Informatique de Tronc Commun

Devoir surveillé n°2 (durée : 2 h)

Les réponses sont à donner sur la copie.
L'usage de la calculatrice n'est pas autorisé.

Exercice 1.

On considère les deux fonctions suivantes :

```
def f1(L):
    """ L est supposée non vide"""
    x = 0
    for k in range(len(L)):
        if L[k] >= x:
            x = L[k]
            y = k
    return y, x
```

```
def f2(L):
    """ L est supposée non vide"""
    x, y = L[0], 0
    for k in range(1, len(L)):
        if L[k] < x:
            x = L[k]
            y = k
    return y, x
```

Q1. Que renvoient les appels suivants :

Q1.a. `f1([11, 12, 13, 14, 15])`

Q1.b. `f1([10, 12, 13, 14, 12])`

Q1.c. `f1([-11, 12, -13, 14, 15])`

Q1.d. `f1([-15, -12, -13, -11, -15])`

Note : les exemples suivants étaient mieux choisis : `[11, 12, 13, 14, 15]`, `[11, 15, 13, 11, 15]` `[-11, 12, -13, 14, 15]`, `[-11, -13, -11, -15, -15]`

Q2. Que renvoient les mêmes appels, mais pour la fonction `f2`.

Q3. Que fait la fonction `f2` ?

Q4.

Q4.a. Définir une fonction `g(L)`, renvoyant la position de la première occurrence du maximum d'une liste non vide d'entiers et la valeur de ce maximum. (que changer pour renvoyer la position de la dernière occurrence ?)

Q4.b. L'appel `g(['toto', 'titi', 'tata'])` déclenche-t-il une erreur ? si oui, pourquoi, si non, que renvoie-t-il ?

Q4.c. L'appel `g([2, '3', '2'])` déclenche-t-il une erreur ? si oui, pourquoi, si non, que renvoie-t-il ?

Q4.d. L'appel `g(['12', '1', '2'])` déclenche-t-il une erreur ? Si oui, pourquoi ? Si non, que renvoie-t-il ?

Exercice 2.

Q1. Définir une fonction `somme`, prenant en argument une liste d'entiers ou de flottants, L, et renvoyant la somme des éléments de L.

Q2. Définir une fonction `moyenne(L)`, faisant appel à la fonction `somme`, et renvoyant la moyenne d'une liste d'entiers ou de flottants, L.

Q3. On rappelle la formule de calcul de la variance d'une liste de n réels $[v_0, v_1, \dots, v_{n-1}]$:

$$V = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2$$

où $\bar{x}$ désigne la moyenne des valeurs $[v_0, v_1, \dots, v_{n-1}]$

Q3.a. Définir une fonction `variance(L)`, prenant en argument une liste d'entiers ou de flottants, L, et renvoyant la variance des éléments de L.

On peut montrer que la variance est aussi donnée par la formule équivalente, appelée formule de Huyguens :

$$V = \left(\frac{1}{n}\sum_{i=0}^{n-1} x_i^2\right) - \bar{x}^2.$$

Q3.b. Rappeler comment on peut, à l'aide d'une compréhension de listes, définir la liste, `L2`, des carrés des valeurs d'une liste L.

Q3.c. Utiliser la réponse à la question précédente pour définir une fonction `variance2(L)`, équivalente à la fonction `variance1(L)`, et faisant appel autant que possible à la fonction `moyenne`.

Exercice 3.

Q1. Donner les valeurs de `T1` et `T2`, définis respectivement par

```
>>> T1 = [0] * 10
>>> T2 = [(-1) ** i for i in range(5)]
```

On considère la fonction suivante :

```
def f(L):
    T = [0, 0, 0, 0, 0, 0]
    for val in L:
        T[val] += 1
    return T
```

Q2. Que renvoie l'appel `f([4, 1, 2, 3, 5, 2, 1, 2, 2, 5, 4, 3, 1])` ?

Q3.

Q3.a. Quelles sont les entrées possibles de la fonction f ?

Q3.b. Pour des entrées correctes, que fait la fonction f ?

Q4. Quelle modification minimale apporter à l'instruction en première ligne du corps de la fonction (en gras ci-dessus) pour qu'elle puisse prendre en entrée la liste suivante : `L0 = [i ** 2 for i in range(-5, 6)]`. Décrire le tableau que renverra la fonction `f` si on l'appelle avec en entrée la liste `L0`.

Exercice 4.

On rappelle la documentation Python sur la fonction `randint` du module `random` :

```
>>> help(randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

Q1. Définir une fonction `f`, sans argument, renvoyant un tuple de trois entiers tirés aléatoirement entre 1 et 6, à l'aide de la fonction `randint`. On effectuera les importations nécessaires.

Q2. Compléter le script suivant, dans lequel on simule le lancer de trois dés à six faces, équilibrés, simultanément, jusqu'à obtenir un triplet dont les éléments sont tous égaux à 6.

Le résultat de chaque tirage sera affiché.

Lorsque la boucle `while` termine, on affiche le nombre de tirages qui aura été nécessaire pour obtenir le triplet (6, 6, 6).

Il est demandé de privilégier de faire appel à la fonction f définie en **Q1**.

```
d1, d2, d3 = ...
...
while ... :
    print(...)
    ...
    ...
...
print("nombre de tirages nécessaires :", ...)
```

L'exécution du script pourra, par exemple, provoquer les affichages suivants

```
4, 7, 4
5, 6, 1
6, 6, 6
nombre de tirages nécessaires : 3
```

Q3. Quelle modification doit-on apporter au script si on veut simuler le lancer de trois dés à six faces, équilibrés, simultanément, jusqu'à obtenir un triplet dont les éléments sont égaux à 4, 2, 1 dans un ordre quelconque ?

Exercice 5.

On considère ici des listes non vides d'entiers.

Préliminaires

Q1. Quels affichages produit l'exécution du script suivant :

```
L = [8, 9, 4, 5, 7, 3, 1, 2]
a, b = 2, 4
for i in range(a, b):
    print(L[i - 1], L[i])
```

Q2. Quelles conditions doivent respecter les valeurs données aux variables `a` et `b` (supposées entières et positives) pour que l'exécution du script ci-avant ne provoque pas d'erreur et affiche bien deux valeurs consécutives de la liste L ?

Q3. Reprendre la question précédente si on remplace l'instruction `print(L[i - 1], L[i])` par `print(L[i], L[i + 1])` ?

Q4. Écrire une fonction `estcroissante`, de paramètre `L`, renvoyant un booléen, `True` ou `False`, selon que la liste `L` en entrée est croissante ou non.

Problème

On rappelle les opérations et conventions du *slicing* sur les listes Python.

Pour une liste L, si n a pour valeur `len(L)`, a et b désignant des entiers

- Si $0 \leq a \leq b < n$: $L[a:b]$ renvoie une copie de la portion, éventuellement vide, de la liste L comprise entre les positions a et b incluses, vide si $a = b$. Par exemple, si `L = [5, 8, 3, 7]`, `L[1:3]` crée la copie [8, 3], `L[0:1]` renvoie [5], `L[2:2]` renvoie [ ], `L[0:4]` renvoie une copie de L.
- Si $b \leq a$: `L[a:b]` renvoie une liste vide.

On considère une liste `L` d'entiers, non vide. On s'intéresse aux sous-suites croissantes de `L`.

Une sous-suite croissante de `L`, une sous-liste d'éléments consécutifs de `L`, compris entre deux positions d'indices `a` et `b` tels que (avec les conventions du *slicing* en Python) `L[a:b+1]` est croissante mais cette sous-liste étendue (si possible) à droite et à gauche ne l'est plus.

Exemples :

Si `L = [5, 4, -1, 3, 3, 17, 22, 19, 18, 20, 13, 20]`, `L` admet 3 sous-suites croissantes qui sont `[-1, 3, 3, 17, 22]`, `[18, 20]` et `[13, 20]`.

Si `L = [5, 5, 4, -1, 3, 3, 17, 22, 19, 18, 20, 13]`, `L` admet 3 sous-suites croissantes qui sont `[5, 5]`, `[-1, 3, 3, 17, 22]` et `[18, 20]`.

Q5. Écrire une fonction `variations`, prenant en argument une liste `L` d'entiers et renvoyant une liste `V`, de même longueur que `L`, et telle que :

- pour tout k compris entre 0 et `len(L)-1`, $V[k] = 1$ si `L[k] <= L[k+1]` et -1 sinon
- pour $k =$`len(L)-1`, $V[k] = -1$.

Exemples :

Si `L = [5, 4, -1, 3, 3, 17, 22, 19, 18, 20, 13, 20]`, on aura :

Indice de position dans la liste	0	1	2	3	4	5	6	7	8	9	10	11
Eléments de L	5	4	-1	3	3	17	22	19	18	20	13	20
Eléments de V	-1	-1	1	1	1	1	-1	-1	1	-1	1	-1

Si `L = [5, 5, 4, -1, 3, 3, 17, 22, 19, 18, 20, 13]`, on aura :

Indice de position dans la liste	0	1	2	3	4	5	6	7	8	9	10	11
Eléments de L	5	5	4	-1	3	3	17	22	19	18	20	13
Eléments de V	1	-1	-1	1	1	1	1	-1	-1	1	-1	-1

Q6. Expliciter en quoi la liste renvoyée par la fonction `variations` renseigne sur les sous-listes croissantes de `L`.

Q7. On souhaite écrire une fonction `sous_listes_croissantes`, prenant en argument une liste non vide d'entiers `L` et renvoyant une liste `S` composée des sous-listes croissantes de longueur supérieure ou égale à 2 dans `L`. On impose que cette fonction fasse appel à la fonction `variations` et que son implémentation respecte le squelette suivant :

```
[1]    def sous_listes_croissantes(L) :
[2]        V = [-1] + variations(L)
[3]        DebutFin = []
[4]        for k in range(len(V)):
[5]            if ... :
[6]                DebutFin.append(...)
[7]            elif ... :
[8]                DebutFin.append(...)
[9]        S = []
[10]       for i in range(...) :
[11]           S.append(L[... : ...])
[12]       return S
```

On rappelle que l'instruction en ligne 2 crée une liste `V` qui est la concaténation des listes `[-1]` et `variations(L)`.

Q7.a. Compléter le bloc d'instructions des lignes [5] à [8] afin que la liste `DebutFin` contienne les indices de début et de fin - alternativement, de toutes les sous-listes croissantes de *L* ayant au moins deux éléments.

Exemple : Si `L = [7, 5, 15, 20, 10, 13]`, la liste `DebutFin` devra valoir `[1, 3, 4, 5]`

Q7.b. Compléter le bloc d'instructions des lignes [10] à [12] afin que la liste `S` soit la liste de toutes les sous-listes croissantes de `L`.

Exercice 6. Chaînes de caractères

On rappelle le principe de construction d'une chaîne caractère par caractère à partir d'une chaîne vide au travers de la donnée de la fonction suivante, `copie`, qui permet de construire et de renvoyer une copie d'une chaîne de caractères, `ch`.

```
def copie(ch):
    ch1 = ""
    for i in range(len(ch)):
        ch1 = ch1 + ch[i]
    return ch1
```

Q1. Écrire une fonction `remplace(ch, old, new)` renvoyant une copie d'une chaîne `ch` en entrée dans laquelle les occurrences du caractère `old` ont été remplacée par le caractère `new`. Par exemple, l'appel `remplace("toto", "o", "i")` renverra la chaîne `"titi"`.

Q2. Écrire une fonction `renverse(ch)` renvoyant une copie d'une chaîne `ch` dans laquelle l'ordre des caractères est inversé. Par exemple, l'appel `renverse("toto")` renverra la chaîne `"otot"`.

Q3. Écrire une fonction `palindrome(ch)` renvoyant un booléen, `True` ou `False`, selon que la chaîne en entrée est un palindrome ou non. **La fonction fera appel à la fonction `renverse`.**

On rappelle qu'un mot palindrome est un mot qui peut se lire indifféremment de gauche à droite et de droite à gauche.

- Exemples :

```
>>> palindrome("toto")
False
```

```
>>> palindrome("kayak")
True
```

Q4. Afin d'obtenir une meilleure complexité pour la fonction (en évitant des copies et de comparaisons inutiles), proposer une fonction `palindrome2(ch)`, dans laquelle on effectue aucune copie de caractères, et un minimum de comparaisons de caractères, et qui renvoie de même un booléen indiquant si la chaîne est un palindrome ou non.

Exercice 7. Fonctions récursives

Les questions **Q1**, **Q2** et **Q3** sont indépendantes.

Q1. On considère la fonction récursive suivante :

```
1. def f(n, i):
2.     if i != n:
3.         print((n - i) * 'x' + i * 'y')
4.         f(n, i + 1)
```

On rappelle que l'opération `n * ch` ou `n` est un entier et `ch` une chaîne de caractères, produit, pour $n > 0$, une chaîne qui est la concaténation de n copies de la chaîne `ch`, et si n est négatif produit une chaîne vide.

Q1.a. Quels affichages obtient-on en exécutant l'appel `f(4, 2)` ?

Q1.b. Combien d'appels récursifs sont engendrés par l'appel `f(3, 0)` ? Quels sont les valeurs des paramètres n et i pour ces appels ?

Q1.c. Que se passe-t-il si on exécute l'appel `f(4, 5)` ? Donner une modification minimale du code de la fonction permettant de modifier ce comportement. Que se passe-t-il alors si on exécute l'appel `f(4, 5)` ?

Q1.d. **Si on échange les lignes 3 et 4 du code de la fonction `f`,** quels affichages obtient-on en exécutant l'appel `f(4, 2)` ? l'appel `f(3, 0)` ?

Q2. On considère la fonction récursive suivante :

```
1. def f(a, b):
2.     if a == 0 or a == b:
3.         return 0
4.     return f(a - 1, b - 1) + f(a, b - 1)
```

Q2.a. Dessiner l'arbre des appels à la fonction `f` pour l'appel initial `f(2, 3)`.

Q2.b. Que renvoie l'appel `f(2, 3)` ?

Q2.c. Justifier que par une modification minime de l'instruction en ligne 2 – on indiquera laquelle, pour deux entiers k, n tels que $0 \leq k \leq n$, un appel `f(k, n)` renverra la valeur de $\binom{n}{k}$.

Q3. On considère la suite $(u_n)_{n\geq 0}$ dite « suite de Syracuse » de premier terme N, définie par :

$$u_0 = N, \text{ et, pour tout entier } n \geq 0,\ u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

On admet ici la conjecture dite « de Syracuse », selon laquelle quelle que soit la valeur de l'entier N, il existe un indice n tel que $u_n = 1$.

Partie A programmation impérative

Dans cette partie, on utilisera si nécessaire des boucles, mais aucune fonction récursive.

Q3.a. Calculer les cinq premiers termes de la suite de Syracuse de premier terme $N = 10$.

Q3.b. Définir une fonction `syracuse`, ~~utilisant la fonction `f`~~, qui prend en argument les entiers N et n et renvoie le terme de rang n de la suite de Syracuse de premier terme N.

Q3.c. Définir une fonction `syracuse_vol`, utilisant la fonction `f`, qui prend en argument l'entier N et renvoie le temps de vol de la suite de Syracuse associée à N, c'est-à-dire le plus petit indice n tel que $u_n = 1$.

Partie B programmation récursive

Dans cette partie, on utilisera aucune boucle.

Q3.d. Écrire une fonction récursive `g` à deux paramètres, N et n, et renvoyant la valeur du terme de rang n de la suite de Syracuse de premier terme N.

Par exemple, `g(8, 0)` renverra 8, `g(15, 2)` renverra 23.

La question **Q3.e.** n'est pas à traiter lors du DS.

Q3.e. Écrire une fonction récursive `h(N, L)`, telle que l'appel `h(N, [])` (ou `h(N)` si on a donné à `L` la valeur par défaut `[]`) renvoie la liste de tous les termes de la suite de Syracuse de premier terme N, depuis son terme de rang 0, $u_0 = N$, jusqu'à son terme de rang $n_0 \geq 0$, inclus, où n_0 est le plus petit entier tel que $u_{n_0} = 1$.

Par exemple, `h(8, [])` renverra `[8, 4, 2, 1]`, `h(3, [])` renverra `[3, 10, 5, 16, 8, 4, 2, 1]`.

Exercice 8.

On rappelle les fonctions et méthodes suivantes pour lire et écrire dans des fichiers textes :

- `f = open(fichier, mode)` crée un « objet-fichier » permettant d'obtenir l'accès à un fichier. Le mode peut être `'r'` (lecture), `'w'` (écriture).
- `f.close()` ferme l'accès au fichier auquel l'accés a été demandé par `f`.
- `f.readlines()` avec `f` un « objet-fichier » donnant accès à un fichier ouvert en lecture, renvoie une liste de chaînes de caractères dont chacune correspond aux caractères composant une ligne du fichier, caractère de saut de ligne, `'\n'`, inclus ;
- `f.write(S)` : avec `f` un tel « wrapper » vers un fichier ouvert en écriture ou ajout, écrit dans le fichier la chaîne de caractères `S`.

On rappelle les méthodes `.strip()` et `.split()` sur les chaînes de caractères. S désignant une chaîne de caractères :

- `S.strip()` renvoie une nouvelle chaîne de caractères qui est une copie de S, privé de son dernier caractère si celui-ci est une espace ou un caractère de passage à la ligne `'\n'` ;
- `S.split(sep)`, `sep` désignant une chaîne de caractères, renvoie une liste de chaîne de caractères, qui est la liste de toutes les portions de la chaîne `S` (sans en changer l'ordre) qui sont comprises entre le début de la chaîne `S` et la première occurrence `sep`, ou entre deux occurrences consécutives de la chaîne `sep` dans `S`, ou entre la dernière occurrence de sep dans S et la fin de la chaîne `S`.
  Par exemple, on aura :

```
>>> 'tatin'.split('t')
['', 'a', 'in']
>>> '1,3,4,'.split(',')
['1', '3', '3', '']
```

On rappelle que l'opérateur de concaténation de deux chaînes de caractères se note « `+` ».

On rappelle les fonctions de conversion de type :

- `int` : appliquée à une chaîne de caractère représentant un chiffre, cette fonction renvoie l'entier correspondant ;
- `str` : appliquée à un entier cette fonction renvoie l'écriture décimale de cet entier sous la forme d'une chaîne de caractères.

Par exemple :

```
>>> int('23')
23
```

```
>>> str(102)
'102'
```

On considère dans cet exercice des tableaux à deux dimensions d'entiers T, de taille 9×9, dont les éléments sont des entiers compris entre 0 et 9.

Q1. Donner une instruction permettant de créer, sans utiliser la méthode `.append`, un tableau T, de taille 9×9, dont toutes les valeurs sont égales à 0, de sorte que l'on ait :

```
>>> T
[[0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Q2. Que réalise l'imbrication de boucles suivantes sur le tableau T défini à la question précédente ?

```
for i in range(len(T)):
    for j in range(i, len(T[0])):
        T[i][j] = 1
```

Q3. Écrire une séquence d'instructions permettant modifier le tableau *T* défini à la question **Q1.** de sorte à ce que chaque ligne de T soit égale à `[1, 2, 3, 4, 5, 6, 7, 8, 9]`.

Q4. Écrire une séquence d'instructions permettant modifier le tableau *T* défini à la question **Q1.** de sorte à ce que l'on ait :

```
>>> T
[[1, 2, 3, 4, 5, 6, 7, 8, 9],
 [2, 3, 4, 5, 6, 7, 8, 9, 1],
 [3, 4, 5, 6, 7, 8, 9, 1, 2],
 [4, 5, 6, 7, 8, 9, 1, 2, 3],
 [5, 6, 7, 8, 9, 1, 2, 3, 4],
 [6, 7, 8, 9, 1, 2, 3, 4, 5],
 [7, 8, 9, 1, 2, 3, 4, 5, 6],
 [8, 9, 1, 2, 3, 4, 5, 6, 7],
 [9, 1, 2, 3, 4, 5, 6, 7, 8]]
```

Q5. Écrire une séquence d'instructions permettant de calculer la somme de tous les éléments d'un tableau T à deux dimensions.

On souhaite enregistrer un tel tableau *T* à deux dimensions dans un fichier texte `'sauv.csv'`, de sorte que le tableau T de la question **Q4.**, par exemple, sera sauvegardé sous la forme :

sauv.csv

```
1;2;3;4;5;6;7;8;9
2;3;4;5;6;7;8;9;1
3;4;5;6;7;8;9;1;2
4;5;6;7;8;9;1;2;3
5;6;7;8;9;1;2;3;4
6;7;8;9;1;2;3;4;5
7;8;9;1;2;3;4;5;6
8;9;1;2;3;4;5;6;7
9;1;2;3;4;5;6;7;8
```

Q6. Écrire une séquence d'instructions permettant, à partir d'une liste de neuf entiers compris entre 0 et 9, $L = [c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9]$ de construire une chaîne `ch`, qui est la concaténation des éléments de L, convertis chacun au type `str` et séparés par des caractères « ; ».

Par exemple, la liste $L = [8, 9, 1, 2, 3, 4, 5, 6, 7]$ sera été transformée en la chaîne `'8;9;1;2;3;4;5;6;7'`.

Q7. Écrire une fonction `sauvegarde(T, nomfichier)`, permettant de sauvegarder un tableau T dans un fichier texte dont le nom sera donné par `nomfichier`. On pourra réutiliser les instructions écrites en question **Q6.**

On souhaite maintenant définir une fonction réalisant l'opération inverse de la fonction précédente.

Q8. Rappeler, si l'on exécute la séquence d'instructions suivantes avec le fichier **sauv.csv** décrit ci-avant, le type de la variable `c`, et la valeur de `len(c)`, `len(c[0])` et de `c[0]`.

```
f = open("sauv.csv")
c = f.readlines()
f.close()
```

Q9. Écrire une instruction ou une séquence d'instructions permettant de créer à partir d'une chaîne `ch` constituée d'une suite d'entiers écrits en base 10 et séparés par des points-virgules, la liste `L` de ces entiers.

Par exemple, à partir de la chaîne `ch = '14,7,0,8'`, on devra produire la liste L égale à `[14, 7, 0, 8]`.

Q10. Compléter le code de la fonction `charge(nomfichier)` ci-dessous de sorte qu'elle réalise l'opération inverse de celle réalisée par la fonction `sauvegarde` : un appel à la fonction `charge`, avec en entrée le nom d'un fichier texte contenant le codage d'un tableau T, renverra le tableau T sous la forme d'une liste de listes.

```
def charge(nomfichier):
    f = ...
    c = f.readlines()
    f.close()
    T = []
    for ligne in c :
        ...
        T.append(...)
    return T
```