Structure de données	C	Python	Java
Tableau (Array)	- Type: Statique (taille fixe). - Implémentation: Déclaré avec `int arr[taille]`. - Accès: Indexé (O(1)). - Limite: Taille définie à la compilation, pas de redimensionnement dynamique natif. - Exemple: `int arr[5] = {1, 2, 3, 4, 5};`	- Type: Dynamique (liste). - Implémentation: `list` (tableau dynamique sous-jacent). - Accès: Indexé (O(1)), redimensionnable. - Fonctionnalités: Méthodes comme `append()`, `pop()`, etc. - Exemple: `arr = [1, 2, 3, 4, 5]`	- Type: Statique (taille fixe). - Implémentation: `int[] arr = new int[taille];`. - Accès: Indexé (O(1)). - Limite: Taille fixée à la création. - Alternative dynamique: `ArrayList`. - Exemple: `int[] arr = {1, 2, 3, 4, 5};` ou `ArrayList<Integer> list = new ArrayList<>();`
Liste chaînée (Linked List)	- Type: Manuelle. - Implémentation: Structures personnalisées avec pointeurs (`struct Node { int data; struct Node* next; };`). - Accès: O(n) pour recherche/insertion. - Avantage: Flexible pour insertions/suppressions. - Inconvénient: Pas de bibliothèque standard.	- Type: Dynamique. - Implémentation: Pas de liste chaînée native, mais `collections.deque` pour des opérations similaires. - Accès: O(n) pour accès par index, O(1) pour ajout début/fin avec `deque`. - Exemple: `from collections import deque; d = deque([1, 2, 3])`	- Type: Dynamique. - Implémentation: `LinkedList` dans `java.util`. - Accès: O(n) pour recherche, O(1) pour ajout début/fin. - Avantage: Bibliothèque standard, facile à utiliser. - Exemple: `LinkedList<Integer> list = new LinkedList<>(); list.add(1);`
Pile (Stack)	- Type: Manuelle. - Implémentation: Via tableau ou liste chaînée personnalisée. - Opérations: LIFO (push/pop, O(1)). - Inconvénient: Implémentation manuelle requise. - Exemple: Implémentation via `struct` ou tableau.	- Type: Dynamique. - Implémentation: Liste (`list`) avec `append()` (push) et `pop()`. - Opérations: LIFO, O(1) pour push/pop. - Exemple: `stack = []; stack.append(1); stack.pop()`	- Type: Dynamique. - Implémentation: `Stack` dans `java.util` ou `Deque` (préféré). - Opérations: LIFO, O(1) pour push/pop. - Exemple: `Stack<Integer> stack = new Stack<>(); stack.push(1); stack.pop();`
File (Queue)	- Type: Manuelle. - Implémentation: Via tableau ou liste chaînée. - Opérations: FIFO (enqueue/dequeue, O(1)).	- Type: Dynamique. - Implémentation: `collections.deque` pour FIFO efficace. - Opérations: O(1) pour	- Type: Dynamique. - Implémentation: `Queue` (interface) via `LinkedList` ou `ArrayDeque`.

	- Inconvénient: Gestion manuelle des limites. - Exemple: Implémentation via `struct`.	enqueue/dequeue. - Exemple: `from collections import deque; q = deque(); q.append(1); q.popleft()`	- Opérations: FIFO, O(1) pour enqueue/dequeue. - Exemple: `Queue<Integer> queue = new LinkedList<>(); queue.offer(1); queue.poll();`
Arbre (Tree)	- Type: Manuelle. - 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.	- Type: Non natif. - Implémentation: Classes personnalisées ou bibliothèques comme `anytree`. - Exemple: `class Node: pass` pour arbre binaire. - Inconvénient: Moins direct, souvent via bibliothèques externes.	- Type: Dynamique. - 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);`