UNIVERSITÉ RENNES 2

Programmation Orientée Objet

Xavier André & Romain Tavenard

1 Rappel : organisation de votre code

Pour ce TD, vous créerez un nouveau fichier `td_poo.py`. Dans ce fichier, votre code sera organisé de la manière suivante :

```
# Imports
from abc import ABC, abstractmethod, abstractproperty

# Classes et Fonctions
class [...]

# Tests
[...]
```

Notamment, vous définirez vos classes fonctions en début de fichier et les appels seront listés en fin de fichier. De cette manière, vous pourrez, d'une question à l'autre, réutiliser les classes et fonctions déjà codées au besoin.

2 La classe `Point`

Soit la classe suivante :

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x, self.y = x, y

    def __str__(self):
        return f'Point(x={self.x},y={self.y})'
```

1. Copiez-collez ce code et ajoutez à cette classe une méthode qui calcule la distance entre deux points fournis en paramètres. Doit-il s'agir d'une méthode d'instance, d'une méthode de classe ou d'une méthode statique ?

3 Création d'une nouvelle classe

3.1 La classe `Intervalle`

Définissez une classe `Intervalle` telle que le code suivant fonctionne comme attendu (c'est-à-dire que l'on rentre dans le `if` si et seulement si `a` est compris entre 10 et 20 inclus) :

```
1 if a in Intervalle(10, 20):
2     # [...]
```

Vous aurez pour cela besoin de définir la méthode spéciale `__contains__(self, v)` qui retourne `True` si `v` est "contenu dans" l'intervalle et `False` sinon.

3.2 La classe `Fraction`

1. Définissez une classe `Fraction` qui permette de représenter une fraction rationnelle. Créez un constructeur qui possède les caractéristiques suivantes :
   — Gestion de valeurs par défaut : numérateur et dénominateur initialisés à 1 ;
   — Interdiction d'instancier une fraction ayant un dénominateur nul ;
   — Définition de trois attributs d'instance :
     — `num` : Valeur absolue du numérateur ;
     — `den` : Valeur absolue du dénominateur ;
     — `signe` : Signe de la fraction (+1 ou -1).
2. Définissez la méthode spéciale `__str__()`, permettant d'afficher la fraction.
   Exemple d'affichage : `(-5/10)`
3. Définissez les méthodes de surcharge d'opérateurs suivantes :
   — `__neg__(self)` retourne la fraction opposée ;
   — `__add__(self, other)` retourne la fraction somme ;
   — `__sub__(self, other)` retourne la fraction différence ;
   — `__mul__(self, other)` retourne la fraction produit.

4 Héritage, surcharge et polymorphisme

4.1 Compte bancaire simple

Nous nous intéressons ici à un compte simple caractérisé par un solde exprimé en Euros qui peut être positif ou négatif.

1. Créez une classe `CompteSimple` respectant les caractéristiques ci-dessus.
2. Surchargez la méthode `__str__()` afin d'obtenir l'affichage suivant :

   > Le solde du compte est de XXX Euro(s).

3. Créez une méthode `enregistrerOperation()` qui permet de créditer le compte ou de le débiter.
4. Testez cette classe.

4.2 Compte bancaire courant

Une banque conserve pour chaque compte l'historique des opérations qui le concernent (on se limite ici aux opérations de crédits et de débits). On souhaite modéliser un tel compte qu'on appelle compte courant. En plus des méthodes d'un compte simple, un compte courant offre les méthodes suivantes :

— `afficherReleve` : affiche l'ensemble des opérations effectuées ;
— `afficherReleveCredits` : affiche seulement les opérations de crédit ;
— `afficherReleveDebits` : affiche seulement les opérations de débit.

Pour représenter l'historique, on utilisera une liste. Pour enregistrer une opération, on conservera simplement le montant de l'opération (Crédit : positif ; Débit : négatif).

1. Créez la classe `CompteCourant` correspondant aux spécifications ci-dessus.
2. Testez cette classe.
3. Ajoutez des attributs calculés `releveCredits` et `releveDebits` et mettez à jour les méthodes `afficherReleveCredits` et `afficherReleveDebits` pour qu'elles reposent sur ces attributs.

UNIVERSITÉ RENNES 2

5 Classe abstraite

On reprend maintenant l'exemple de la classe `Intervalle` définie plus haut. On souhaite être capable de prendre en compte deux types d'intervalles : les intervalles ouverts (pour lesquels les bornes ne sont pas incluses dans l'intervalles) et les intervalles fermés.

1. Définissez une classe `IntervalleAbstrait` qui hérite de la classe `ABC` et contient deux méthodes :
   — une méthode `__init__` qui prend en entrée les deux bornes de l'intervalle,
   — une méthode `__str__`,
   — une méthode abstraite `__contains__` qui indique si une valeur passée en argument est considérée comme faisant partie de l'intervalle ou non.
2. Définissez une classe `IntervalleOuvert` qui hérite de `IntervalleAbstrait` en ne redéfinissant que la ou les méthodes strictement nécessaire(s).
3. Définissez une classe `IntervalleFerme` qui hérite de `IntervalleAbstrait` en ne redéfinissant que la ou les méthodes strictement nécessaire(s).