UFR de mathématique et d'informatique
Université de Strasbourg

Licence 1 Semestre 1 Automne
Algorithmique et Programmation 1
Contrôle continu
Vendredi 14 décembre 2018
Durée : 2 heures
Cours et documents interdits
(sauf deux feuilles A4 manuscrites recto-verso)

L'objectif de cette épreuve est d'évaluer vos compétences en algorithmique et en programmation *python*. Donc dans chaque question une fraction des points qualifie l'algorithme et une autre fraction qualifie le respect de la syntaxe *python* et la lisibilité de vos réponses. Si l'algorithme que vous décrivez (en *python* ou en français) est correct, alors il aura les points qualifiant l'algorithme, sauf si votre description ou votre syntaxe est tellement approximative qu'elle laisse un doute sur cet algorithme. En particulier, nous vous invitons à faire extrêmement attention à votre indentation (les caractères TAB) puisqu'elle influe grandement sur l'algorithme.

Toutes les questions ne sont pas indépendantes. Cependant, si la réponse à une question B nécessite l'utilisation d'une fonction demandée à la question A, sentez-vous libre d'utiliser cette fonction, même si vous n'avez pas pu répondre à la question A. C'est pourquoi nous vous invitons à lire toutes les questions, même si vous pensez qu'elles sont trop difficiles pour vous et même si vous n'avez pas pu faire les questions précédentes.

Le barème de chaque question est donné à titre indicatif.

1 L'écharpe blanche (4 points : 1+1+2)

Dans cette section, pour créer des chaînes de caractères vous pouvez :

— soit dupliquer des chaînes de caractères existants ;

— soit partir d'une chaîne vide et, en utilisant une boucle, y ajouter d'autres caractères.

1. Écrire une fonction `etoiles` qui, à partir d'un entier `longueur` retourne une chaîne de caractères constituée de `longueur` caractères `"*"` (étoiles). Par exemple l'appel `etoiles(5)` retourne la chaîne `"*****"`.

2. Écrire une fonction `etoilesBandeBlanche` qui, à partir de trois entiers `longueur`, `pos` et `larg_bande`, retourne une chaîne de caractères constituée de `longueur` caractères. Il s'agit de caractères étoiles `"*"` sauf une bande de largeur `larg_bande` et commençant à l'indice `pos`. Cette bande sera constituée de caractères espace `" "`. Par exemple l'appel `etoileBandeBlanche(10,3,2)` retourne la chaîne `"***  *****"`, c'est à dire une chaîne constituée de 10 caractères. Des étoiles, sauf une bande de longueur 2 et commençant à l'indice 3 constituée d'espaces. On suppose que la bande blanche doit toujours être entière. Autrement dit, si `pos + larg_bande > longueur`, la fonction devra émettre un message d'erreur et retourner la chaîne vide.

3. Écrire une fonction `afficheEcharpeBlanche` qui, à partir de deux entiers `longueur` et `larg_bande` ne retourne aucune valeur mais affiche, sur le terminal, `longueur-larg_bande+1` lignes, chacune constituée de `longueur` caractères. Le résultat doit représenter un rectangle constituée d'étoiles sauf une bande blanche allant du coin supérieur gauche au coin inférieur droit (cf figure ci-contre). Pour ce faire, nous vous demandons d'utiliser la fonction `etoilesBandeBlanche`.

```
   *******
*    ******
**    *****
***    ****
****    ***
*****    **
******    *
*******
```

Le résultat de l'appel `afficheEcharpeBlanche(10,3)` : un rectangle avec 8 lignes de 10 caractères.

Dans les sections 2 et 3, nous vous demandons d'utiliser des algorithmes itératifs. Nous vous demandons également de n'utiliser aucune fonction ou méthode *python* prédéfinie sauf `range` et `len`.

2 Fonctions opérant sur des listes (4 points : 1+1+1+1)

1. Écrire une fonction `sommeLst` qui, à partir d'une liste de nombres `lst` retourne un nombre représentant la somme des éléments de `lst`. Pour une liste vide, `sommeLst` devra retourner 0. Par exemple `sommeLst([1,2,3])` devra retourner 6.
2. Écrire une fonction `normaliser` qui, à partir d'une liste `lst` de nombres strictement positifs, calcule la somme des éléments de `lst` et divise tous les éléments de `lst` par la valeur de cette somme. À la fin de cette opération, la somme des éléments de `lst` doit être égale à 1. Cette fonction ne retourne rien (retourne `None`).
3. On souhaite effectuer la même tâche, mais sans modifier la liste `lst`. Écrire une fonction `creeNormal` qui, à partir d'une liste `lst` de nombres strictement positifs, calcule la somme `S` des éléments de `lst`. Ensuite cette fonction devra créer et retourner une autre liste dont les éléments sont égaux à ceux de `lst` divisés par `S`. Par exemple le programme :

```
lst=[2,1,1]
nrm=creerNormal(lst)
print("lst : ",lst)
print("nrm : ",nrm)
```

devra afficher :

```
lst : [2,1,1]
nrm : [0.5,0.25,0.25]
```

4. Écrire une fonction `cumul` qui, à partir d'une liste de nombres `lst`, crée et retourne une autre liste de nombres de même longueur. Dans cette nouvelle liste, la valeur de l'élément d'indice `n` devra être la somme des éléments d'indice 0 à `n` de `lst`. Par exemple, `cumul([2,3,0,6,2])` devra être la liste `[2,5,5,11,13]`.

3 Tri de listes (3 points : 1+1+1)

1. Écrire une fonction `inserer` dont les paramètres sont : une liste quelconque `lst`, un nombre `nb` et un entier `ind` compris entre 0 et la longueur de `lst`. La valeur de retour de cette fonction est une autre liste contenant les mêmes éléments que `lst` mais avec un élément supplémentaire égal à `nb` inséré avant l'élément d'indice `ind`. Si `ind` est égal à la longueur de `lst`, la fonction insère le nombre à la fin de la liste.
   - `inserer([7,5,9,11,1],4,3)` insère 4 avant 11 (l'élément d'indice 3) et retourne `[7,5,9,4,11,1]` ;
   - `inserer([7,5,9,11,1],4,0)` insère 4 avant 7 (l'élément d'indice 0) et retourne `[4,7,5,9,11,1]` ;
   - `inserer([7,5,9,11,1],4,5)` insère 4 à la fin de la liste et retourne `[7,5,9,11,1,4]` ;

   Vous n'avez pas le droit d'utiliser l'opération `insert` de la bibliothèque standard, vous devez écrire votre propre fonction.
2. Écrire une fonction `indiceIns` qui, à partir d'une liste `lst` croissante et d'un nombre `n` retourne un entier qui situe `n` dans `lst`. Plus précisément :
   - si `lst` est vide, alors la fonction retourne 0 ;
   - sinon, la fonction retourne l'indice du premier élément de `lst` qui soit supérieur ou égal à `n`.
   - Si `n` est supérieur à tous les éléments de `lst`, alors la fonction retourne la longueur de la liste.

   Par exemple pour `lst=[1,3,7,9]`,
   - `indiceIns(lst,4)` retourne `2` ;
   - `indiceIns(lst,7)` retourne `2` ;
   - `indiceIns(lst,-5)` retourne `0` ;
   - `indiceIns(lst,20)` retourne `4`.
3. Écrire une fonction `triInsertion` qui, à partir d'une liste de nombres `lst` trie les éléments de `lst` dans le sens croissant en mettant en œuvre l'algorithme du *tri par insertion*.

4 Récursivité (3 points : 1+1+1)

1. Écrire une fonction récursive `sommeDiv23` qui, à partir d'un entier `n` retourne la somme de tous les nombres inférieurs ou égal à `n` et divisibles par 2 ou par 3. Par exemple `sommeDiv23(4)` vaut `4+3+2=9`.
2. Écrire une version récursive de la fonction `sommeLst` ;
3. Écrire une fonction récursive `minimum` qui, à partir d'une liste non-vide de nombres retourne l'élément le plus petit de la liste.

5 Complexité (4 points : 1+1+1+1)

Les deux algorithmes suivants opèrent sur une liste `lst` composée de `n` nombres.

1. L'algorithme suivant permet de calculer l'indice de l'élément le plus grand dans une liste. Combien est-ce que cet algorithme effectue de comparaisons ?

```
soit max initialise a lst[0]
soit n le nombre d'elements de lst
pour i allant de 0 a n-1
   si lst[i] > min
      max <- lst[i]
      indmax <- i
   finsi
fin pour
```

2. Pour de très grandes valeurs de `n` comment croît la complexité de cet algorithme ?
3. L'algorithme suivant effectue un tri, dans lequel il a besoin de calculer l'élément le plus grand de la liste. Pour cela, il utilise l'algorithme de la question 1. Combien de comparaisons est-ce que cet algorithme effectue (en comptant celles nécessaires au calcul du maximum) ?

```
soit n le nombre d'elements de lst
pour i allant de 0 a n-1
    soit l la sous-liste de lst constituee des elements d'indice i a n-1
    soit ind l'indice de l'element le plus grand de l
    permuter les elements d'indice i et ind
fin pour
```

4. Pour de très grandes valeurs de `n` comment croît la complexité de cet algorithme ?

6 La monnaie (2 points)

Dans un distributeur automatique (de billets, de boissons ou autres), pour donner une somme d'argent N à l'utilisateur, il faut pouvoir exprimer cette somme sous la forme d'un ensemble de pièces ou de billets. Par exemple la liste suivante représente les pièces et billets disponibles dans la zone euro. Cette liste est triée dans le sens décroissant.

```
monnaie=[500, 200, 100, 50, 20, 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01]
```

Cette liste représente respectivement : les billets de 500€ , 200€ , etc. Écrire une fonction `decomposer` (itérative ou récursive) qui, à partir de la liste ci-dessus et d'une somme d'argent `montant` retourne une liste de longueur minimale composée des nombres ci-dessus et dont la somme est égale à `montant`. Par exemple, `decomposer(monnaie,472.32)` retourne la liste :

```
[200, 200, 50, 20, 2, 0.2, 0.1, 0.02]
```

Autrement dit, pour avoir 472.32 €, on doit avoir :

- 2 billets de 200€ ;
- 1 billet de 50€ ;
- 1 billet de 20€ ;
- 1 pièce de 2€ ;
- 1 pièce de 0.2€ (20 centimes) ;
- 1 pièce de 0.1€ (10 centimes) ;
- 1 pièce de 0.02€ (2 centimes).