

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

- L'**implémentation** de la structure de données est la façon dont elle est représentée et codée en mémoire et n'est pas forcément accessible à l'utilisateur.

Définition : structure de données

- En informatique, une **structure de données** est une façon d'organiser, de gérer et de stocker des données permettant d'accéder et de modifier ces données de façon efficace.

Les tableaux fixes du C sont un exemple de structure de données.

- L'**interface** de la structure de données est l'ensemble des opérations accessibles à un utilisateur de la structure de données.

Par exemple, la notation `[]` permet d'accéder à un élément du tableau, pour le lire ou le modifier. Par contre, la taille du tableau ne fait pas partie de l'interface.

- L'**implémentation** de la structure de données est la façon dont elle est représentée et codée en mémoire et n'est pas forcément accessible à l'utilisateur.

On peut utiliser les listes de OCaml via leur interface sans savoir comment elles sont représentées en mémoire par le langage.

Caractérisation par l'interface

- La différence entre **interface** et **implémentation** est fondamentale et doit être bien comprise. En effet une même structure de données peut avoir plusieurs implémentations. C'est comme par exemple un téléviseur, on retrouve toujours la même interface (la télécommande) mais l'implémentation peut être différente (tube cathodique, écran plasma, lcd, led, ...).

Caractérisation par l'interface

- La différence entre **interface** et **implémentation** est fondamentale et doit être bien comprise. En effet une même structure de données peut avoir plusieurs implémentations. C'est comme par exemple un téléviseur, on retrouve toujours la même interface (la télécommande) mais l'implémentation peut être différente (tube cathodique, écran plasma, lcd, led, ...).
- Lorsqu'on définit un *cahier des charges* pour une structure de données (ensemble des données et opérations possibles), on définit ce qu'on appelle un **type abstrait de données**. Ainsi, une structure de données peut être vue comme une implémentation d'un type abstrait de données.

Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

elt4
elt3
elt2
elt1

Piles

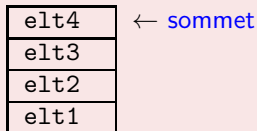
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.

elt4
elt3
elt2
elt1

- L'élément situé en haut de la pile s'appelle le **sommet**.

Piles

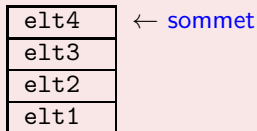
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.

Piles

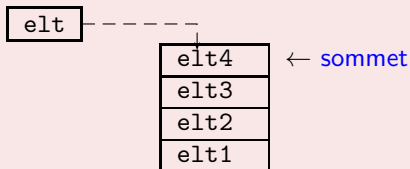
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile

Piles

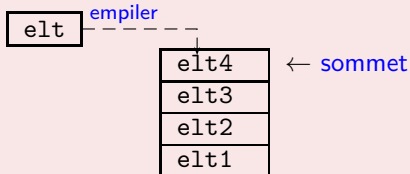
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile

Piles

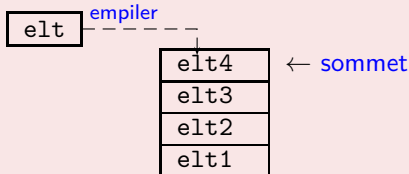
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile

Piles

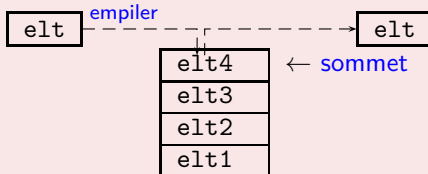
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

Piles

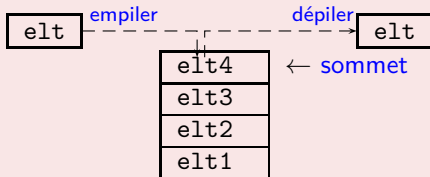
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

Piles

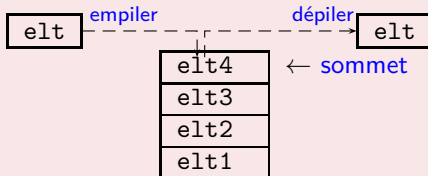
- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile

Piles

- Au niveau sémantique, une **pile** est semblable à une pile d'objet dans la vie de tous les jours.



- L'élément situé en haut de la pile s'appelle le **sommet**.
- Empiler signifie ajouter un élément au sommet de la pile
- Dépiler signifie retirer l'élément situé au sommet de la pile
- Ainsi le premier élément entré dans la pile sera aussi le dernier à en sortir, on dit qu'une pile est une structure **LIFO** *Last In First Out*

Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.

Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` qui ajoute un élément au sommet de la pile.

Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` qui ajoute un élément au sommet de la pile.
- `depiler()` qui retire l'élément situé au sommet (cela n'est possible que si la pile n'est pas vide).

Piles comme structures de données

L'interface d'une structure de données Pile se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la pile est vide ou non.
- `empiler()` qui ajoute un élément au sommet de la pile.
- `depiler()` qui retire l'élément situé au sommet (cela n'est possible que si la pile n'est pas vide).

Utilisation

En dépit de sa simplicité, cette structure de données a de nombreuses applications en informatique : pile d'appel récursif, pile d'évaluation d'une expression, ...

Manipulation de piles

Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?

Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :


Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
 - ① Appuie sur la touche "o"


Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
 - ➊ Appuie sur la touche "o"
 - ➋ Appuie sur la touche "i"

Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
 - Appuie sur la touche "o"
 - Appuie sur la touche "i"
 - Appuie sur la touche  (*backspace*)

Manipulation de piles

- On considère la pile : $P = | "A", "L", "I", "X" >$ (le sommet est "X"). Quelle suite d'opération permet d'obtenir $P = | "A", "L", "E", "X" >$?
- Un programmeur décide d'utiliser une pile afin de stocker une réponse entrée au clavier. Chaque caractère tapé doit être empiler. Traduire en terme d'opérations sur cette pile les actions suivantes :
 - Appuie sur la touche "o"
 - Appuie sur la touche "i"
 - Appuie sur la touche  (*backspace*)
 - Appuie sur la touche "k"

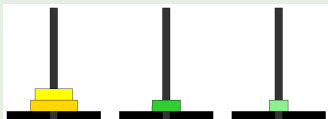
Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

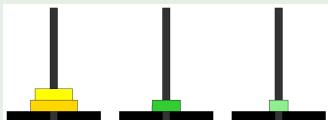
- 1 Quel est le contenu de chacune des piles dans la situation ci-dessous? (un disque est représenté dans la pile par sa taille)



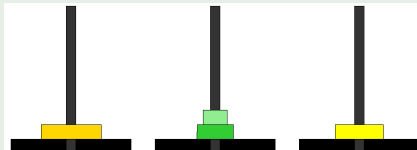
Manipulation de piles

Au jeu des tours des Hanoï, on gère les trois tours T1, T2 et T3 à l'aide de trois piles.

- 1 Quel est le contenu de chacune des piles dans la situation ci-dessous ? (un disque est représenté dans la pile par sa taille)



- 2 Ecrire les opérations permettant de passer la situation précédente à celle ci-dessous :



Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.

Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



Files

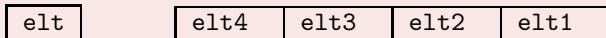
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

Files

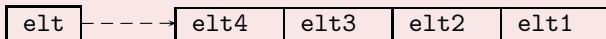
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

Files

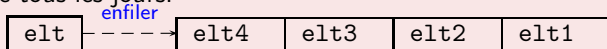
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

Files

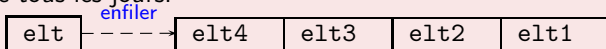
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file

Files

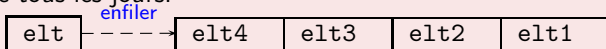
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

Files

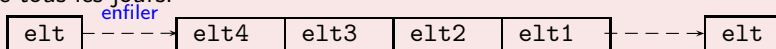
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

Files

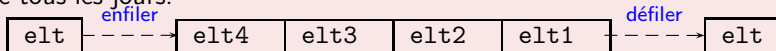
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

Files

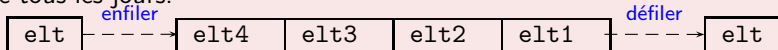
- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.

Files

- Au niveau sémantique, une **file** est semblable à une file d'attente dans la vie de tous les jours.



- Enfiler signifie ajouter un élément en fin de file
- Défiler signifie retirer l'élément situé au début de la file.
- Ainsi le premier élément entré dans la file sera aussi le premier à en sortir, on dit qu'une file est une structure **FIFO** *First In First Out*

Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.

Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.

Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.
- `defiler()` qui retire l'élément situé au début de la file (cela n'est possible que si la file n'est pas vide).

Files comme structures de données

L'interface d'une structure de données File se limite donc aux opérations suivantes :

- `est_vide()` qui renvoie un booléen indiquant si la file est vide ou non.
- `enfiler(element)` qui ajoute un élément à la fin de la file.
- `defiler()` qui retire l'élément situé au début de la file (cela n'est possible que si la file n'est pas vide).

Utilisation

Comme pour les piles, cette structure de données a de nombreuses applications en informatique : file d'attente d'une imprimante, simulation de files d'attentes réelles, ...

Manipulation de files

Manipulation de files

- On considère la file : $F = \langle "E", "L", "S", "A" \rangle$. Quelle suite d'opération permet d'obtenir $F = \langle "N", "O", "E", "L" \rangle$?

Manipulation de files

- On considère la file : $F = \langle "E", "L", "S", "A" \rangle$. Quelle suite d'opération permet d'obtenir $F = \langle "N", "O", "E", "L" \rangle$?
- On simule la file d'attente d'une imprimante à l'aide d'une file. A quelle opération sur cette file correspond l'envoi d'une nouvelle impression ? La fin de l'impression en cours ? Dans quel ordre seront effectuées les impressions ?

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).
 - Une opération renvoyant la **queue** de la liste, c'est à dire la liste privée de sa tête.

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).
 - Une opération renvoyant la **queue** de la liste, c'est à dire la liste privée de sa tête.
 - Une opération permettant d'accéder à un élément de la liste par son index.

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).
 - Une opération renvoyant la **queue** de la liste, c'est à dire la liste privée de sa tête.
 - Une opération permettant d'accéder à un élément de la liste par son index.
- Deux implémentations des listes sont généralement utilisées :

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).
 - Une opération renvoyant la **queue** de la liste, c'est à dire la liste privée de sa tête.
 - Une opération permettant d'accéder à un élément de la liste par son index.
- Deux implémentations des listes sont généralement utilisées :
 - Par un tableau (éléments contigus en mémoire)

Listes

- Au niveau sémantique, une **liste** est suite finie d'éléments (avec répétition éventuelle) on peut noter une liste : e_0, e_1, \dots, e_n .
- L'interface d'une liste contient généralement les éléments suivants :
 - Un constructeur permettant de créer la liste vide.
 - Une opération renvoyant un booléen suivant que la liste soit vide ou non.
 - Une opération permettant d'ajouter un élément en début (ou en fin) de liste.
 - Une opération renvoyant le premier élément de la liste (on dit la **tête** de la liste).
 - Une opération renvoyant la **queue** de la liste, c'est à dire la liste privée de sa tête.
 - Une opération permettant d'accéder à un élément de la liste par son index.
- Deux implémentations des listes sont généralement utilisées :
 - Par un tableau (éléments contigus en mémoire)
 - Par une liste chaînée (chaque élément contient alors un lien vers le suivant)



Ambiguïté

- Le terme `liste` devient ambiguë, en effet c'est à la fois le nom du type abstrait de données que nous venons de définir mais aussi celui d'une structure de données existante en Python.



Ambiguïté

- Le terme `liste` devient ambiguë, en effet c'est à la fois le nom du type abstrait de données que nous venons de définir mais aussi celui d'une structure de données existante en Python.
- D'autres part, pour amplifier encore cette ambiguïté, le terme *liste chaînée*, est parfois abrégé en `liste`, alors qu'il s'agit d'une implémentation possible et pas du type abstrait de données défini plus haut.

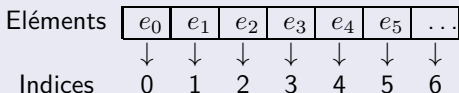
Python et le TAD liste

- En Python, le TAD liste est implémenté sous forme de tableau, c'est à dire que les éléments d'une liste de Python sont **contigus** en mémoire :

Eléments	e_0	e_1	e_2	e_3	e_4	e_5	\dots
	↓	↓	↓	↓	↓	↓	↓
Indices	0	1	2	3	4	5	6

Python et le TAD liste

- En Python, le TAD liste est implémenté sous forme de tableau, c'est à dire que les éléments d'une liste de Python sont **contigus** en mémoire :



- L'autre implémentation que nous avons évoquée (celle par les listes chaînées) peut être importé à partir du module `collections` ou écrite par exemple à l'aide de la POO. Dans ce cas, les éléments d'une liste ne sont plus contigus en mémoire mais chaque élément doit aussi contenir un lien vers son suivant dans la liste :



Avantages et inconvénients de ces implémentations

Ces deux implémentations ont des conséquences importantes en terme de complexité des opérations mais aussi d'occupation mémoire :

- Sous forme de liste chaînée, l'occupation mémoire est plus importante (on stocke en plus de l'élément le lien vers l'élément suivant).

Avantages et inconvénients de ces implémentations

Ces deux implémentations ont des conséquences importantes en terme de complexité des opérations mais aussi d'occupation mémoire :

- Sous forme de liste chaînée, l'occupation mémoire est plus importante (on stocke en plus de l'élément le lien vers l'élément suivant).
- Accéder à un élément sous forme de tableau est direct alors que pour une liste chaînée il faut avoir parcouru au préalable tout ceux qui le précèdent depuis la tête.

Avantages et inconvénients de ces implémentations

Ces deux implémentations ont des conséquences importantes en terme de complexité des opérations mais aussi d'occupation mémoire :

- Sous forme de liste chaînée, l'occupation mémoire est plus importante (on stocke en plus de l'élément le lien vers l'élément suivant).
- Accéder à un élément sous forme de tableau est direct alors que pour une liste chaînée il faut avoir parcouru au préalable tout ceux qui le précèdent depuis la tête.
- L'insertion d'un élément dans un tableau oblige à décaler ceux qui le suivent alors que l'opération est directe en liste chaînée.

Avantages et inconvénients de ces implémentations

Ces deux implémentations ont des conséquences importantes en terme de complexité des opérations mais aussi d'occupation mémoire :

- Sous forme de liste chaînée, l'occupation mémoire est plus importante (on stocke en plus de l'élément le lien vers l'élément suivant).
- Accéder à un élément sous forme de tableau est direct alors que pour une liste chaînée il faut avoir parcouru au préalable tout ceux qui le précèdent depuis la tête.
- L'insertion d'un élément dans un tableau oblige à décaler ceux qui le suivent alors que l'opération est directe en liste chaînée.
- L'ajout d'un élément en fin de liste oblige à agrandir le tableau si la taille initialement prévue est dépassée alors qu'avec une liste cela ne pose pas de problèmes.

Choix d'une structure de données

En conclusion : on utilisera

Choix d'une structure de données

En conclusion : on utilisera

- Un tableau (et donc une liste de Python) lorsque la taille des données est connu par avance et/ou lorsqu'on a besoin d'accéder aux éléments par leur indice rapidement.

Choix d'une structure de données

En conclusion : on utilisera

- Un tableau (et donc une liste de Python) lorsque la taille des données est connu par avance et/ou lorsqu'on a besoin d'accéder aux éléments par leur indice rapidement.
- Une liste chaînée (module deque) pour les structures à taille variable et/ou lorsqu'on a besoin d'insérer ou de supprimer les éléments situés au début ou à la fin.