Learning Quiz 27: Inheritance

Due Nov 13 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm
Time Limit None **Allowed Attempts** Unlimited

Instructions

Prior to completing this quiz, be sure to read:

- Sections 8.5: Inheritance (p. 264-272)

Please also go over Practice Problem 8.8 in the textbook (solution at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	8 minutes	2.87 out of 5

Score for this attempt: **2.87** out of 5
Submitted Nov 19 at 1:20pm
This attempt took 8 minutes.

Question 1

0 / 1 pts

In Python, you can use classes within classes. Consider the case that you have a class Vehicle. Inside the Vehicle, you have another class Seat, which is either occupied or not occupied. Below, we'll see a Vehicle class that adopts x number of Seats. We also create a function getseat() to get the seat status. Some other functions have been omitted:

```
class Seat:
    def __init__(self):
        'initializes the seat and seat variables'
        self.status = "vacant"

    def occupy(self):
        'makes the seat status occupied'
        self.status = "occupied"

    def vacate(self):
        'makes the seat status to vacant'
        self.status = "vacant"

    def getstatus(self):
        'returns whether the seat is occupied or vacant'
        return self.status


class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000, seats =
 5):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0
        self.seats = []
        for i in range(seats):
            self.seats.append(Seat())

    def getseat(self, num):
        'returns the status of the seat'
        return self.seats[num].getstatus()


hisCar = Vehicle('green', 'Honda', 'Accord')
print(hisCar.getseat(2))
```

In the example above, we have five generic seats in a car. Suppose we would like to create two separate variables, one for the driver's seat and one for all other seats. How would you modify the following line of code to get two variables to represent all seats in the car:

```
self.seats = []
for i in range(seats):
    self.seats.append(Seat())
```

```
self.driverseat = Seat()
self.seats = []
for i in range(seats):
    self.seats.append(Seat())
```

```
self.driverseat = Seat()
self.seats = []
for i in range(seats - 1):
    seats.append(Seat())
```

```
driverseat = Seat()
seats = []
for i in range(seats - 1):
    seats.append(Seat())
```

orrect Answer

```
self.driverseat = Seat()
self.seats = []
for i in range(seats - 1):
    self.seats.append(Seat())
```

ou Answered

[✓]
```
self.driverseat = Seat()
self.seat1 = Seat()
self.seat2 = Seat()
self.seat3 = Seat()
self.seat4 = Seat()
```

Question 2

0.2 / 1 pts

Suppose we have our class Vehicle, but we want a more specific class called Car and Motorcycle. Rather than re-write our classes all over again, we can adopt the concept of **inheritance**. For our Car class which will adopt all of the properties of Vehicle, this might look like:

```
class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0

    def getcolor(self):
        'returns the color of the vehicle'
        return self.color

    def setcolor(self, color):
        'sets the vehicle color'
        self.color = color

    def getmake(self):
        'returns the make of the vehicle'
        return self.make

    def getmodel(self):
        'returns the model of the vehicle'
        return self.model

    def refuel(self, gallons):
        'returns the model of the vehicle'
        self.fuel += gallons

class Car(Vehicle):              ## Note, we have Vehicle in parentheses
    pass

hisCar = Car('green', 'Honda', 'Accord')
print(hisCar.getmodel())
```

Note that I created a Car class and used the keyword 'pass' to create an empty block. Although I don't have any defined functions under Car, functions under my Vehicle class can also be used for my Car class. That's because the Car class inherited all of the features of the Vehicle superclass.

Given the code in the example above, which functions can we use for hisCar?

ou Answered

[✓ hisCar.setyear(2000)]

ou Answered

[✓ hisCar.setmodel('Camry')]

Correct!

[✓ hisCar.getmodel()]

Correct!

[✓] hisCar.refuel(5)

[ ] hisCar.getyear()

[ ] hisCar.getseat(3)

ou Answered

[✓] hisCar.setmake('Toyota')

Correct!

[✓] hisCar.getmake()

Correct!

[✓] hisCar.setcolor('black')

orrect Answer

[ ] hisCar.getcolor()

Question 3

0.67 / 1 pts

You can define additional functions under your Car class. In the example below, we create a getwheels() function:

```
class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0

    def getcolor(self):
        'returns the color of the vehicle'
        return self.color

    def setcolor(self, color):
        'sets the vehicle color'
        self.color = color

    def getmake(self):
        'returns the make of the vehicle'
        return self.make

    def getmodel(self):
        'returns the model of the vehicle'
        return self.model
    def refuel(self, gallons):
        'returns the model of the vehicle'
        self.fuel += gallons

class Car(Vehicle):
    def getwheels(self):
        'returns the number of wheels on the car'
        return self.wheels

    def getfuel(self):
        'returns the fuel availability in car'
        return self.fuel

hisCar = Car('green', 'Honda', 'Accord')
herCar = Vehicle('black', 'Toyota', 'Camry')
hiswheels = hisCar.getwheels()
herwheels = herCar.getwheels()  # error because herCar is a Vehicle()
```

Note, that since the getwheels() function is only in the Car class, only objects Car() can use the function. We'll get an error if we have a Vehicle() that tries to use getwheels().

Consider the code in given in this question. Which of the following functions can be used?

Correct! ✓ herCar.getcolor()

Correct! ✓ hisCar.getfuel()

Correct! ✓ hisCar.getmake()

ou Answered ✓ herCar.getfuel()

Correct! ✓ hisCar.getwheels()

Correct! ✓ hisCar.getcolor()

Correct! ✓ herCar.getmake()

ou Answered ✓ herCar.getwheels()

Question 4

1 / 1 pts

Suppose we have a function in the superclass Vehicle that we'd also like to use in our Car class. However, we would like for the functions to behave differently in our Car class. You can modify an existing function by simply overwriting it. Below, we overwrite setColor() in our Car class so that it prints an extra statement:

```
class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0

    def getcolor(self):
        'returns the color of the vehicle'
        return self.color

    def setcolor(self, color):
        'sets the vehicle color'
        self.color = color

    def getmake(self):
        'returns the make of the vehicle'
        return self.make

    def getmodel(self):
        'returns the model of the vehicle'
        return self.model

    def refuel(self, gallons):
        'returns the model of the vehicle'
        self.fuel += gallons

class Car(Vehicle):
    def getwheels(self):
        'returns the number of wheels on the car'
        return self.wheels

    def getfuel(self):
        'returns the fuel availability in car'
        return self.fuel

    def setcolor(self, color):
        'sets the vehicle color'
        self.color = color
        print('The car has been repainted', color)
hisCar = Car('green', 'Honda', 'Accord')
herCar = Vehicle('black', 'Toyota', 'Camry')
hisCar.setcolor('white')
herCar.setcolor('gray')
```

Note that while both hisCar and herCar are able to use the function setcolor(), hisCar will print an extra statement, as we have modified in the Car class.

Consider the code in this question's example. Which other functions in our superclass Vehicle can be modified in our Car class?

- ☐ getfuel()
- Correct! ☑ getmodel()
- Correct! ☑ refuel()
- ☐ getwheels()
- Correct! ☑ getmake()
- Correct! ☑ getcolor()

Question 5

1 / 1 pts

We can also modify the __init__() constructor to use variables to use a different set of variables from our superclass. See the example below (some functions have been omitted):

```
class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0

class Car(Vehicle):
    'represents a car'
    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the car and car variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0
        self.lockstatus = 'locked'
```

However, if you have a series of variables that are repeated from the Vehicle() class, and you simply want to add a new variable (such as lockstatus above) you can call the superclass Vehicle __init__() constructor as follows

:

```
class Vehicle:
    'Represents a drive-able vehicle'

    def __init__(self, color, make, model, wheels = 4, year = 2000):
        'initializes the vehicle and vehicle variables'
        self.color = color
        self.wheels = wheels
        self.make = make
        self.model = model
        self.year = year
        self.fuel = 0

class Car(Vehicle):
    'represents a car'

    def __init__(self, color, make, model, wheels = 4, year = 2000, seats =
 5):
        'initializes the car and car variables'
        self.lockstatus = 'locked'
        Vehicle.__init__(self, color, make, model, wheels = 4, year = 2000,
 seats = 5)
```

Now, all of Vehicle()'s variables and lockstatus are available to Car.

Consider the following classes:

```
class WritingTool:
    'represents a writing tool'

    def __init__(self, color, weight, length):
        self.color = color
        self.weight = weight
        self.length = length

class Pencil(WritingTool):
    'represents a pencil'

    def __init__(self, weight, length, type):
        WritingTool.__init__(self,"gray",weight,length)
        self.type = type

class LeadPencil(Pencil):
    'represents a lead pencil'
    def __init__(self, weight, length, type):
        Pencil.__init__(self, weight, length, type)
        self.leadsize = 0.7
        self.leadsticks = 0
```

Select the classes that the variable is available in:

Correct!	**color**	WritingTool, Pencil, anc
Correct!	**weight**	WritingTool, Pencil, anc
Correct!	**length**	WritingTool, Pencil, anc
Correct!	**type**	Pencil and LeadPencil
Correct!	**leadsize**	LeadPencil only
Correct!	**leadsticks**	LeadPencil only

Other Incorrect Match Options:

- Pencil only
- WritingTool only
- WritingTool and LeadPencil only
- WritingTool and Pencil only

Quiz Score: **2.87** out of 5