

Exercice 1 : le compte en banque

Pour une banque, on souhaite créer une classe **Compte** correspondant à un compte bancaire. Un compte est défini par 2 attributs : `_solde` et `_titulaire`. Lors de la création d'un compte, il faut donner le nom du titulaire et le solde est mis à 0. Les méthodes sont les suivantes :

Méthode	Description
<code>__init__(self, titulaire)</code>	Crée un nouveau compte et met le solde à 0.
<code>solde(self)</code>	Renvoie le solde du compte.
<code>crediter(self, montant)</code>	Ajoute <code>montant</code> au solde.
<code>debiter(self, montant)</code>	Enlève <code>montant</code> au solde.
<code>est_positif(self)</code>	Renvoie un booléen indiquant si le solde du compte est positif.

QUESTION 1 : Donner la suite d'instructions permettant de créer un compte `c` dont le titulaire est "Bob", d'y verser 200€, de retirer 50 puis d'afficher le solde, avec la commande `print`.

QUESTION 2 : Compléter les méthodes de la classe :

```
class Compte:  
    def __init__(self, titulaire):  
  
        def solde(self):  
  
            def crediter(self, montant):  
  
                def debiter(self, montant):  
  
                    def est_positif(self):
```

QUESTION 3 : Écrire la fonction `transferer(compteA, compteB, montant)` qui transfère le montant du `compteA` sur le `compteB`. Vous pouvez utiliser les méthodes de la classe `Compte`, mais pas les attributs.

```
def transferer(cA, cB, m):
```

QUESTION 4 : On souhaite écrire une fonction `equilibrer(compteA, compteB)` qui transfère une partie de l'argent d'un des comptes sur l'autre afin que les deux soient positifs, si c'est possible et nécessaire. Sinon rien n'est fait. En cas de transfert, le compte qui reçoit l'argent est mis à 0.

- 1) Déterminer les soldes des comptes après l'appel de `equilibrer(compteA, compteB)`:
 - a) Avant: `compteA.solde()` = 200 et `compteB.solde()` = 200
Après: `compteA.solde()` = et `compteB.solde()` =
 - b) Avant: `compteA.solde()` = 200 et `compteB.solde()` = -100
Après: `compteA.solde()` = et `compteB.solde()` =
 - c) Avant: `compteA.solde()` = -50 et `compteB.solde()` = 200
Après: `compteA.solde()` = et `compteB.solde()` =
 - d) Avant: `compteA.solde()` = -20 et `compteB.solde()` = -60
Après: `compteA.solde()` = et `compteB.solde()` =
 - e) Avant: `compteA.solde()` = 50 et `compteB.solde()` = -60
Après: `compteA.solde()` = et `compteB.solde()` =
- 2) En utilisant `compteA.solde() + compteB.solde()`, donner une expression équivalente au fait qu'il est possible de faire l'équilibrage.
- 3) S'il y a assez d'argent pour faire l'opération, comment savoir s'il faut transférer l'argent sur `compteB`?
- 4) S'il y a assez d'argent sur `compteA.solde()` pour combler le découvert de `compteB.solde()`, compléter l'expression qui permet de transferer la bonne quantité d'argent sur le bon compte:
`transferer(. , ,)`
- 5) Écrire de code de la fonction:

```
def equilibrer(cA, cB):
```

Exercice 2 : la maison intelligente

Pour cette partie, on considère un système domotique permettant de vérifier et de contrôler à distance les portes, les fenêtres et le chauffage de la maison.

Voici les attributs, qui peuvent être modifiés, et méthodes des classes :

Classe Fenetre	Description
nom	Attribut, de type str , qui est le nom de la fenêtre.
position	Attribut, de type int , qui indique le pourcentage d'ouverture du store. 0 c'est fermé et 100 c'est ouvert.
__init__ (self , nom)	Constructeur de la classe. Met la position à 0.
Classe Porte	Description
fermee	Attribut, de type bool , qui indique si la porte est fermée.
__init__ (self)	Constructeur de la classe. Ferme la porte.
Classe Garage	Description
position	Attribut, de type int , qui indique le pourcentage d'ouverture de la porte de garage.
__init__ (self)	Constructeur de la classe. Met la position à 0.
Classe Chauffage	Description
thermostat	Attribut, de type float , qui indique la valeur du thermostat.
__init__ (self)	Constructeur de la classe. Met le thermostat à 20.
Classe Maison	Description
porte	Attribut, de type Porte.
garage	Attribut, de type Garage.
fenetres	Attribut, une liste de valeurs de type Fenetre.
chauffage	Attribut, de type Chauffage.
__init__ (self)	Constructeur de la classe.

QUESTION 1 :

1) Quelle est la commande qui permet de créer une porte, stockée dans une variable **porte**?

2) Quelle est la commande à utiliser pour créer une fenêtre s'appelant '**chambre**' stockée dans une variable **f_chambre**?

3) Compléter le constructeur de la classe Fenetre.

```
def __init__(self, nom):  
    self.nom =  
    self.position =
```

4) On a utilisé la commande suivante, où chacun des paramètres est un objet du type attendu :

```
maison = Maison(porte, garage, liste_fenetres, chauffage)
```

a) Quelle expression permet d'avoir accès à la porte de garage de **maison**?

- b) Quelle expression permet d'obtenir la valeur de l'attribut `fermee` de la porte de `maison`?

- c) Entourer l'instruction qui permet de fermer les stores à 50% pour la fenêtre d'indice 0 de `fenetres` de `maison`:

- `maison.fenetres[0].position = 50`
- `position(fenetre(0, maison)) = 50`
- `self.fenetres[0].position = 50`

QUESTION 2 : On se place dans la classe `Maison`. Compléter les méthodes suivantes.

- 1) La méthode `scenario_arrivee(self)` ouvre la porte du garage et met le thermostat à 20.

```
def scenario_arrivee(self):
```

- 2) La méthode `ouvrir_stores(self)` permet d'ouvrir au maximum tous les stores de la maison.

```
def ouvrir_stores(self):
```

- 3) La méthode `verif_fermeture(self)` renvoie un booléen qui est vrai si tous les stores des fenêtres, la porte d'entrée et la porte du garage sont fermés.

```
def verif_fermeture(self):  
    for f in self.fenetres:  
        if .....:  
            return .....  
    # On regarde la porte d'entrée et la porte du garage  
    return .....
```

- 4) La méthode `plus_fermee(self)` renvoie le nom de la fenêtre dont le store est le plus fermé. S'il y a plusieurs fenêtres à égalité, c'est la première de la liste qui est renvoyée. Vous ferez attention à prendre une valeur de `pos_min` suffisamment grand.

```
def plus_fermee(self):  
    pos_min = .....  
    fenetre_min = ""  
    for f in .....:  
        .....  
    return .....
```