Lists

Le sujet de TD

1 *Liste Itérative* → Python

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

En Algo , les listes sont indexées à partir de 1, en Python, elles sont en indexées à partir de 0.

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 si
   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
```

Exercice 1.2 (Type abstrait ↦ Python)

Implémenter l'opération suivante en Python.

L'opération supprimer

OPÉRATIONS

$supprimer$: Liste × Entier → Liste

PRÉCONDITIONS

$supprimer(\lambda, k)$ **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

Comment implémenter cette opération "en place" (la liste est modifiée par la fonction, aucune autre liste ne doit être utilisée) ?

2 Constructions

Exercice 2.1 (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 2.2 (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'occurrences 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.

3 Ordre et tris

Exercice 3.1 (Recherche dans une liste triée)

Modifier la dernière fonction de l'exercice 1.1 (est-present) : elle retourne la position du premier x trouvé (-1 sinon) et la liste en paramètre est cette fois-ci supposée triée en ordre croissant.

Exercice 3.2 (Remplacement)

Écrire la fonction `replace(L, k, new_v)` où `L` est une liste de couples `(key, value)` triés par clés `(key)` croissantes. Elle remplace la valeur `(value)` associée à la clé `k` par la nouvelle valeur `new_v` si la clé est présente. Les valeurs sont supposés non ordonnables et les clés sont supposées distinctes.

Exemples d'applications :

```
>>> L = [(1, 'one'), (2, 'two'), (3, 'three'), (5, 'five'), (8, 'eight')]
>>> replace(L, 3, 'trois')
>>> L
[(1, 'one'), (2, 'two'), (3, 'trois'), (5, 'five'), (8, 'eight')]
>>> replace(L, 4, 'quatre')
>>> L
[(1, 'one'), (2, 'two'), (3, 'trois'), (5, 'five'), (8, 'eight')]
```

Exercice 3.3 (Croissant)

Écrire la fonction `is_sorted(L)` qui vérifie si les éléments de la liste L passée en paramètre sont triés dans l'ordre croissant.

Exemples d'applications :

```
>>> is_sorted([1, 5, 5, 12, 25])
True
>>> is_sorted([4, 3, 2, 1])
False
>>> is_sorted([4])
True
>>> is_sorted([])
True
```

Complexité ?

En notant n le nombre d'éléments de la liste, donner pour chaque tri ci-dessous :

— le nombre de comparaisons entre éléments effectuées ;

— le nombre de copies d'éléments.

Exercice 3.4 (Tri par sélection (Selection Sort))

1. Écrire la fonction `minimum(`L`, `d`, `f`)` qui détermine la position de la valeur minimum dans la liste L entre les positions d et f comprises (avec $0 \leq d < f < len(L)$).

   Par exemple, dans la liste suivante :

	0	1	2	3	4	5	6	7	8	9
L	4	-5	3	-3	8	-2	0	3	-6	7

   Entre les positions $d = 2$ et $f = 7$, le minimum est à la position 3.

2. Utiliser la fonction précédente pour écrire une fonction qui trie une liste en ordre croissant **en place** (la liste est modifiée par la fonction, aucune autre liste ne doit être utilisée).

   Exemple d'application :

```
>>> L = [4, -5, 3, -3, 8, 2, 0, 3, -6, 7]
>>> select_sort(L)
>>> L
[-6, -5, -3, 0, 2, 3, 3, 4, 7, 8]
```

3. **Bonus** Quels problèmes pose l'implémentation **pas en place** du tri par sélection ? (en particulier en python)

Exercice 3.5 (Tri par insertion (Insertion Sort))

1. Écrire une fonction qui insère un élément `(k, v)` à sa place dans une liste `L` où `L` est une liste de couples `(key, value)` triés par clés `(key)` croissantes. Les valeurs sont supposés non ordonnables et les clés sont supposées distinctes.

2. Utiliser la fonction précédente pour écrire une fonction qui trie en ordre croissant une liste de couples `(key, value)` selon les clés `(key)` (pas en place : une nouvelle liste est construite). Les valeurs sont supposés non ordonnables et les clés sont supposées distinctes.
   Exemples d'applications :

```
>>> L = [(5, 'five'), (3, 'three'), (1, 'one'), (2, 'two'), (8, 'eight')]
>>> insertion_sort(L)
[(1, 'one'), (2, 'two'), (3, 'three'), (5, 'five'), (8, 'eight')]
```

3. **Bonus** Comment implémenter en place le tri par insertion ?

Exercice 3.6 (Bonus : Tri à bulles (Bubble Sort))

Implémenter en Python le *tri à bulles*.

Indice : les seules opérations nécessaires pour implémenter le tri à bulles sont la comparaison de deux éléments consécutifs et leur changement de position si besoin.

4 Diviser pour régner (Divide and Conquer)

Exercice 4.1 (Dichotomie (Binary Search))

1. Écrire une version plus optimale de la fonction qui recherche un élément x dans une liste triée L : elle retourne la position de x s'il est présent dans L, la position où il devrait être sinon.
2. Avec cette nouvelle version de la recherche, la suppression et l'insertion dans une liste triée sont-elles plus optimales ?

Exercice 4.2 (Tri fusion (Merge sort))

Pour trier une liste `L`, on procède (récursivement) de la façon suivante :

- ▷ Une liste de longueur < 2 est triée.
- ▷ Une liste de longueur ≥ 2 :
  - — on partitionne la liste `L` en deux sous-listes `L1` et `L2` de longueurs quasi identiques (à 1 près) ;
  - — on trie récursivement les deux listes `L1` et `L2` ;
  - — enfin, on fusionne les listes `L1` et `L2` en une liste triée.

Ecrire la fonction `merge_sort` qui trie en ordre croissant une liste (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

Exemple d'application :

```
>>> merge_sort([5,3,2,8,7,1,5,4,0,6,1])
[0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

5 Bonus

Exercice 5.1 (Bonus : Ératosthène)

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

Exercice 5.2 (Bonus : Quicksort ?)

Voici une version Caml du *quicksort* :

```
# let rec partition_pivot x = function
    [] -> [],[]
  | e::l -> let (l1,l2) = partition_pivot x l in
              if e < x then
                (e::l1,l2)
              else
                (l1,e::l2) ;;
val partition_pivot : 'a -> 'a list -> 'a list * 'a list = <fun>

# partition_pivot 5 [1; 8; -1; 5; 6; 7; 10; 3] ;;
- : int list * int list = ([1; -1; 3], [8; 5; 6; 7; 10])

# let rec quick_sort =function
    [] -> []
  | e::l -> let (l1,l2) = partition_pivot e l in
              (quick_sort l1) @ (e::(quick_sort l2)) ;;
val quick_sort : 'a list -> 'a list = <fun>

# quick_sort  [5; 8; 3; 1; -1; 0; -5; 12; -1; 8; 10; 7; 3] ;;
- : int list = [-5; -1; -1; 0; 1; 3; 3; 5; 7; 8; 8; 10; 1
```

Implémenter le *quicksort* en Python, en optimisant au maximum bien entendu.