- ⋆ 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.**

Tournoi et championnat de ping-pong

Les élèves de MPSI décident d'organiser un tournoi et un championnat de ping-pong. L'objectif de ce sujet est d'automatiser le déroulement de ces compétitions à l'aide de Python.

1 Préliminaires

Les fonctions écrites dans cette partie pourront être réutilisées librement dans la suite.

1. Écrire une fonction `compter(L: list[str], s: str) -> int` qui renvoie le nombre d'occurrences de `s` dans `L` (c'est à dire le nombre de fois que `s` apparaît dans `L`). Par exemple :

```
L = ["ping-pong", "tennis", "ping-pong", "pétanque", "ping-pong"]
print(compter([], "ping-pong")) # Affiche 0
print(compter(L, "ping-pong"))  # Affiche 3
print(compter(L, "tennis"))     # Affiche 1
print(compter(L, "Badminton"))  # Affiche 0
```

Suppression des doublons - version 1. On souhaite écrire une fonction

`doubl(L: list[str]) -> list[str]`

qui supprime les doublons de `L`. Par exemple :

```
L = ["ping-pong", "tennis", "ping-pong", "pétanque", "ping-pong"]
print(doubl(L)) # Affiche ['ping-pong', 'tennis', 'pétanque']
L = ["do", "do", "do", "ré", "mi", "ré", "do", "mi", "ré", "ré", "do"]
print(doubl(L)) # Affiche ['do', 'ré', 'mi']
```

2. À l'aide de la fonction `compter`, écrire une fonction `doublV1(L: list[str]) -> list[str]` qui supprime les doublons de `L`. Voir les exemples ci-dessus (l'ordre des éléments dans la liste en sortie n'est pas important).

Suppression des doublons - version 2. On s'intéresse maintenant à une seconde méthode plus efficace. L'idée est de trier les éléments de `L` par ordre alphabétique ce qui aura pour conséquence de regrouper les éléments en double. Il sera alors plus facile de les supprimer. Pour trier `L`, on utilisera la fonction « `sorted(L: list[str]) -> list[str]` » qui est déjà définie en Python. Par exemple :

```
L = ["ping-pong", "tennis", "ping-pong", "pétanque", "ping-pong"]
print(sorted(L)) # ['ping-pong', 'ping-pong', 'ping-pong', 'pétanque', 'tennis']
L = ["do", "do", "do", "ré", "mi", "ré", "do", "mi", "ré", "ré", "do"]
print(sorted(L)) # ['do', 'do', 'do', 'do', 'do', 'mi', 'mi', 'ré', 'ré', 'ré', 'ré']
```

3. Écrire une fonction `doublV2(L: list[str]) -> list[str]` qui supprime les doublons dans `L`. Votre fonction devra avoir la forme suivante :

```
def doublV2(L):
    M = sorted(L)
    # Dans la suite de la fonction, vous n'avez plus le droit d'utiliser L et
    # vous ne pouvez parcourir les éléments de M qu'une seule fois. Vous ne
    # devez pas non plus utiliser la fonction doublV1.
```

2 Inscriptions au tournoi de ping-pong

Les étudiants souhaitant participer au tournoi de ping-pong ont inscrit leurs prénoms sur une liste représentée en Python par une variable « `J: list[str]` ». Par exemple, avec `J1` et `J2` les listes définies dans l'annexe à la fin du sujet, si `J = J1` alors 17 élèves participent au tournoi et si `J = J2`, alors 26 élèves y participent.

4. À l'aide de la fonction `compter`, écrire une fonction `dist(J: list[str]) -> bool` qui renvoie `True` si les éléments de `J` sont distincts deux à deux, et `False` sinon. Par exemple, `dist(J1)` vaut `True` et `dist(J2)` vaut `False` (voir `J1` et `J2` en annexe).

Dans le cas où certains joueurs ont le même prénom, on souhaite les numéroter pour pouvoir les distinguer. Par exemple, si la liste des joueurs est `J2`, les Hugo seront numérotés entre 1 et 4, les Enzo entre 1 et 2, et les Nathan entre 1 et 2. Après numérotation des joueurs de `J2`, on obtient la liste `J3` de l'annexe.

5. Soit `elemMult` la fonction :

```
# L est une liste de chaînes de caractères.
def elemMult(L):
    return [s for s in L if compter(L, s) > 1]
```

   (a) Donner la signature de `elemMult`.

   (b) Donner les valeurs de `elemMult(J1)` et `elemMult(J2)` où `J1` et `J2` sont définis en annexe.

6. Écrire une fonction `numAux(J: list[str], s: str) -> list[str]` qui renvoie une copie de `J` dans laquelle les occurrences de `s` ont été numérotées. Par exemple, avec les listes de l'annexe, `numAux(J2, "Hugo")` vaut `J4`.

7. À l'aide des fonctions précédentes, écrire une fonction `num(J: list[str]) -> list[str]` qui renvoie une copie de `J` dans laquelle les éléments qui apparaissent au moins deux fois ont été numérotés. Par exemple, `num(J1)` vaut `J1` et `num(J2)` vaut `J3`.

Dans la suite du sujet, on pourra supposer sans le vérifier que les prénoms des joueurs sont distincts deux à deux.

3 Organisation du tournoi

3.1 Choix des matchs

Étant donnée la liste des joueurs « `J: list[str]` », les matchs à jouer sont stockés dans une autre liste « `M: list[str, str]` ». Par exemple, si `J = J1` et `M = M1` (voir annexe), cela signifie que le premier match oppose Diego et Gustave, que le deuxième oppose Sam et Soheil, etc ... Dans le cas où le nombre de joueurs est impair, l'un des joueurs n'a pas d'adversaire et le match correspondant est représenté par un couple `(s1, s2)` avec `s1 = ""` ou `s2 = ""` (c'est le cas de Sacha dans `M1`).

On décide de générer les matchs aléatoirement avec la procédure suivante :

→ Si le nombre de joueurs est impair, on ajoute dans `J` une chaîne de caractères vide.

→ On choisit au hasard dans `J` les deux personnes qui disputeront le premier match. Par exemple dans `M1`, les deux joueurs choisis pour le premier match sont Diego et Gustave.

→ On supprime ces deux personnes de `J`.

→ On réitère les deux étapes précédentes pour choisir le deuxième match, puis le troisième match et ainsi de suite jusqu'à ce que tout le monde participe à exactement un match.

À partir de maintenant, on considère que le module `random` a été importé :

```
import random
```

Rappelons que la fonction « `random.randint(a: int, b: int) -> int` » renvoie un entier aléatoire de l'intervalle ⟦`a`; `b`⟧

8. (a) Écrire une fonction `supprInd(J: list[str], k: int) -> list[str]` qui renvoie une copie de `J` dans laquelle l'élément d'indice `k` a été supprimé. Par exemple `supprInd(J1, 3)` vaut `J5` (voir l'annexe). Si `k` n'est pas un indice positif valide pour `J`, votre fonction déclenchera une erreur.
   (b) En déduire une fonction `makeM(J: list[str]) -> list[str, str]` qui construit la liste `M` évoquée au début de la partie 3.1. Bien sûr, vous devez suivre la procédure décrite ci-dessus.

3.2 Cohérence des résultats

Comme dans la partie précédente, on note « `M: list[str, str]` » le liste contenant les matchs à jouer. De plus, les résultats sont stockés dans une liste « `R: list[str]` » tel que `R[i]` contient le nom du gagnant du match `M[i]`. Par exemple avec les listes de l'annexe, si `M = M1` et `R = R1`, cela signifie que Gustave a gagné son match contre Diego, que Sam a gagné son match contre Soheil, etc ... Si un joueur n'a pas d'adversaire (c'est le cas de Sacha dans `M1`), alors il remporte nécessairement son match.

On dira que deux listes « `M: list[str, str]` » et « `R: list[str]` » sont "cohérentes" si :

→ Ces deux listes ont la même taille.

→ `R` ne contient pas la chaîne de caractères vide.

→ Pour tout indice `i` : `R[i]` est l'un des deux éléments du couple `M[i]`.

Par exemple, `M1` et `R1` sont cohérents, mais `M1` et `R2` ne le sont pas car Ilanka ne participe pas au match `M1[2]`.

9. Écrire une fonction `coh(M: list[str, str], R: list[str]) -> bool` qui renvoie `True` si `M` et `R` sont cohérentes et `False` sinon.

3.3 Simulation du tournoi

Dans le but de tester les fonctions précédentes, nous allons simuler le déroulement des matchs, éliminer les perdants, puis recommencer jusqu'à ce qu'il ne reste plus qu'un seul participant qui sera alors déclaré comme le gagnant du tournoi.

Deux conditions doivent être réunies pour qu'un joueur j gagne un match de ping-pong :

→ j doit avoir marqué au moins 11 points.

→ j doit avoir marqué au moins 2 points de plus que son adversaire.

Par exemple, un match ne peut pas se terminer sur le score de 11 à 10. En effet, même si le joueur 1 a marqué 11 points, il n'a qu'un seul point de plus que son adversaire. Dans cette situation, le match continue : si le joueur 1 remporte le prochain échange, le match se termine sur le score de 12 à 10, sinon le match continue jusqu'à ce que l'un des deux joueurs ait rempli les 2 conditions de victoire.

Dans les simulations qui suivent, on considère que lors d'un échange, chacun des joueurs a une chance sur deux de marquer le point.

10. (a) Écrire une fonction `match() -> (int, int)` qui simule un match et renvoie la score final. Bien sûr, vous devez suivre les explications ci-dessus.
    (b) En déduire une fonction `makeR(M: list[str, str]) -> list[str]` qui renvoie la liste `R` décrite au début de la partie 3.2. Le score de chaque match sera le résultat d'un appel à `match`.

Un tournoi de ping-pong se décompose alors en plusieurs manches. Lors de chaque manche, les matchs sont choisis au hasard avec la fonction `makeM` et les résultats sont générés avec `makeR`. Les joueurs ayant perdu leur match sont éliminés, puis on passe à la manche suivante. Le tournoi s'arrête lorsqu'il n'y a plus qu'un seul joueur en lice.

11. Écrire une fonction `tournoi(J: list[str]) -> str)` qui simule un tournoi et renvoie le prénom du vainqueur. Si `J` est vide, votre fonction déclenchera une erreur. Bien sûr, vous devez suivre les explications ci-dessus.

4 Championnat de ping-pong

Suite à l'engouement pour le tournoi de la partie 3, les étudiants de MPSI décident d'organiser un championnat. Dans un tournoi, certains élèves ne font qu'un seul match. À l'inverse dans un championnat, chaque élève rencontre une fois chacun des autres élèves.

4.1 Numérotation des joueurs

Soit $n \in \mathbb{N}$ le nombre joueurs. On décide d'indicer les joueurs de 0 à $n-1$ en respectant l'ordre donné par la liste `J`. Par exemple, si `J = J1` (voir annexe), alors le joueur d'indice 0 est Gustave, celui d'indice 1 est Soheil etc ...

12. (a) Écrire une fonction `conv1(J: list[str], L: list[int]) -> list[str]` qui renvoie une liste « `M: list[str]` » de même taille que `L` telle que `M[i]` est le prénom du joueur d'indice `L[i]`. On pourra supposer sans le vérifier que tous les éléments de `L` sont des indices valides pour `J`.
    (b) Écrire une fonction `conv2(J: list[str], L: list[str]) -> list[int]` qui renvoie une liste « `M: list[int]` » de même taille que `L` telle que `M[i]` est l'indice du joueur dont le prénom est `L[i]`. On pourra supposer sans le vérifier que tous les éléments de `L` appartiennent à `J`.

Lorsque tous les matchs du championnat sont terminés, on note « `S: list[int]` » la liste telle que `S[i]` est le nombre de matchs gagnés par le joueur d'indice `i`. Le ou les gagnants du championnat sont les étudiants ayant remporté le plus de matchs.

13. Écrire une fonction `championnat(J: list[str], S: list[int]) -> list[str]` qui renvoie la liste des prénoms des gagnants du championnat.

4.2 Choix des matchs

Dans cette partie, les joueurs sont représentés par leurs indices et non par leurs prénoms. On suppose que le nombre de joueurs $n \in \mathbb{N}$ est pair. Un championnat s'organise en plusieurs "rondes" numérotées de 1 à $n-1$. Lors de chaque ronde, chaque joueur participe à un et un seul match. Notre objectif est d'organiser les rondes.

Table de Berger. La première méthode pour organiser les rondes est d'utiliser une "table de Berger". Pour tout couple de joueurs $(i,j) \in [\![0, n-1]\!]^2$ tels que $i < j$, notons $r(i,j) \in [\![1, n-1]\!]$ le numéro de la ronde lors de laquelle se rencontrent i et j. Alors :

$\rightarrow$ Si $j \neq n-1$ et $i+j+1 < n$: $r(i,j) = i+j+1$.
$\rightarrow$ Si $j \neq n-1$ et $i+j+1 \geqslant n$: $r(i,j) = i+j-n+2$.
$\rightarrow$ Si $j = n-1$ et $2i+2 \leqslant n$: $r(i,j) = 2i+1$.
$\rightarrow$ Si $j = n-1$ et $2i+2 > n$: $r(i,j) = 2i+2-n$.

On admet qu'avec ces règles, tout joueur $k \in [\![0, n-1]\!]$ rencontre chaque autre joueur $\ell \in [\![0, n-1]\!] \setminus \{k\}$ une et une seule fois.

14. Écrire une fonction `berger(n: int, k: int) -> list[int]` qui renvoie la liste contenant les adversaires du joueur k de la ronde 1 à la ronde $n-1$. Votre fonction déclenchera une erreur si n est impair ou si k n'est pas l'indice d'un joueur. Votre fonction devra être la plus efficace possible. Par exemple, `berger(10, 7)` vaut `[2, 3, 4, 5, 6, 9, 8, 0, 1]` ce qui signifie que lorsque $n = 10$, le joueur $k = 7$ va rencontrer 2 lors de la ronde 1, puis 3 lors de la ronde 2, puis 4 lors de la ronde 3, ..., puis 1 lors de la ronde 9.

Méthode du ruban. La seconde méthode pour organiser les rondes est la "méthode du Ruban". Illustrons en le principe avec $n = 10$. Pour la ronde 1, on écrit les nombres de 0 à $n - 1$ sur un ruban composé de 2 lignes et $n/2$ colonnes. Pour les rondes suivantes, chaque entier $k \neq 0$ se trouvant sur le ruban est déplacé dans le sens des aiguilles d'une montre :

0	1	2	3	4
9	8	7	6	5

Ronde 1

0	9	1	2	3
8	7	6	5	4

Ronde 2

0	8	9	1	2
7	6	5	4	3

Ronde 3

...

Les matchs d'une ronde opposent alors les participants se trouvant sur la même colonne du ruban. Pour $n = 10$, on obtient :

Ronde 1 :	0 contre 9,	1 contre 8,	2 contre 7,	3 contre 6,	4 contre 5.
Ronde 2 :	0 contre 8,	9 contre 7,	1 contre 6,	2 contre 5,	3 contre 4.
Ronde 3 :	0 contre 7,	8 contre 6,	9 contre 5,	1 contre 4,	2 contre 3.

...

15. Écrire une fonction `ruban(n: int) -> NoneType` qui affiche dans la console les matchs à jouer selon la méthode du ruban. Votre fonction déclenchera une erreur si n est impair. Par exemple, pour $n = 10$ on obtient :

```
Ronde 1: 0-9, 1-8, 2-7, 3-6, 4-5,
Ronde 2: 0-8, 9-7, 1-6, 2-5, 3-4,
Ronde 3: 0-7, 8-6, 9-5, 1-4, 2-3,
Ronde 4: 0-6, 7-5, 8-4, 9-3, 1-2,
Ronde 5: 0-5, 6-4, 7-3, 8-2, 9-1,
Ronde 6: 0-4, 5-3, 6-2, 7-1, 8-9,
Ronde 7: 0-3, 4-2, 5-1, 6-9, 7-8,
Ronde 8: 0-2, 3-1, 4-9, 5-8, 6-7,
Ronde 9: 0-1, 2-9, 3-8, 4-7, 5-6,
```

5 Matchs en équipe

Les étudiants les plus motivés souhaitent former des équipes de 2 pour jouer des matchs en double. On utilise les mêmes notations que dans la partie 4.1 : les joueurs sont notés $0, 1, 2, \ldots, n-1$ avec $n \in \mathbb{N}$ un nombre pair. On appelle "répartition de n joueurs" une liste « `rep: list[int,int]` » de taille $n/2$ dont les éléments sont de la forme `(j1, j2)` avec `j1` et `j2` les membres d'une équipe. Par exemple, voici deux répartitions possibles pour $n = 6$:

```
rep1 = [(0, 4), (1, 3), (2, 5)]          rep2 = [(3, 1), (4, 0), (2, 5)]
```

Dans `rep1`, la première équipe est composée des joueurs 0 et 4, la deuxième équipe de 1 et 3 et la troisième de 2 et 5. De plus, on considère que `rep1` et `rep2` représentent la même répartition puisque chaque joueur a le même partenaire dans les deux listes.

16. Écrire une fonction `equipes(n: int) -> list[list[int,int]]` qui renvoie une liste contenant une et une seule fois chaque répartition de n joueurs. Votre fonction déclenchera une erreur si n est impair. Par exemple pour $n = 6$, on obtient :

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

Annexe

```
J1 = ["Gustave", "Soheil", "Sam", "Diego", "Réda",
      "Marius", "Joakim", "Khalil", "Timothé", "Erin",
      "Ilanka", "Sacha", "Bao", "Zoé", "Rafael", "Mélinée", "Armand"]

J2 = ["Iyass", "Loise", "Enzo", "Noé", "Hugo", "Romane", "Elisa",
      "Abdel-Moulla", "Hugo", "Nicolas", "Jeremy", "Robin", "Eliot",
      "Clara", "Guillaume", "Arthur", "Nathan", "Chloé", "Simon", "Hugo",
      "Enzo", "Martin", "Hugo", "Vivian", "Clément", "Nathan"]

J3 = ['Iyass', 'Loise', 'Enzo 1', 'Noé', 'Hugo 1', 'Romane', 'Elisa',
      'Abdel-Moulla', 'Hugo 2', 'Nicolas', 'Jeremy', 'Robin', 'Eliot',
      'Clara', 'Guillaume', 'Arthur', 'Nathan 1', 'Chloé', 'Simon', 'Hugo 3',
      'Enzo 2', 'Martin', 'Hugo 4', 'Vivian', 'Clément', 'Nathan 2']

J4 = ['Iyass', 'Loise', 'Enzo', 'Noé', 'Hugo 1', 'Romane', 'Elisa',
      'Abdel-Moulla', 'Hugo 2', 'Nicolas', 'Jeremy', 'Robin', 'Eliot',
      'Clara', 'Guillaume', 'Arthur', 'Nathan', 'Chloé', 'Simon', 'Hugo 3',
      'Enzo', 'Martin', 'Hugo 4', 'Vivian', 'Clément', 'Nathan']

J5 = ['Gustave', 'Soheil', 'Sam', 'Réda',
      'Marius', 'Joakim', 'Khalil', 'Timothé', 'Erin',
      'Ilanka', 'Sacha', 'Bao', 'Zoé', 'Rafael', 'Mélinée', 'Armand']

M1 = [('Diego', 'Gustave'), ('Sam', 'Soheil'), ('Bao', 'Zoé'),
      ('Ilanka', 'Armand'), ('', 'Sacha'), ('Erin', 'Marius'),
      ('Joakim', 'Timothé'), ('Réda', 'Mélinée'), ('Khalil', 'Rafael')]

R1 = ['Gustave', 'Sam', 'Zoé', 'Armand', 'Sacha',
      'Erin', 'Timothé', 'Mélinée', 'Rafael']

R2 = ['Gustave', 'Sam', 'Ilanka', 'Armand', 'Sacha',
      'Erin', 'Timothé', 'Mélinée', 'Rafael']
```