Lists

1 *Liste Itérative* → Python

Type abstrait : Liste itérative λ : Liste	Python : type list type(L) = list
$\lambda \leftarrow$ *liste-vide*	L = []
longueur(λ)	len(L)
ième(λ, k)	L[k]

Exercice 1.1 (ALGO ↦ Python)

Traduire les deux algorithmes suivants en Python.

1.
```
fonction compte(Element x, Liste λ) : entier
   variables
      entier    i, cpt
debut
   cpt ← 0
   pour i ← 1 jusqu'a longueur(λ) faire
      si x = ieme(λ, i) alors
         cpt ← cpt + 1
   fin pour
   retourne cpt
fin
```

2.
```
fonction est-present(Element x, Liste λ) : booleen
   variables
      entier    i
debut
   i ← 1
   tant que (i <= longueur(λ)) et (x <> ieme(λ, i)) faire
      i ← i + 1
   fin tant que
   retourne (i <= longueur(λ))
fin
```

3. Modifier la fonction précédente : elle retourne la position du premier x trouvé et la liste en paramètre est triée (en ordre croissant)

Construire une liste

Exercice 1.2 (Construire une liste)

```
>>> help(list.append)
Help on method_descriptor:
append(...)
    L.append(object) -> None -- append object to end
>>> L = []
>>> L.append(1)
>>> L.append(2)
>>> L.append(3)
>>> L
[1, 2, 3]
```

Écrire une fonction qui retourne une nouvelle liste de n valeurs *val*.

Exercice 1.3 (Type abstrait ↦ Python)

Implémenter les opérations suivantes en Python.

1. *L'opération supprimer*

   OPÉRATIONS
   supprimer : Liste × Entier → Liste

   PRÉCONDITIONS
   supprimer(λ, k) **est-défini-ssi** $1 \leq k \leq longueur(\lambda)$

   AXIOMES
   $\lambda \neq liste\text{-}vide \;\&\; 1 \leq k \leq longueur(\lambda) \Rightarrow longueur(supprimer(\lambda, k)) = longueur(\lambda) - 1$
   $\lambda \neq liste\text{-}vide \;\&\; 1 \leq k \leq longueur(\lambda) \;\&\; 1 \leq i < k \Rightarrow ième(supprimer(\lambda, k), i) = ième(\lambda, i)$
   $\lambda \neq liste\text{-}vide \;\&\; 1 \leq k \leq longueur(\lambda) \;\&\; k \leq i \leq longueur(\lambda) - 1 \Rightarrow ième(supprimer(\lambda, k), i) = ième(\lambda, i + 1)$

   AVEC
   λ : Liste
   k, i : Entier

2. *L'opération insérer*

   OPÉRATIONS
   insérer : Liste × Entier × Élément → Liste

   PRÉCONDITIONS
   insérer(λ, k, e) **est-défini-ssi** $1 \leq k \leq longueur(\lambda) + 1$

   AXIOMES
   $1 \leq k \leq longueur(\lambda) + 1 \Rightarrow longueur(insérer(\lambda, k, e)) = longueur(\lambda) + 1$
   $1 \leq k \leq longueur(\lambda) + 1 \;\&\; 1 \leq i < k \Rightarrow ième(insérer(\lambda, k, e), i) = ième(\lambda, i)$
   $1 \leq k \leq longueur(\lambda) + 1 \;\&\; k = i \Rightarrow ième(insérer(\lambda, k, e), i) = e$
   $1 \leq k \leq longueur(\lambda) + 1 \;\&\; k < i \leq longueur(\lambda) + 1 \Rightarrow ième(insérer(\lambda, k, e), i) = ième(\lambda, i - 1)$

   AVEC
   λ : Liste
   k, i : Entier
   e : Élément

Comment implémenter ces opérations "en place" ?

2 Une impression de déjà vu

Exercice 2.1 (Histogramme)

1. Écrire une fonction qui donne un histogramme des caractères présents dans une chaîne de caractères : une liste de longueur 256 qui donne pour chaque caractère son nombre d'occurences dans la chaîne.
2. Écrire une fonction qui compte le nombre de caractères différents dans une chaînes de caractères.
3. Écrire une fonction qui retourne le caractère le plus fréquent d'une chaîne, ainsi que son nombre d'occurrences.

Exercice 2.2 (Ératosthène)

Écrire une fonction qui donne la liste de tous les nombres premiers jusqu'à une valeur n donnée. Utiliser la méthode du "crible d'Eratosthène" (voir wikipedia).

3 Ordre et tris

Exercice 3.1 (Colline – *Final P1# 2017*)

Une liste est une colline si elle est constituée d'une suite (éventuellement vide) croissante suivie d'une suite (éventuellement vide) décroissante.

Exemples

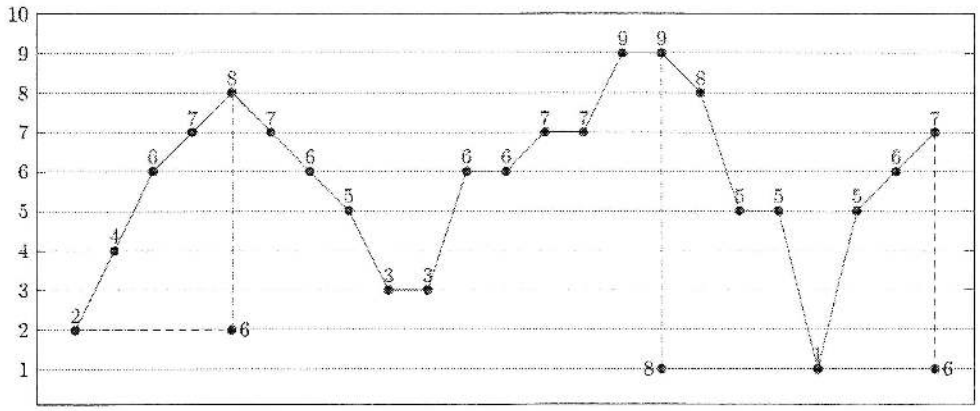
Les listes suivantes sont des collines :
- 1, 4, 8, 12, 5, 2
- 12, 25, 40, 52
- 15, 8, 3

Les listes suivantes ne sont pas des collines :
- 1, 5, 10, 8, 6, 12
- 15, 12, 11, 9, 10, 14, 16

1. Écrire une fonction qui détermine si une liste non vide est une colline. Si c'est le cas, elle retourne son point culminant (la valeur la plus haute), -1 sinon.
2. Écrire la fonction `hills_size` qui détermine la hauteur maximum des collines présentes dans une liste d'entiers.

```
hills_size([3,6,9,10])                                  # retourne 7
hills_size([8,5,2])                                     # retourne 6
hills_size([2,10])                                      # retourne 8
hills_size([6,6,6,6])                                   # retourne 0
hills_size([2,4,6,7,8,7,6,5,3,3,6,6,7,7,9,9,8,5,5,1,5,6,7])  # retourne 8
```

Exemple visuel de la dernière liste testée :



Exercice 3.2 (Communs - *Final P1# 2017*)

Soient deux listes d'entiers **strictement décroissantes**. Écrire la fonction `shared(`L_1`,` L_2`)` qui retourne le nombre d'éléments communs aux deux listes L_1, L_2.

Exemples :

	0	1	2	3	4	5	6	7	8	9
L_1	15	12	7	5	4	-1	-2	-3	-6	-8

	0	1	2	3	4	5	6	7	8
L_2	8	7	4	3	0	-2	-3	-5	-6

Le nombre d'éléments communs aux deux listes L_1 et L_2 est 5.

```
>>> shared([15,12,7,5,4,-1,-2,-3,-6,-8], [8,7,4,3,2,0,-2,-3,-5,-6])
5

>>> shared([15,12,7,5,4], [3,2,0,-2,-3,-5,-6])
0
```

Modifier la fonction pour quelle retroune la liste des éléments communs.

Exercice 3.3 (Tri par sélection (Select Sort))

1. Écrire la fonction `minimum` qui détermine la position de la valeur minimum dans une liste.
2. Utiliser la fonction précédente (en la modifiant si nécessaire) pour écrire une fonction qui trie une liste en ordre croissant.

en place

Exercice 3.4 (Tri par insertion (Insertion Sort))

1. Écrire une fonction qui insère un élément x à sa place dans une liste L triée.
2. Utiliser la fonction précédente pour écrire une fonction qui trie une liste.

en place

Exercice 3.5 (Tri à bulles (bubble sort))

Implémenter en Python le *tri à bulles*. (On pourra également modifier ce tri pour obtenir un *shaker sort*...)

4 The Problem : Kaprekar

Procédé de Kaprekar :

Soit un nombre entier à p chiffres.

- Prendre les chiffres pour former deux nombres : le plus grand et le plus petit.
- Soustraire en complétant éventuellement par des 0 à gauche pour obtenir à nouveau un nombre de p chiffres.
- Recommencer le procédé avec le résultat.

Remarques :

- 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).
- Les chiffres du nombres de départ ne doivent pas être tous égaux.
- Le procédé de Kaprekar peut être utilisé avec un nombre de chiffres p quelconque. Selon ce nombre p, on s'arrêtera avec une valeur fixe, ou on obtiendra un cycle.

Écrire un script Python qui applique le procédé de Kaprekar et affiche les différentes valeurs calculées.