Listes chaînées

Chapitre 16

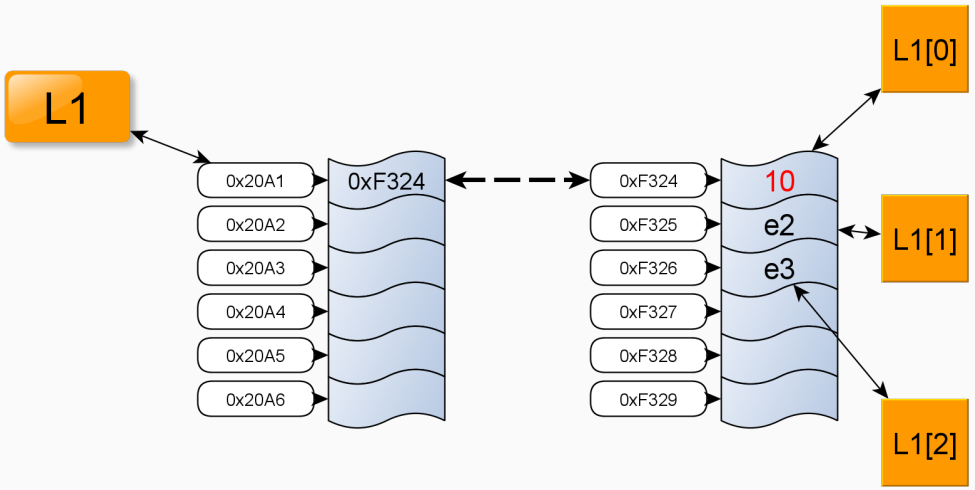
NSI2

2 décembre 2021

Caractéristiques

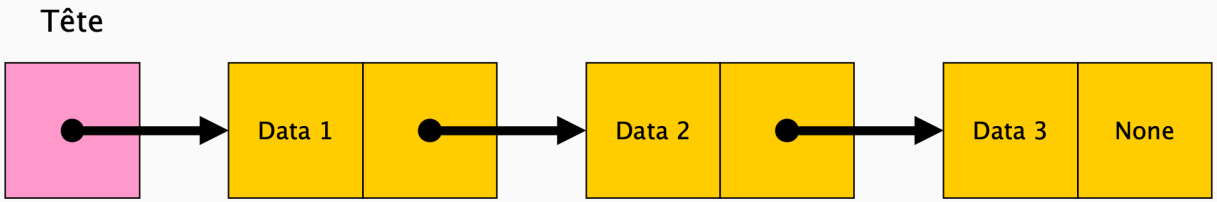
- Rien à voir avec le type `list` de Python...
- Structure linéaire : éléments rangés « les uns à la suite des autres ».
- Cet ordre est un ordre sur les places des éléments, pas sur les éléments.

Type `list` de Python



Pour simplifier, une variable de type `list` est un peu comme un tableau : Les adresses des éléments sont connues et l'accès à tel ou tel élément se fait en une seule fois (on dit « en temps constant »).

Liste



La structure de donnée *liste* est bien différente.

Interface

Interface de la structure de donnée

- *liste_vide()*
  renvoie une... liste vide.
- *element(liste, position)*
  renvoie l'élément à la position donnée.
- *longueur(liste)*
  renvoie la longueur de la liste.
- *supprimer(liste, position)*
  supprime l'élément de la liste qui est à la position donnée.
- *inserer(liste, valeur, position)*
  insère la valeur à la position donnée.

On convient que, comme en Python, la première position est 0.

Pré-conditions

Ce sont les conditions qui doivent être vérifiées pour pouvoir se servir de l'interface. Par exemple :

element(liste, position) n'est défini que si $position < longueur(liste)$.

Il y en a d'autres, elles sont faciles à trouver.

Axiomes (hors programme)

En théorie on doit énoncer des propriétés appelées *axiomes*, pour que la structure de donnée soit cohérente. Par exemple :

- *longueur(liste_vide)=0* ;
- si liste est non vide et que $0 \leqslant k \leqslant longueur(liste)$ alors

$$longueur(supprimer(liste, k)) = longueur(liste) - 1$$

- si $0 \leqslant k \leqslant longueur(liste)$ alors

$$longueur(inserer(liste, valeur, k)) = longueur(liste) + 1$$

Il y en a d'autres, nous n'en parlerons pas

Intérêt des pré-conditions et des axiomes

- Connaître les pré-conditions est utile lors de l'implémentation, pour lever des erreurs.
- Les axiomes peuvent servir à fabriquer des tests unitaires.

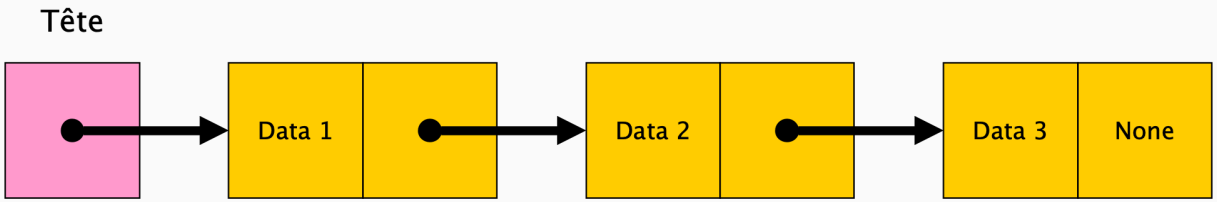
Implémentations

- Avec une variable de type `list` . (Trop facile !)
- De même, encapsulé dans un objet
- En POO.

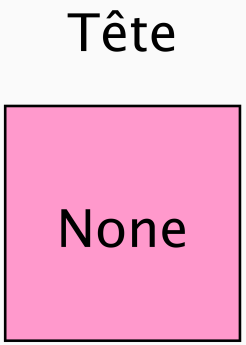
Liste chaînée

Pour disposer d'une liste chaînée non vide, il faut un lien vers le premier élément de la liste : la tête.
Chaque élément de la liste doit contenir :

- une valeur (on supposera qu'elles sont toutes du même type) ;
- un lien vers la cellule suivante, le dernier élément ne contenant pas de lien (on notera `None` l'absence de lien).



Fonction liste_vide



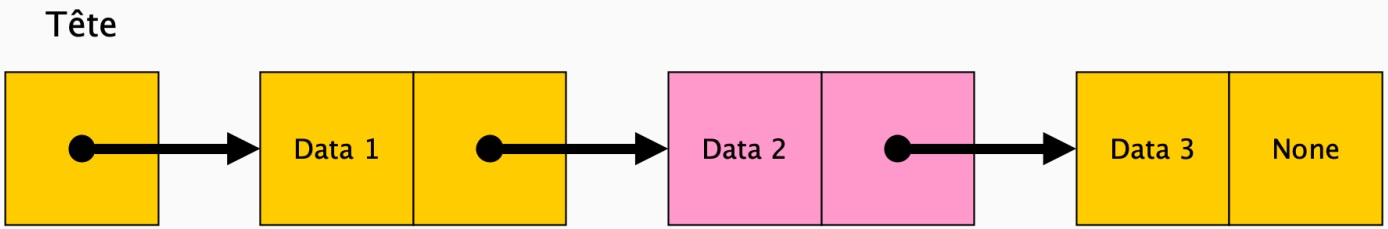
Fonction element

element(liste,position) parcourt la liste en commençant par la tête et passant au maillon suivant tant que nécessaire.

Fonction longueur

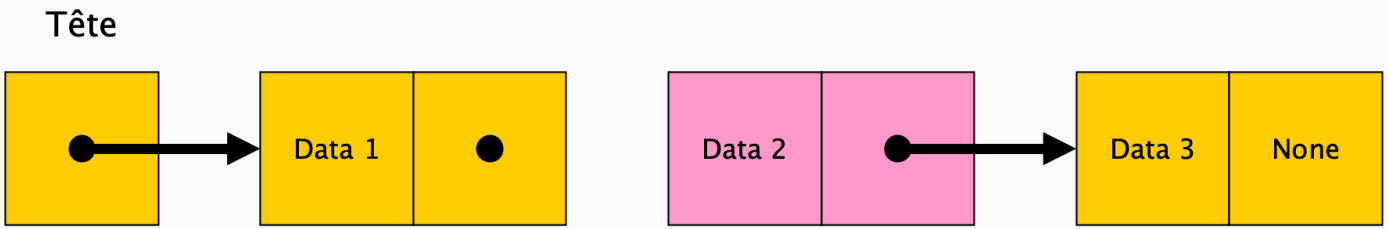
longueur(liste) parcourt la liste en commençant par la tête jusqu'au dernier maillon (qui n'a pas de successeur) et en comptant combien il y en a.

Supprimer



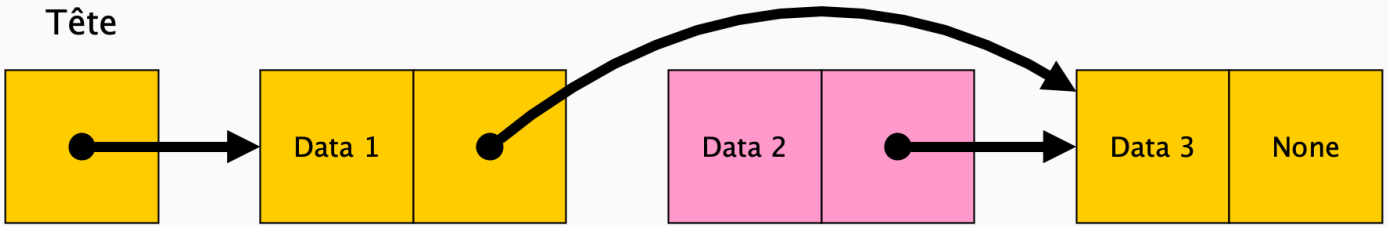
On veut supprimer un élément.

Supprimer



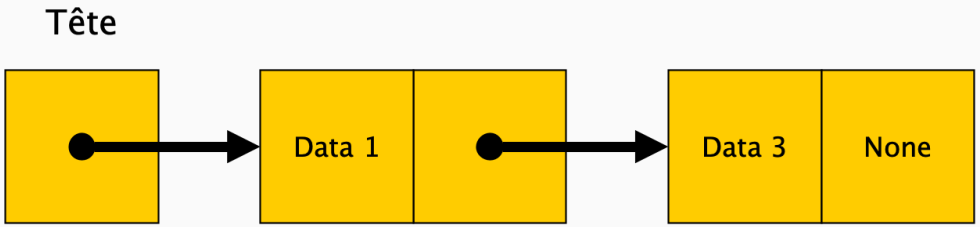
On le déconnecte de son prédécesseur (ou de la tête).

Supprimer



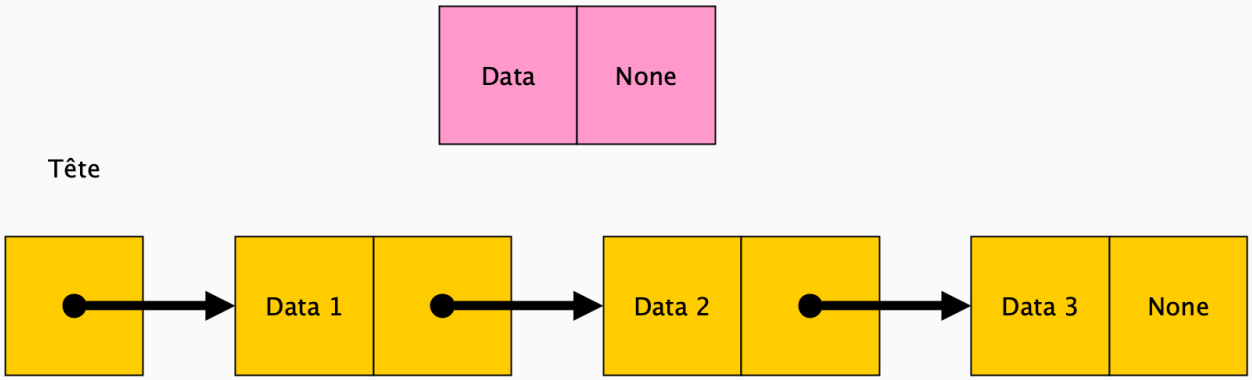
On connecte son prédécesseur à son successeur (s'il y en a un).

Supprimer



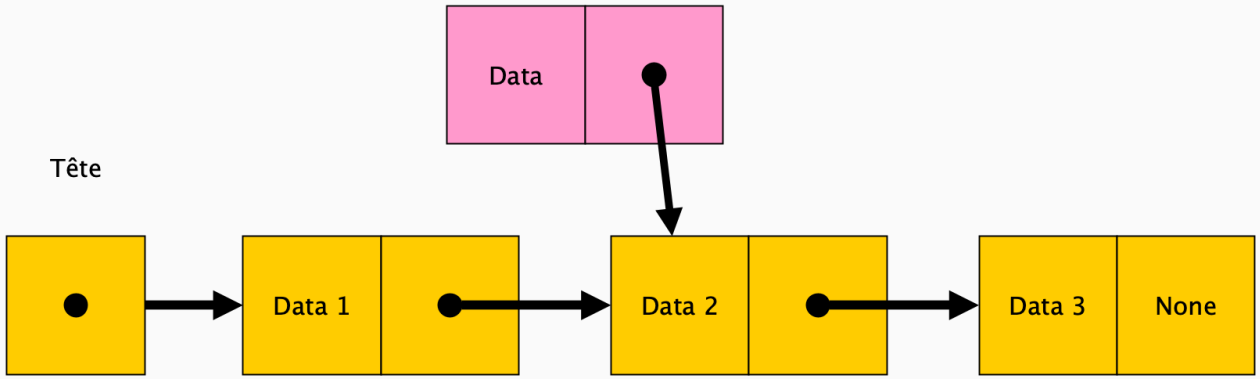
On « efface » définitivement l'élément (avec un `del` par exemple).

Inserer



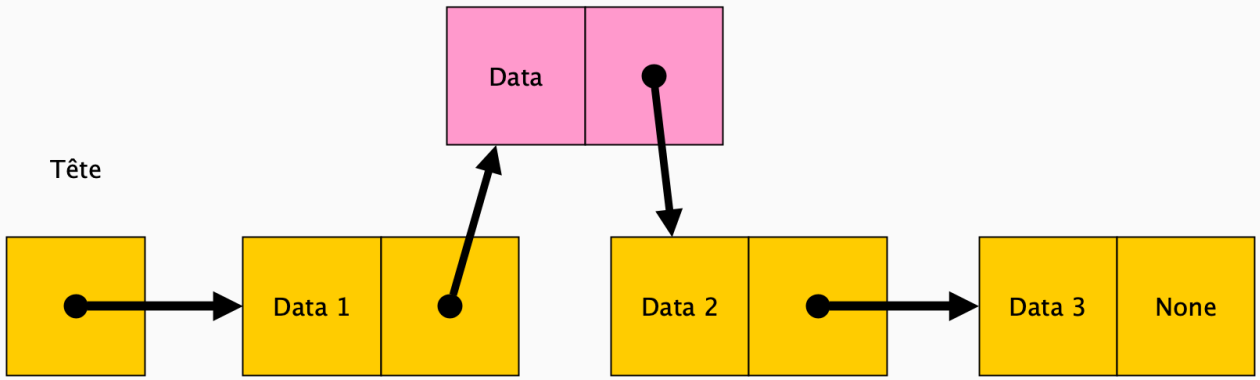
On veut insérer un élément.

Inserer



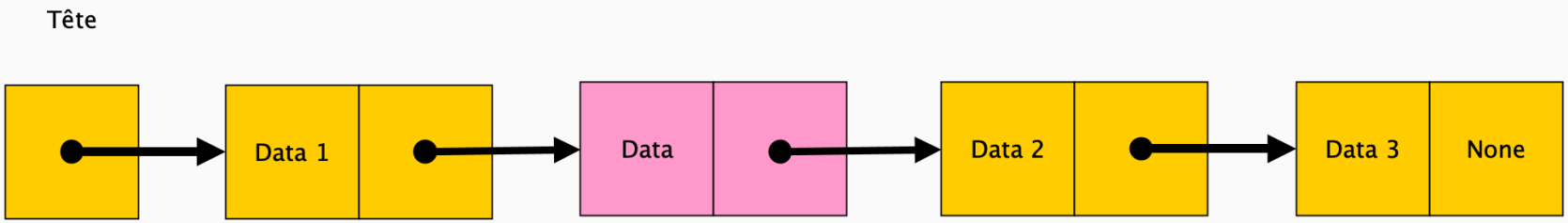
On le connecte à son (éventuel) futur successeur.

Inserer



On le connecte à son futur prédécesseur (ou tête).

Inserer



Et voilà !

Quelques différences `list` / liste chaînée

- **temps de suppression du premier élément :**
  - Très court pour une liste chaînée
  - Proportionnel à la taille de la liste Python
- **temps de suppression du dernier élément :** c'est le contraire
- **temps d'accès à la longueur :** immédiat pour la liste Python et proportionnel à la longueur de la liste chaînée
- **temps d'accès à un élément :** immédiat pour la liste Python et proportionnel à la place de l'élément pour la liste.