⋆ Sauf si l'énoncé vous le demande explicitement, il n'est pas nécessaire de donner les signatures des fonctions.

⋆ Merci d'indiquer au début de votre copie si vous avez déjà fait de l'informatique avant d'arriver en MPSI. Si oui, indiquez dans quel cadre. La notation de ce DS est la même pour tout le monde, elle ne sera pas affectée par votre réponse. **En revanche, si vous ne donnez pas de réponse, vous perdrez 1 point.**

Élection présidentielle

1 Introduction

Ce sujet s'intéresse à une élection présidentielle à laquelle participent `n` $\geqslant 2$ candidats. En Python, chaque candidat est représenté par son prénom (de type `str`) et par un numéro `i` $\in [\![0, n-1]\!]$ (de type `int`). Tout au long de cet exercice, nous manipulerons une liste `C` de taille `n` telle que pour tout `i` $\in [\![0, n-1]\!]$, `C[i]` est le prénom du candidat numéro `i`. Par exemple, si `C = C0` où :

```
C0 = ["Jean-Luc", "Yannick", "Emmanuel", "Valérie", "Marine", "Éric"]
```

cela signifie que le numéro du candidat `"Emmanuel"` est le `2` et que celui de la candidate `"Marine"` est le `4`.

1. Quelle expression Python permet d'obtenir `n` à partir de `C` ?
2. (a) Écrire une fonction `appartient(L: list[str], x: str, i0: int) -> bool` qui :
   → Déclenche une erreur si `i0` est strictement négatif.
   → Renvoie `True` si `x` est présent dans `L` à un indice supérieur ou égal à `i0`.
   → Renvoie `False` sinon.

   Par exemple, « `appartient(C0, "Emmanuel", i0)` » :
   → Déclenche une erreur pour `i0` $\leqslant -1$.
   → Vaut `True` lorsque `i0` vaut `0`, `1` ou `2`.
   → Vaut `False` lorsque `i0` $\geqslant 3$.

   (b) En déduire une fonction `tousDiff(C: list[str]) -> bool`, qui renvoie `True` si les prénoms des candidats sont distincts deux à deux, et `False` sinon.

Si plusieurs candidats avec le même prénom se présentent, l'élection est annulée et personne n'est élu. À partir de maintenant, nous supposons que les candidats ont des prénoms différents.

2 Premier tour du vote

Lors du premier tour du scrutin, chaque électeur choisit un candidat, écrit son prénom sur un bulletin de vote et met le bulletin dans une urne. Soit `V` la liste des prénoms apparaissant dans l'urne. Par exemple, si `V = V0` avec :

```
V0 = ["Emmanuel", "Marine", "Superman",
      "Emmanuel", "", "Marine", "Emmanuel", "Jean-Luc"]
```

cela signifie que :

→ Trois personnes ont voté pour `"Emmanuel"`, deux personnes ont voté pour `"Marine"` et une personne a voté pour `"Jean-Luc"`.

→ Personne n'a voté pour `"Yannick"`, `"Valérie"` ou `"Éric"`.

→ Une personne a voté blanc (le vote `""`) et l'un des votes est nul (le vote pour `"Superman"`).

Lors du dépouillement, on comptabilise le nombre de voix pour chaque candidat. On obtient alors une liste « `R: list[int]` » telle que pour chaque `i` $\in [\![0, n-1]\!]$, l'entier `R[i]` est le nombre de voix pour le candidat numéro `i`. Dans l'exemple précédent, `R` vaut `[1, 0, 3, 0 , 2, 0]`. Étudions maintenant deux méthodes pour construire `R`.

Méthode 1.

3. (a) Écrire une fonction `nbOcc(V: list[str], s: str) -> int` qui renvoie le nombre d'occurrences de `s` dans `V` (le nombre d'occurrences est le nombre d'apparitions). Par exemple, dans `V0`, le nombre d'occurrences de `"Emmanuel"` est `3`, le nombre d'occurrences de `"Marine"` est `2` et le nombre d'occurrences de `"Yannick"` est `0`.
   (b) En déduire une fonction `makeR1(C: list[str], V: list[str]) -> list[int]` qui renvoie la liste `R`.

Méthode 2. Dans la méthode 1 implémentée à la question 3, la liste `V` est parcourue plusieurs fois (une fois pour chaque appel à la fonction `nbOcc`). Essayons maintenant de construire `R` en ne parcourant `V` qu'une seule fois.

4. (a) Écrire une fonction `nomToNum(C: list[str], s: str) -> int ou NoneType` qui renvoie le numéro du candidat dont le prénom est `s`. Si `s` n'est pas le prénom d'un candidat, votre fonction devra renvoyer `None`.
   (b) Écrire une fonction `makeR2(C: list[str], V: list[str]) -> list[int]` qui renvoie la liste `R`. La liste `V` ne devra être parcourue qu'une seule fois.

Dans la suite, on pourra utiliser au choix la fonction `makeR1` ou `makeR2`.

Résultats du premier tour. Un candidat est élu lors du premier tour s'il obtient la majorité absolue, c'est à dire strictement plus de 50% des voix exprimées (une voix exprimée est un vote qui n'est ni blanc ni nul). Si aucun candidat n'a obtenu la majorité absolue, les deux meilleurs candidats s'affrontent au second tour. Dans l'exemple précédent, il y a 6 voix exprimées et il faut au minimum 4 voix pour obtenir la majorité absolue, il y aura donc un second tour entre `"Emmanuel"` et `"Marine"`.

5. (a) Écrire une fonction `nbVotes` qui prend en entrée la liste `R` et renvoie le nombre de voix exprimées.
   (b) Écrire une fonction `majAbs` qui prend en entrée la liste `R` et renvoie le nombre minimum de voix pour obtenir la majorité absolue.
   (c) Donner les signatures des deux fonctions précédentes.
6. (a) Écrire une fonction `tour1(R: list[int]) -> int ou NoneType` qui renvoie le numéro du candidat ayant obtenu la majorité absolue. Dans le cas où aucun candidat n'a obtenu la majorité absolue, votre fonction renverra `None`.
   (b) Écrire une fonction `max2(R: list[int]) -> (int, int)` qui prend en entrée la liste `R`, et renvoie un couple d'entiers `(i1,i2)` où `i1` est le numéro du candidat ayant obtenu le plus de votes et `i2` est le numéro du candidat ayant obtenu le plus de votes après `i1`. Afin de traiter le cas où un troisième candidat a obtenu le même nombre de voix que `i2`, on demande que `i1` et `i2` soient les plus petits possibles. De plus, si `i1` et `i2` ont le même nombre de voix, vous devez faire en sorte que `i1 < i2`. On rappelle enfin qu'au début de l'exercice, le nombre de candidats a été supposé supérieur ou égal à 2.

3 Second tour du vote

Plaçons-nous dans le cas où aucun candidat n'a obtenu la majorité absolue au premier tour et notons `(i1,i2)` les numéros des deux candidats sélectionnés pour le second tour comme dans la question 6b. Lors de ce second tour, tous les électeurs ont la même stratégie : ils votent pour le candidat ayant le programme le plus proche du candidat pour lequel ils ont voté au premier tour. Il se trouve que la liste `C` est organisée de telle manière que si un électeur vote pour le candidat numéro $\texttt{i} \in [\![\texttt{0}, \texttt{n}-\texttt{1}]\!]$ lors du premier tour alors lors du second tour :

- Il vote pour le candidat numéro `i1` si $|\texttt{i}-\texttt{i1}| \leqslant |\texttt{i}-\texttt{i2}|$.
- Il vote pour le candidat numéro `i2` sinon.

De plus, une personne qui a voté blanc votera de nouveau blanc et une personne qui a voté nul votera de nouveau nul. Dans l'exemple précédent, une personne qui a voté `"Jean-Luc"`, `"Yannick"`, `"Emmanuel"` ou `"Valérie"` au premier tour votera `"Emmanuel"` au second tour et une personne qui a voté `"Marine"` ou `"Éric"` au premier tour votera pour `"Marine"` au second tour.

Finalement, le candidat élu est le candidat ayant obtenu le plus de voix au second tour. Dans le cas où les deux candidats ont le même nombre de voix, c'est le candidat avec le plus petit numéro qui l'emporte.

7. (a) À l'aide d'une instruction conditionnelle (c'est à dire d'un `if`), écrire une fonction

   `valAbs(a: int) -> int`

   qui renvoie la valeur absolue de `a`.

   (b) Écrire une fonction `tour2(R: list[int], i1: int, i2: int) -> int` qui renvoie le numéro du candidat vainqueur au second tour. Les entiers `i1` et `i2` sont ceux renvoyés par la fonction de la question 6b.

8. Écrire une fonction `elu(C: list[str], V: list[str]) -> str ou NoneType` qui prend en entrée les listes `C` et `V`, et renvoie le prénom de la personne élue. En particulier, votre fonction devra vérifier que tous les prénoms sont différents et si ce n'est pas le cas, elle devra renvoyer `None`.

4 Test des fonctions

Dans cette partie, le but est de tester les fonctions `makeR1` et `makeR2` sur des listes générées aléatoirement. Pour cela, on crée d'abord deux listes aléatoires `(C,R)`, puis on construit `V` à partir du couple `(C,R)` et pour finir on vérifie que les appels à `makeR1(C,V)` et `makeR2(C,V)` renvoient bien `R`. Voici des fonctions utiles pour la suite :

→ La fonction `randint` du module `random` prend en entrée deux entiers i_1, i_2 et renvoie un entier aléatoire de l'intervalle $[\![i_1; i_2]\!]$.

→ La fonction `chr` prend en entrée un entier et renvoie un caractère (c'est à dire une chaîne de caractères de taille 1). La fonction `ord` est la fonction réciproque de `chr`. Par exemple :

i	97	98	99	...	121	122
chr(i)	"a"	"b"	"c"	...	"y"	"z"

c	"a"	"b"	"c"	...	"y"	"z"
ord(c)	97	98	99	...	121	122

9. On souhaite pouvoir accéder à la fonction `randint` grâce à l'expression « `random.randint` ». Quelle étape préalable est nécessaire en Python ? Dans la suite, on suppose que cette étape préalable a été effectuée.

4.1 Génération de la liste `C`

Dans le but de générer aléatoirement la liste `C`, on écrit les fonctions suivantes :

```
def f(m):
    s = ""
    i1 = ord('a')
    i2 = ord('z')
    for _ in range(m):
        c = chr(random.randint(i1,i2))
        s = s + c
    return s
```

```
def g(n):
    C = []
    while len(C) < n:
        m = random.randint(4, 10)
        s = f(m)
        if not appartient(C, s, 0):
            C.append(s)
    return C
```

10. (a) Expliquer ce que renvoie le fonction `f` et en donner la signature.

    (b) Même question pour la fonction `g`.

4.2 Génération de la liste `R`

Soit $m \in \mathbb{N}^*$ et $s \in \mathbb{N}^*$ deux entiers fixés. On souhaite générer m entiers aléatoires $(a_1, a_2, \ldots, a_m) \in \mathbb{N}^m$ vérifiant $\sum_{i=1}^{m} a_i = s$. Pour cela on propose deux méthodes.

Méthode 1. Chaque a_i est choisi aléatoirement grâce à la fonction `random.randint` de la manière suivante :

→ On choisit a_1 dans l'intervalle $[\![0; s]\!]$.

→ On choisit a_2 dans l'intervalle $[\![0; s - a_1]\!]$.

→ On choisit a_3 dans l'intervalle $[\![0; s - a_1 - a_2]\!]$.

→ ...

→ On choisit a_{m-1} dans l'intervalle $[\![0; s - \sum_{i=1}^{m-2} a_i]\!]$.

→ On pose $a_m = s - \sum_{i=1}^{m-1} a_i$.

11. Écrire une fonction `alea1(m: int, s: int) -> list[int]` qui génère une liste `A` contenant les entiers a_i obtenus par la méthode 1. Vous pouvez supposer sans le vérifier que $m \in \mathbb{N}^*$ et $s \in \mathbb{N}^*$.

Méthode 2. Le problème avec la fonction `alea1` est qu'un entier situé au début de la liste aura plus de chances d'être grand qu'un entier situé en fin de liste. Par exemple, a_1 a environ 50% de chance de vérifier $a_1 \geqslant s/2$, et dans ce cas tous les autres éléments de la liste vérifient $a_i \leqslant s/2$. On propose donc une seconde méthode qui évite ce problème.

Afin de générer les entiers a_i, on construit un tableau contenant $(s + m - 1)$ cases dans lequel on place $m - 1$ croix. Par exemple, pour $m = 6$ et $s = 20$, on peut obtenir :

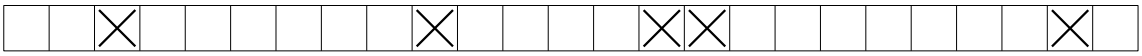


FIGURE 1

On définit alors :

→ a_1 le nombre de cases se trouvant avant la première croix.

→ a_2 le nombre de cases se trouvant entre la première et la deuxième croix.

→ a_3 le nombre de cases se trouvant entre la deuxième et la troisième croix.

→ ...

→ a_{m-1} le nombre de cases se trouvant entre la $(m-2)^{\text{ème}}$ et la $(m-1)^{\text{ème}}$ croix.

→ a_m le nombre de cases se trouvant après la $(m-1)^{\text{ème}}$ croix.

Pour la figure 1, on obtient :

$$a_1 = 2, \qquad a_2 = 6, \qquad a_3 = 4, \qquad a_4 = 0, \qquad a_5 = 7, \qquad a_6 = 1.$$

12. Écrire une fonction `alea2(m: int, s: int) -> list[int]` qui génère une liste `A` contenant les entiers a_i obtenus par la méthode 2. Vous pouvez supposer sans le vérifier que $m \in \mathbb{N}^*$ et $s \in \mathbb{N}^*$.

13. À l'aide de la fonction de la question 12, écrire une fonction

```
aleaR(n: int, s: int) -> (list[int], int, int)
```

qui renvoie un triplet `(R, v1, v2)` où `R` est une liste de taille `n` (plus tard, `v1` sera le nombre de votes blancs et `v2` le nombre de votes nuls). On fera en sorte que la somme des éléments de `R` plus `v1` plus `v2` soit égale à `s`.

4.3 Génération de la liste `V`

Le ***mélange de Fisher-Yates*** permet de permuter aléatoirement les éléments d'une liste `L`. En voici le principe :

→ Soit `n` la taille de `L`.

→ Pour `i` variant de `n-1` à `1` :

- Soit `j` un entier aléatoire de $[\![0\,;\,i]\!]$
- On échange les éléments `L[i]` et `L[j]`.

14. Écrire une fonction `melange` qui prend en entrée une liste `L` et lui applique le mélange de Fisher-Yates.

Supposons avoir à notre disposition les listes `C` et `R`, ainsi que deux entiers naturels `(v1,v2)`. Pour construire la liste `V`, on crée d'abord une liste `V0` telle que :

→ `V0` contient `R[i]` votes pour le candidat numéro `i` où `i` parcourt l'ensemble des numéros des candidats.

→ `V0` contient `v1` votes blancs et `v2` votes nuls. Un vote nul est une chaîne de caractères de taille 11 générée avec la fonction `f` de la partie 4.1.

L'ordre des éléments de `V0` est arbitraire (c'est à dire que vous pouvez choisir l'ordre qui vous arrange). La liste `V` est ensuite obtenue à partie de `V0` en lui appliquant un mélange de Fisher-Yates.

15. Écrire une fonction

```
aleaV(C: list[str], R: list[int], v1: int, v2: int) -> list[str]
```

qui génère la liste `V`.

4.4 Tests des fonctions

16. À l'aide des fonctions précédentes, écrire une fonction `test(n: int, s: int) -> bool` qui génère des listes `C, R, V`, et :

→ Renvoie `True` si `makeR1(C, V)` et `makeR2(C,V)` valent tous les deux `R`.

→ Renvoie `False` sinon.

L'entier `n` est le nombre de candidats et l'entier `s` est le nombre total de votes. On pourra tester l'égalité entre des listes `L1` et `L2` avec `L1 == L2`.

5 Nomination des ministres

Une fois élu, le président doit désigner son premier ministre qui se charge de nommer un gouvernement composé de ministres.

5.1 Premier ministre

Afin de choisir son premier ministre, le président établit une liste de personnes qu'il envisage de nommer à ce poste. À chacune de ces personnes est associé un score représentant la confiance que lui accorde le président. Dans la suite, ces informations seront stockées dans une liste « `P: list[str, float]` ». Par exemple, si `P = P0` où :

```
P0 = [('Édouard', 9.19), ('Jean', 9.07), ('Sébastien', 4.5), ('François', 5.58),
      ('Michel', 6.9), ('Gabriel', 7.31), ('Élisabeth', 8.18)]
```

cela signifie que 7 personnes sont pressenties pour devenir premier ministre et que le score de Gabriel est de `7.31`.

Au cours de son mandat, le président se voit contraint de changer plusieurs fois de premier ministre. Il dispose donc d'une liste « `A: list[str]` » contenant la liste des personnes qui ont déjà occupé le poste. Par exemple, si Édouard, Jean et Élisabeth sont les premiers ministres précédents, alors `A = A0` où :

```
A0 = ['Édouard', 'Jean', 'Élisabeth']
```

Étant données les listes `P` et `A` décrites ci-dessus, le président décide de nommer comme premier ministre la personne ayant le meilleur score dans `P`, mais qui n'apparaît pas dans `A`. Par exemple, avec `P = P0` et `A = A0`, le choix du président se portera sur Gabriel.

17. Dans cette question, vous pouvez supposer sans le vérifier que les noms présents dans `P` sont distincts deux à deux.
    (a) Écrire une fonction :
    `premM(P: list[str, float], A: list[str]) -> str ou NoneType`
    qui renvoie le nom du nouveau premier ministre. En cas d'égalité de score, on choisira le candidat dont le nom apparaît en premier dans `P`. La fonction renverra `None` si toutes les personnes de `P` apparaissent dans `A`.
    (b) En déduire une fonction `triP(P: list[str, float]) -> list[str]` qui renvoie la liste des personnes présentes dans `P` triées par ordre décroissant de score. Par exemple, si `P = P0`, on obtient :

```
['Édouard', 'Jean', 'Élisabeth', 'Gabriel', 'Michel', 'François', 'Sébastien']
```

5.2 Distribution des ministères

Le premier ministre est chargé de former un gouvernement composé de $n \in \mathbb{N}$ ministres. Il dispose pour cela d'une liste « `M: list[str]` » de $m \in \mathbb{N}$ personnes avec $m \geqslant n$. Sa mission est de construire une liste en choisissant n éléments dans `M`. Par exemple, si `n = 3`, `m = 4`, `M = M0` où :

```
M0 = ['Manuel', 'Gérald', 'Bruno', 'Rachida']
```

alors le premier ministre peut construire 24 listes différentes :

```
['Manuel', 'Gérald', 'Bruno']          ['Bruno', 'Manuel', 'Gérald']
['Manuel', 'Gérald', 'Rachida']        ['Bruno', 'Manuel', 'Rachida']
['Manuel', 'Bruno', 'Gérald']          ['Bruno', 'Gérald', 'Manuel']
['Manuel', 'Bruno', 'Rachida']         ['Bruno', 'Gérald', 'Rachida']
['Manuel', 'Rachida', 'Gérald']        ['Bruno', 'Rachida', 'Manuel']
['Manuel', 'Rachida', 'Bruno']         ['Bruno', 'Rachida', 'Gérald']
['Gérald', 'Manuel', 'Bruno']          ['Rachida', 'Manuel', 'Gérald']
['Gérald', 'Manuel', 'Rachida']        ['Rachida', 'Manuel', 'Bruno']
['Gérald', 'Bruno', 'Manuel']          ['Rachida', 'Gérald', 'Manuel']
['Gérald', 'Bruno', 'Rachida']         ['Rachida', 'Gérald', 'Bruno']
['Gérald', 'Rachida', 'Manuel']        ['Rachida', 'Bruno', 'Manuel']
['Gérald', 'Rachida', 'Bruno']         ['Rachida', 'Bruno', 'Gérald']
```

Notons en particulier que l'ordre des ministres dans la liste finale est importante. Par exemple les listes ci-dessous représentent deux gouvernements différents :

```
['Manuel', 'Gérald', 'Bruno']          ['Manuel', 'Bruno', 'Gérald']
```

18. Écrire une fonction `tousGouv(M: list[str], n: int) -> NoneType` qui affiche toutes les listes que peut construire le premier ministre.