

Nom :

Prénom :

Groupe :

## Avertissement

- La plupart des questions sont indépendantes.
- À chaque question, vous pouvez répondre, au choix, par un algorithme ou par un programme Python.
- Les indentations des fonctions écrites en Python doivent être **respectées**.
- Répondez directement sur cette feuille.

Question	Points	Score
Compréhension de fonction	3	
Somme des inverses	2	
Multiples de 9	7	
Tableaux	8	
Total:	20	

## Exercice 1 : Compréhension de fonction

(3 points)

La fonction `mystere(t,s)` donnée ci-dessous prend en paramètre deux tableaux `t` et `s`.

```
def mystere(t,s):
    lt = len(t)
    ls = len(s)
    if lt != ls:
        return False
    i = 0
    while i < lt and t[i] == s[ls-i-1]:
        i+=1
    return i==lt
```

1. (1 point) Que retournent les appels suivants ?
  - a. `mystere([], [])`
  - b. `mystere([1,2,3,4], [4,3,2,1])`
  - c. `mystere([1,2,5,4], [4,3,2,1])`

- d. En général, quel est le résultat d'un appel de la fonction `mystere`? Justifier brièvement.
2. (1 point) Évaluer la complexité en temps (meilleur des cas et pire des cas) de la fonction `mystere`. Justifier brièvement.
3. (1 point) Lors de l'appel `mystere([1,2,3,4], [4,3,2,1])`
- pour quelles valeurs de la variable `i` la comparaison `i < lt` est-elle effectuée?
  - pour quelles valeurs de la variable `i` la comparaison `t[i] == s[ls-i-1]` est-elle effectuée?
- Justifier ces réponses.

## Exercice 2 : Somme des inverses

(2 points)

La suite  $u_n$  est définie pour  $n \geq 1$  par la formule suivante :

$$u_n = \sum_{k=1}^n \frac{1}{k} = 1 + \frac{1}{2} + \dots + \frac{1}{n}.$$

1. (1/2 point) Écrire une définition par récurrence de cette suite.
2. (1 1/2 points) Écrire une fonction **réursive**
- `sommeInverses(n)`
- qui calcule  $u_n$  lorsque  $n \geq 1$ . Pour  $n \leq 0$ , la fonction retournera 0.

**Exercice 3 : Multiples de 9****(7 points)**

1. ( $\frac{1}{2}$  point) Écrire une fonction `sommeChiffres(n)` qui calcule la somme des chiffres du nombre entier `n`.

On veut écrire une fonction qui teste si un nombre est multiple de 9 en utilisant la propriété suivante :

*Un nombre est multiple de 9 si et seulement si la somme de ses chiffres est multiple de 9.*

Le principe est de répéter le calcul de la somme des chiffres jusqu'à obtenir un nombre d'un seul chiffre (rappel : 0 aussi est multiple de 9).

Exemple : on teste si 9565938 est multiple de 9 en calculant la somme de ses chiffres, qui est 45. On recommence : la somme des chiffres de 45 est 9. Le nombre 9565938 est donc multiple de 9.

2. (1 point) Écrire une fonction **non récursive** `estMultipleDe9(n)` qui renvoie `True` si l'entier `n` est multiple de 9 et `False` sinon, en utilisant la propriété indiquée. En particulier, elle devra utiliser la fonction `sommeChiffres` de la question 1, et ne pas utiliser les opérateurs (`/`, `//`, `%`, `*`).

3. (1 point) Écrire une version **récursive** `estMultipleDe9Rec(n)` de la fonction écrite en question 2.

4. ( $\frac{1}{2}$  points) Écrire une fonction `indiceDernierMultipleDe9(t)` qui, étant donné un tableau `t` d'entiers, retourne le dernier (le plus grand) indice de `t` où se trouve un multiple de 9, et retourne `None` si aucun multiple de 9 n'est présent dans `t`.

On demande un algorithme qui minimise le temps d'exécution dans le cas le plus favorable (c'est-à-dire quand le dernier élément du tableau est un multiple de 9).

5. (3 points) Écrire une fonction `supprimerPremierMultipleDe9(t)` qui supprime du tableau `t` d'entiers le premier élément qui est un multiple de 9. Vous pouvez utiliser les primitives de la bibliothèque `bibTableau.py` du cours.

#### Exercice 4 : Tableaux

(8 points)

On veut représenter un tableau `t` de nombres par un autre tableau `c` appelé *codage* de `t`. Les suites consécutives de valeurs identiques de `t` sont représentées dans `c` par deux nombres `r,v` où `r` est le nombre de répétitions de la valeur `v` dans une telle suite. Une valeur `v` de `t` qui ne se répète pas est donc représentée par `1,v`. Une valeur `v` qui se répète deux fois consécutivement est représentée par `2,v`, et ainsi de suite.

Par exemple, le codage de `t = [0,0,0,0,5,-2,-2,-2,0,0,0,0,0]` est `c = [4,0,1,5,3,-2,5,0]`. Un entier à un indice pair dans `c` (dans l'exemple 4, 1, 3 et 5) représente donc un nombre de répétitions consécutives d'une valeur de `t`.

1. (3 points) Écrire une fonction `decoder(c)` qui « décode » un tableau `c` de longueur paire, c'est-à-dire qui renvoie le tableau `t` dont `c` est le codage.

```
>>> decoder([4, 0, 1, 5, 3, -2, 5, 0])
[0, 0, 0, 0, 5, -2, -2, -2, 0, 0, 0, 0, 0]
```

2. (1 point) Quelle est la complexité de votre fonction `decoder`? Justifiez la réponse.

On veut écrire la fonction de codage qui prend en entrée un tableau `t` et renvoie son codage `c`.

3. (2 points) Écrire une fonction `longueurBloc(t, pos)` qui renvoie la longueur du plus long bloc de positions consécutives du tableau `t` commençant en position `pos`, et composé de cases consécutives qui contiennent toutes la valeur `t[pos]`. Par exemple

```
>>> t = [0,0,0,0,5,-2,-2,-2,0,0,0,0,0]
>>> longueurBloc(t, 0)
4
>>> longueurBloc(t, 1)
3
>>> longueurBloc(t, 4)
1
>>> longueurBloc(t, 8)
5
```

4. (2 points) En utilisant la fonction `longueurBloc`, écrire une fonction `codage(t)` qui renvoie le codage du tableau `t`. Vous pouvez utiliser les primitives de la bibliothèque `bibTableau.py` du cours.