Programmation 2

Par Arsene Fokam Poka,

INTRODUCTION

- Le cours de programmation 1 est en prerequis
  - Python 3
  - Types de donnees de base
  - Programmation oriente objets
  - Structures de contrôle
    - Structures conditionnelles
    - Structures itteratives
- Les structures de donnees au cœur de la programmation
  - Independament du langage de programmation

INTRODUCTION

- Les structures de données au cœur de la programmation
  - Présentation des structures de données
  - Comment implémenter les structures de données?
  - Quelles structures de données conviennent a un problème précis?
  - Quels algorithmes est plus appropries?

Chapitre I : Aperçu général

Par Arsene Fokam Poka,

Chapitre I : Aperçu général

1. Notions de structures de données et d'algorithmes?
2. Notions d'abstraction et de type de données abstraits
3. Listing des structures de données
4. Aperçue générale des algorithmes
5. POO

Chapitre 1 : Aperçu général

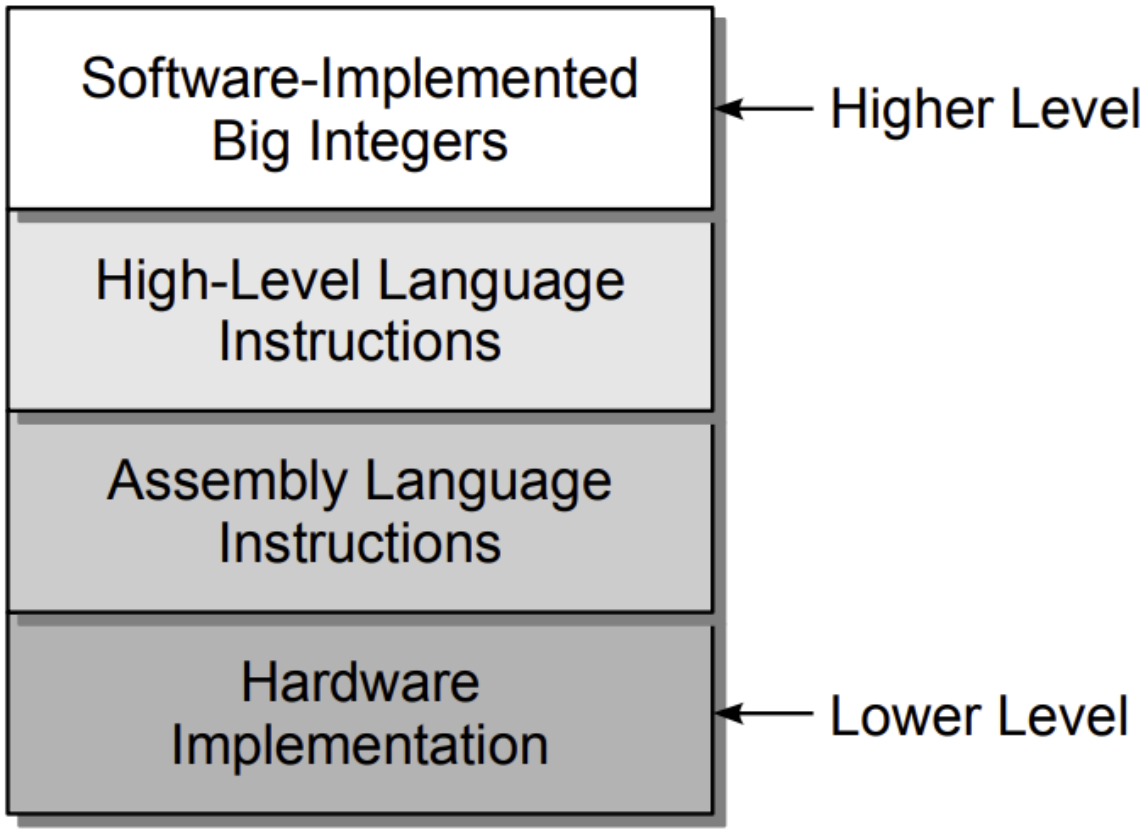
1- Notions de structures de données et d'algorithmes

- Une **structure de données** est une manière d'organiser physiquement et de stocker des données de manière qu'elles puissent être utilisées **efficacement**. Cela implique généralement de définir comment les données seront stockées, accessibles et manipulées..
- Un **algorithme** est une séquence d'instructions claires et précises, étapes par étapes, pour résoudre un problème
- Leur traduction selon un langage de programmation abouti à un programme
- La structure de données est importante non seulement pour la vitesse d'exécution, mais aussi pour la signification des informations ainsi qu'à leurs interactions
- Un algorithme et une structure de données évoluent bien s'ils fonctionnent aussi efficacement que possible à mesure que les données augmentent

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait

- Une **abstraction** est un mécanisme permettant de séparer les propriétés d'un objet et de restreindre l'attention à celles pertinentes dans le contexte actuel.
- Deux types courants d'abstractions rencontrés en informatique sont l'**abstraction procédurale ou fonctionnelle** et l'**abstraction de données.**
- L'**abstraction procédurale** est l'utilisation d'une fonction ou d'une méthode en sachant ce qu'elle fait mais en ignorant comment elle est accomplie**.**
- L'**abstraction de données** est la séparation des propriétés d'un type de données (ses valeurs et opérations) de l'implémentation de ce type de données.

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait

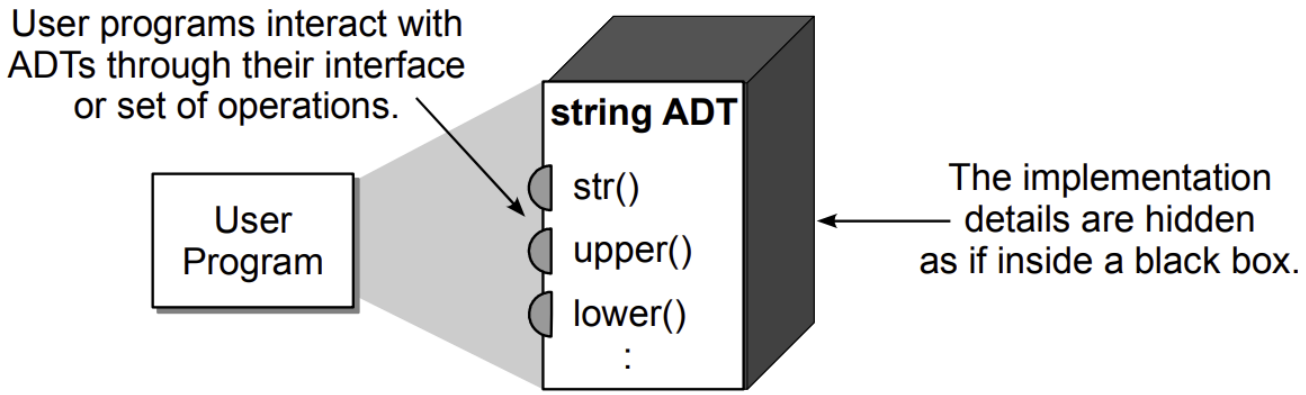


- En règle générale, les abstractions de problèmes complexes se produisent en couches, chaque couche supérieure ajoutant plus d'abstraction que la précédente.
- La capture suivante illustre les niveaux d'abstraction mises en œuvre dans l'implementation du type entier en programmation

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait

- Un **type de données abstrait** est un type de données défini par le programmeur qui spécifie un **ensemble de valeurs de données** et un **ensemble d'opérations bien définies** pouvant être effectuées sur ces valeurs.
- Ils sont définis indépendamment de leur implémentation, ce qui nous permet de nous concentrer sur leur utilisation.
- Cette séparation est généralement appliquée en exigeant une interaction avec le type de données abstrait via une interface ou un ensemble d'opérations défini.

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait



- Les types de données abstraits peuvent être visualisés comme des boîtes noires, comme illustré dans la figure
- Les programmes utilisateur interagissent avec les instances de l'ADT en appelant l'une des nombreuses opérations définies par son interface.

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait

- L'ensemble des opérations peut être regroupé en quatre catégories:
  i. **Constructeurs** : créent et initialisent de nouvelles instances de l'ADT.
  ii. **Accesseurs** : renvoient les données contenues dans une instance sans la modifier
  iii. **Mutateurs** : modifient le contenu d'une instance ADT.
  iv. **Itérateurs** : traitent les composants de données individuels de manière séquentielle

  La mise en œuvre des différentes opérations est cachée à l'intérieur de la boîte noire dont nous n'avons pas besoin de connaître le contenu pour utiliser l'ADT.

Chapitre 1 : Aperçu général
2- Les Abstractions et les types de données abstrait

- Les avantages à travailler avec des types de données abstraits .
  i. Se concentrer sur la résolution du problème actuel au lieu de nous enliser dans les détails de mise en œuvre.
  ii. Réduire les erreurs logiques pouvant survenir suite à une mauvaise utilisation accidentelle des structures de stockage et des types de données en empêchant l'accès direct à l'implémentation.
  iii. L'implémentation du type de données abstrait peut être modifiée sans avoir à modifier le code du programme qui utilise l'ADT.
  iv. Decouper des programmes plus importants en modules plus petits, permettant ainsi à différents membres d'une équipe de travailler sur des modules séparés.

Chapitre 1 : Aperçu général
3- Listing des structures de données

- Toutes les structures de données ne sont pas adaptées à tous les types de problèmes. Certaines conviennent mieux à certains problèmes.
- Par exemple, pour gérer un répertoire téléphonique, on doit :
  - Ajouter un nouveau contact
  - Supprimer un contact
  - Rechercher un contact
  - Lister les contacts
  - Modifier un contact
- L'ajout, la suppression, la recherche, le trie sont généralement les opérations qui seront à effectuer sur chacune des structures de données .

Chapitre 1 : Aperçu général
2- Listing des structures de données

Structures de données	Avantages	Inconvénients
Tableau	• Simple • Insertion rapide • Parcours rapide	• Recherche lente, • la taille doit être connue d'avance
Tableau trie	• Comme le tableau • Recherche plus rapide	• Comme le tableau
Pile	• Simple • Rapide pour les insertions/suppressions Last-In First-Out • Parcours rapide	• Recherches lentes pour les éléments autres que la dernière entrée

Chapitre 1 : Aperçu général
2- Listing des structures de données

Structures de données	Avantages	Inconvénients
File	• Simple • Rapide pour les insertions/suppressions First-In First-Out • Parcours rapide	• Recherches lentes pour les éléments autres que la dernière entrée ou la première sortie
Liste chaînée	• Légèrement complexe • Insertions/suppressions rapides pour les éléments a une position connue • Facile à étendre, rétrécir ou parcourir	• Recherche lente
Arbre binaire de recherche	• Légèrement complexe • Rapide pour les recherches, insertions • Facile à étendre, rétrécir ou parcourir	• Suppression complexe

Chapitre 1 : Aperçu général
2- Listing des structures de données

Structures de données	Avantages	Inconvénients
Table de hachage	• Rapide pour les insertions/suppressions First-In First-Out • Parcours rapide	• Complexe, peut occuper plus d'espaces que d'autres structures de donnees. • Légèrement plus lent à parcourir qu'un tableau ordinaire
Tas	• Légèrement complexe • les insertions/suppressions rapide pour les tas tries	• Recherche lente pour les items autres que le min et le max
	• Insertion rapide	• Recherche/suppression lente et complexe • Plus consommant en espace memoire

Chapitre 1 : Aperçu général
3- Types de données abstraits

- Toutes les structures de données ne sont pas adaptées à tous les types de problèmes. Certaines conviennent mieux à certains problèmes.
- Par exemple, pour gérer un répertoire téléphonique, on doit :
  - Ajouter un nouveau contact
  - Supprimer un contact
  - Rechercher un contact
  - Lister les contacts
  - Modifier un contact
- L'ajout, la suppression, la recherche, le trie sont généralement les opérations qui seront à effectuer sur chacune des structures de données .

Chapitre 2. Les tableaux et les listes

Par Arsene Fokam Poka,

Chapitre 2 : Les tableaux et les listes
2.1 Introduction

- Les tableaux sont les structures de données les plus basiques utilisees pour collecter et acceder aux donnees
- La plupart des langages de programmations mettent en œuvre un type de donnees tabulaires et permettent la creation de tableaux de dimension multiple.
- Dans ce chapitre nous allons implementer un tableau uni-dimensionnel et réutiliser cette structure pour pour implémenter un tableau pluri-dimensionel et une matrice

2.1 Les tableaux

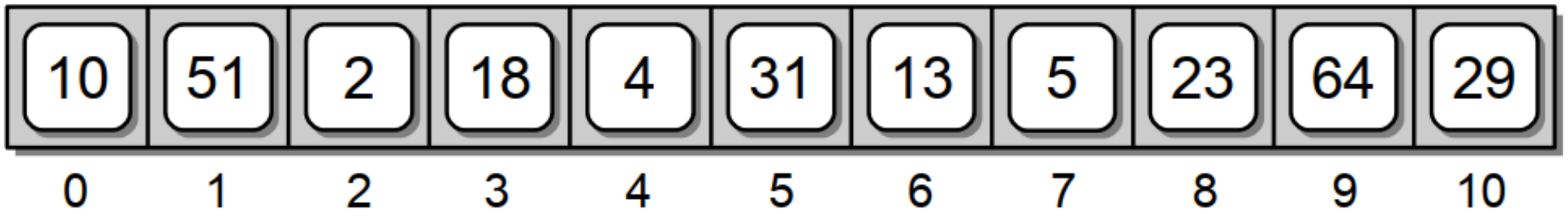


Figure 2.1: A sample 1-D array consisting of 11 elements.

2.1.1 Structure

- Il consiste en de multiples données séquentielles stockées de façon contigue en mémoire
- Les enregistrements sont directement accessibles individuellement au moyen d'un entier(index) précisant le décalage par rapport au premier élément du tableau
- La taille d'un tableau ne saurait être change (contrairement aux **List** en python). Ils sont donc utiles pour les problèmes dont nous connaissons d'avance la taille des éléments et qu'elle ne change pas.

2.1.2 Structure de données abstraite

- Le tableau se trouve dans la plupart des langages de programmation entant que type primitif, mais python ne fournit que la structure de List pour créer des séquences mutables
- Pour pouvons définir le type abstrait **Array** pour représenter un tableau unidimensionnel qui fonctionnera en python de la même manière que les tableaux d'autres langages.

2.1.2 Structure de données abstraite

De façon abstraite, on peut definir notre classe Array ainsi qu'il suit :

- ***Array(size)*** : Cree un tableau consistant en ***size*** éléments initialement ***None***. ***Size > 0***
- ***length()*** : retourne le nombre d'éléments dans le tableau
- ***getItem(index)*** : retourne la valeur stockée a la position ***index*** dans le tableau
- ***setItem(index, value)*** : modifie l'élément a la position ***index*** du tableau par la value
- ***clearing(value)*** : Efface le tableau en définissant chaque élément du tableau sur ***value***
- ***iterator()*** : cree et retourne un itérateur qui peut etre utilise pour parcourir le tableau élément apres élément.
- Le code source est l'objet du fichier *array.ipynb* dans accessible à ce lien : https://github.com/arseneRuben/INF1093

2.2 Les listes

- Les listes diffèrent fondamentalement des tableaux en ce qu'elles sont mutables, c’est-à-dire , à mesure que les éléments y sont ajoutés ou retires , leur taille évolue.
- Dans cette section, on examinera
  - l'implémentation des listes,
  - leurs différences majeures avec les tableaux
  - Les opérations communes sur les listes

2.2.1 Création

- L'instruction myList = [4, 12, 2, 34, 17]

  Implique l'appel du constructeur list()
  cree un objet myList initialise avec les valeurs mentionnees

  Ces valeurs sont incluses dans un tableau plus grand en taille que le nombre de veleurs reelles, reservant ainsi l'espace pour une future expension

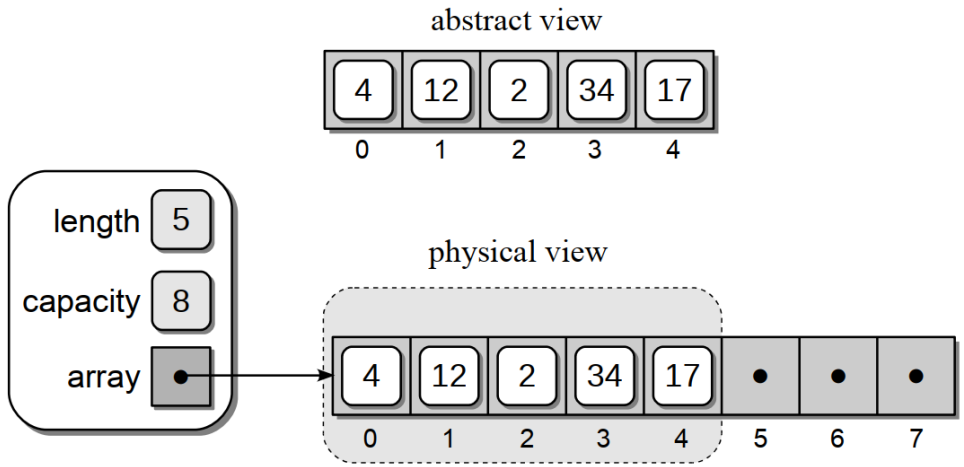


Figure 2.2: The abstract and physical views of a list implemented using an array.

2.2.2 Ajout d'item

- On peut ajouter un élément à la fin d'une liste
- *myList.append(50)*
- Si il y a de l'espace dans le tableau, le nouvel élément est ajoute au prochain slot disponible
- *myList.append(18)*

  myList.append(64)

  MyList.append(6)

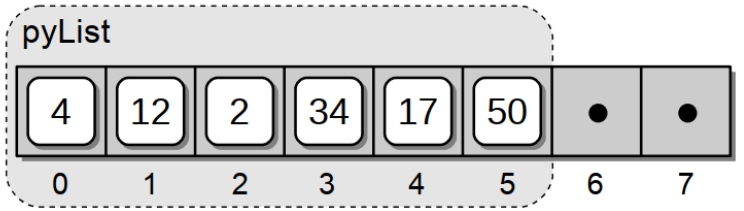


Figure 2.3: Result of appending value 50 to the list.

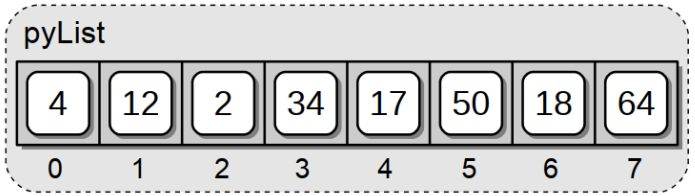


Figure 2.4: A full array resulting after appending three values.

Que se passe t-il lorsque la taille de la liste est atteinte?

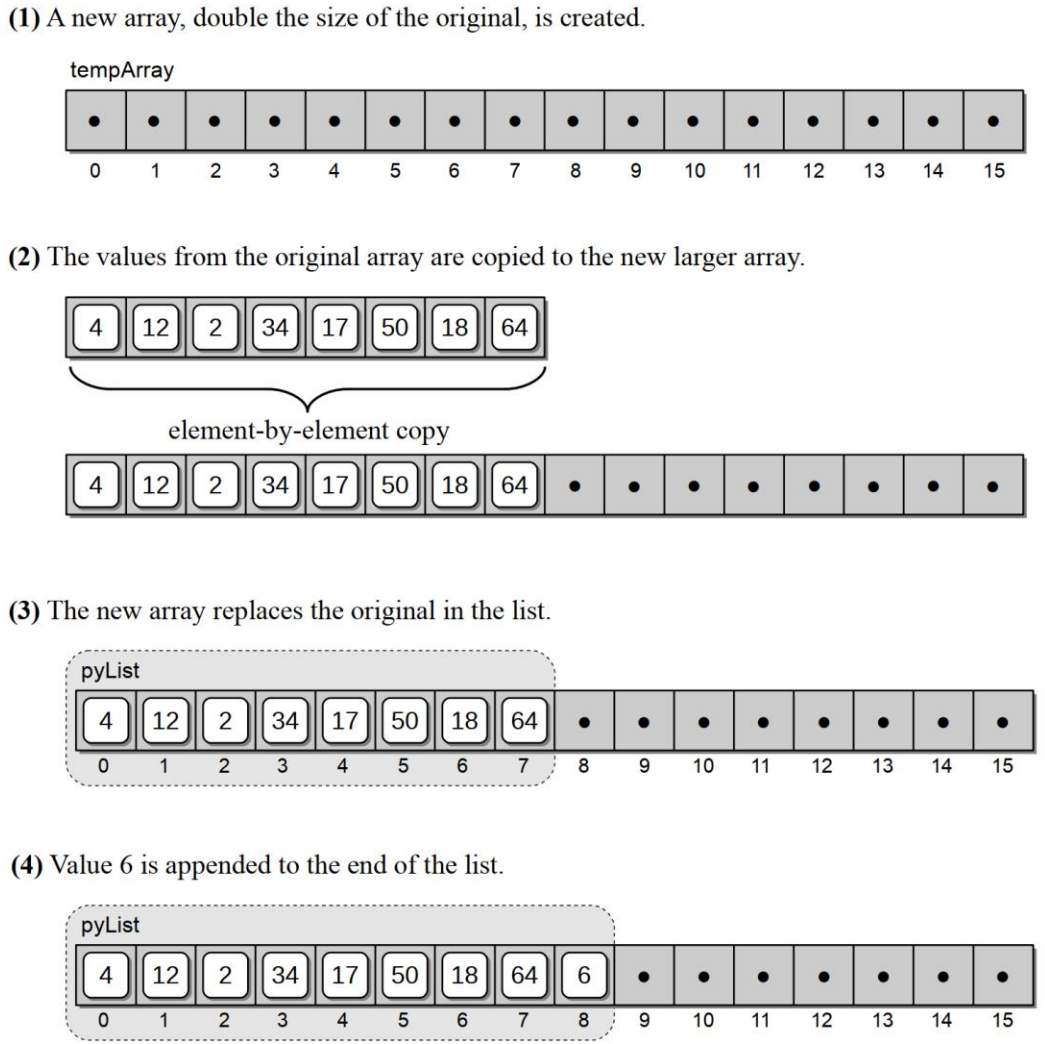


Figure 2.5: The steps required to expand the array to provide space for value 6.

2.2.3 Extension de liste

- *Une liste peut etre ajoutée à la fin une seconde liste à l'aide de la methode* ***extend()***

```
listA = [34, 12]
listB = [4, 6, 31, 9]
listA.extend(listB)
```

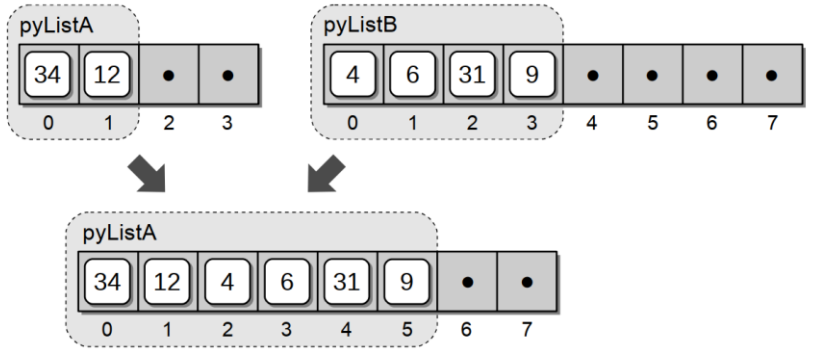
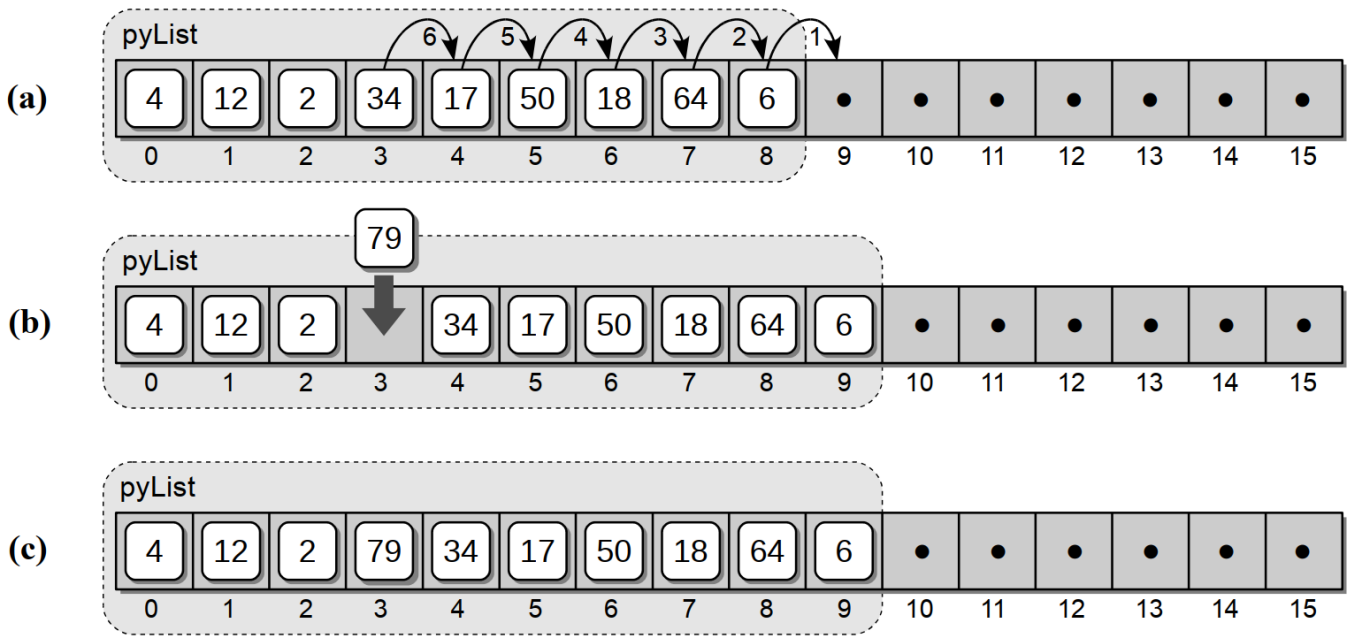


Figure 2.6: The result of extending `pyListA` with `pyListB`.

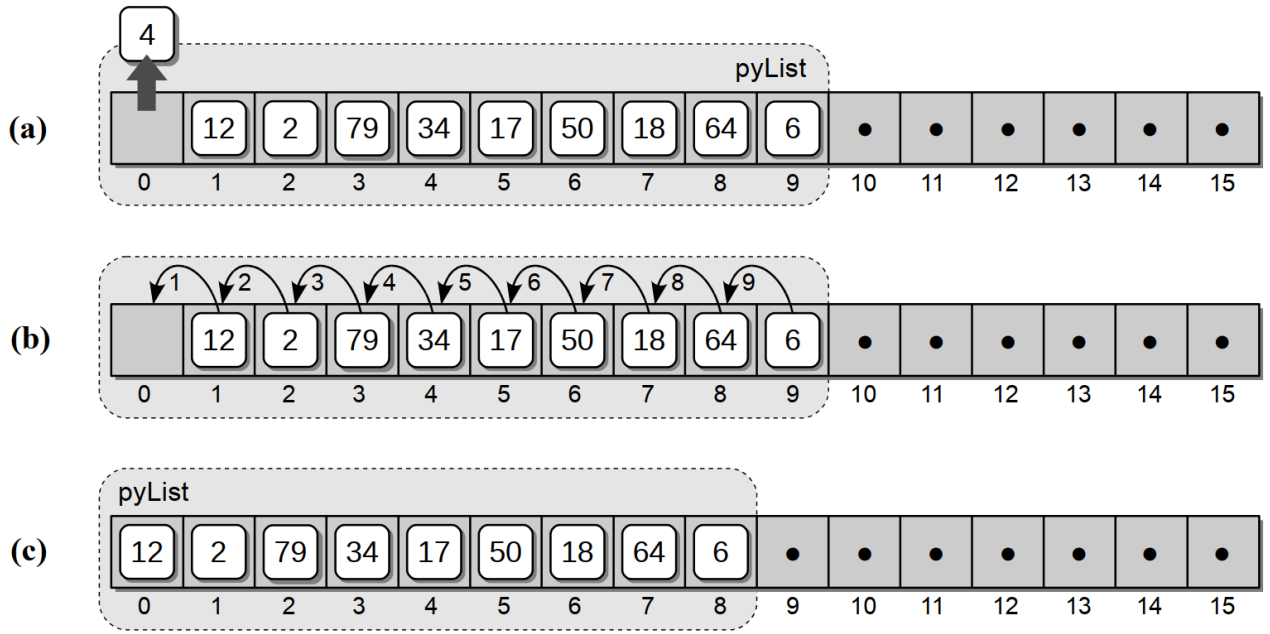
2.2.4 Insertion d'item

- Un item peut être inséré n'importe où dans une liste grâce à la méthode insert
- ***MyList.insert(3, 79)*** insert la valeur 79 a la position 3. Du moment où il y a déjà un item a cette position, il se passe un décalage de tous les items à partir de celle a cette position et ceux d'après.



2.2.5 Retrait d'item

- Un item peut également être retire à n'importe quelle position par la methode pop()
- **pop(0)** retire le premier item
- **pop()** retire le dernier item
- Ce retrait s'accompagne d'un décalage a gauche et de la réduction de la taille du tableau sur la base de la technique mise en œuvre lors de l'extension.



2.2.6 Tranche de liste

- Trancher une liste python revient à en créer une nouvelle consistant en un ensemble contigu d'éléments de la liste originale
- La liste originale n'est pas modifiée par cette opération
- Cela est fait en python via l'opérateur des crochets
- Le premier chiffre désigne l'index de départ le second chiffre le nombre d'elements de la tranche. Ces deux chiffres sont séparés par :

aSlice = pyList[2:3]

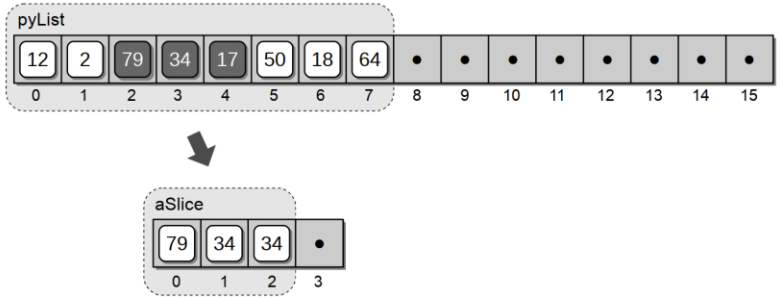


Figure 2.9: The result of creating a list slice.

2.3 Tableau bi-dimensionnel

Certains problèmes nécessitent l'usage de table da deux dimensions qui organise les données en lignes et colonnes similaires. Un élément est accessible au moyen de deux indices

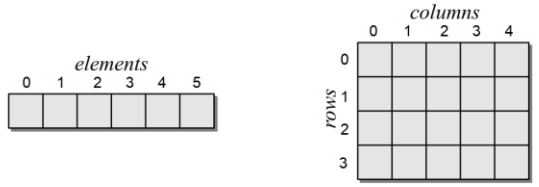


Figure 2.10: Sample arrays: (left) a 1-D array viewed as a sequential list and (right) a 2-D array viewed as a rectangular table or grid.

2.3.1 Tableau bi-dimensionnel : Type de données abstrait

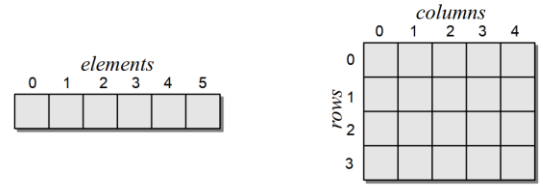


Figure 2.10: Sample arrays: (left) a 1-D array viewed as a sequential list and (right) a 2-D array viewed as a rectangular table or grid.

- Python n'offre pas directement de type représentant les tableaux,
- Autant que les tableaux a dimension 1, il est possible de modéliser un tableau a dimension 2
- Pour le faire, nous allons créer notre classe ***Array2D***
- ***Array2D(nRows, nCols)*** va créer un tableau de dimension 2 consistant en ***nRows*** lignes et ***nCols*** colonnes.
- ***numRows()*** retourne le nombre de lignes
- ***numCols()*** retourne le nombre de colonnes
- ***clear(value)*** réinitialise le tableau avec comme valeur par défaut ***value***
- ***getitem(i, j)*** retourne la valeur positionne a la ***jeme*** colonne de la ***ieme*** ligne
- ***setitem(i1, i2, value)*** modifie la valeur positionne a la ***jeme*** colonne de la ***ieme*** ligne

-

2.3.2 Implémentation

Nous allons implémenter le tableau 2D comme un tableau de tableaux.
Lorsque nous utilisons un tableau de tableaux pour stocker les éléments d'un tableau 2D, nous stockons chaque ligne du tableau 2D dans son propre tableau 1D.

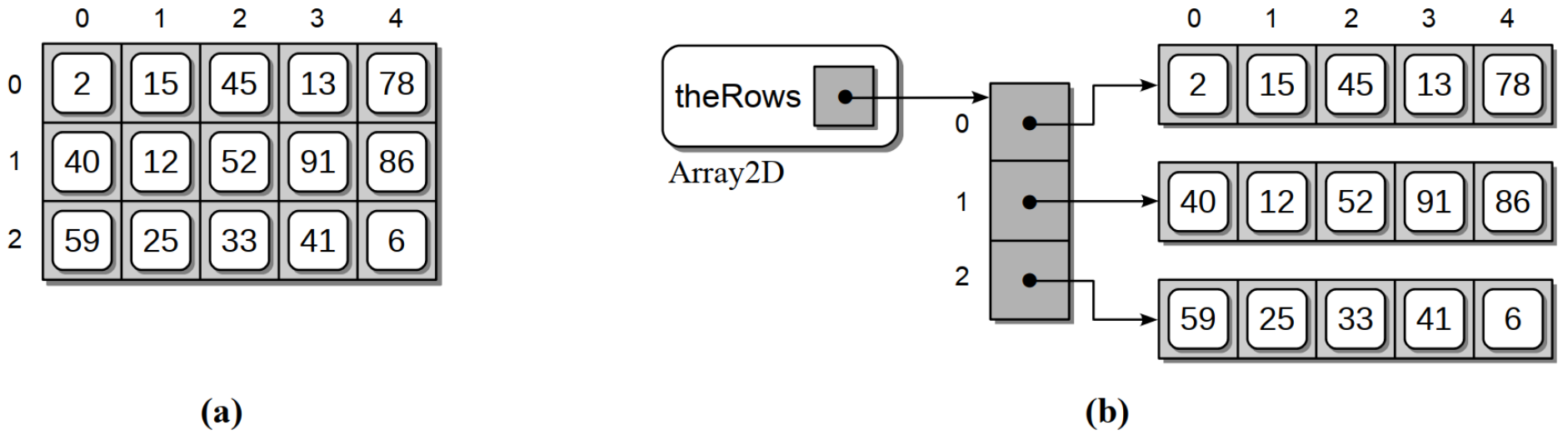
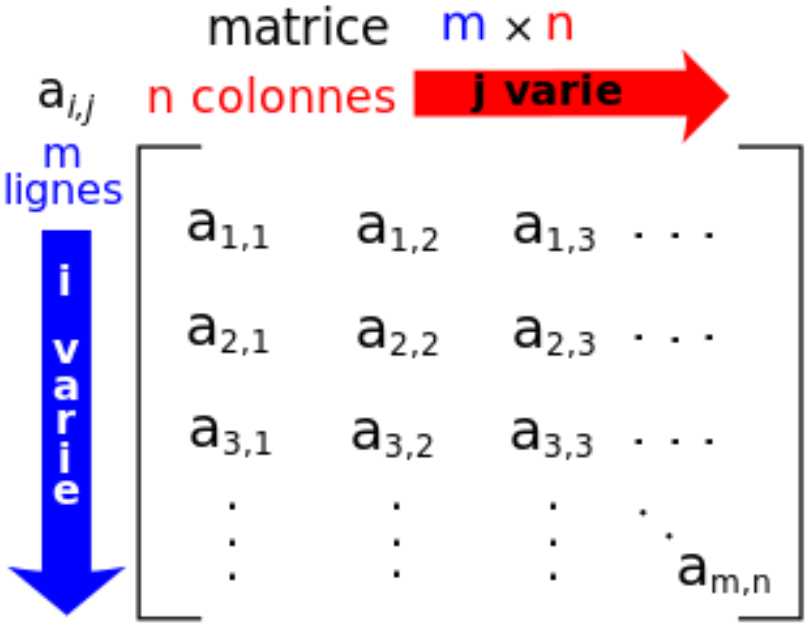


Figure 2.12: A sample 2-D array: (a) the abstract view organized into rows and columns and (b) the physical storage of the 2-D array using an array of arrays.

2.4 TP : Implémentation d'une classe Matrix

- Une matrice est une collection de valeurs scalaires en lignes et colonnes de dimensions fixes. Les éléments sont accessibles aux moyens d'indices partant de 0
- Dans ces travaux pratiques, nous allons implémenter une classe Matrix



2.4.1 Définition de la classe Matrix

Matrix(nrows, ncols) : crée une nouvelle matrice ayant ***nrows*** lignes et ***ncols*** colonnes

nRows() : retourne le nombre de lignes de la matrice

nCols() retourne le nombre de colonnes de la matrice

setItem(row, col) retourne la valeur stocke a l'intercession entre la ligne ***row*** et la colonne ***col***

getItem(row, col, value) modifie l'élément qui se situe à la ligne ***row*** et la colonne ***col*** par ***value***

scaleBy(value) multiplie chaque élément de la matrice par value

transpose() retourne une nouvelle matrice qui vaut la transposée de la matrix

2.4.1 Définition de la classe Matrix

add(mat) crée et retourne une nouvelle matrice qui est le résultat de l'addition de la matrice a ***mat***

subtract(mat) Effectue la même opération que add( ) , mais par soustraction

multiply(mat) Effectue la même opération que add( ) , mais par multiplication

loadFromFile(fileName) charge la matrice a partir d'un fichier exerne filename

print() : affiche le tableau des valeurs numeriques de la matrice