INFORMATIQUE DS N° 1

02 DÉCEMBRE 2016

EXERCICE 1 - UN ORDINATEUR

1. Qui est considéré comme le "père" de l'informatique ?
2. Pourquoi un lecteur de DVD n'est pas un ordinateur ?
3. Pour mon TIPE je dois stocker les résultats d'une expérience. 1000 valeurs de pH d'une solution par heure sur une semaine. Chaque valeur de pH est codée sur 4 bits. Je ne dispose que d'une vielle clé de 1Mo. Pourrais-je stocker toutes mes valeurs ?

EXERCICE 2 - CODAGE DES NOMBRES

1. Écrire les entiers 14 et 55 en binaire.
2. Écrire les nombres binaires 1101 et 101011 en décimal.
3. Combien d'entiers puis je encoder sur 16 bits en base 2 ?
4. En complément à 2, quels sont les entiers que l'on code sur 16 bits ?
5. Décrire en quelques lignes la façon dont sont codés les nombres réels en machine, et les problèmes que cela pose.

EXERCICE 3 - ORGANISATION D'UN TOURNOI

Partie A : Recherches dans un tableau

Dans toute cette partie, l'utilisation de la commande `in` de Python sera rayée et moquée.
On cherche tout d'abord à définir des fonctions de recherche d'un élément dans un tableau, de plusieurs façons différentes.

1. Créer une fonction `recherche(t,a)` qui renvoie `True` ou `False` selon qu'un élément `a` soit ou non présent dans le tableau `t`.
2. Créer une fonction `position(t,a)` qui renvoie la position éventuelle de l'élément `a` dans `t`, et `None` sinon.
3. Créer une fonction `toutes_positions(t,a)` qui renvoie une liste (éventuellement vide) des positions de l'élément `a` dans `t`.

Partie B : Maximum dans un tableau

Dans toute cette partie, l'utilisation de la commande `max` de Python sera rayée et moquée.
On cherche à définir les positions du maximum d'un élément dans un tableau.

1. Créer une fonction `maximum(t)` qui renvoie la valeur du maximum des éléments de `t`.
2. Créer une fonction `tt_pos_max(t)` qui renvoie une liste des positions du maximum de `t`.

Partie C : Le Tournoi

Un tournoi d'un célèbre jeu de carte en ligne de n joueurs ($n > 100$ en pratique) va se dérouler. Les organisateurs réfléchissent sur la manière de faire jouer les matchs.

1. Si chaque joueur doit affronter chacun de ses adversaires, combien de matchs doivent être joués ?
2. Si on applique une règle d'élimination directe (finale, avant cela demi finales, avant cela quart de finales etc.) combien de matchs doivent être joués ? Combien maximum un joueur jouera-t-il de matchs ? Et au minimum ?

Peu satisfaits de ces deux façons de procéder les organisateurs choisissent un format appelé ≪ la ronde Suisse≫. Un nombre donné de ≪ rounds ≫ seront joués. A chaque round un joueur affronte un joueur qu'il n'a jamais affronté. A la fin du premier Round, un gagnant affrontera un gagnant, un perdant un perdant. A la fin du second Round, un joueur à 2 victoires affrontera un joueur à 2 victoires, un joueurs à 1 victoire affrontera un joueur à 1 victoire, et un joueur à 0 victoires affrontera un joueur à 0 victoires, et ainsi de suite.

NOTE AUX BENÊTS : SUITE AU VERSO !

Si un appariement exact n'est pas possible (par exemple 3 joueurs ont 4 victoires et 0 défaites), un joueur peut affronter un adversaire ayant une victoire de plus ou de moins de que lui.
Les n joueurs sont repérés par leur numéro, attribué de 0 à $n-1$. On construit deux listes :
- une liste `victoires` de longueur n, initialisée à `[0,0,...,0]` et actualisée à chaque round telle que `victoire[i]` donne le nombre de victoires du joueur i.
- une liste `historique` contenant des quadruplets de la forme `(round,a,b,res)` résumant chaque match. `round` est le numéro du round, `a,b` sont les numéros des joueurs avec `a<b` et `res` vaut 1 si le joueur `a` gagne, 2 si c'est le joueur `b`.
Ces deux variables sont définies dans le script, et sont donc globales pour la suite, inutile de les redéfinir ou de les passer en arguments.

3. On doit avant de se faire rencontrer deux adversaires vérifier qu'ils ne se sont pas déjà rencontrés et que leur nombre de victoires soientt compatibles. Proposer une fonction `possible(a,b)` renvoyant un booléen selon que la match de `a` contre `b` ait déjà été joué. On supposera que l'appel est fait avec `a<b` .
4. Écrire une fonction `actualise(round,a,b,res)` qui modifie la liste `victoire` selon le résultat du match, , et complète la liste `historique`.
5. Créer une fonction `en_tete()` sans argument qui renvoie la liste des joueurs ayant le plus de victoire.
6. A la fin des 10 rounds, on souhaite connaitre l'historique des matchs d'un joueur précis. Écrire une fonction `histo(i)` qui renvoie une liste des quadruplets `(round,a,b,res)` des matchs joués par `i`.

EXERCICE 4 - CALCULS ET COMPLEXITÉ

On cherche à effectuer le calcul de la somme $S = \sum_{k=0}^{n} \frac{x^k}{k!}$.
On considère que la fonction suivante est déjà saisie :

```
def puissance(x,n):
    P=1
    for k in range(n):
        P=P*x
    return(P)
```

1. Décrire l'appel `puissance(2,5)`. Que renvoie la fonction `puissance` ?
2. Écrire une fonction `facto(n)` qui prends en argument un entier `n` et calcule `n!`.
3. On considère la fonction suivante :

```
def calcul(x,n):
    S=0
    for k in range(n+1):
        S = S + puissance(x,k)/facto(k)
    return(S)
```

   Démontrer que la propriété H_i : S_k contient la valeur $\sum_{k=0}^{i} \frac{x^k}{k!}$ où S_i est le contenu de la variable S au bout de i passages en boucles est un invariant de boucle.
4. Déterminer la complexité en multiplications/divisions de cet algorithme.
5. Proposer un algorithme pour calculer S dont la complexité soit de l'ordre de n, en justifiant.