Programmation objet

Fiche Exercices n°2 - Corrigés

Exercice 1

Correction :

```
class Voiture:
    def __init__(self, couleur, marque, modele, puissance):
        # Attributs d'instance
        self.couleur = couleur
        self.marque = marque
        self.modele = modele
        self.puissance = puissance

    def afficher_details(self):
        print(f"Voiture: {self.marque} {self.modele}, Couleur: {self.couleur}, Puissance:
{self.puissance} chevaux")

class Parking:
    # Attribut de classe (partagé entre toutes les instances de Voiture)
    nombre_voitures = 0

    def __init__(self):
        # On pourrait aussi avoir des attributs d'instance pour chaque parking s'il y en avait
plusieurs
        self.voitures = []

    def ajouter_voiture(self, voiture):
        self.voitures.append(voiture)
        Parking.nombre_voitures += 1  # Incrémenter l'attribut de classe
        print(f"{voiture.marque} {voiture.modele} a été ajoutée au parking.")

    def retirer_voiture(self, voiture):
        if voiture in self.voitures:
            self.voitures.remove(voiture)
            Parking.nombre_voitures -= 1  # Décrémenter l'attribut de classe
            print(f"{voiture.marque} {voiture.modele} a été retirée du parking.")
        else:
            print("Cette voiture n'est pas dans le parking.")

    def afficher_nombre_voitures():
        print(f"Nombre total de voitures dans le parking : {Parking.nombre_voitures}")
```

Exercice 2

Correction :

```
class Personne:
    def __init__(self, nom, prenom, age, ville):
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.ville = ville

    def afficher_details(self):
        print(f"Personne: {self.prenom} {self.nom}, Âge: {self.age}, Ville: {self.ville}")

    # Implémentation de la méthode __eq__ pour comparer uniquement nom et prénom
    def __eq__(self, other):
        if not isinstance(other, Personne):
            return NotImplemented
        return self.nom == other.nom and self.prenom == other.prenom

# Exemple d'utilisation

# Créer des instances de Personne
personne_1 = Personne("Dupont", "Marie", 30, "Paris")
personne_2 = Personne("Dupont", "Marie", 25, "Lyon")
personne_3 = Personne("Durand", "Paul", 40, "Marseille")

# Comparaison des personnes
if personne_1 == personne_2:
    print("Les personnes 1 et 2 sont considérées comme égales.")
else:
    print("Les personnes 1 et 2 sont différentes.")

if personne_1 == personne_3:
    print("Les personnes 1 et 3 sont considérées comme égales.")
else:
    print("Les personnes 1 et 3 sont différentes.")
```

Exercice 3

Correction :

```
class Livre:
    def __init__(self, titre, auteur, nombre_pages):
        self.titre = titre
        self.auteur = auteur
        self.nombre_pages = nombre_pages

    def __str__(self):
        return f"Livre: {self.titre} par {self.auteur}, Nombre de pages: {self.nombre_pages}"

    # Implémentation de __eq__ pour vérifier l'égalité sur le nombre de pages
    def __eq__(self, other):
        if not isinstance(other, Livre):
            return NotImplemented
        return self.nombre_pages == other.nombre_pages

    # Implémentation de __gt__ pour comparer si un livre a plus de pages qu'un autre
    def __gt__(self, other):
        if not isinstance(other, Livre):
            return NotImplemented
        return self.nombre_pages > other.nombre_pages

    # Implémentation de __lt__ pour comparer si un livre a moins de pages qu'un autre
    def __lt__(self, other):
        if not isinstance(other, Livre):
            return NotImplemented
        return self.nombre_pages < other.nombre_pages

    # Implémentation de __ge__ pour comparer si un livre a au moins autant de pages qu'un
autre
    def __ge__(self, other):
        if not isinstance(other, Livre):
            return NotImplemented
        return self.nombre_pages >= other.nombre_pages

    # Implémentation de __le__ pour comparer si un livre a au plus autant de pages qu'un autre
    def __le__(self, other):
        if not isinstance(other, Livre):
            return NotImplemented
        return self.nombre_pages <= other.nombre_pages
```

```
# Exemple d'utilisation

# Créer des instances de Livre
livre_1 = Livre("Le Petit Prince", "Antoine de Saint-Exupéry", 96)
livre_2 = Livre("Harry Potter à l'école des sorciers", "J.K. Rowling", 320)
livre_3 = Livre("Le Hobbit", "J.R.R. Tolkien", 320)

# Comparaison des livres avec différents opérateurs
if livre_1 < livre_2:
    print(f"'{livre_1.titre}' a moins de pages que '{livre_2.titre}'")

if livre_2 == livre_3:
    print(f"'{livre_2.titre}' a le même nombre de pages que '{livre_3.titre}'")

if livre_3 >= livre_1:
    print(f"'{livre_3.titre}' a au moins autant de pages que '{livre_1.titre}'")

if livre_1 <= livre_2:
    print(f"'{livre_1.titre}' a au plus autant de pages que '{livre_2.titre}'")
```

Exercice 4

Correction :

```
class Livre:
    def __init__(self, titre, auteur):
        self.titre = titre
        self.auteur = auteur

    def __repr__(self):
        return f"'{self.titre}' par {self.auteur}"

class Bibliotheque:
    def __init__(self):
        self.livres = []  # Liste pour stocker les livres

    def ajouter_livre(self, livre):
        self.livres.append(livre)

    # Implémentation de __len__ pour retourner le nombre de livres
    def __len__(self):
        return len(self.livres)

    # Implémentation de __contains__ pour vérifier si un livre est dans la bibliothèque (sans any)
    def __contains__(self, titre):
        for livre in self.livres:
            if livre.titre == titre:
                return True
        return False

    # Implémentation de __getitem__ pour accéder à un livre par index
    def __getitem__(self, index):
        return self.livres[index]
```

```
# Exemple d'utilisation

# Créer des instances de Livre
livre_1 = Livre("Le Petit Prince", "Antoine de Saint-Exupéry")
livre_2 = Livre("Harry Potter à l'école des sorciers", "J.K. Rowling")
livre_3 = Livre("Le Hobbit", "J.R.R. Tolkien")

# Créer une instance de Bibliotheque et ajouter des livres
biblio = Bibliotheque()
biblio.ajouter_livre(livre_1)
biblio.ajouter_livre(livre_2)
biblio.ajouter_livre(livre_3)

# Utilisation de __len__ pour obtenir le nombre total de livres
print(f"Nombre de livres dans la bibliothèque : {len(biblio)}")

# Utilisation de __contains__ pour vérifier si un livre est dans la bibliothèque
if "Le Hobbit" in biblio:
    print("Le Hobbit est dans la bibliothèque.")
else:
    print("Le Hobbit n'est pas dans la bibliothèque.")

if "Les Misérables" in biblio:
    print("Les Misérables est dans la bibliothèque.")
else:
    print("Les Misérables n'est pas dans la bibliothèque.")

# Utilisation de __getitem__ pour accéder à un livre par index
print(f"Le premier livre dans la bibliothèque est : {biblio[0]}")
print(f"Le deuxième livre dans la bibliothèque est : {biblio[1]}")
```

Exercice 5

Correction

Ce que vous devez retenir :

- Copie superficielle (copy.copy()) : Cela crée une nouvelle instance de la playlist, mais les objets Chanson contenus dans la playlist sont toujours les mêmes (référencés). Si vous modifiez une chanson dans la copie superficielle, cela affectera également la playlist originale.

- Copie profonde (copy.deepcopy()) : Ici, une nouvelle playlist est créée, et toutes les chansons de cette playlist sont également copiées en profondeur. Ainsi, si vous modifiez une chanson dans la copie profonde, cela n'affectera pas la playlist originale.

```
import copy

# Classe Chanson
class Chanson:
    def __init__(self, titre, artiste):
        self.titre = titre
        self.artiste = artiste

    def __repr__(self):
        return f"'{self.titre}' par {self.artiste}"

# Classe Playlist
class Playlist:
    def __init__(self, nom):
        self.nom = nom
        self.chansons = []  # Liste pour stocker les chansons

    def ajouter_chanson(self, chanson):
        self.chansons.append(chanson)

    def afficher_playlist(self):
        print(f"Playlist: {self.nom}")
        for chanson in self.chansons:
            print(f"  {chanson}")
```

```
# Exemple d'utilisation

# Créer des chansons
chanson_1 = Chanson("Imagine", "John Lennon")
chanson_2 = Chanson("Bohemian Rhapsody", "Queen")
chanson_3 = Chanson("Hotel California", "Eagles")

# Créer une playlist et ajouter des chansons
playlist_1 = Playlist("Rock Classics")
playlist_1.ajouter_chanson(chanson_1)
playlist_1.ajouter_chanson(chanson_2)
playlist_1.ajouter_chanson(chanson_3)

# Afficher la playlist originale
playlist_1.afficher_playlist()

# Créer une copie superficielle de la playlist
playlist_copie_superficielle = copy.copy(playlist_1)

# Créer une copie profonde de la playlist
playlist_copie_profonde = copy.deepcopy(playlist_1)

# Modifions le titre d'une chanson dans la copie superficielle
playlist_copie_superficielle.chansons[0].titre = "Imagine (Remastered)"

# Modifions aussi le titre d'une chanson dans la copie profonde
playlist_copie_profonde.chansons[1].titre = "Bohemian Rhapsody (Live)"

# Afficher les playlists après les modifications
print("\n--- Après modification ---")
print("Playlist originale :")
playlist_1.afficher_playlist()

print("Copie superficielle :")
playlist_copie_superficielle.afficher_playlist()

print("Copie profonde :")
playlist_copie_profonde.afficher_playlist()
```