TP 7 – API de gestion de comptes et tests avec PyTest

Dans ce TP, nous revenons sur l'API de gestion de comptes que nous avons implémenté dans un précédent TP. Nous allons ici donner une version orientée objet et "pythonique" de l'API. Nous allons aussi proposer des tests systématiquement en utilisant `pytest`, un des framework de test les plus utilisés en Python.

Exercice 1. Pipenv, environnement virtuels et installation

En Python, il existe le concept d'environnement virtuel. Il s'agit d'un environnement Python complet mais isolé du reste du système. Cela permet d'installer des versions de librairies sans toucher à l'intégriter du système. En général, on créer un environnement virtuel par projet. La création d'un environnement virtuel est souvent pénible, il existe des wrappers permettant d'aider à leurs gestion. La dernière en date et en passe de devenir un stardard est `pipenv`. Pour installer `pipenv`, nous allons utiliser `pip`, le gestionnaire de paquets pour python.

1. Installer `pipenv` en utilisant `pip` avec la commande suivante :
`pip3 install --user pipenv` (il est important d'utiliser `pip3` ici si Python 3 n'est pas la version par défaut).

Pour créer un environnement virtuel et installer `pytest`, il suffit de se placer dans le répertoire du projet Python et de rentrer la commander suivante :
`pipenv --three install --dev pytest`.

2. Créer un répertoire `test-python` (que nous allons supprimer par la suite) et créez un environnement virtuel pour celui-ci.

3. Activer l'environnement virtuel en utilisant la commande `pipenv shell`

4. Dans votre répertoire `test-python`, créez un répertoire `tests` qui accueillera vos tests et à l'intérieur, un fichier `__init__.py` et un fichier `test_premier.py`. Attention, le nommage est important, `pytest` travaille par conventions, il est donc nécessaire que les tests et les noms de fichiers commencent par `test_`. Dans votre fichier de test, inscrivez le code suivant.

```
def test_myfirsttest():
  a = 4
  assert a == 4
```

5. Lancer le test en tappant `pytest`. Essayez ensuite de faire échouer le test.

6. Une fois fait, quitter l'environnement virtuel avec CTRL-D (ou `exit`) et supprimer l'environnement virtuel en utilisant la commande suivante dans le projet de test : `pipenv --rm`

Exercice 2. API de gestion de comptes

Dans cette première partie, nous allons faire un peu de conception orientée objet. Les contraintes que nous avons sont presque les mêmes que celles du TP précédent. Nous avons des comptes qui sont caractérisés par un numéro d'identification unique, un utilisateur le possédant (une simple chaine de caractère ici) et un montant. Chaque compte appartient maintenant à une banque en particulier.

1. Proposez le diagramme de classe très simple exprimant ce modèle. Votre diagramme doit avoir au moins deux entitées.

Dans un premier temps, nous allons nous concentrer uniquement sur la gestion des comptes utilisateurs, pas sur celui de la banque.

2. Créer un nouveau projet et initialiser un environnement virtuel dans celui-ci.

3. Écrire les tests pour la création d'un compte.

4. Écrire le code associé à la création du type compte et à la création d'instances.

Maintenant qu'il est possible de créer des comptes, il est nécessaire de fournir les méthodes nécessaires pour ajouter et retirer de l'argent.

5. Écrire les tests pour l'ajout d'argent à un compte. Quels sont les cas limites ?

6. Écrire la méthode d'ajout d'argent à un compte.
7. Écrire les tests pour le retrait d'argent à un compte. Quels sont les cas limites ?
8. Écrire la méthode de retrait d'argent à un compte.
Nous allons rajouter aussi la possibilité d'additionner des comptes pour donner le montant total. Nous allons ici redéfinir l'opération `+` sur la classe que nous avons créés.
9. Écrire les tests pour une méthode d'addition de deux comptes.
10. Écrire le code associé à cette addition en surchargeant la dunder methode `__add__(self, other)`.
11. Il est possible d'utiliser des dunder méthodes pour fournir un moyen plus transparent pour l'ajout et le retrait d'argent en redéfinissant les opérations `+=` et `-=`. Cherchez les dunder methodes qui sont associées à ces opérateurs et surchargez les (*Indication :* ces deux méthodes doivent retourner l'objet qui a été modifié).

Exercice 3. Gestion de la banque et de la liste des comptes
Maintenant que nous avons la gestion des comptes, nous allons rajouter la gestion de la banque. Une banque est associé à un compte et, à partir d'un compte, il est possible de remonter jusqu'à une banque.
1. Écrire les tests pour la création d'une banque. Une banque doit-elle posséder obligatoirement un compte ? Écrire le code associé à la création du type banque et d'une instance de banque.
2. Modifier le type compte pour qu'il puisse garder l'information de la banque à qui il appartient. Est-il nécessaire de modifier les tests existants sur les comptes à la suite de cette modification pour pouvoir les exécuter à nouveau ?
3. Écrire les tests d'une méthode de création de compte depuis une banque et le code associé.
Nous allons maintenant modifier la banque pour réutiliser les opérateurs et structures de python, (encore une fois en implémentant des dunder méthodes sur la classe banque).
4. Écrire les tests pour vérifier si un compte existe dans une banque (en supposant que nous avons un compte noté `moncompte` avec l'identifiant `1324` et une banque notée `mabanque`, il doit être possible d'écrire `moncompte in mabanque` ou `1324 in mabanque` et d'avoir le même résultat).
5. Écrire la méthode associé dans la classe banque en surcharchant la bonne dunder methode.
6. Nous voulons maintenant que la banque soit itérable, c'est-à-dire que l'on puisse utiliser les structure `for` sur elle. Chaque itération du `for` doit retourner un compte. Écrire les tests associés à une telle fonctionnalité, et implémenter un tel comportement dans la classe banque en surchargeant la méthode `__iter__(self)` qui doit renvoyer un itérateur sur la collection à traverser (*Indication :* la fonction `iter` permet de renvoyer un itérateur à partir d'une collection existante).
7. Nous voulons maintenant une méthode permettant d'accéder à un compte par rapport à un numéro de compte. Est-il possible d'utiliser une dunder méthode pour exposer cette fonctionnalité (pensez aux accès aux éléments d'un dictionnaire) ? Écrire les tests d'une telle fonctionnalité et l'implémentation associée.