III - TP 1

Listes et algorithmes récursifs

1 Présentation

Stocker les données est un problème fondamental en informatique. Nous verrons au cours de cette année différentes manières de le faire. Étant donné une collection finie de valeurs a, b, c, ..., on décide de les stocker de manière *séquentielle* : le premier élément de la liste est a, le second b, etc.

On utilise pour cela la structure de *liste*. Si l est une liste de nombres, on dispose alors des fonctions suivantes :

Fonction	Description
`creer_vide()`	Renvoie une liste vide
`est_vide(l)`	Renvoie `True` si et seulement si la liste est vide
`tete(l)`	Renvoie le premier élément de la liste l (l'élément dit **de tête**)
`queue(l)`	Renvoie la liste constituée de tous les éléments de l à l'exception du premier
`ajoute(l, e)`	Renvoie la liste constituée de l'élément e, suivi des éléments de l
`affiche(l)`	Affiche la liste des éléments de l sur la sortie standard, séparés par le caractère `-`. La liste vide est représentée par `x`.

On considère la liste $l = (0, 1, 42, "Bonjour", \pi)$.

1. Que renvoie `est_vide(l)` ?
2. Que renvoie `tete(l)` ?
3. Que renvoie `queue(l)` ? Quelle est le type de `queue(l)` ?
4. Que renvoie `ajoute(l, 93)` ?
5. Que fait l'instruction `affiche(l)` ?

Remarque. On peut donner une définition récursive de la structure de liste. En effet, une liste c'est : .....
.....

1.1 Interface

Code python

```
from Liste import creer_vide, est_vide, tete, queue, ajoute, affiche

l = creer_vide()
affiche(l)
l = ajoute(l, 0)
affiche(l)
l = ajoute(l, 1)
ajoute(l, 2)
affiche(l)
```

Résultat

```
x
0 - x
1 - 0 - x
```

Commenter l'affichage réalisé.

1.2 Premier exemple

Écrire les instructions python permettant de stocker les listes

- $l_1 = ()$
- $l_2 = (-1)$
- $l_3 = (5, 6, 9, -1)$
- $l_4 = (-5, 9, 4, 9, -5, 9)$.

Vous utiliserez uniquement les fonctions `creer_vide`, `ajoute`. Vous n'avez pas le droit d'utiliser des tableaux.

Code python

```
# l1 = creer_vide()
# ...
```

Code python

```
affiche(l1)
affiche(l2)
affiche(l3)
affiche(l4)
```

Résultat

```
x
-1 - x
5 - 6 - 9 - -1 - x
-5 - 9 - 4 - 9 - -5 - 9 - x
```

2 Exercices

Toutes les questions suivantes doivent être traitées en utilisant uniquement les fonctions décrites dans le paragraphe d'introduction. Vous n'avez pas le droit d'utiliser les tableaux. Vous n'avez pas non plus le droit d'utiliser de boucle `for`, ou de boucle `while`.

2.1 Liste singleton

Écrire une fonction `est_singleton` qui prend en argument une liste `l` et qui renvoie `True` si la liste est constituée d'un seul élément, `False` dans tous les autres cas.

Code python

```
def est_singleton(l):
    """ Liste -> bool
    Détermine si la liste est constituée d'un seul élément. """
    pass
```

Code python

```
for l in [l1, l2, l3, l4]:
    print(est_singleton(l), end = " ")
```

Résultat

```
False True False False
```

2.2 Création de liste I

Écrire une fonction `singleton` qui prend en argument un entier `n` et qui renvoie la liste ayant pour seul élément le nombre `n`.

Code python

```
def singleton(n):
    """ int -> Liste
    Renvoie la liste (n) """
    pass
```

Code python

```
affiche(singleton(3))
```

Résultat

```
3 - x
```

2.3 Création de liste II

Écrire une fonction récursive `nombres` qui prend en argument un entier `n` supérieur à 1 et qui renvoie la liste des entiers de `n` (inclu) à 1 (inclu) dans l'ordre décroissant.

Code python

```
def nombres(n):
    """ int -> Liste
    Renvoie la liste (n, n-1, n-2, ..., 3, 2, 1) """
    pass
```

Code python

```
affiche(nombres(5))
```

Résultat

```
5 - 4 - 3 - 2 - 1 - x
```

2.4 Création de liste III

On se propose dans cet exercice d'écrire une fonction `nombresII` qui prend en argument un entier `n` supérieur à 1 et qui renvoie la liste des entiers de 1 (inclus) à `n` (inclu) dans l'ordre croissant.

- Écrire une fonction `nombresII_aux` qui étant donné deux entiers `n` et `i`, renvoie la liste constituée des entiers de `i` à `n` dans l'ordre croissant.
- En déduire une fonction `nombresII` qui renvoie la liste dans l'ordre croissant des entiers de 1 à `n`. Vous appellerez pour cela la fonction `nombresII_aux` avec des arguments bien choisis.

Code python

```
def nombresII_aux(n, i):
    """ int, int -> liste
    Renvoie la liste de nombres (i, i + 1, ..., n-1, n) """
    pass

def nombresII(n):
    """ int, int -> liste
    Renvoie la liste de nombres (1, 2, ..., n-1, n) """
    pass
```

Code python

```
affiche(nombresII(5))
```

Résultat

```
 1 - 2 - 3 - 4 - 5 - x
```

2.5 Longueur d'une liste

On définit la **longueur** d'une liste comme étant le nombre d'éléments constituant cette liste. Par exemple, si $l = (0, 1, 5, -59)$, la longueur de l est 4. Par convention, la longueur de la liste vide est 0. Écrire une fonction python récursive `longueur` qui étant donné une liste, en renvoie la longueur.

Code python

```
def longueur(l):
    """ Liste -> int
    Renvoie la longueur de la liste l """
    pass
```

Code python

```
for i in range(1, 4):
    l = nombres(i)
    print(longueur(l), end = " ")
```

Résultat

```
 1 2 3
```

2.6 Appartenance à une liste

Écrire une fonction python récursive `appartient` qui étant donné une liste `l` et un élément `e`, renvoie `True` si et seulement si l'élément `e` est un des éléments de la liste `l`.

Code python

```
def appartient(l, e):
    """ Liste, int -> bool
    Détermine si l'élément e fait partie de la liste l """
    pass
```

Code python

```
l = nombres(5)
for e in [-1, 5, 1, 0]:
    print(appartient(l, e), end = " ")
```

Résultat

```
 False True True False
```

2.7 Nombre d'occurrences

Écrire une fonction python récursive `nombre_occurrences` qui étant donné une liste `l` et un élément `e` détermine le nombre d'occurrences de l'élément `e` dans la liste `l`.

Quelle est l'avantage de l'instruction `appartient(l, e)` par rapport à l'instruction `nombre_occurrences(l, e) == 0` pour déterminer si l'élément `e` appartient à la liste `l` ?

Code python

```python
def nombre_occurrences(l, e):
    """ Liste, int -> int
    Compte le nombre d'occurrences de e dans l """
    pass
```

Code python

```python
for e in [-5, 4, 9, 0]:
    print(nombre_occurrences(l4, e), end = " ")
```

Résultat

```
2 1 3 0
```

2.8 Somme des éléments d'une liste

Écrire une fonction python récursive `somme` qui étant donnée une liste `l` d'entiers non vide en renvoie la somme.

Code python

```python
def somme(l):
    """ Liste -> int
    Calcule la somme des éléments de la liste l """
    pass
```

Code python

```python
print(somme(nombres(10))) # 1 + 2 + ... + 10 = 55
```

Résultat

```
55
```

2.9 Maximum des éléments d'une liste

Écrire une fonction `maximum2` qui étant donné deux entiers `a` et `b` renvoie le plus grand entier parmi `a` et `b`.

En déduire une fonction python récursive `maximum` qui étant donnée une liste d'entiers non vide en renvoie le plus grand.

Code python

```python
def maximum2(a, b):
    """ int, int -> int
    Calcule l'élément maximum parmis a et b
    """
    pass

def maximum(l):
    """ Liste -> int
    Renvoie le plus grand élément de l """
    pass
```

Code python

```python
for l in [l2, l3, l4]:
    print(maximum(l), end = " ")
```

Résultat

```
-1 9 9
```

3 Pour aller plus loin

3.1 Suppression d'un élément

Écrire une fonction `supprime` qui prend en argument une liste `l` et un entier `e`, et renvoie la liste des éléments de `l` dans laquelle la première occurrence de `e` a été supprimée. On suppose que l'élément `e` appartient à la liste `l`. La liste initiale ne sera pas modifiée.

Code python

```python
def supprime(l, e):
    """ Liste, int -> Liste
    Supprime la première occurrence de e la liste l """
    pass
```

Code python

```python
affiche(supprime(l4, 9))
affiche(l4)
```

Résultat

```
-5 - 4 - 9 - -5 - 9 - x
-5 - 9 - 4 - 9 - -5 - 9 - x
```

3.2 Insérer dans une liste

Écrire une fonction `insere` qui étant donné une liste `l`, un élément `e` et un entier `i` renvoie la liste constituée des éléments de `l` dans lequel l'élément `e` a pour indice `i`. On supposera que `i` est un entier tel que $0 \leq i \leq \texttt{len}(l)$. Aucun comportement particulier n'est attendu dans le cas où `i` ne vérifierait pas ces conditions. La liste initiale `l` ne sera pas modifiée.

Code python

```python
def insere(l, e, i):
    """ Liste, int, int -> Liste
    Insère l'élément e dans la liste l à l'indice i """
    pass
```

Code python

```python
affiche(insere(l3, 7, 2))
affiche(l3)
```

Résultat

```
5 - 6 - 7 - 9 - -1 - x
5 - 6 - 9 - -1 - x
```

3.3 Concaténation de deux listes

Si l_1 et l_2 sont deux listes, on appelle **concaténation de** $l1$ **avec** $l2$ la liste constituée des éléments de l_1 puis des éléments de l_2. Par exemple, la concaténation des listes $l_1 = (0,1,6)$ et $l_2 = (9,4,1)$ est la liste $(0,1,6,9,4,1)$; par contre la concaténation des listes l_2 et l_1 (dans cet ordre) est la liste $(9,4,1,0,1,6)$. La concaténation de la liste vide avec la liste l_1 est la liste $(0,1,6)$.

Écrire une fonction python récursive `concatene` qui étant donné deux listes `l1` et `l2` renvoie la concaténation de ces deux listes. Les listes `l1` et `l2` ne seront pas modifiées.

Code python

```
def concatene(l1, l2):
    """ Liste, Liste -> Liste
    Concatène les deux listes """
    pass
```

Code python

```
affiche(concatene(l3, l4))
affiche(l3)
affiche(l4)
```

Résultat

```
5 - 6 - 9 - -1 - -5 - 9 - 4 - 9 - -5 - 9 - x
5 - 6 - 9 - -1 - x
-5 - 9 - 4 - 9 - -5 - 9 - x
```

3.4 Division de listes

1. Écrire une fonction `est_2ton` qui prend en argument une liste `l` et qui renvoie `True` si et seulement si la liste `l` est constituée d'exactement deux éléments.

2. On souhaite écrire une fonction `divise` qui divise en deux une liste `l`. Les longueurs des deux listes résultantes doivent différer au plus d'un, et tous les éléments de la liste `l` doivent se retrouver dans l'une uniquement des deux listes de sortie. Ainsi si `divise(l)`, renvoie les listes `l1` et `l2`, on a : `concatene(l1, l2)` qui contient les mêmes éléments que `l`.

   a. `l` est la liste $(1,2,3,4,5)$. Parmi les propositions ci-dessous, déterminer celle qui peuvent-être renvoyées par l'instruction `divise(l)` :

   - `l1 = (1, 2, 3),l2 = (3, 4, 5)`
   - `l1 = (1, 3, 5),l2 = (2, 4)`
   - `l1 = (1, 3),l2 = (2, 4)`
   - `l1 = (1, 3, 4, 5),l2 = (2)`

   Justifier votre réponse.

   b. En déduire une fonction divise qui réponde au problème posé. Pour le cas général, on remarquera que si une liste `l` est constituée d'au moins trois éléments, on peut la diviser en deux de la manière suivante :

   - on met à part les deux premiers éléments `x1` et `x2` de la liste `l` ;
   - on divise récursivement en deux listes `l1` et `l2` les éléments restants ;
   - on ajoute à `l1` l'élément `x1` et à `l2` l'élément `x2`.

Code python

```
def divise(l):
    """ Liste -> Liste, Liste
    Divise la liste l en deux listes """
    pass
```

Code python

```python
for l in [l1, l2, l4]:
    print("Liste initiale : ", end= ""), affiche(l)
    lp, ls = divise(l)
    print("Premier morceau : ", end= ""), affiche(lp)
    print("Second morceau : ", end= ""), affiche(ls)
    print()
```

Résultat

```
Liste initiale : x
Premier morceau : x
Second morceau : x

Liste initiale : -1 - x
Premier morceau : -1 - x
Second morceau : x

Liste initiale : -5 - 9 - 4 - 9 - -5 - 9 - x
Premier morceau : -5 - 4 - -5 - x
Second morceau : 9 - 9 - 9 - x
```

4 Pour les plus fous

L'objectif ici est de déterminer l'ensemble des parties d'un ensemble E à n éléments, que l'on note $\mathcal{P}(E)$. Par exemple, si l'ensemble E est $\{1,2,3\}$, l'ensemble des parties de E est :

$$\mathcal{P}(E) = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$$

On remarquera plusieurs choses :

- les éléments de $\mathcal{P}(E)$ sont des ensembles ;
- $\emptyset \in \mathcal{P}(E)$ (on note parfois l'ensemble vide $\{\}$ ou $\emptyset$) et $E \in \mathcal{P}(E)$;

1. Si $E = \emptyset$, que vaut $\mathcal{P}(E)$?

2. Soit $E = \{1,2,3\}$, et soit $e = 3$.

a. Déterminer la liste de tous les sous-ensembles de E qui ne contiennent pas e. On note cette liste E'.

b. Déterminer la liste de tous les sous-ensembles de E qui contiennent e. On note cette liste E''.

c. Écrire E en fonction de E' et de E''. On pourra utiliser les opérations ensemblistes $\cup$ et $\cap$.

3. Déduire des fonctions suivantes une fonction python `liste_sous_ensembles` qui étant donné un ensemble E, détermine la liste de tous ses sous-ensembles. On utilisera pour cela le type python `list`, ainsi que les méthodes `pop`, et `append`. On supposera que tous les éléments des listes manipulées sont uniques.

4. Si E possède n éléments, combien d'éléments possède $\mathcal{P}(E)$?

Code python

```python
def liste_sous_ensembles(E):
    """ list -> list
    Renvoie la liste de tous les
    ↪ sous-ensembles de E """
    pass
```

Code python

```python
print(liste_sous_ensembles([1, 2]))
```

Résultat

```
[[], [1], [2], [1, 2]]
```