Devoir Surveillé 1 : Correction

1 Références partagées

Dans chacun des trois cas suivants, qu'affiche `print(a[1])` ? Justifier en décrivant (éventuellement à l'aide d'un dessin) les évolutions des états de la mémoire.

Question 1. Premier cas :

```
a = [1, 3]
b = a
b[1] = 6
```

- Python crée une liste de longueur 2. Le premier pointeur de cette liste renvoie vers l'entier 1 et le deuxième vers 3. Python crée aussi une variable `a`, qui pointe vers cette liste.
- Python crée une variable `b`, qui pointe vers la même liste que `a`.
- Python modifie le pointeur de la case d'indice 1 de la liste vers laquelle pointe `b` et lui associe désormais la valeur 6.

Comme la variable `a` pointe vers la même liste que la variable `b`, il s'ensuit que `print(a[1])` renvoie la valeur `6`.

Question 2. Deuxième cas :

```
a = ["alpha", "beta"]
b = a[:]
b[1] = 'gamma'
```

- Une liste de longueur 2 est crée, dont l'élément d'indice 0 pointe vers la chaîne de caractères de longueur un `"alpha"` et l'élément d'indice 0 pointe sur la chaîne de caractères `"beta"`. Une variable `a` est créée qui pointe vers cette liste.
- Python crée une deuxième liste de longueur 2, qui pointe vers les mêmes chaîne de caractères que `a` et une variable `b` pointant vers cette chaîne.
- L'élément d'indice 1 de la liste `b` pointe désormais vers la chaîne de caractères `"gamma"` (il n'y a pas de différence ici entre guillemets simples et doubles).

Comme la variable `a` ne pointe vers la même liste que la variable `b`, il s'ensuit que `print(a[1])` renvoie `beta` (sans guillemets, alors que `print(a)` renverrait `['alpha','beta']`), indiquant le type `string`.

Question 3. Troisième cas :

```
a = [True, [True]]
b = a[:]
b[1][0] = False
```

`print(a[1])` renvoie `[False]`. *Cf.* cours ou explication orale.

2 Parcours de listes

Question 4. Écrire une fonction `test01` qui prend en argument une liste L et renvoie le booléen `True` si cette liste ne contient que des 0 et des 1 et qui renvoie le booléen `False` sinon.

Voici deux solutions. Il y en a d'autres. La première est plus concise, mais potentiellement plus lente car elle parcourt toute la liste.

```
def test01(L):
      return L.count(0) + L.count(1) == len(L)

def test01(L):
      for x in L:
            if x!= 0 and x!= 1:
                  return False
      return True
```

Question 5. Écrire une fonction `testlongueur` qui prend en argument une liste L et renvoie l'entier correspondant à la longueur de la plus grande sous-liste de L composée uniquement de 1.

Par exemple, `testlongueur([0,1,0,1,1,1,0,1,0,1,1,1,1])` renverra 4.

La moindre des choses est de trouver une solution dont la complexité soit linéaire. On peut écrire deux fonctions, l'une comptant le nombre de 1 à partir de l'indice `i` et une autre utilisant la première et effectuant des sauts de puce. Voici une solution avec une seule fonction. Noter que les tests `i<n` sont mis dans les deux boucles. Dans le test `i < n and L[i] == 1`, l'ordre est fondamental car Python n'évalue pas toute l'expression booléenne s'il tombe sur un `out of range`... On notera enfin l'usage efficace de l'affectation par tuple.

```
def testlongueur(L):
      a, b, i, n = 0, 0, 0, len(L) # a est le nombre courant de 1 consécutifs, b est le record
      while i < n:
            while i < n and L[i] == 1:
                  a += 1; i += 1
            i += 1;
            a, b = 0, max(a,b)
      return b
```

Question 6. La loi d'une variable aléatoire discrète finie X sur $[\![0,n]\!]$ est représentée par une liste `L` de $n+1$ éléments où `L[i]` = $\mathbf{P}(X=i)$.

1. Écrire une fonction `testva(L)` qui teste si une liste `L` peut représenter la loi d'une variable aléatoire à valeurs dans $[\![0,n]\!]$. On rappelle que cela signifie que `L[i]` $\in[0,1]$ et que $\sum_{i=0}^{n}$`L[i]` $=1$. Cette fonction renverra un booléen.

```
def testva(L):
      s = 0
      for x in L:
            if not 0 <= x <= 1:
                  return False
            s += x
      return abs(s - 1) < 1e-10
```

Le test avec la valeur absolue est préférable à une égalité `s == 1`, toujours susceptible de tomber en défaut numériquement, pour des raisons de codage des flottants.

2. Écrire une fonction `esperance(L)` qui renvoie l'espérance $\mathbf{E}(X)$ où X est la v.a. dont la loi est représentée par `L`, soit $\mathbf{E}(X)=\sum_{i=0}^{n} i\mathbf{P}(X=i)$ si `L` représente bien une v.a. et un message d'erreur sinon (on utilisera la fonction de la question 1). On n'utilisera pas la commande `sum`.

```
def esperance(L):
      assert testva(L)
      E = 0
      for i in range(len(L)):
            E += i * L[i]
      return E
```

Noter ici l'intérêt de parcourir la liste par ses indices.

3 Chaîne de caractères

Question 7. Écrire une fonction `chaine_caractere` qui prend en argument une chaîne de caractères `chaine` et un caractère `caractere` et qui renvoie le booléen indiquant si `caractere` est présent dans `chaine`.

Une première solution consiste à parcourir la liste (plus exactement ses éléments, il n'y a aucun intérêt à parcourir les indices ici) et tester la présence du caractère cherché *à la main*. Le `return False` ne sera éventuellement exécuté qu'après le parcours de la boucle. Cette solution doit être maîtrisée :

```
def chaine_caractere(chaine, caractere):
      for ch in chaine:
            if ch==caractere:
                  return True
      return False
```

Il est également possible d'utiliser une "pythonnerie", même si les sujets d'écrit pourraient interdire l'utilisation du `in` et qu'un examinateur d'oral pourrait demander une solution alternative.

```
def chaine_caractere(chaine, caractere):
      return caractere in chaine
```

Question 8. Écrire une fonction `chaine_souschaine` qui prend en argument deux chaînes de caractères `chaine` et `souschaine` et qui renvoie le booléen indiquant si la sous-chaîne `souschaine` est présente dans `chaine`.

La recherche d'une sous-chaîne dans une chaîne de caractères est au programme de première année. La première solution, qui consiste à écrire une double boucle imbriquée doit être connue. Voici une solution possible :

```
def recherche_souschaine(ch,sousch):
      for i in range(len(ch)-len(sousch)+1):
            c=0
            while c<len(sousch) and ch[i+c]==sousch[c]:
                  c=c+1
            if c==len(sousch):
                  return True
      return False
```

Il est également possible, avec les mêmes réserves qu'à la question précédente, de se contenter d'utiliser les techniques de slicing et d'un boucle simple :

```
def chaine_souschaine(chaine, souschaine):
      for i in range(len(chaine)-len(souschaine)+1):
            if chaine[i:i+len(souschaine)]==souschaine:
                  return True
      return False
```

Enfin, le mot-clé `in` dévoile toute sa puissance dans la solution suivante :

```
def chaine_souschaine(chaine, souschaine):
      return souschaine in chaine
```

Question 9. Un fichier externe, nommé `"smalltalk.txt"`, stocké dans le même répertoire que les scripts Python contient un texte.

Écrire une suite d'instructions qui permet d'ouvrir ce fichier, de le lire et de tester si la chaîne de caractères `"info"` est présente dans le texte.

```
fichier = open('smalltalk.txt', 'r', encoding = 'utf8')
chaine = fichier.open()
fichier.close()
print(chaine_souschaine(chaine, "info"))
```

Noter que le `encoding=...` est optionnel et ne doit être utilisé que si l'on connaît effectivement l'encodage du fichier externe.

4 Listes de listes

Question 10. La fonction `zip` construit un itérateur associant en tuples des éléments de plusieurs listes en s'arrêtant quand la liste la plus courte est épuisée. Par exemple :

```
list(zip([1,3,5], [2,4,6,8])) = [(1,2), (3,4), (5,6)]
```

Écrire une fonction `monzip` qui prend en arguments deux listes `L1` et `L2` et qui renvoie la liste de tuples de 2 éléments qui serait renvoyée par la commande `list(zip(L1,L2))`.

```
def monzip(L1, L2): # détruit les listes
    P = []
    while L1 != [] and L2 != []:
        P.append((L1.pop(0),L2.pop(0)))
    return P
```

```
def monzip(L1, L2): # ne détruit pas les listes
    P = []
    m = min(len(L1), len(L2))
    for i in range(m):
        P.append((L1[i], L2[i]))
    return P
```

Question 11. Écrire une fonction `monzip2` qui prend en argument une liste de listes `LL` et qui renvoie la liste de tuples de $n =$`len(LL)` éléments qui serait renvoyée par la commande `list(zip(LL))`.

```
def monzip2(LL):
    P = []
    m = min([len(x) for x in LL])
    for i in range(m):
        P.append(tuple(x[i] for x in LL))
    return P
```

Il est à noter que la syntaxe `(e(i) for i in L)` ne construit pas un tuple mais un générateur d'objet. On pourra donc transtyper en `tuple` une liste construite par compréhension ou utiliser une construction comme celles proposée par le corrigé.