Choisir ses conteneurs

Fabrice Lembrez - PSI*

Lycée Pierre de Fermat

Plan

1. Stockage de données concrètes
   - Concept central : le tableau physique
   - Types de données concrets
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python

Types primitifs, types composites

Python est un langage "de haut niveau" : beaucoup de choses cachées sous le capot quand on manipule même les types de données les plus élémentaires.

Sur des langages "bas niveau", comme C, la situation est plus simple à décrire

Types primitifs, types composites

Python est un langage "de haut niveau" : beaucoup de choses cachées sous le capot quand on manipule même les types de données les plus élémentaires.

Sur des langages "bas niveau", comme C, la situation est plus simple à décrire

Les types primitifs sont les blocs de base : entiers, flottants, booléens, (en C : caractères)...

Les types composites associent ces blocs de base : le type essentiel est le tableau

Types primitifs, types composites

Python est un langage "de haut niveau" : beaucoup de choses cachées sous le capot quand on manipule même les types de données les plus élémentaires.

Sur des langages "bas niveau", comme C, la situation est plus simple à décrire

Les types primitifs sont les blocs de base : entiers, flottants, booléens, (en C : caractères)...

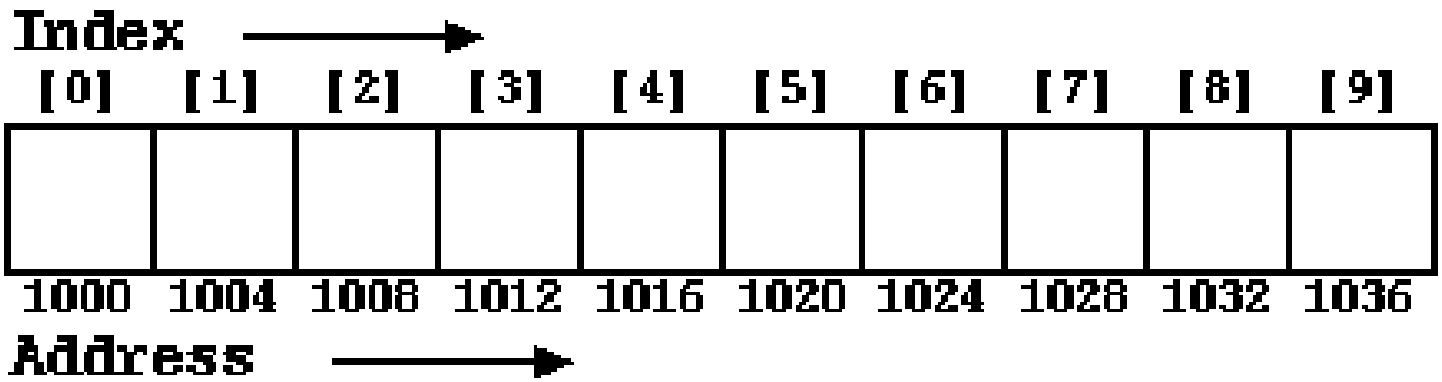
Les types composites associent ces blocs de base : le type essentiel est le tableau

On peut considérer qu'en décortiquant complètement la structure on se ramène à peu près toujours à des tableaux : c'est un concept universel.

Le tableau "physique"

Caractéristiques ordinaires d'un tableau

- formé de **données du même type,**
- mémorisation dans des emplacements contigus
- avantage : accès aux données très rapide
- contrainte : **taille définie à l'avance** pour réserver le bloc mémoire



Accès très rapide à la donnée A[i] à partir de la valeur de l'index i et de la taille des données

Plan

1. Stockage de données concrètes
   - Concept central : le tableau physique
   - Types de données concrets
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python

Types de données concrets

Déf - Type de données concret (dans un langage)

Il définit précisément la façon dont les donnés sont implantées en machine et les opérations disponibles. Exemples : le type "booléen", le type "list" en Python.

Très variable selon les langages, on s'y intéresse a minima pour éviter les pièges syntaxiques.

Ne pas trop regarder sous le capot

Chaque langage offre des types de données concrets et une syntaxe spécifiques.

En dernière analyse, les données sont pourtant stockées sous forme de tableau. Une implémentation intelligente permet de contourner les contraintes de ces tableaux. On verra comment les choses s'organisent concrètement dans deux cas

- pour les "listes Python" (cf cours de Sup, et on reviendra dessus)
- pour les dictionnaires (cf plus tard)

Mais essentiellement on va faire tout autre chose : de l'algorithmique abstraite, avec des types de données abstraits.

Plan

1. Stockage de données concrètes
2. Types de données abstraits
   - Modèles de calcul de la complexité
   - Types et structures de données
   - Exemple - la pile
3. Structures linéaires
4. Le type "list" de Python

Modèles abstraits de calcul de la complexité

Les outils de calcul

quel type de données "élémentaires"

quel type d'"opérations élémentaires" (de coût 1)

Exemple - chercher le maximum dans une liste d'entiers :

données de base : les entiers

opérations de base = lecture de case, comparaison d'entiers.

Modèles abstraits de calcul de la complexité

Les outils de calcul

quel type de données "élémentaires"

quel type d'"opérations élémentaires" (de coût 1)

Exemple - chercher le maximum dans une liste d'entiers :

données de base : les entiers

opérations de base = lecture de case, comparaison d'entiers.

Remarque : on considère que les opérations s'effectuent les unes après les autres (pas de parallélisme)

Plan

1. Stockage de données concrètes
2. Types de données abstraits
   - Modèles de calcul de la complexité
   - Types et structures de données
   - Exemple - la pile
3. Structures linéaires
4. Le type "list" de Python

Types et structures de données

Déf - Type ou structure de données (abstrait)

Modèle mathématique décrit par des "opérations élémentaires" et leurs propriétés. Il permet des preuves formelles et est indépendant de l'implantation machine.

Exemples : pile, file, arbre, tas, file de priorité...

Avec un langage donné, on se demandera comment implémenter concrètement tel ou tel type abstrait.

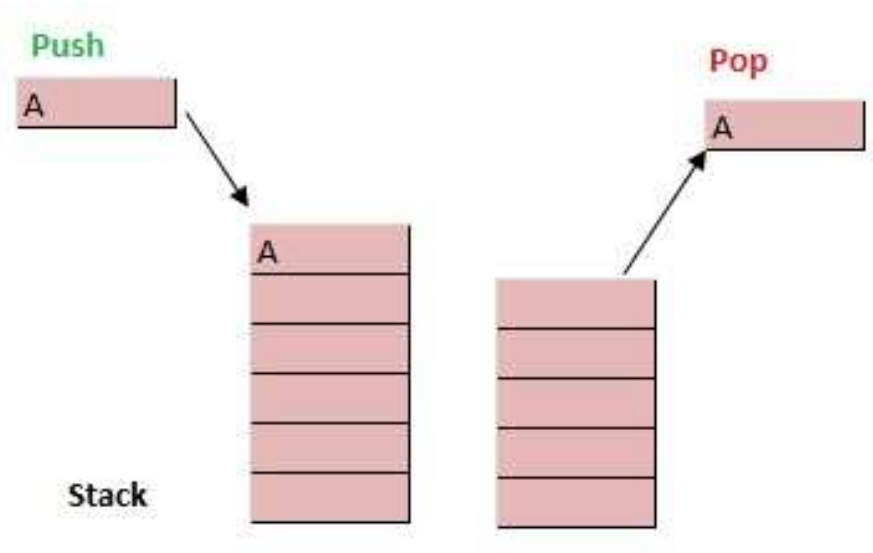
Plan

1. Stockage de données concrètes
2. Types de données abstraits
   - Modèles de calcul de la complexité
   - Types et structures de données
   - Exemple - la pile
3. Structures linéaires
4. Le type "list" de Python

La pile, type abstrait

Déf - Pile - description "informelle"

Une pile est une collection ordonnée de données auxquelles on accède exclusivement selon le principe LIFO (last in, first out). Les deux opérations de base sont donc
- l'ajout d'une nouvelle donnée à la pile (empilement, "push")
- le retrait de la dernière donnée ajoutée (dépilement, "pop")



Propriétés et opérations fondamentales

empiler : push(x,P) empile la donnée x sur la pile P

depiler : pop(P) dépile la dernière valeur et la renvoie
on la récupère par x$\leftarrow$ pop(P)

pileVide : la pile qui ne contient aucun élément

estVide : estVide(P) teste si la pile est la pile vide
(le résultat est un booléen)

Caractérisation des piles

A quoi ressemblerait une véritable description formelle des piles : les piles sont "tout ce qu'on peut construire à partir de pileVide et des opérations précédentes", avec les contraintes suivantes :

- estVide(pileVide)=True
- pour tout x, estVide(push(x,pileVide))=False
- faire pop(push(x,P)) redonne la pile P et renvoie x

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
   - Structures linéaires
   - Tableaux au sens strict
   - Tableaux redimensionnables (ou "vecteurs", "dynamic arrays")
   - Variantes : pile, file, dèque
   - Dictionnaires
4. Le type "list" de Python

Structures de données linéaires

Données organisées de façon séquentielle, permettant un parcours simple. Selon les opérations de base, cela se décline en

- tableaux
- tableaux redimensionnables
- piles
- files
- listes chaînées (hors programme pour nous),
- évent. dictionnaires, ...

Principe d'utilisation des structures qui suivent

Respectez le choix s'il est imposé. Si le choix de structure vous revient, prenez la plus simple adaptée aux besoins.

Question de base : **quelles opérations élémentaires vais-je employer ?**

Plan

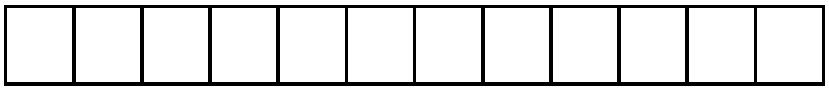
1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
   - Structures linéaires
   - Tableaux au sens strict
   - Tableaux redimensionnables (ou "vecteurs", "dynamic arrays")
   - Variantes : pile, file, dèque
   - Dictionnaires
4. Le type "list" de Python

Tableaux (abstraits)

Déf - Tableau

Un tableau est un type de données linéaire ayant une taille fixée N : on accède à son contenu $a_0, \ldots, a_{N-1}$ avec la syntaxe `a[i]`. Les opérations élémentaires sont uniquement : **lire ou affecter une case**.

Il correspond au tableau "physique" vu au-dessus



Efficace, il existe dans la plupart des langages. Les listes Python sont bien plus que de "simples tableaux", mais il y a des tableaux (ndarray) dans le module numpy.

Rien n'empêche ensuite de faire des tableaux de tableaux, ou tableaux à deux indices : A[i][j]=A[i,j], ou plus.

Opérations élémentaires ou composées

Elémentaire : lire/affacter

Non élémentaire : tout ce qui demande parcours et/ou recopie

Même si elles sont pré-implantées dans un langage de haut niveau tel que Python, ces opérations ont un coût de l'ordre de N, voire plus :

- Enlever un élément en position quelconque (et "boucher le trou")
- Insérer un élément en position quelconque (et "décaler")
- Sélectionner concrètement une "sous liste" (ie la copier en mémoire)
- Donner la première occurrence d'une certaine valeur
- Trier les éléments pour une certaine relation d'ordre

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
   - Structures linéaires
   - Tableaux au sens strict
   - Tableaux redimensionnables (ou "vecteurs", "dynamic arrays")
   - Variantes : pile, file, dèque
   - Dictionnaires
4. Le type "list" de Python

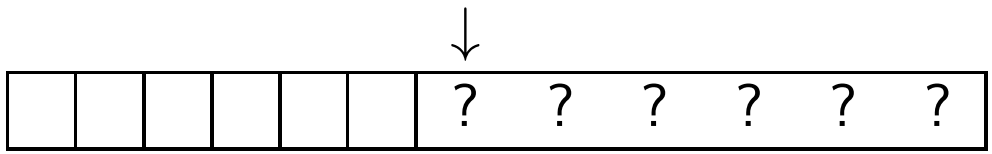
Tableaux redimensionnables

Déf - Tableau redimensionnable (ou "vecteur")

Un tableau redimensionnable est un type de données analogue à un tableau mais pour lequel on ajoute deux opérations supplémentaires : **ajouter ou supprimer une case en fin de tableau**.

Ici append() et pop() sont élemntaires mais pas pop(i), qui est à éviter.

Pourquoi c'est pertinent : c'est une adaptation et un assouplissement du tableau, dans lequel on réserve un espace mémoire un peu trop grand. Il y a une technique intelligente pour gérer les débordements de la zone réservée : ils se produisent mais pas trop souvent.



Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
   - Structures linéaires
   - Tableaux au sens strict
   - Tableaux redimensionnables (ou "vecteurs", "dynamic arrays")
   - Variantes : pile, file, dèque
   - Dictionnaires
4. Le type "list" de Python

Variantes : pile, file, dèque

Situation fréquente :
- lecture directe dans la case i inutile,
- accès aux données en **lecture/écriture uniquement "par les bouts"**

pile - op. de base = ajoute/lit/enlève un élément en fin de pile,

file - op. de base = ajouter à la fin, lire/enlever le début de la file

dèque - op. de base = ajouter/enlever par les deux bouts

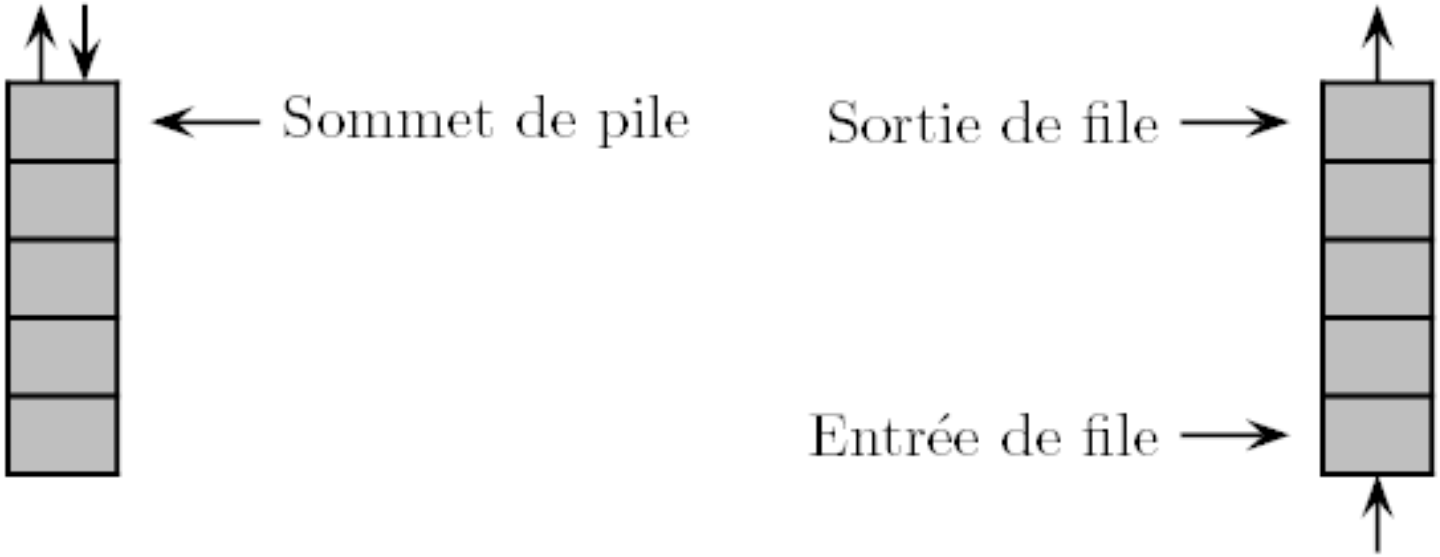
Variantes : pile, file, dèque

Situation fréquente :
- lecture directe dans la case i inutile,
- accès aux données en **lecture/écriture uniquement "par les bouts"**

pile - op. de base = ajoute/lit/enlève un élément en fin de pile,

file - op. de base = ajouter à la fin, lire/enlever le début de la file

dèque - op. de base = ajouter/enlever par les deux bouts



Variantes : pile, file, dèque

Situation fréquente :
- lecture directe dans la case i inutile,
- accès aux données en **lecture/écriture uniquement "par les bouts"**

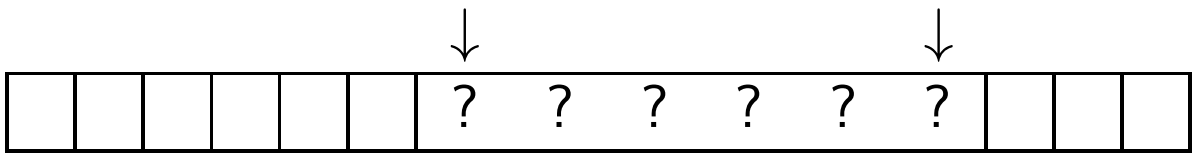
pile - op. de base = ajoute/lit/enlève un élément en fin de pile,

file - op. de base = ajouter à la fin, lire/enlever le début de la file

dèque - op. de base = ajouter/enlever par les deux bouts

Ces structures ne sont pas prédéfinies en Python. Pour la pile on peut reprendre un tableau redimensionnable dont on "bride" l'utilisation. Pour une file ou plus généralement un dèque : il y a `collections.deque`.

Pour donner une idée de stockage physique concret (plus recopies si débordement)



Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
   - Structures linéaires
   - Tableaux au sens strict
   - Tableaux redimensionnables (ou "vecteurs", "dynamic arrays")
   - Variantes : pile, file, dèque
   - Dictionnaires
4. Le type "list" de Python

Dictionnaires (pour mémoire)

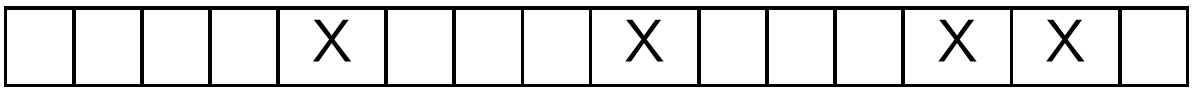
Déf - Dictionnaire (ou table d'association)

Application de la forme clef $\rightarrow$ valeur sur un certain ensemble de clefs. Les opérations élémentaires sont **lire ou modifier la valeur associée à une clef, et aussi rechercher l'existence d'une clef, ajouter une nouvelle association clef $\rightarrow$ valeur**.

Pourquoi c'est pertinent : de façon sous-jacente il y a un stockage sous forme de tableau, de nouveau dans un espace trop grand. Il y a une gestion intelligente des adresses dans le tableau (fonction de hachage) et des débordements éventuels.

Clefs x,x,x,x

$\downarrow$ hachage $\downarrow$



Une différence importante : plus d'ordre de parcours naturel.

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python
   - Fonctionnement général
   - Rappel sur l'indexation
   - Mutable vs non mutable (POO)
   - Affectation vs modification
   - En algorithmique, se limiter aux tableaux (redimensionnables)

Commment fonctionne le type list

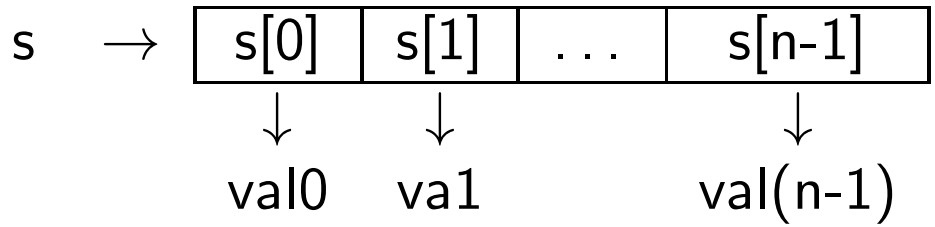
Problème de vocabulaire : "liste" est un "type abstrait", qui peut s'implémenter par exemple avec la structure de "liste chaînée".
Ici je parle d'autre chose : le type (concret) "list" du langage Python.

Commment fonctionne le type list

Problème de vocabulaire : "liste" est un "type abstrait", qui peut s'implémenter par exemple avec la structure de "liste chaînée".
Ici je parle d'autre chose : le type (concret) "list" du langage Python.

Déf - Une liste Python, c'est essentiellement :

- un tableau "redimensionnable", élargi en fonction des besoins
- où on ne stocke pas des valeurs, mais des "références" qui permettent d'accéder à l'endroit où sont ces valeurs



Les options pour la création de listes

- Création directe d'une petite liste : s=[3,5,18,34]
- Création progressive : initialisation avec s=[ ] puis une succession de s.append(valeur)
- Création directe par compréhension s=[i**2 for i in range(5)]

Remarque : attention range ne donne pas directement une liste : c'est un itérateur qu'il faut convertir : s=list(range(5))

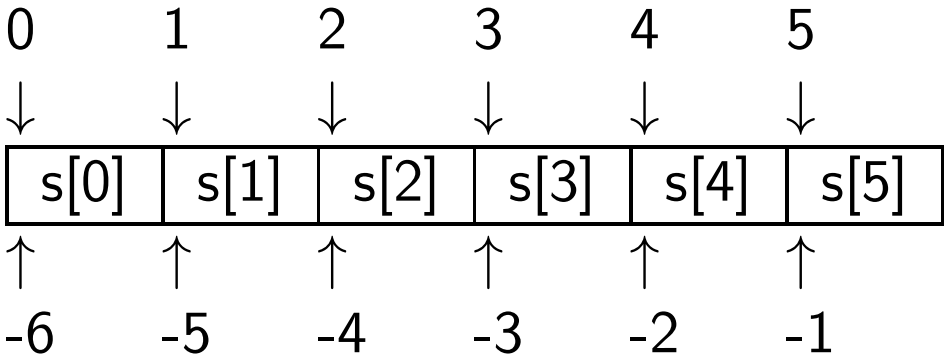
Pas de création par concaténations (+) successives. Utiliser la méthode append

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python
   - Fonctionnement général
   - Rappel sur l'indexation
   - Mutable vs non mutable (POO)
   - Affectation vs modification
   - En algorithmique, se limiter aux tableaux (redimensionnables)

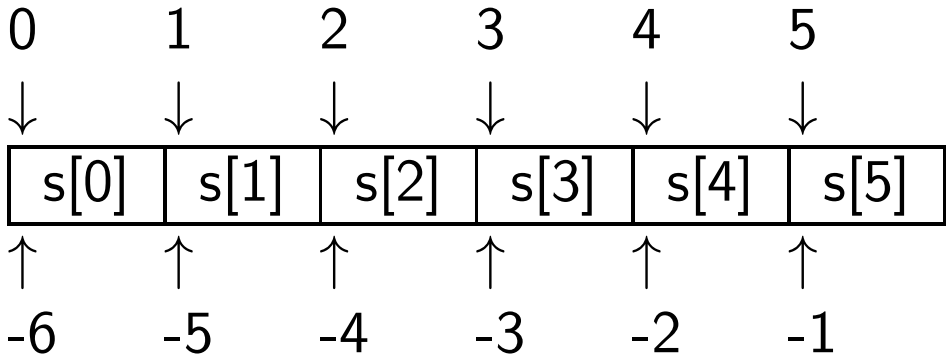
Indexation des listes et tableaux numpy

Exemple avec une liste s de longueur 6



Indexation des listes et tableaux numpy

Exemple avec une liste s de longueur 6

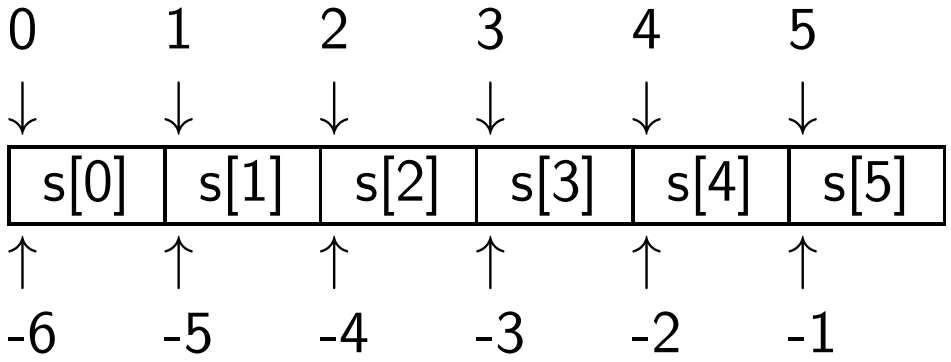


Ce qui permet de comprendre le "slicing" pour les listes Python et les tableaux numpy

- s[:3] : cases numéro 0, 1 et 2
- s[1:3] : cases numéro 1 et 2
- s[3:-1] : cases numéro 3 et 4
- s[-2:] : les deux dernières cases

Indexation des listes et tableaux numpy

Exemple avec une liste s de longueur 6



Ce qui permet de comprendre le "slicing" pour les listes Python et les tableaux numpy

- ▷ s[:3] : cases numéro 0, 1 et 2
- ▷ s[1:3] : cases numéro 1 et 2
- ▷ s[3:-1] : cases numéro 3 et 4
- ▷ s[-2:] : les deux dernières cases

Pour mémoire : on peut aussi parcourir de deux en deux, par exemple s[::2] ou à l'envers : s[::-1].

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python
   - Fonctionnement général
   - Rappel sur l'indexation
   - Mutable vs non mutable (POO)
   - Affectation vs modification
   - En algorithmique, se limiter aux tableaux (redimensionnables)

Objets mutables/non mutables

- Une liste s est mutable : on peut modifier le contenu s[1]=3 sans créer une nouvelle liste
- Une chaîne de caractères n'est pas mutable
- Tuple $t = (1, 2, 3)$: ressemble à la liste, mais non mutable

Objets mutables/non mutables

- Une liste s est mutable : on peut modifier le contenu s[1]=3 sans créer une nouvelle liste
- Une chaîne de caractères n'est pas mutable
- Tuple $t = (1, 2, 3)$: ressemble à la liste, mais non mutable

Objets mutables, notamment passés en paramètres

Donner la référence d'un objet mutable c'est permettre de modifier le contenu.
Ainsi, quand on fait un appel de fonction f(s) avec une liste s, on passe non pas "le contenu de s" mais "une référence" pointant sur le contenu. Et on donne la possibilité de lire et de modifier le contenu de façon pérenne : s[2]=3, s.append(18)...

Exemple de fausses manœuvres

Que donnera le code suivant :

```
s=[[0,0]]*3 # liste avec 3 fois le point origine
s[0][1]=1
```

Exemple de fausses manœuvres

Que donnera le code suivant :

```
s=[[0,0]]*3 # liste avec 3 fois le point origine
s[0][1]=1
```

s vaudra [[0,1],[0,1],[0,1]] ! On a passé 3 fois la même référence...

Exemple de fausses manœuvres

Que donnera le code suivant :

```
s=[[0,0]]*3 # liste avec 3 fois le point origine
s[0][1]=1
```

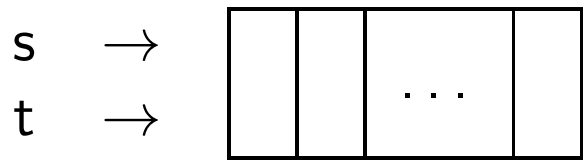
s vaudra [[0,1],[0,1],[0,1]] ! On a passé 3 fois la même référence...

Mais en créant s=[[0,0] for i in range(3)] ce ne serait pas pareil !

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python
   - Fonctionnement général
   - Rappel sur l'indexation
   - Mutable vs non mutable (POO)
   - Affectation vs modification
   - En algorithmique, se limiter aux tableaux (redimensionnables)

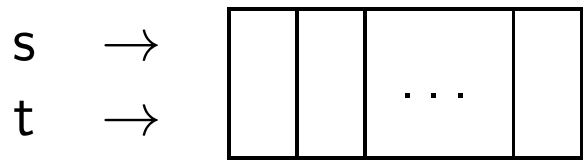
Affectations et modifications



Affecter ce n'est pas pareil que modifier le contenu des cases !

Si s est une liste, l'affectation t=s ne crée pas de copie de s mais une autre référence (fléchant sur la même adresse).
On le voit si on affecte s[1]=5 puis on fait print(t).

Affectations et modifications



Affecter ce n'est pas pareil que modifier le contenu des cases !

Si s est une liste, l'affectation t=s ne crée pas de copie de s mais une autre référence (fléchant sur la même adresse).
On le voit si on affecte s[1]=5 puis on fait print(t).

Utiliser le module copy pour les copies de listes

- utiliser sa fonction copy (crée une copie superficielle)
- et même pour des listes de listes : la fonction deepcopy
- autre solution : t=s[:] (seulement superficiel !)

NB : pour les tableaux numpy c'est pareil, on a une référence vers l'ensemble du tableau. Utiliser t=np.copy(s)

Plan

1. Stockage de données concrètes
2. Types de données abstraits
3. Structures linéaires
4. Le type "list" de Python
   - Fonctionnement général
   - Rappel sur l'indexation
   - Mutable vs non mutable (POO)
   - Affectation vs modification
   - En algorithmique, se limiter aux tableaux (redimensionnables)

En algorithmique : se limiter aux tableaux

Quand on se pose des questions de complexité algorithmique, il faut éviter les Pythonneries et privilégier des solutions universelles. Concrètement, dans ce contexte (ex : sujets de l'X), on n'utilisera les listes presque que comme tableaux :

- privilégier absolument les affectations a[i]=...
- ne pas concaténer des listes [1,2,3]+[4,5]
- éviter même l'opérateur $*$ (notamment pour les listes de listes)
- ne pas supprimer d'élément, ne pas en intercaler
- à plus forte raison éviter les méthodes avancées remove, insert, sort, index : on les reprogrammera si besoin.

En revanche sur des sujets Centrale, voire Mines, il arrive que ce genre de manipulations soit explicitement autorisé.