Object Oriented Programming

Python Review

Python Review

```
print(“Hello World!”)
print(27+3)
print(2*2)
print(2**2)

myName=“technocamps”
print(myName)

myLastName = input(“Please provide your last name: ”)
print(myLastName)

grade=70
if grade >= 70 :
    print(“You got an A”)
elif grade >= 60 :
    print(“You got a B”)
else :
    print(”You got a C”)
```

```
counter=1
while(counter<5):
    print(”The counter is ”, counter)
    counter+=1
print(“End”)

count=0
for count in range(0,5):
    print(“The count is “, count)
print(“End”)
```

Object Oriented Programming

Object Oriented Programming

Object Oriented Programming (OOP) contains:

- Classes with properties and methods
- Objects (instances of the class)

Object Oriented Programming

There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Activity: Write in your workbooks what you think these concepts mean.

Activity: Collect Information

Activity: Collect Information

Let's look back at the information you collected!

Name: Casey

Age: 10

Date of birth:21/05/2008

All of this information is about Students. We can create a class from this.

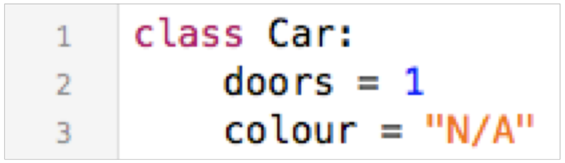
Classes

Classes

A **Class** is like recipe. It is used to build each object.

To create a class, use the keyword class.

Here we have a **class** Car with some defined **properties:** doors and colour.



```
class Car:
    doors = 1
    colour = "N/A"
```

Classes Activity

Think back to the information we collected.

What would we call the class to store this information?

What properties would the class have?

What should we set the default property values to? What values do you all share?

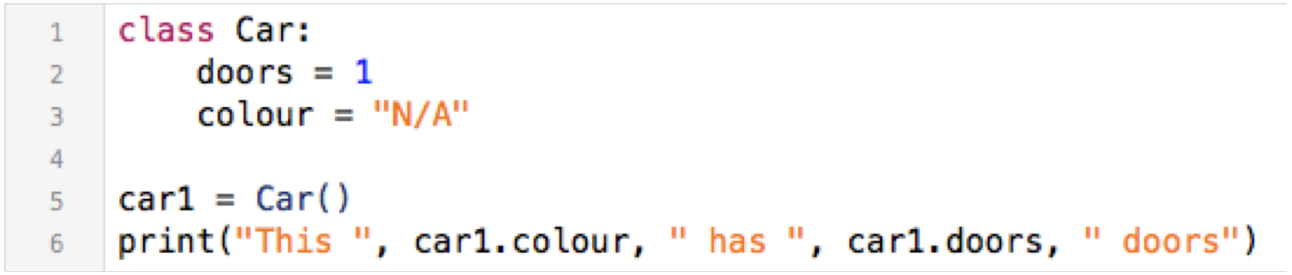
Write a class to hold this information.

Objects

Objects

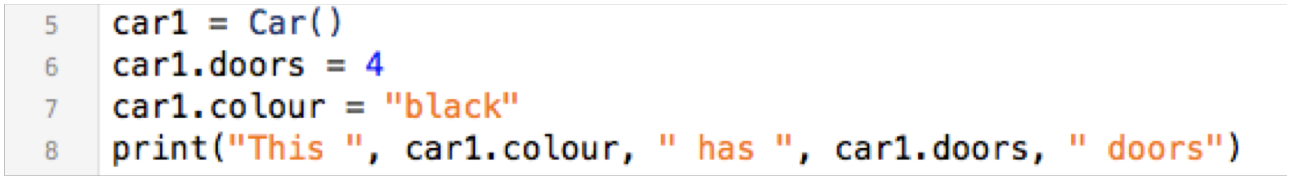
An **Object** is an **instance** of the **class.**

An object can either use the default values set by the class or it can change the values to suit itself.



```python
class Car:
    doors = 1
    colour = "N/A"

car1 = Car()
print("This ", car1.colour, " has ", car1.doors, " doors")
```

uses default values



```python
car1 = Car()
car1.doors = 4
car1.colour = "black"
print("This ", car1.colour, " has ", car1.doors, " doors")
```

sets it's own value for doors and colours

Objects Activity

Create an object that holds the correct information about your classmate.

Which properties do we need to set ourselves?

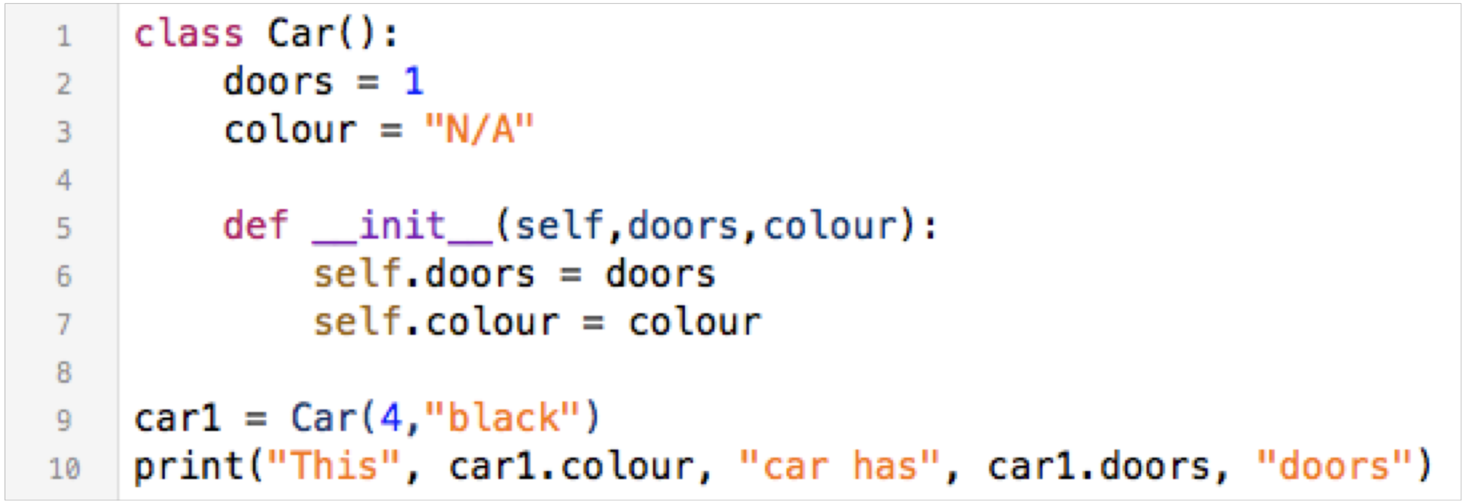
Which properties can use the default values?

Methods

Methods

All classes have a function called __init__(), which is always executed when the class is being initiated.

The __init__() function allows you to assign values to object properties, or complete other operations that are necessary to do when the object is created.



```python
class Car():
    doors = 1
    colour = "N/A"

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

car1 = Car(4,"black")
print("This", car1.colour, "car has", car1.doors, "doors")
```

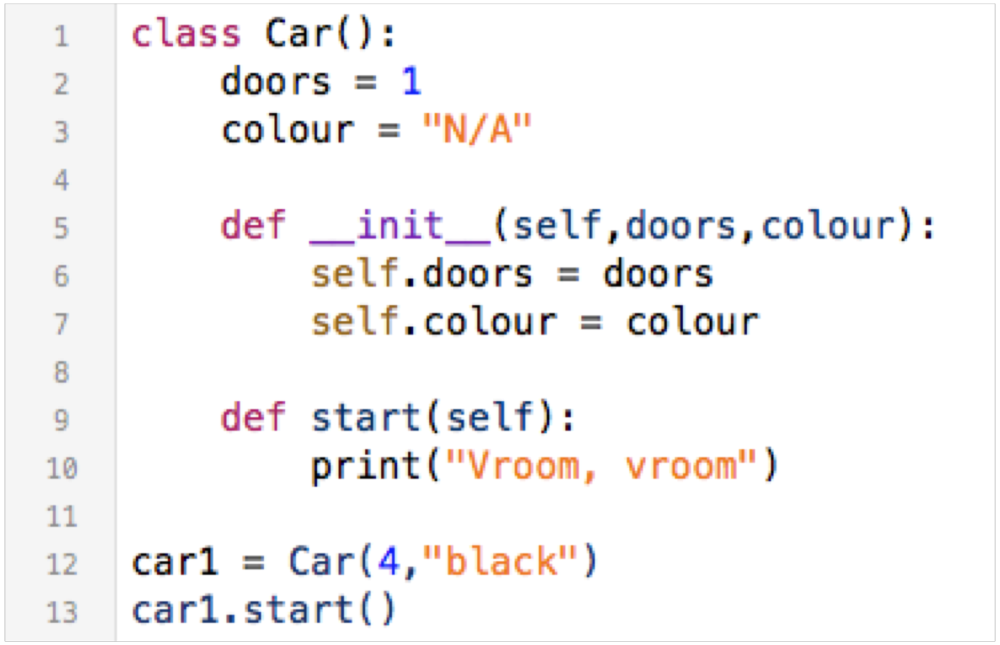
Methods

Classes can also have other methods which the object can call.

Think of them as instructions.

If you had an object “Dog” what can you tell it to do?

Methods



```python
class Car():
    doors = 1
    colour = "N/A"

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("Vroom, vroom")

car1 = Car(4,"black")
car1.start()
```

“self” refers to the object that is calling the method.

Methods Activity

Create a method called addStudent that does the following:

- Takes the students name, age, day, month and year of birth and adds the student to a list of students

Create another method called printStudents that does the following:

- Iterates through the list of students and prints their name, age and date of birth in dd/mm/yyyy format

Write these into your project and test they work.

Adding the GUI

Adding the GUI

Using the GUI code provided for you in your workbooks make sure your program works as expected.

https://cstechnocloud.swan.ac.uk/owncloud/index.php/s/r7WWmEzrOqxlhgL

What does the add student button do?

Do we need to add validation?

How do we know a student is added?

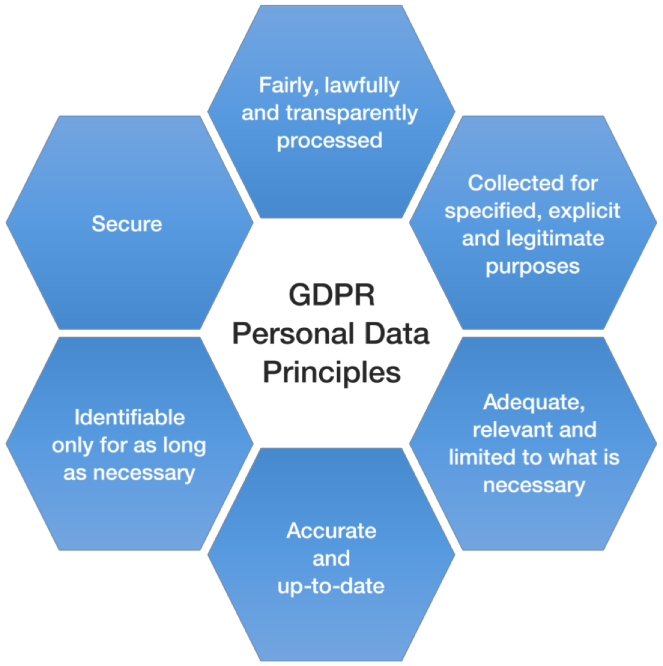
What can be done to improve the code?

Improving the GUI

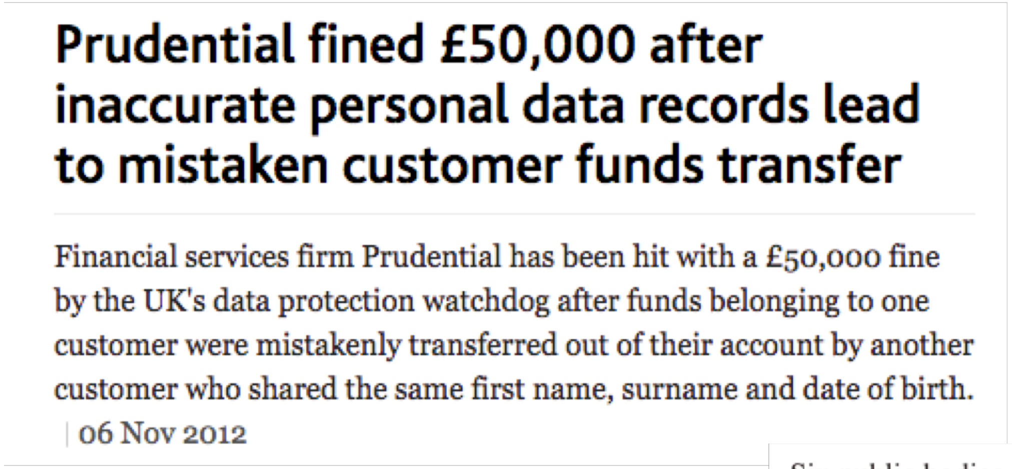
Validation

Validation is important because:

- Data should be accurate
- Data should be up to date
- If it is invalid it won't be useful


Fairly, lawfully and transparently processed
Secure
Collected for specified, explicit and legitimate purposes
GDPR Personal Data Principles
Identifiable only for as long as necessary
Adequate, relevant and limited to what is necessary
Accurate and up-to-date

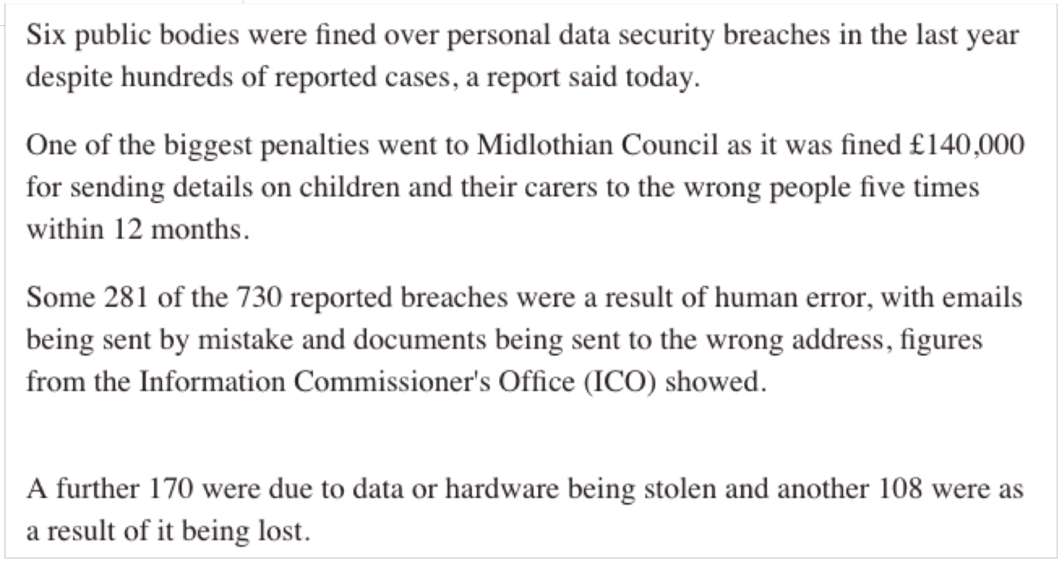
Validation



Prudential fined £50,000 after inaccurate personal data records lead to mistaken customer funds transfer

Financial services firm Prudential has been hit with a £50,000 fine by the UK's data protection watchdog after funds belonging to one customer were mistakenly transferred out of their account by another customer who shared the same first name, surname and date of birth.

06 Nov 2012



Six public bodies were fined over personal data security breaches in the last year despite hundreds of reported cases, a report said today.

One of the biggest penalties went to Midlothian Council as it was fined £140,000 for sending details on children and their carers to the wrong people five times within 12 months.

Some 281 of the 730 reported breaches were a result of human error, with emails being sent by mistake and documents being sent to the wrong address, figures from the Information Commissioner's Office (ICO) showed.

A further 170 were due to data or hardware being stolen and another 108 were as a result of it being lost.

Validation

Firstly add a check to the addStudent method to make sure the name and age isn’t blank.

Now add the date validation method in your code and validate the date of birth passed in the addStudent method. Test the following dates:

- 30/02/2003
- 01/03/2000
- 32/12/2001

Improving the GUI

We can actually validate the date using the code below:

```
import datetime

def validate(d,m,y):
    date_string = d+"-"+m+"-"+y
    date_format = '%d-%m-%Y'
    Result = True
    try:
        date_obj = datetime.datetime.strptime(date_string, date_format)
    except ValueError:
        Result = False
        print("Date of birth is not valid")
    return Result
```

Object Oriented Programming

There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

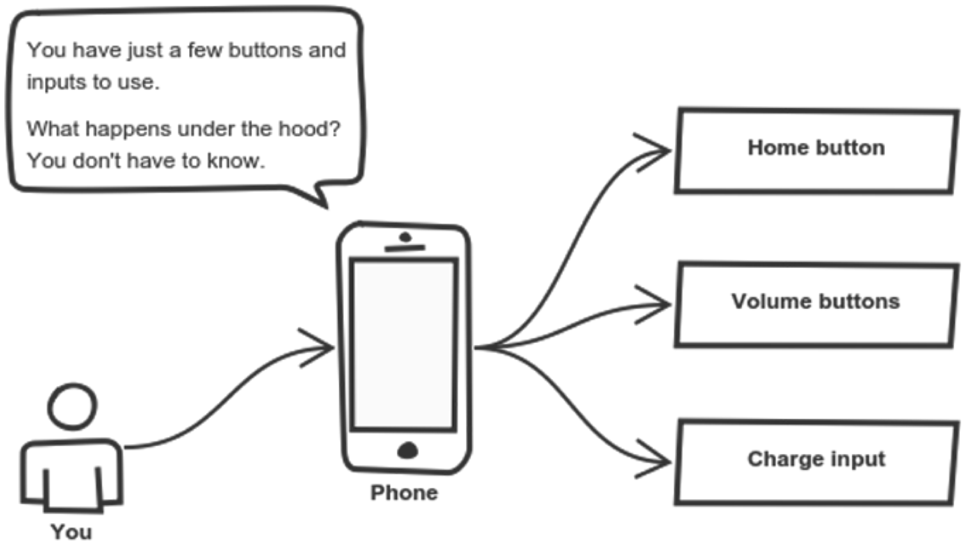
Abstraction

Abstraction

Abstraction is the process of removing unnecessary detail and simplifying.

Applying abstraction means that each object should **only** expose a high-level mechanism for using it.

Think — a coffee machine. It does a lot of stuff and makes quirky noises under the hood. But all you have to do is put in coffee and press a button.


You have just a few buttons and inputs to use.
What happens under the hood? You don't have to know.
Home button
Volume buttons
Charge input
Phone
You

Abstraction Activity

Think of other every day objects you see/use but don’t know what goes on behind the scenes.

Write down how you interact with the object and the outcome.

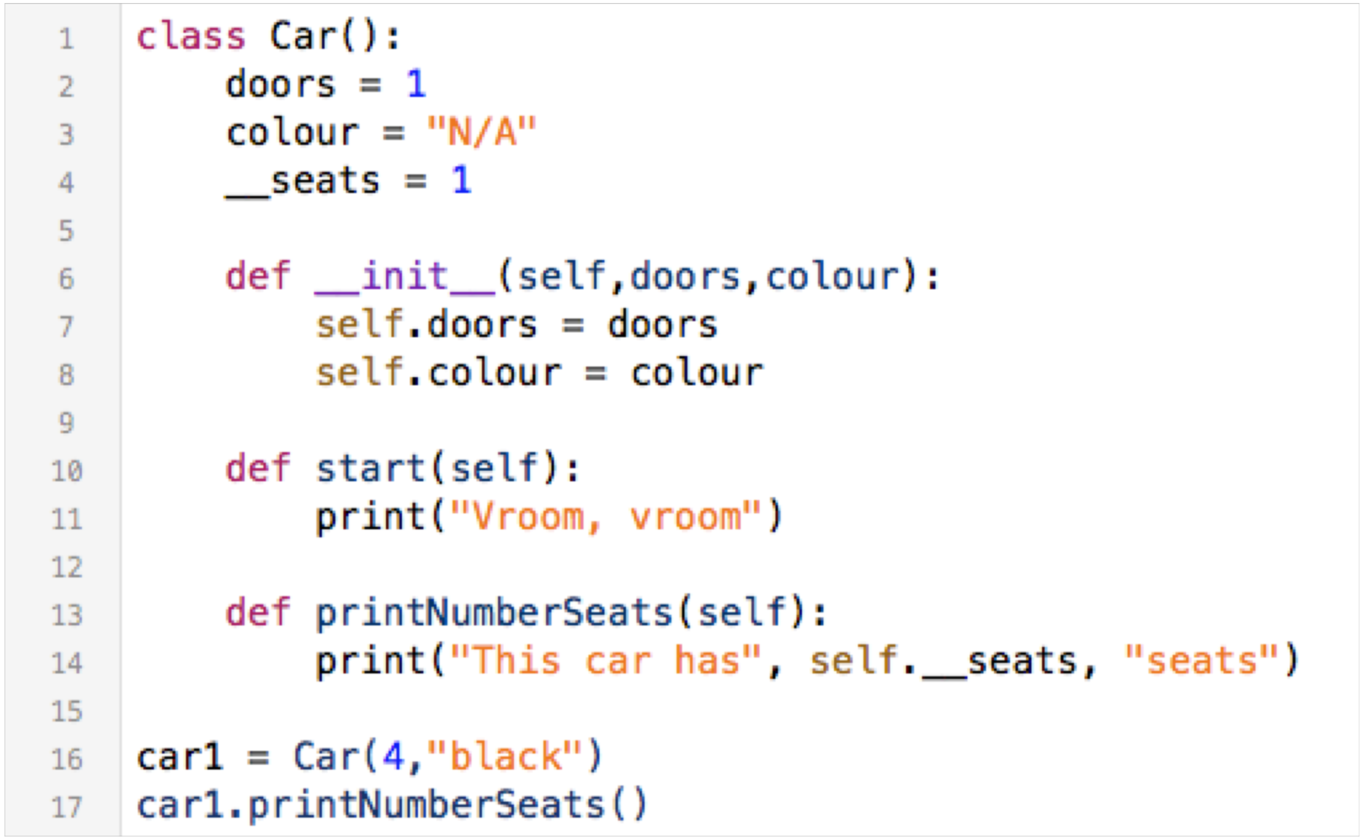
Encapsulation

Encapsulation

Encapsulation is achieved when each object keeps its state **private**, inside a class. Other objects don't have direct access to this state. Instead, they can only call a list of public functions — called methods.

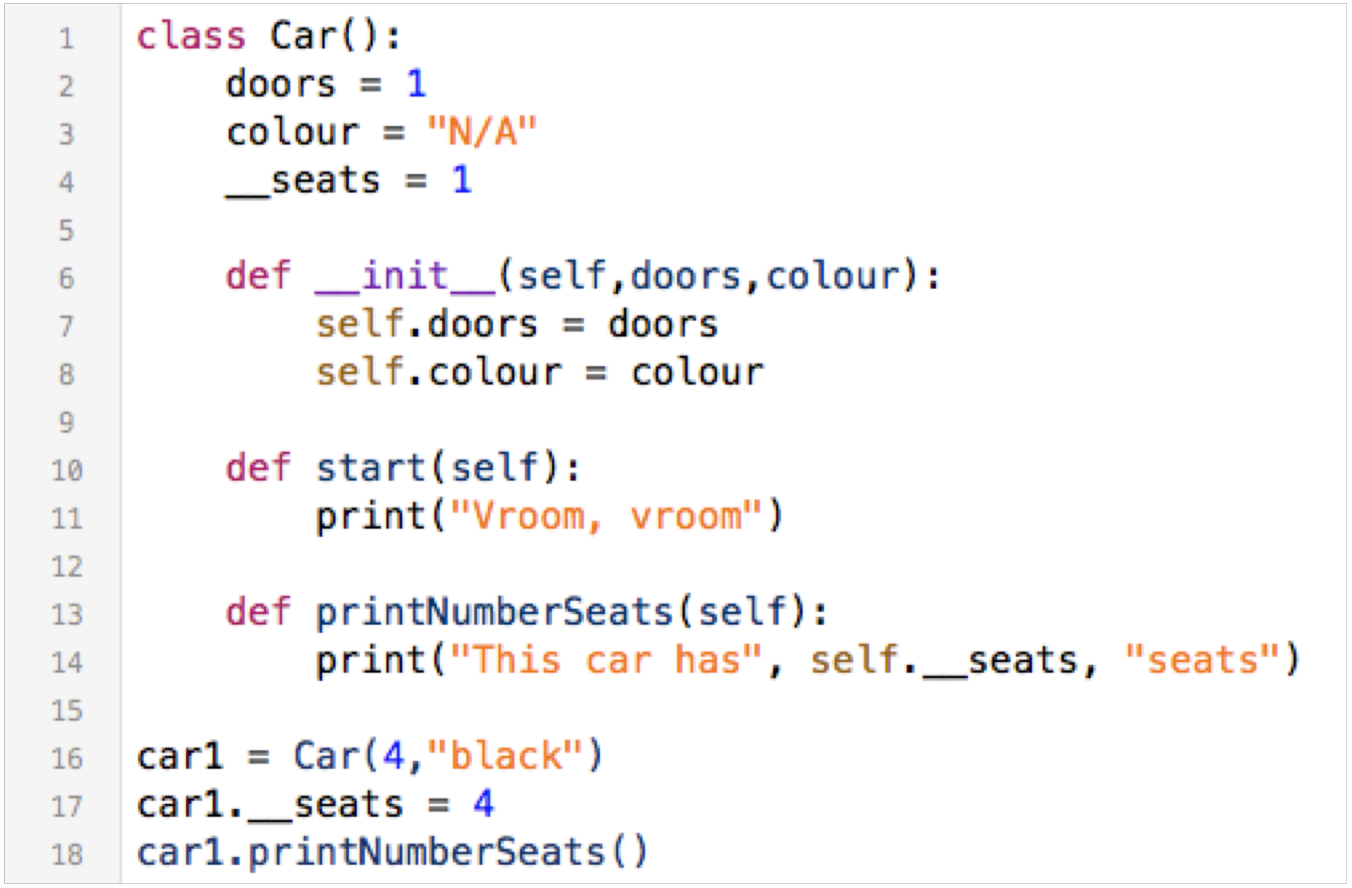
In python __ sets a variable to private.

Encapsulation



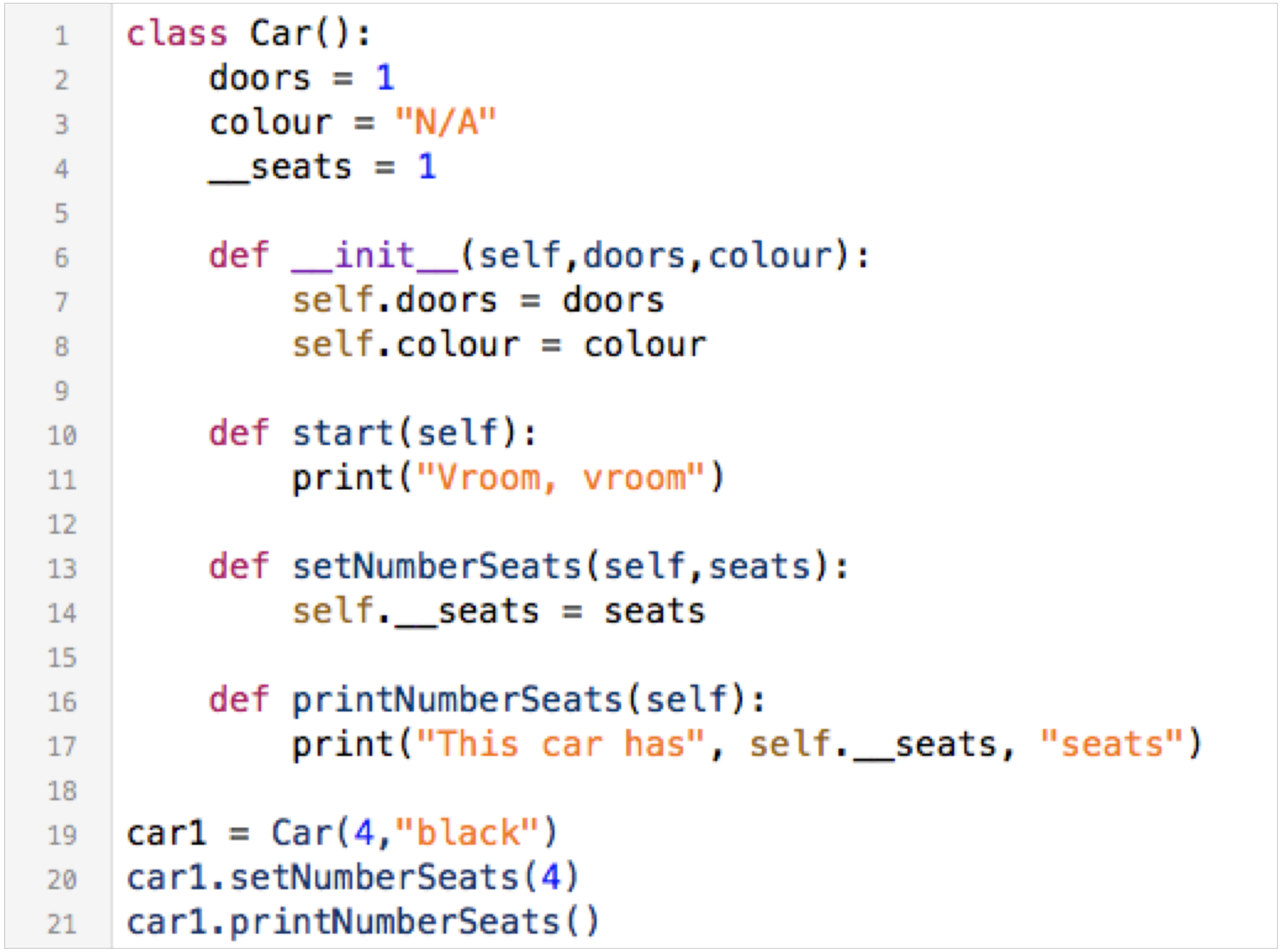
```python
class Car():
    doors = 1
    colour = "N/A"
    __seats = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("Vroom, vroom")

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

car1 = Car(4,"black")
car1.printNumberSeats()
```

Encapsulation



```python
class Car():
    doors = 1
    colour = "N/A"
    __seats = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("Vroom, vroom")

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

car1 = Car(4,"black")
car1.__seats = 4
car1.printNumberSeats()
```

Encapsulation



```python
class Car():
    doors = 1
    colour = "N/A"
    __seats = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("Vroom, vroom")

    def setNumberSeats(self,seats):
        self.__seats = seats

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

car1 = Car(4,"black")
car1.setNumberSeats(4)
car1.printNumberSeats()
```

Inheritance

Inheritance

Objects are often very similar. They share common logic. But they're not **entirely** the same.

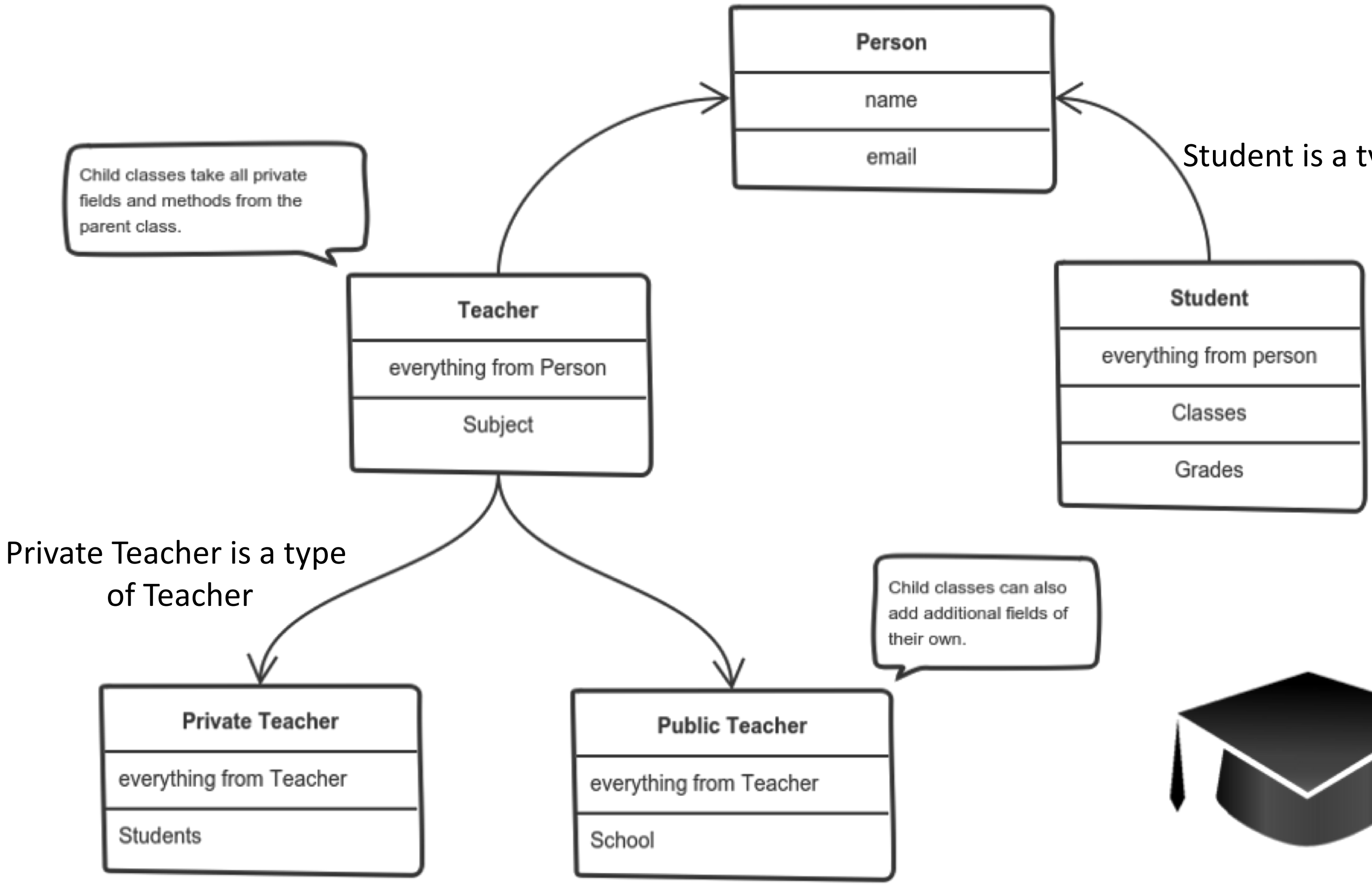
So how do we reuse the common logic and extract the unique logic into a separate class? One way to achieve this is inheritance.

It means that you create a (child) class by deriving from another (parent) class. This way, we form a hierarchy.

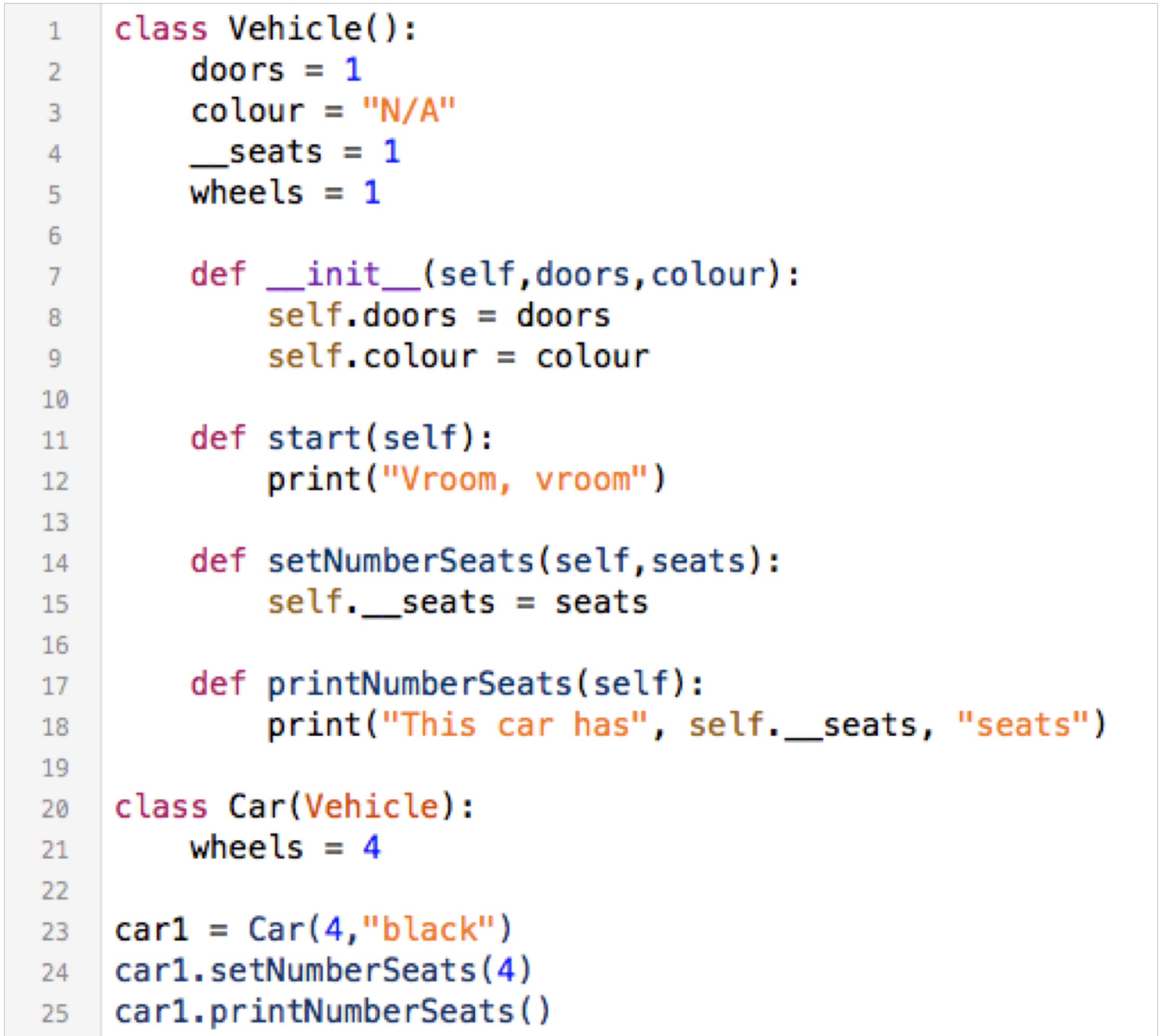
The child class reuses all fields and methods of the parent class (common part) and can implement its own (unique part).

Think – *subclass* is a type of *parent.*

Inheritance


Person
name
email
Student is a type of person
Child classes take all private fields and methods from the parent class.
Teacher
everything from Person
Subject
Student
everything from person
Classes
Grades
Private Teacher is a type of Teacher
Child classes can also add additional fields of their own.
Private Teacher
everything from Teacher
Students
Public Teacher
everything from Teacher
School

Inheritance



```python
class Vehicle():
    doors = 1
    colour = "N/A"
    __seats = 1
    wheels = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("Vroom, vroom")

    def setNumberSeats(self,seats):
        self.__seats = seats

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

class Car(Vehicle):
    wheels = 4

car1 = Car(4,"black")
car1.setNumberSeats(4)
car1.printNumberSeats()
```

Inheritance Activity

Match the subclasses to the parent class. Think carefully about what the subclasses will inherit.

Some parent classes may have their own parents!

Polymorphism

Polymorphism

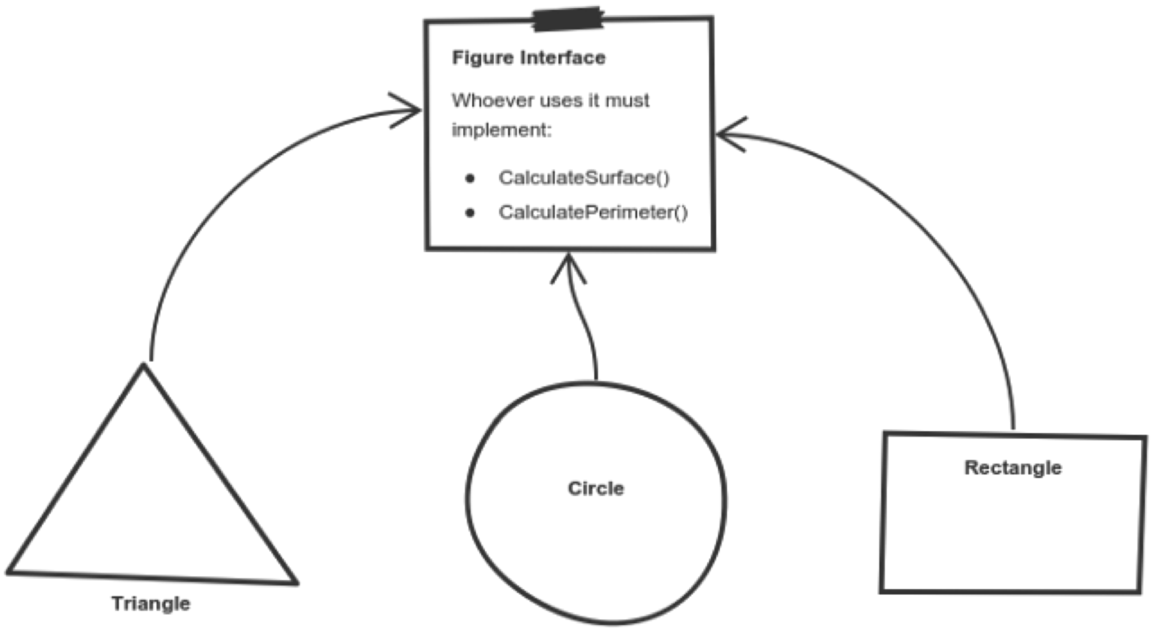
Polymorphism gives a way to use a class exactly like its parent so there's no confusion with mixing types. But each child class keeps its own methods as they are.

This typically happens by defining a (parent) interface to be reused. It outlines a bunch of common methods. Then, each child class implements its own version of these methods.

Polymorphism is perhaps the most complex concept.

- Polymorphism means that different types respond to the same function.
- Polymorphism is very useful as it makes programming more intuitive and therefore easier.

Polymorphism

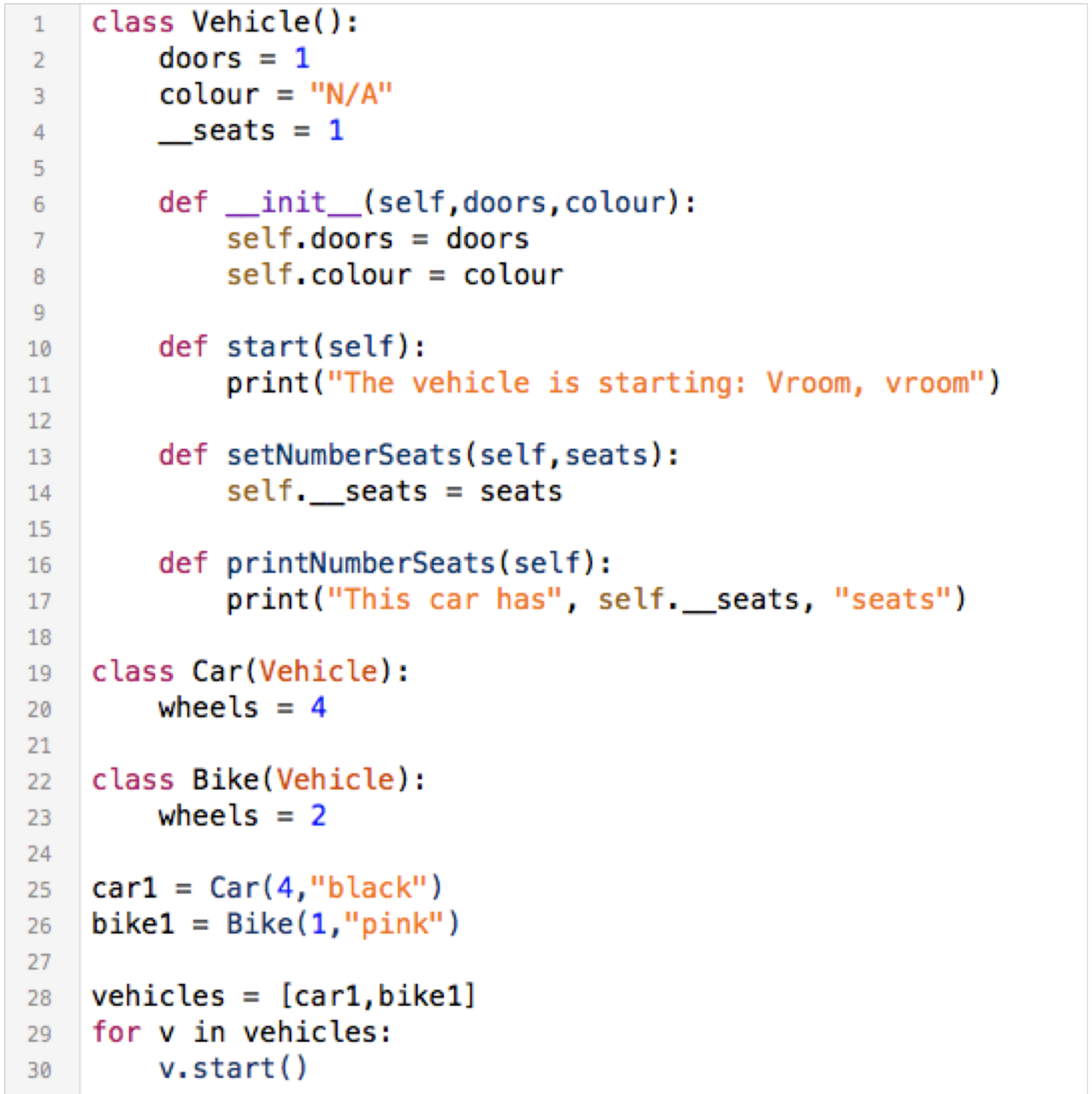


Triangle, **Circle** and **Rectangle** inherit the Figure interface or abstract class.

They implement their own versions of **CalculateSurface()** and **CalculatePerimeter()**.

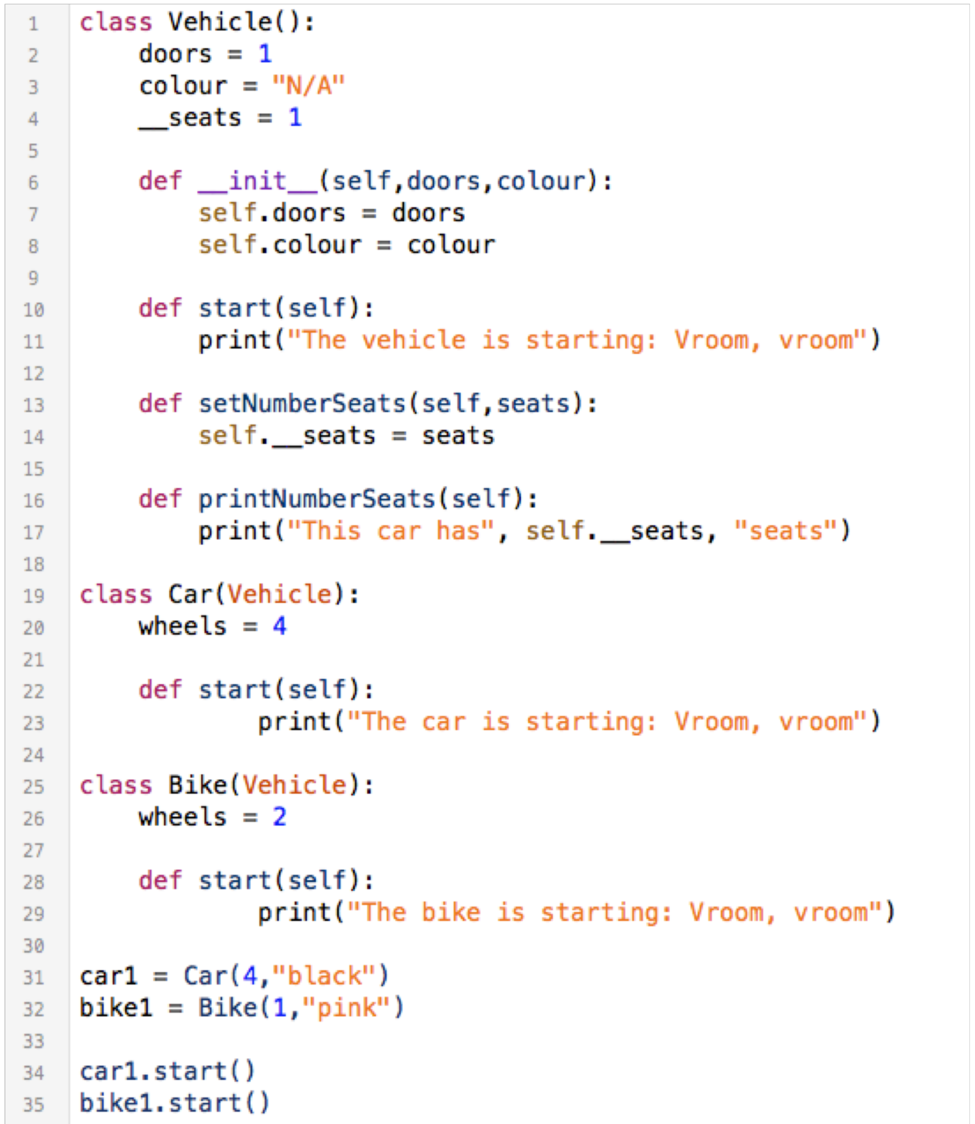
They can be used in a mixed collection of Figures.

Polymorphism



```python
class Vehicle():
    doors = 1
    colour = "N/A"
    __seats = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("The vehicle is starting: Vroom, vroom")

    def setNumberSeats(self,seats):
        self.__seats = seats

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

class Car(Vehicle):
    wheels = 4

class Bike(Vehicle):
    wheels = 2

car1 = Car(4,"black")
bike1 = Bike(1,"pink")

vehicles = [car1,bike1]
for v in vehicles:
    v.start()
```

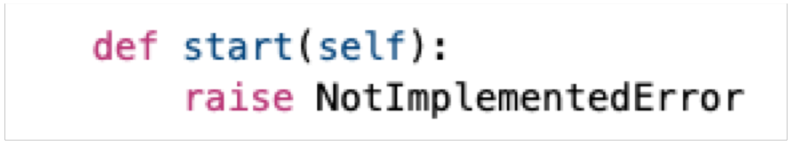
Polymorphism – Method Overriding



```
class Vehicle():
    doors = 1
    colour = "N/A"
    __seats = 1

    def __init__(self,doors,colour):
        self.doors = doors
        self.colour = colour

    def start(self):
        print("The vehicle is starting: Vroom, vroom")

    def setNumberSeats(self,seats):
        self.__seats = seats

    def printNumberSeats(self):
        print("This car has", self.__seats, "seats")

class Car(Vehicle):
    wheels = 4

    def start(self):
            print("The car is starting: Vroom, vroom")

class Bike(Vehicle):
    wheels = 2

    def start(self):
            print("The bike is starting: Vroom, vroom")

car1 = Car(4,"black")
bike1 = Bike(1,"pink")

car1.start()
bike1.start()
```

Polymorphism

Interfaces are classes that define the methods but have no implementation:



```
def start(self):
    raise NotImplementedError
```

Select two parent classes from the inheritance activity and think of two or more methods that would be defined in the parent (interface) class but implemented in the child class.

Note: Interfaces aren't always necessary in Python but prove useful for other languages such as Java.

Zoo Activity

Zoo Activity

Create a GUI program that hold the following information about animals in a zoo.

- 3 Ducks – Daffy, Donald, Daisy. Daffy is 5 years old, Donald is 3 years old and Daisy is 1. How many legs do ducks have? What sound do they make?
- 2 Giraffes – George and Gerald. George is 7 and Gerald is 10 years old. How many legs do giraffes have? What sound do they make?
- 1 Elephant – Nelly. Nelly is 12 years old. How many legs do elephants have? What sound do they make?

- Make a method that prints the following:

"Hello! My name is ____. I am a (type of animal). I have ___ legs. (Noise/Sound)"

Think! What defaults can be set? Do some animals share the same values?

Conclusion

There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

To ensure we use these concepts we need to utilise classes, objects and methods.