Université de Lille 1 SCIENCES ET TECHNOLOGIES

ufr d'IEEA
Formations en
Informatique de
Lille 1

L1 MASS/SESI/PEIP - 2015–2016

Algorithmes et Programmation 1

DS1 - 05/03/2016

durée 2h - documents non autorisés (sauf memento Python3)

Ce devoir contient trois exercices indépendants.

Sauf mention explicite du contraire, toutes les fonctions que vous réaliserez devront comporter une documentation avec contraintes et exemples d'utilisation.

Le barême est donné à titre indicatif.

Exercice 1 *Kesaco* (5 pts)

Le but de cet exercice est de comprendre ce que fait la fonction `kesaco` écrite ci-dessous et de la compléter avec de nouvelles instructions.

```
def kesaco(e, l):
    """
    CU : l est une liste dont les éléments sont triés par ordre croissant.
    """
    a = 0
    b = len(l)-1

    while a <= b :
        m = (a+b)//2
        if l[m] == e :
            return m
        elif l[m] > e :
            b = m-1
        else :
            a = m+1
    return -1
```

Question 1

Quelles assertions ajouter au début de la fonction pour vérifier que `l` est bien une liste triée ? Pour cela, vous pourrez utiliser le prédicat `est_trie` qui prend en paramètre une liste `l` et qui renvoie le booléen `True` si cette liste est triée et `False` dans le cas contraire.

Question 2

Décrire les valeurs prises par les différentes variables de la fonction `kesaco` à la fin de chaque tour de boucle, pour les 3 appels à cette fonction indiqués ci-dessous.

```
kesaco(5, [10, 12, 14, 15])
kesaco(12, [10, 12, 14, 15])
kesaco(13, [10, 12, 14, 15])
```

Pour une meilleure lisibilité, les valeurs seront indiquées dans un tableau avec les colonnes suivantes :

`e | a | b | m | l[m]`

Enfin, vous préciserez la valeur renvoyée par `kesaco`.

Question 3 À l'aide de ces résultats, déduire ce que fait la fonction `kesaco`. Compléter la documentation de cette fonction en décrivant en français ce qu'elle renvoie et en ajoutant des doctests.

Question 4 Indiquer les conditions d'utilisation supplémentaires qui doivent être respectées pour que la fonction `kesaco` fonctionne correctement.

Exercice 2 *Cinéma* (10 pts)

On veut écrire un programme en PYTHON permettant de stocker et de manipuler des titres de films et les prénoms et noms de leurs acteurs principaux. On souhaite ainsi pouvoir stocker par exemple les informations suivantes :

Titre	Acteurs
Indiana Jones et la dernière croisade	Harrison Ford, Sean Connery
Star Wars VII	Harrison Ford, Adam Driver
Frances Ha	Greta Gerwing, Adam Driver
Star Wars III	Hayden Christensen, Natalie Portman, Ewan McGregor
Goldfinger	Sean Connery
Léon	Jean Réno, Natalie Portman

Ces informations seront stockées en PYTHON dans un dictionnaire composé :
— des titres de films, (chaînes de caractères), comme clés ;
— de l'ensemble des prénoms et noms des acteurs principaux, comme valeurs associées.
Ainsi, l'exemple précédent serait stocké dans le dictionnaire nommé `FILMS` dans la suite :

```
FILMS = {
      'Indiana Jones et la dernière croisade' : {'Harrison Ford', 'Sean Connery'},
      'Star Wars VII' : {'Harrison Ford', 'Adam Driver'},
      'Frances Ha' : {'Greta Gerwing', 'Adam Driver'},
      'Star Wars III' : {'Hayden Christensen', 'Natalie Portman', 'Ewan McGregor'},
      'Goldfinger' : {'Sean Connery'},
      'Léon' : {'Jean Réno', 'Natalie Portman'}}
```

Dans tout l'exercice, on considérera que le prénom et le nom de chaque acteur sont toujours stockés ainsi dans une seule chaîne de caractères.

Question 1

Écrire une fonction `nb_acteurs` qui prend en argument un dictionnaire `films` (comme l'exemple précédent) et le titre d'un film `titre`, et qui renvoie le nombre d'acteurs (principaux) de ce film. Par exemple :

```
>>> nb_acteurs(FILMS, 'Star Wars III')
3
>>> nb_acteurs(FILMS, 'Goldfinger')
1
```

Question 2 Écrire une fonction `filmographie` qui prend en argument un dictionnaire `films` et le nom d'un acteur `acteur`, et qui renvoie l'ensemble des films dans lequel cet acteur a joué. Par exemple :

```
>>> filmographie(FILMS, 'Harrison Ford')
{'Indiana Jones et la dernière croisade', 'Star Wars VII'}
```

```
>>> filmographie(FILMS, 'Hayden Christensen')
{'Star Wars III'}
>>> filmographie(FILMS, 'Eric Wegrzynowski')
set()
```

Question 3 Écrire une fonction `ensemble_acteurs` qui prend en argument un dictionnaire `films`, et qui renvoie l'ensemble de tous les acteurs ayant joué dans au moins un des films contenus dans `films`. Par exemple :

```
>>> acteurs = ensemble_acteurs(FILMS)
>>> for acteur in acteurs:
...     print (acteur)
...
Hayden Christensen
Harrison Ford
Sean Connery
Adam Driver
Ewan McGregor
Greta Gerwing
Jean Réno
Natalie Portman
```

Question 4 Écrire une fonction `co_acteurs` qui prend en argument un dictionnaire `films` et le prénom et nom d'un acteur `acteur`, et qui renvoie l'ensemble de tous les acteurs ayant joué avec l'acteur `acteur`. Par exemple :

```
>>> co_acteurs(FILMS, 'Natalie Portman')
{'Ewan McGregor', 'Hayden Christensen', 'Jean Réno'}
```

Question 5 On souhaite désormais construire la «table inversée» qui permet de lister pour chaque acteur les films dans lesquels il a tenu un rôle principal. Cette table sera aussi implémentée en PYTHON par un dictionnaire dont les clés sont les prénom et nom des acteurs sous forme d'une chaîne, et les valeurs associées sont l'ensemble des titres de films dans lesquels cet acteur a joué.

Écrire une fonction `table_acteurs` qui prend en argument un dictionnaire `films`, et qui renvoie la table inversée de `films`. Par exemple :

```
>>> tab_act = table_acteurs(FILMS)
>>> tab_act['Harrison Ford']
{'Indiana Jones et la dernière croisade', 'Star Wars VII'}
>>> tab_act['Natalie Portman']
{'Star Wars III', 'Léon'}
```

Question 6 Écrire une fonction `acteurs_populaires` qui prend en argument un dictionnaire `films`, et qui renvoie la liste des prénoms et noms du ou des acteurs les plus populaires, à savoir celui ou ceux ayant joué dans le plus grand nombre de films. On prendra comme hypothèse que `films` contient au moins un film avec au moins un acteur, et on n'utilisera pas la fonction `max()` de PYTHON. Par exemple :

```
>>> acteurs_populaires(FILMS)
['Harrison Ford', 'Sean Connery', 'Adam Driver', 'Natalie Portman']
```

Exercice 3 *Tri pair-impair* (5 pts)

On donne le code de la fonction suivante :

```
def pair_impair(l,comp=compare):
    """
    effectue une étape du tri pair et impair
    CU : l liste homogène d'éléments comparables selon comp
    """
    res=True
    for k in range(0,len(l)-1,2):
        if comp(l[k],l[k+1])>0:
            echanger(l,k,k+1)
            res=False
    for k in range(1,len(l)-1,2):
        if comp(l[k],l[k+1])>0:
            echanger(l,k,k+1)
            res=False
    return res
```

où `compare` est une fonction qui renvoie respectivement -1 ou bien 0 ou bien 1 selon que son premier paramètre est respectivement strictement inférieur ou bien égal ou bien strictement supérieur à son second paramètre.

La fonction `echanger` possède trois paramètres qui sont une liste `l` et deux entiers qui doivent être des indices valides `i` et `j` de la liste. La fonction `echanger` a pour effet d'échanger `l[i]` avec `l[j]`. Par exemple si `l=[1,3,7,9]` alors après l'exécution de `echanger(l,1,2)` la liste `l` est égal à `[1,7,3,9]`.

Question 1 Écrire le code de `echanger`. (La documentation n'est pas demandée)

Question 2 On donne `l=[1, 2, 6, 7, 3, 4, 5, 0]`. Quelles sont les valeurs successives de la liste `l` **après** chaque étape des boucles `pour` lorsqu'on exécute `pair_impair(l)` ? Présentez votre réponse sous forme d'un tableau, à trois colonnes, la première donnant l'indice `k` , la seconde la valeur de `l` et la troisième celle de `res`.

Question 3 Le code de la fonction `pair_impair` est un peu maladroit. Il y a deux groupes de quatre lignes qui se ressemblent énormément. Proposez une nouvelle version de `pair_impair`.

Question 4 Que dire de la liste `l` lors de l'appel `pair_impair(l)` lorsque le résultat est `True` ?

Dans ce qui suit, n désigne la longueur de la liste `l`.

Question 5 En fonction de n, combien de comparaisons d'éléments sont effectuées lors de l'appel `pair_impair(l)` ?

Il est prouvé qu'en répétant au plus $\frac{n}{2}$ appels consécutifs `pair_impair(l)` la liste `l` est triée.

Question 6 Écrire le code d'une fonction de `tri_pair_impair` à deux paramètres, une liste `l` et une fonction de comparaison `comp` (valant par défaut `compare`), qui trie la liste `l`.

Question 7 Quelle est la complexité de `tri_pair_impair` en nombre de comparaisons d'éléments en fonction de la longueur n de la liste ? S'il y a lieu, distinguez un meilleur et un pire des cas.
