NSI · TERMINALE / COURS

La programmation orientée objet

Astrolab Cours · astrolabcours.fr

01 Le problème

Un jeu gère cent personnages. Chacun a un nom, des points de vie, une position. Avec des listes séparées (`noms`, `vies`, `positions`), on finit par mélanger les indices. Comment regrouper tout ce qui concerne un personnage ?

01 L'idée : le moule et les gâteaux

Une **classe** est un moule : elle décrit la forme une fois pour toutes. Chaque **objet** est un gâteau sorti du moule : il a la même forme, mais sa propre garniture. On fabrique autant de gâteaux qu'on veut avec un seul moule.

01 Classe, objet, attributs, méthodes

- Une **classe** définit un nouveau type.
- Un **objet** (ou **instance**) est une valeur de ce type.
- Les **attributs** sont les données de l'objet.
- Les **méthodes** sont les fonctions qui agissent sur l'objet.

```
class Compte:
    def __init__(self, titulaire, solde=0):
        self.titulaire = titulaire
        self.solde = solde

    def deposer(self, montant):
        self.solde = self.solde + montant

    def retirer(self, montant):
        if montant > self.solde:
            return False
        self.solde = self.solde - montant
        return True

c = Compte("Léa", 50)
```

© Astrolab Cours · astrolabcours.fr/nsi/terminale/programmation-objet/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

```
c.deposer(20)
print(c.solde)   # 70
```

Le constructeur et self

`__init__` est le **constructeur** : il est appelé à la création de l'objet (`Compte("Léa", 50)`) et initialise les attributs.

`self` désigne l'objet sur lequel la méthode travaille. Écrire `c.deposer(20)` revient à appeler `Compte.deposer(c, 20)`.

Interface et encapsulation

L'**interface** d'une classe, c'est ce qu'on a le droit d'utiliser : ici `deposer` et `retirer`. L'**implémentation**, c'est la façon dont c'est codé à l'intérieur. En passant par les méthodes plutôt qu'en modifiant `c.solde` directement, on garde les règles (pas de retrait supérieur au solde) : c'est l'**encapsulation**.

Exemple guidé

Écrire une classe `Personnage` qui peut subir des dégâts et dire s'il est encore en vie.

```
class Personnage:
    def __init__(self, nom, vie):
        self.nom = nom
        self.vie = vie

    def subir(self, degats):
        self.vie = max(0, self.vie - degats)

    def est_vivant(self):
        return self.vie > 0

hero = Personnage("Ana", 30)
hero.subir(12)
hero.vie          # 18
hero.est_vivant()  # True
```

1. `__init__` crée les attributs de **chaque** objet : Ana a sa propre vie, distincte de celle des autres.
2. `subir` modifie l'objet sur lequel on l'appelle, grâce à `self`. Le `max` empêche la vie de passer sous zéro.
3. Le reste du jeu appelle `subir` et `est_vivant` sans toucher directement à `vie` : c'est l'interface.

Retour au problème

Chaque personnage devient un objet qui porte son nom, sa vie et sa position. Cent personnages, c'est une liste de cent objets, et plus aucun risque de décaler les indices.

Les pièges

- Oublier `self` dans la définition d'une méthode, ou oublier `self.` devant un attribut (on crée alors une

simple variable locale).
- Confondre la classe `Compte` et l'objet `c`.
- `__init__` s'écrit avec deux tirets bas de chaque côté.

© Astrolab Cours · astrolabcours.fr/nsi/terminale/programmation-objet/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits