NOM : ..... **PRÉNOM :** .....

Code python

```python
def qui_pour_lap(classe, seuil):
    """ {str:int} -> [str]
    Renvoie la liste des élèves ayant une moyenne inférieure ou égale à
    ↪ seuil. """
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
```

Code de la classe CompteBancaire

```
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
```

Question 1b. Instanciation des variables cpt1 et cpt2

```
.....
.....
```

État de la mémoire après l'instanciation de `cpt1` et `cpt2`.

Code python

```python
def plus_riche(banque):
    """ [CompteBancaire] -> CompteBancaire """
    montant_max = banque[0].fonds
    indice_max = 0
    for i in range(len(banque)):
        if ..... > montant_max:
            montant_max = .....
            indice_max = .....
    return .....
```

Code python

```python
def evolue(banque, nb_annees):
    """ [CompteBancaire], int -> None """
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
```

Exercice 1. Un professeur de mathématiques utilise python pour gérer informatiquement ses classes. En particulier, pour chacune de ses classes il a à sa disposition un dictionnaire `classe` structuré de la manière suivante :

- Les clés du dictionnaire sont les prénoms des élèves de sa classe. On suppose que ces prénoms sont uniques.
- La valeur associée à la clé `k` est la moyenne actuelle (sur 20) de l'élève `k`.

Écrire une fonction `qui_pour_lap` qui étant donné un dictionnaire `classe` représentant une des classes du professeur, et un entier `seuil` renvoie la liste constituée des prénoms des élèves de la classe dont la moyenne actuelle (sur 20) est inférieure ou égale à `seuil`.

Code python

```
classe = {"Tom": 14.25, "Zoe": 10, "Samy":9.75, "Lina": 7, "Wassym": 13}
print(qui_pour_lap(classe, 10))
```

Résultat

```
['Zoe', 'Samy', 'Lina']
```

Exercice 2. On souhaite écrire une classe python qui nous permette de modéliser de manière très simplifiée la gestion de comptes bancaires. Toutes les réponses aux questions seront écrites sur la feuille de réponse. En particulier, on prendra garde à écrire les réponses aux questions **au bon endroit** (veiller aux noms des cadres) afin que le code proposé soit syntaxiquement correct. Afin de gagner du temps, vous n'écrirez **aucune** docstring (elles ont déjà été écrites dans l'énoncé).

1. a. Écrire le code une classe `CompteBancaire` qui comporte les attributs suivants : `titulaire` (le nom et le prénom du titulaire du compte), `fonds` (l'argent disponible sur le compte, en euros).

b. Écrire le code python qui permet d'instancier deux variables de type `CompteBancaire` :

- `cpt1`, dont le titulaire est `"Florian Picard"`, et dont les fonds disponibles sont de 1€.
- `cpt2`, dont le titulaire est `"Jeff Bezos"`, et dont les fonds disponibles sont de 200 milliards d'euros.

c. Représenter par un schéma l'état de la mémoire après l'instanciation des variables `cpt1` et `cpt2`.

2. a. À chaque fois qu'une personne reçoit de l'argent, un message de remerciement contenant le nom et le prénom du titulaire du compte est automatiquement affiché.
Écrire le code d'une méthode `recevoir` de la classe `CompteBancaire` qui prend en argument un flottant positif `montant` (en euros) et qui ajoute aux `fonds` disponibles du compte `self` le `montant` spécifié, et affiche un message de remerciement de la part du titulaire de compte.

Code python

```
def recevoir(self, montant):
    """ CompteBancaire, float -> None
    montant > 0 """
    pass
```

Code python

```
print(cpt1.fonds)
cpt1.recevoir(1000)
print(cpt1.fonds)
```

Résultat

```
1
Florian Picard dit merci !
1001
```

b. Écrire une méthode `impots` de la classe `CompteBancaire` qui prend en argument un entier `pourcent` et qui diminue de `pourcent` % les fonds disponibles du compte `self`.

La méthode renverra le montant prélevé, en euros.

Code python

```python
def impots(self, pourcent):
    """ CompteBancaire, int -> float """
    pass
```

Code python

```python
print(cpt1.fonds)
taxe = cpt1.impots(10)
print(taxe)
print(cpt1.fonds)
```

Résultat

```
1001
100.1
900.9
```

c. Écrire une méthode `envoyer` de la classe `CompteBancaire` qui prend en argument un compte `other` et un flottant positif `montant` (exprimé en euros) et envoie `montant` euros au compte `other` :

- si `montant` est supérieur aux `fonds` disponible, on ne fait rien et on affiche `"Fonds insuffisants"` ;
- sinon : on diminue les `fonds` disponibles de `montant` et le compte `other` reçoit le `montant`.

Code python

```python
def envoyer(self, other, montant):
    """ CompteBancaire, CompteBancaire, float -> None
    montant > 0 """
    pass
```

Code python

```python
cpt1.envoyer(cpt2, 100000)
print(cpt1.fonds, cpt2.fonds)
cpt2.envoyer(cpt1, 100000)
print(cpt1.fonds, cpt2.fonds)
```

Résultat

```
Fonds insuffisants
900.9 200000000000
Jeff Bezos envoie 100000
Florian Picard dit merci !
100900.9 199999900000
```

3. Dans les questions suivantes, on appellera "banque" toute liste de comptes bancaires. Ainsi la variable `banque` aura pour type `[CompteBancaire]`.

Afin de mettre en place une plus grande redistribution des richesses, une nouvelle loi vient d'être promulguée. Chaque année, chaque banque devra prélever du compte le plus riche de la banque un impôt de 5%, et reverser la somme ainsi prélevée au compte le moins riche de la banque.

a. Compléter le code de la fonction `plus_riche` qui étant donné une variable `banque` (une liste de comptes) renvoie l'objet de type `CompteBanquaire` dont les fonds sont les plus importants.

b. On suppose écrite une fonction `moins_riche` qui étant donné une liste de comptes `banque` renvoie le compte ayant le moins de fonds de la liste.

En déduire une fonction `evolue` qui étant donné une liste de comptes `banque` et un entier `n_annees` simule l'évolution des comptes de la liste `banque` suite à la nouvelle loi pendant `n_annees` années.

Code python

```python
print(cpt1.fonds, cpt2.fonds)
evolue([cpt1, cpt2], 3)
print(cpt1.fonds, cpt2.fonds)
```

Résultat

```
100900.9 199999900000
Florian Picard dit merci !
Florian Picard dit merci !
Florian Picard dit merci !
28525086638.4 171474914262.5
```