À crédit, dans l'ordre et en bullant

INFORMATIQUE COMMUNE - Devoir n° 2 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

R Les parties A,B et C sont indépendantes. Il est **fortement** conseillé de les aborder dans l'ordre. Le langage Python est le seul langage informatique autorisé dans les réponses. On s'appliquera à bien respecter les indentations.

R Sauf mention contraire, pour répondre à une question, il est autorisé d'utiliser une fonction définie dans les questions qui précèdent la question.

A Crédit

Un individu emprunte un capital initial $C_0 = 10000$ € auprès de sa banque pour financer l'achat d'un bien de consommation. Le prêt est contracté au taux d'intérêt mensuel $t = 0,5\%$ et est remboursable par mensualités constantes $M = 300$ € à la fin de chaque mois.

Soit C_n le capital restant dû (ou le solde) au mois n. Selon cette définition, on a donc : $C_0 = 10\,000$. Le capital restant dû chaque mois se calcule à l'aide de la formule suivante :

$$C_{n+1} = (1+t)C_n - M \tag{1}$$

Les intérets mensuels dus pour l'échéance du mois $n+1$ valent donc tC_n.

A1. Écrire une fonction de signature `caprd_suivant(C: float, t: float, M: float)-> float` qui renvoie la valeur du capital restant dû le mois suivant en fonction du capital restant dû `C`, du taux d'emprunt `t` et du montant d'une mensualité `M`.

Solution :

```
1 def caprd_suivant(C: float, t: float, M: float) -> float:
2     return (1+t)*C - M
```

A2. Écrire une fonction de signature `duree(C: float,t: float, M: float)-> int` qui renvoie la durée de remboursement du prêt en nombres de mois.

Solution :

```
def duree(C: float, t: float, M: float):
    N = 0
    while C > 0:
        C = caprd_suivant(C, t, M)
        N += 1
    return N
```

A3. Écrire une fonction de signature `c_list(C: float,t: float,M: float)—> list[float]` qui renvoie la liste des capitaux restants dus tout au long de la durée du prêt. Par exemple, pour un prêt de 800 € à un taux de 0,5 % et des mensualités de 90€ on obtient [800, 710.34, 620.62, 530.88, 441.10, 351.29, 261.43, 171.54, 81.61].

Solution :

```
def c_list(C: float, t: float, M: float) —> list[float]:
    R = [C]
    while C > 0:
        C = caprd_suivant(C, t, M)
        R.append(C)
    R.pop()
    return R
```

A4. Comment peut-on déduire de la question précédente le montant de la dernière mensualité ? Donner une suite d'instructions en Python pour stocker cette valeur dans une variable.

Solution : C'est le dernier terme de la liste `R` générée par le fonction précédente.

```
R = c_list(C, t, M)
derniere_mensualite = R[—1]
```

A5. Écrire une fonction de signature `cout_total(C: float, t: float, M: float)—> float` qui renvoie le coût total du crédit, c'est-à-dire le montant total remboursé par l'emprunteur.

Solution :

```
def cout_total(C: float, t: float, M: float) —> float:
    N = duree(C, t, M)
    R = c_list(C, t, M)
    derniere_mensualite = R[—1]
    return M * (N — 1) + derniere_mensualite
```

On sait calculer tous ces éléments grâce à l'étude des suites arithmético-géométriques. Ainsi, on peut calculer la mensualité d'après le montant emprunté, le taux et la durée n du prêt.

$$M = C_0 \frac{t}{1-(1+t)^{-n}} \tag{2}$$

A6. Écrire une fonction de signature `mensualite(C: float, t: float, n: int)-> float` qui renvoie la mensualité à payer pour un montant emprunté `C`, un taux `t` et une durée `n`.

Solution :

```
def mensualite(C: float, t: float, n: int) -> float:
    return C * t / (1 - (1 + t) ** (-n))
```

A7. (bonus points) Écrire une fonction **récursive** de signature `caprd(C0: float, t: float, M: float, n: int)-> float` qui renvoie C_n.

Solution :

```
def caprd(C0: float, t: float, M: float, n: int) -> float:
    if n == 0:
        return C0
    else:
        return (1 + t) * caprd(C0, t, M, n - 1) - M
```

B Des pièces à assembler

Dans une usine d'assemblage, les N pièces nécessaires à la construction d'un produit π sont numérotées de 0 à $N-1$: ce numéro permet de savoir dans quel ordre assembler les pièces. Ces pièces sont fabriquées dans d'autres usines, sont receptionnées à l'entrée puis triées afin de pouvoir procéder au processus d'assemblage. Plusieurs pièces possédant le même numéro arrivent donc à l'usine, ce qui permet de construire un certain nombre de produits π.

B1. Écrire une fonction de signature `generer_une_piece(N: int)-> int` qui renvoie le numéro d'une pièce aléatoirement choisie entre $[\![0, N-1]\!]$. On pourra utiliser la fonction `randrange` en l'important correctement. On rappelle que `randrange(5)` renvoie un entier choisi aléatoirement entre 0 et 4.

Solution :

```
from random import randrange

def generer_une_piece(N: int) -> int:
    return randrange(N)
```

B2. Écrire une fonction de signature `receptionner(M: int, N: int)-> list[int]` qui crée une liste aléatoire de `M` pièces à assembler. Les pièces sont numérotées entre 0 et $N-1$.

Solution :

```
def receptionner(M: int, N: int) -> list[int]:
    P = []
    for i in range(M):
```

```
        P.append(generer_une_piece(N))
    return P
```

B3. Écrire une fonction de signature `v_min(T: list[int])-> int` qui renvoie la valeur minimale d'une liste.

Solution :

```
def v_min(T: list[int]) -> int:
    if T == []:
        return None
    else:
        m = T[0]
        for i in range(1, len(T)):
            if T[i] < m:
                m = T[i]
        return m
```

B4. Écrire une fonction de signature `v_max(T: list[int])-> int` qui renvoie la valeur maximale d'une liste.

Solution :

```
def v_max(T: list[int]) -> int:
    if T == []:
        return None
    else:
        m = T[0]
        for i in range(1, len(T)):
            if T[i] > m:
                m = T[i]
        return m
```

B5. Écrire une fonction de signature `denombrer(P: list[int])-> list[int]` qui renvoie la liste du nombre d'occurrences de chaque élément dans la liste `P`. Par exemple, `denombrer([1, 0, 0, 1, 0, 0, 2])` renvoie `[4, 2, 1]`.

Solution :

```
def denombrer(P: list[int]) -> list[int]:
    M = v_max(P)
    c = [0 for _ in range(M + 1)]
    for e in P:
        c[e] += 1
    return c
```

B6. Écrire une fonction de signature `max_assemblage(P: list[int], N: int)—> int` qui renvoie le nombre maximal de produits π que l'on peut construire grâce à la liste de pièces `P`.

Solution :

```
def max_assemblage(P: list[int], N: int) —> int:
    c = denombrer(P)
    if len(c) < N:
        return 0
    m = v_min(c)
    return m
```

B7. Écrire une fonction de signature `trier(P: list[int])—> list[int]` qui **crée une nouvelle liste** des pièces triées. On utilisera le tri par insertion.

Solution :

```
def trier(P: list[int]) —> None:
    T = []
    for e in P:
        T.append(e)
        j = len(T) — 1
        while T[j — 1] > e and j > 0:
            T[j] = T[j — 1]
            j —= 1
        T[j] = e
    return T
```

C Buller

Le tri bulle est un tri comparatif dont le principe est décrit par l'algorithme 1. En partant à chaque itération du début du tableau, on fait monter l'élément courant si l'élément suivant est plus petit. Ainsi, à chaque itération, le plus grand restant est positionné correctement à la fin du tableau.

Algorithme 1 Tri bulle

```
1 : Fonction FAIRE_MONTER(t, place)
2 :     pour j de 0 à place-1 répéter
3 :         si t[j] > t[j+1] alors
4 :             ÉCHANGER(t, j, j+1)
5 : Fonction TRI_BULLE(t)
6 :     n ← taille(t)
7 :     pour place de n-1 à 1 répéter
8 :         FAIRE_MONTER(t, place)          ▷ faire monter si le suivant est plus petit
```

C1. Sur le tableau `[5, 0, 3, 1, 6, 4, 9, 2]`, appliquer à la main l'algorithme du tri bulle en détaillant le résultat de chaque itération de boucle de la fonction TRI_BULLE sur le tableau.

Solution :

```
[5, 0, 3, 1, 6, 4, 9, 2]
0 [0, 3, 1, 5, 4, 6, 2, 9]
1 [0, 1, 3, 4, 5, 2, 6, 9]
2 [0, 1, 3, 4, 2, 5, 6, 9]
3 [0, 1, 3, 2, 4, 5, 6, 9]
4 [0, 1, 2, 3, 4, 5, 6, 9]
5 [0, 1, 2, 3, 4, 5, 6, 9]
6 [0, 1, 2, 3, 4, 5, 6, 9]
7 [0, 1, 2, 3, 4, 5, 6, 9]
```

C2. Écrire une fonction de prototype `echanger(t: list[int], i: int, j: int)—> None` qui implémente la fonction ÉCHANGER, c'est-à-dire qui échange la place de l'élément d'indice `i` avec celle de l'élément d'indice `j` dans une liste `t`.

Solution :

```
def echanger(t, i, j):
    tmp = t[i]
    t[i] = t[j]
    t[j]  = tmp

# ou bien

def echanger(t, i, j):
    t[i], t[j] = t[j], t[i]
```

C3. Écrire une fonction de signature `faire_monter(t: list[int], place: int)—> None` qui implémente la fonction homonyme de l'algorithme 1.

Solution :

```
def faire_monter(t, dernier):
    for j in range(dernier):
        if t[j] > t[j + 1]:
            echanger(t, j, j + 1)
```

C4. Écrire une fonction de signature `tri_bulle(t)` qui implémente le tri bulle d'un tableau d'entiers.

Solution :

```
def tri_bulle(t):
    for i in range(len(t)):
        faire_monter(t, len(t) - 1 - i)

def tri_bulle(t):
    for i in range(len(t) - 1, 0, -1):
```

```
7         faire_monter(t, i)
```

R S'il n'y a pas d'échanges lors d'un parcours des éléments qui restent à trier, on peut en conclure que tout le tableau est trié.

C5. Écrire une fonction de signature `opt_tri_bulle(t)` qui implémente le tri bulle optimisé, c'est-à-dire qui s'arrête dès qu'il n'y a plus d'échanges lors d'une itération.

Solution :

```
def opt_tri_bulle(t):
    trie = False
    i = len(t) - 1
    while i > 0 and not trie:
        trie = True
        for j in range(i):
            if t[j] > t[j + 1]:
                echanger(t, j, j + 1)
                trie = False
        i -= 1
```

On observe que les dernières itérations sont supprimées :

```
        [5, 0, 3, 1, 2, 6, 4, 9]
        7 [0, 3, 1, 2, 5, 4, 6, 9]
        6 [0, 1, 2, 3, 4, 5, 6, 9]
        5 [0, 1, 2, 3, 4, 5, 6, 9]
```

C6. Compléter les lignes marquées `# X` code ci-dessous. Quel est le résultat de cette fonction et à quoi faut-il l'appliquer ?

```
def mystery(t : list[int], elem : int) -> int :
    g = 0
    d = # 1
    while # 2:
        m = (d + g) // 2
        if t[m] == elem:
            return m
        elif # 3:
            g = m + 1
        else:
            d = m - 1
    return None
```

Solution :

```
    d = len(t) - 1
    while g <= d:
```

```
3      elif t[m] < elem:
```

Il s'agit de la recherche d'un élément dans une liste triée par dichotomie. La liste doit être triée. La fonction renvoie la position de l'élément ou `None` s'il n'y est pas.