

Programmation orientée objet en Python

Fiche de synthèse

① VOCABULAIRE

La **programmation orientée objet** réunit dans un même objet des données et les opérations qui s'y appliquent.

Définition 1 (Objet et classe) Un **objet** regroupe des données, ses **attributs**, et des fonctions qui agissent sur lui, ses **méthodes**. Une **classe** décrit une sorte d'objets, avec leurs attributs et leurs méthodes, et sert à les fabriquer : chaque objet fabriqué est une **instance** de la classe.

② DÉFINIR UNE CLASSE

```
class Point:
    """Point du plan."""

    def __init__(self, abscisse, ordonnee):
        self.x = abscisse
        self.y = ordonnee

    def translater(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
```

- Le nom d'une classe commence par une majuscule.
- Le premier paramètre de chaque méthode, nommé `self` par convention, reçoit l'objet auquel on applique la méthode.
- La méthode `__init__`, le **constructeur**, crée les attributs de chaque objet dès sa création. Sans elle, un objet n'a aucun attribut et `translater` échouerait sur une erreur `AttributeError`.

③ UTILISER DES OBJETS

```
p = Point(12, 6)          # appelle __init__
p.translater(1, 2)        # self reçoit p
print(f"{p.x} {p.y}")    # affiche 13 8
```

On appelle la classe comme une fonction pour créer une instance. La **notation pointée** donne accès aux attributs (`p.x`) et aux méthodes (`p.translater(1, 2)`). Chaque objet a ses propres valeurs : ce sont des **attributs d'instance**.

④ INTERFACE ET ENCAPSULATION

Définition 2 (Interface) L'**interface** d'une classe est l'ensemble des méthodes que l'utilisateur peut appeler, avec leur spécification. On utilise la classe à travers elle, sans lire le code des méthodes.

Définition 3 (Encapsulation) L'**encapsulation** consiste à ne lire et ne modifier les attributs qu'à travers les méthodes de l'interface, pour que l'objet reste dans un état cohérent.

```
class CompteBancaire:
    def __init__(self, titulaire):
        self.titulaire = titulaire
        self._solde = 0          # attribut interne

    def deposter(self, montant):
        assert montant > 0
        self._solde = self._solde + montant

    def retirer(self, montant):
        assert 0 < montant <= self._solde
        self._solde = self._solde - montant

    def solde(self):
        return self._solde
```

Par convention, un nom qui commence par un tiret bas, comme `_solde`, appartient à l'**implémentation** : on ne l'utilise pas hors de la classe. Python ne l'interdit pas, mais l'implémentation peut alors changer sans modifier les programmes qui utilisent la classe.

⑤ MÉTHODES SPÉCIALES

En Python, toutes les valeurs sont des objets : une liste est une instance de la classe `list`, et `L.append(4)` appelle une méthode. Les **méthodes spéciales**, encadrées par deux tirets bas, sont appelées par Python lui-même quand on utilise une fonction ou un opérateur.

On écrit	Python appelle
<code>Point(12, 6)</code>	<code>__init__(self, ...)</code>
<code>print(p)</code>	<code>__str__(self)</code>
<code>repr(p)</code>	<code>__repr__(self)</code>
<code>p == q</code>	<code>__eq__(self, other)</code>
<code>p + q</code>	<code>__add__(self, other)</code>
<code>len(p)</code>	<code>__len__(self)</code>

```
def __str__(self):
    return f"({self.x}, {self.y})"

def __add__(self, other):
    return Point(self.x + other.x,
                 self.y + other.y)

def __eq__(self, other):
    return (self.x == other.x
            and self.y == other.y)
```

Avec ces méthodes ajoutées à la classe `Point`, `print(Point(12, 6) + Point(3, 4))` affiche `(15, 10)`, et `Point(1, 2) == Point(1, 2)` vaut `True`. Sans `__eq__`, `==` vérifierait seulement s'il s'agit du même objet et donnerait `False`.

Remarque 1. `__str__` et `__repr__` renvoient une chaîne avec `return`; c'est `print` qui l'affiche. Pour afficher une liste de points, Python appelle `__repr__` sur chaque élément.

À retenir.

- Une classe décrit et fabrique des objets; un objet est une instance de sa classe.
- `__init__` crée les attributs; `self` désigne l'objet sur lequel la méthode agit.
- On accède aux attributs et aux méthodes par la notation pointée.
- L'interface est ce que l'utilisateur doit connaître; l'encapsulation protège l'état de l'objet.
- Les méthodes spéciales définissent l'affichage, les opérateurs et les comparaisons des objets.