Partie 1

Types abstraits

La résolution d'un problème par un programme nécessite :

- de définir les données du problème
- d'utiliser des instructions du langage sur ces données

Souvent, la résolution du problème demande d'arranger ces données dans un *type abstrait*, afin d'avoir une résolution plus efficace.

Un type abstrait est défini par son *interface*, qui est indépendante de son *implémentation*.

Les types abstraits sont des types *structurés*. On a déjà vu des types *structurés* natifs : les types séquences de texte (string), les types séquentiels (listes, tuples), et les mappages (dictionnaires).

Un type abstrait est caractérisé par une *interface* de programmation qui permet de manipuler les données de ce type. Sans pour autant avoir connaissance du contenu des fonctions proposées par l'interface.

Spécifier un type abstrait, c'est définir son *interface*, sans prejuger de la façon dont ses opérations sont implémentées.

Implémenter, c'est fournir le code de ces opérations. Plusieurs implémentations peuvent correspondre à la même spécification. On verra dans ce chapitre la programmation dans un style fonctionnel. Il viendra plus tard la programmation par objet.

On verra dans un premier temps, deux exemples de types abstraits : Les *listes chainées* et les *tableaux*. Dans un chapitre ultérieur, nous verrons les types abstraits *Piles*, *Files*, et *Graphes*.

Partie 2

Tableaux, choix d'une structure de données native en python

Les **tableaux statiques** sont des structures de données de taille fixe, où chaque donnée est du même type. C'est comme ceci que sont représentées les collections dans des langages comme C, Java, Ocaml, ...

La taille du tableau est fixe à sa création : on ne peut pas l'agrandir ou le réduire sans en recréer un nouveau.

Un tableau peut être représenté en Python à l'aide d'une liste (type list), un tuple (type tuple), ou bien une matrice numpy (numpy.array).

2.1 List python

Les tableaux en python sont implémentés par le type `list`, qui est en réalité un **tableau dynamique** : leur taille est variable, et les éléments stockés peuvent être de types différents (idem pour Javascript). Pour utiliser le type `list` en tant que tableau statique, on s'interdira les aspects dynamiques.

1. Création

On rappelle que, pour créer une liste python, on peut utiliser les techniques suivantes :

- création par extension : en listant les éléments du tableau : `T = [1,2,3]`
- par concaténation : On peut utiliser une somme d'éléments à l'aide des opérateurs `+` et `*`, par exemple : `T = [0] * 100`
- par compréhension : on s'appuie sur un itérateur (fonction génératrice), exemple : `T = [i for i in range(ord('A'), ord('Z')+1]`, ou `T = [x**2 for x in range(1,11)]`

- par ajout dans une table vide (append)
- à l'aide de la fonction `list` : `codes = list(range(ord('A'), ord('Z')+1)`,`T = list('bonjour')`

2. Manipuler On accède à un élément d'un tableau à l'aide d'un indice, qui peut être positif : `T[1]` est le deuxière élément par exemple, `T[0]` étant le premier. Ou bien à l'aide d'un entier négatif, ce qui fait que `T[len(T)-1]` est le même que `T[-1]`
3. Parcourir

- Le plus simple est de parcourir directement **par élément**

```
for elem in L:
  print(elem)
```

- parcourir **par indice** :

Très utile si on veut modifier les valeurs lors du parcours :

```
for i in range(len(T)):
  T[i] += 1
```

ou bien lorsque l'on veut parcourir 2 tableaux à la fois :

```
prenoms = ['Agnes','Kev','Sam']
age = [34,28,21]
for i in range(len(prenoms)):
  print("prenom: {}, age:{}".format(prenoms[i],age[i]))
```

2.2 numpy.array

En Python, les tableaux fixes peuvent être implémentés via la classe `array` du module `numpy`.

La dimension est fixée à la construction. Les indices commencent à 0.

```
import numpy as np
t = np.array([1, 2, 3, 4])
print(t[1])  # Affiche 2
```

Pour des tableaux à plusieurs dimensions : On peut accéder aux éléments d'un array A en utilisant la syntaxe `A[i, j]` , où i et j sont respectivement le numéro de ligne et le numéro de colonne de l'élément au sein du tableau. L'opérateur : permet de sélectionner plusieurs éléments, voire tout une ligne ou colonne.

2.3 Tuple

Pour un tuple, celui-ci contiendra 2 données :

- la liste des valeurs, de dimension fixe. On mettra `None` pour les valeurs non renseignés.
- la valeur de la taille de la liste.

Par exemple :

```
T = ([6, 7, 8, None, None], 5)
```

C'est alors un objet qui est :

- **statique** : sa taille ne varie pas une fois celui-ci créé
- **non mutable** : mais avec un élément qui lui, peut être *mutable* :

On ne peut pas modifier `T[0]` ou `T[1]`. Ce sont les éléments d'un tuple.

Mais on peut modifier en partie `T[0]`, par exemple avec une instruction du genre : `T[0][i] = x`. La copie de T[0] se fait par reference.

Un tableau peut servir à implémenter une grille de notes par exemple :

	C1	C2	C3
Kyle	6	7	8
Sean	10	0	10
Quentin	10	0	14
Zinedine	15	0	12

FIGURE 1 – Tableau de notes

Supposons, pour simplifier, que le tableau ne contient que les notes de Kyle :

	C1	C2	C3	C4	C5
Kyle	6	7	8	Non noté	Absent

FIGURE 2 – Tableau de notes de Kyle

On peut représenter cet ensemble de notes par le tableau :

```
T = ([6, 7, 8, None, None], 5)
```

Petits rappels sur le tuples :

- Construction par extension :

```
fruits = ('pommes','oranges')
```

- Construction par concaténation :

```
t = ()
t = t + (1,) # ne pas oublier , sinon cela génère une list
t = t + (2,)
print(t)
(1,2)
```

- Construction par compréhension :

```
carres = tuple(x**2 for x in range(1,11))
# ne pas oublier de mettre tuple au debut
```

- avec la fonction `list` :

```
codes = list(range(ord('A'),ord('Z')+1))
lettres_bonjour = list('bonjour')
```

2.4 L'interface d'un tableau

fonctions qui implémentent un tableau T (Array)	nom
taille du tableau	`taille(T)`
demander l'élément au rang i	`element(T,i)`
remplacer l'élément au rang i par e	`remplacer(T,i,e)`

2.5 Tableaux multidimentionnels

```
hugo = [21, 1400, 0]
richard = [54, 2800, 2]
emilie = [27, 3700, 3]
tableau = [hugo, richard, emilie]
```

Alors, pour accéder à l'un des éléments, on fait : `tableau[ligne i][colonne j]` :

List et tuple

```
>>> tableau[0][1] # correspond à la 2e (j=1) note de hugo (i=0)
1400
```

Array

```
>>> tab1 = np.array([[1, 2],
    [3, 4],
    [5, 6]])
# tab1 est un array de 3 lignes et 2 colonnes
>>> tab2 = np.array([[1, 2, 3],
    [4, 5, 6]])
# tab2 est un array de 2 lignes et 3 colonnes
>>> np.ones((3, 5))
# un tableau de 3x5 rempli de 1
>>> np.zeros((4, 4))
# un tableau de 4 lignes et de 4 colonnes contenant que des 0
```

2.6 Tableaux dynamiques et tableaux statiques

Les tableaux vus ci-dessus sont des tableaux *statiques* : leur taille ne peut pas être modifiée. Dans le cas où l'on ait besoin d'agrandir le tableau, il faut le copier dans un nouveau tableau, plus grand.

Python implémente naturellement un autre type de tableau, que l'on appelera *dynamique* : Les *Listes Python*.

Ce problème de dimension n'apparait pas dans les Listes Python, qui apportent de surcroit des méthodes bien pratiques comme `append` et `pop`.

Partie 3

La liste python dynamique (type list)

Le type `list` python est dynamique. Il peut lui aussi implémenter une Liste chainée ou bien un Tableau.

Pour un Tableau, si on l'implémente avec le type `list`, la taille de celui-ci ne devra pas être modifiée à la fin du traitement. # La liste python dynamique (type list) Le type `list` python est dynamique. Il peut lui aussi implémenter une Liste chainée ou bien un Tableau.

Pour un Tableau, si on l'implémente avec le type `list`, la taille de celui-ci ne devra pas être modifiée à la fin du traitement.