Structure de données - Les listes

Introduction

Remarque :
Python abuse du terme liste qu'il utilise pour ce qui sont des tableaux dynamiques munis de méthodes d'accès typiques des listes. Nous nous intéressons ici à ce que les informaticiens appellent vraiment des listes.

Une liste est une structure de données permettant de regrouper des données.
Elle est composée de 2 parties :

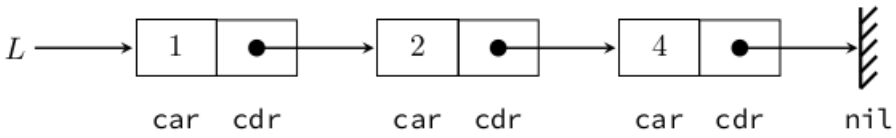
- **sa tête notée car** (pour : *contents of address register*), qui correspond au dernier élément ajouté à la liste.
- **sa queue notée cdr** (pour : *contents of decrement register*), qui correspond au reste de la liste.

Le langage de programmation Lisp (inventé par John McCarthy en 1958) a été un des premiers langages de programmation à introduire cette notion de liste (Lisp signifie "list processing").

Voici les opérations qui peuvent être effectuées sur une liste :

- créer une liste vide (souvent notée **nil**)
- tester si une liste est vide
- ajouter un élément en tête de liste
- supprimer la tête d'une liste et renvoyer cette tête
- compter le nombre d'éléments présents dans une liste
- un constructeur, historiquement appelé **cons**, qui permet d'obtenir une nouvelle liste à partir d'une liste et d'un élément (L1 = cons(x,L)).
  Il est possible "d'enchaîner" les cons et d'obtenir ce genre de structure :
  cons(x, cons(y, cons(z,L)))

Voici la représentation schématique d'une liste comportant, dans cet ordre, les trois éléments 1, 2 et 4.



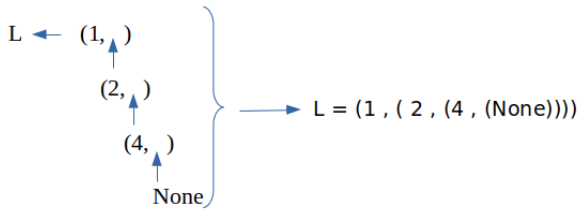
La liste est constituée de trois cellules, représentées ici par des rectangles.
Chaque cellule possède une première partie, nommée car , qui contient l'entier correspondant, et d'une deuxième partie, nommée cdr , qui contient un lien vers la cellule suivante.
On a donc affaire à une chaîne de cellules, ce qui justifie la dénomination courante de liste chaînée.
Le champ cdr de la dernière cellule renvoie vers une liste vide, conventionnellement représentée par un hachurage.
On aura noté que le premier champ d'une cellule contient ici un entier (car on considère une liste d'entiers) alors que le deuxième est une liste, c'est-à-dire un lien vers une autre cellule.

Nous avons défini les contours de la structure de liste. Celle-ci est un concept ("une vue de l'esprit") de ce qu'est une liste.
On dit que c'est un **un type abstrait de données**.

Pour implémenter cette structure il faudra, quelque soit le langage utilisé, que l'implémentation permette de retrouver les fonctions qui ont été définies pour le type abstrait.

Une 1ère implémentation de la structure liste

En utilisant des tuples pour implémenter la structure de liste.




```
def vide():
    return None
def cons(x,L):
    return(x,L)
def ajouteEnTete(L,x):
    return cons(x,L)
def supprEnTete(L):
    return (L[0],L[1])
def estVide(L):
    return L is None
def compte(L):
    if estVide(L):
        return 0
    return 1 + compte(L[1])
```

À faire 1:

Vérifier le bon fonctionnement de cette implémentation en exécutant ces instructions:

- nil = vide()
- afficher estVide(nil)
- L = cons(5, cons(4, cons(3, cons(2, cons(1, cons(0,nil))))))
- afficher estVide(L)
- afficher compte(L)
- L = ajouteEnTete(L,6)
- afficher compte(L)
- x, L=supprEnTete(L)
- afficher x
- afficher compte(L)
- x, L=supprEnTete(L)
- afficher x
- afficher compte(L)

Une 2ème implémentation

Cette fois-ci on utilise deux classes:

La classe Cellule qui crée un objet (instance) cellule avec les attributs **car** et **cdr**

```
class Cellule:
    def __init__(self, tete, queue):
        self.car = tete
        self.cdr = queue
```

La classe Liste dont les instances (objets) seront de type Cellule, avec quelques méthodes :

```
class Liste:
    def __init__(self, c):
        self.cellule = c
    def estVide(self):
        return self.cellule is None
    def car(self):
        assert not(self.cellule is None), 'Liste vide'
        return self.cellule.car
    def cdr(self):
        assert not(self.cellule is None), 'Liste vide'
        return self.cellule.cdr
```

La fonction **cons** qui permet de construire la liste:

```
def cons(tete, queue):
    return Liste(Cellule(tete, queue))
```

Ainsi pour créer une la liste L=[5, 4, 3, 2, 1, 0], il suffit d'exécuter les instructions:

```
nil=Liste(None)
L = cons(5, cons(4, cons(3, cons(2, cons(1, cons(0,nil))))))
```

À FAIRE 2:

tester les instructions suivantes, et écrire l'instruction qui permet d'afficher dernier élément de la liste.

```
print(L.estVide())
print(L.car())
print(L.cdr().car())
print(L.cdr().cdr().car())
```

.....
.....
.....

À FAIRE 3:

Voici deux fonctions à rajouter :

```
def longueurListe(L):
    n = 0
    while not(L.estVide()):
        n += 1
        L = L.cdr()
    return n
```

```
def listeElements(L):
    t = []
    while not(L.estVide()):
        t.append(L.car())
        L = L.cdr()
    return t
```

Que font-elles ?

.....
.....
.....
.....

À FAIRE 4:

Que produit l'instruction:

```
L=cons(6,L)
```

.....
.....

À FAIRE 5:

Écrire une fonction `ajouteEnTete` qui prend en paramètre une liste et un nombre et qui renvoie une liste dans la quelle le nombre a été ajouté en entête.

.....
.....

À FAIRE 6:

Que produisent les instructions:

```
x=L.car()
L=cons(L.cdr().car(),L.cdr().cdr())
```

.....
.....
.....

À FAIRE 7:

Écrire une fonction `supprEnTete` qui prend en paramètre une liste et qui retourne son entête et la liste amputée de l'entête.

.....
.....
.....
.....

Pour nos usages, on utilisera les listes de Python (qui sont bien plus que de simple listes). Mais bien pratiques...