Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Introduction to Programming with Python

II.4. Exceptions, OOP

Oscar Delgado
oscar.delgado@uam.es

Exceptions

Exceptions

```
a = 3
b = 0
c = a/b
print('Hi there')
```

```
a = 3
b = 0
try:
    c=a/b
except ZeroDivisionError:
    print('Calculation error')
print('Hi there')
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

Several exceptions We can have multiple statements in the **try** block and also and multiple **except** blocks, like below:

```
try:
    a = eval(input('Enter a number: '))
    print (3/a)
except NameError:
    print('Please enter a number.')
except ZeroDivisionError:
    print('Can't enter 0.')
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

Not specifying exception Generally not recommended, as this will catch every exception, including ones that maybe you aren't anticipating when you write the code. This will make it hard to debug your program.

```
try:
    a = eval(input('Enter a number: '))
    print (3/a)
except:
    print('A problem occurred.')
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

Using the exception In any case, it is better to use an Exception object and show the problema to the user

```
try:
    c = a/0
except Exception as e:
    print(e)
```

```
int division or modulo by zero
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

else clause The else clause is executed is an exception is NOT thrown

```
try:
    file = open('filename.txt', 'r')
except IOError:
    print('Could not open file')
else:
    s = file.read()
    print(s)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

finally clause The code in the finally clause is executed ALWAYS, even if a exception is first thrown

```
f = open('filename.txt', 'w')
s = 'hi'
try:
    # some code that could potentially fail goes here
finally:
    f.close()
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Exceptions

Context manager The **finally** block can be used along with **except** and **else** blocks. This sort of thing with files is so common that it is has its own syntax:

```
s = 'hi'
with open('filename.txt') as f:
    print(s, file=f)
```

Object-oriented programming

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

OOP

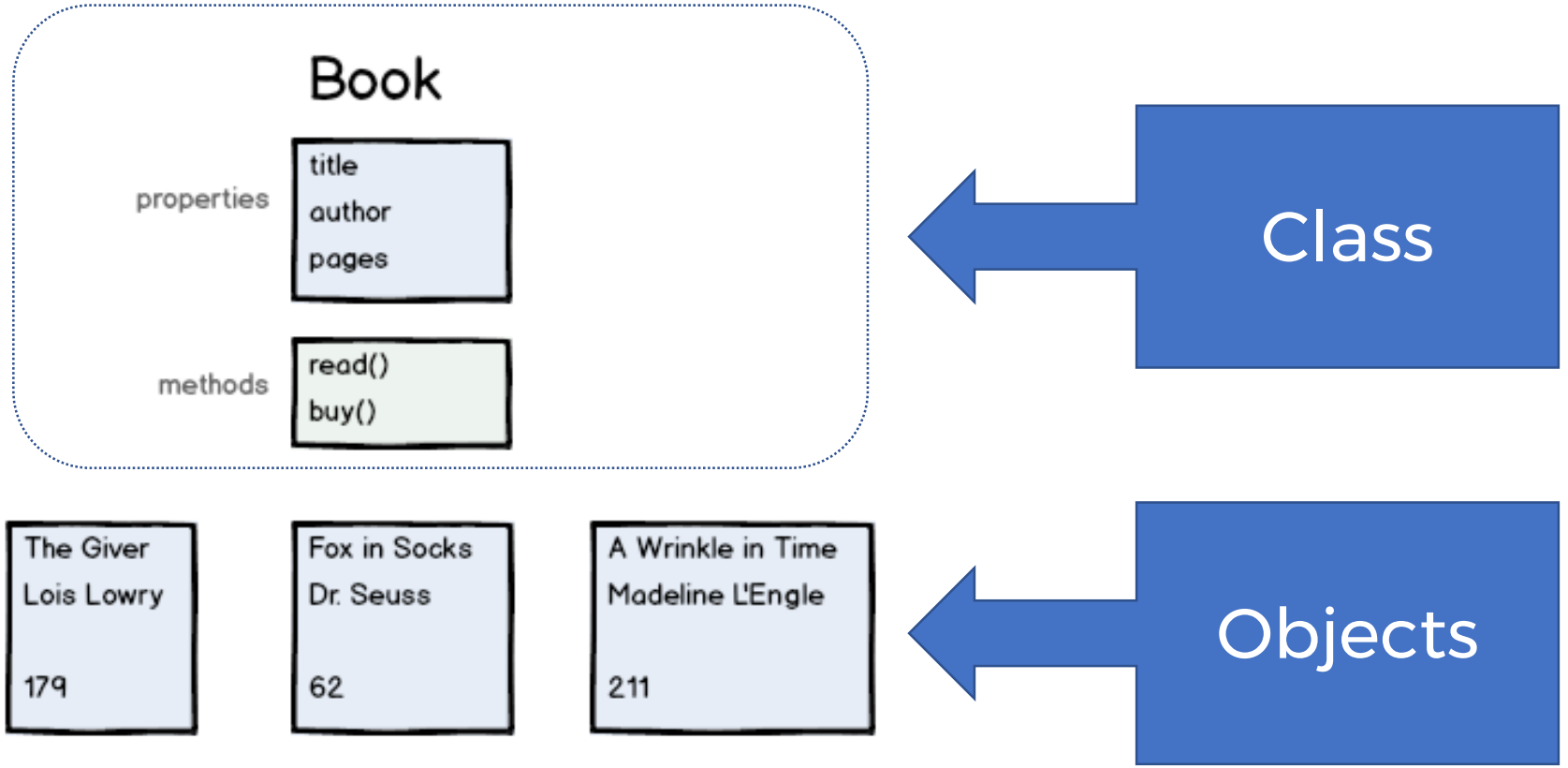
Procedure-oriented

- Top down
- Structured programming
- Centered around an algorithm
- Identify tasks; how someting is done

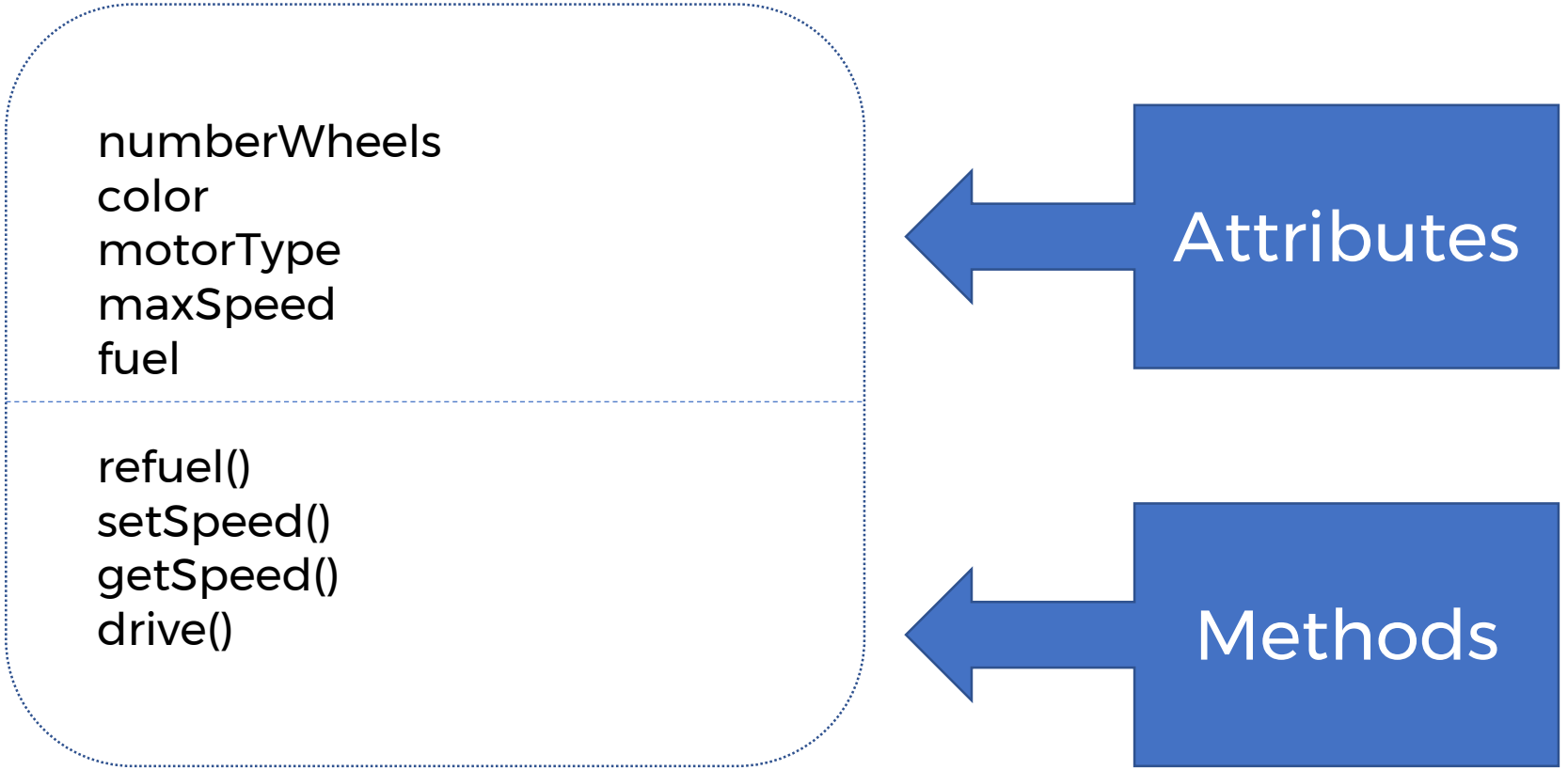
Object-oriented

- Identify objects to be modeled
- Concentrate on what an object does
- Hide how an object performs its task
- Identify behaviour

Classes - Objects



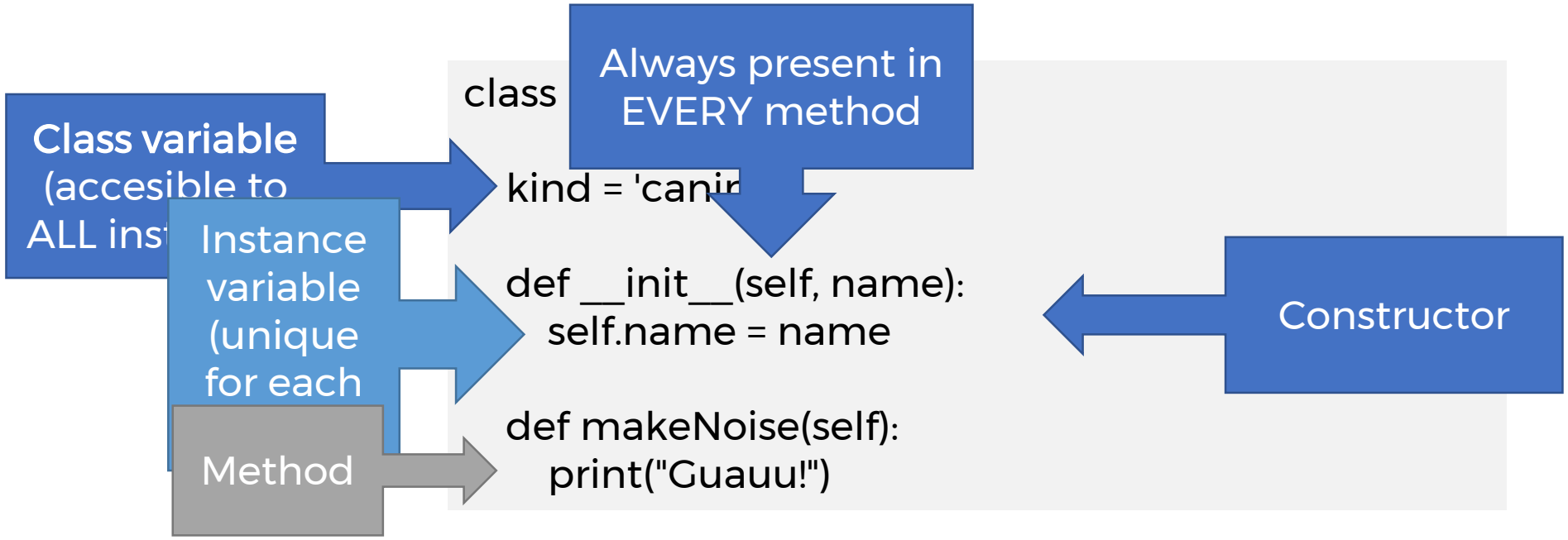
How to define a vehicle?



Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Defining a class in Python



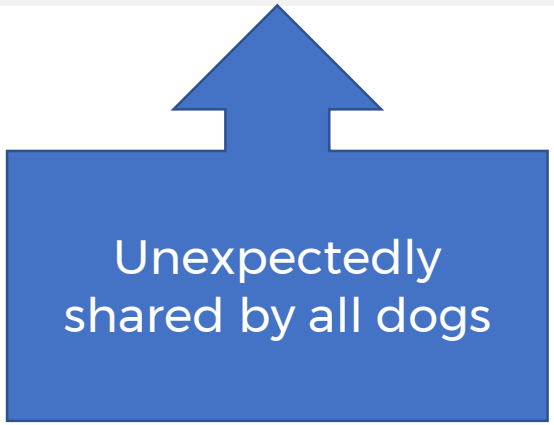

```
toby = Dog("Tobby")
toby.makeNoise()
print("Tobby is a", toby.kind)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Class vs Instance variables

```
class Dog:

    tricks = []

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)
```

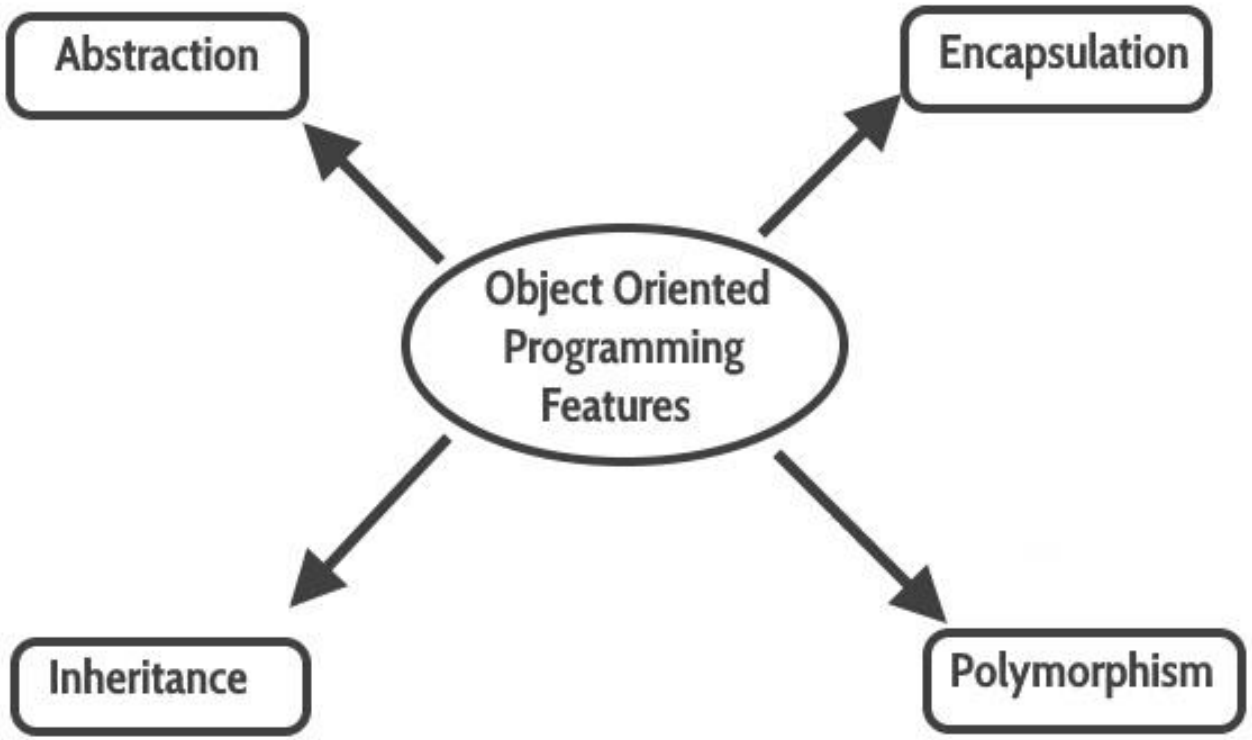



```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []

    def add_trick(self, trick):
        self.tricks.append(trick)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

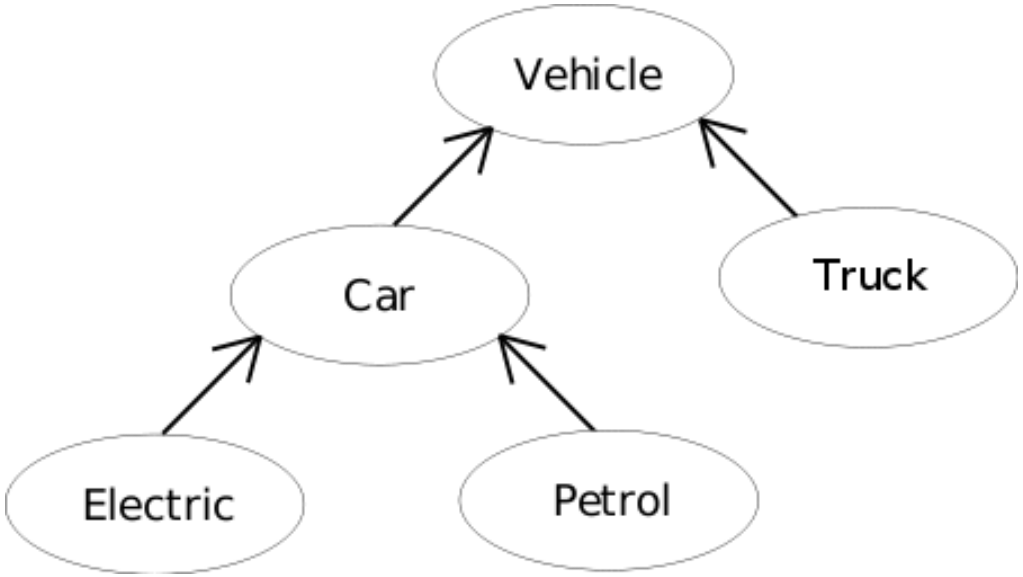
OOP main features



Inheritance

What if we have classes that share common characteristics?

An electric car shares some attributes and methods and differs in others



Inheritance & Polymorphism

```
class Parent:
    def __init__(self, a):
        self.a = a
    def method1(self):
        print(self.a*2)
    def method2(self):
        print(self.a+'!!!')

class Child(Parent):
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def method1(self):
        print(self.a*7)
    def method3(self):
        print(self.a + self.b)
```

```
p = Parent('hi')
c = Child('hi', 'bye')

print('Parent method 1: ', p.method1())
print('Parent method 2: ', p.method2())
print()
print('Child method 1: ', c.method1())
print('Child method 2: ', c.method2())
print('Child method 3: ', c.method3())
```

Polymorphism

It can *override* some methods

Inheritance

New class inherets all variables and methods from *base class*

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Printing objects

```
class Person:
    def __init__(self, name, id_number, age):
        self.name = name
        self.id_number = id_number
        self.age = age

    def __str__(self):
        return "Name: {0}\nID Number: {1}\nAge:
{2}".format(self.name, self.id_number,
self.age)

raul = Person("Raul Perez", "05920673J", 37)
print(raul)
```

Name: Raul Perez
ID Number: 05920673J
Age: 37

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID