**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.

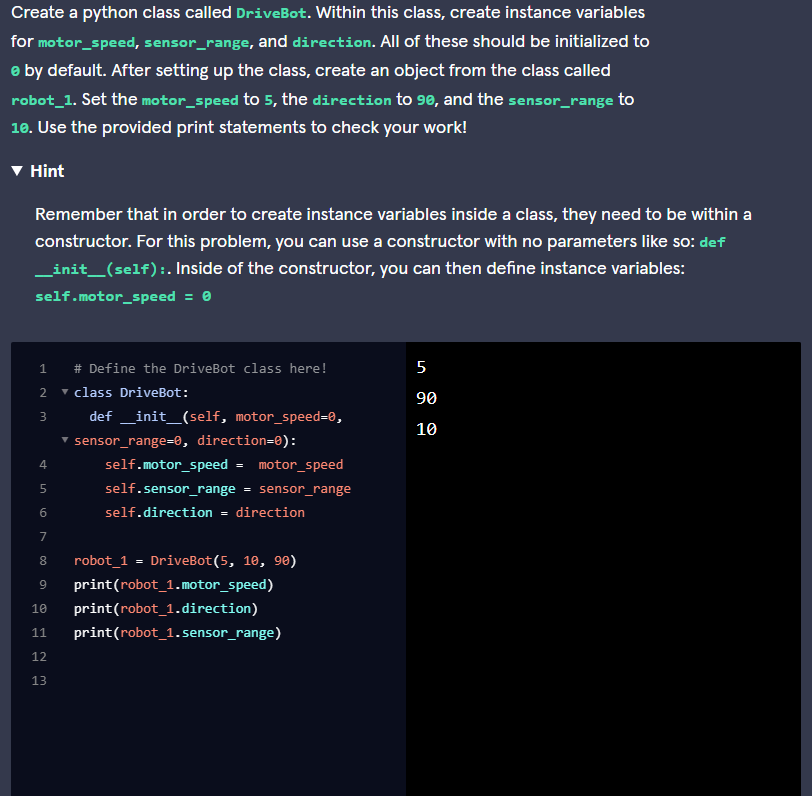
**Challenges**

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**



Here is how we solved this:

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:

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



Here are the methods we added:

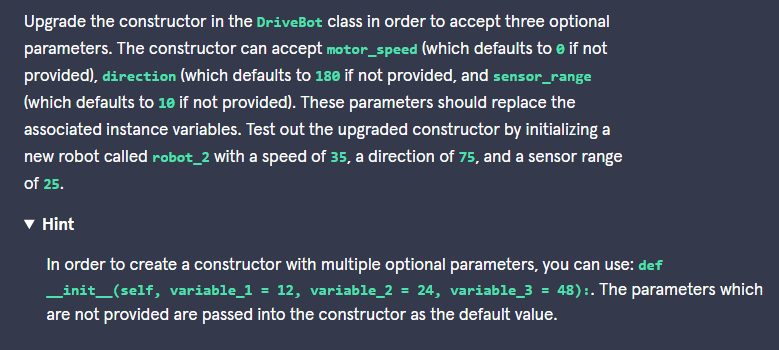
def control\_bot(self, new\_speed, new\_direction):  
    self.motor\_speed = new\_speed  
    self.direction = new\_direction  
   
def adjust\_sensor(self, new\_sensor\_range):  
    self.sensor\_range = new\_sensor\_range

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



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

    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:

def \_\_init\_\_(self, motor\_speed = 0, direction = 180, sensor\_range = 10):  
    self.motor\_speed = motor\_speed  
    self.direction = direction  
    self.sensor\_range = sensor\_range

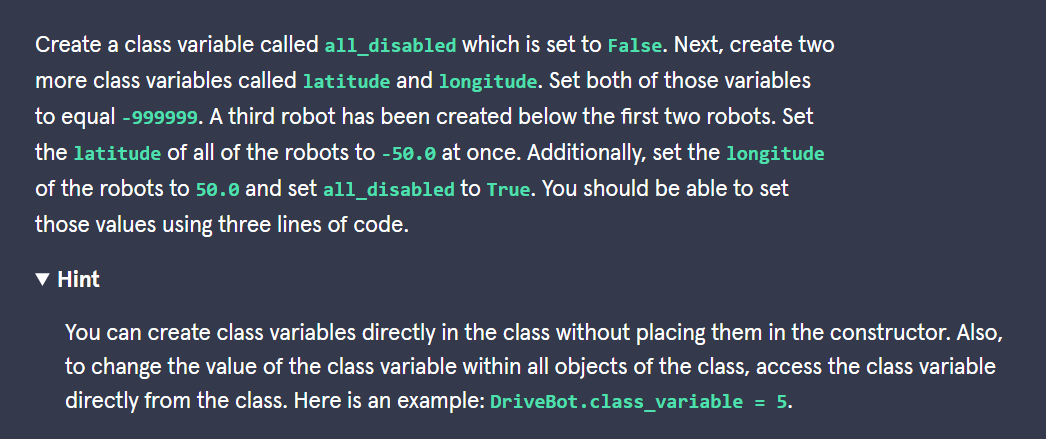
This upgraded constructor includes input parameters as well as default values for those parameters. This means that if no value is provided for those parameters, then the value they are set equal to will be used. Here are some examples of different ways to use the constructor:

# 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**



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 using 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**.

