Exercice : Programmation orientée objet - Parcours et relation de Chasles

Objectifs et consignes

Objectifs pédagogiques :
- Mettre en œuvre les principes de la programmation orientée objet en Python.
- Concevoir des classes représentant des objets mathématiques (point, vecteur, chemin).
- Appliquer la relation de Chasles pour modéliser un déplacement global.

1. Classe Point

Classe Point :
- Attributs : x, y (coordonnées entières).
- Méthodes :
  - __init__, __sub__ (soustraction de deux points → vecteur),
  - __add__ (addition point + vecteur → nouveau point),
  - __repr__.

2. Classe Vecteur

Classe Vecteur :
- Attributs : dx, dy (composantes du déplacement).
- Méthodes :
  - __init__, __add__ (somme de vecteurs), __repr__.

3. Classe Chemin

Classe Chemin :
- Attributs : liste de points.
- Méthodes :
  - ajouter(point) : ajoute un point,
  - vecteurs() : liste des vecteurs entre points consécutifs,
  - chasles(i, j) : applique la relation de Chasles de i à j.

Exemple d'utilisation

```
chemin = Chemin()
chemin.ajouter(Point(10, 20))
chemin.ajouter(Point(13, 22))
chemin.ajouter(Point(16, 25))
chemin.ajouter(Point(20, 30))

print("Vecteurs :", chemin.vecteurs())
print("Chasles de 0 à 3 :", chemin.chasles(0, 3))
```

Corrigé (implémentation complète)

```
class Point:
    def __init__(self, x, y):
        self.x = x
```

```
        self.y = y

    def __sub__(self, other):
        return Vecteur(self.x - other.x, self.y - other.y)

    def __add__(self, vecteur):
        return Point(self.x + vecteur.dx, self.y + vecteur.dy)

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

class Vecteur:
    def __init__(self, dx, dy):
        self.dx = dx
        self.dy = dy

    def __add__(self, other):
        return Vecteur(self.dx + other.dx, self.dy + other.dy)

    def __repr__(self):
        return f"Vecteur({self.dx}, {self.dy})"

class Chemin:
    def __init__(self):
        self.points = []

    def ajouter(self, point):
        self.points.append(point)

    def vecteurs(self):
        return [self.points[i+1] - self.points[i] for i in range(len(self.points) - 1)]

    def chasles(self, i, j):
        if i < 0 or j >= len(self.points) or i >= j:
            raise ValueError("Indices invalides")
        vects = [self.points[k+1] - self.points[k] for k in range(i, j)]
        resultat = vects[0]
        for v in vects[1:]:
            resultat += v
        return resultat
```