how we solved this:

```python
class DriveBot:
    def __init__(self):
        self.motor_speed = 0
        self.direction = 0
        self.sensor_range = 0
```

This shows the structure of a simple class which only contains instance variables. The three instance variables are set to 0 for now, which means that they can only be changed by manually by accessing them from an object of the `DriveBot` class.

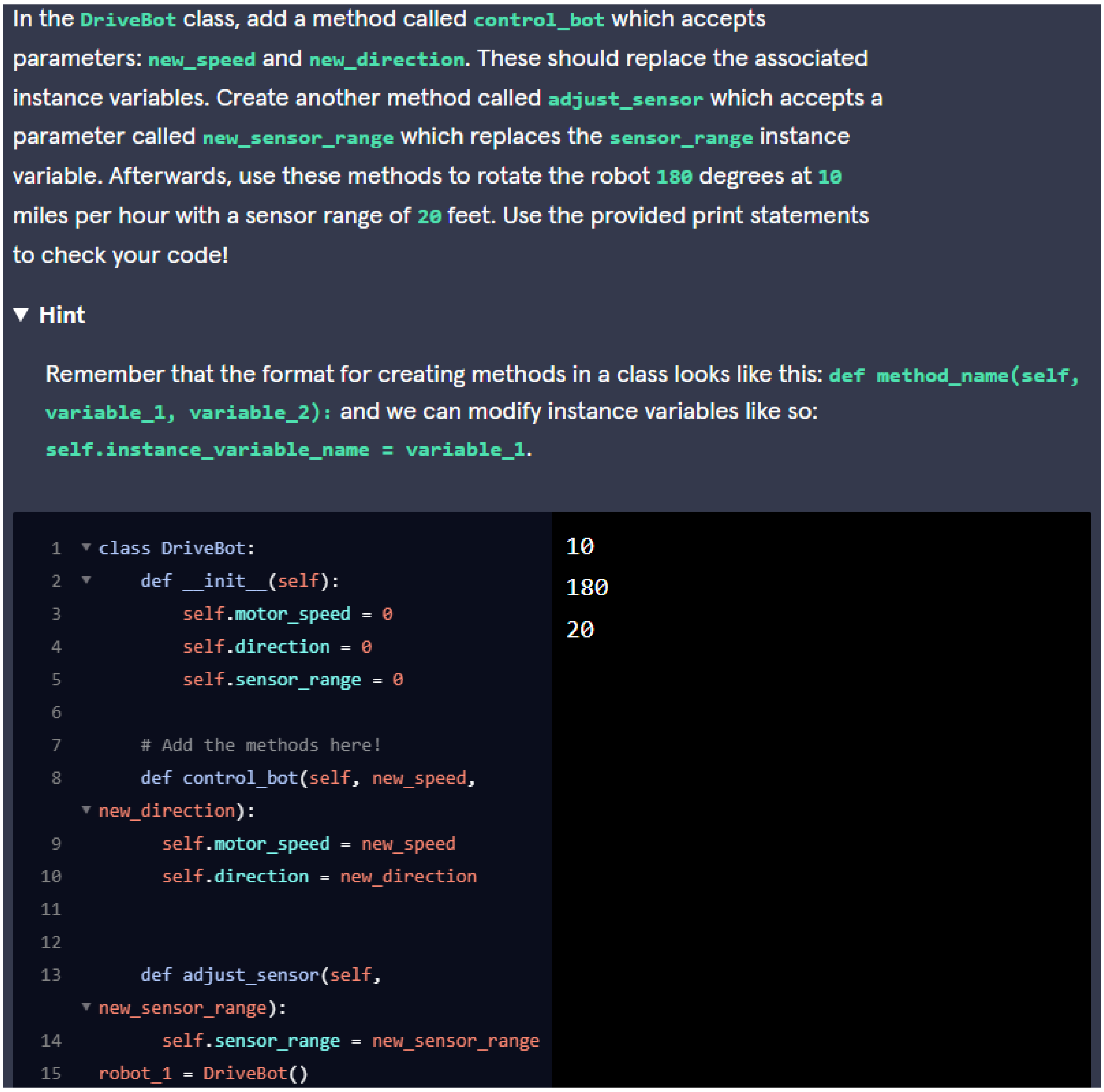
Here is an example of how you can manually set the values for an object of the `DriveBot` class:

```python
test_bot = DriveBot()
test_bot.motor_speed = 30
test_bot.direction = 90
test_bot.sensor_range = 25
```

2. Adding Robot Logic

Now we want to add some logic to our robot. It would be very useful to be able to control our robot, so we are going to create a `control_bot` method which changes the `speed` and `direction`. We are also going to create a method called `adjust_sensor`. This method is used to change the range of our robot's sensors which are used to detect obstacles. Here are the steps:

1. Define a function within the `DriveBot` class which accepts two additional parameters: one for a new speed and one for a new direction
2. Replace the instance variables for `speed` and `direction` with the input parameters
3. Define another function called `adjust_sensor` which accepts one additional parameter
4. Replace the instance variable `sensor_range` with the input parameter



In the `DriveBot` class, add a method called `control_bot` which accepts parameters: `new_speed` and `new_direction`. These should replace the associated instance variables. Create another method called `adjust_sensor` which accepts a parameter called `new_sensor_range` which replaces the `sensor_range` instance variable. Afterwards, use these methods to rotate the robot `180` degrees at `10` miles per hour with a sensor range of `20` feet. Use the provided print statements to check your code!

▼ Hint

Remember that the format for creating methods in a class looks like this: `def method_name(self, variable_1, variable_2):` and we can modify instance variables like so: `self.instance_variable_name = variable_1`.

```python
class DriveBot:
    def __init__(self):
        self.motor_speed = 0
        self.direction = 0
        self.sensor_range = 0

    # Add the methods here!
    def control_bot(self, new_speed, new_direction):
        self.motor_speed = new_speed
        self.direction = new_direction


    def adjust_sensor(self, new_sensor_range):
        self.sensor_range = new_sensor_range
robot_1 = DriveBot()
```

```
10
180
20
```

Here are the methods we added:

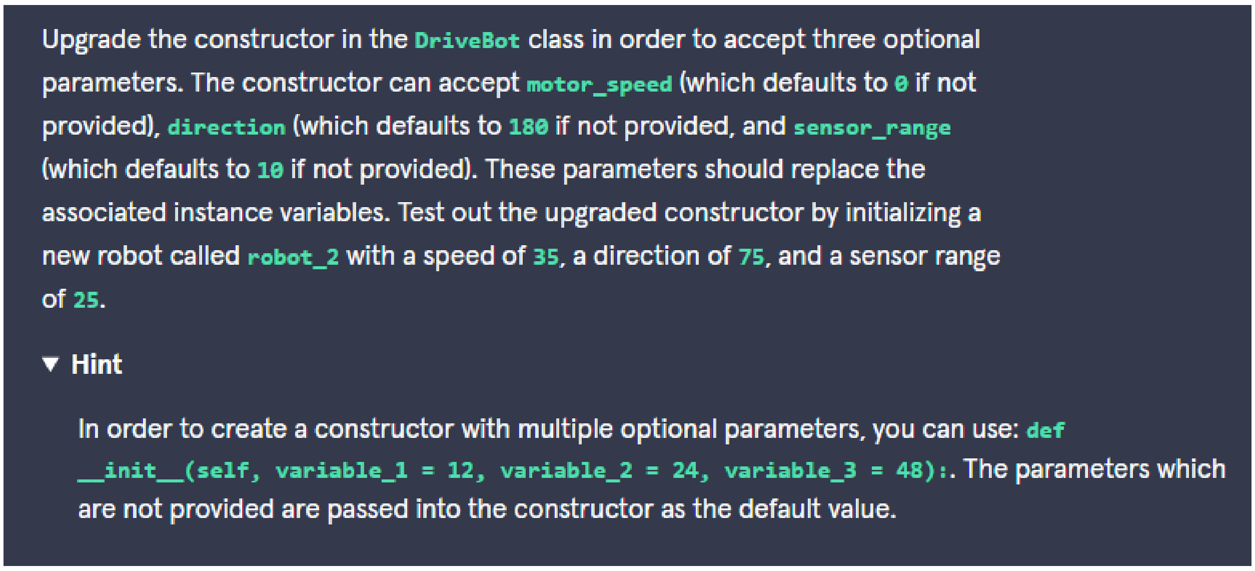
```python
def control_bot(self, new_speed, new_direction):
    self.motor_speed = new_speed
    self.direction = new_direction

def adjust_sensor(self, new_sensor_range):
    self.sensor_range = new_sensor_range
```

These two methods were added inside of the `DriveBot` class. They are used to replace the instance variables with new values from the input parameters. We use `self.variable_name` to access a certain instance variable within the class.

3. Enhanced Constructor

It can be tedious manually setting the values for each instance variable after we have created an object from the `DriveBot` class. We can enhance our constructor to automatically assign the values we provide to the instance variables. Instead of taking zero parameters, we are going to make the constructor take three parameters. Here is what we need to do:

1. Modify the constructor to take three parameters (in addition to `self`): one for `motor_speed`, one for `direction`, and one for `sensor_range`
2. For the first parameter, make the default value `0`
3. For the second parameter, make the default value `180`
4. For the third parameter, make the default value `10`
5. Within the constructor, replace the instance variables with the variables from the input parameters



Upgrade the constructor in the `DriveBot` class in order to accept three optional parameters. The constructor can accept `motor_speed` (which defaults to `0` if not provided), `direction` (which defaults to `180` if not provided, and `sensor_range` (which defaults to `10` if not provided). These parameters should replace the associated instance variables. Test out the upgraded constructor by initializing a new robot called `robot_2` with a speed of `35`, a direction of `75`, and a sensor range of `25`.

▼ **Hint**

In order to create a constructor with multiple optional parameters, you can use: `def __init__(self, variable_1 = 12, variable_2 = 24, variable_3 = 48):`. The parameters which are not provided are passed into the constructor as the default value.

```
class DriveBot:
    # Update this constructor!
    def __init__(self, motor_speed=0, direction=180, sensor_range=10):
        self.motor_speed = motor_speed
        self.direction = direction
        self.sensor_range = sensor_range
```

```python
    def control_bot(self, new_speed, new_direction):
        self.motor_speed = new_speed
        self.direction = new_direction

    def adjust_sensor(self, new_sensor_range):
        self.sensor_range = new_sensor_range

robot_1 = DriveBot()
robot_1.motor_speed = 5
robot_1.direction = 90
robot_1.sensor_range = 10

# Create robot_2 here!
robot_2 = DriveBot(35, 75, 25)

print(robot_2.motor_speed)
print(robot_2.direction)
print(robot_2.sensor_range)
```

Here is the updated constructor:

```python
def __init__(self, motor_speed = 0, direction = 180, sensor_range = 10):
    self.motor_speed = motor_speed
    self.direction = direction
    self.sensor_range = sensor_range
```

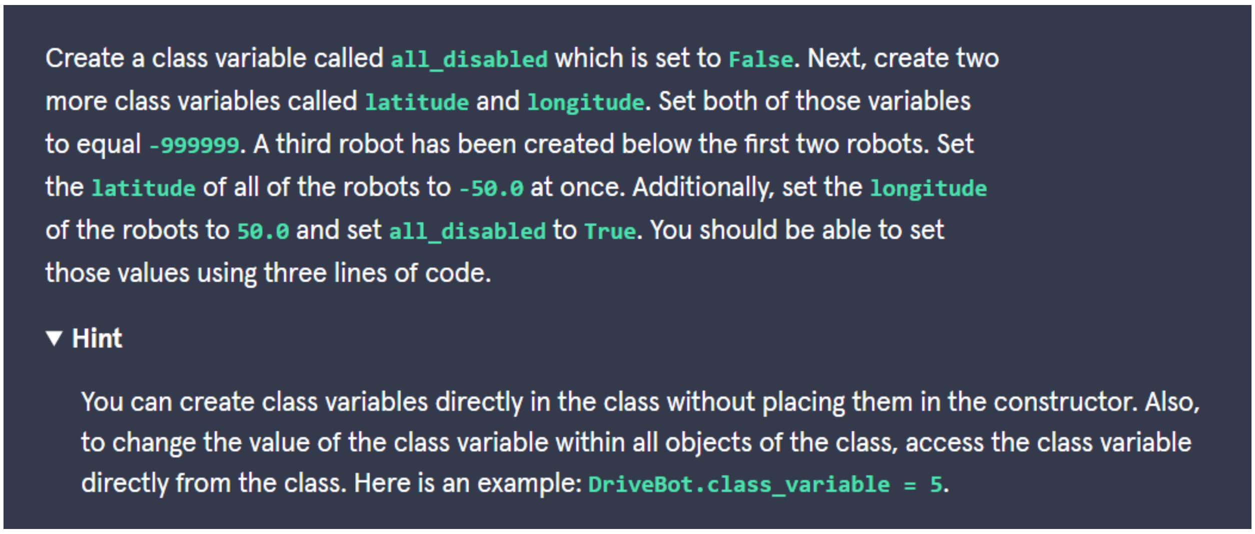
This upgraded constructor includes input parameters as well as default values for those parameters. This means that if no value is provided for those parameters, then the value they are set equal to will be used. Here are some examples of different ways to use the constructor:

```python
# sensor_range defaults to 10
test_bot_1 = DriveBot(10, 270)

# direction defaults to 180
test_bot_2 = DriveBot(sensor_range = 20, motor_speed = 45)

# direction defaults to 180 and sensor_range defaults to 10
test_bot_3 = DriveBot(50)

# all default values are used
test_bot_4 = DriveBot()

# no default values are used
test_bot_5 = DriveBot(18, 95, 30)
```

4. Controlling Them All

We want to add a new feature which allows the use to control multiple robots at once. The robots should be able to all have the same `latitude` and `longitude` GPS destination coordinates as well as a setting for disabling them all called `all_disabled`. We can accomplish this using class variables. Here is how we can do it:

1. Create a new class variable within the `DriveBot` class called `all_disabled` and set it equal to `False`
2. Create a new class variable within the `DriveBot` class called `latitude` and set it equal to -999999
3. Create a new class variable within the `DriveBot` class called `longitude` and set it equal to -999999
4. Outside of the class, test the class variables by setting the `longitude` of all robots to `50.0`, the `latitude` to `-50.0` and `all_disabled` to `True`



Create a class variable called `all_disabled` which is set to `False`. Next, create two more class variables called `latitude` and `longitude`. Set both of those variables to equal `-999999`. A third robot has been created below the first two robots. Set the `latitude` of all of the robots to `-50.0` at once. Additionally, set the `longitude` of the robots to `50.0` and set `all_disabled` to `True`. You should be able to set those values using three lines of code.

▼ **Hint**

You can create class variables directly in the class without placing them in the constructor. Also, to change the value of the class variable within all objects of the class, access the class variable directly from the class. Here is an example: `DriveBot.class_variable = 5`.

```python
class DriveBot:
  # Create the class variables!
  all_disabled = False
  latitude = -999999
  longitude = -999999

  def __init__(self, motor_speed = 0, direction = 180, sensor_range = 10):
        self.motor_speed = motor_speed
        self.direction = direction
        self.sensor_range = sensor_range

  def control_bot(self, new_speed, new_direction):
        self.motor_speed = new_speed
        self.direction = new_direction

  def adjust_sensor(self, new_sensor_range):
        self.sensor_range = new_sensor_range
```

```
robot_1 = DriveBot()
robot_1.motor_speed = 5
robot_1.direction = 90
robot_1.sensor_range = 10

robot_2 = DriveBot(35, 75, 25)
robot_3 = DriveBot(20, 60, 10)

# Change the latitude, longitude, and all_disabled values for all three robots us
ing only three lines of code!
DriveBot.longitude = 50.0
DriveBot.latitude = -50.0
DriveBot.all_disabled = True

print(robot_1.latitude)
print(robot_2.longitude)
print(robot_3.all_disabled)
```

Here are the changes we made in the class:

```
class DriveBot:
    all_disabled = False
    latitude = -999999
    longitude = -999999
```

We placed the class variables at the top of the class outside of the constructor. These variables can be accessed within the scope of the entire class. This means that the class variables contained within every object from the `DriveBot` class will change if we modify the class variable directly. Here is an example of how to change each of these class variables:

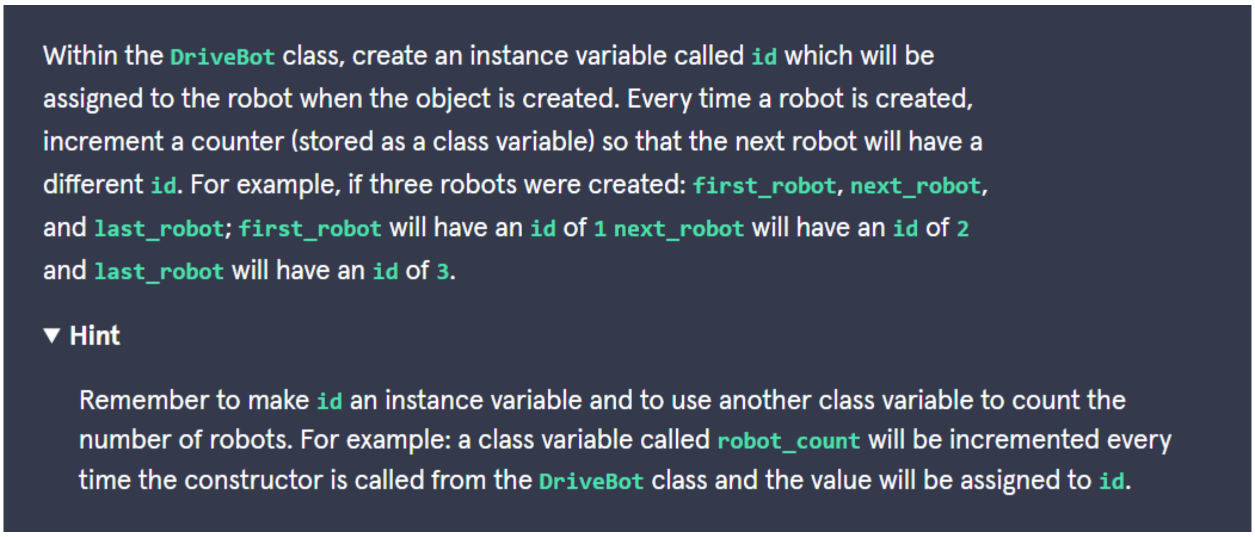
```
DriveBot.longitude = -79.98553
DriveBot.latitude = 40.60793
DriveBot.all_disabled = False
```

5. Identifying Robots

In order to keep track of the robots we are creating, we want to be able to assign an ID value to each robot when it is created. If we create five robots in a row, we want the IDs of each robot to be 1, 2, 3, 4, and 5 respectively. This can be accomplished by using a class variable counter which increments and is assigned to an instance variable for the ID whenever the constructor is called. Here are the steps:

1. Create a new class variable in the `DriveBot` class called `robot_count`
2. In the constructor, increment the `robot_count` by `1`

3. After incrementing the value, assign the value of `robot_count` to a new instance variable called `id`.



Within the `DriveBot` class, create an instance variable called `id` which will be assigned to the robot when the object is created. Every time a robot is created, increment a counter (stored as a class variable) so that the next robot will have a different `id`. For example, if three robots were created: `first_robot`, `next_robot`, and `last_robot`; `first_robot` will have an `id` of `1` `next_robot` will have an `id` of `2` and `last_robot` will have an `id` of `3`.

▼ **Hint**

Remember to make `id` an instance variable and to use another class variable to count the number of robots. For example: a class variable called `robot_count` will be incremented every time the constructor is called from the `DriveBot` class and the value will be assigned to `id`.

```
class DriveBot:
  # Create a counter to keep track of how many robots were created
    all_disabled = False
    latitude = -999999
    longitude = -999999
    robot_count = 0

    def __init__(self, motor_speed = 0, direction = 180, sensor_range = 10):
        self.motor_speed = motor_speed
        self.direction = direction
        self.sensor_range = sensor_range
        # Assign an `id` to the robot when it is constructed by incrementing the
counter and assigning the value to `id`
        DriveBot.robot_count += 1
        self.id = DriveBot.robot_count



    def control_bot(self, new_speed, new_direction):
        self.motor_speed = new_speed
        self.direction = new_direction

    def adjust_sensor(self, new_sensor_range):
        self.sensor_range = new_sensor_range

robot_1 = DriveBot()
robot_1.motor_speed = 5
robot_1.direction = 90
robot_1.sensor_range = 10
robot_1.id = 1

robot_2 = DriveBot(35, 75, 25)
robot_3 = DriveBot(20, 60, 10)

print(robot_1.id)
print(robot_2.id)
print(robot_3.id)
```

Here is the complete class:

```
class DriveBot:
    all_disabled = False
    latitude = -999999
    longitude = -999999
    robot_count = 0

    def __init__(self, motor_speed = 0, direction = 180, sensor_range = 10):
        self.motor_speed = motor_speed
        self.direction = direction
        self.sensor_range = sensor_range
        DriveBot.robot_count += 1
        self.id = DriveBot.robot_count

    def control_bot(self, new_speed, new_direction):
        self.motor_speed = new_speed
        self.direction = new_direction

    def adjust_sensor(self, new_sensor_range):
        self.sensor_range = new_sensor_range
```

The final modifications to this class can be seen at the top of the class and in the constructor. We use a class variable to keep track of the total number of robots. This information is shared across all robot objects we create from the class. Every time a robot object is created, the constructor is called and the count is incremented. Each robot has an instance variable for `id` which is local to that robot object. This is assigned at the time of construction and stores what the count was at that time.