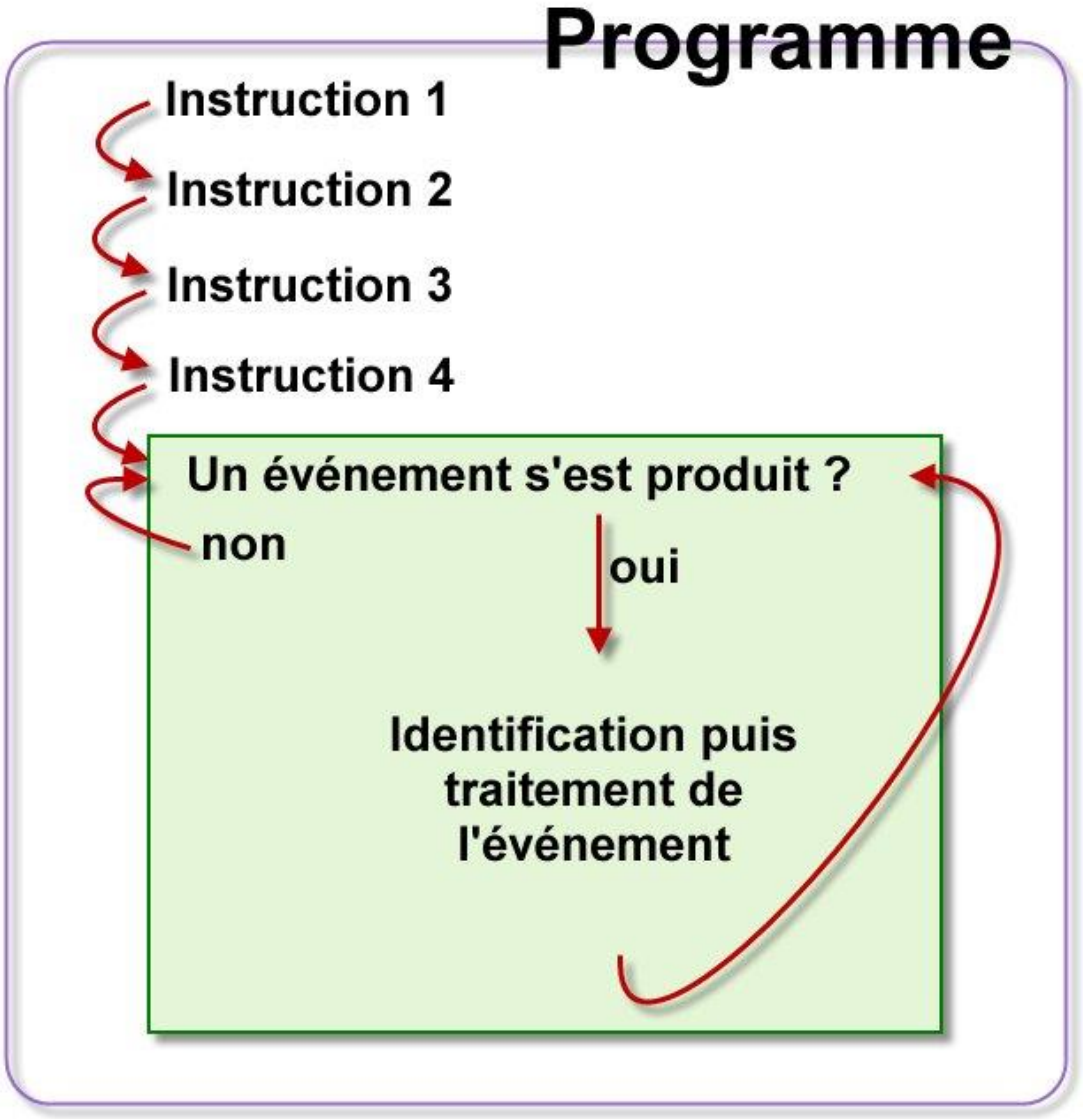
Programmation Orientée Objet

Programmation STRUCTUREE PROCEDURALE

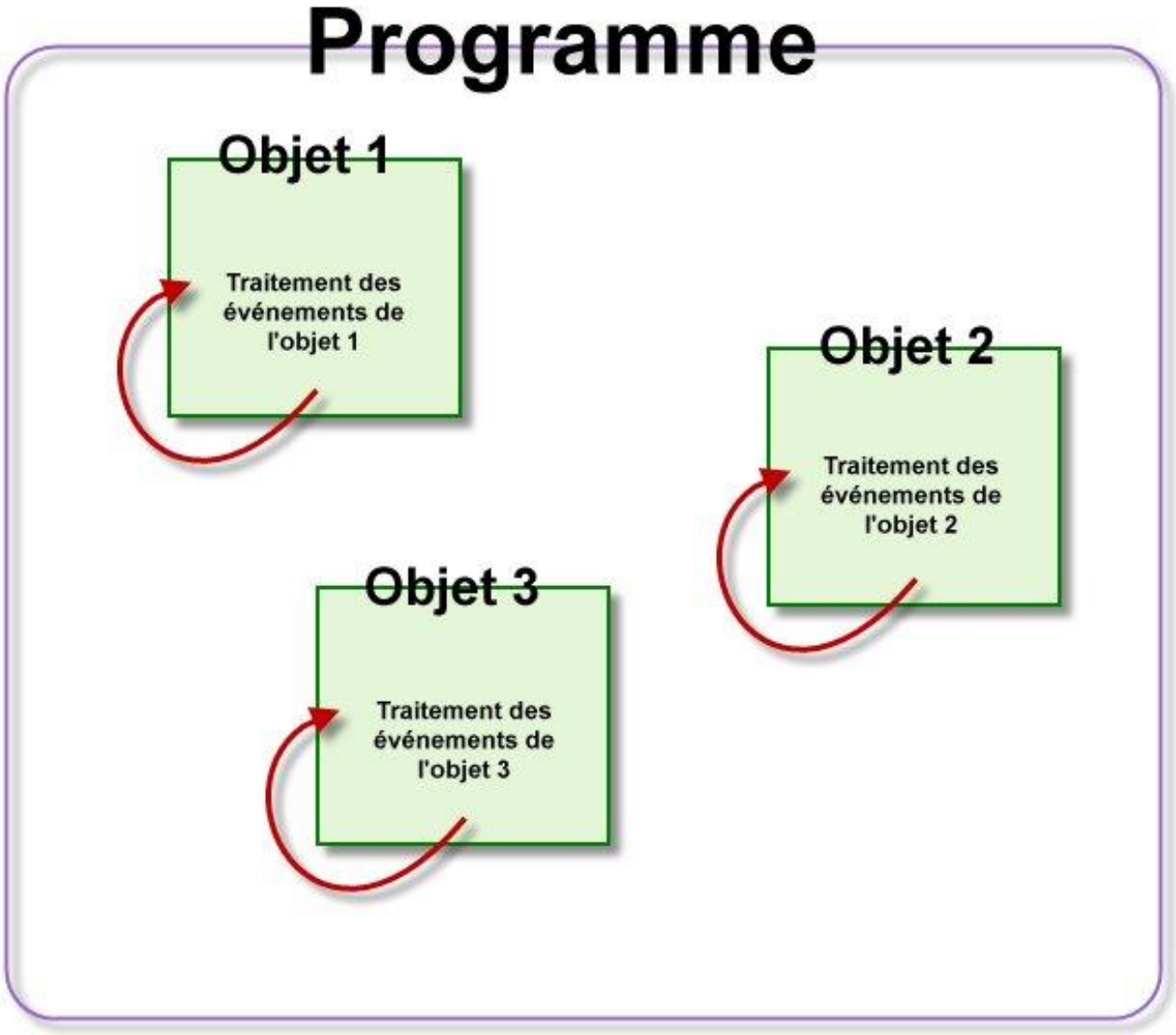


Ensemble de procédures et fonctions

Sans forcément de liens particuliers

Agissant sur des données plus ou moins dissociées

Programmation OBJET



Regrouper les fonctions et les procédures agissant sur un même jeu de données

Objets et classes

Un objet est une brique logicielle autonome regroupant des informations et des mécanismes concernant un sujet, manipulés dans un programme.

Une classe représente une catégorie d'objets. On l'utilise aussi comme comme un modèle, un moule, une usine à partir de laquelle il est possible de créer ces mêmes objets

Les classes déclarent des attributs (données membres) représentant l'état des objets et des méthodes (fonctions membres) représentant leur comportement.

REGLE IMPORTANTE : l'état d'un objet devrait toujours être modifié par ses méthodes !!

Instances

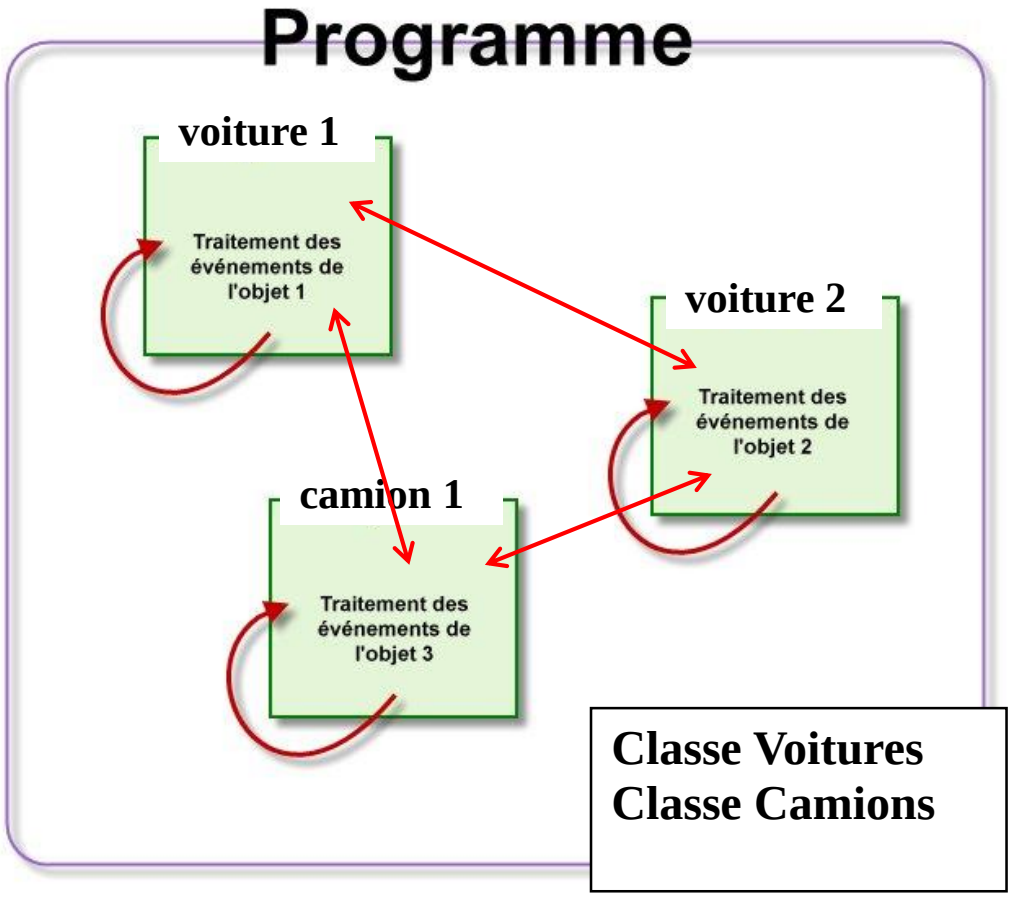
Un objet est une instance d'une classe (anglicisme).

Plusieurs objets autonomes peuvent être créés à partir d'une même classe et donc lui appartenir.

Les objets peuvent interagir entre eux.

Instancier est l'action de créer un objet
à partir d'une classe.

L'instanciation est la création d'un objet
à partir d'une classe.



Déclaration de classes et instanciation en Python

```python
class Voiture():
    """Une classe définissant un modèle de voiture"""
    def __init__(self, param_moteur, param_couleur, param_puissance):
        self.moteur = param_moteur
        self.couleur = param_couleur
        self.puissance = param_puissance
    def caracteristiques(self):
        return {"moteur": self.moteur, "couleur": self.couleur, "puissance": self.puissance}

    def modifie_couleur(self, nouvelle_couleur):
        self.couleur = nouvelle_couleur
    def __del__(self):
        print('un dernier calcul avant de mourir :', 2 + 2)
        print('je meurs .......')


voiture_1 = Voiture('diesel', 'rouge', 90)
voiture_2 = Voiture('essence', 'bleu', 140)

voiture_2.couleur = 'jaune'
voiture_2.modifie_couleur('vert')

retour_fonction = voiture_2.caracteristiques()
print(retour_fonction)

del(voiture_1)
```

les parenthèses vides ne sont pas obligatoires

Constructeur

param
d'entré

Attributs

Méthodes

Destructeur

instanciation

modification directe d'un attri

modification avec un mu

appel d'une méthod
ici un accesseur

destruction

Encapsulation

Dans un objet, données (attributs) et programme (méthodes) sont structurés

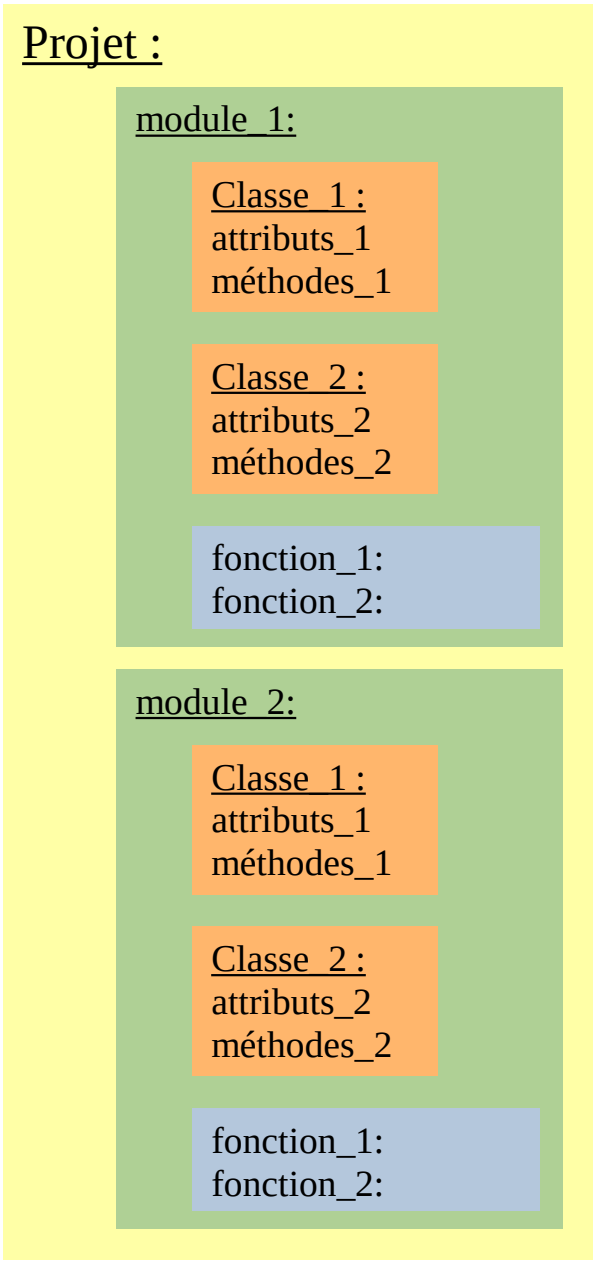
Les méthodes et les attributs sont encapsulés dans l'objet

En Python, les attributs sont automatiquement privés

En Python, les méthodes sont automatiquement publiques
C'est à l'auteur de les déclarer privées en fonction de l'interface
(ajout d'un _ devant les noms des méthodes)

Toutefois en Python ... rien n'est réellement privé …
Ce n'est qu'une question de convention.

help() **dir()**



Précisions sur les méthodes

On les classe généralement en :

- Accesseurs en lecture : **accesseurs** (getters)
- Accesseurs en écriture : **mutateurs** (setters)
- Tests : **reconnaisseurs, validateurs, prédicateurs, ...**

Méthodes spéciales de Python :

__init__(self) : initiateur (constructeur par abus), appelée juste après la création de l'objet
__str__(self) : appelée par str(objet), print(objet)

__len__(self) : appelée par len(objet)
__getitem__(self, key) : renvoie objet[key]
__setitem__(self, key, value) : objet[key] = value

__lt__(self, other) : objet < autre objet
__le__(self, other) : objet ≤ autre objet
__eq__(self, other) : objet == autre objet
__ne__(self, other) : objet != autre objet
__gt__(self, other) : objet > autre objet
__ge__(self, other) : objet ≥ autre objet

__add__(self, other) : objet + autre objet
__sub__(self, other) : objet - autre objet
__mul__(self, other) : objet * autre objet

… etc ...

https://docs.python.org/fr/3.9/reference/datamodel.html