|  |
| --- |
| STRUCTURES DE DONNEES EN C |
| TRAVAUX DE RECHERCHES |
| DJOSSOU\_SEGUE\_TOGNI |

|  |
| --- |
| 04/06/2025 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Structure de données** | **C** | **Python** | **Java** |
| **Tableau (Array)** | **- Implémentation**: Déclaré avec :  int arr[taille].  **- Accès**: Par indiçage  - **Limite**: Taille définie à la compilation, pas de redimensionnement dynamique natif.  - Exemple: int arr[5] = {1, 2, 3, 4, 5}; | - **Implémentation**: list (tableau dynamique sous-jacent). - **Accès** : Par indiçage  - **Fonctionnalités**: Méthodes comme append(), pop(), etc.  - Exemple: arr = [1, 2, 3, 4, 5] | - **Implémentation**:  int[] arr = new int[taille];.  - **Accès**: Par Indiçage  - **Limite**: Taille fixée à la création.  - Exemple: int[] arr = {1, 2, 3, 4, 5}; ou ArrayList<Integer> list = new ArrayList<>(); |
| **Liste chaînée** | - **Implémentation**: Structures personnalisées avec pointeurs (struct Node { int data; struct Node\* next; };).  - **Avantage**: Flexible pour insertions/suppressions.  - **Inconvénient**: Pas de bibliothèque standard. | - **Implémentation**: Pas de liste chaînée native, mais collections.deque pour des opérations similaires.  - Exemple: from collections import deque; d = deque([1, 2, 3]) | - **Implémentation**: LinkedList dans java.util.  - **Avantage**: Bibliothèque standard, facile à utiliser.  - Exemple: LinkedList<Integer> list = new LinkedList<>(); list.add(1); |
| **Pile (Stack)** | - **Implémentation**: Via tableau ou liste chaînée personnalisée.  - **Opérations**: LIFO (push/pop, O(1)).  - **Inconvénient**: Implémentation manuelle requise.  - Exemple: Implémentation via struct ou tableau. | .- **Implémentation**: Liste (list) avec append() (push) et pop().  - **Opérations**: LIFO, O(1) pour push/pop.  - Exemple: stack = []; stack.append(1); stack.pop() | - **Implémentation**: Stack dans java.util ou Deque (préféré).  - **Opération**s: LIFO, O(1) pour push/pop.  - Exemple: Stack<Integer> stack = new Stack<>(); stack.push(1); stack.pop(); |
| **File (Queue)** | - **Implémentation:**  Via tableau ou liste chaînée. - **Opérations**: FIFO (enqueue/dequeue, O(1)).  - **Inconvénient**: Gestion manuelle des limites.  - Exemple: Implémentation via struct. | - **Implémentation:** collections.deque pour FIFO efficace.  - **Opérations**: O(1) pour enqueue/dequeue.  - Exemple: from collections import deque; q = deque(); q.append(1); q.popleft() | - **Implémentation:**  Queue (interface) via LinkedList ou ArrayDeque. - **Opérations**: FIFO, O(1) pour enqueue/dequeue.  - Exemple: Queue<Integer> queue = new LinkedList<>(); queue.offer(1); queue.poll(); |
| **Arbre (Tree)** | - **Implémentation**: Structures personnalisées avec pointeurs (struct Node { int data; struct Node\* left; struct Node\* right; };).  - Exemple: Arbre binaire, implémentation manuelle. - Inconvénient: Pas de bibliothèque standard. | - **Implémentation**: Classes personnalisées ou bibliothèques comme anytree.  - Exemple: class Node: pass pour arbre binaire.  - Inconvénient: Moins direct, souvent via bibliothèques externes. | - **Implémentation**: TreeSet ou TreeMap pour arbres binaires de recherche, ou classes personnalisées. - Exemple: TreeSet<Integer> tree = new TreeSet<>(); tree.add(1);. - Avantage: Bibliothèque standard pour certains arbres. |
| **Tableau associatif (Hash Map)** | - Type: Manuelle. - Implémentation: Tables de hachage personnalisées avec tableaux et listes chaînées. - Accès: O(1) moyen pour recherche/insertion. - Inconvénient: Pas de bibliothèque standard, complexe à implémenter. | - Type: Dynamique. - Implémentation: dict (tableau associatif natif). - Accès: O(1) moyen pour recherche/insertion. - Exemple: d = {'key': 'value'}; d['key'] | - Type: Dynamique. - Implémentation: HashMap dans java.util. - Accès: O(1) moyen pour recherche/insertion. - Exemple: HashMap<String, Integer> map = new HashMap<>(); map.put("key", 1); |
| **Ensemble (Set)** | - Type: Manuelle. - Implémentation: Via tables de hachage ou arbres personnalisés. - Inconvénient: Pas de bibliothèque standard, implémentation complexe. | - Type: Dynamique. - Implémentation: set (basé sur table de hachage). - Opérations: O(1) moyen pour ajout/recherche. - Exemple: s = {1, 2, 3}; s.add(4) | - Type: Dynamique. - Implémentation: HashSet ou TreeSet dans java.util. - Opérations: O(1) pour HashSet, O(log n) pour TreeSet. - Exemple: HashSet<Integer> set = new HashSet<>(); set.add(1); |