Bataille

Consignes

Les consignes suivantes **doivent** être respectées sous peine d'être pénalisé par des **points négatifs**.

Le code doit être clair.

- Évitez les lignes à rallonge. S'il faut introduire une variable intermédiaire, faîte le.
- Nommez vos variables avec du sens. Évitez toutefois des noms trop longs. Auquel cas, décrivez le rôle de la variable en commentaire.
- Ne surchargez pas de commentaires. Les commentaires utiles sont :
  - Description du rôle d'une variable
  - Description du rôle d'une boucle complexe.

Soignez la présentation de votre code.

- Marquez clairement les indentations (2 grands carreaux).
- Éviter les ratures trop nombreuses dans un code. Une ou deux, maximum.
- Écrivez votre instruction sur plusieurs lignes si elle est longue en ajoutant simplement une indentation après le retour à la ligne.

Respectez la syntaxe Python. - *0 à l'exercice si syntaxe inventée.*

- Tout ce qui n'est pas sur la fiche de syntaxe ne peut pas être utilisé.
- Les petites erreurs n'implique pas 0 (oublie de :, = au lieu de ==, …)
- Si vous avez une idée, mais que vous n'arrivez pas à l'écrire, n'inventez pas. Écrivez votre idée en commentaire.

Utilisation de `print` ou de `input` interdite - *0 à l'exercice sinon*

Pour nommer une fonction, utilisez le nom donné dans l'énoncé - *0 à l'exercice sinon*

I Algorithme Classiques

Écrivez les algorithmes ayant la spécification suivante :

Question 1. *Recherche dichotomique dans une liste triée*

Entrées : Une liste d'entiers triés dans l'ordre croissant et un entier `x`.
Sortie : `-1` si `x` n'est pas dans `L` ; l'indice de `x` dans `L` sinon.

Question 2. *Copie d'une matrice*

Entrées : m une matrice non vide.
Sortie : Une copie de m

II Bataille

On se propose de simuler un jeu de *Bataille*. La *Bataille* est un jeu de carte à deux joueurs dont le but est d'obtenir toutes les cartes de son adversaire. Les joueurs seront nommés Alice et Bob. Les cartes possèdent une valeur de 0 à 12 dans l'ordre suivant : 2, 3, 4, 5, 6, 7, 8, 9, 10, Valet, Dame, Roi, As, et une couleur parmi Cœur (♡), Carreau (♢), Trèfle (♣) et Pique (♠). Initialement les 52 cartes sont réparties aléatoirement en deux tas de taille égale. Lors d'un tour de jeux :

1. Alice *puis* Bob, placent au centre chacun la carte au-dessus de leur tas.
2. Si les cartes sont de valeurs égales, Alice *puis* Bob, rajoutent au centre une nouvelle carte du dessus de leur tas, puis recommencent l'étape 1.
3. Sinon, le joueur ayant placé la carte de valeur la plus grande remporte les cartes du centre et les placent sous son tas. C'est la fin du tour.
4. À tout moment du tour, si un joueur ne peut pas prendre une carte de son tas, l'autre joueur remporte les cartes du centre et les placent sous son tas. C'est la fin du tour.

Une partie consiste en une succession de tour. Et cela tant que les deux joueurs ont des cartes dans leur tas. Le premier joueur à avoir un tas vide perd la partie.

On définit dans l'environnement global, les variables (globales) suivantes :

```
s_valeurs = '23456789TVDRA'
s_couleurs = ("CO", "CA", "TR", "PI")
```

Question 3. *Valeur des cartes*

Les valeurs des cartes seront représentées par un caractère de la chaîne `s_valeurs = '23456789TVDRA'`. Les chiffres sont représentés par le caractère correspondant, 10 est représenté par `'T'`, le Valet par `'V'`, la Dame par `'D'` et le roi par `'R'`. La valeur d'une carte représentée par le caractère `c` correspond donc à l'indice de `c` dans la chaîne `s_valeurs`.

En utilisant une recherche linéaire dans la chaîne `s_valeurs` implémenter une fonction `valeur(c)` qui prend un caractère en entrée et renvoi la valeur de la carte associée à `c`, ou `-1` si `c` ne correspond à aucune carte.

Question 4. *Bataille*

Les couleurs des cartes seront représentées par une chaîne de caractère du quadruplé `s_couleurs`. Une *carte* est donc représenté par un couple de deux chaînes de caractère, sa *valeur* et sa *couleur, e.g.* `("R", "PI")` ou `("3", "CA")`.

Implémenter une fonction `bataille(c1,c2)` qui prend en entrée deux cartes et renvoie :

- 1 si la valeur de `c1` et plus grande que celle de `c2`
- 2 si la valeur de `c2` et plus grande que celle de `c1`
- 0 sinon

Question 5. *Mélange du paquet*

Le module `random` contient une fonction `shuffle` qui prend en entrée une liste et mélange aléatoirement ses éléments par effet de bord. `shuffle` ne *renvoie rien*.

Écrivez le code qui permet d'importer `shuffle` depuis le module `random` sans importer tout le module.

Puis implémentez une fonction `paquet()` qui ne prend pas d'entrées et renvoie une liste de toutes les cartes, mélangée aléatoirement à l'aide de `shuffle`.

Question 6. *Vérification du mélange*

On veut vérifier qu'un paquet mélangé n'a pas été truqué. On doit vérifier pour cela que le paquet contient 52 cartes toutes différentes.

Implémentez une fonction `verification(p)` qui prend en entrée un paquet mélangé aléatoirement et renvoie un booléen vrai si et seulement si `p` contient 52 éléments tous distincts.

Question 7. *Désérialisation*

Un paquet mélangé peut être représenté par une chaîne de caractère `s` de telle manière à ce que chaque groupe de 3 lettres corresponde à une carte dans l'ordre du paquet. Ainsi la chaîne :

`2COTPI4CADCA ... RTRACO`

représente le paquet :

```
[('2','CO'), ('T', 'PI'), ('4','CA'), ('D','CA'), ... ('R','TR'), ('A','CO')]
```

Implémentez une fonction `chargement(s)` qui prend en entrée une chaîne de caractère représentant un paquet mélangé et renvoie ce paquet. On supposera que la chaîne est bien formée et peut bien être convertie en un paquet mélangé de cartes. On vérifiera à la fin que le paquet n'est pas truqué à l'aide de la fonction `verification`. Si le paquet est truqué, la fonction renvoie `None`.

Question 8. *Distribution*

On considère un *paquet mélangé* de carte, `p`, représenté donc par une liste de 52 cartes (couples de chaînes) dans un ordre aléatoire. La *distribution* des cartes consiste en la répartition des cartes du paquet en deux listes, `tas_alice` et `tas_bob`. La distribution se fait de manière alternée, *i.e.* la première carte de `p` est ajoutée dans `tas_alice`, la deuxième dans `tas_bob`, la suivante dans `tas_alice` et ainsi de suite jusqu'à la 52è carte qui est ajouté dans `tas_bob`. Les cartes sont ajoutées à la fin du tas, (fin de la liste).

Implémentez une fonction `distribution(p)` qui prend en entrée un paquet mélangé aléatoirement et renvoie deux listes correspondant au tas d'Alice et Bob respectivement. La fonction commencera par une *assertion* qui vérifie si le paquet contient bien 52 cartes.

Question 9. *Tour de jeu*

On rappelle que le principe d'un tour de jeux est donné par les quatre points en introduction. Pour simuler un tour de jeux, on représentera le tas du centre par une liste de cartes, vide au début du tour. La carte *au-dessus* du tas d'un joueur est la *dernière* carte de la liste qui représente le tas du joueur.

Implémenter une fonction `tour(tas_alice, tas_bob)` qui prend en entrée les tas de chacun des deux joueurs et simule un tour de jeux complet puis renvoi. Le nouveau tas d'Alice et le nouveau tas de Bob après un tour complet de jeu. On autorise la modification par effet de bord des tas de départ.

Question 10. *Partie*

On rappelle qu'une partie consiste en la distribution d'un paquet de carte mélangé en deux tas, puis d'une succession de tours de jeu s'arrêtant lorsque l'un des deux tas est vide.

Implémenter une fonction `partie()` qui ne prend rien en entrée, crée un paquet de carte mélangé, distribue le paquet au deux joueurs, simule des tours de jeux jusqu'à ce que l'un des tas soit vide puis renvoie deux entiers `g` et `nb_tour` tels que

- `g` soit égal à 1 si Alice à gagner et 2 si Bob à gagner.
- `nb_tour` est le nombre de tours de jeu qu'il a fallu faire pour gagner.