Class in Python

Stan Kozin
Ilya Stepanko

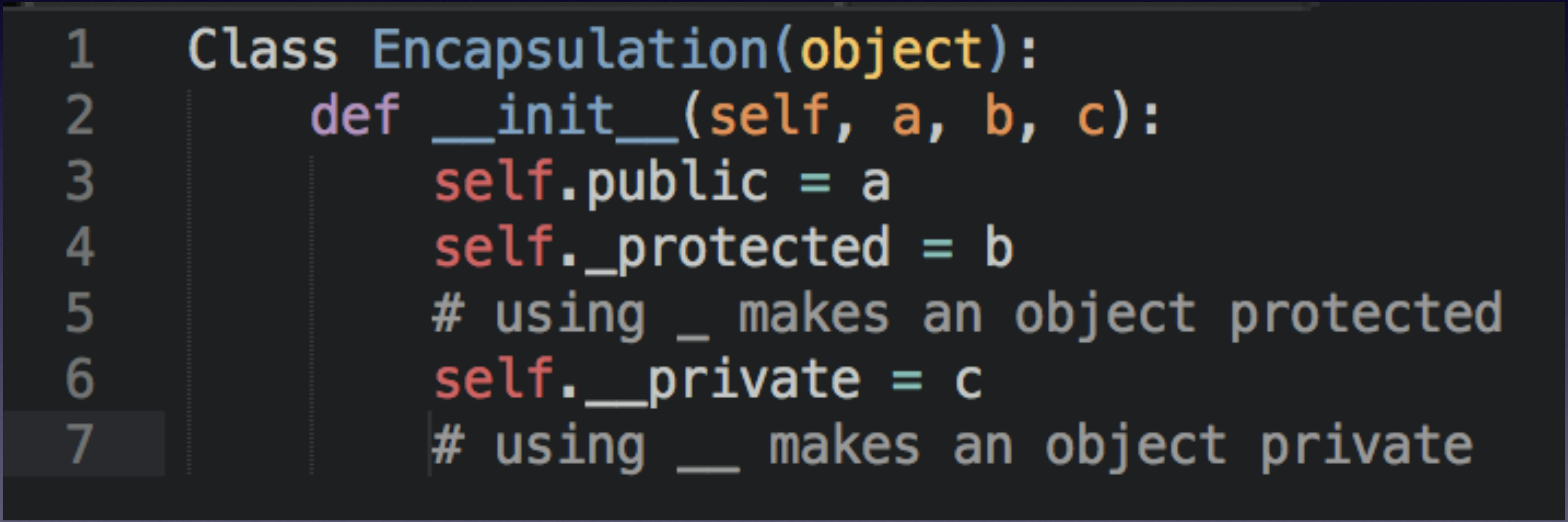
OOP for Beginners

Object-oriented programming is a style of coding that allows developers to group similar tasks into classes. This helps keep code following the tenet "don't repeat yourself" (DRY) and easy-to-maintain.

OOP Features

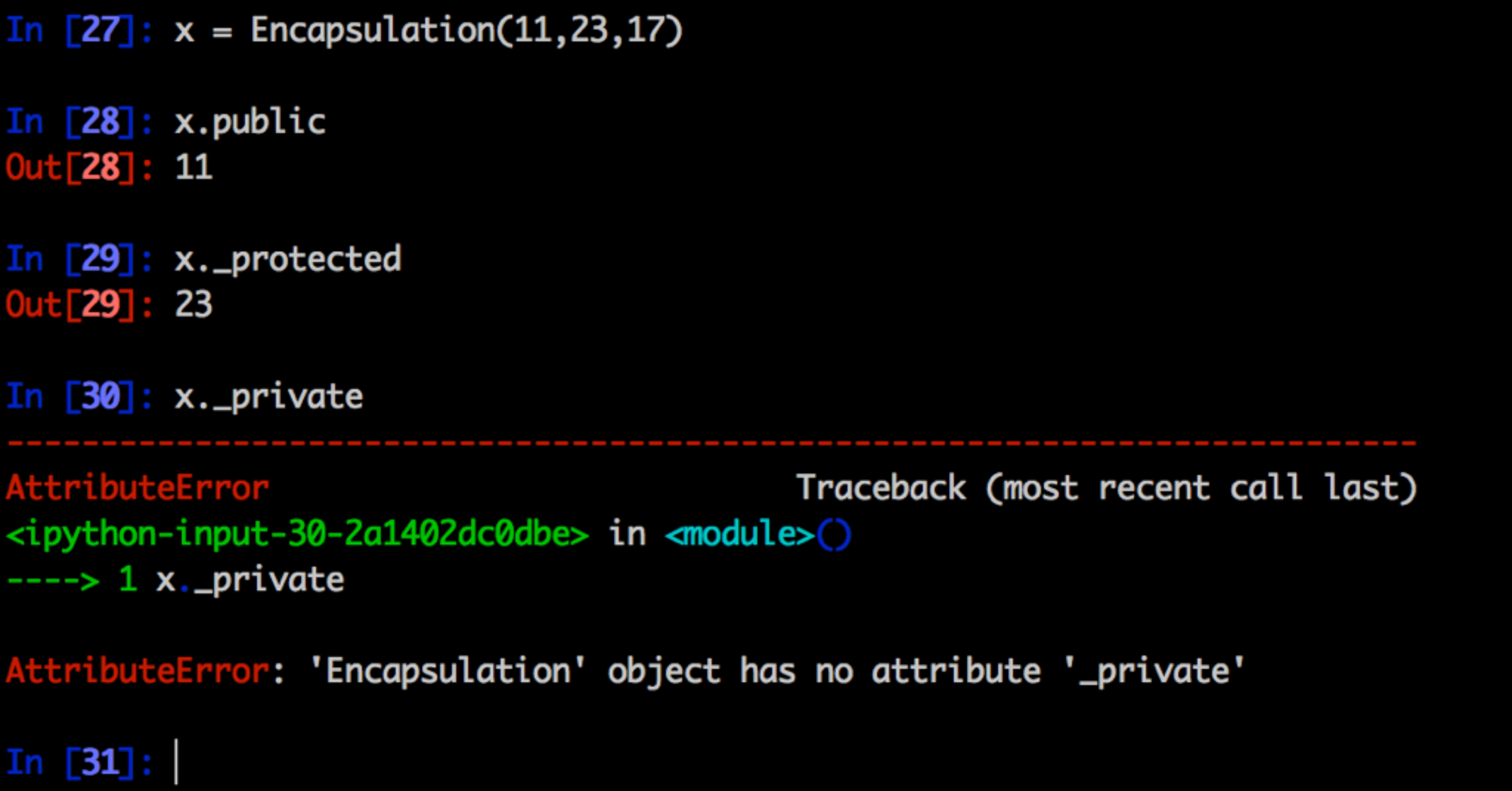
- Encapsulation
- Abstraction
- Inheritance

Encapsulation



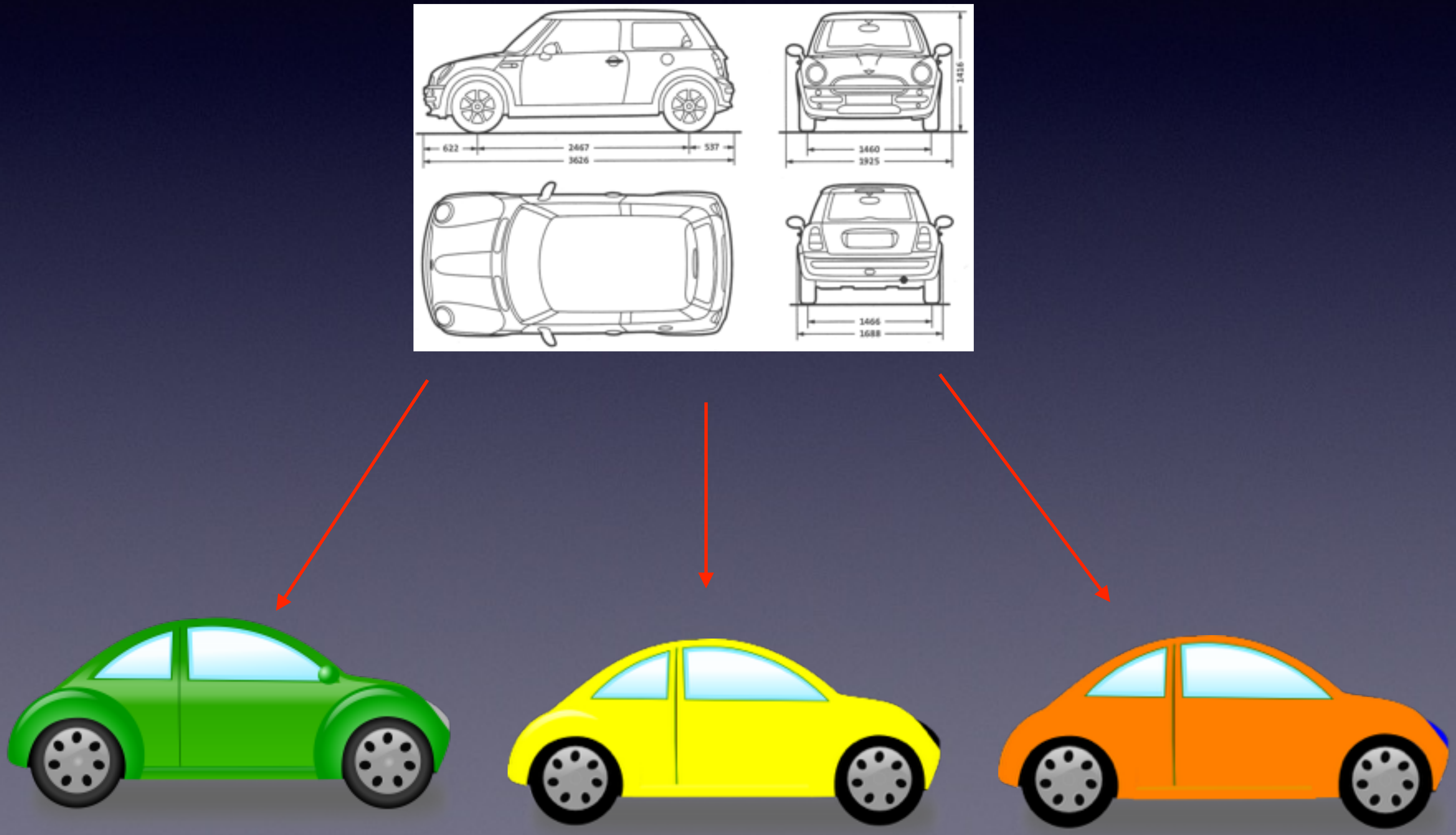
```
Class Encapsulation(object):
    def __init__(self, a, b, c):
        self.public = a
        self._protected = b
        # using _ makes an object protected
        self.__private = c
        # using __ makes an object private
```

Encapsulation

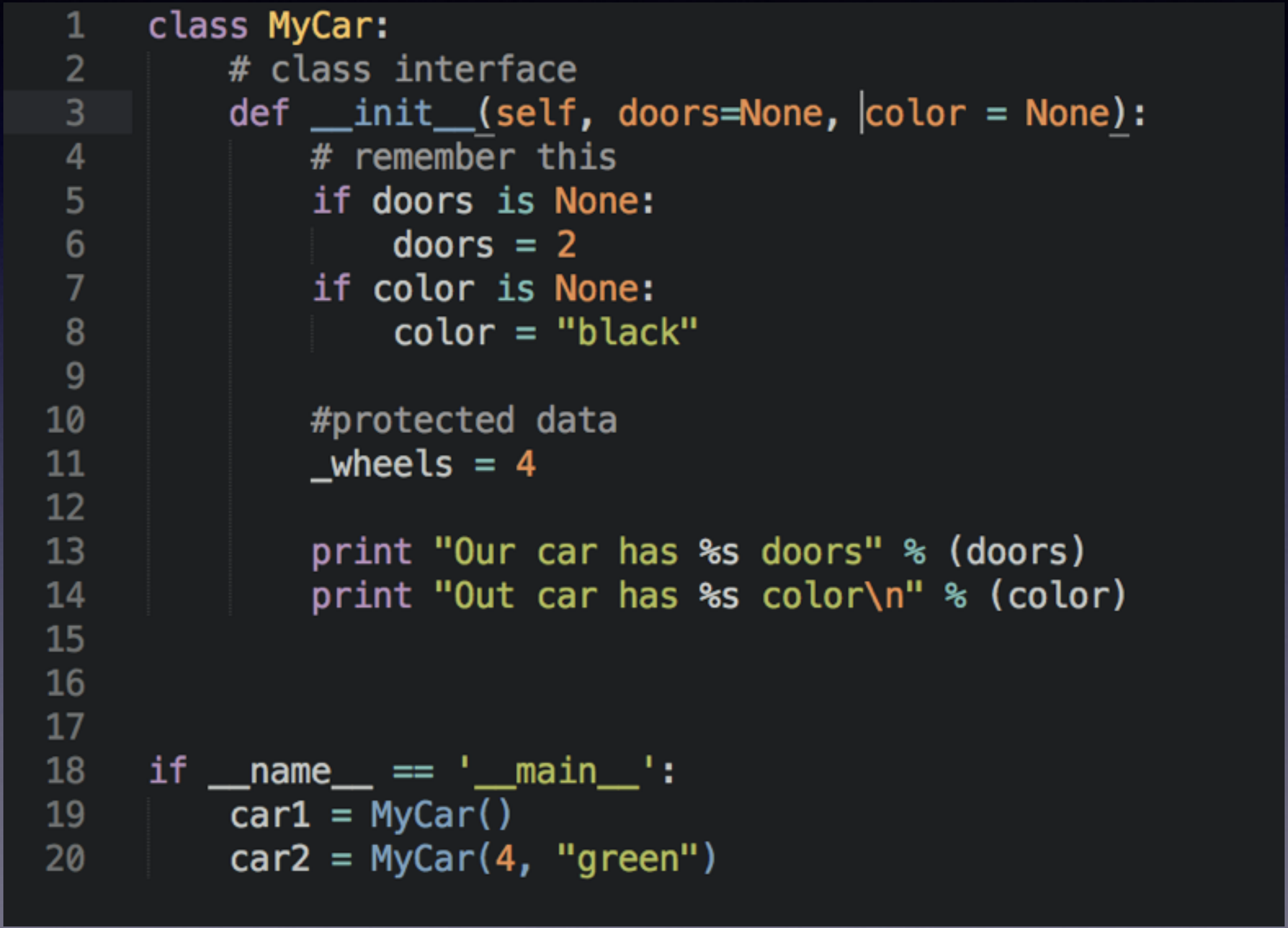


```
In [27]: x = Encapsulation(11,23,17)

In [28]: x.public
Out[28]: 11

In [29]: x._protected
Out[29]: 23

In [30]: x._private
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-2a1402dc0dbe> in <module>()
----> 1 x._private

AttributeError: 'Encapsulation' object has no attribute '_private'

In [31]: |
```

Class and Object

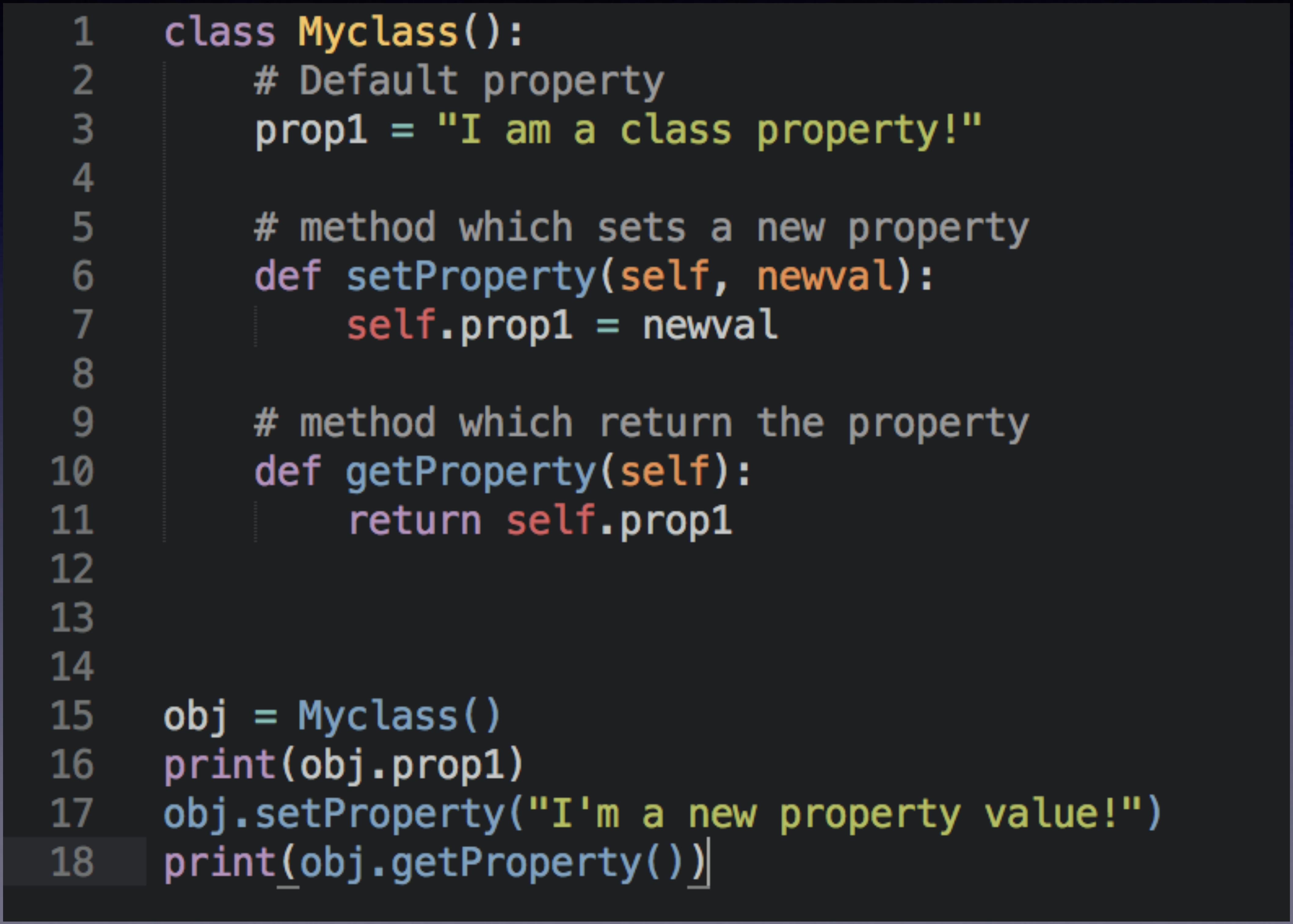


Structuring Classes



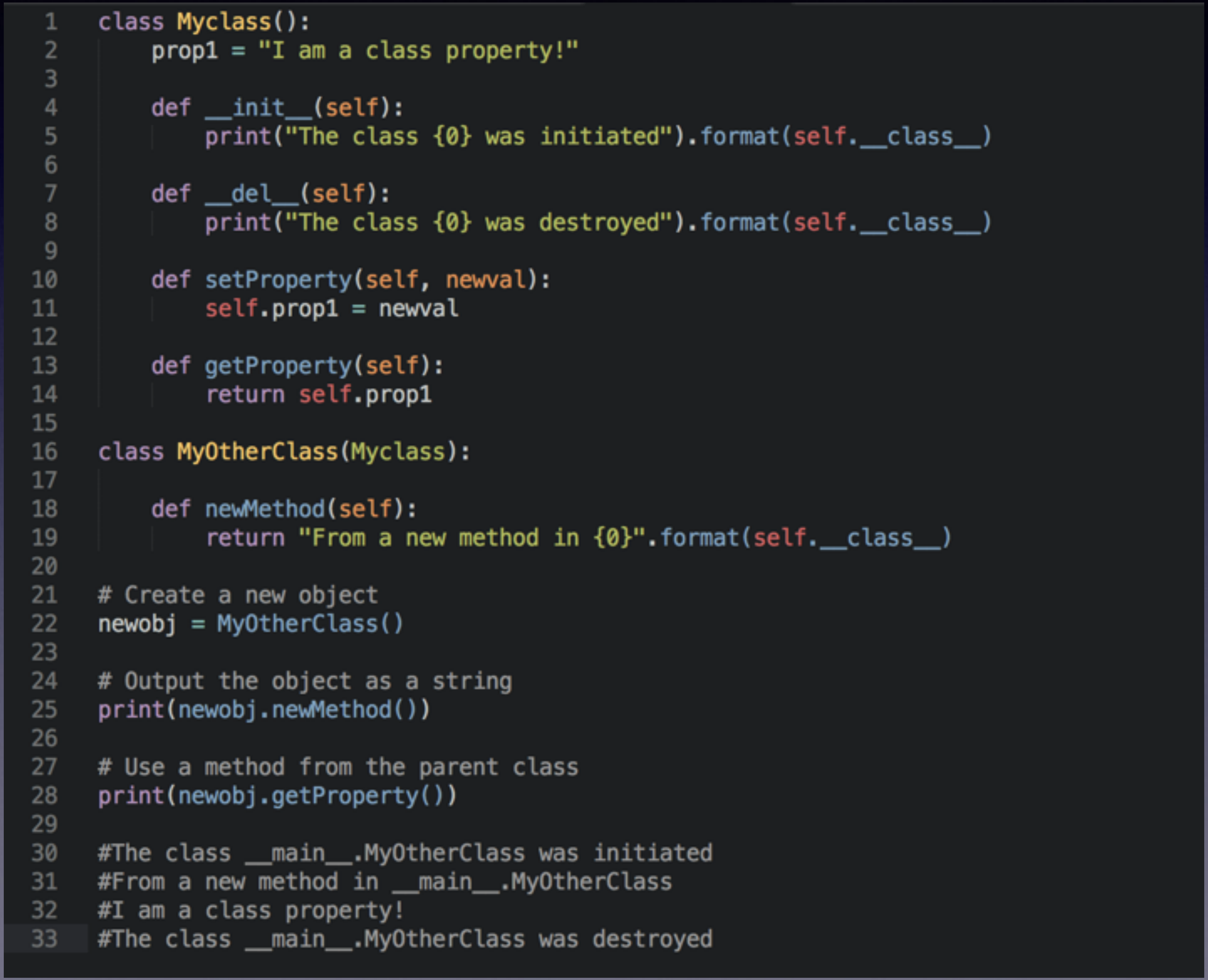
```
class MyCar:
    # class interface
    def __init__(self, doors=None, color = None):
        # remember this
        if doors is None:
            doors = 2
        if color is None:
            color = "black"

        #protected data
        _wheels = 4

        print "Our car has %s doors" % (doors)
        print "Out car has %s color\n" % (color)



if __name__ == '__main__':
    car1 = MyCar()
    car2 = MyCar(4, "green")
```

Defining Class Methods



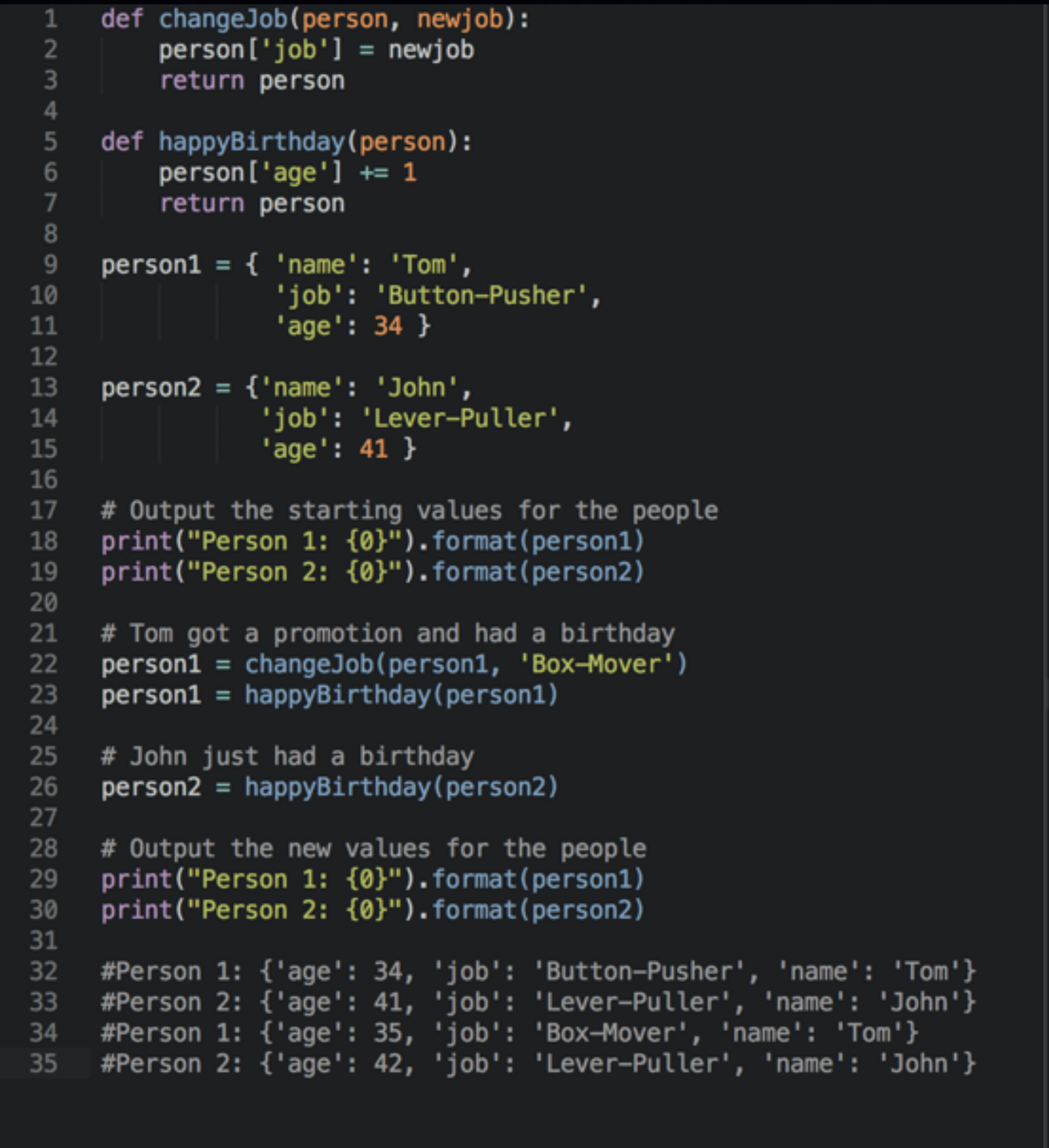
```python
class Myclass():
    # Default property
    prop1 = "I am a class property!"

    # method which sets a new property
    def setProperty(self, newval):
        self.prop1 = newval

    # method which return the property
    def getProperty(self):
        return self.prop1



obj = Myclass()
print(obj.prop1)
obj.setProperty("I'm a new property value!")
print(obj.getProperty())
```

Using Class Inheritance

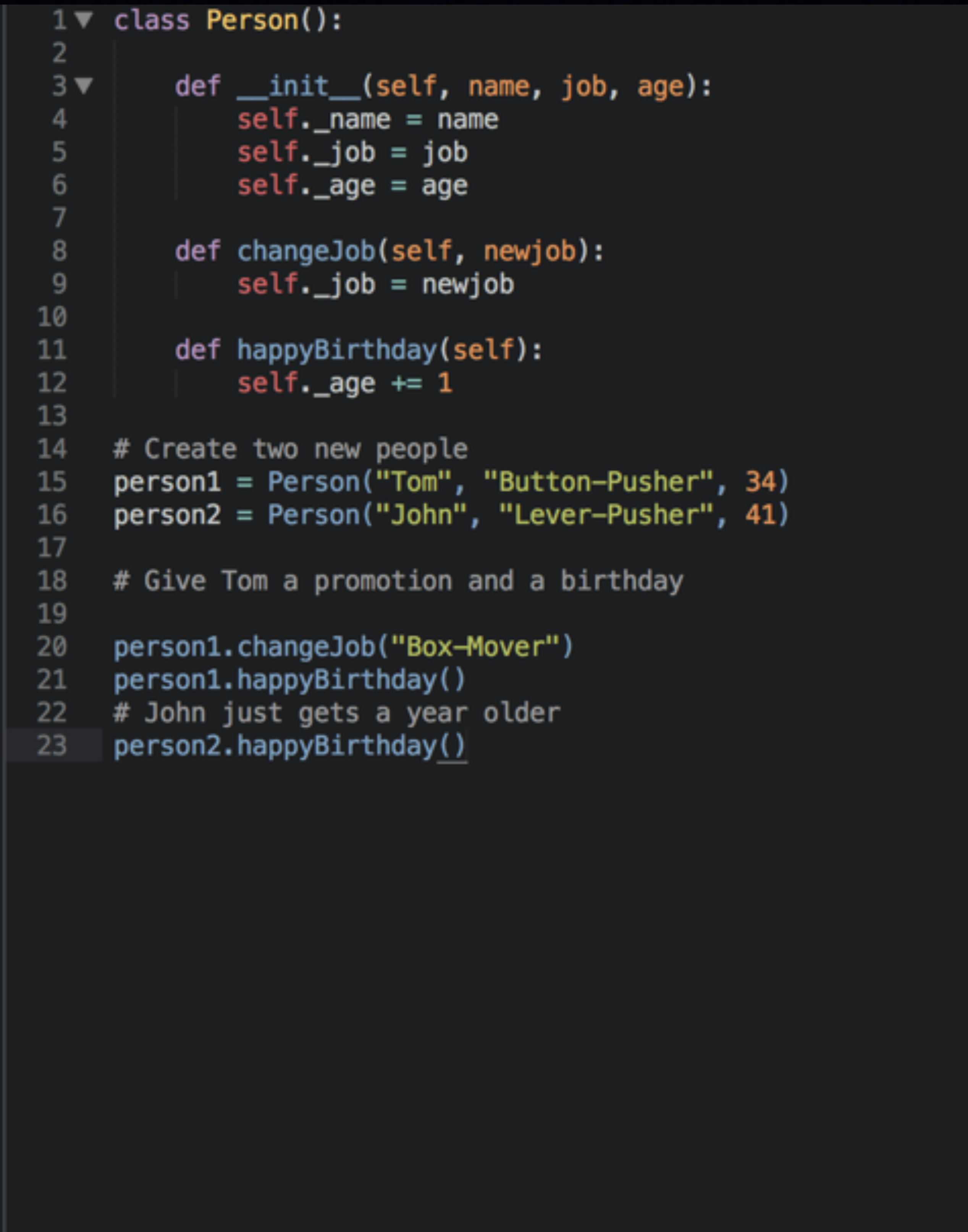


```python
class Myclass():
    prop1 = "I am a class property!"

    def __init__(self):
        print("The class {0} was initiated").format(self.__class__)

    def __del__(self):
        print("The class {0} was destroyed").format(self.__class__)

    def setProperty(self, newval):
        self.prop1 = newval

    def getProperty(self):
        return self.prop1

class MyOtherClass(Myclass):

    def newMethod(self):
        return "From a new method in {0}".format(self.__class__)

# Create a new object
newobj = MyOtherClass()

# Output the object as a string
print(newobj.newMethod())

# Use a method from the parent class
print(newobj.getProperty())

#The class __main__.MyOtherClass was initiated
#From a new method in __main__.MyOtherClass
#I am a class property!
#The class __main__.MyOtherClass was destroyed
```

Functional vs OOP



```python
def changeJob(person, newjob):
    person['job'] = newjob
    return person

def happyBirthday(person):
    person['age'] += 1
    return person

person1 = { 'name': 'Tom',
            'job': 'Button-Pusher',
            'age': 34 }

person2 = {'name': 'John',
           'job': 'Lever-Puller',
           'age': 41 }

# Output the starting values for the people
print("Person 1: {0}").format(person1)
print("Person 2: {0}").format(person2)

# Tom got a promotion and had a birthday
person1 = changeJob(person1, 'Box-Mover')
person1 = happyBirthday(person1)

# John just had a birthday
person2 = happyBirthday(person2)

# Output the new values for the people
print("Person 1: {0}").format(person1)
print("Person 2: {0}").format(person2)

#Person 1: {'age': 34, 'job': 'Button-Pusher', 'name': 'Tom'}
#Person 2: {'age': 41, 'job': 'Lever-Puller', 'name': 'John'}
#Person 1: {'age': 35, 'job': 'Box-Mover', 'name': 'Tom'}
#Person 2: {'age': 42, 'job': 'Lever-Puller', 'name': 'John'}
```



```python
class Person():

    def __init__(self, name, job, age):
        self._name = name
        self._job = job
        self._age = age

    def changeJob(self, newjob):
        self._job = newjob

    def happyBirthday(self):
        self._age += 1

# Create two new people
person1 = Person("Tom", "Button-Pusher", 34)
person2 = Person("John", "Lever-Pusher", 41)

# Give Tom a promotion and a birthday

person1.changeJob("Box-Mover")
person1.happyBirthday()
# John just gets a year older
person2.happyBirthday()
```

Homework

- Read and learn about OOP - http://en.wikipedia.org/wiki/Object-oriented_programming
- Read about classes in Python - https://docs.python.org/2/tutorial/classes.html
- Read and learn about __init__, __del__ and other