Informatique : Dictionnaires

PSI : Lycée Rabelais

Pré-requis

Cours d'informatique de première année : types `string`, `tuple`, `list`...

Objectifs

Savoir construire un dictionnaire sous `Python` : clés, valeurs, principe du hachage et collisions.

Savoir utiliser un dictionnaire : syntaxe pour l'écriture des dictionnaires ; parcours d'un dictionnaire.

1 Qu'est ce qu'un dictionnaire ?

1.1 Structure d'un dictionnaire

Un dictionnaire `Python` est un objet de type construit, contenant des valeurs indexées par des clés. Contrairement aux listes, aux chaînes de caractères ou aux tuples, indexés par une liste d'entiers, un dictionnaire accepte des clés de différents types pourvus qu'ils soient immuables.

Type mutable, type immuable (ou non mutable)

Un type `Python` est mutable lorsqu'il supporte une opération d'affectation : ainsi les listes sont mutables (`L[1]=0` est une opération autorisée). Un type non mutable, ou immuable, n'admet pas cette opération : c'est le cas des chaînes de caractères (type `str`) et des tuples (type `tuple`).

Un dictionnaire peut donc être vu comme un tableau à deux colonnes : la première colonne contient les *clés*, la deuxième colonne contient les *valeurs*. L'ensemble (clé + valeur) constitue donc un élément du dictionnaire.

1.2 Création d'un dictionnaire

1.2.1 Initialisation d'un dictionnaire

Soit `C` l'ensemble des clés `{c1,c2,c3,...}` d'un dictionnaire, et `V` l'ensemble des valeurs `{v1,v2,v3,...}` associées. Le dictionnaire `d` s'écrit : `d = {c1:v1,c2:v2,c3:v3,...}`

Remarque : les éléments d'un dictionnaire ne sont pas nécessairement ordonnés.

Exemple de création d'un dictionnaire On considère la liste des notes d'informatique d'une classe, présentée sous la forme d'un tableau `notes` :

Prénom	Nom	DS1	DS2
Pierre	Cesdeuhel	12.0	11.0
Marquise	Demerteuil	16.0	18.0
Vicomte	Devalmont	16.0	9.0
Madame	Detourvel	8.0	8.0
Cecile	Volanges	4.0	17.0

Tableau `notes`

On souhaite obtenir les notes d'un élève à l'aide d'une instruction de la forme `notes['Nom de l'élève']['Devoir']`. Les indices de l'objet `notes` sont `'Nom de l'élève'` et `'Nom de l'élève'`. Ce sont des chaînes de caractères : `notes` doit donc être un dictionnaire.

Création du dictionnaire `notes` : On rappelle que les clés ne peuvent pas être mutables. On choisit ici pour clés des chaînes de caractères "Prénom Nom". Les valeurs associées sont de type quelconque, laissé au choix de l'utilisateur. Deux exemples sont présentés ci-dessous :

Exemple 1 : les valeurs sont des listes On crée le dictionnaire à l'aide de l'instruction suivante :

```
...
...
...
...
```

- Les notes de 'Marquise Demerteuil' sont obtenues par l'instruction :
- Si l'on désire obtenir uniquement sa note au DS1, on écrira :

Exemple 2 : les valeurs sont des dictionnaires L'exemple précédent nécessite la connaissance de l'ordre dans lequel les éléments de la liste ont été rentrés : `DS1` correspond au premier élément, `DS2` au second. Pour rendre les instructions plus lisibles, on utilise des valeurs de type dictionnaire elles-mêmes :

```
...
...
...
...
```

- Les notes de 'Marquise Demerteuil' sont obtenues par l'instruction :
- Si l'on désire obtenir uniquement sa note au DS1, on écrira :

Remarque : si `notes` avait été écrite sous forme d'une liste de listes :

```
notes = [['Pierre','Cesdeuhel',12.0,11.0],
['Marquise','Demerteuil',16.0,18.0],
['Vicomte','Devalmont',16.0,9.0],
['Madame','Detourvel',8.0,8.0]
'Cecile','Volanges',4.0,17.0]]
```

Les instructions précédentes deviendraient alors `notes[1]` et `notes[1][1]`. Cette notation est moins explicite (bien qu'utilisable !) que celle obtenue avec un dictionnaire.

1.2.2 Opérations autorisées sur un dictionnaire

- créer un dictionnaire vide : `d = {}`
- associer une valeur `v` à une clé `c` dans le dictionnaire `d` : `d[c]=v`. Si `c` n'existe pas, cette opération devient une opération d'insertion : un nouvel élément `(c,v)` est inséré dans `d`
- renvoyer la valeur associée à la clé `c` : `d[c]`
- supprimer l'entrée correspondant à `c` dans `d` : `del d[c]`. Cette opération déclenche une erreur si la clé `c` n'est pas dans `d`. On peut également utiliser `d.pop(c)`, qui renvoie en plus l'élément supprimé.
- obtenir l'ensemble des clés du dictionnaire `d` : `d.keys()`.

On peut ainsi vérifier l'existence d'une clé en parcourant l'ensemble des clés d'un dictionnaire `d` (instruction `for c in d :`), mais aussi modifier les valeurs associées à ces clés : un dictionnaire est donc un type mutable et itérable, comme les listes.

2 Nécessité d'un dictionnaire

2.1 Contexte

On possède un fichier `pays_langues.csv` qui contient un tableau donnant les pays du monde avec quelques unes de leurs caractéristiques : code international ISO, population, superficie... Les premières lignes du tableau sont les suivantes :

ISO	Country	Capital	Area(in sq km)	Population	Continent	Currency	Languages	Neighbours
AD	Andorra	Andorra la Vella	468	77006	EU	Euro	ca	ES,FR
AE	United Arab Emirates	Abu Dhabi	82880	9630959	AS	Dirham	ar-AE, fa,en,hi,ur	SA,OM
AF	Afghanistan	Kabul	647500	37172386	AS	Afghani	fa-AF, ps,uz-AF,tk	TM,CN,IR, TJ,PK,UZ
AG	Antigua and Barbuda	St. John's	443	96286	NA	Dollar	en-AG	
...	...	...	...	...	...	...	...	...

On cherche à manipuler ces données à l'aide d'un script `Python`. Pour cela, il est nécessaire de les extraire afin que `Python` puisse les interpréter. Nous cherchons par exemple à obtenir les informations concernant la France sous la forme d'un objet manipulable par `Python`. Pour cela, on peut utiliser entre autres une liste ou un dictionnaire.

2.2 Cas 1 : manipulation de listes

On commence par extraire les données du fichier `pays_langues.csv` à l'aide des instructions suivantes :

```
# ==================================
# Ouverture du fichier pays_langues.csv
# ==================================

fic=...                                  # Ouverture du fichier en lecture
texte=...                                # Lecture de l'ensemble des lignes
fic.close() # Fermeture du fichier

pays=[] # pays est une liste de listes contenant l'ensemble des
    informations sur chaque pays

for ..............  in ..............    :
    ...
    ...
    ...
```

L'instruction `pays[0]` renvoie :

```
['ISO',
 'Country',
 'Capital',
 'Area(in sq km)',
 'Population',
 'Continent',
 'Currency',
 'Languages',
 'neighbours']
```

Pour obtenir les informations relatives à la France, il faut écrire `pays[75]` dans la fenêtre de commandes. On obtient alors :

```
['FR',
 'France',
 'Paris',
 '547030',
 '66987244',
 'EU',
 'Euro',
 'fr-FR,frp,br,co,ca,eu,oc',
 'CH,DE,BE,LU,IT,AD,MC,ES']
```

On comprend alors qu'un tel code rend difficile la manipulation des données du fichier `pays_langues.csv`. En effet, si l'on cherche cette fois les informations concernant l'Italie, il faudra dans un premier temps trouver son indice dans la liste `pays`, puis faire manuellement la correspondance du tableau renvoyé avec les éléments de la liste `pays[0]`...

Il serait plus agréable de pouvoir utiliser une instruction de la forme `pays['France']` ou `pays['Italie']` afin de récupérer les informations sur ces pays. Le formalisme des listes ne le permet cependant pas : nous allons donc passer par un dictionnaire.

2.3 Cas 2 : utilisation d'un dictionnaire

Exemple de construction d'un dictionnaire On reprend le tableau de l'exemple précédent. Le dictionnaire `Andorre` qui serait rattaché aux caractéristiques du pays `'Andorra'` s'écrit :

```
Andorre = {'ISO':'AD', 'Country' : 'Andorra', 'Capital' : 'Andorra la
    Vella', 'Area(in sq km)' : 468, 'Population' : 77006, 'Continent':'EU'
    , 'Currency':'Euro', 'Languages':'ca', 'Neighbours':'ES,FR'}
```

On remarquera la construction `d={c:v}` qui associe la valeur `v` à la clé `c`.

Question 1. Ecrire de même le dictionnaire `Emirats Arabes Unis` qui contient les éléments propres au pays nommé *United Arab Emirates* dans le tableau.

Question 2. Quelle instruction écrire pour générer un dictionnaire `pays_partiel` contenant les dictionnaires `Andorre` et `Emirats Arabes Unis` ?

Question 3. Ecrire les instructions nécessaires pour retrouver, à partir de `pays_partiel` :

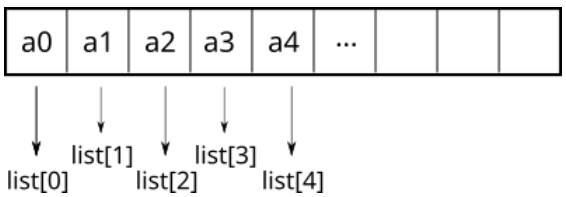
1. l'ensemble des informations connues sur Andorre ;
2. la capitale d'Andorre ;
3. la population des Emirats Arabes Unis

Question 4. Que renvoie l'instruction `pays_partiel[0]` ?

3 Gestion du dictionnaire : notion de hachage

3.1 Rappel : adressage des éléments d'une liste

Lorsqu'une liste `list` est créée, son premier élément `list[0]` possède une certaine adresse mémoire : soit `a0` cette adresse. L'adresse `a1` de l'élément suivant, `list[1]`, est calculée en ajoutant à `a0` une taille `t` fixée (souvent 8 ou 4 octets). On poursuit ainsi pour tous les éléments de la liste : `list[n]` a donc pour adresse `a0 + n × t`. En connaissant l'adresse du premier élément, on en déduit l'adresse de n'importe quel élément de la liste, et donc la valeur de l'élément qui s'y trouve.



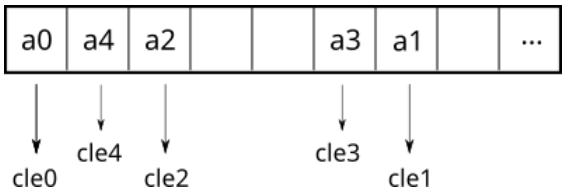
3.2 Stockage des éléments d'un dictionnaire

Contrairement à une liste, il n'existe pas de relation d'ordre entre les éléments d'un dictionnaire. Pour implémenter un dictionnaire `Python`, il faut adresser chaque clé à l'aide d'une **table de hachage**. Pour cela, on dispose d'une **fonction de hachage** qui à chaque clé associe un nombre entier appelé **valeur de hachage** de la clé. Ce nombre entier sert à calculer un indice utilisé pour stocker chaque couple (clé,valeur) dans un tableau. La table

de hachage est donc un tableau de tableaux. Il suffit de parcourir ce nouveau tableau pour obtenir les clés et les valeurs associées, et ainsi lire le dictionnaire.

La fonction de hachage doit nécessairement être bijective : deux éléments distincts du dictionnaire doivent avoir des adresses distinctes. C'est pour cela que les clés sont des objets immuables : si elle était mutable, la modification de la clé entraînerait un changement d'adresse et rendrait difficile la gestion du stockage.

`Python` dispose d'une fonction pour calculer la valeur de hachage à partir d'un objet immuable `cle` : `hash(cle)`. Si `t` est la taille de la table de hachage, les indices de chaque clé seront calculés par l'opération `hash(cle)% t`. La réduction modulo `t` assure de tomber sur une case valide de la table.



Propriétés de la fonction `hash` La fonction `hash` est construite de sorte à être efficace :

- si `i` est un entier différent de -1 et tel que $-2^{61}+1 <$ `cle` $< 2^{61}-1$, alors `hash(i)` vaut `i` ;
- `hash`$(-2^{61}+1)$ et `hash`$(2^{61}-1)$ valent 0, `hash(-1)` vaut -2 (utilisé pour la suppression de clé) ;
- si `x` est un flottant égal à $\frac{p}{q}$ (p et q entiers) alors `hash(x)` vaut `(int(p*M/q))%M` avec $M = 2^{61}-1$;
- si `cle` est une chaîne de caractères, la valeur de hachage est calculée avec une part de hasard à chaque nouvelle session et a pour valeur un entier qui s'écrit sur 64 bits. C'est aussi le cas pour un type composé.

Exemple : On considère le dictionnaire `d={97:"a",101:"e",114:"r",111:"o"}` et un tableau `T` de huit octets supposé être la capacité du dictionnaire. `d` ne contient que quatre couples (clé, valeur) : il peut encore en stocker quatre autres.

On note `p` la position de la clé `c` dans le tableau `T`, calculée par la fonction de hashage : `p=hash(c)%8`. Si la clé `c` se trouve à la k-ème position dans `d`, alors : `T[p] = k`. On obtient ainsi le tableau de correspondance suivant pour le dictionnaire `d` défini précédemment :

Clé c	Indices dans d	p=hash(c)%8	T[p]

• Sachant qu'une entrée vide est codée par le nombre 255 (ou -1), le tableau `T` s'écrit : ...

En cas de suppression d'une clé, $T[1]$ devient égal à -2 (ou 254) et dans le tableau des données, 97 devient 0.

Accès à une clé Pour accéder à une clé, il faut lire le tableau `T` l'élément correspondant à l'indice calculé par la fonction de hashage. Par exemple, pour la clé 101, il faut lire `T[5]`, qui renvoie 1. On lit dans `d` que le deuxième élément (position 1 du dictionnaire) est `"e"`.

3.3 Collisions

3.3.1 Présentation des collisions

La fonction `hash` ne prend en compte que les valeurs des objets appelés. Ainsi, deux objets `a = [1,2]` et `b = [1,2]` sont différents pour `Python` (adresses mémoires différentes), mais `hash(a)` et `hash(b)` sont identiques. De même, 0 et $2^{61}-1$ ont la même valeur de hachage.

Ceci est problématique : deux objets différents seront référencés de la même manière dans un dictionnaire. Lors de la construction du dictionnaire, seul le dernier élément sera conservé, les autres de même valeur de hachage seront écrasés. En effet, considérons le dictionnaire précédent, dont la clé `101` est désormais la clé `105` : `d2={97:"a",105:"e",114:"r",111:"o"}`. Nous cherchons à générer le tableau `T` de leur adressage en mémoire. On définit pour cela la fonction `dic(couples)` qui a un ensemble de couples (clé,val) renvoie le tableau `T` de leur associé à leurs valeurs de hachage :

```
def dic(dico):
    liste=8*[None]
    for c in dico.keys():
        couple=(c,dico[c])
        liste[hash(c)%8] = couple
    return liste

d = {97:"a",101:"e",114:"r",111:"o"}
d2 = {97:"a",105:"e",114:"r",111:"o"}
print(dic(d),dic(d2))
```

Le résultat obtenu est `[None, (97, 'a'), (114, 'r'), None, None, (101, 'e'), None, (111, 'o')]` et `[None, (105, 'e'), (114, 'r'), None, None, None, None, (111, 'o')]`.

- **Justification :**

Le phénomène observé pour le dictionnaire `d2` s'appelle **une collision**. Il est problématique car entraîne une perte de données, et doit donc être traité. Nous présentons ici deux manières de gérer les collisions : par adressage ouvert, ou par adressage fermé.

3.4 Gestion des collisions

Résolution par chaînage ou adressage ouvert : si plusieurs éléments ont le même indice dans `T` suite au hachage, on peut demander à créer une liste dont le premier élément est indicé par l'indice issu du hachage. Les autres éléments le suivent, comme dans une liste. On consomme alors de la place en mémoire car plusieurs éléments se trouvent dans une même alvéole de `T`

Exemple : dans l'exemple précédent, cela reviendrait à : `[None, [(97,'a'),(105, 'e')], (114, 'r'), None, None, None, None, (111, 'o')]`. La liste `[(97,'a'),(105, 'e')]` est une liste chaînée.

Résolution par adressage fermé : on calcule l'indice de la première place libre qui suit la place dont l'indice est obtenu par la fonction `hash`. Cela revient à calculer `(hash(cle)+i)%(n)`, avec n la longueur du dictionnaire (8

dans les cas précédents) et $i = 0, 1, 2, \ldots$. Le coût en temps est plus important qu'en adressage ouvert, mais il ne nécessite pas d'espace mémoire supplémentaire.

Exemple : dans l'exemple précédent, le dictionnaire est de taille 8. On appelle `h` la valeur de `hash(cle)`. En cas de collision à la place i, la nouvelle place est calculée par la relation `i = (5*i+1)%8`

Vérification : dans notre cas, `T[1]` est déjà occupé par (97,'a'). On pose $i = 1$. Les valeurs successives de i prises avec la relation précédente sont 6, 7, 4, 5, 2, 3 et 0. On balaie bien tous les indices possibles de `T` : la première entrée nulle sera alors remplacée par `(105, 'e')`, soit dans notre cas l'entrée d'indice 6.

```
def dicmodif(dico):
    liste=8*[None]
    for c in dico.keys():
        couple=(c,dico[c])
        i=hash(c)%8
        while liste[i] is not None :
            i=(5*i+1)%8
        liste[i] = couple
    return liste

d = {97:"a",101:"e",114:"r",111:"o"}
d2 = {97:"a",105:"e",114:"r",111:"o"}
print(dicmodif(d),dicmodif(d2))
```

Comme attendu, ce programme renvoie `[None, (97, 'a'), (114, 'r'), None, None, (101, 'e'), None, (111, 'o')]` et `[None, (97, 'a'), (114, 'r'), None, None, None, (105, 'e'), (111, 'o')]`