Calculatrices interdites. Pensez à numéroter vos feuilles.
Sur votre copie, les questions doivent apparaître dans l'ordre du sujet.

Élection présidentielle

Voici quelques conseils :

- Les questions ne sont pas triées par ordre de difficulté : certaines questions faciles se trouvent après des questions difficiles. À la fin du temps imparti, il faut que vous ayez répondu à toutes les questions qui vous semblent faciles.
- Lorsque vous répondez à une question, pensez à utiliser les fonctions des questions précédentes. Même si vous ne réussissez pas à écrire une fonction, vous pouvez utiliser cette fonction dans la suite.
- Lorsque c'est utile, pensez à écrire des fonctions intermédiaires.
- Utiliser des fonctions Python non vues en cours vous pénalisera. Par exemple, vous n'avez pas le droit d'utiliser `del`, `split`, ...

Merci d'indiquer au début de votre feuille 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.**

1 Introduction

On s'intéresse à une élection présidentielle à laquelle participent $\texttt{n} \geqslant 2$ candidats. En Python, chaque candidat est représenté par son prénom (de type `str`) et par un numéro $\texttt{i} \in [\![\texttt{0}, \texttt{n - 1}]\!]$ (de type `int`). Tout au long de cet exercice, on manipulera une liste `C` de taille `n` telle que pour tout $\texttt{i} \in [\![\texttt{0}, \texttt{n - 1}]\!]$, `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 commande Python permet d'obtenir `n` à partir de `C` ?
2. (a) Écrire une fonction `appartient` qui prend en entrée trois paramètres : `L`, `x` et `i0` où `L` est une liste et `i0` un entier. Cette fonction doit :
   - Déclencher une erreur si `i0` est strictement négatif.
   - Renvoyer `True` si `x` est présent dans `L` à un indice supérieur ou égal à `i0`.
   - Renvoyer `False` sinon.

   Par exemple, `appartient(C0, "Emmanuel", i0)` :
   - Déclenche une erreur pour $\texttt{i0} \leqslant \texttt{-1}$.
   - Vaut `true` lorsque `i0` vaut `0`, `1` ou `2`.
   - Vaut `false` lorsque $\texttt{i0} \geqslant \texttt{3}$.

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

Si plusieurs candidats avec le même prénom se présentent, l'élection est annulée et personne n'est élu. Dans la suite, on supposera 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 qui apparaissent 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"`.
- Une personne a voté pour `"Jean-Luc"`.
- Personne n'a voté pour `"Yannick"`, `"Valérie"` ou `"Éric"`.

– Une personne a voté blanc (le vote `""`).

– L'un des votes est nul (le vote pour `"Superman"`).

Lors du dépouillement, on comptabilise le nombre de voix pour chaque candidat et on construit 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]`. On propose 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 `makeR(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`). On essaye 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 `makeRBis(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 `makeR` ou `makeRBis`.

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 `vainqueurTour1(R: list[int]) -> int ou NoneType` qui renvoie le numéro du candidat qui a obtenu la majorité absolue. Dans le cas où aucun candidat n'a obtenu la majorité absolue, votre fonction renverra `None`.

   (b) Écrire une fonction `deuxMeilleurs(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, on fera 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

On se place dans le cas où aucun candidat n'a obtenu la majorité absolue au premier tour et on note `i1` et `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 `i` $\in [\![0, n-1]\!]$ lors du premier tour alors lors du second tour :

– Il vote pour le candidat numéro `i1` si $|i - i1| \leqslant |i - i2|$.

– 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, écrire une fonction `valAbs(a: int) -> int` qui renvoie la valeur absolue de `a`.

(b) Écrire une fonction `vainqueurTour2(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 `vainqueurElection(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 `makeR` et `makeRBis` sur des listes générées aléatoirement. Pour cela, on créé 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 `makeR(C,V)` et `makeRBis(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. Quelle étape préalable est nécessaire en Python pour pouvoir utiliser la fonction `randint` ? 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 `listeAlea1(m: int, s: int) -> list[int]` qui génère une liste `A` contenant les entiers a_i obtenus par la méthode 1. On pourra supposer sans le vérifier que $m \in \mathbb{N}^*$ et $s \in \mathbb{N}^*$.

Méthode 2. Le problème avec la fonction `listeAlea1` 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 `listeAlea2(m: int, s: int) -> list[int]` qui génère une liste `A` contenant les entiers a_i obtenus par la méthode 2. On pourra 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 `genererR(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 créé d'abord une liste `V0` telle que :

- `V0` contient `R[i]` votes pour le candidat numéro `i` où `i` parcourt l'ensemble des numéros valides des candidats.
- `V0` contient `v1` votes blancs.
- `V0` contient `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 `genererV(C: list[str], R: list[str], 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 `makeR(C, V)` et `makeRBis(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, L2` avec `L1 == L2`.