S3H - PROG – Examen de fin de module
vendredi 6 janvier 2016 – 10h30 (Examens A4)
durée :2h - calculatrices/documents non autorisés (sauf mémento Python 3)

Université de Lille 1 SCIENCES ET TECHNOLOGIES — UFR IEEA Formations en Informatique de Lille 1

[le barème donné est indicatif : sur 25 points donc 5 points de bonus]

Exercice 1 (4 pt) : TRI RAPIDE

On donne le code du tri rapide présenté en cours :

```
def my_quicksort(l,u,v):
    if u < v:
        # (a) partition :
        p = l[u]    # pivot
        i = u + 1   # first index
        j = v       # last index
        while i <= j:
            if l[i] < p :
                i += 1
            elif l[j] >= p:
                j -= 1
            else:
                l[i],l[j] = l[j],l[i]
                i += 1
                j -= 1
        # (b) swap «p» and the «last element known to be < p» :
        l[i-1],l[u] = l[u],l[i-1]
        # (c) recursive call x 2 :
        my_quicksort(l,u,i-2)
        my_quicksort(l,i,v)
```

Q1-1 (2.0 pt): Déroulez les différents appels de `my_quicksort(l,u,v)` sur `l=[3,5,2,1,4]` en choisissant les bonnes valeurs de `u` et `v` pour que `l` soit triée en totalité. Faites un schéma global des appels.

Q1-2 (2.0 pt):
Les deux lignes :

```
# (b) swap «p» and the «last element known to be < p» :
l[i-1],l[u] = l[u],l[i-1]
```

sont remplacées par ceci :

```
# (b) swap «p» and the «[FIXME : que fait i ? où va i ? code de Raymond]»
l[i],l[u] = l[u],l[i]
```

(a) Le tri est-il toujours réalisé correctement dans tous les cas ? Justifiez ou donnez un contre-exemple
(b) Le programme s'arrête-il dans certains cas sur une exception ? Justifiez ou donnez un exemple

---

Exercice 2 (9 pt) : *LISTES en définition récursive (comme en AP2)*

On donne cette définition d'un constructeur de « maillons » pour réaliser une liste chainée:

```
def create (v, n):
   return {'value' : v, 'next' : n}
```

Q2-1 (0.5 pt): Quel est le *type retour* de cette fonction ? Le décrire et *interpréter* également son contenu (*interpréter* en français : pas de code demandé).

Q2-2 (1.0 pt): Déroulez (à l'aide d'un schéma pour chaque ligne) les lignes successives de ce code

```
lc = create ("pif",None)          # (a)
lc = create ("paf",lc)            # (b)
lc = create ("toto",lc['next'])   # (c)
```

Q2-3 (1.5 pt): Donnez le code itératif (ou récursif) de la fonction `read(ll, p)` qui retourne la valeur du maillon situé à l'indice `p (p ≥ 0)` de la liste `ll`. Vous pouvez lever une exception du type `IndexError` si la liste est trop courte.

Q2-4 (1.5 pt): Expliquez, en déroulant (au moins) deux exemples (avec de plusieurs schémas pour chaque exemple), le principe de cette fonction :

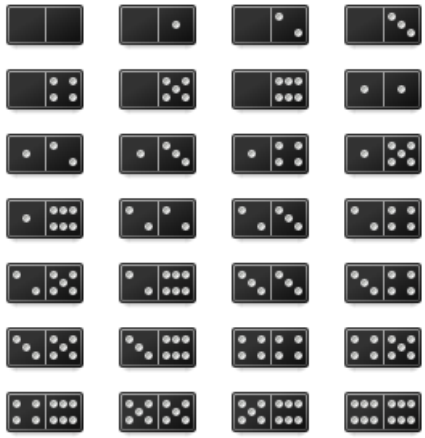
```
def insert(l, i, v):
    if i == 0:
        return create(v,l)          # (a)
    else:
        l['next'] = insert(l['next'], i-1, v) # (b.1)
        return l                              # (b.2)
```

Q2-5 (0.5 pt): Quel cas général peut déclencher une exception ici ? De quel type ?

Q2-6 (1.5 pt): Proposez une version *itérative* ayant les mêmes types de paramètres et de retour et le même comportement que la fonction des deux questions précédentes.

Q2-7 (2.5 pt): Donnez le code itératif ou récursif de la fonction **`delete(ll, p)`** qui enlève un maillon (mais ne retourne pas sa valeur, cette fonction retournera toujours la liste après effacement du maillon choisi)

Exercice 3 (12 pt) : module *DOMINO et jeu de dominos*



On propose d'implémenter un module « domino » (voir image) contenant un constructeur **create(l,r)** (avec **l** ≤ **r**), deux sélecteurs **get_l(d)** et **get_r(d)** pour obtenir les deux valeurs **l** et **r** (0 ≤ **l** ou **r** ≤ 6) du domino courant. ***On supposera que le « ZERO » est une valeur comme les autres (ce n'est donc pas un joker)***.

Q3-1 (2.5 pt): Proposez une implémentation (avec une documentation rapide) de ce module.

Q3-2 (1.0 pt): Proposez une *méthode* **rotate(d)** pour inverser les deux valeurs **l** et **r** du domino **d** courant. Cette *méthode* doit si possible être une *procédure* (sinon une *fonction*) : vous justifierez dans les deux cas la raison de votre choix.

Afin de réaliser une partie de dominos, on souhaite avoir, dans une liste python, la totalité des dominos du schéma. On souhaite dans un second temps mélanger cette liste de dominos pour créer une « pioche ». On ***ne demande pas ici*** *de documenter chacune des fonctions suivantes. Toutes les fonctions de cet exercice se basent sur des « listes python » (ne cherchez pas à utiliser des listes chainées : ça n'est pas l'objectif !!!)*

Q3-3 (1.0 pt): Proposez une fonction **dominoes()** qui retourne la *liste python* des 28 dominos dans l'*ordre de lecture* de la figure (*gauche → droit puis ligne → ligne*).

Q3-4 (2.0 pt):

(a) Quelle méthode vue en **cours et TP** peut mélanger « de façon homogène » cette liste ?

(b) Pouvez vous rappeler son principe de fonctionnement **vu en cours** et le justifier ?

Partie à deux joueurs au tour à tour *: on distribue initialement 7 dominos à chaque joueur, celui qui a le double (même valeur des deux cotés du domino) le plus fort le pose (si pas de double dans les 14 dominos, on pioche à tour de rôle : c'est un cas relativement rare). Le joueur suivant pose à l'une des extrémités un domino dont le coté adjacent a le même nombre que le 1er posé. Chaque joueur, s'il peut poser un domino, le pose à l'un des extrémités de la chaîne de jeu formée, sinon il pioche et garde le domino pioché pour les coups suivants. Il y a arrêt de la partie si l'un des deux joueurs a posé toute sa main de dominos : il est désigné gagnant. Il y a également arrêt si personne ne peut poser et que la pioche est vide : pour faciliter le programme, on supposera alors que le jeu est alors ex-aequo.*

Q3-5 (0.5 pt): Proposez l'implémentation d'une fonction **pioche(liste_pioche)** qui enlève et renvoie un domino de la pioche **liste_pioche**, sauf si cette pioche est vide (retourne dans ce cas *None*).

Q3-6 (0.5 pt): Proposez ensuite une implémentation d'une fonction **cree_main(liste_pioche)** qui renvoie une main (liste python de 7 dominos) à partie de la pioche initiale **liste_pioche**.

Q3-6 (1.5 pt): Proposez l'implémentation d'un fonction **designe (liste_main1, liste_main2, liste_pioche)** qui renvoie un couple : ce couple contient à la fois la liste **liste_jeu** (liste à un seul domino double qui a pu être posé), ainsi que le numéro du joueur qui a alors commencé cette pose. Il faut noter que cette fonction peut éventuellement modifier les mains des deux joueurs lorsque les deux listes représentant les mains ne contiennent initialement aucun domino double.

Q3-8 (2.0 pt): Proposez l'implémentation d'une fonction **poser(liste_jeu, liste_main)** *qui renvoie vrai si et seulement si la pose d'une des pièces de* **liste_main** *a été effectuée à une des extrémités de* **liste_jeu ;** cette fonction peut bien entendu modifier **liste_main** et **liste_jeu**.

Q3-9 (1.0 pt): Proposez enfin une simulation d'une partie qui retourne soit le numéro du gagnant, sinon une valeur nulle s'il n'y a pas de gagnant.