Contrôle continu - Programmation (MASCO 1)
Durée : 2h
Documents autorisés : le polycopié du cours

A lire avant de commencer

- Les exercices sont de difficultés différentes, le nombre d'étoiles indique la difficulté.
- Dans chacune des questions de cet examen, on vous demande d'écrire une fonction. Avant d'écrire cette dernière, décrivez en une phrase ou deux le principe de fonctionnement de votre fonction, sauf pour les fonctions à une étoile.
- Vous pouvez utiliser votre ordinateur pour tester vos fonctions. Si vous souhaitez le faire, ne commencez pas à programmer avant d'avoir une idée très claire de ce que vous voulez faire !
- Certaines fonctions que l'on vous demande d'écrire existent déjà en Python. Si c'est le cas, vous n'avez **pas le droit** de les utiliser dans votre solution. L'idée est de comprendre comment ces fonctions ont été programmées !
- Ecrivez de manière la plus lisible possible, les copies illisibles ne seront pas lues.

1 Tri et recherche

Q1.1 * Ecrire une fonction `argmin(L)` qui prend en argument une liste `L` de nombres décimaux et renvoie l'**indice** du plus petit élément de `L`.

```
def argmin(L):
    i =0;
    arg = 0;
    while i < len(L) :
        if L[arg] > L[i] :
            arg = i
        i = i + 1
    return arg
```

Q1.2 ** Ecrire une fonction `tri(L)` qui prend en argument une liste `L` de nombres décimaux et renvoie une nouvelle liste comportant les éléments de `L` triés dans l'ordre croissant. La fonction `tri` fera appel à la fonction `argmin` de l'exercice précédent et ne fera pas appel à une fonction de tri de Python.

```
def tri(L):
    res = []
    liste = L.copy()
    while len(liste) > 0 :
        arg = argmin(liste)
        res.append(liste[arg])
        liste.pop(arg)
    return res
```

Q1.3 *** Etant donné un liste triée `L`, écrire une fonction récursive `recherche(L, debut, fin, x)` qui recherche l'élément `x` dans la partie de la liste `L` commençant à l'indice `debut` et terminant à l'indice `fin`. Le principe de la recherche est le suivant :

(a) On détermine `milieu`, l'indice du milieu de `L` en utilisant les indices `debut` et `fin`.
On peut alors "découper" `L` en deux sous listes, une sous liste gauche (`G`) qui commence à l'indice `debut` et se termine à l'indice `milieu - 1`.
Et une sous liste droite (`D`), qui commence à l'indice `milieu + 1` et se termine à l'indice `fin`.

(b) Si `L[milieu]` est égal à `x` alors on renvoie `True`.

(c) Si `L[milieu]` est supérieur à `x`, alors on recherche `x` dans `G`

(d) Si `L[milieu]` est inférieur à `x`, alors on recherche `x` dans `D`

(e) Si à un moment donné `debut` devient supérieur à `fin`, alors on renvoie `False`.

```
def recherche_dicho(arr, l, r, x):
    if r >= l:
        mid = l + (r - l)/2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return recherche_dicho(arr, l, mid-1, x)
        else:
            return recherche_dicho(arr, mid + 1, r, x)
    else:
        return -1
```

2 Matrices

Les éléments d'une liste peuvent eux même être des listes. Cela permet en particulier de représenter des matrices, que l'on peut voir comme des listes de lignes, les lignes étant elles mêmes des liste de nombres décimaux.

On supposera que lorsqu'une ligne se termine par des zéros, ceux-ci peuvent être omis, comme dans l'exemple suivant, où la liste `[[1,2],[3,4,5],[6]]` correspond à la matrice

$$\begin{matrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 0 & 0 \end{matrix}$$

Q2.1 * Ecrire la fonction `NbLignes(M)` qui prend en argument une matrice `M` et qui renvoie le nombre de lignes de la matrice `M`

```
def nbLignes(M):
    return len(M)
```

Q2.2 ** Ecrire la fonction `NbCol(M)` qui prend en argument une matrice `M` et qui renvoie le nombre de colonnes de la matrice `M`

```
def nbCol(M):
    col = 0
    for l in M:
        if len(l) > col :
            col = len(l)
    return col
```

Q2.3 ** Ecrire la fonction `element(M, l, c)` qui renvoie l'élément de la matrice se trouvant à la ligne `l` et à la colonne `c`. Si cet élément n'existe pas, la fonction renvoie `None` si cet élément est un zéro implicite, la fonction renvoie `0`

```
def element(M, l, c):
    if l > nbLignes(L): return None
    if c > nbCol(M): return None
    ligne = M[l]
    if c < len(ligne): return ligne[c]
    return 0
```

Q2.4 ** Ecrire la fonction `affiche(M)` qui affiche la matrice `M`, y compris les zéros implicites. Pensez à utiliser les fonctions définies dans les trois exercices précédents.

```
def affiche(M):
    for l in range(nbLignes(M)):
        for c in range(nbCol(M)):
            print(element(M,l,c), ' ', end='')
        print()
```

3 Recherche de Motifs

Soit `s` une chaîne de caractères et `m` une autre chaîne de caractère, appelée motif. On s'intéresse, dans cet exercice à retrouver des occurrences du motif `m` dans la chaîne `s`.

Q3.1 ** Ecrire une fonction `recherche(s,m)` qui renvoie `False` si `m` n'apparaît pas dans `s` et `True` sinon.

```
def recherche(s,m):
    i = 0
    while i < len(s):
        if s[i:i+len(m)] == m :
            return True
        i = i + 1
    return False
```

Q3.2 ** Ecrire une fonction `recherche2(s,m)` qui renvoie une liste contenant les indices de début de chaque occurrence de `m` dans `s`. Exemple :

```
>>> print(recherche2('abbabaab', 'ab'))
>>> [0, 3, 6]
```

```
def recherche2(s,m):
    i = 0
    l = []
    while i < len(s):
        if s[i:i+len(m)] == m :
            l.append(i)
        i = i + 1
    return l
```

4 Palindromes

Un palindrome est une chaîne de caractères qui peut se lire indiféremment de gauche à droite ou de droite à gauche, comme `kayak` ou `serres`.

Q4.1 ** Ecrire une fonction non récursive `palindrome(s)` qui renvoie `True` si `s` est un palindrome et `False` sinon.

```
def palindrome(s):
    debut = 0
    fin = len(s) -1
    while(debut < fin):
        if s[debut] != s[fin]:
            return False
        else:
            debut += 1
            fin -= 1
    return True
```

Q4.2 ** On peut remarquer qu'en otant la première et la dernière lettre d'un palindrome, on obtient toujours un palindrome (ou une lettre unique ou rien du tout). Utilisez cette remarque pour écrire une fonction récursive `palindromeRec(s)` qui qui renvoie `True` si `s` est un palindrome et `False` sinon.

```
def palindromeRec(s):
    if len(s) == 0 or len(s) == 1:
        return True
    if s[0] == s[-1]:
        return palindromeRec(s[1:-1])
    return False
```