Computational Analysis of Physical Systems (Lecture 9)

Object Oriented Programming (OOP)

Terminology (1)

- **Objects** are collections of data and functions that operate on that data.

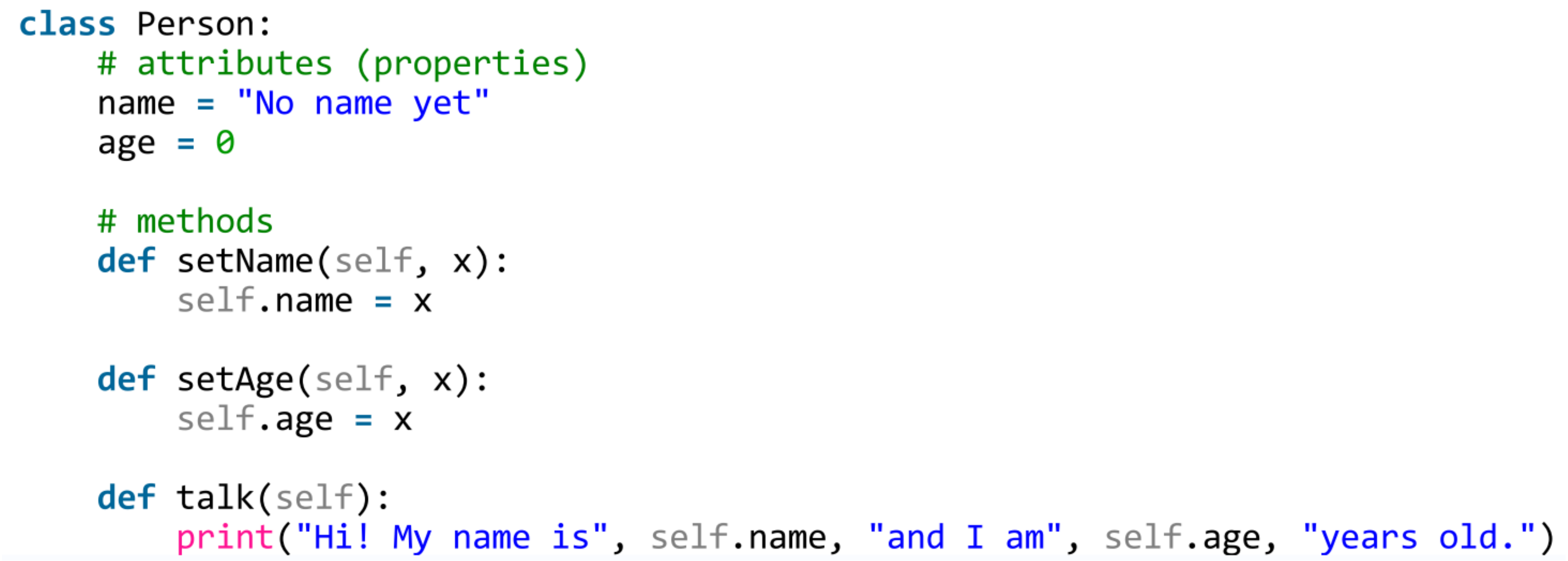
- An object in Object Oriented Programming is characterized by
  a) its **attributes** (properties)
  b) **methods** that do something with the attributes (for example change them)

Terminology (2)

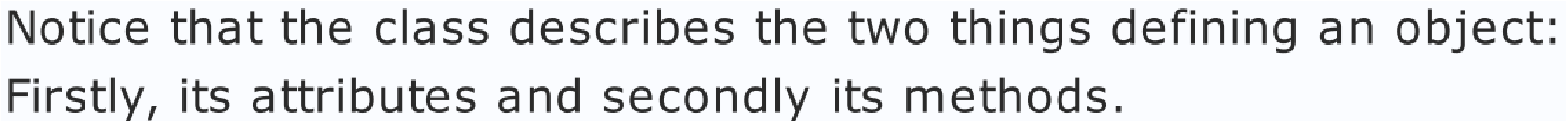
- Just as data has various types so objects can have different types. These collections of objects with identical characteristics are collectively known as a **class.**

- Marie Curie, Isaac Newton and Emmy Noether have something in common. They are persons. They are objects of the class “Person”.
- Apple, banana, mango: These are objects of the class “Fruit”.
- Jacket, blouse, pants: These are objects of the class “Clothes”.

Class Person

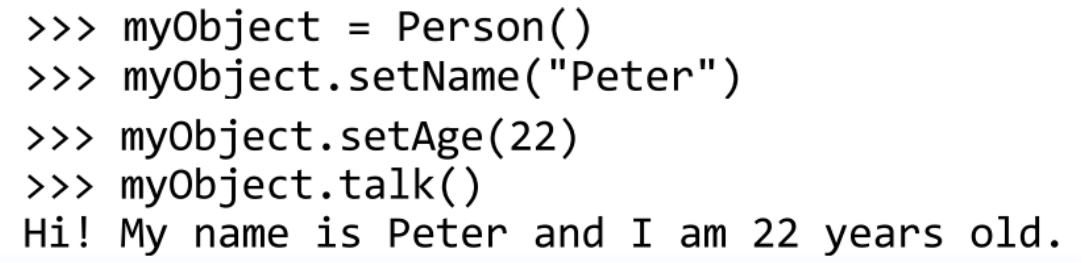


```python
class Person:
    # attributes (properties)
    name = "No name yet"
    age = 0

    # methods
    def setName(self, x):
        self.name = x

    def setAge(self, x):
        self.age = x

    def talk(self):
        print("Hi! My name is", self.name, "and I am", self.age, "years old.")
```

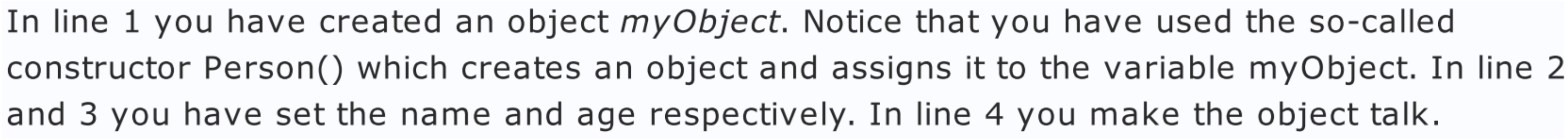


Notice that the class describes the two things defining an object: Firstly, its attributes and secondly its methods.

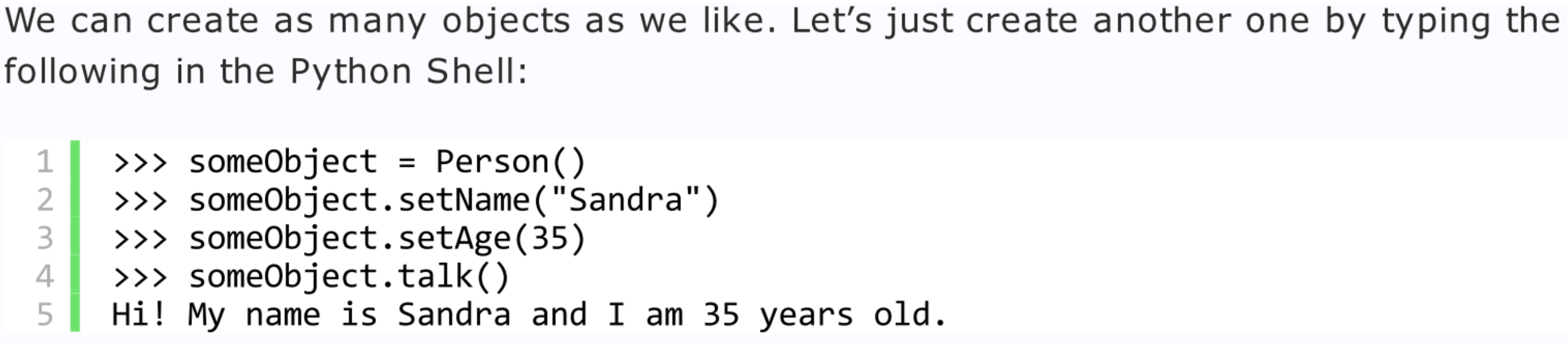
Class Person



```
>>> myObject = Person()
>>> myObject.setName("Peter")
>>> myObject.setAge(22)
>>> myObject.talk()
Hi! My name is Peter and I am 22 years old.
```



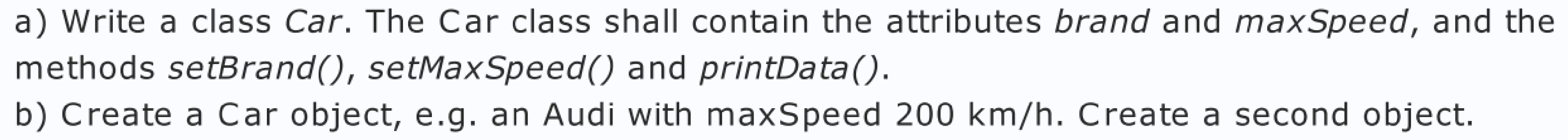
In line 1 you have created an object *myObject*. Notice that you have used the so-called constructor Person() which creates an object and assigns it to the variable myObject. In line 2 and 3 you have set the name and age respectively. In line 4 you make the object talk.



We can create as many objects as we like. Let’s just create another one by typing the following in the Python Shell:

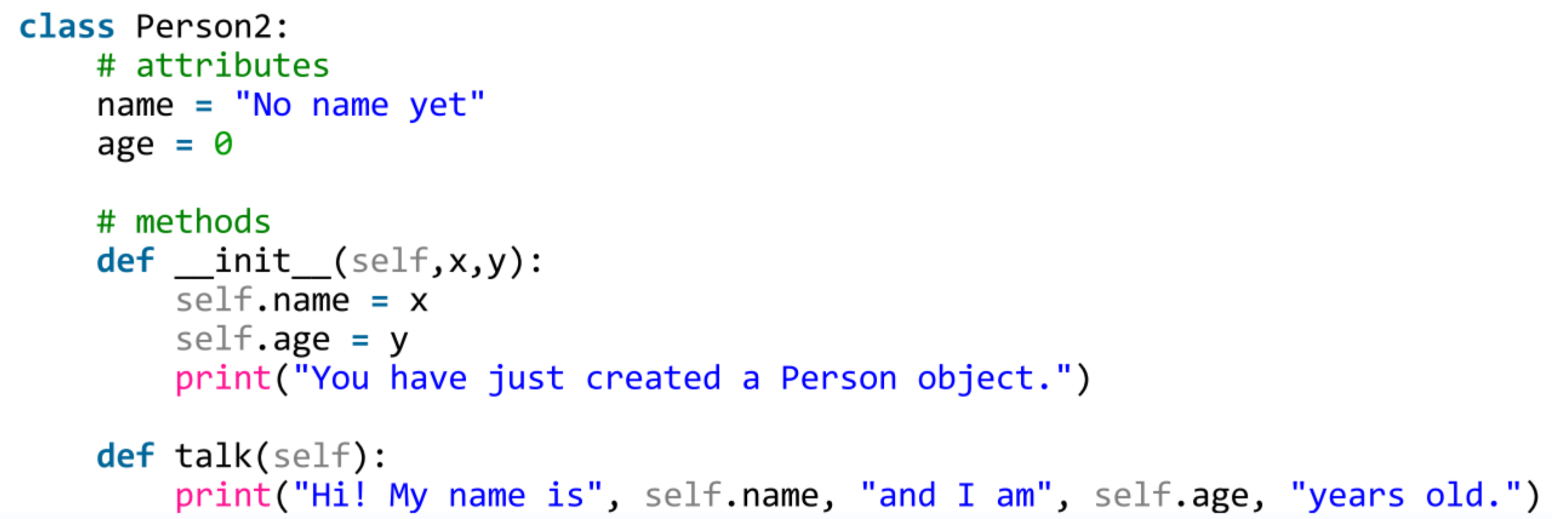
```
>>> someObject = Person()
>>> someObject.setName("Sandra")
>>> someObject.setAge(35)
>>> someObject.talk()
Hi! My name is Sandra and I am 35 years old.
```

Exercise

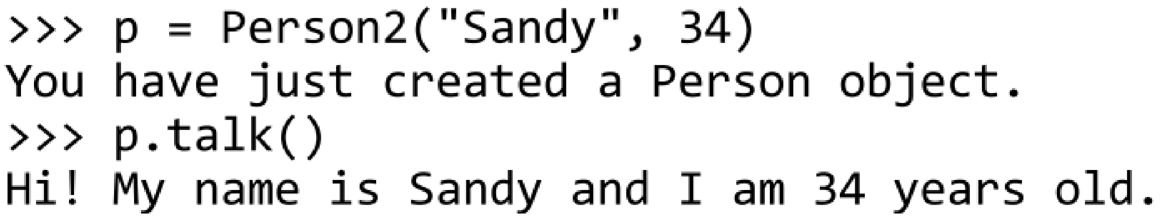


a) Write a class *Car*. The Car class shall contain the attributes *brand* and *maxSpeed*, and the methods *setBrand()*, *setMaxSpeed()* and *printData()*.
b) Create a Car object, e.g. an Audi with maxSpeed 200 km/h. Create a second object.

Class person2

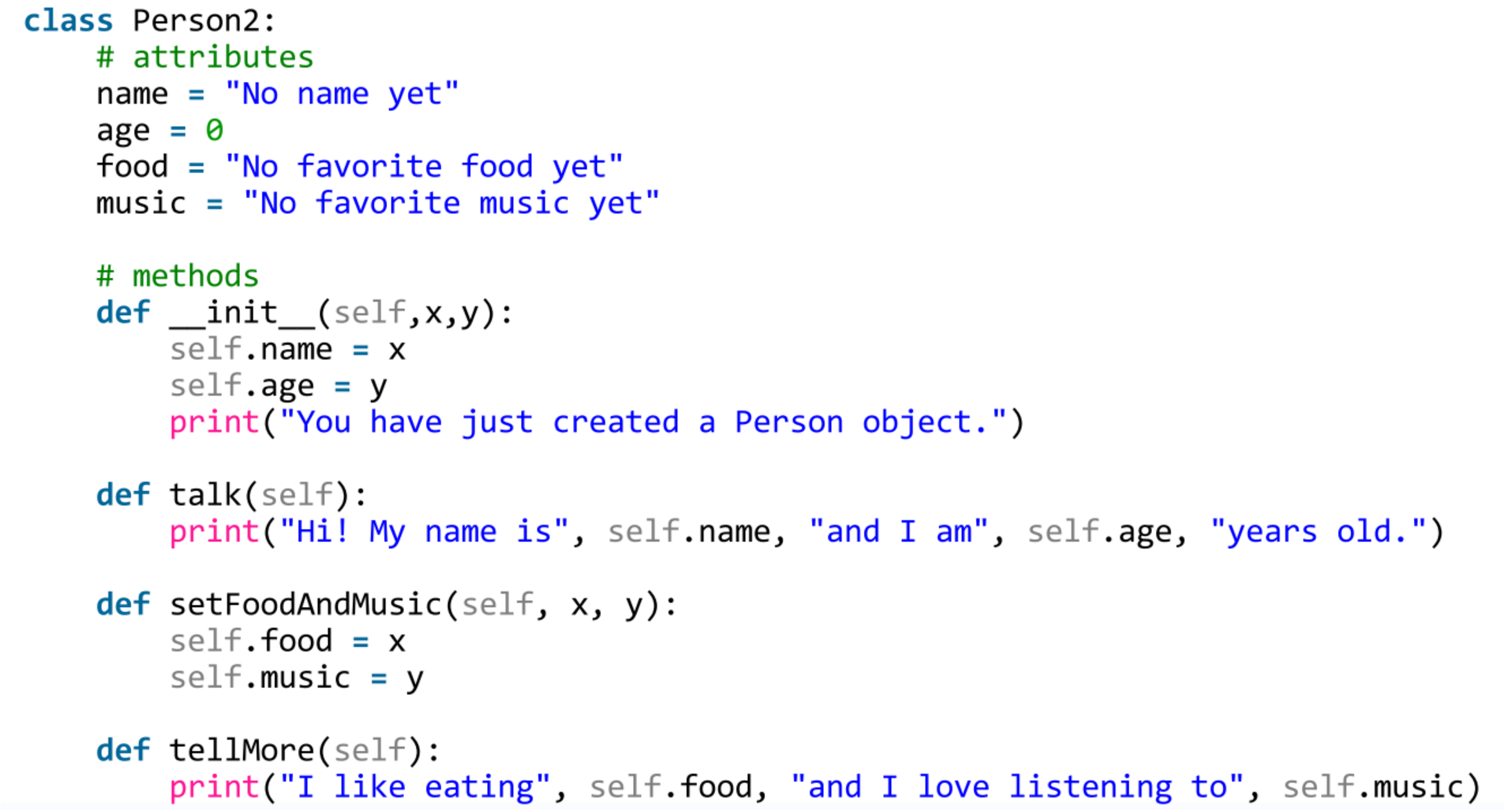


```
class Person2:
    # attributes
    name = "No name yet"
    age = 0

    # methods
    def __init__(self,x,y):
        self.name = x
        self.age = y
        print("You have just created a Person object.")

    def talk(self):
        print("Hi! My name is", self.name, "and I am", self.age, "years old.")
```



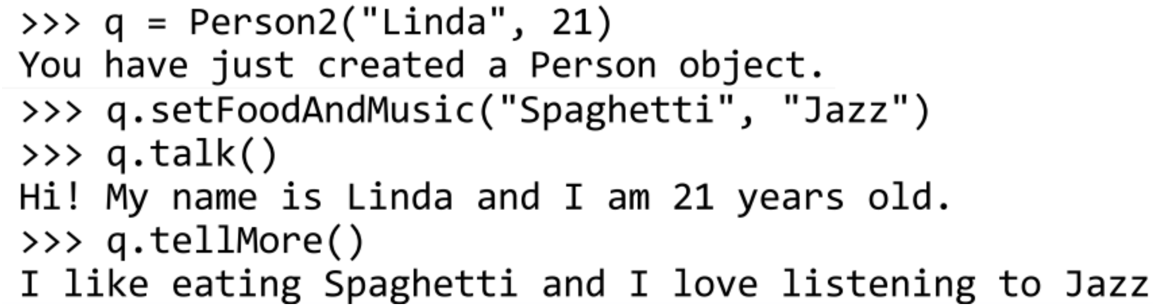
```
>>> p = Person2("Sandy", 34)
You have just created a Person object.
>>> p.talk()
Hi! My name is Sandy and I am 34 years old.
```

Class person2 with Food and Music



```
class Person2:
    # attributes
    name = "No name yet"
    age = 0
    food = "No favorite food yet"
    music = "No favorite music yet"

    # methods
    def __init__(self,x,y):
        self.name = x
        self.age = y
        print("You have just created a Person object.")

    def talk(self):
        print("Hi! My name is", self.name, "and I am", self.age, "years old.")

    def setFoodAndMusic(self, x, y):
        self.food = x
        self.music = y

    def tellMore(self):
        print("I like eating", self.food, "and I love listening to", self.music)
```

Class person2 with Food and Music

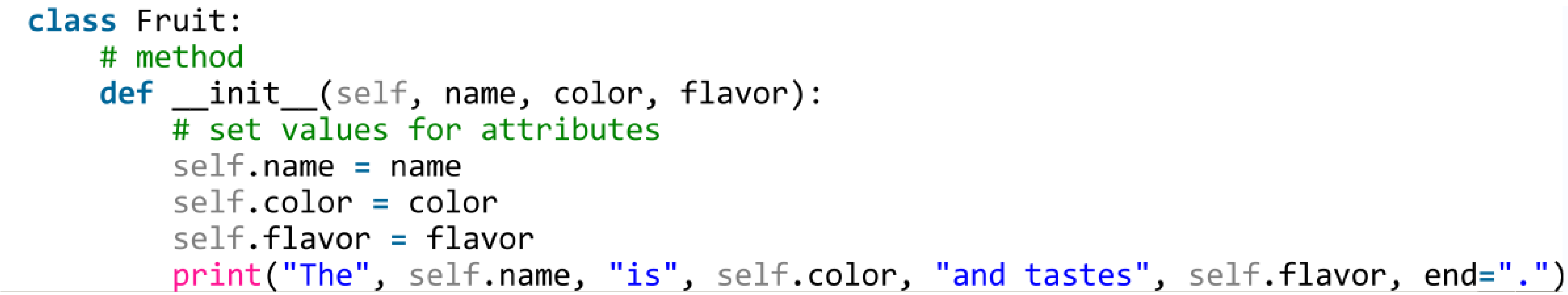


```
>>> q = Person2("Linda", 21)
You have just created a Person object.
>>> q.setFoodAndMusic("Spaghetti", "Jazz")
>>> q.talk()
Hi! My name is Linda and I am 21 years old.
>>> q.tellMore()
I like eating Spaghetti and I love listening to Jazz
```

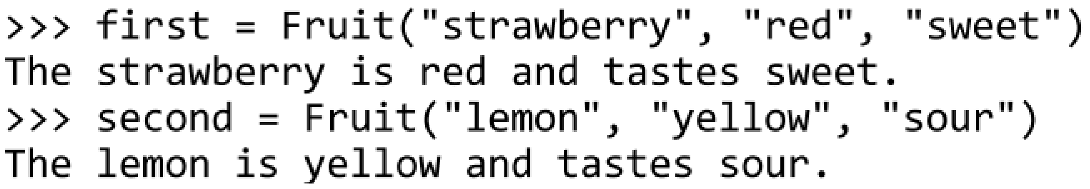
Exercise

- Rewrite the car class with __init__ method.

Class fruit

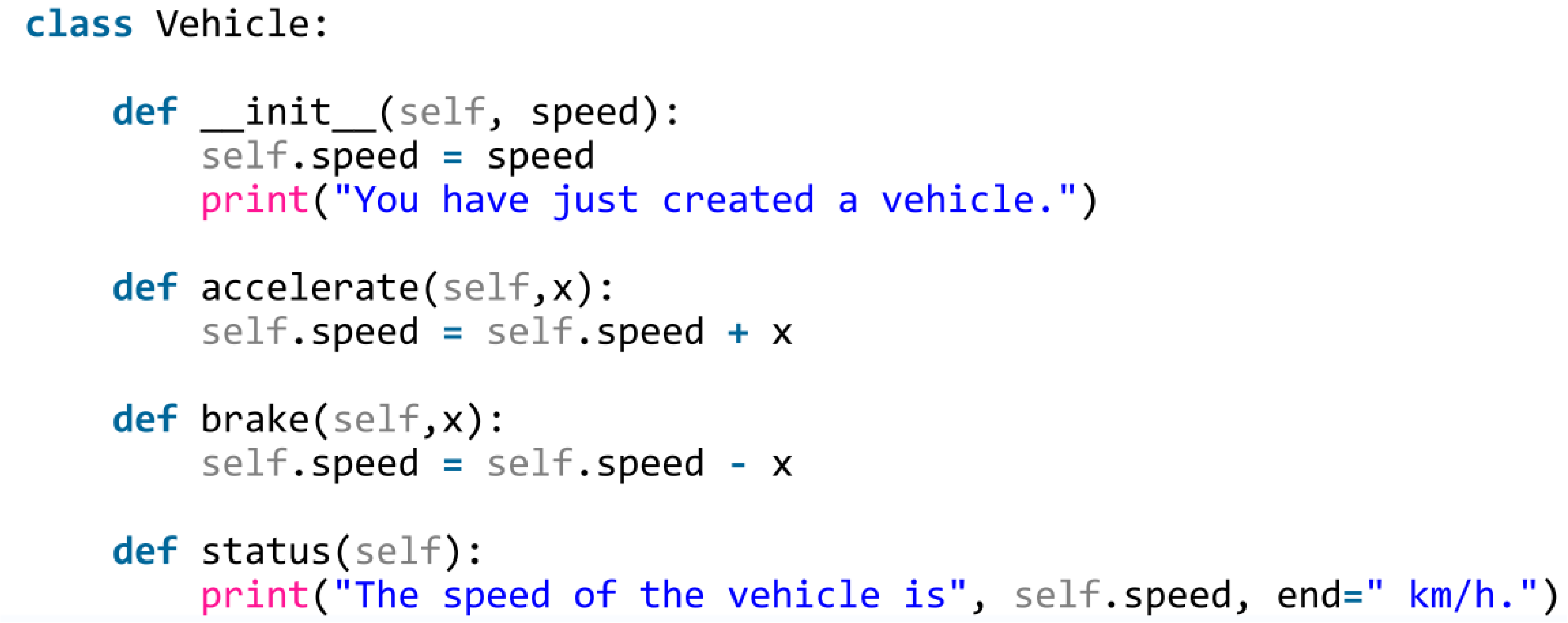


```python
class Fruit:
    # method
    def __init__(self, name, color, flavor):
        # set values for attributes
        self.name = name
        self.color = color
        self.flavor = flavor
        print("The", self.name, "is", self.color, "and tastes", self.flavor, end=".")
```



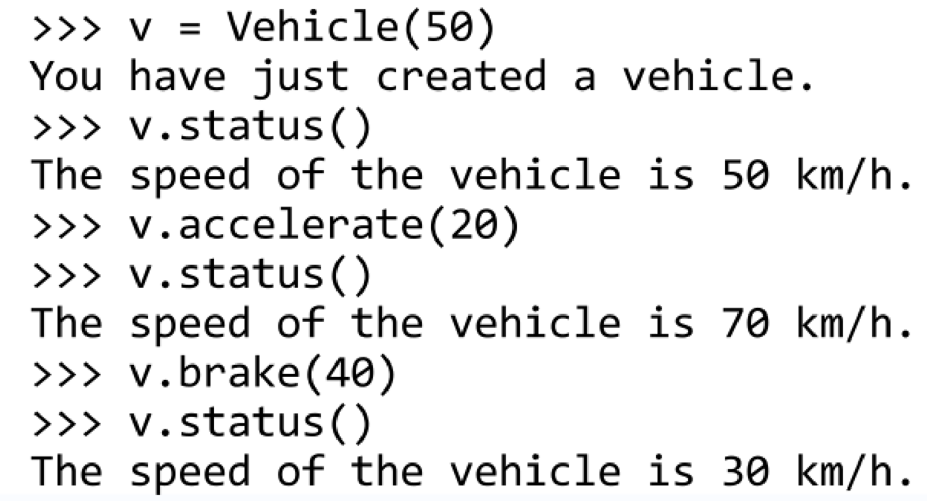
```
>>> first = Fruit("strawberry", "red", "sweet")
The strawberry is red and tastes sweet.
>>> second = Fruit("lemon", "yellow", "sour")
The lemon is yellow and tastes sour.
```

Class vehicle



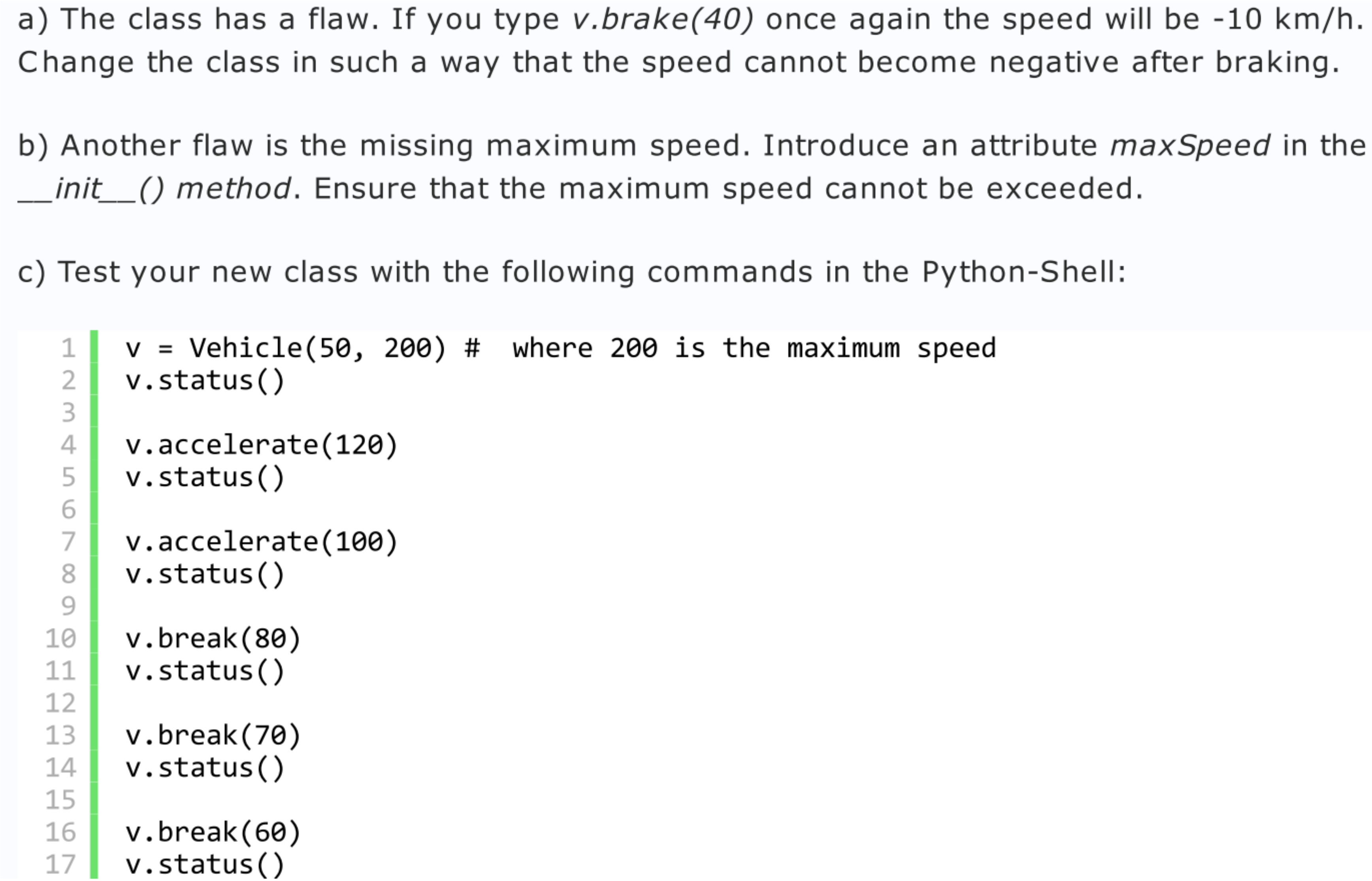
```
class Vehicle:

    def __init__(self, speed):
        self.speed = speed
        print("You have just created a vehicle.")

    def accelerate(self,x):
        self.speed = self.speed + x

    def brake(self,x):
        self.speed = self.speed - x

    def status(self):
        print("The speed of the vehicle is", self.speed, end=" km/h.")
```

Class vehicle



```
>>> v = Vehicle(50)
You have just created a vehicle.
>>> v.status()
The speed of the vehicle is 50 km/h.
>>> v.accelerate(20)
>>> v.status()
The speed of the vehicle is 70 km/h.
>>> v.brake(40)
>>> v.status()
The speed of the vehicle is 30 km/h.
```

Exercise



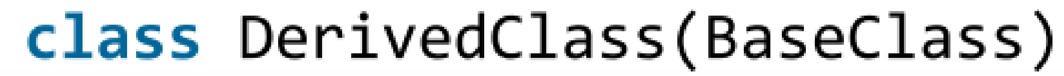
a) The class has a flaw. If you type *v.brake(40)* once again the speed will be -10 km/h. Change the class in such a way that the speed cannot become negative after braking.

b) Another flaw is the missing maximum speed. Introduce an attribute *maxSpeed* in the *__init__() method*. Ensure that the maximum speed cannot be exceeded.

c) Test your new class with the following commands in the Python-Shell:

```
v = Vehicle(50, 200) #  where 200 is the maximum speed
v.status()

v.accelerate(120)
v.status()

v.accelerate(100)
v.status()

v.break(80)
v.status()

v.break(70)
v.status()

v.break(60)
v.status()
```

Inheritance



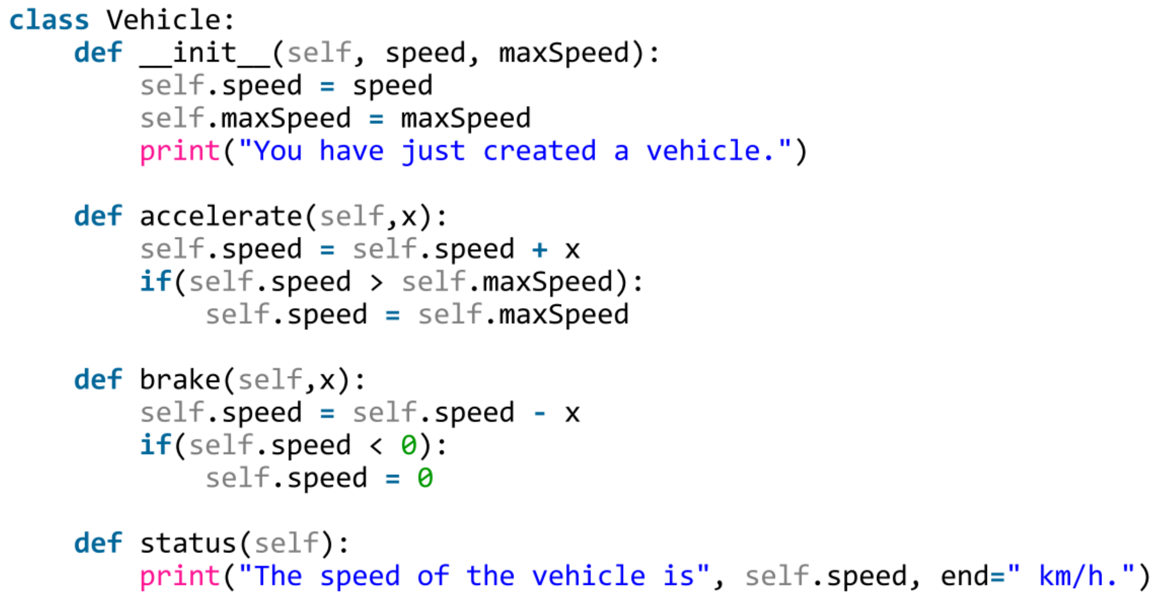
```
class DerivedClass(BaseClass)
```

We wrote a **Vehicle** class that allowed us to create objects
with the attribute *speed* and methods to change the speed.

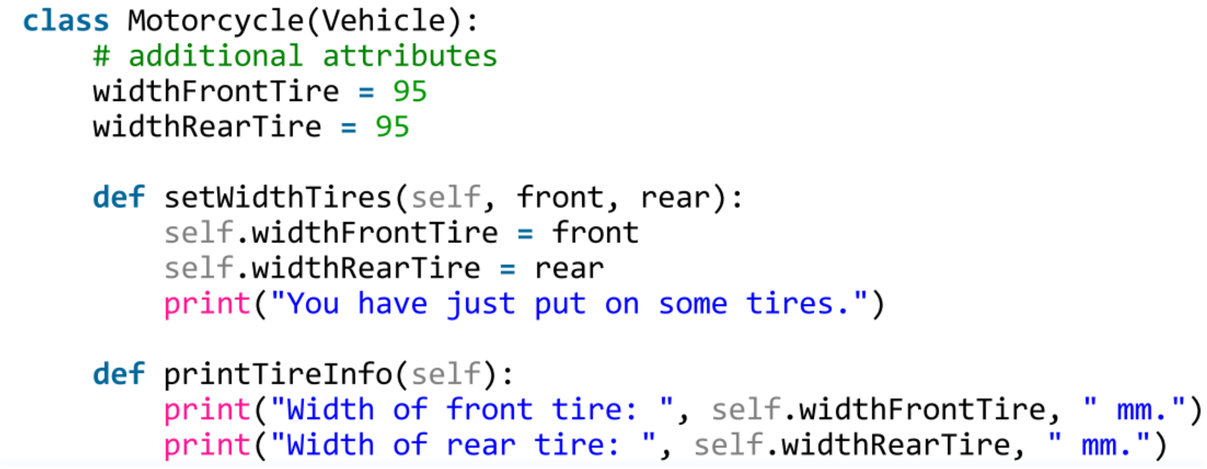
Suppose we wanted to create a **motorcycle** object that has
additional attributes such as the *width of its tires*.

Then we could write a completely new class *Motorcycle*
or **instead** reuse the Vehicle class to build the Motorcycle class.

Class motorcycle
(inherited from class vehicle)



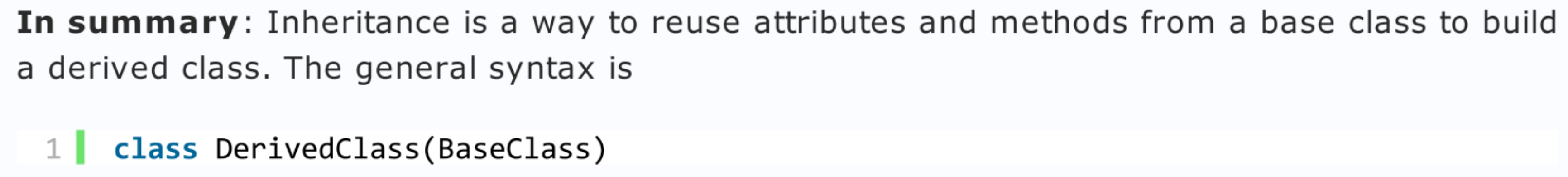
```python
class Vehicle:
    def __init__(self, speed, maxSpeed):
        self.speed = speed
        self.maxSpeed = maxSpeed
        print("You have just created a vehicle.")

    def accelerate(self,x):
        self.speed = self.speed + x
        if(self.speed > self.maxSpeed):
            self.speed = self.maxSpeed

    def brake(self,x):
        self.speed = self.speed - x
        if(self.speed < 0):
            self.speed = 0

    def status(self):
        print("The speed of the vehicle is", self.speed, end=" km/h.")
```



```python
class Motorcycle(Vehicle):
    # additional attributes
    widthFrontTire = 95
    widthRearTire = 95

    def setWidthTires(self, front, rear):
        self.widthFrontTire = front
        self.widthRearTire = rear
        print("You have just put on some tires.")

    def printTireInfo(self):
        print("Width of front tire: ", self.widthFrontTire, " mm.")
        print("Width of rear tire: ", self.widthRearTire, " mm.")
```

Class motorcycle
(inherited from class vehicle)

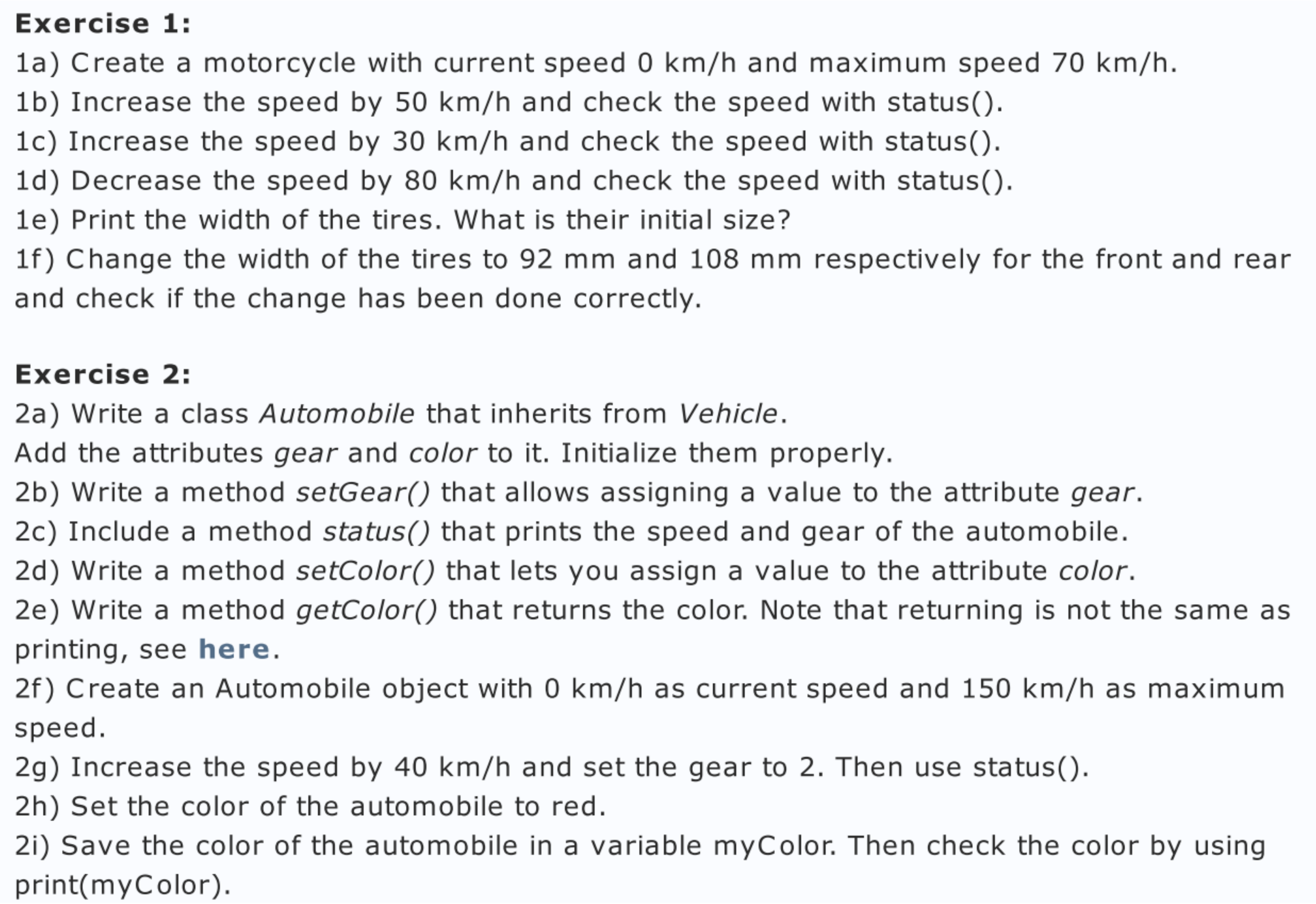
```
>>> m = Motorcycle(40, 120)
You have just created a vehicle.
>>> m.status()
The speed of the vehicle is 40 km/h.
>>> m.setWidthTires(90, 100)
You have just put on some tires.
>>> m.printTireInfo()
Width of front tire:  90  mm.
Width of rear tire:  100  mm.
```



In summary: Inheritance is a way to reuse attributes and methods from a base class to build a derived class. The general syntax is

```
class DerivedClass(BaseClass)
```

Exercises



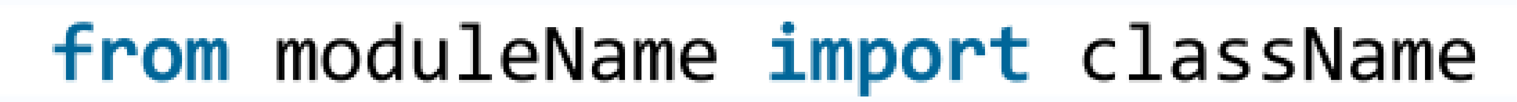
Exercise 1:

1a) Create a motorcycle with current speed 0 km/h and maximum speed 70 km/h.
1b) Increase the speed by 50 km/h and check the speed with status().
1c) Increase the speed by 30 km/h and check the speed with status().
1d) Decrease the speed by 80 km/h and check the speed with status().
1e) Print the width of the tires. What is their initial size?
1f) Change the width of the tires to 92 mm and 108 mm respectively for the front and rear and check if the change has been done correctly.

Exercise 2:

2a) Write a class *Automobile* that inherits from *Vehicle*.
Add the attributes *gear* and *color* to it. Initialize them properly.
2b) Write a method *setGear()* that allows assigning a value to the attribute *gear*.
2c) Include a method *status()* that prints the speed and gear of the automobile.
2d) Write a method *setColor()* that lets you assign a value to the attribute *color*.
2e) Write a method *getColor()* that returns the color. Note that returning is not the same as printing, see **here**.
2f) Create an Automobile object with 0 km/h as current speed and 150 km/h as maximum speed.
2g) Increase the speed by 40 km/h and set the gear to 2. Then use status().
2h) Set the color of the automobile to red.
2i) Save the color of the automobile in a variable myColor. Then check the color by using print(myColor).

Modules

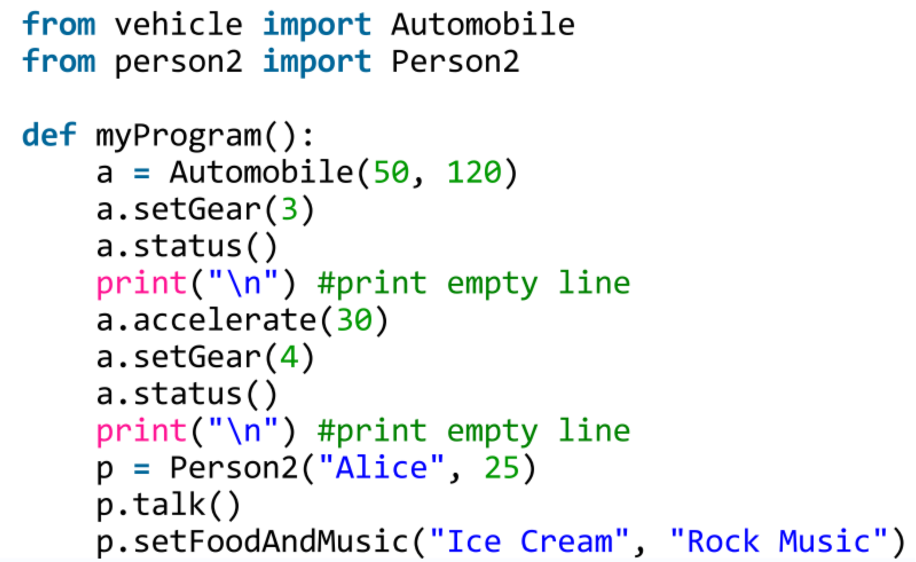


```
from moduleName import className
```

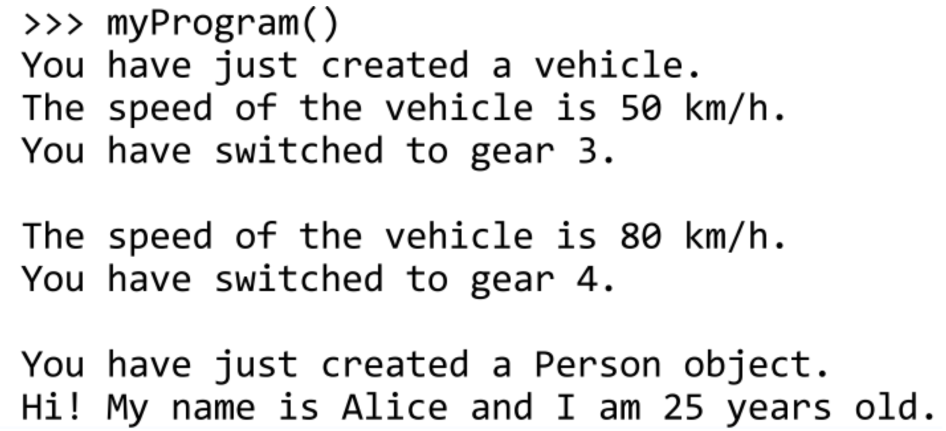
```
from vehicle import Automobile
```

```
from vehicle import *
```

Modules

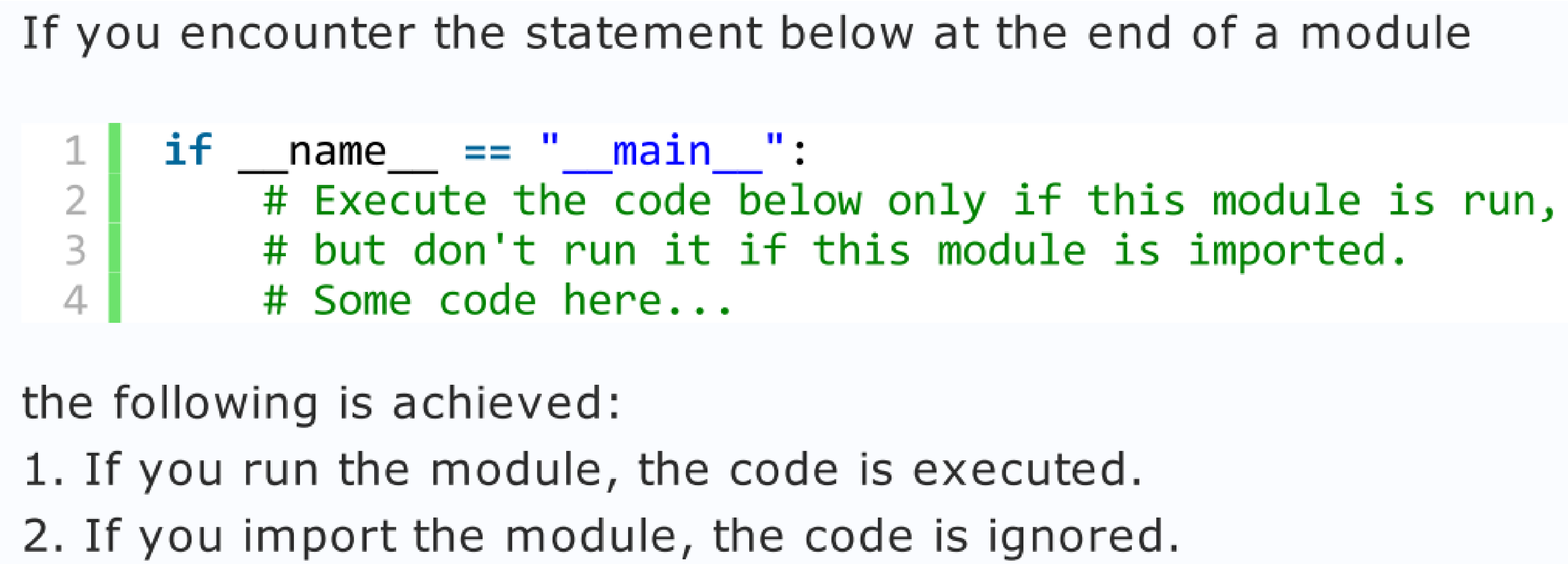


```
from vehicle import Automobile
from person2 import Person2

def myProgram():
    a = Automobile(50, 120)
    a.setGear(3)
    a.status()
    print("\n") #print empty line
    a.accelerate(30)
    a.setGear(4)
    a.status()
    print("\n") #print empty line
    p = Person2("Alice", 25)
    p.talk()
    p.setFoodAndMusic("Ice Cream", "Rock Music")
```



```
>>> myProgram()
You have just created a vehicle.
The speed of the vehicle is 50 km/h.
You have switched to gear 3.

The speed of the vehicle is 80 km/h.
You have switched to gear 4.

You have just created a Person object.
Hi! My name is Alice and I am 25 years old.
```

if __name__ == “__main__”

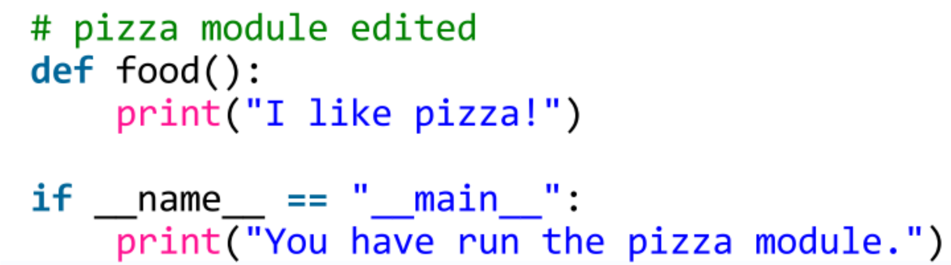


If you encounter the statement below at the end of a module

```python
if __name__ == "__main__":
    # Execute the code below only if this module is run,
    # but don't run it if this module is imported.
    # Some code here...
```

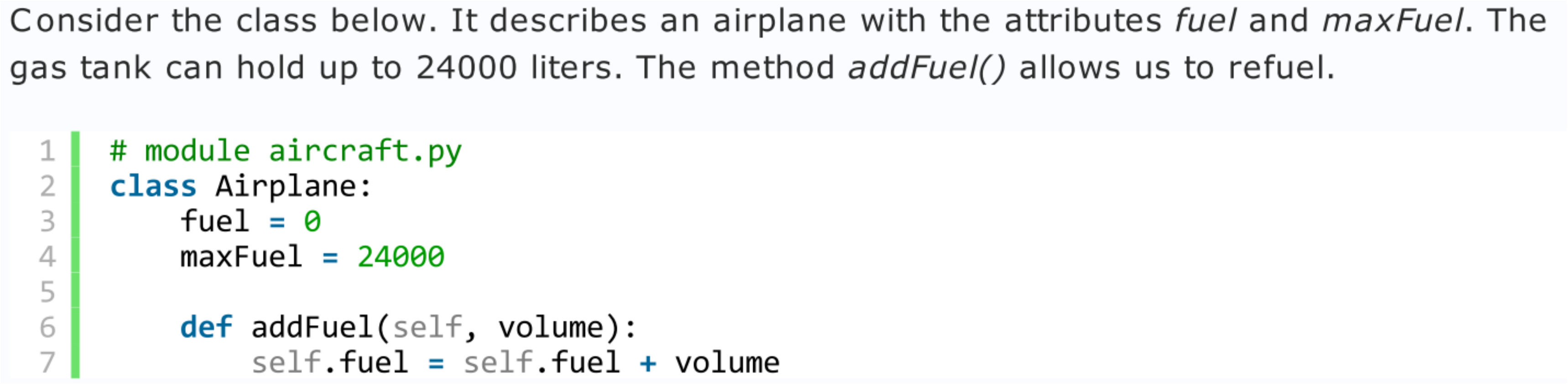
the following is achieved:

1. If you run the module, the code is executed.
2. If you import the module, the code is ignored.



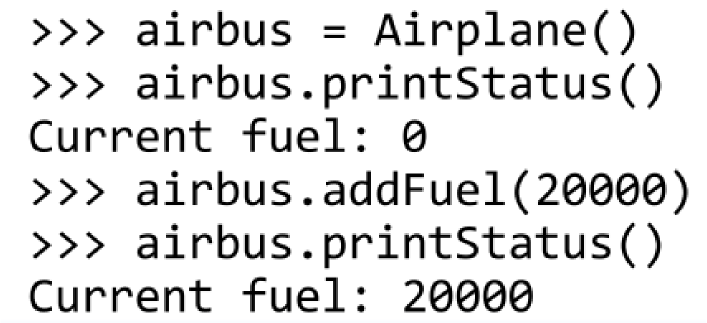
```python
# pizza module edited
def food():
    print("I like pizza!")

if __name__ == "__main__":
    print("You have run the pizza module.")
```

Private variables and methods



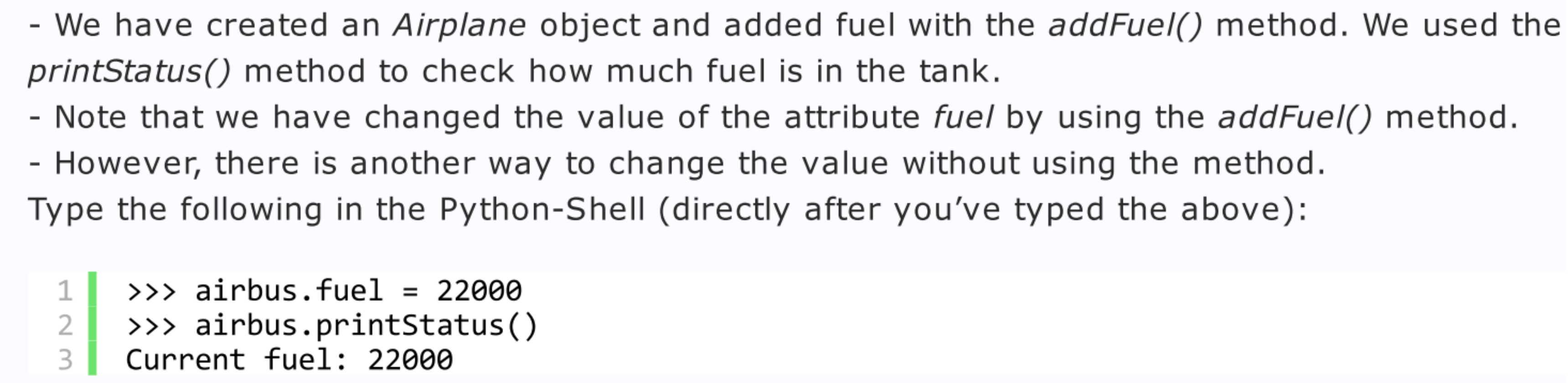
Consider the class below. It describes an airplane with the attributes *fuel* and *maxFuel*. The gas tank can hold up to 24000 liters. The method *addFuel()* allows us to refuel.

```
# module aircraft.py
class Airplane:
    fuel = 0
    maxFuel = 24000

    def addFuel(self, volume):
        self.fuel = self.fuel + volume
```



```
>>> airbus = Airplane()
>>> airbus.printStatus()
Current fuel: 0
>>> airbus.addFuel(20000)
>>> airbus.printStatus()
Current fuel: 20000
```

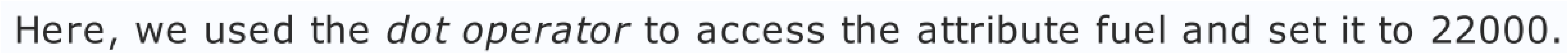
Adding fuel with “dot”



- We have created an *Airplane* object and added fuel with the *addFuel()* method. We used the *printStatus()* method to check how much fuel is in the tank.
- Note that we have changed the value of the attribute *fuel* by using the *addFuel()* method.
- However, there is another way to change the value without using the method.

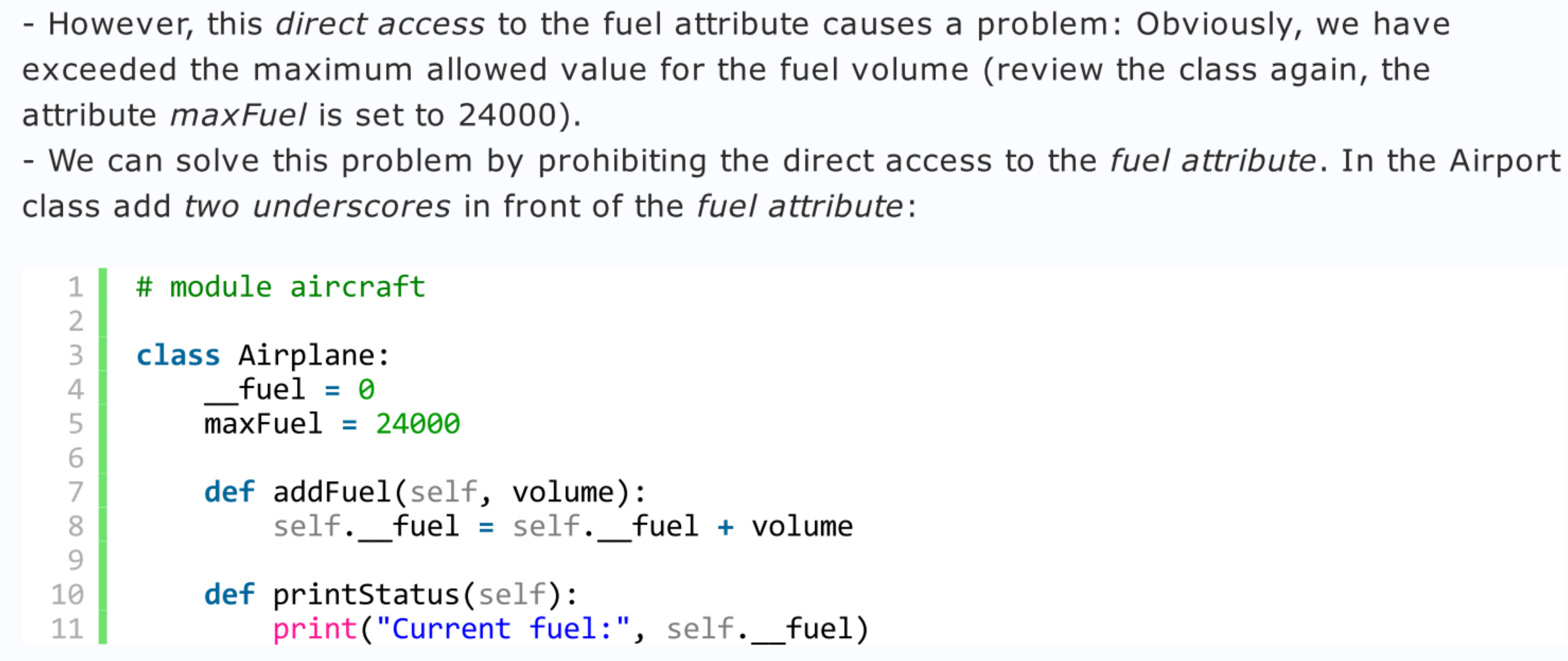
Type the following in the Python-Shell (directly after you’ve typed the above):

```
>>> airbus.fuel = 22000
>>> airbus.printStatus()
Current fuel: 22000
```



Here, we used the *dot operator* to access the attribute fuel and set it to 22000.

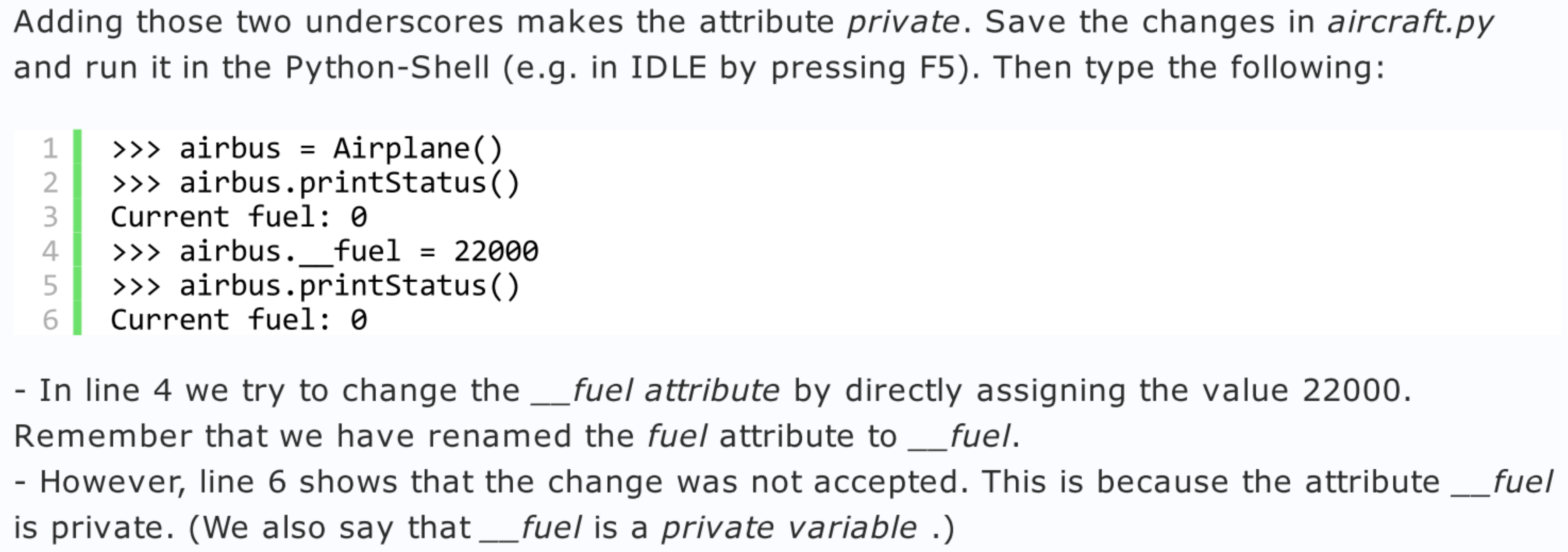
A problem with “dot”



- However, this *direct access* to the fuel attribute causes a problem: Obviously, we have exceeded the maximum allowed value for the fuel volume (review the class again, the attribute *maxFuel* is set to 24000).
- We can solve this problem by prohibiting the direct access to the *fuel attribute*. In the Airport class add *two underscores* in front of the *fuel attribute*:

```python
# module aircraft

class Airplane:
    __fuel = 0
    maxFuel = 24000

    def addFuel(self, volume):
        self.__fuel = self.__fuel + volume

    def printStatus(self):
        print("Current fuel:", self.__fuel)
```

Use private attributes



Adding those two underscores makes the attribute *private*. Save the changes in *aircraft.py* and run it in the Python-Shell (e.g. in IDLE by pressing F5). Then type the following:

```
1  >>> airbus = Airplane()
2  >>> airbus.printStatus()
3  Current fuel: 0
4  >>> airbus.__fuel = 22000
5  >>> airbus.printStatus()
6  Current fuel: 0
```

- In line 4 we try to change the __*fuel attribute* by directly assigning the value 22000. Remember that we have renamed the *fuel* attribute to __*fuel*.
- However, line 6 shows that the change was not accepted. This is because the attribute __*fuel* is private. (We also say that __*fuel* is a *private variable* .)