MINISTÈRE
DE L'ÉDUCATION
NATIONALE,
DE LA JEUNESSE
ET DES SPORTS
*Liberté
Égalité
Fraternité*

VOIE GÉNÉRALE

2DE | 1RE | TLE

Numérique et sciences informatiques

ENSEIGNEMENT
SPÉCIALITÉ

TYPES ABSTRAITS DE DONNÉES - PRÉSENTATION

Introduction

Acquis de première

En première, les tableaux ont été introduits en utilisant le type `list` de Python. Seuls des tableaux d'éléments de même type ont été utilisés (programme de première, p5). Le vocabulaire propre à Python peut induire en erreur et amener à penser que le type « liste » est déjà connu. Utiliser le vocabulaire « liste chainée » pour le type abstrait de données « liste » lorsque c'est pertinent peut permettre de lever l'ambigüité, comme cela est fait dans la seconde ressource dédiée à l'implémentation en Python de ces structures. En revanche, la notion de type construit a été présentée et les élèves ont déjà pu utiliser les structures de type `list`, `dict` et `tuple` par leur interface Python.

Place dans la progression de terminale

Cette partie du cours sur les structures linéaires de données peut être traitée sans pré-requis du programme de terminale, au moins dans un premier temps. La ressource « Types abstraits de données - Implantations et propositions de mise en œuvre » permet de voir que la récursivité et la programmation orientée objet peuvent y trouver une place, notamment dans le cadre d'une progression spiralaire.

Type abstrait de données

Un type de données est un ensemble de valeurs associé aux actions - les méthodes - que l'on peut mener dessus. L'objet de cette ressource est de décrire les types abstraits de données liste, pile et file.

Du côté de l'utilisateur du type abstrait de données

Du côté de l'utilisateur, un type abstrait de données peut être vu comme un type de données dont on ne connaît pas la réalisation concrète. Ainsi, les structures de

type `list`, `dict` et `tuple` correspondent à des types abstraits de données. Un type abstrait de données peut donc être vu par l'utilisateur comme un ensemble de données associé à une description précise de ce qu'il peut réaliser dessus. Par exemple, un dictionnaire associe à un ensemble de clés un ensemble de valeurs (ce sont les données) et permet notamment les opérations :

- d'ajout d'un couple clé-valeur ;
- de suppression d'une clé, et donc de la valeur correspondante ;
- de modification de la valeur associée à une clé et ;
- de recherche de la valeur correspondant à une clé.

On remarque que ce type abstrait de données est mutable (on peut modifier son contenu après création).

Du côté du concepteur du type abstrait de données

Du côté du concepteur de la mise en œuvre de cette structure de données, c'est la réalisation concrète dans le langage choisi qui va être prépondérante. De la qualité de cette implantation va dépendre l'efficacité de la structure.

À l'interface de l'utilisateur et du concepteur : l'API ou la bibliothèque

Les API (ou interface de programmation, en français) ont permis d'utiliser tableaux, dictionnaires ou p-uplets en première. L'usage des bibliothèques permet à chaque programmeur d'ajouter des structures réalisant des types abstraits de données, cette implantation n'étant pas nécessairement connue de l'utilisateur de la structure. Cette méthode de conception logicielle, utilisant l'encapsulation, permet à la fois :

- le développement séparé de l'application et l'implantation de la structure ;
- la modification de l'implantation sans modification de ses utilisations ;
- l'utilisation facile de l'implantation de la structure dans des programmes à venir ;
- la limitation des erreurs ;
- l'ajout :
  - de vérifications sous forme d'assertions ;
  - d'outils de correction des problèmes de programmation ;
- une meilleure lisibilité du code de l'application.

Types abstraits de données linéaires

Dans cette ressource, on se limite aux types abstraits de données qui stockent les données sous forme de collection unidimensionnelle ; ce sont les types linéaires, dont nous présentons les listes, les piles et les files.

Liste

Présentation

Comme on l'a déjà vu, la structure `list` de Python réalise en fait l'implantation du type abstrait de données « tableau dynamique » et doit ici être laissée de côté, malgré l'utilisation du même vocabulaire. La liste permet de stocker des données et d'y accéder directement. C'est un type abstrait de données :

- linéaire : les données sont stockées dans une structure unidimensionnelle ;
- indexé : chaque donnée est associée à une valeur ;
- ordonné : les données sont présentées les unes après les autres.

Le type abstrait de données

Une liste est une collection finie de données. On appelle « tête » le premier élément de la liste et « queue » la liste privée de son premier élément. Il est seulement possible d'ajouter et de lire une donnée en tête de la liste.

5 primitives permettent de définir le type abstrait de données :

- `listeCree()`, qui crée une liste vide ;
- `listeAjout(liste, element)`, qui ajoute un élément en tête de liste ; ces 2 première primitives peuvent parfois se regrouper en une seule ;
- `listeTete(liste)`, qui renvoie la valeur de l'élément en tête de liste ;
- `listeQueue(liste)`, qui renvoie la liste privée de son premier élément ;
- `listeEstVide(liste)`, qui renvoie `vrai` si la liste est vide, `faux` sinon.

On peut constater que le type abstrait de données est non mutable. L'intérêt de ce choix a déjà été développé dans la ressource « Types mutables et problèmes associés » dédiée au programme de première.

Remarques

De nombreux autres algorithmes sur les listes peuvent être ajoutés en fonction des besoins et peuvent faire l'objet d'exercices en fonction des besoins des activités proposées :

- renvoyer une liste dont l'ordre des éléments a été inversé ;
- renvoyer la longueur, c'est-à-dire le nombre des éléments de la liste ;
- accéder au $i^{ième}$ élément d'une liste ;
- rechercher un élément dans une liste en renvoyant au choix :
  - "Vrai" si l'élément sa position, « Faux » sinon ;
  - la position de la première occurrence de l'élément recherché ;
  - la liste des positions de toutes ses occurrences ;
- renvoyer une nouvelle liste dont le $i^{ème}$ élément a été supprimé ;
- renvoyer une nouvelle liste correspondant à la liste d'origine à laquelle un élément a été ajouté à la fin ;
- toute primitive pouvant correspondre à un besoin spécifique lié notamment à un projet ou à un exercice.

D'autre part, des types abstraits de données plus élaborés peuvent être construits à partir du type initial, que l'on nomme alors « liste chaînée » :

- listes doublement chaînées permettant de se déplacer vers la queue, mais aussi vers la tête ;
- listes circulaires dont le premier élément suit le dernier ;
- listes circulaires doublement chainées combinant les deux propriétés précédentes.

Pile

Présentation

La pile, comme la liste, permet de stocker des données et d'y accéder. La différence se situe au niveau de l'ajout et du retrait d'éléments.

On parle de **mode LIFO** (Last In, First Out, donc, dernier arrivé, premier sorti), c'est-à-dire que le dernier élément ajouté à la structure sera le prochain élément auquel on accèdera. Les premiers éléments ayant été ajoutés devront « attendre » que tous les éléments qui ont été ajoutés après eux soient sortis de la pile. Contrairement aux listes, on ne peut donc pas accéder à n'importe quelle valeur de la structure (pas d'index). Pour gérer cette contrainte, on définit alors le **sommet** de la pile qui caractérise l'emplacement pour ajouter ou retirer des éléments.

On peut s'imaginer une pile d'assiettes pour mieux se représenter mentalement cette structure. On ajoute des nouvelles assiettes au sommet de la pile, et quand on veut en retirer une, on est obligé de prendre celle située au sommet.

Le type abstrait de données

Une pile est une collection de données. On appelle **sommet** le dernier élément ajouté à la pile, qui est aussi le seul élément auquel on peut accéder, ou qu'on peut retirer. Quand un élément est ajouté à la pile, il devient le nouveau sommet, et l'ancien sommet est alors son élément **suivant**. Quand un élément est retiré de la pile, le nouveau sommet est l'élément qui suivait le sommet avant le retrait.

6 primitives permettent de définir le type abstrait de données :

- `pileCree()`, qui crée une pile vide ;
- `pileTaille(pile)`, qui permet de connaître le nombre d'éléments contenus dans la pile ;
- `pileEstVide(pile)`, qui renvoie `vrai` si la pile est vide, `faux` sinon ;
- `pileEmpiler(pile, element)`, qui ajoute un élément au sommet de la pile (qui devient le nouveau sommet) ;
- `pileDepiler(pile)`, qui retire et renvoie l'élément situé au sommet de la pile (le nouveau sommet devient l'élément qui suivait l'ancien sommet) ;
- `pileSommet(pile)`, qui renvoie l'élément situé au sommet de la pile (sans le retirer).

Remarques

La pile est utile dans différents types de problèmes :

- algorithme d'un navigateur pour pouvoir mémoriser les pages web et revenir en arrière (ou ré-avancer) sur certaines pages ;
- stocker des actions et les annuler (ou les réappliquer), sur l'ordinateur (fonction CTRL+Z, et CTRL+Y) ;

- coder une calculatrice en notation polonaise inversée ;
- algorithme du **parcours en profondeur** pour les arbres et les graphes, par exemple, pour résoudre un labyrinthe, trouver un trajet sur une carte…
- écrire des versions itératives de certains algorithmes récursifs ;
- illustration du fonctionnement de la pile d'appels des fonctions lors de l'exécution d'un programme.

File

Présentation

La file, comme la liste, permet de stocker des données et d'y accéder. La différence se situe au niveau de l'ajout et du retrait d'éléments.

On parle de **mode FIFO** (First in, First out, donc, premier arrivé, premier sorti), c'est-à-dire que le premier élément ayant été ajouté à la structure sera le prochain élément auquel on accèdera. Les derniers éléments ajoutés devront « attendre » que tous les éléments ayant été ajoutés avant eux soient sortis de la file. Contrairement aux listes, on ne peut donc pas accéder à n'importe quelle valeur de la structure (pas d'index). Pour gérer cette contrainte, la pile est caractérisée par deux « emplacements » :

- la **tête de file**, sortie de la file (début de la structure), où les éléments sont retirés ;
- le **bout de file**, entrée de la file (fin de la structure), où les éléments sont ajoutés.

On peut s'imaginer une file d'attente, dans un cinéma par exemple. Les premières personnes à pouvoir acheter leur place sont les premières arrivées, et les nouveaux arrivants se placent au bout de la file.

Le type abstrait de données

Une file est une collection de données. On appelle **tête de file** le premier élément de la structure et **bout de file** le dernier élément. Quand un élément est ajouté à la file, on l'ajoute en bout de file et il devient le nouveau bout de file c'est-à-dire l'élément « suivant » l'élément situé précédemment en bout de file. Quand un élément est retiré de la file, on le sélectionne à la tête de la file et la nouvelle tête est l'élément qui suivait l'ancienne tête. Lorsqu'on ajoute un élément à une file vide, celui-ci est donc à la fois la tête et le bout de la file.

6 primitives permettent de définir le type abstrait de données :

- `fileCree()`, qui crée une file vide ;
- `fileTaille(file)`, qui permet de connaître le nombre d'éléments contenus dans la file ;
- `fileEstVide(file)`, qui renvoie `vrai` si la file est vide, `faux` sinon ;
- `fileAjouterFin(file, element)`, qui ajoute un élément au bout de la file (et devient le nouveau bout de file) ;
- `fileRetirerTete(file)`, qui retire et renvoie l'élément situé à la tête de la file (la nouvelle tête devient l'élément qui suivait l'ancienne tête) ;
- `fileTete(file)`, qui renvoie l'élément situé à la tête de la file (sans le retirer).

Remarques

La file est utile dans différents types de problèmes :

- pour une imprimante, gestion de la file d'attente des documents à imprimer ;
- modélisation du jeu de la bataille (on révèle la carte au-dessus du paquet et on place celles gagnées en dessous…) ;
- gestion de mémoires tampon, pour gérer les flux de lecture et d'écriture dans un fichier, par exemple ;
- matérialisation d'une file d'attente, pour un logiciel (visioconférence par exemple) ou un jeu (gestion des connexions des utilisateurs, des tours de jeu…),…
- algorithme du **parcours en largeur** pour les arbres et les graphes, par exemple, pour trouver le plus court trajet sur une carte, ou récupérer les valeurs d'une structure dans l'ordre croissant…

Conclusion

Les différents types abstraits au programme ont été décrits dans cette ressource. Leur mise en œuvre ainsi que des possibilités d'intégration à la progression en terminale sont présentés dans la seconde ressource « Types abstraits de données - Implantations et propositions de mise en œuvre ».