2. Exemple pratique 1 : Approche inspirée des jeux vidéo

Objectif : Créer des personnages de jeu vidéo avec différentes caractéristiques et comportements.

Dans cet exemple pratique, nous allons créer une **classe** `Personnage` qui représente un personnage de jeu vidéo avec des attributs comme des points de vie et une force. Ensuite, nous allons créer deux personnages (un héros et un monstre) et simuler une interaction de combat entre eux en utilisant des méthodes pour attaquer et afficher leur état. Le but est de comprendre comment les objets (instances de classes) interagissent entre eux et comment les méthodes modifient les attributs de ces objets.

Étape 1 : Création d'une classe `Personnage`

Voici le code qui définit la classe `Personnage`. Cette classe contient trois méthodes principales : l'initialisation du personnage (avec des attributs comme le nom, les points de vie et la force), une méthode pour attaquer un autre personnage, et une méthode pour vérifier si le personnage est toujours vivant.

```
class Personnage:
    def __init__(self, nom, points_de_vie, force):
        """
        Initialisation du personnage avec son nom, ses points de vie et sa
force.
        """
        self.nom = nom
        self.points_de_vie = points_de_vie
        self.force = force

    def attaquer(self, autre_personnage):
        """
        Le personnage attaque un autre personnage, réduisant ses points de vie
        en fonction de sa force.
        """
        autre_personnage.points_de_vie -= self.force
        print(f"{self.nom} attaque {autre_personnage.nom} et lui inflige
{self.force} dégâts.")

    def est_vivant(self):
        """
        Vérifie si le personnage est toujours vivant (points de vie > 0).
        """
        return self.points_de_vie > 0

    def afficher_etat(self):
        """
        Affiche l'état actuel du personnage (ses points de vie restants ou
        un message indiquant qu'il est mort).
        """
        if self.est_vivant():
            print(f"{self.nom} a encore {self.points_de_vie} points de vie.")
        else:
            print(f"{self.nom} est mort.")
```

Explication du code :

- La méthode `__init__` est appelée automatiquement lorsqu'on crée un personnage. Elle prend trois paramètres : `nom`, `points_de_vie`, et `force`, qui sont stockés en tant qu'attributs de l'objet.
    - `self.nom` : Le nom du personnage.
    - `self.points_de_vie` : Les points de vie actuels du personnage.
    - `self.force` : La force d'attaque du personnage (dégâts infligés lors d'une attaque).
- La méthode `attaquer` permet à un personnage d'attaquer un autre personnage. Elle diminue les points de vie de l'autre personnage de la valeur de `self.force`.
- La méthode `est_vivant` retourne `True` si les points de vie du personnage sont supérieurs à zéro, sinon `False`.
- La méthode `afficher_etat` affiche le nombre de points de vie restants si le personnage est encore en vie, ou indique qu'il est mort.

Étape 2 : Création de plusieurs personnages

Maintenant que nous avons une classe `Personnage`, nous allons créer deux personnages : un **Héros** et un **Monstre**. Voici comment on peut créer ces personnages et les faire interagir.

```
# Création de deux personnages
heros = Personnage("Héros", 100, 15)  # Le héros a 100 points de vie et une
force de 15
monstre = Personnage("Monstre", 80, 10)  # Le monstre a 80 points de vie et une
force de 10

# Simulation d'une attaque du héros contre le monstre
heros.attaquer(monstre)

# Affichage de l'état du monstre après l'attaque
monstre.afficher_etat()
```

Explication du code :

1. **Création d'objets** : Nous créons deux instances de la classe `Personnage` :
    - `heros` avec 100 points de vie et une force de 15.
    - `monstre` avec 80 points de vie et une force de 10.
2. **Interaction** : Le héros attaque le monstre en appelant la méthode `attaquer`. Cela réduit les points de vie du monstre de 15 points.
3. **Affichage** : Après l'attaque, la méthode `afficher_etat` est appelée pour afficher le nombre de points de vie restants du monstre.

Sortie attendue après cette étape :

```
Héros attaque Monstre et lui inflige 15 dégâts.
Monstre a encore 65 points de vie.
```

Étape 3 : Boucle de jeu simple

Maintenant, nous allons créer une boucle pour simuler un combat entre le héros et le monstre jusqu'à ce que l'un des deux personnages meure. La boucle continue tant que les deux personnages sont vivants.

```
# Simuler un combat jusqu'à la mort d'un des personnages
while heros.est_vivant() and monstre.est_vivant():
    # Le héros attaque le monstre
    heros.attaquer(monstre)
    monstre.afficher_etat()

    # Si le monstre est toujours vivant, il riposte
    if monstre.est_vivant():
        monstre.attaquer(heros)
        heros.afficher_etat()
```

Explication du code :

- **Boucle de combat** : La boucle `while` continue tant que les deux personnages sont vivants (vérifié via la méthode `est_vivant`).
- **Attaques successives** :
    - Le héros attaque le monstre en premier.
    - Si le monstre survit à l'attaque, il riposte en attaquant le héros.
- La boucle se termine lorsque l'un des personnages meurt (c'est-à-dire lorsque ses points de vie tombent à zéro).

Sortie attendue (par exemple) :

```
Héros attaque Monstre et lui inflige 15 dégâts.
Monstre a encore 65 points de vie.
Monstre attaque Héros et lui inflige 10 dégâts.
Héros a encore 90 points de vie.
Héros attaque Monstre et lui inflige 15 dégâts.
Monstre a encore 50 points de vie.
Monstre attaque Héros et lui inflige 10 dégâts.
Héros a encore 80 points de vie.
...
Monstre attaque Héros et lui inflige 10 dégâts.
Héros a encore 10 points de vie.
Héros attaque Monstre et lui inflige 15 dégâts.
Monstre est mort.
```

Dans cet exemple, le combat continue jusqu'à ce que le monstre meure.

Objectif atteint

À travers ces trois étapes, nous avons mis en pratique les concepts suivants :

- **Classes et Objets** : Nous avons créé des objets (personnages) à partir d'une classe.
- **Attributs et Méthodes** : Nous avons défini et manipulé les attributs (nom, points de vie, force) via des méthodes (attaquer, afficher_etat).
- **Interaction entre objets** : Le héros et le monstre interagissent dans un combat, montrant comment des objets peuvent modifier les attributs d'autres objets.

Extensions possibles :

Pour aller plus loin, vous pourriez :

- Ajouter d'autres attributs, comme la défense ou la magie.
- Créer d'autres types de personnages (ex. : `Guerrier`, `Mage`) en utilisant l'héritage.
- Mettre en place un système de niveaux où chaque attaque donne de l'expérience au personnage.

Ce premier exemple est une excellente introduction à la POO appliquée dans des scénarios pratiques comme les jeux vidéo.