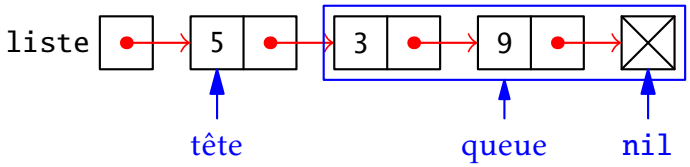
Listes chaînées

Structures de données

Les listes sont probablement un des types construits les plus utilisés en Python. Mais il ne s'agit pas de ce que l'on appelle une liste de façon générale en informatique. Les listes Python sont en fait des tableaux redimensionnables qui peuvent faire plus de choses que les listes "classiques". Elles permettent, par exemple, de facilement ajouter un élément à la fin. Par contre, pour ajouter un élément au début, il faut rajouter un élément fictif à la fin, puis recopier tous les éléments dans la position suivante, avant d'insérer la valeur désirée en première position. Le coût de cet algorithme est proportionnel au nombre d'éléments. Insérer un élément en tête est donc de plus en plus coûteux au fur et à mesure qu'il y a des éléments.
Si on a besoin d'insérer des éléments au début, ce n'est pas une structure appropriée. En fait, en fonction des besoins, on peut être amené à utiliser diverses structures. Savoir identifier celle à utiliser dans telle ou telle situation est très important pour pouvoir résoudre efficacement des problèmes complexes.

Listes chaînées

Une **liste chaînée**, ou plus simplement **liste**, est composée d'une suite de **maillons**, ou **cellules**, contenant chacun une valeur et un pointeur vers le maillon suivant. La liste est simplement un pointeur vers le premier maillon. Afin de terminer cette liste, on utilise un maillon vide, souvent noté `nil`.



L'inferface d'une liste donne l'accès à au moins 4 fonctions :

Fonction	Description
`tete(liste)`	Renvoie la valeur du premier maillon de `liste`, qui ne doit pas être vide.
`queue(liste)`	Renvoie la liste sur laquelle pointe le premier maillon de `liste`, qui ne doit pas être vide.
`cons(valeur, liste)`	Renvoie une nouvelle liste correspondant à l'ajout de `valeur` en début de `liste`.
`est_vide(liste)`	Renvoie un booléen indiquant si `liste` est vide ou non.

Pour une liste non vide, `liste` et `cons(tete(liste), queue(liste))` sont équivalents.
On note parfois `v::l` pour représenter une liste, où `v` est la tête et `l` la queue. L'opérateur `::` correspond à l'ajout d'un élément en tête.
La figure précédente peut donc être définie par `cons(5, cons(3, cons(9, nil)))` ou plus simplement `5::3::9::nil`.

Parcours d'une liste

Pour les fonctions nécessitant de manipuler autre chose que le contenu du premier maillon, comme la recherche d'un élément ou le calcul de la longueur, il faut forcément parcourir

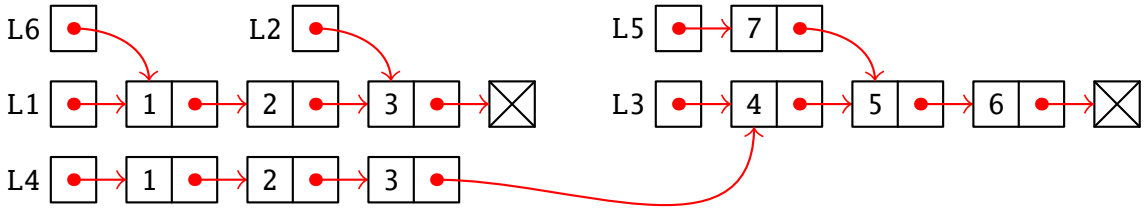
la liste. La construction de la liste se prête particulièrement à des fonctions récursives. Les fonctions suivantes peuvent s'écrire assez facilement en faisant un parcours de la liste :

Fonction	Description
`longueur(liste)`	Renvoie le nombre d'éléments de `liste`. La longueur de la liste vide est de 0.
`appartient(v, liste)`	Renvoie un booléen indiquant si `v` est dans `liste`.
`concat(L1, L2)`	Renvoie une nouvelle liste correspondant à l'ajout de `L2` à la fin d'un copie de `L1`.
`n_ieme(i, liste)`	Renvoie la valeur de l'élément d'indice `i` de `liste` s'il existe et sinon une exception ou un code erreur.
`renverse(liste)`	Renvoie une liste correspondant à `liste` dans l'ordre inverse.

Voici la représentation des listes obtenues avec les instructions suivantes :

```
L1 = cons(1, cons(2, cons(3, nil)))
L2 = queue(queue(L1))
L3 = cons(4, cons(5, cons(6, nil)))
```

```
L4 = concat(L1, L3)
L5 = cons(7, queue(L3))
L6 = L1
```



Exercice 1 : Déterminer le résultat des expressions suivantes, à l'aide de la figure ci-dessus.

1) `appartient(4, L1)`
2) `longueur(L4)`
3) `tete(queue(L5))`
4) `est_vide(queue(L2))`
5) `appartient(6, L5)`
6) `n_ieme(4, L4)`

Exercice 2 : Représenter graphiquement le résultat des instructions suivantes :

```
L1 = cons(5, cons(4, cons(3, nil)))
L2 = renverse(L1)
L3 = cons(tete(L2), queue(L1))
```

```
L4 = concat(L1, cons(7, L2))
L5 = cons(8, queue(L3))
L6 = queue(queue(L2))
```

Immuables ou pas

Aucune des fonctions précédentes ne modifie les listes. Pourtant, écrire des fonctions qui le font n'est pas plus compliqué. On pourrait par exemple ajouter un élément au début d'une liste ou au contraire enlever la tête. On pourrait aussi modifier la valeur du i-ième élément, ou complétement l'enlever. Mais il faut faire attention avec de telles fonctions.
Si on prend l'exemple de l'exercice 1, si on modifie le dernière élément de la liste `L3`, on modifiera également celui de `L4` ou `L5`. C'est ce que l'on appelle un **effet de bord**. Ce n'est pas forcément une mauvaise chose, mais il faut en avoir conscience.

D'autres types de listes

Il existe d'autres types de listes. Dans les **listes cycliques**, le dernier maillon pointe sur le premier et non sur `nil`. Dans les **listes doublements chaînées** chaque maillon dispose de 2 pointeurs : un vers le suivant et un autre vers le précédent. Cela permet de revenir en arrière lorsqu'on parcourt la liste.