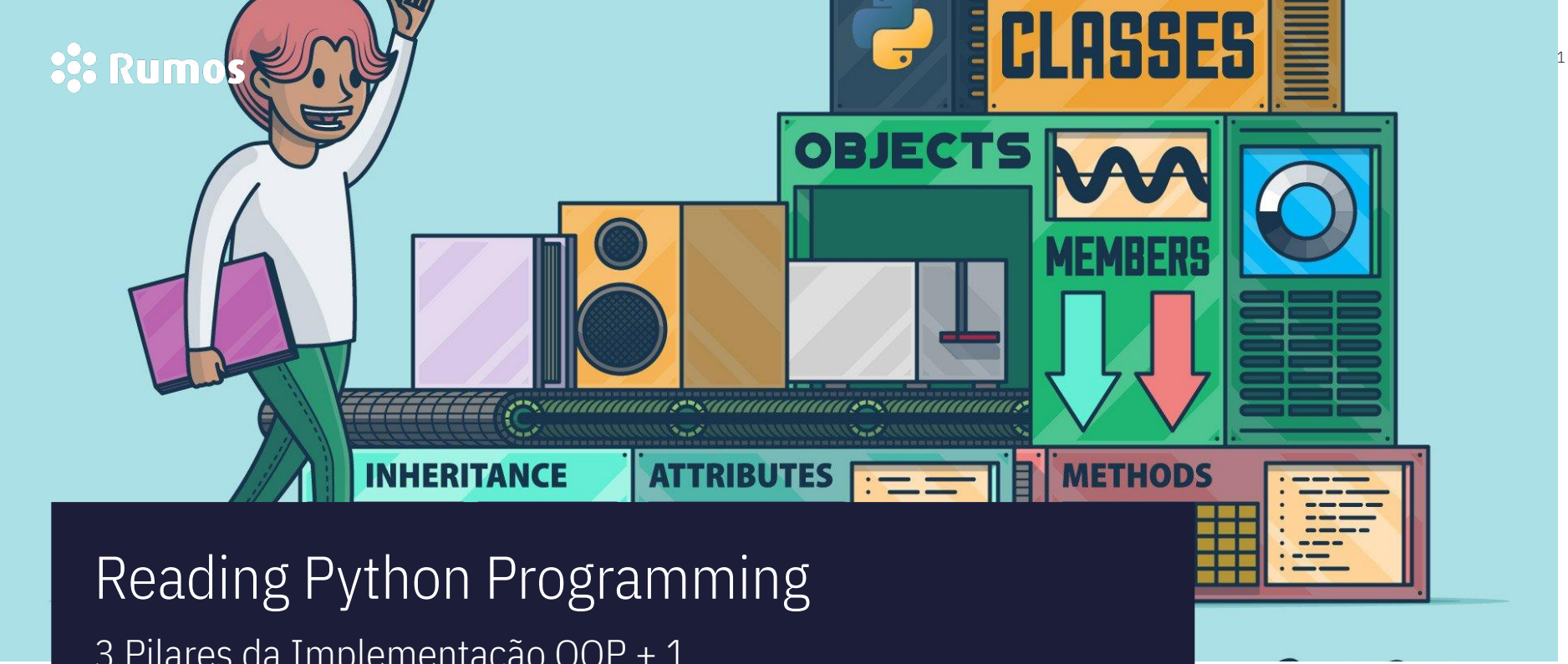


Reading Python Programming

3 Pilares da Implementação OOP + 1

16 de julho de 2024

Agenda da Sessão

01. Composição

02. Herança e Abstração e Polimorfismo

01 Composição

Rumos

Composição

- {classe} has a {classe} A classe que se instancia é composta por objetos que se iniciam aquando a criação
- Permite que algumas ações sejam abstraídos da classe principal e possam ser usados noutros objetos diferenciados
- Desacopla a lógica dessas classes permitindo uma maior manutenção e escalabilidade do código

```
class CPU:
    def on(self):
        print("CPU's on")

    def off(self):
        print("CPU's off")

class Computer:
    def __init__(self):
        self.cpu = CPU()

    def start(self):
        self.cpu.on()

    def stop(self):
        self.cpu.off()

computer = Computer()
computer.start()
computer.stop()
```

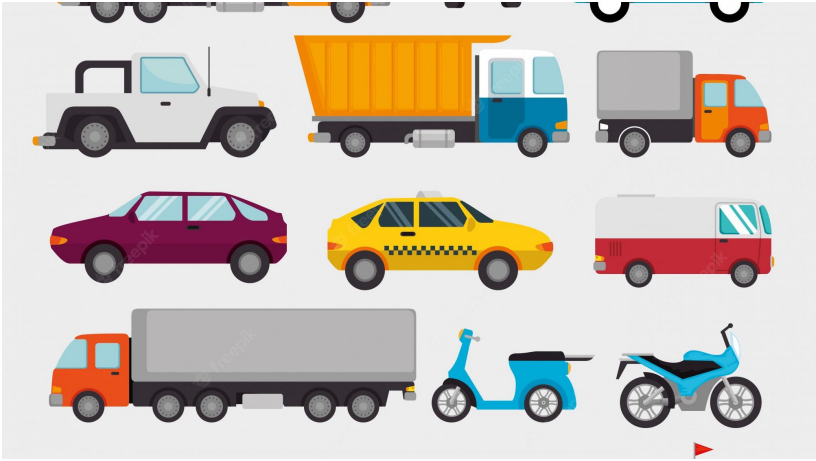
Vrum Vrum

Criar veículos diversos

Objetivo

A partir das 3 classes (que representam componentes de veículos) existentes no ficheiro vehicles, criar 3 classes de veículos diferentes compostas por esses componentes.

Os elementos que compõem os veículos devem ser impressos na consola.



02 Herança, Abstração e Polimorfismo

Herança

- Permite que uma classe filha (ou subclasse) herde atributos e métodos de uma classe mãe (ou superclasse), promovendo a reutilização de código e a criação de hierarquias de classes
- {classe} is a {classe} Indica que a subclasse é uma especialização da superclasse.

```
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    pass

dog = Dog("Max")
print(dog.name)
```

Abstração

- A superclasse pode ter métodos abstratos ou ser uma classe abstrata – sem implementação – e que é/são definidos nas subclasses
- Permite que objetos com a mesma superclasse mas implementações diferentes possam ser tratados da mesma forma

```
class Animal:
    def __init__(self, name):
        self.name = name

    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"
```

Polimorfismo

- Permite que diferentes classes, mesmo que possuam estruturas diferentes, sejam tratadas de maneira uniforme por meio de interfaces comuns
- Isso significa que, embora essas classes possam executar operações diferentes, elas podem ser manipuladas usando o mesmo conjunto de métodos, simplificando a interação com objetos de diferentes tipos

```
class Animal():
    def __init__(self, name):
        self.name = name

    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

def sound(animal):
    return print(animal.sound())

cat = Cat(“Tom”)
dog = Dog("Max")
sound(dog)
sound(cat)
```

zoo_logical

Criar e manter um zoo programático

Objetivo

No ficheiro zoo.py criar 3 classes de Animais partindo da Classe Animal.

Completar os métodos da classe Zoo e criar o zoo_logical.

Rumos

Formação e Certificação Técnica que potenciam **Profissionais e Organizações**

Lisboa

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 217 824 100
Email info@training.rumos.pt

Porto

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 222 006 551
Email info@training.rumos.pt

Siga-nos

Rumos