Algorithmique
Partiel n° 1 (P1)

Info-sup S1
Epita

7 janvier 2020 - 13h-15h

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00

Exercice 1 (Types Abstraits : liste itérative (supprimer) – *3 points*)

Supposons le type abstrait algébrique *liste itérative* vu en cours et dont la signature est rappelée ci-dessous.

TYPES
liste, place

UTILISE
entier, élément

OPÉRATIONS

liste-vide	:	$\rightarrow$ liste
accès	:	liste $\times$ entier $\rightarrow$ place
contenu	:	place $\rightarrow$ élément
ième	:	liste $\times$ entier $\rightarrow$ élément
longueur	:	liste $\rightarrow$ entier
insérer	:	liste $\times$ entier $\times$ élément $\rightarrow$ liste
succ	:	place $\rightarrow$ place

Le type abstrait donné ci-dessus est incomplet. L'opération de suppression a été oubliée...
Son profil est le suivant :

OPÉRATIONS

supprimer	:	liste $\times$ entier $\rightarrow$ liste

où *supprimer*(λ, k) supprime le $k^{ème}$ élément de la liste λ (k commence à 1).

1. Précisez l'éventuel domaine de définition de l'opération *supprimer*.
2. Donner les axiomes définissant de manière suffisamment complète l'opération *supprimer*.

Exercice 2 (Dichotomie : "chemin" de recherche – *2 points*)

Supposons des listes d'entiers triées en ordre croissant. Si on effectuait dans celles-ci une recherche dichotomique de la valeur 66 :

Parmi les séquences suivantes, lesquelles pourraient correspondre à la suite des valeurs rencontrées lors de la recherche ?

① 46 - 65 - 81 - 73 - 70 - 66
② 31 - 62 - 90 - 72 - 61 - 66
③ 36 - 70 - 53 - 40 - 42 - 66
④ 35 - 51 - 55 - 58 - 61 - 66

Procédé de Kaprekar

Les exercices de cette partie sont indépendants. Le dernier exercice utilise les fonctions des exercices précédents, mais il n'est pas obligatoire de les avoir écrites.
Pour toutes les fonctions, on supposera que les paramètres sont valides : il n'y a pas de test à faire, pas d'exception à déclencher.

Exercice 3 (Test - *1 point*)

Soit la fonction `test` suivante :

```
1  def test(x, L):
2      i = len(L) - 1
3      while i >= 0 and L[i] != x:
4          i = i - 1
5      return i >= 0
```

Que fait cette fonction ?

Exercice 4 (Entiers ↔ liste – *6 points*)

1. Écrire la fonction `int_to_list(`n`, `p`)` qui convertit le nombre n (entier naturel à au plus p chiffres) en une liste de ses chiffres éventuellement complétés par des `0` pour atteindre p chiffres.

   Exemples d'applications (l'ordre des chiffres dans le résultat importe peu) :

```
>>> int_to_list(27972, 5)
[2, 7, 9, 7, 2]

>>> int_to_list(42, 5)
[2, 4, 0, 0, 0]

>>> int_to_list(258, 4)
[8, 5, 2, 0]
```

2. Écrire la fonction `list_to_ints(`L`)` qui, à partir de la liste L non vide, ne contenant que des chiffres (de 0 à 9), retourne le couple $(left, right)$, avec
   — $left$: le nombre construit avec les chiffres de L "lus" de gauche à droite,
   — $right$: le nombre construit avec les chiffres de L "lus" de droite à gauche.

   Exemples d'applications :

```
>>> list_to_ints([1, 2, 3, 4, 5, 6])
(123456, 654321)

>>> list_to_ints([2, 4, 0, 0, 0])
(24000, 42)
```

Exercice 5 (Histogramme et tri – *4 points*)

Nous travaillons ici avec des listes qui ne contiennent que des chiffres (de 0 à 9).

1. Écrire la fonction `hist(`L`)` qui retourne un histogramme (sous la forme d'une liste) des chiffres présents dans la liste L.

   Rappel : l'histogramme H est une liste telle que $H[i]$ est le nombre d'occurrences de la valeur i. Par exemple dans la première application ci-dessous, il y a 5 valeurs 0, 3 valeurs 1, 3 valeurs 2....

   Exemples d'applications :

```
>>> hist([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[5, 3, 3, 1, 0, 3, 0, 0, 0, 1]
>>> hist([1,0,1,0,1,0,1,0,1])
[4, 5, 0, 0, 0, 0, 0, 0, 0, 0]
```

2. Utiliser la fonction `hist` pour écrire la fonction `sort(`L`)` qui trie la liste L en ordre croissant (la fonction construit une nouvelle liste qui doit être retournée).

   Exemple d'application :

```
>>> sort([1,5,9,3,0,1,2,0,1,0,2,5,0,5,2,0])
[0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 5, 5, 5, 9]
```

Exercice 6 (Kaprekar – *5 points*)

Procédé de Kaprekar :

Soit un nombre entier n à p chiffres. Par exemple $n = 6264$.

— Prendre les p chiffres de n pour former deux nombres : le plus grand et le plus petit : 6642 et 2466.

— Soustraire en complétant éventuellement par des 0 à gauche pour obtenir à nouveau un nombre à p chiffres : $n = 6642 - 2466 = 4176$.

— Recommencer le procédé avec le résultat. $4176 \rightarrow 7641 - 1467 = 6174$

Le procédé de Kaprekar peut être utilisé avec un nombre quelconque. Dans tous les cas, quelque soit le nombre de chiffres, **on arrivera à une valeur déjà rencontrée** (dans les exemples donnés plus loin : on retrouve 6174 dans le premier, et 63 dans le deuxième), qui sera la valeur 0 si tous les chiffres étaient identiques.

Remarque :

Nous travaillons ici toujours avec p chiffres. Cela signifie que si un résultat intermédiaire est inférieur à 10^{p-1} (par exemple 999 lorsque $p = 4$), les deux nombres seront construits à partir des chiffres du nombre complétés par des 0 (ici 999 et 9990).

La fonction à écrire `Kaprekar(`n`, `p`)` prend en paramètre un entier naturel n et son nombre de chiffres p et applique le procédé de Kaprekar. Elle construit et retourne la liste les différentes valeurs calculées c, comme dans les exemples suivants.

Exemples d'applications :

```
>>> Kaprekar(1574, 4)
[1574, 6084, 8172, 7443, 3996, 6264, 4176, 6174, 6174]  # 6174 met twice

>>> Kaprekar(42, 2)
[42, 18, 63, 27, 45, 9, 81, 63]  # 63 met twice

>>> Kaprekar(666, 3)
[666, 0, 0]   # 0 met twice
```

Vous pouvez utiliser les fonctions des exercices 3 à 5 même si elles ne sont pas écrites.

Annexe : Fonctions et méthodes autorisées

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes ainsi que la fonction `range` :

```
>>> L = []

>>> for i in range(5):
        L.append(i)

>>> L
[0, 1, 2, 3, 4]

>>> len(L)
5
```

Aucun opérateur n'est autorisé sur les listes (`+`, `*`, `==` ...).

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.