Object Oriented Programming in Python: Inheritance

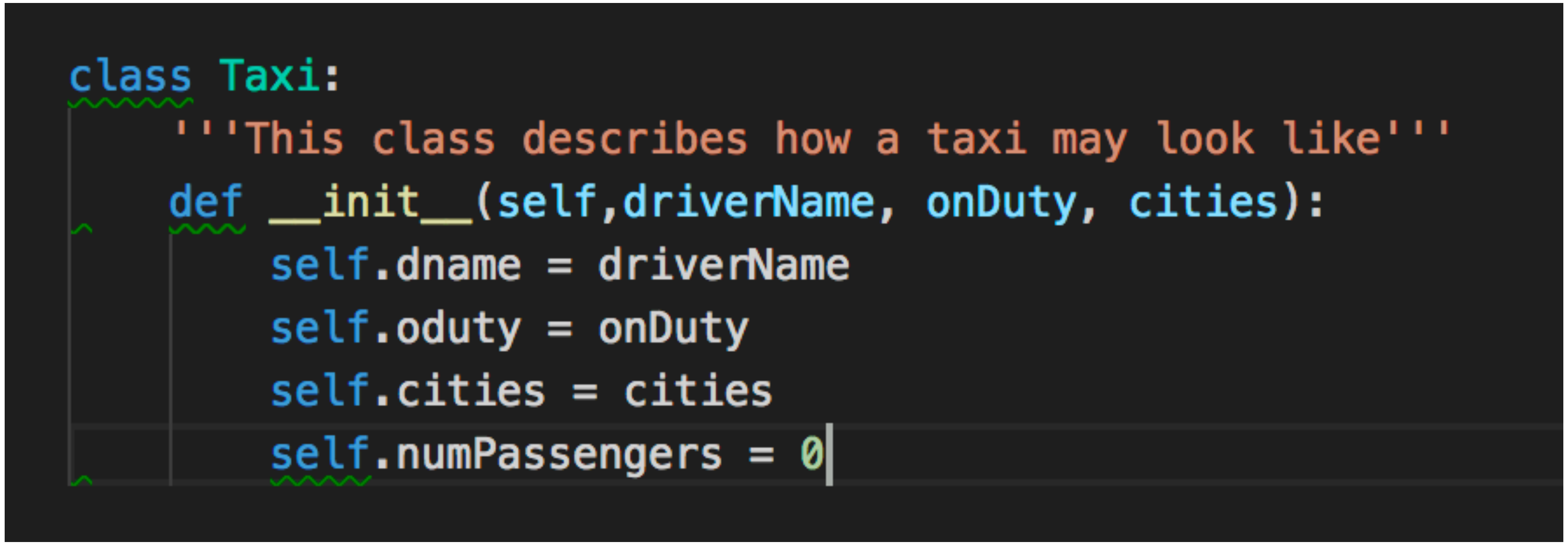
Aleksander Fabijan

Today’s Goals

1. Provide an introduction to inheritance in OOP.
    - **Why** and **when** should we inherit from other objects?
    - **How** do we inherit from objects in Python?

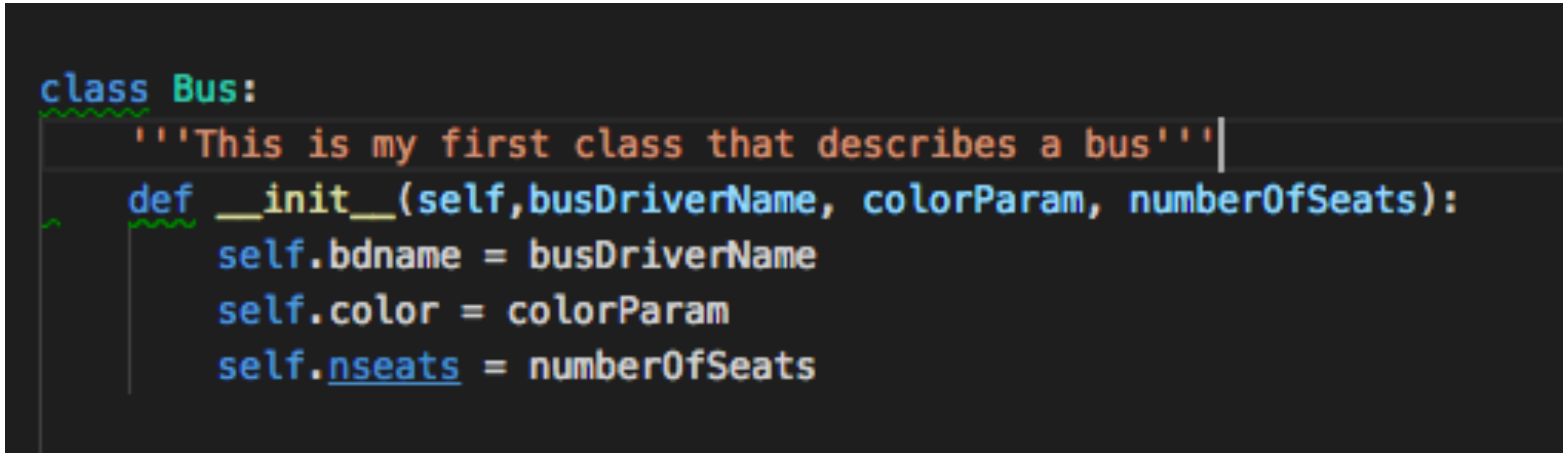
2. Provide an introduction to method overriding.

From Last Time



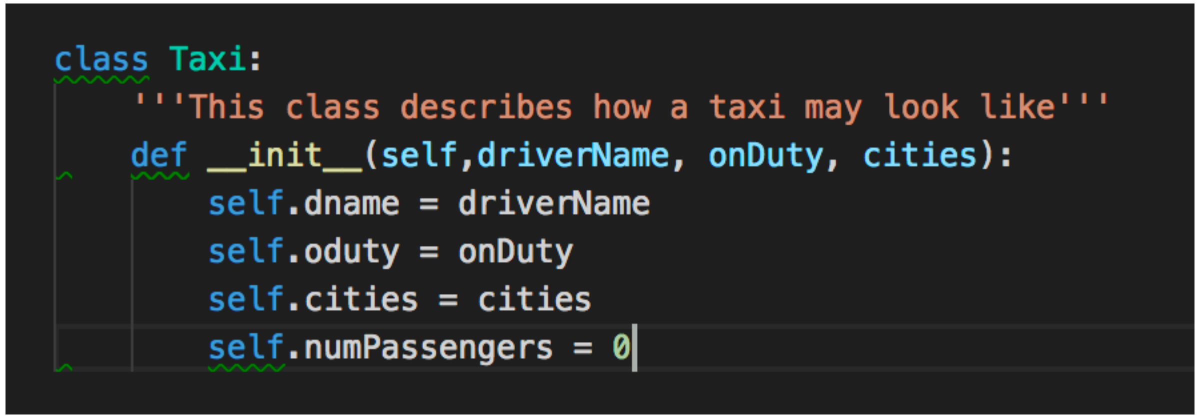
```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

From Last Time (cnt.)

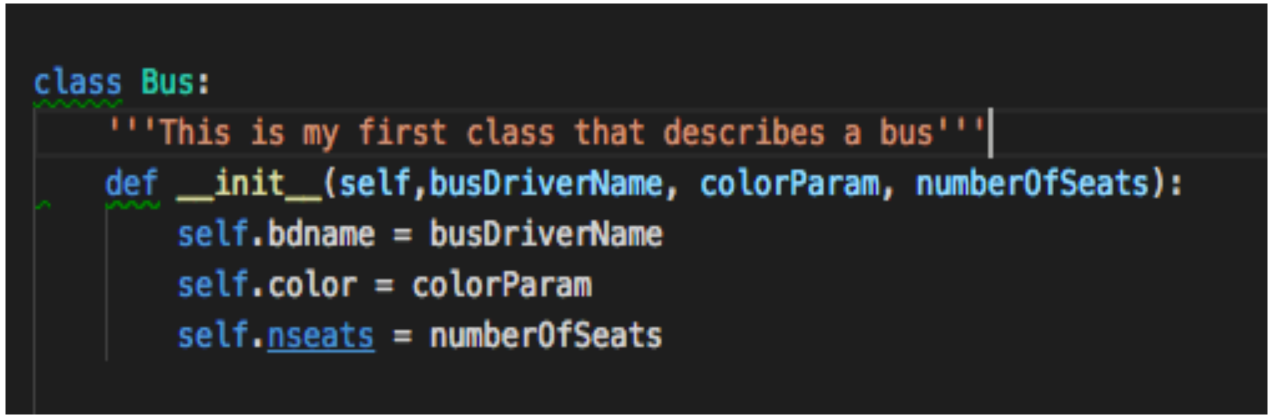


```python
class Bus:
    '''This is my first class that describes a bus'''
    def __init__(self,busDriverName, colorParam, numberOfSeats):
        self.bdname = busDriverName
        self.color = colorParam
        self.nseats = numberOfSeats
```

Comparing Bus & Taxi

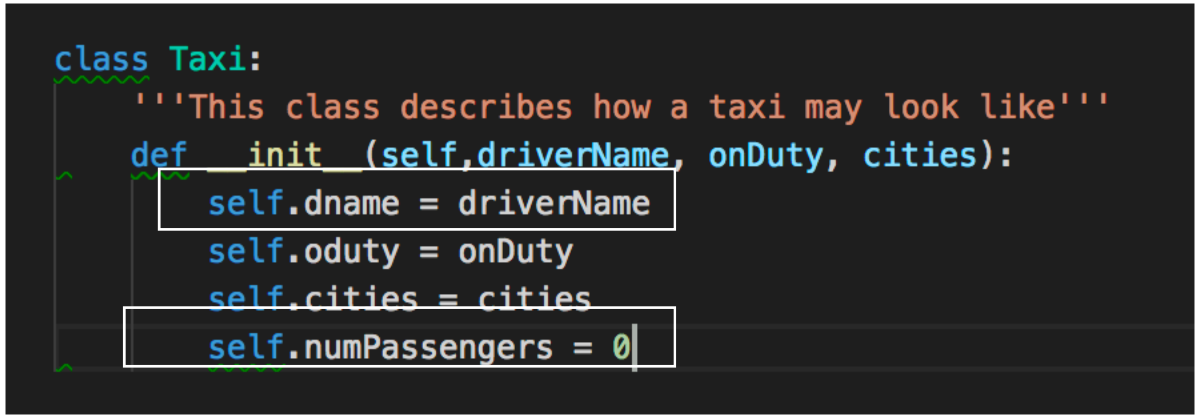


```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

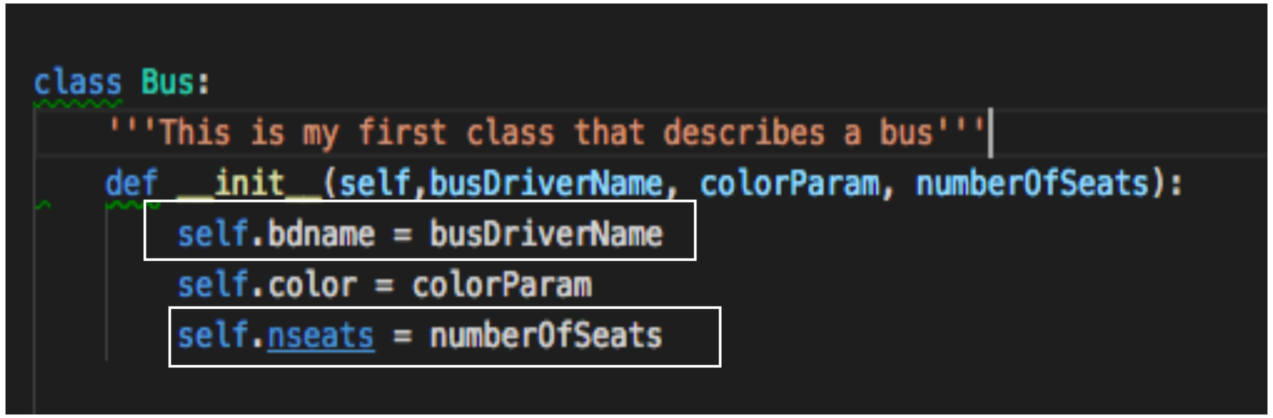


```
class Bus:
    '''This is my first class that describes a bus'''
    def __init__(self,busDriverName, colorParam, numberOfSeats):
        self.bdname = busDriverName
        self.color = colorParam
        self.nseats = numberOfSeats
```

Comparing Bus & Taxi



```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```



```
class Bus:
    '''This is my first class that describes a bus'''
    def __init__(self,busDriverName, colorParam, numberOfSeats):
        self.bdname = busDriverName
        self.color = colorParam
        self.nseats = numberOfSeats
```

Classes share similar variables

Inheritance

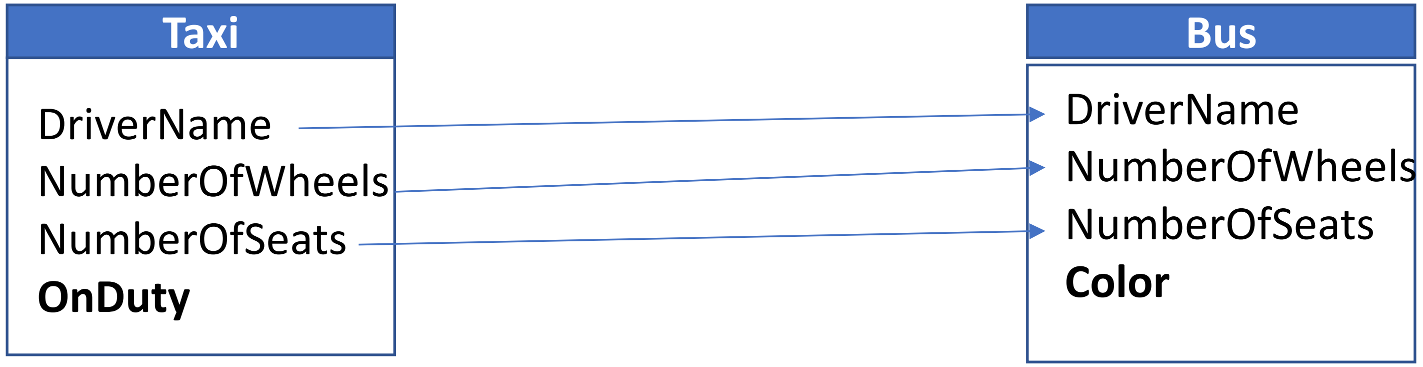
- Inheritance simplifies our code through reuse of the code that has been already written.
    - **Think about the Taxi and Bus, and what they have in common.**

- Inheritance is a relation between a **parent class** (e.g. *Vehicle*) and **children classes** (e.g. Taxi, Bus, Truck**,** etc.)

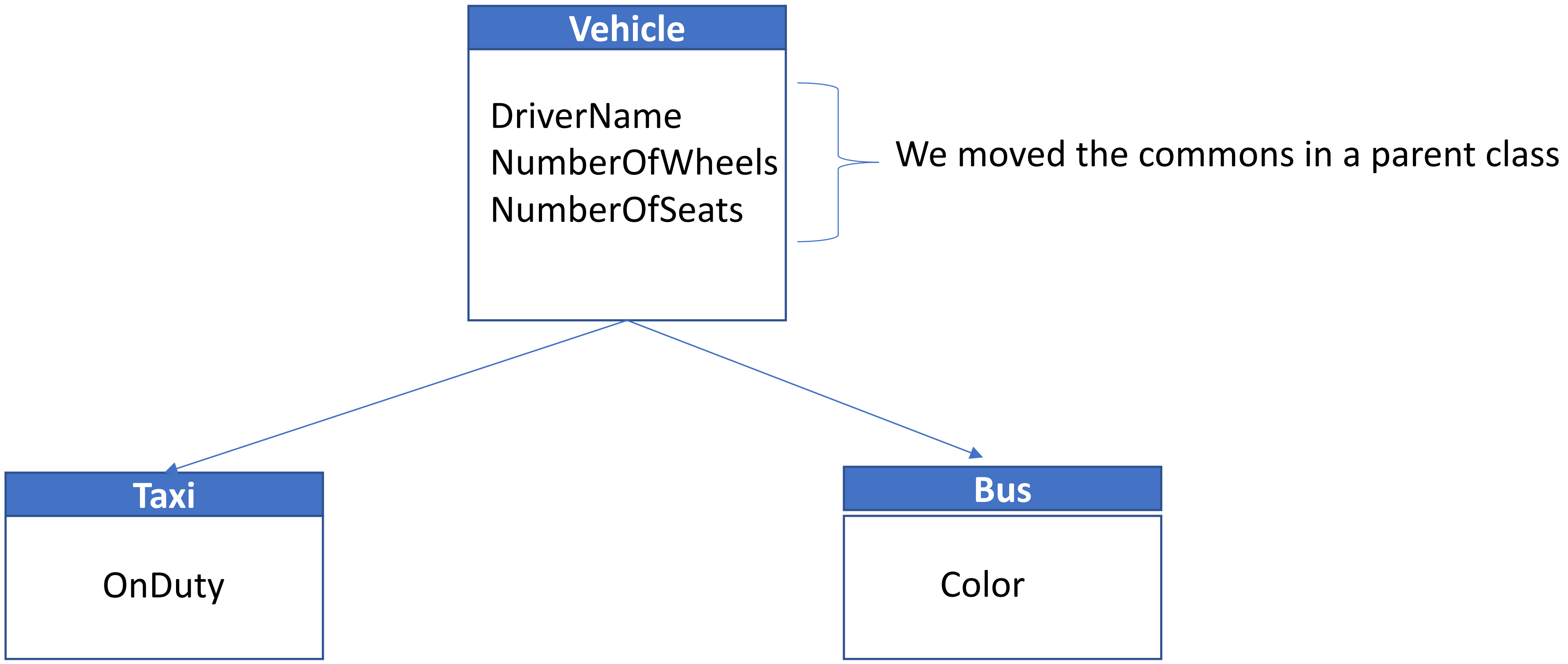
- A class inherits **attributes** and **behavior** methods from its parent classes.

Taxi
DriverName
NumberOfWheels
NumberOfSeats
OnDuty

Bus
DriverName
NumberOfWheels
NumberOfSeats
Color



Wait, we wrote the same three lines of code in both classes?
There must be a better way!!!



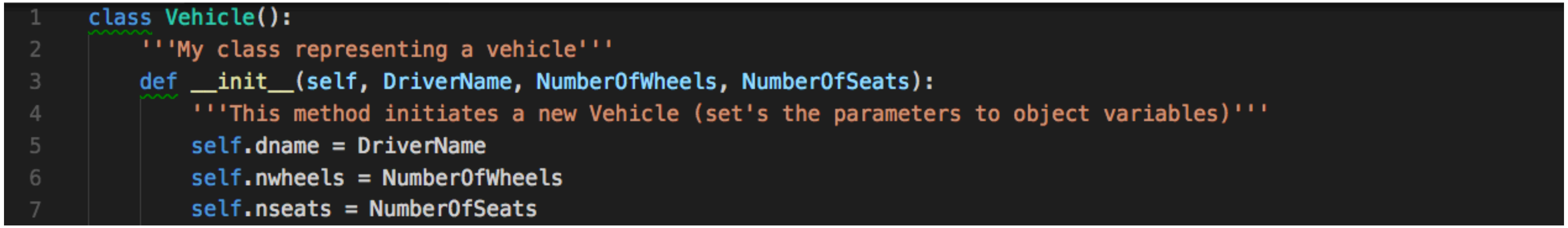
The child classes only keep the attributes and methods relevant to them

OOP Inheritance in Python

1. Create a parent class (e.g. Vehicle) with the **common attributes** and **common methods**.

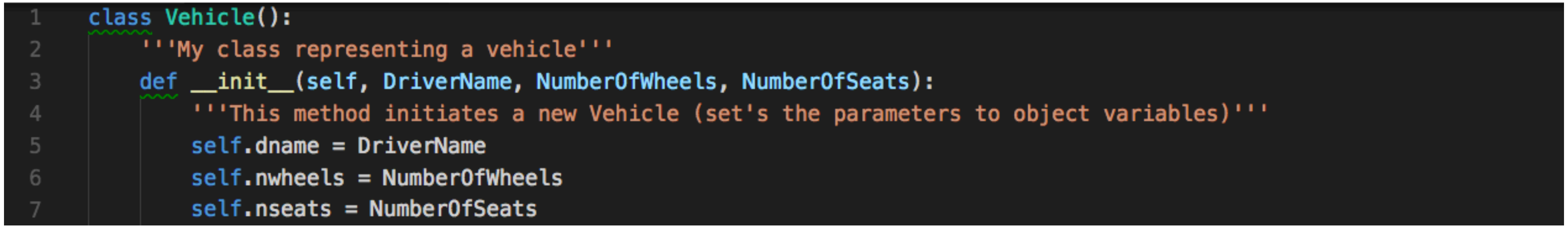
2. Create child classes (e.g. Bus and Taxi) with the **extended attributes** and **extended methods**.
   - Pass the class definition to the child (e.g. **Class Bus(Vehicle):** …)
   - Use the parent attributes and methods through **super().**

In Python code

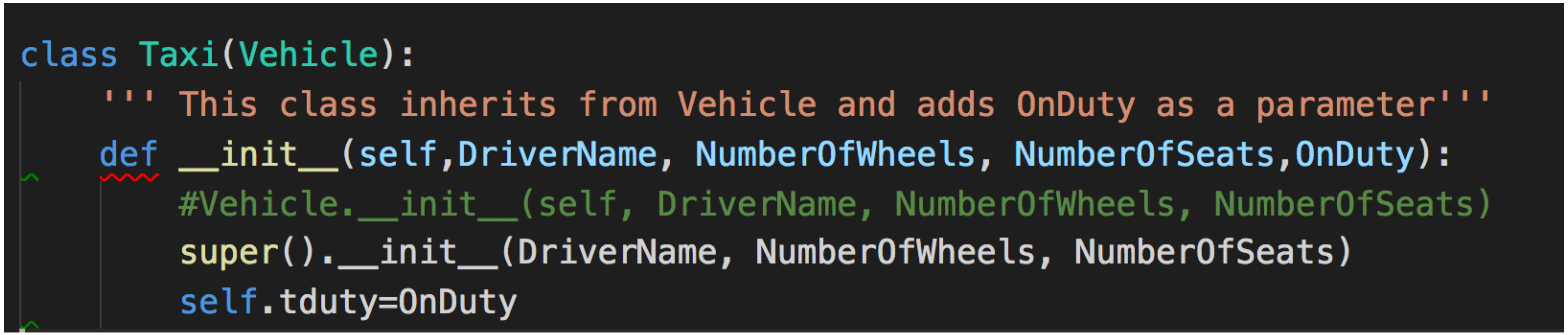


```python
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```

In Python code

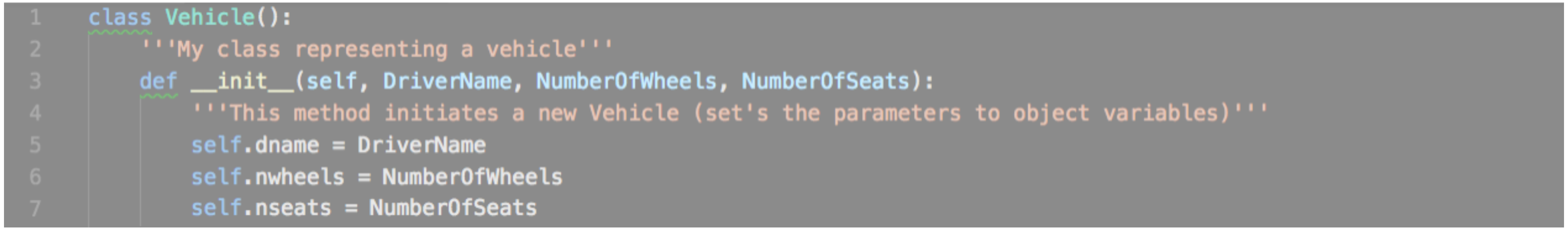


```python
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```

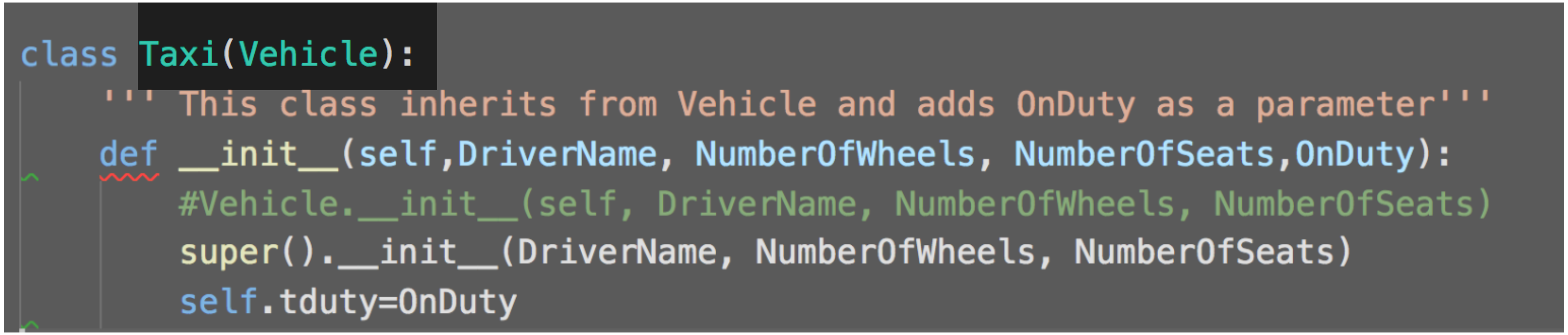


```python
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        #Vehicle.__init__(self, DriverName, NumberOfWheels, NumberOfSeats)
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
```

In Python code

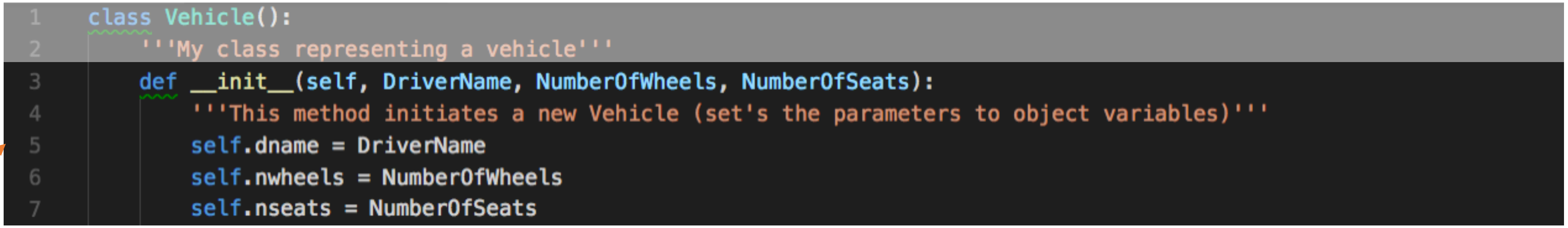


```python
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```

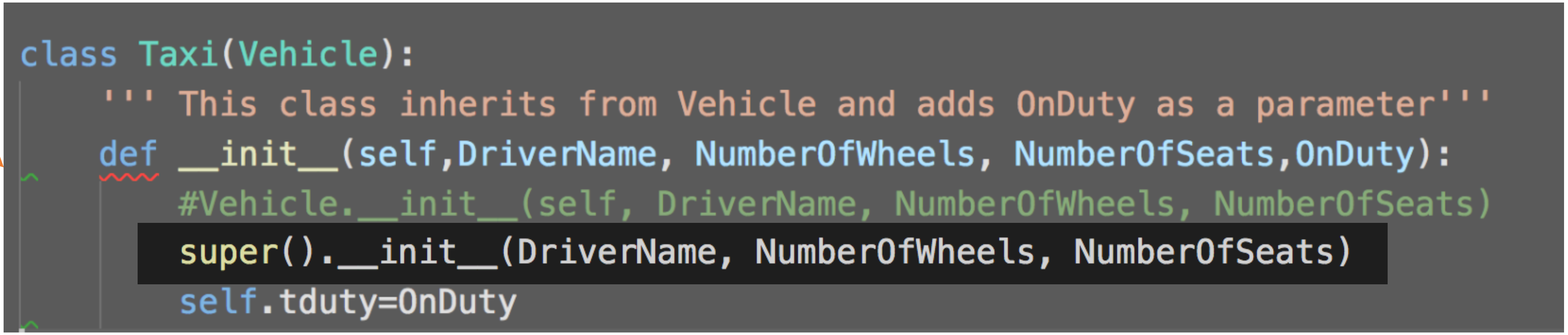


```python
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        #Vehicle.__init__(self, DriverName, NumberOfWheels, NumberOfSeats)
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
```

In Python code

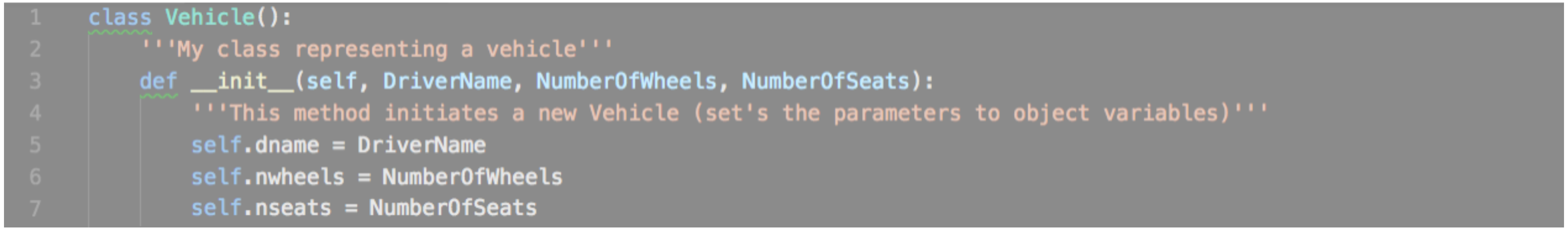


```python
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```

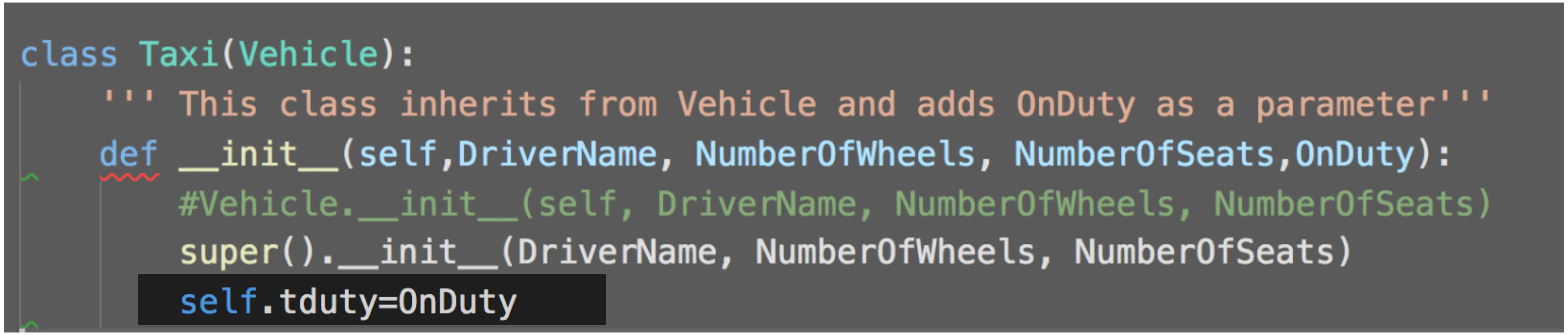


```python
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        #Vehicle.__init__(self, DriverName, NumberOfWheels, NumberOfSeats)
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
```

In Python code



```python
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```



```python
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        #Vehicle.__init__(self, DriverName, NumberOfWheels, NumberOfSeats)
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
```

Exercise time!

Model the following problem in Python code:

- Frida Jacobsson is a **student** at MAH. Her Skype nickname is frida96.
- Aleksander Fabijan is a **researcher** at MAH. He teaches DA712 and DA374.
- They are both **Humans**.

Exercise time!

Model the following problem in Python code:

- Frida Jacobsson is a **Student** at MAH. Her Skype nickname is frida96.
- Aleksander is a **Researcher** at MAH. He teaches DA712 and DA374.
- They are both **Humans**.

Suggestion:

1) Create a **class Human** that initiates a new human with a name.
2) Next, **create two classes** (e.g. ***Student*** and ***Researcher***) that <u>inherit from human</u>,
3) Finally, add the skype nickname and the list of courses to the new classes.

Code snippets for help:

- class Taxi(Vehicle):   #creates a child class from Human
- super().__init__(name, lastname)   #calls the parent's __init__method

LC

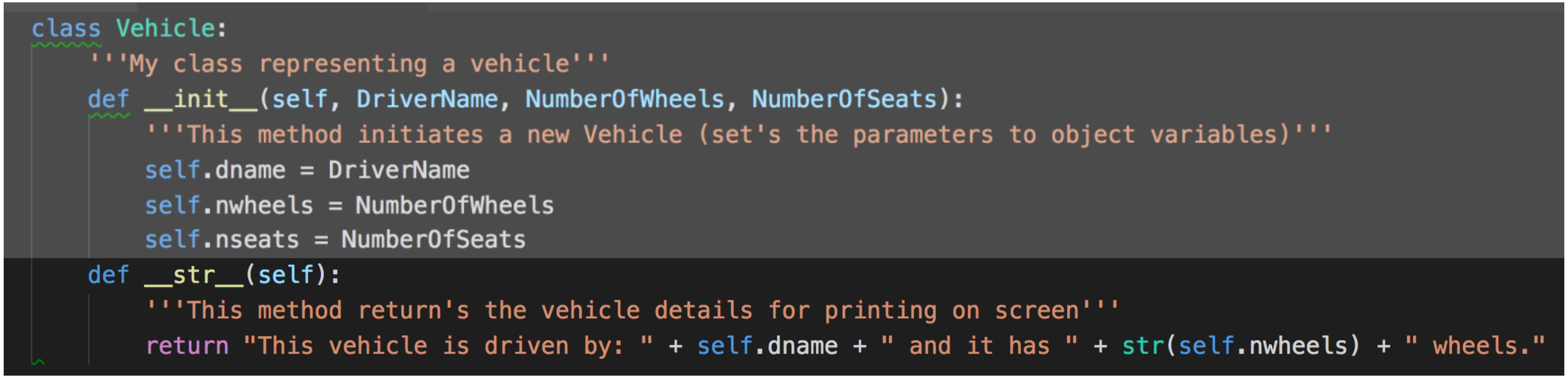
Method Overriding

- Method overriding is an object-oriented programming feature that allows a subclass to provide a different implementation of a method that is already defined by its superclass or by one of its superclasses.

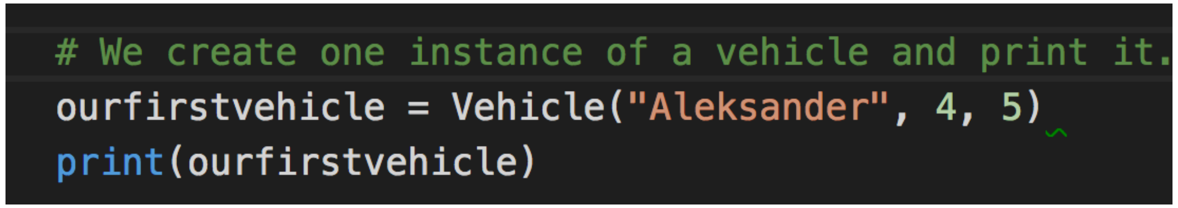
- __init__ in the child class (e.g. Taxi) overrides the __init__ method from the parent class.

Example of overriding __str__

Let’s add a __str__ method that nicely prints our Vehicle details on the screen.

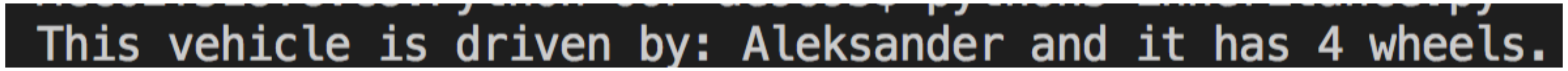


```python
class Vehicle:
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
    def __str__(self):
        '''This method return's the vehicle details for printing on screen'''
        return "This vehicle is driven by: " + self.dname + " and it has " + str(self.nwheels) + " wheels."
```



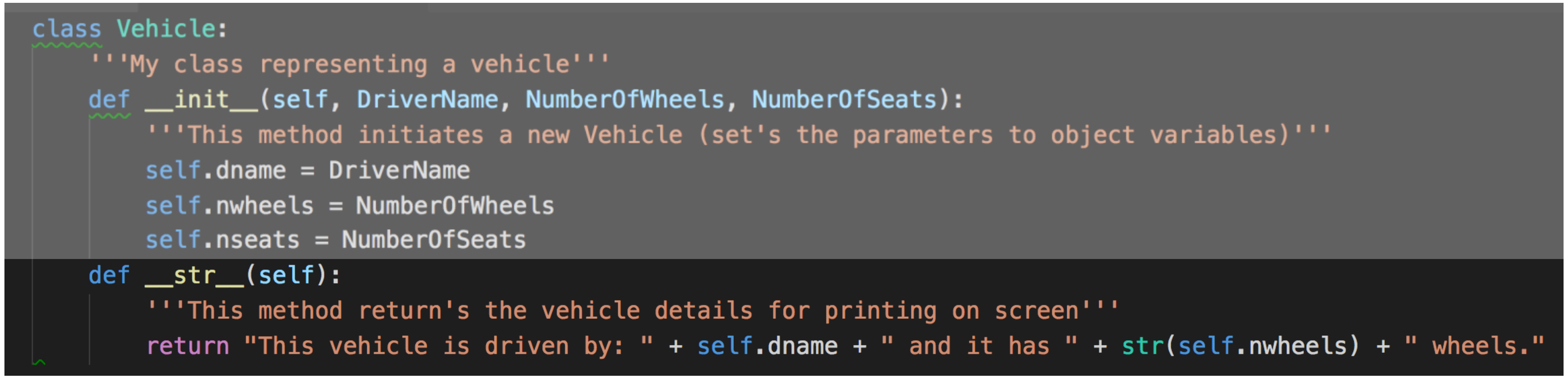
```python
# We create one instance of a vehicle and print it.
ourfirstvehicle = Vehicle("Aleksander", 4, 5)
print(ourfirstvehicle)
```

Output:

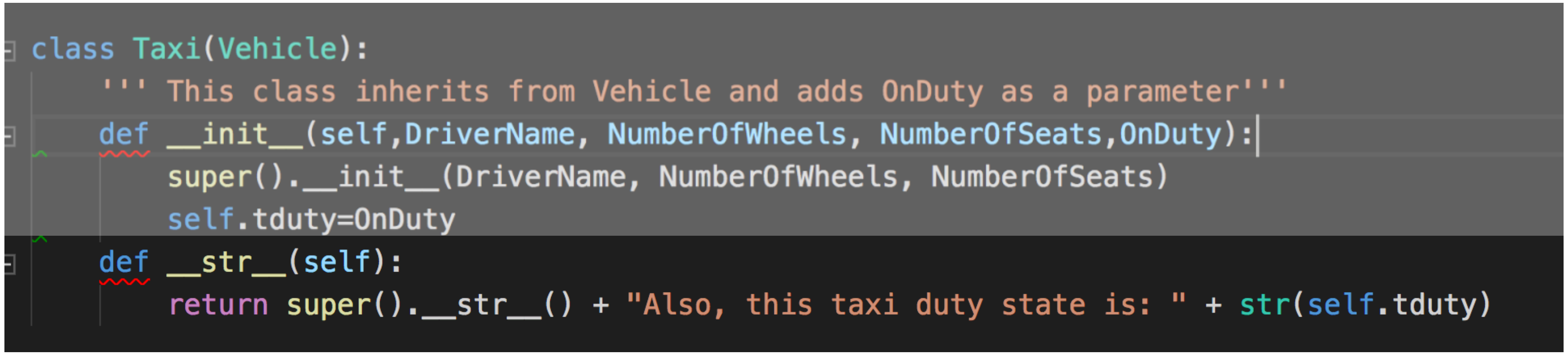


```
This vehicle is driven by: Aleksander and it has 4 wheels.
```

Example of overriding __str__

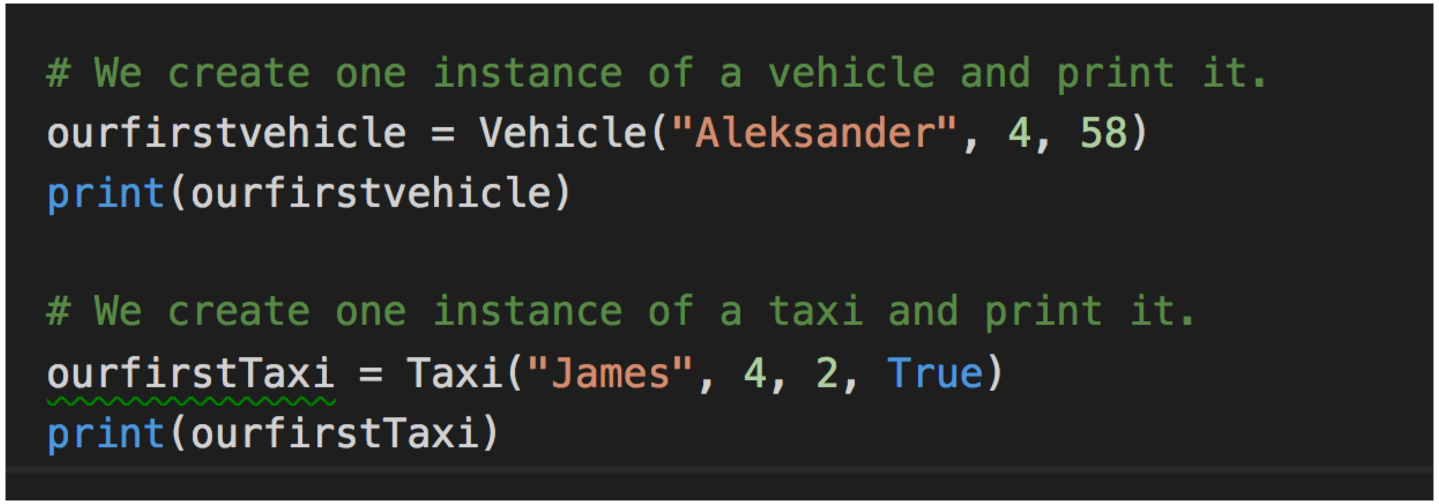


```python
class Vehicle:
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
    def __str__(self):
        '''This method return's the vehicle details for printing on screen'''
        return "This vehicle is driven by: " + self.dname + " and it has " + str(self.nwheels) + " wheels."
```



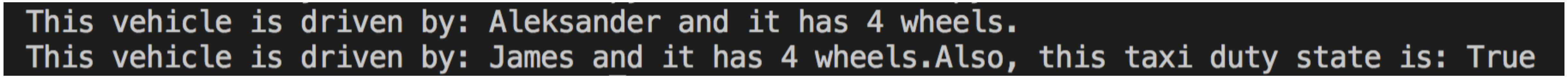
```python
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
    def __str__(self):
        return super().__str__() + "Also, this taxi duty state is: " + str(self.tduty)
```

Example of overriding __str__



```python
# We create one instance of a vehicle and print it.
ourfirstvehicle = Vehicle("Aleksander", 4, 58)
print(ourfirstvehicle)

# We create one instance of a taxi and print it.
ourfirstTaxi = Taxi("James", 4, 2, True)
print(ourfirstTaxi)
```



```
This vehicle is driven by: Aleksander and it has 4 wheels.
This vehicle is driven by: James and it has 4 wheels.Also, this taxi duty state is: True
```

Exercise Time

Part1: Update your **class Human** with a __str__ method that can be used on print. It should return the name and lastname of the human. Try it out by creating one human in code.

Part2: Update your **class student** and **class researcher** by overriding the __str__ method.

- __str__ in the child classes should use **super().__str__(args)** to call its parent method to print out the name and lastname.

Part3: For student, override the __str__(args) method so it returns in addition to the name and lastname, also the skype nickname. Do the same for the researcher with his phone extension.

Takeaways

Today, we learned how and when to use inheritance in python OOP.

- Whenever our classes can reuse the attributes and methods from parent classes.
- We inherit from parent classes by passing their name as a parameter to our child class.
- We reuse the methods and attributes from parent classes by using super().

We also learned how to override methods in python.