Feedback – Mission 8

1 Soumissions reçues et remarques générales

Soumissions reçues :

- Noah et Kaan : partiellement correct
- Nizar et Alistaire : partiellement correct

Remarques générales :

- Si vous vérifiez dans le constructeur que les paramètres ont des valeurs correctes, il n'y a pas besoin de le refaire dans d'autres fonctions comme to_secondes()
- N'oubliez pas de gérer les éventuelles erreurs qui peuvent arriver pendant l'exécution de votre programme (par exemple, essayer de lire un fichier qui n'existe pas). Ce n'est pas parce que le sujet de la mission n'est plus la gestion des erreurs qu'il ne faut pas le faire.
- On ne peut pas mettre de "return False" dans un constructeur, ce n'est pas une bonne pratique. Si les arguments fournis ne sont pas du type attendu, il vaut mieux faire un try ... except que de retourner False.
- Attention à bien lire les spécifications des fonctions qui vous sont demandées.
- Quand vous écrivez les tests, pensez bien aux différents cas possibles et pas seulement aux plus évidents/courants.

2 Exemples de code

Pouvez-vous corriger la fonction suivante ?

```
def to_secondes(self):
    m = 60 * m
    h = self.heures
    h = 3600 * h
    return m + s + h
```

Quelle est l'erreur dans le code suivant ?

```
    def delta(self, d):
        """
        @pre:  d est une instance de la classe Duree
        @post: Retourne la différence en secondes entre cette durée (self)
            et la durée d passée en paramètre.
            Cette valeur renovoyée est positif si cette durée (self)
            est plus grand que la durée d, négatif sinon.
        Par exemple, si cette durée (self) est 8h 41m 25s (donc 31285 secondes)
        et la durée d est 0h 1m 25s, la valeur retournée est 31200.
        Inversement, si cette durée (self) est 0h 1m 25s et la durée
        d est 8h 41m 25s, alors la valeur retournée est -31200
        """
        if self.to_secondes() > d.to_secondes():
            return "{0}".format(self.to_secondes() - d.to_secondes())
        else:
            return "{0}".format(self.to_secondes() - d.to_secondes())
```

3 Questions de bilan final

3.1 Première question

Écrivez une classe **complète** `Employe` dont les instances représentent un employé d'entreprise. Un employé est caractérisé par son `nom` (un string) et son `salaire` (un entier positif). Il doit être possible de créer un nouvel employé avec un nom et un salaire et d'effectuer les opérations suivantes sur un objet `Employe` :

- obtenir le nom de l'employé ;
- obtenir le salaire de l'employé ;
- obtenir un texte descriptif représentant l'employé sous la forme "`nom : salaire`" ;
- augmenter le salaire de l'employé d'un certain pourcentage.

Après avoir défini cette classe `Employe` correctement, l'exécution du code suivant :

```
charles = Employe("Charles", 2500)
charles.augmente(10)
print(charles)
```

devrait imprimer `Charles : 2750.0`.

3.2 Seconde question

(a) Corrigez les erreurs. Maintenant, supposez qu'on ajoute la méthode d'instance suivante à la définition de la classe `Employe`. Le but de cette méthode est d'ajouter le nom de famille au nom de l'employé.

```
def ajout_nom_famille(nomfamille):
    nom += " " + nomfamille
```

Avec cette définition, le code suivant produirait une erreur :

```
charles.ajout_nom_famille("Pecheur")
print(charles)
```

Expliquez quelle erreur serait produit et corrigez la méthode ci-dessus afin qu'elle imprime correctement `Charles Pecheur : 2750.0`
Attention : il y a deux erreurs différentes à corriger dans la définition de méthode ci-dessus.

(b) Appels chaînés. Maintenant considérez l'instruction suivante. On s'attend à ce que le string `Kim Mens : 2640.0` soit imprimé à l'écran. Est-ce que ce serait le cas pour votre code ? *Plus spécifiquement, est-ce que votre code supporte des appels chaînés comme dans l'instruction suivante ?* Sinon, corrigez vos méthodes précédentes afin que cette instruction ait le comportement voulu.

```
print(Employe("Kim", 2400).augmente(10).ajout_nom_famille("Mens"))
# doit imprimer "Kim Mens : 2640.0"
```

3.3 Troisième question

Écrivez ensuite quelques instructions Python qui effectuent ces différentes opérations :

- créer deux objets `Employe` représentant les frères Pierre et Nicolas ayant le même salaire de `2410` euros ;
- applique une augmentation de salaire de 3% à Pierre ;
- affiche les descriptifs de Pierre et Nicolas à l'écran.

Représentez graphiquement les instances créées avant et après l'augmentation du salaire de Pierre.