FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 7:
Les listes

Licence : Physique Chimie

Pr: Youssef Ouassit

Structures de données

Définition:

Les structures de données sont des manières d'organiser et de stocker des données dans un ordinateur afin de les utiliser efficacement. Voici un aperçu des principales structures de données :

- Les tableaux
- Les listes
- Les graphes
- Les arbres
- …

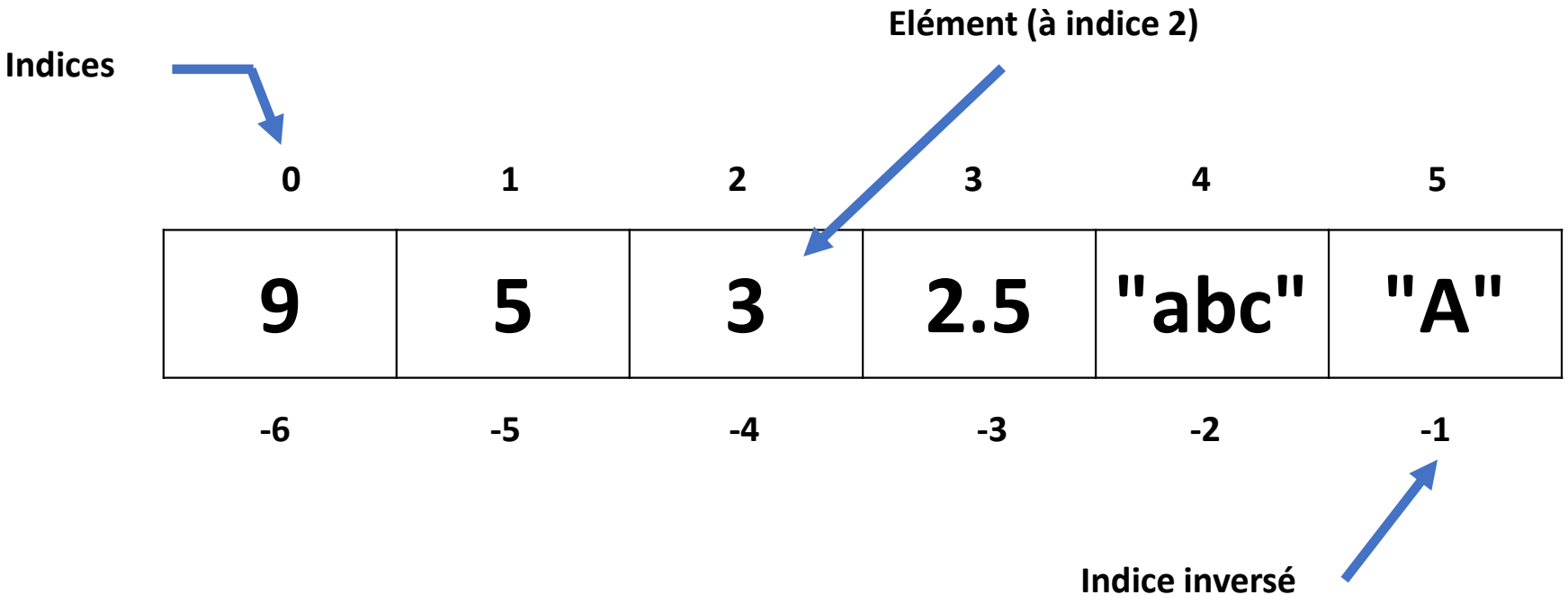
Structures de données

Comment choisir une structure de données:

Choisir la bonne **structure de données** est une étape cruciale dans la conception d'un programme ou d'un système, car cela impacte la **performance**, la **lisibilité**, et la **maintenabilité** du code. Pour faire ce choix, il est important de considérer plusieurs facteurs, tels que les **opérations** que vous devez effectuer, les **caractéristiques** des données, les **contraintes de performance**, et l'**usage prévu** de la structure.

Définition

Une liste est une séquence d'éléments linéaire, que nous pouvons représenter graphiquement sous la forme suivante :



Définition

En Python, les listes sont des structures de données très flexibles et largement utilisées. Voici un aperçu de leurs caractéristiques, de leur utilisation et de quelques opérations courantes.

Caractéristiques des Listes en Python

1. **Dynamique :** Les listes peuvent contenir un nombre variable d'éléments. Tu peux ajouter ou retirer des éléments facilement.
2. **Ordonnée :** Les éléments d'une liste conservent leur ordre d'insertion.
3. **Hétérogène :** Les listes peuvent contenir des éléments de différents types (nombres, chaînes, objets, etc.).
4. **Mutabilité :** Les listes sont mutables, ce qui signifie que tu peux modifier leurs éléments après leur création.

Déclaration

Pour déclarer une liste vide:

```
L = [ ]
```

Pour déclarer une liste initialisée:

```
A = [ 9, 16 , 2 , 7 , 5 , 30 ]
B = [ 9, 5 , 3 , 2.5 , "abc" , "A" ]
```

Les opérateurs sur les listes

Opérateur d'indexation:

Pour accéder à un élément on utilise l'opérateur d'indexation [ ]:

L = [9, 1, 3, 9, 5]

- print(L[2]) # affiche la valeur 3
- L[4] = 30 # permet de modifier la valeur de la case d'indice **4** par **30**.

Les opérateurs sur les listes

Opérateur d'indexation:

Pour accéder à un élément on utilise l'opérateur d'indexation []:

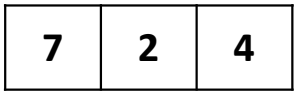
```
fruits = ["pomme", "banane", "cerise"]

print(fruits[0])  # 'pomme'
print(fruits[-1])  # 'cerise'
```

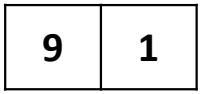
Les opérateurs sur les listes

Opérateur de concaténation :

A = [7 , 2 , 4]



B = [9 , 1]



C = A + B

Résultat :

C = [7 , 2 , 4 , 9 , 1]

Les opérateurs sur les listes

Opérateur d'appartenance :

Pour vérifier si une valeur appartient à une liste, on utilise l'opérateur **in** :

```
L = [ 5  ,  8 ,  7  , 2  , 4  ]

5 in L   # donne True
1 in L   # donne False
```

Les opérateurs sur les listes

Opérateur de duplication :

Nous pouvons multiplier une liste par un entier, les éléments de la liste seront multipliés:

```
L = [ 5  ,  8 ,  1 ]

A = L * 2
A = [ 5  ,  8 ,  1 , 5  ,  8 ,  1 ]
```

Les opérateurs sur les listes

Opérateur de slicing (les tranches) :

Il est possible d'extraire une partie d'une liste avec l'opérateur L[debut : fin : pas] :

```
L = [0, 1, 2, 3, 4, 5]
L1 = L[1:4]  # [1, 2, 3]
L2 = L[:3]   # [0, 1, 2]
L3 = L[::2]  # [0, 2, 4]
L4 = L[:]  # [0, 1, 2, 3, 4, 5]
```

Les fonctions sur les listes

Nombre des éléments:

La fonction **len** permet de calculer le nombre des éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

n = len(L)
n = 3
```

Les fonctions sur les listes

Valeur maximale d'une liste:

La fonction **max** permet de calculer le maximum des éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

n = max(L)
n = 8
```

Les fonctions sur les listes

Valeur minimale d'une liste:

La fonction **min** permet de calculer le minimum des éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

n = min(L)
n = 1
```

Les fonctions sur les listes

Somme des éléments d'une liste:

La fonction **sum** permet de calculer la somme des éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

n = sum(L)
n = 14
```

Les fonctions sur les listes

Trier les éléments d'une liste:

La fonction **sorted** permet de trier les éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

A = sorted(L)
A = [1, 5, 8]
B = sorted(L, reverse=True)
B = [8, 5, 1]
```

Les méthodes des listes

Méthode `append()`

Description : Ajoute un élément à la fin de la liste.

Syntaxe : `liste.append(élément)`

Exemple :

```python
fruits = ["pomme", "banane"]
fruits.append("cerise")  # ["pomme", "banane", "cerise"]
```

Les méthodes des listes

Méthode `extend()`

Description : Ajoute les éléments d'une autre liste à la fin de la liste courante.

Syntaxe : `liste.extend(autre_liste)`

Exemple :

```python
liste1 = [1, 2, 3]
liste2 = [4, 5, 6]
liste1.extend(liste2)  # [1, 2, 3, 4, 5, 6]
```

Les méthodes des listes

Méthode `insert()`

Description : Insère un élément à une position spécifique de la liste.

Syntaxe : `liste.insert(index, élément)`

Exemple :

```python
fruits = ["pomme", "cerise"]
fruits.insert(1, "banane")  # ["pomme", "banane", "cerise"]
```

Les méthodes des listes

Méthode `remove()`

Description : Supprime la première occurrence d'un élément dans la liste.

Syntaxe : `liste.remove(élément)`

Exemple :

python

```python
fruits = ["pomme", "banane", "cerise", "banane"]
fruits.remove("banane")  # ["pomme", "cerise", "banane"]
```

Les méthodes des listes

Méthode `pop()`

Description : Supprime et retourne l'élément à une position donnée (par défaut, le dernier).

Syntaxe : `liste.pop([index])`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
dernier_fruit = fruits.pop()  # "cerise"
```

Les méthodes des listes

Méthode `clear()`

Description : Supprime tous les éléments de la liste.

Syntaxe : `liste.clear()`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
fruits.clear()  # []
```

Les méthodes des listes

Méthode `index()`

Description : Retourne l'index de la première occurrence d'un élément dans la liste.

Syntaxe : `liste.index(élément)`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
index_banane = fruits.index("banane")  # 1
```

Les méthodes des listes

Méthode `count()`

Description : Compte le nombre de fois qu'un élément apparaît dans la liste.

Syntaxe : `liste.count(élément)`

Exemple :

```python
fruits = ["pomme", "banane", "cerise", "banane"]
nombre_banane = fruits.count("banane")  # 2
```

Les méthodes des listes

Méthode `sort()`

Description : Trie les éléments de la liste en place.

Syntaxe : `liste.sort([key=None, reverse=False])`

Exemple :

python

Copier le code

```python
chiffres = [3, 1, 4, 1, 5, 9]
chiffres.sort()  # [1, 1, 3, 4, 5, 9]
```

Les méthodes des listes

Méthode `reverse()`

Description : Inverse l'ordre des éléments de la liste en place.

Syntaxe : `liste.reverse()`

Exemple :

```python
chiffres = [1, 2, 3]
chiffres.reverse()  # [3, 2, 1]
```

Les méthodes des listes

Méthode `copy()`

Description : Retourne une copie superficielle de la liste.

Syntaxe : `liste.copy()`

Exemple :

python Copier le code

```python
fruits = ["pomme", "banane", "cerise"]
fruits_copie = fruits.copy()  # ["pomme", "banane", "cerise"]
```

Les tuples

Définition d’un tuple

Un tuple est une séquence ordonnée d'éléments, qui peuvent être de types différents (par exemple, entiers, chaînes de caractères, listes, etc.).

Contrairement aux listes, les tuples sont immuables, ce qui signifie que leurs éléments ne peuvent pas être modifiés après leur création. Une fois qu'un tuple est défini, il ne peut pas être modifié, étendu ou réduit.

Les tuples

Déclaration

```
# Tuple vide
tuple_vide = ()

# Tuple avec des éléments
mon_tuple = (1, 2, 3, 4, 5)

# Tuple avec des éléments sans parenthèses
mon_tuple = 1, 2, 3, 4, 5

# Tuple avec des types différents
mon_tuple_melangee = (1, "Python", 3.14, True)
```

Les tuples

Déclaration

```
# Tuple vide avec tuple()
mon_tuple_vide = tuple()

# Tuple à partir d'une chaîne de caractères
mon_tuple_chaine = tuple("Python") # ('P','y','t','h','o','n’)

# Tuple à partir d’une liste
mon_tuple_tuple = tuple([1, 2, 3])  # (1, 2, 3)

# Tuple avec un seul élément
mon_tuple_vide = (1, )
```

Les tuples

Opérateurs de base

Opérateur d'Appartenance **in**

```
fruits = ("pomme", "banane", "cerise" )
print("pomme" in fruits)  # True
print("orange" in fruits)  # False
```

Opérateur de concaténation **+**

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
tuple3 = tuple1 + tuple2  # (1, 2, 3, 4, 5, 6)
```

Les tuples

Opérateurs de base

Opérateur de répétition *

```
tuple = [1, 2, 3]
resultat = tuple * 3  # (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Opérateur d’indexation []

```
fruits = ("pomme", "banane", "cerise« )
print(fruits[0])  # 'pomme'
print(fruits[-1])  # 'cerise'
```

Opérateurs de base

Opérateur de Slicing **[debut : fin : pas]**

```
chiffres = (0, 1, 2, 3, 4, 5)
sous_tuple = chiffres[1:4]  # (1, 2, 3)
sous_tuple2 = chiffres[:3]   # (0, 1, 2)
sous_tuple3 = chiffres[::2]  # (0, 2, 4)
sous_tuple4 = chiffres[:]  # (0, 1, 2, 3, 4, 5)
```

Les tuples

Les méthodes

Méthode : `count`

Description : Retourne le nombre d'occurrences d'un élément spécifique dans le tuple.

Syntaxe : `tuple.count(element)`

Exemple :

```python
mon_tuple = (1, 2, 2, 3)
print(mon_tuple.count(2))  # Affiche : 2
```

Les tuples

Les méthodes

Méthode : `index`

Description : Retourne l'index de la première occurrence d'un élément spécifique dans le tuple. Lève une exception `ValueError` si l'élément n'est pas trouvé.

Syntaxe : `tuple.index(element, start=0, end=len(tuple))`

Exemple :

```python
mon_tuple = (1, 2, 3)
print(mon_tuple.index(2))  # Affiche : 1
```

Les fonctions

Les fonctions sur les listes sont aussi applicable sur les tuples :

- len
- min
- max
- sum
- sorted
- reversed