Programmation Orientée Objet en Python

#2 Encapsulation

par David Albert

2023

Table des matières

01 Classes et objets
Classes. Objets. Attributs et méthodes. Instances.

02 Encapsulation
Données privées, publiques. Getter / Setter. Mot-clé self.

03 Instanciation
Constructeur. Constructeur par défaut.

03 Built-in functions
Fonctions intégrées : __init__, __str__, __eq__, ...

01

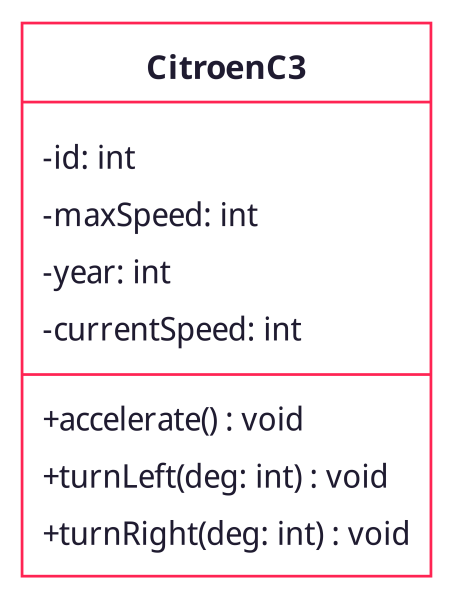
Classes et objets

Les classes

Une **classe** est un **type de données composite** constitué:

- de données que l'on appelle **attributs** (des variables primitives ou des objets)
- de **méthodes** permettant de traiter ces données et des données extérieures à la classe

Syntaxe UML



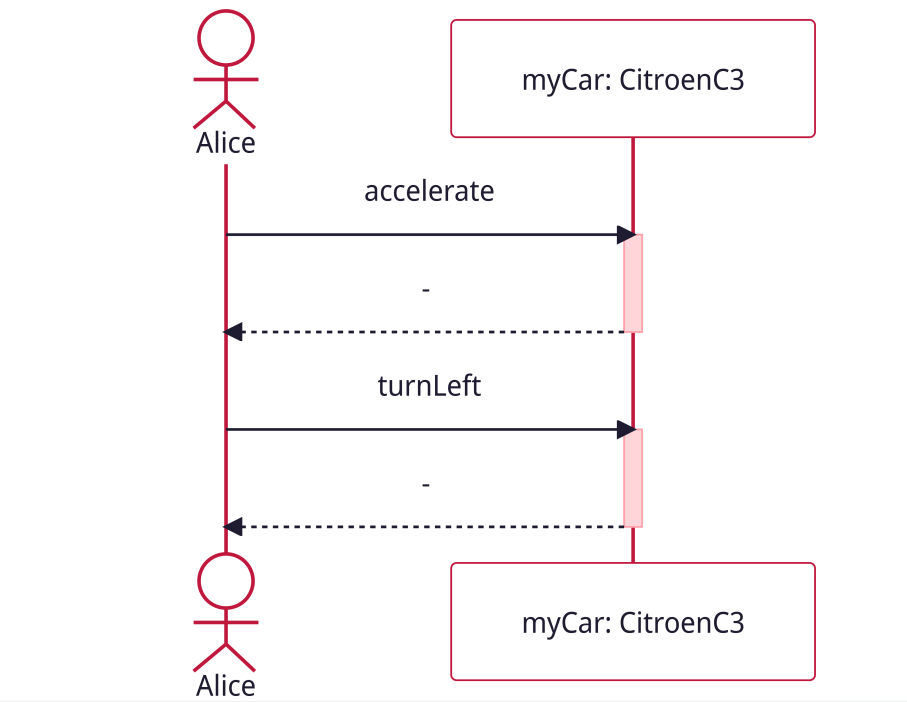
Syntaxe python

```python
class CitroenC3:
    def __init__(self):
        self._id = 0
        self._currentSpeed = 0
        self._maxSpeed = 210
        self._year = 2010

    def accelerate(self):
        pass

    def turnLeft(self, deg):
        pass

    def turnRight(self, deg):
        pass
```

Les objets

Un **objet** est une **variable** dont le type est une classe particulière
→ on parle **instance de classe**.

Syntaxe UML



Syntaxe Python

Pour instancier un objet en python on fait appel à une méthode particulière: le **constructeur**.

```
  myCar : CitroenC3 = CitroenC3()
# ------   ---------   -----------
# objet     classe     constructeur
```

> Dans un même programme il y a généralement plusieurs instances d'une même classe.

Objets et classes

Différence classe et objet

Pour mieux comprendre

Classe = Le moule pour fabriquer des objets = type de données contenant des données (attributs) et des fonctions (méthodes)

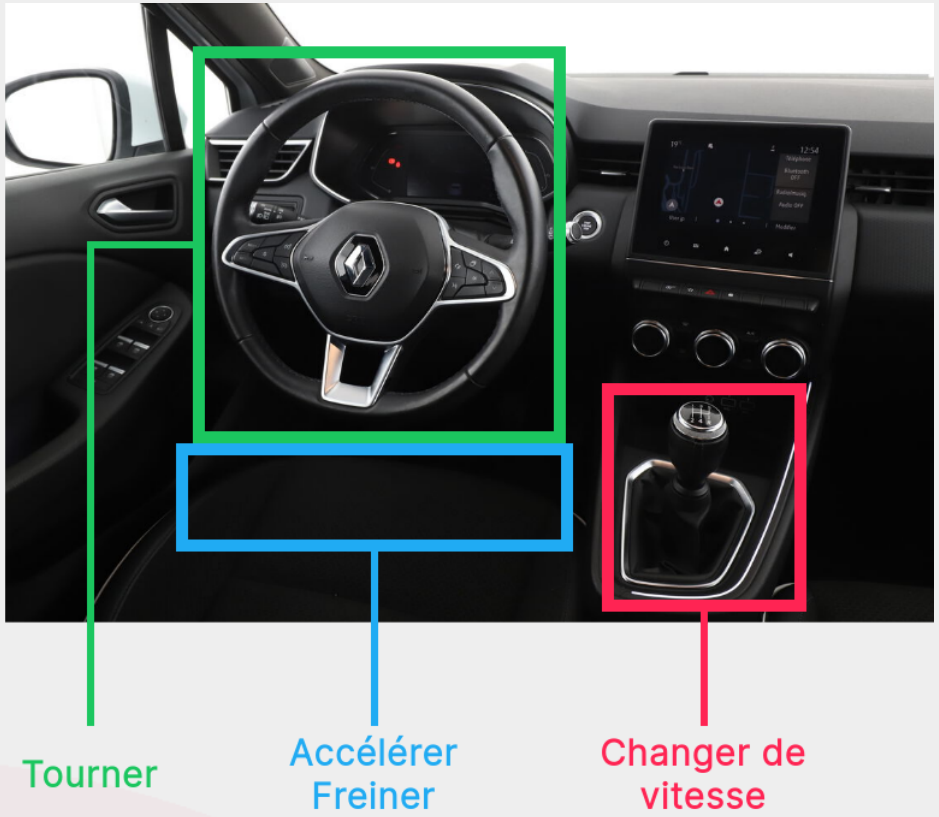
Objet = une instance de classe (l'objet une fois créé) = une donnée spécifique

02

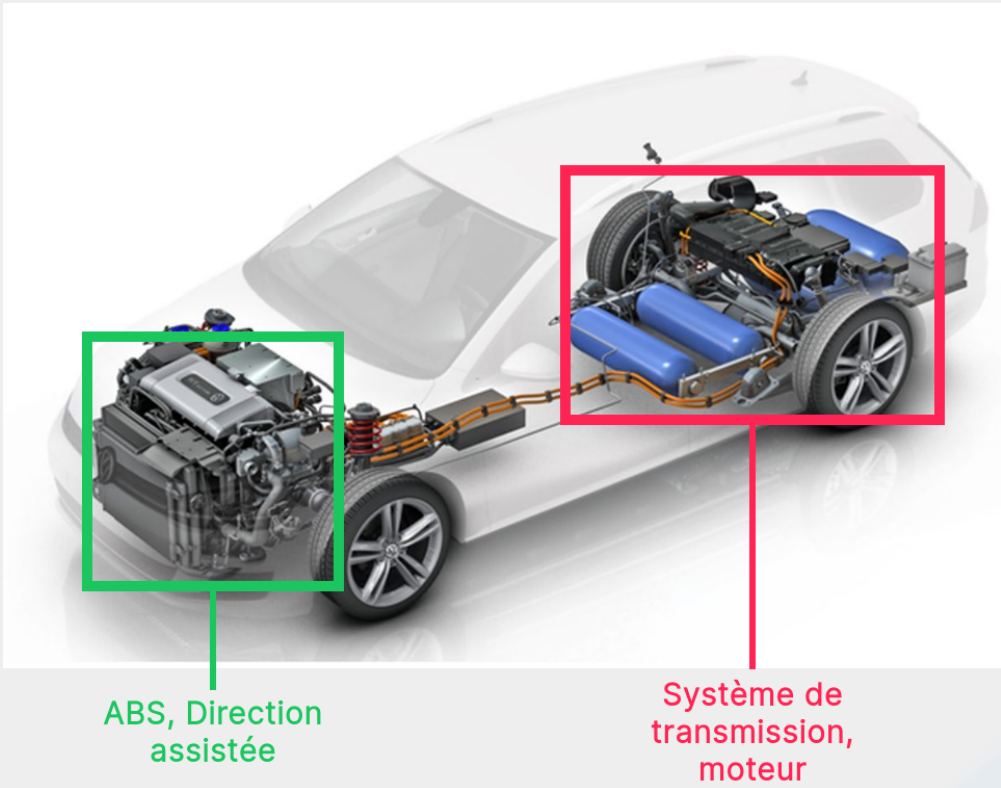
Encapsulation

Princpe

Usage simple et visible



Fonctionnement complexe et caché



Visibilité des attributs et méthodes

L'encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme.

Une donnée peut être déclarée en accès :

- **public** : les autres objets peuvent accéder à la valeur de cette donnée ainsi que la modifier
- **privé** : les autres objets n'ont pas le droit d'accéder directement à la valeur de cette donnée (ni de la modifier). En revanche, ils peuvent le faire indirectement par grâce aux méthodes publiques de l'objet concerné

!

Par convention, les attributs et méthodes privés commencent par **_** en *python*.

Exemple: `_currentSpeed` et `_increaseSpeed()`

Bonnes pratiques

- ne rendre publique que le stricte nécessaire
  - les fonctions nécessaires à l'usage (`accelerate`, `turnLeft`, `turnRight`)
  - et pas plus (`_increaseSpeed`, `_turnLeftWheel`)
- n'utiliser que des **attributs privés**
  - utiliser un **getter** si besoin de lire depuis l'extérieur
  - utiliser un **setter** si besoin de modifier depuis l'extérieur
- suivre le principe de **responsabilité unique** - **S** de **S**OLID
  - exemple Animal / AnimalDB

Encapsulation

Getters / Setters

```python
class Music:
    def __init__(self, title, duration = 0):
        self._title = title
        self._duration = duration

    # Ceci est un getter
    def get_title(self, name):
        return self._title

    # Ceci est un setter
    def set_title(self, title):
        self._title = title
```

Mot clé self

Utilisé pour accéder à l'instance d'une classe. On pourra l'utiliser dans la méthode pour accéder aux données.

!

Il doit impérativement être le **premier argument** de chaque méthode.

03

Constructeur : __init__

Constructeur

Le **constructeur** est une méthode spéciale (**__init__** en python) qui est appelée au moment de la création de l'objet.

Définition du constructeur

```python
class Music:
    # Définition du constructeur
    def __init__(self, title, artists):
        self._title = title
        self._artists = artists

    def hasAuthor(self, name):
        return name in self._artists
```

Appel du constructeur

```python
# Instanciation d'un objet de type 'Music'
m = Music('La pluie', ['Stromae', 'Orelsan'])

print(m.hasAuthor('Stromae'))
# true
```

Constructeur par défaut

Le constructeur (comme toute fonction) peut prendre des arguments par défaut.

Définition du constructeur

```
class Music:
    # Définition du constructeur
    # avec arguments par défaut
    def __init__(self, title, artists = []):
        self._title = title
        self._artists = artists

    def hasAuthor(self, name):
        return name in self._artists
```

Appel du constructeur par défaut

```
# Appel du constructeur par défaut
m2 = Music('La pluie')

print(m.hasAuthor('Stromae'))
# false
```

04

Built-in functions

(1) Built-in functions

Toute classe python contient un ensemble de méthodes intégrées (ou **built-in**).

- **__init__** : retourne une instance de la classe
  `MyClass()` ou `obj.__init__()`
- **__str__** : retourne une représentation lisible pour l'humain de l'objet
  `str(obj)` ou `obj.__str__()`
- **__repr__** : retourne une représentation machine de l'objet
  `repr(obj)` ou `obj.__repr__()`
- **__len__** : retourne la longueur de l'objet
  `len(obj)` ou `obj.__len__()`

i

On peut redéfinir chacune de ces méthodes pour un usage personnalisé.

(2) Built-in functions

Toute classe python contient un ensemble de méthodes intégrées (ou **built-in**).

- **__eq__** : redéfinit l'opérateur d'égalité
  `obj1 == obj2` ou `obj1.__eq__(obj2)`
- **__le__** : redéfinit l'opérateur inférieur ou égal
  `obj1 <= obj2` ou `obj1.__le__(obj2)`
- **__ue__** : redéfinit l'opérateur supéreur ou égal
  `obj1 >= obj2` ou `obj1.__ue__(obj2)`
- **__lt__** : redéfinit l'opérateur inférieur strict
  `obj1 < obj2` ou `obj1.__lt__(obj2)`
- **__ut__** : redéfinit l'opérateur supéreur strict
  `obj1 > obj2` ou `obj1.__ut__(obj2)`

i

On peut redéfinir chacune de ces méthodes pour un usage personnalisé.