“Un dictionnaire est toujours très lourd à cause du poids des mots.”

Jérôme Duhamel

REPRESENTATION DES DONNEES :
Types construits

- *Listes et tableaux*
- *Tuples (p-uplets)*
- *Dictionnaires*

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Connaître des types construits pour représenter des données :
  - Savoir construire un p-uplet, une liste, un dictionnaire
  - Être capable de lire et de modifier des éléments d’un tableau
  - Maîtriser les itérations sur les éléments d’un tableau

1. Rappels et compléments sur les listes :

1.1 Définition

Une liste représente en mémoire un objet dans lequel on peut stocker plusieurs valeurs généralement de même type. Ce sont les items de la liste.

Une liste est une variable à part entière (de type *list*). On parle de structure de données ou bien d'objet conteneur : un objet qui en contient d'autres. Ce peut être des entiers, des réels, des listes, etc… L'utilisation des listes est très répandue en Python car celles-ci sont **modifiables** (**mutables**). Les listes sont définies entre 2 crochets et les items sont séparés par une virgule.

Exemple :

```
liste_ville = ["Clermont-Ferrand","Le Puy-en-Velay","Saint-Etienne","Lyon"]
liste_num = [15,20,36,14]
liste_liste = [[1,2],[2,1,3],[0,1]]
```

1.2 Création d'un liste

1.2.1 Liste vide

Pour créer une liste vide et pour ajouter des éléments par la suite, on utilise simplement les crochets vides

Exemple :

```
liste_vide = []
```

1.2.2 Liste par duplication

Si l'on veut créer une liste constituée d'un même élément ou d'une même séquence. On utilise l'opérateur Python * qui signifie duplication et non multiplication pour les variables de type liste.

Exemple 1 : on souhaite créer une liste de 10 items initialisé à 0.

```
liste_de_0 = [0]*10
```

Exemple 2 : on souhaite créer une liste de 9 items dont la séquence [1,2,3] se répète 3 fois.

```
liste_3_123 = [1,2,3]*3
```

1.2.3 Liste par compréhension

Il existe parfois un lien logique entre chaque item de la liste. Dans ce cas, on peut être plus efficace qu'une saisie manuelle en créant la liste par compréhension.

Exemple 1 : on souhaite créer une liste avec les nombres entiers de 6 à 99.

```
liste_entiers = [i for i in range(0,100,1) if (i>5)]
```

Exemple 2 : on souhaite créer une liste avec les carrés des nombres entiers de 0 à 15.

```
liste_entiers = [i**2 for i in range(0,16,1)]
```

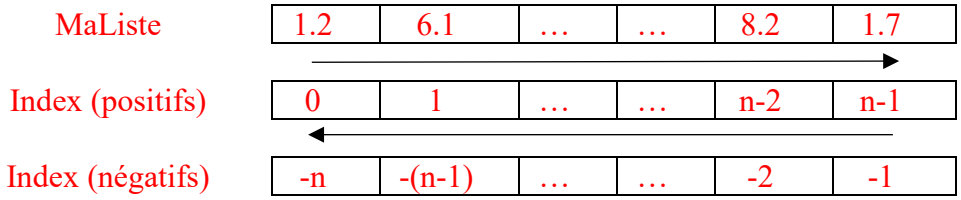
1.3 Lire et modifier les items d'une liste à l'aide de leurs index

On peut associer à chaque **item** de la liste un **index** qui peut prendre :

- une valeur positive si l'on parcours la liste de la gauche vers la droite. Le premier élément commence à l'index 0.
- une valeur négative si l'on parcours la liste de droite vers la gauche. Le premier élément commence à -1.

Exemple :

MaListe = [1.2, 6.1, …., 8.2, 1.7]



MaListe[0] vaut 1.2 tandis que MaListe[-1] vaut 1.7

Si on souhaite modifier la valeur d'un item, il suffit de le remplacer directement à l'aide de son index.

Exemple :

MaListe = [1, 2, 3]
MaListe[1] = 4

On a modifié le deuxième élément en remplaçant 2 par 4.

1.4 Opérations sur les listes

Opérateurs	Opérations effectuées
len(L)	Donner le nombre d'élément dans la liste L
L.append(e)	Ajouter l'élément e à la fin de la liste L
del L[i]	Supprimer l'item d'index i de la liste L
L.reverse()	Inverser les éléments de la liste L (le dernier élément sera le premier, etc.)
L.sort()	Trier la liste L dans l'ordre croissant ou alphabétique
L1 + L2	Concaténer (mettre bout à bout) les 2 listes (L1 et L2)

1.5 Manipulations diverses sur les listes

1.5.1 Transformer une liste en chaine de caractères

La méthode ***join*** permet de transformer une liste en chaine de caractères. Elle permet d'insérer, si besoin, un caractère entre chaque item.

Exemple :

MaListe = [10, 20, 30, 40, 50]
MaChaine = '-'.join(MaListe)

MaChaine vaut '10-20-30-40-50'

1.5.2 Tester la présence d'un item dans une liste

On peut tester la présence d'un item dans une liste avec l'opérateur ***in*** selon la syntaxe suivante : *x in MaListe*. Si *x* est présent dans *MaListe*, cette instruction retourne le booléen True sinon il retourne le booléen False.

Exemple :

MaListe = [10, 20, 30, 40, 50]
Presence = 30 in MaListe

Presence vaut True

1.5.3 Les parties de listes

On peut accéder à une partie d'une liste (MaListe) avec les index et le caractère « : » en utilisant la syntaxe MaListe[*index_debut_inclus* : *index_fin_exclus* : *pas*]

Si l'*index_debut_inclus* n'est pas renseigné, on prend le premier (suivant le sens de parcours des index) sinon il vaut -1. De même Si l'*index_fin_exclus* n'est pas renseigné, on prend le dernier (suivant le sens de parcours des index). Le *pas* vaut 1 par défaut si non renseigné.

Exemple :

MaListe = [10, 20, 30, 40, 50]

MaListe[:-1] = [10, 20, 30, 40]
MaListe[1:-1] = [20, 30, 40]
MaListe[::2] = [10, 30, 50]
MaListe[::-1] = [50, 40, 30, 20, 10]
MaListe[::-2] = [50, 30, 10]
MaListe[1:3] = [20, 30]

2. Les tuples (ou p-uplets) :

2.1 Définition

Un tuple est en fait une liste non modifiable.

Un tuple est aussi une variable à part entière (de type *tuple*). On déclare un tuple comme une liste, si ce n'est que les crochets sont remplacés par des parenthèses. Elles ne sont pas obligatoires mais rendent la lecture plus claire. Les éléments d'un tuple sont appelés les **arguments**.

Exemple :

MonPoint = ('A',1,-2)

On accède au contenu d'un tuple de la même manière qu'une liste avec les index.

Exemple :

MonPoint[1] vaut 1

2.2 Usage

On utilise très souvent les tuples comme résultats retournés par une fonction. Une fois crée, un tuple ne peut en aucune manière être modifié. Un tuple est **non-mutable**. Un argument ne peut donc pas être modifié ni supprimé. Le code est plus sûr si on « protège en écriture » les données qui n'ont pas besoin d'être modifiées.

Exemple :

Q,R = divmod(7,3)

La fonction divmod(7,3) renvoie un tuple (un couple) dont le premier argument est le quotient de la division euclidienne de 7 par 3 et le second étant le reste.

2.3 Transformer un tuple en liste et vice et versa

Les tuples peuvent être convertis en listes et vice-versa. La fonction prédéfinie ***tuple*** prends une liste et retourne un tuple contenant les mêmes éléments et la fonction ***list*** prends un tuple et retourne une liste. En fait, ***tuple*** gèle une liste et ***list*** dégèle un tuple.

Exemple 1 :

MonTuple = tuple(['A',1,-2])

Exemple 2 :

MaListe = list(('A',1,-2))

3. Les dictionnaires :

3.1 Définition

Un dictionnaire est une sorte de liste si ce n'est qu'à la place des index (nombre entier relatif), on utilise des clés (des valeurs qui peuvent être autre que des nombres entiers relatifs).

Un dictionnaire est délimité par des accolades et est **mutable**. Les informations ne sont pas stockées dans un ordre précis comme pour les listes et les tuples. Pour accéder aux éléments contenus dans le dictionnaire, on utilise les clés. Le dictionnaire est aussi une variable à part entière (de type *dict*). Les éléments contenus dans un dictionnaire sont appelés les **entrées**. Chaque entrée est constituée d'une clé et d'une valeur.

Exemple 1 : création d'un dictionnaire en une seule fois

MonDico = {'nom' : 'Toto', 'tel' : '06 12 34 56 78', 'age' : 29}

Exemple 2 : création d'un dictionnaire en créant d'abord un dictionnaire vide et en le remplissant au fur et à mesure

MonDico = {}
MonDico['nom'] = 'Toto'
MonDico['tel'] = '06 12 34 56 78'
MonDico['age'] = 29

3.2 Manipulations avec les dictionnaires

3.2.1 Récupérer une valeur

La méthode ***get*** permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, on peut donner une valeur à retourner par défaut.

Exemple :

MonDico = {'nom' : 'Toto', 'age' : 29}
Print(MonDico.get('nom')) *affichera 'Toto'*
Print(MonDico.get('adresse','Adresse inconnue')) *affichera 'Adresse inconnue'*

3.2.2 Tester la présence d'une clé

On peut tester la présence d'une entrée dans un dictionnaire avec l'opérateur ***in*** selon la syntaxe suivante : *cle in MonDico*. Si *cle* est présente dans *MonDico*, cette instruction retourne le booléen True sinon il retourne le booléen False.

Exemple :

MonDico = {'nom' : 'Toto', 'age' : 29}
Presence = 'nom' in MonDico

Presence vaut *True*

3.2.3 Supprimer une entrée

Pour supprimer une entrée, on utilise ***del*** comme pour les listes mais en indiquant la clé.

Exemple :

```
MonDico = {'nom' : 'Toto', 'age' : 29}
del MonDico['age']

MonDico vaut maintenant  {'nom' : 'Toto'}
```

3.2.4 Accéder à toutes les clés, ou à toutes les valeurs, ou à tout le contenu

La méthode ***keys*** permet d'accéder à toutes les clés d'un dictionnaire. Attention le résultat n'est pas récupérable dans une variable !

Exemple :

```
MonDico = {'nom' : 'Toto', 'age' : 29}
for cle in MonDico.keys() :
    print(cle)
```

La méthode ***values*** permet de récupérer toutes les valeurs d'un dictionnaire. Attention le résultat n'est pas récupérable dans une variable !

Exemple :

```
MonDico = {'nom' : 'Toto', 'age' : 29}
for val in MonDico.values() :
    print(val)
```

La méthode ***items*** permet de récupérer tout le contenu d'un dictionnaire. Attention le résultat n'est pas récupérable dans une variable !

Exemple :

```
MonDico = {'nom' : 'Toto', 'age' : 29}
for item in MonDico.items() :
    print(item)
```

4. Les tableaux et matrices

4.1 Définition

Un tableau ou une table est une liste où les items peuvent être des listes, des tuples ou des dictionnaires. Si ce tableau ne contient que des entiers ou des réels, on parle de matrices.

Exemple 1:

MonTableau = [[1,2,3] , [4,5,6]]

C'est la représentation du tableau suivant :

1ére liste	1	2	3
2de liste	4	5	6

Ce tableau correspond également à la matrice suivante $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

Exemple 2:

MonTableau = [{'nom' : 'Jojo', 'age' : 18} , {'nom' : 'Bébert', 'age' : 21}]

C'est la représentation du tableau suivant :

	'nom'	'age'
1er dictionnaire	'Jojo'	18
2d dictionnaire	'Bébert'	21

4.2 Accès aux éléments

Pour accéder aux éléments d'un **tableau** constitué d'une **liste de listes ou de tuples**, on utilise les index. Par exemple ***MonTableau[i][j]*** donne accès à l'élément situé sur **la (*i+1*)ème ligne** et sur **la (*j+1*)ème colonne**

Exemple :

MonTableau = [[1,2,3] , [4,5,6]]

MonTableau[1][2] donne bien accès à l'élément situé sur la 2ème ligne et sur la 3ème colonne. Il vaut donc 6

Pour accéder aux éléments d'un **tableau** constitué d'une **liste de dictionnaires**, on utilise les index et les clés. Par exemple ***MonTableau[i]['clé']*** donne accès à l'élément situé sur **la (*i+1*)ème ligne** et dont **la colonne s'intitule 'clé'**

Exemple :

MonTableau = [{'nom' : 'Jojo', 'age' : 18} , {'nom' : 'Bébert', 'age' : 21}]

MonTableau[1]['age'] donne bien accès à l'élément situé sur la 2ème ligne et dont la colonne s'intitule 'age'. Il vaut donc 21

Remarques :

Toutes les actions associées aux listes, tuples et dictionnaires sont bien évidemment utilisable dans les tableaux.