# DEVOIR MAISON D'INFORMATIQUE 2
# Programmation d'un système de jeu de rôle

Ce devoir maison est facultatif et non noté, et doit être pris comme une opportunité pour s'entrainer. Vous pouvez déposer votre travail (réalisé sur machine) sur cahier de prepa avant le 4 mai. N'hésitez pas à me poser vos questions par mail (pompigne@crans.org) en cas de blocage.

## 1 Principe du système

Nous considérons dans ce sujet le jeu Buffon et Bastons, proposant d'incarner des personnages héroïques de fiction. Un tel personnage est principalement décrit par ses valeurs dans les caractéristiques suivantes : Force (F), Agilité (A), Ruse (R), Volonté (V). On représentera en Python un tel personnage par un dictionnaire associant aux clés `'F'`, `'A'`, `'R'` et `'V'` leur valeur. Par exemple, Imara est une barbare des steppes gelées décrite par la fiche suivante :

```
Imara = {'F' : 10, 'A' : 8, 'R' : 6, 'V' : 5}
```

Un défi opposé à un personnage est décrit par une chaîne de quatre caractères correspondant à des caractéristiques, en autorisant des doublons. Dans notre exemple, à la suite d'une aventure improbable, Imara est confrontée à un défi périlleux : un DS d'ITC, décrit par la chaîne `'RRRV'`. Pour déterminer avec quel niveau de réussite le défi est surmonté, on procède de la manière suivante :

- On tire 4 cartes au hasard dans un paquet de 52 cartes, contenant quatre cartes de chaque valeur de 1 à 13 ;
- On compare chaque i-ème carte avec la i-ème caractéristique du défi : si la valeur de la carte est inférieure ou égale à la valeur du personnage dans cette caractéristique, on compte un succès.
- Le nombre total de succès, entre 0 et 4, donne le niveau de réussite :
  — 4 succès : réussite totale (sur l'exemple du devoir, la copie d'Imara est parfaite) ;
  — 3 succès : réussite (Imara rend une excellente copie) ;
  — 2 succès : réussite partielle (Imara rend une copie convenable) ;
  — 1 succès : échec (Imara rend une copie médiocre) ;
  — 0 succès : échec total (le devoir est un désastre, Imara rend copie blanche).

Dans notre exemple, Imara tire quatre cartes de valeurs 8, 8, 2, 6, et les compare à ses valeurs selon la chaîne `'RRRV'`. Elle a 6 en Ruse et 5 en Volonté, effectue donc les comparaisons $8 \leq 6, 8 \leq 6, 2 \leq 6, 6 \leq 5$, et compte alors un seul succès : c'est un échec. Le même tirage sur un défi `'FFFA'`, comme se bagarrer avec un mammouth des steppes, aurait donné une réussite totale, Imara ayant privilégié l'aspect Bastons sur l'aspect Buffon du jeu.

1. Écrire une fonction `paquet` prenant en entrée deux entiers $n$ et $m$, et renvoyant une liste correspondant à un paquet de $n \times m$ cartes formé de $n$ fois la valeur 1, puis $n$ fois la valeur 2, et ainsi de suite jusqu'à $n$ fois la valeur $m$.

   Par exemple, `paquet(2,3)` doit renvoyer `[1, 1, 2, 2, 3, 3]`.

2. Pour mélanger le paquet de façon équiprobable, nous allons implémenter l'algorithme de Fisher-Yates, dont le pseudo-code est :

   ```
   soit n la longueur de la liste à mélanger
   pour i de 0 à n - 2 inclus :
       on tire un entier aléatoire j entre i et n-1 inclus
       on échange les éléments d'indices i et j
   ```

   Écrire une procédure `melange` implémentant cet algorithme. On utilisera la fonction `randint` de la bibliothèque `random`, `randint(a, b)` renvoyant un entier aléatoire entre les entiers $a$ et $b$ inclus.

3. Écrire une fonction `resoudre_defi` prenant en entrée un dictionnaire `perso` représentant un personnage et une chaîne `defi` représentant un défi, procédant à un tirage et renvoyant le nombre de succès décomptés. On rappelle qu'on peut parcourir une chaîne de caractères comme si c'était une liste de caractères.

4. Écrire une fonction `stats` prenant en entrée un personnage, un défi et un entier $n$, procédant à $n$ résolutions du défi et renvoyant la liste des fréquences (en pourcentages) d'obtention de chaque nombre de succès.

Par exemple, avec l'appel `stats(Imara, 'RRRV', 100000)`, vous devez obtenir un résultat proche (à $0,1$ près) de `[8.9, 31.0, 37.8, 18.9, 3.4]`.

# 2 Bonus et malus

Consciente que le défi s'annonce redoutable, et que l'option de se bagarrer avec le professeur d'informatique n'est pas raisonnable, Imara choisit de mettre toutes les chances de son côté en obtenant deux éléments favorables : avoir travaillé le DM, et avoir révisé sérieusement. Le système traduit ce type de bonus et de malus de la façon suivante :

- On compte le bonus net $x$ comme la différence entre le nombre d'éléments favorables et le nombre d'éléments défavorables (dans l'exemple, $x = 2$) ;
- si $x > 0$ , plutôt que de tirer 4 cartes, on en tire $4 + x$, et on garde les 4 cartes les plus faibles, sans changer leur ordre relatif. En cas d'égalité, on garde en priorité les cartes tirées en premier. Par exemple, avec $x = 2$, si on tire `[3, 8, 1, 8, 10, 2]`, on garde `[3, 8, 1, 2]` dans cet ordre.
- si $x < 0$, plutôt que de tirer 4 cartes, on en tire $4 - x$, et on garde les 4 cartes les plus fortes, sans changer leur ordre relatif. En cas d'égalité, on garde en priorité les cartes tirées en premier.

1. Écrire une fonction `k_plus_petits` prenant en entrée une liste `L` et un entier $k$, et qui renvoie les $k$ plus petites valeurs de `L` dans l'ordre dans lequel elles apparaissent initialement (en cas d'égalité, on conserve en priorité les plus à gauche). Écrire une fonction `k_plus_grands` sur le même principe.
   **Indication** : on pourra former la liste des couples (valeur, indice) et commencer par trier cette liste selon les valeurs avec un tri stable.
2. Déterminer la complexité des fonctions précédentes en fonction de la taille de la liste.
3. Écrire une fonction `resoudre_defi_bonus` adaptant la fonction `resoudre_defi` en prenant également le bonus en entrée, et en le prenant en compte dans le tirage.
4. Vérifier que les fréquences obtenues sur l'exemple du devoir d'ITC avec un bonus de 2 sont proches de `[2.4, 13.6, 29.7, 31.6, 22.6]`

# 3 Se bagarrer avec le destin

Dans la suite, on appelle **sélection** d'un tirage de $n$ valeurs une liste de 4 valeurs du tirage, dans le même ordre où elles apparaissent dans le tirage. On considère dans cette partie qu'Imara a acquis la capacité **Se bagarrer avec le destin** suivante :

Lorsque vous résolvez un défi en bénéficiant d'un bonus positif, vous choisissez la sélection plutôt que de forcément prendre les cartes les plus petites.

1. Donner un exemple de tirage illustrant l'intérêt de cette capacité.
2. Écrire une fonction `selection_optimale` prenant en entrée une liste `seuils` de 4 valeurs, et une liste `tirage` de longueur $n \geq 4$, et renvoyant la sélection de `tirage` réalisant le plus grand nombre de succès lorsqu'elle est comparée avec la liste `seuils`. On procédera en énumérant toutes les sélections.
3. Déterminer la complexité de la fonction précédente en fonction de $n$.
4. Pour améliorer la complexité de ce calcul, on définit, pour $i \in [\![0, n]\!], j \in [\![0, 4]\!]$, $m_{i,j}$ comme le nombre maximum de succès qu'on peut obtenir en sélectionnant jusqu'à $j$ cartes parmi les $i$ premières du tirage, et en les comparant aux premiers seuils.
   - Justifier $\forall i \in [\![1, n]\!], m_{i,0} = 0$ et $\forall j \in [\![0, 4]\!],\ m_{0,j} = 0$.
   - Justifier $\forall i \in [\![1, n]\!], j \in [\![1, 4]\!],\ m_{i,j} = \max(m_{i-1,j}, m_{i-1,j-1} + s_{i,j})$, où $s_{i,j}$ vaut 1 si la i-ème carte est inférieure ou égale au j-ième seuil, et 0 sinon.
   - Écrire une fonction `nombre_succes_optimal` prenant en entrée la liste de seuils et le tirage, calculant la matrice des $m_{i,j}$ case par case en exploitant la relation précédente, et renvoyant $m_{n,4}$.
   - Déterminer la complexité de la fonction précédente en fonction de $n$.
5. En déduire une fonction `resoudre_defi_destin` adaptant la fonction `resoudre_defi_bonus` pour y appliquer la capacité se bagarrer avec le destin en faisant le choix optimal des cartes à conserver. Vous devez obtenir des fréquences proches de `[2.2, 12.9, 29.0, 32.9, 23.0]`.
6. Démontrer que cette capacité permet au maximum de gagner trois succès.