Chapitre 2 - Les structures de données

Objectifs :

- ▷ Remettre en mémoire les structures de données vues en Première
- ▷ Montrer qu'il existe plusieurs implémentations possibles

1 Introduction

En informatique, on manipule essentiellement des données. Lorsqu'elles sont simples comme des nombres, des chaînes de caractères ou des booléens, on peut les stocker dans des variables qui seront typées en fonction de la nature de la donnée.

2 Définition

A retenir !

Une structure de données est une organisation logique des données permettant de simplifier ou d'accélérer leur traitement.

3 Exemple

Par exemple, si on a écrit un code avec les affectations suivantes :

```
n = 4
p = -5
r = 0.7
phrase = 'hello'
test = True
```

Python est un langage avec un **typage dynamique**, c'est-à-dire que le type de la variable est automatiquement déterminé la première fois qu'elle est déclarée.

Ainsi, on obtient en retour les types des variables suivants :

```
>>> type(n)
<class 'int'>
>>> type(p)
<class 'int'>
>>> type(r)
<class 'float'>
>>> type(phrase)
<class 'str'>
>>> type(test)
<class 'bool'>
```

En revanche, lorsque l'on a besoin de manipuler un grand nombre de données (du même type ou non), on utilise des **structures de données** comme les tuples, les *listes*, les *tableaux* ou les *dictionnaires*.

4 Utilisation d'une structure de données

Rappel

Les *chaînes de caractères*, les *tuples* et les *listes* sont des **séquences**, c'est-à-dire des suites ordonnées d'éléments indexés par une suite d'entiers.

On rappelle que :

- ▷ Les *chaînes de caractères* et les *tuples* sont **non mutables** (on ne peut les modifier).
- ▷ Les *listes* sont **mutables**.
- ▷ Un *tableau* est une liste de liste.
- ▷ Les *tuples* et les *listes* peuvent contenir des éléments de n'importe quel type, y compris d'autres tuples, listes ou encore des dictionnaires...
- ▷ Les *chaînes de caractères*, listes et tuples sont appelés des **types construits**. Ils possèdent un certain nombre de méthodes qui permettent d'agir sur l'objet (ajout, suppression, tris, ..., etc.)
- ▷ Dans un *dictionnaire* les éléments sont indexés par des clés et sont modifiables.

5 Application

On voudrait stocker dans une seule variable, pour un élève les données suivantes :

- ▷ Nom : Terieur
- ▷ Prénom : Alain
- ▷ Date de naissance : 01/01/2000
- ▷ Programmation : 12
- ▷ Algorithmique : 10
- ▷ Projet : 15

Pour ce faire, il y a plusieurs possibilités. On pourrait utiliser :

- **un tuple** (les données ne seront pas modifiables) :

```
1 eleve1 = ('Terieur', 'Alain', '01/01/2000', 12,10,15)
```

- un **tuple contenant des listes** (modifiables) :

```
1 eleve1 = (['Terieur', 'Alain', '01/01/2000'], [12,10,15])
```

- un **dictionnaire** :

```
1 eleve1 = {"Nom" : 'Terieur', 'Prenom' : 'Alain', 'Date' : '01/012000',
    'Programmation' : 12, 'Algorithmique' : 10, 'Projet' : 15}
```

Ou encore une liste, ou une liste de listes, ou une liste de tuples, ..., etc.

Si maintenant on souhaite, le faire pour une classe entière, on pourra stocker les données dans une variable :

```
classe1 = [eleve1,eleve2,...]
```

On peut créer une fonction qui calcule la moyenne d'un élève :

```
def moyenne(eleve : dict)-> float:
    ''' Fonction qui  calcule la moyenne(float) des notes
        Entrees :
            cles(type) -> Nom(str), Prenom(str), Date(str),
               Programmation(float), Algorithmique(float), Projet(float)
        Retour : moy(float)
    '''
    moy = (eleve['Programmation'] + eleve['Algorithmique'] +
       eleve['Projet']) / 3

    return moy

#---------- Programme principal -----------------

eleve1 = {'Nom' : 'Terieur', 'Prenom' : 'Alain', 'Date' : '01/012000',
   'Programmation' : 12, 'Algorithmique' : 10, 'Projet' : 15}

print('moyenne de ', eleve1['Prenom'], eleve1['Nom'], ' : ',
   moyenne(eleve1))
```

On obtient alors :

```
>> moyenne de Alain Terieur : 12.33333333333334
```

On pourra créer une variable contenant les moyennes de chaque élève, les moyennes de chaque matière et par exemple créer des classements suivant des critères.

6 Méthode

Le cahier des charges :

- ▷ Stocker les données (nom, prénom, date de naissance, notes) pour les élèves d'une classe.
- ▷ Calculer et stocker leur moyenne.
- ▷ Calculer et stocker la moyenne générale de chaque matière.

Choix de la structure de données :

- ▷ Pour un élève
- ▷ Pour la classe

Implémentation : Attention, votre implémentation ne doit pas modifier le cahier des charges ni le choix de la structure de données.

- ▷ Enregistrement des données
- ▷ Implémentation des fonctions

7 Conclusion

A retenir !

Il n'y a pas **unicité** de la représentation concrète des données, l'important ce sont les données pas leur représentation !

8 Exercices

Exercice 1 :

1. Implémenter le cahier des charges précédent pour une classe contenant les 3 élèves suivants :

Nom : Terieur Prénom : Alain Date : 01/01/2005 Programmation : 12 Algorithmique : 10 Projet : 15	Nom : Neymar Prénom : Jean Date : 01/07/2006 Programmation : 7 Algorithmique : 14 Projet : 11	Nom: Duff Nom: John Date : 01/11/2007 Programmation : 13 Algorithmique : 8 Projet : 17

Les données seront déjà structurées et notre classe de 3 élèves pourrait être enregistrée sous la forme d'un fichier `csv` comme suit :

```
Nom,Prenom,Date,Programmation,Algorithmique,Projet
Terieur,Alain,01/01/2005,12,10,15
Neymar,Jean,01/07/2006,7,14,11
Duff,John,01/11/2007,13,8,17
```

On peut alors récupérer ce fichier dans un programme, pour réaliser les calculs souhaités. On obtiendra comme précédemment une liste de dictionnaires contenant les données des élèves, cependant les notes seront de type `str`.

```
import csv

reader = csv.DictReader(open('eleves.csv', 'r'), encoding = 'utf-8')
classe = []

for row in reader:
    classe.append(dict(row))

print(classe)
```

2. Avec un éditeur de texte (par exemple *Notepad ++*), créer le fichier `eleves.csv` , puis réaliser un programme qui affiche les résultats suivants :

```
moyenne de Alain Terieur : 12.33333333334
moyenne de Jean Neymar : 10.66666666666
moyenne de John Duff : 12.66666666666
moyenne de programmation : 10.666666666
moyenne de algorithmique : 10.666666666
moyenne de Projet : 14.333333333333334
```

Vous devriez pouvoir utiliser les fonctions écrites à la question 1 en forçant les notes à être lues comme des nombres `float`.

Exercice 2 : *Ajouter un nombre*

Ecrire une fonction `ajout_nombre(L)` qui ajoute un entier à chaque élément d'une liste `L` La liste et le nombre à ajouter seront passés en paramètres.

Exemple :

```
> L = [1,2,3,4,5]
> ajout_nombre(3)
[4, 5, 6, 7, 8]
```

Exercice 3 : *Entier* $\rightarrowtail$ *Tuple*

Écrire une fonction `entier_tuple(n)` qui renvoie sous forme de tuple les chiffres d'un nombre entier positif `n` passé en paramètre.

Exemple :

```
> n = 123456
> entier_tuple(n)
(1,2,3,4,5,6)
```

Exercice 4 : *Tuple* $\rightarrowtail$ *Entier*

Écrire une fonction `tuple_entier(t)` qui renvoie sous forme d'entier le tuple `t` passé en paramètre.

Exemple :

```
> t = (1,2,3,4,5,6)
> tuple_entier(t)
123456
```

Exercice 5 : *Supression doublon*

Écrire une fonction `supprime_doublon(L)` qui supprime les éléments en double d'une liste `L` passée en paramètre.

Exemple :

```
> L = [1,2,5,8,6,2,5,9,1,8,8]
> supprime_doublon(L)
[1, 2, 5, 8, 6, 9]
```

Exercice 6 : *concaténation de listes*

Écrire une fonction `concatene_listes(L1,L2)` qui concatène sans doublons deux listes d'entiers `L1` et `L2` passées en paramètre.

Exemple :

```
> L1 = [13, 15, 12, 17, 15 ]
> L2 = [18, 15, 14, 13, 19, 20]
> concatene_listes(L1,L2)
[13, 15, 12, 17, 18, 14, 19, 20]
```

Exercice 7 : *D'une liste à un dictionnaire*

1. Ecrire en Python une fonction `liste_en_dico(liste)` qui admet en argument une liste de la forme `[clé1, valeur1, clé2, valeur2, ...]` et qui retourne le dictionnaire correspondant de la forme : `clé1 : valeur 1, clé2 : valeur2, ...`

   Vous pouvez tester votre programme avec la liste ci-dessous :

   ```
   Capitales = ['France', 'Paris', 'Angleterre', 'Londres', 'Espagne', 'Madrid', 'Allemagne',
   'Berlin']
   ```

2. Ecrire un **DocString** pour cette fonction.

Exercice 8 : *Fabricant de tee-shirts*

Un fabricant décide de créer des tee-shirts dont la taille peut-être : XS , S, M, L, XL, XXL.

A chaque taille son prix ; il adopte le principe suivant : 8€ pour la taille XS et il ajoute 2€ en passant à la taille supérieure, jusqu'au XXL.

1. Implémenter en Python, ces informations dans la structure de données la mieux adaptée.

   Ce même fabricant décide de changer sa façon de fixer les prix de revente des tee-shirts. Ceux dont la taille est XS sont toujours à 8 €, mais cette fois-ci, pour passer d'une taille à la suivante, il ajoute au prix de la taille inférieure la moitié de sa racine carrée.

   Par exemple, pour obtenir le prix de la taille S, il fait : $8 + \sqrt{8}/2$
2. Ecrire un fonction python `prix(liste)` qui admet en argument la liste des tailles et retourne les tailles et le prix dans une structure de données adaptée.