S3H - PROG - Contrôle de TD
lundi 2 novembre 2015
durée :1h30 - 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 (long mais sur 24 points, donc 4 points de bonus)]

Exercice 1 (5 pt) : *LISTES*

On définit à titre d'exemple la liste suivante `l = ['a','b','c','d','e']`

Q1-1 (0.5 pt) : Qu'affiche ce code python 3 ? Combien de lignes seront affichées ?

```
for e in l:
    print(e)
```

Q1-2 (1.0 pt) : Donnez le code python 3 de la procédure `miroir(l)` qui inverse les éléments de la liste `l` passée en paramètre : vous devez obtenir par exemple `l = ['e','d','c','b','a']` sur la liste proposée en exemple après l'avoir passée en paramètre . Notez que vous ne devez pas créer de nouvelle liste, ni ne retourner de valeur de type liste (puisqu'il s'agit d'une procédure).

Q1-3 (1.5 pt): Donnez le code python 3 de la fonction `combine(l1,l2)` qui retourne une nouvelle liste issue de la *combinaison* de deux listes `l1` et `l2` de même taille. La *combinaison* consiste à créer une liste, donc chacun des éléments est lui même une liste composé de deux éléments issues respectivement du même indice de `l1` et de `l2`.

exemple : si `l1=[1,2,3,4,5]` et `l2=['a','b','c','d','e'],combine(l1,l2)` retourne `[[1,'a'],[2,'b'],[3,'c'],[4,'d'],[5,'e']]`

Q1-4 (2.0 pt): Donnez le code python 3 de la fonction `transcombine(ll)` qui, pour une liste de listes de mêmes tailles, retourne la *transposition* (ou *combinaison multiple*) de ces listes

exemple : si `ll = [[1,2],['a','b'],['X','Y']], transcombine(ll)` retourne `[[1,'a','X'],[2,'b','Y']]`

Exercice 2 (5 pt) : *TRANCHES DE LISTES et RANGE*

On définit la liste suivante `l = ['a','b','c','d','e']`

Q2-1 (0.5 pt): Qu'affiche cette suite d'instructions dans l'interpréteur python 3 ?

```
>>>l[2:]
>>>l[:2]
>>>l[:2] + ['bc'] + l[2:]
```

Q2-2 (1.0 pt): Qu'affiche cette suite d'instructions dans l'interpréteur python 3 ?

```
>>>l[1:2] = ['bb'] ; l
>>>l[2:] = [] ; l
```

Q2-3 (1.0 pt): Qu'affiche cette suite de 4 blocs d'instructions dans l'interpréteur python ?

```
>>> for i in range(5):
>>>         print(i,end=' ')
>>> for i in range(1,5):
>>>         print(i,end=' ')
>>> for i in range(1,5,-1):
>>>         print(i,end=' ')
>>> for i in range(5,1,-1):
>>>         print(i,end=' ')
```

Q2-4 (2.5 pt): Donnez le code python 3 pour tester si un entier `e` est à la fois :

- compris entre 1 et 42,
- est pair, mais n'est pas multiple de 3 :

(a) avec un seule expression booléenne courte (composition au choix : les « % », « and », « not », « <= », « >= », « == » ou tests bit à bit « & » et « | » vu en codage sont autorisés) ?

(b) à l'aide de deux tests d'appartenance à deux *ranges* (que vous définirez auparavant), et une expression booléenne utilisant seulement un « and » et un « not »

Exercice 3 (5 pt) : *FIBONNACCI*

On donne cette définition de la suite de Fibonnacci suivante

f_0,f_1 = 0,1

f_n = f_(n-1) + f_(n-2) pour n > 1

On souhaite calculer la liste des nombres de Fibonnacci jusqu'au rang « n-1 », pour un « n » positif ou nul passé en paramètre.

exemple : si **`n = 8`** alors **`liste_fibo(n)`** retourne **`[0,1,1,2,3,5,8,13]`**

Q3-1 (3.5 pt): Proposez un implémentation efficace de cette fonction **`liste_fibo(n)`**

Q3-2 (1.5 pt): Justifiez son efficacité par rapport à une version récursive standard (non terminale)

Exercice 4 (9 pt) : *TUPLES, DICTIONNAIRES et ENSEMBLES*

On donne une longue liste d'entiers **`l = [12912, 82918176, 12219, 81827691, ...]`**

On souhaite détecter les ensembles d'entiers ayant la même *composition* en chiffres en base 10
exemple : "1206" est de même composition que "6021" mais pas de même composition que "621"

Q4-1 (0.5 pt) : Quel est le type et le résultat de l'instruction **`str(6021)`** en python 3 ?

Q4-2 (0.5 pt) : Quelle méthode implémentée dans le *TP anagrammes* vous a permis de passer de la chaine de caractères "6021" à la chaine de caractères triée "0126" ? L'avez vous programmé ou est-elle prédéfinie ?

Q4-3 (1.0 pt) : En utilisant cette précédente fonction (sans nécessairement la redéfinir), proposez un code minimal pour réaliser la fonction **`meme_composition(e1,e2)`** qui retourne **`True`** si (et seulement si) les entiers **`e1`** et **`e2`** sont de même composition en base 10.

Q4-4 (3.0 pt) :

(a) On ne souhaite pas effectuer de une comparaison en réalisant une double itération dans la liste **`l`** ? A votre avis pourquoi ?

> *Nous n'utiliserons donc pas* ***`meme_composition(e1,e2)`*** *directement, mais le principe utilisé en* ***Q4-2*** *peut être intéressant pour la suite de la question …*

(b) Quelle(s) structure(s) de données (et fonctions additionnelles) utiliseriez vous pour extraire rapidement les ensembles d'entiers de la liste **`l`** qui ont la même composition ? Proposez en un code permettant cette extraction.

Q4-5 (1.0 pt) : Comment éviter de « traiter » les éléments qui sont en doublons de la liste **`l`** ?
Vous proposerez une modification/ajout de code sur la question **Q4-4** en conséquence.

Q4-6 (3.0 pt) : Etant donné qu'il n'y a que 10 chiffres décimaux, et que les entiers pythons peuvent être grands, peut-on améliorer ce programme ? Proposez un deuxième code qui prend en compte ce constat.