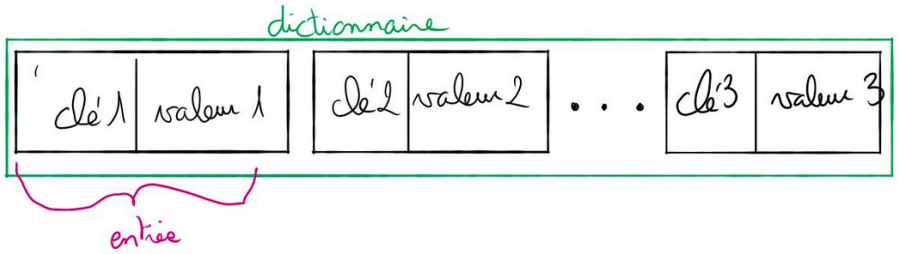
NSI Terminale	Cours/TP - Les dictionnaires	Séquence 3

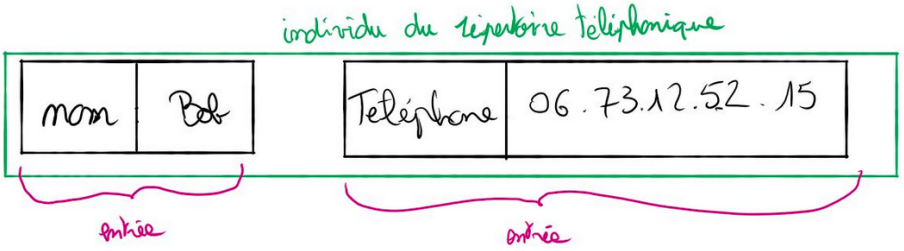
I – Introduction : approche de première

Nous allons étudier un autre type abstrait de données : les dictionnaires. Ils ont déjà été étudiés en classe de première. Pour rappel, ce type de données, aussi appelé *tableau associatif*, permet de stocker des **valeurs** et d'y accéder au moyen d'une **clé**, contrairement au tableau qui permet d'accéder à une donnée au moyen d'un **indice**. Ils sont utilisés pour des applications où la recherche d'une valeur est prioritaire et doit se faire le plus rapidement possible.

Définition :
Un dictionnaire est une structure de données qui permet d'associer une valeur à une clé. Cette clé peut être un mot ou un entier. L'ensemble clé-valeur est appelé entrée.



Ex :
On peut modéliser un individu d'un répertoire téléphonique avec un dictionnaire. On pourrait prendre comme clés : nom et téléphone.



nom et *Téléphone* sont des clés ; *Bob* et *06 73 12 52 15* sont des valeurs.

Voici l'utilisation en python :

```
dico = dict()
dico["Nom"] = 'Bob'
dico["Téléphone"] = '06.73.12.52.15'
print(dico)
```

donne :

```
{'Nom': 'Bob', 'Téléphone': '06.73.12.52.15'}
```

Exercice 1 :
Imaginons que l'on veuille stocker les données correspondantes à ses amis : le nom, l'âge et la commune. On choisi comme clé le nom et comme valeur une liste [age, commune].

1) Créer un dictionnaire avec les données de trois amis.
2) Afficher toutes les clés du dictionnaire.
3) Afficher toutes les valeurs du dictionnaire.
4) Afficher toutes les couples clés : valeurs du dictionnaire.
5) Afficher vos données sous cette forme : « Nom : Lucas Age : 17 ans Commune : Brunoy »

Y. Coudert - Lycée Le Bon Sauveur Le Vésinet – Terminale spécialité NSI
D'après un document collectif :de S. GUILLAUD et C. ROBLAIN – Lycée Talma

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

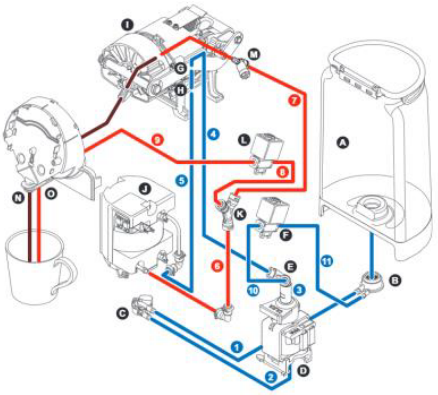
.....

II – Interface et implémentation

Rappel : différence INTERFACE et IMPLEMENTATION

Les types abstraits sont définies par leur **INTERFACE** (comment on s'en sert) et non par leur **IMPLÉMENTATION** (comment ils fonctionnent).

On peut prendre l'image d'une machine à café à capsule, l'utilisateur sait utiliser son interface (boutons, voyants, levier…) mais il n'est pas utile qu'il connaisse l'existence des électrovannes, de la pompe et du bloc de chauffe ni même le déroulement des opérations permettant de faire couler son petit expresso du matin.



1) Interface :

Voici les opérations primitives que l'on a généralement sur le type abstrait « dictionnaire » :

- `ajouter : Dict, [cle, valeur]  Rien`

`ajouter(dico, [key, value])` ajoute le couple `(key :value)` dans le dictionnaire `dico`.

- `modifier : Dict,[cle, valeur]  Rien`

`modifier(dico, [key, value])` modifie le couple (clé:valeur) en remplaçant la valeur associée à la clé `key` par une autre valeur `value` (la clé `key` restant identique) dans le dictionnaire `dico`.

- `supprimer : Dict, cle  Rien`

`supprimer(dico, key)` supprime la clé `key` (et donc aussi la valeur qui lui est associée) dans le dictionnaire `dico`

- `rechercher : Dict, cle  valeur`

`rechercher(dico, key)` renvoie la valeur associée à la clé `key` dans le dictionnaire `dico`

Exercice 2 :

Soit le dictionnaire D composé des couples `clé:valeur` suivants

⇨ `prenom:Kevin, nom:Durand, date-naissance:17-05-2005`

Pour chaque exemple ci-dessous, on repart du dictionnaire d'origine :

- `ajouter(D,tel:06060606)` ; le dictionnaire D est maintenant composé des couples suivants :

→ `prenom:Kevin, nom:Durand, date-naissance:17-05-2005, tel:06060606`

- `modifier(D,nom:Dupont)` ; le dictionnaire D est maintenant composé des couples suivants :

→ `prenom:Kevin, nom:Dupont, date-naissance:17-05-2005, tel:06060606`

- `supprimer(D,date-naissance)` ; le dictionnaire D est maintenant composé des couples suivants :

→ `prenom:Kevin, nom:Dupont, tel:06060606`

- `rechercher(D,prenom)` ; la fonction renvoie Kevin

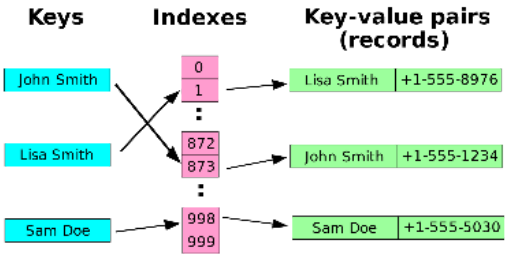
La recherche dans un dictionnaire est optimisée pour s'effectuer **sur les clés** et **non sur les valeurs**. Par exemple avec le dictionnaire que nous avons créé précédemment, la commande `"nom" in dico` renverra `True` alors que `"Durand" in dico` renverra `False`.

Dans un dictionnaire, les *clés* et les *valeurs* ne jouent donc pas du tout le même rôle et ne sont pas interchangeables.

2) Implémentations :

Plusieurs implémentations sont possibles :

- L'implémentation que vous connaissez déjà est le type construit '**dict**' de python qui est une implémentation avec ce qu'on appelle une table de hachage (voir ci-dessous pour des compléments hors programme).
- Nous verrons en TD/TP une implémentation en **POO**.



III – Les clés

Une clé peut être d'un autre type que chaîne de caractère, du moment que c'est un **objet non mutable**, c'est à dire qui ne peut pas être modifié. Une clé ne **peut pas être une liste** par exemple car une liste est un objet mutable que l'on peut modifier, par exemple au travers de la méthode **.append()**.

Ex : une clé peut être un tuple

```
>>> dico[(1,2)] = 'ça marche'
>>> print(dico)
  {(1, 2): 'ça marche'}
```


Y. Coudert - Lycée Le Bon Sauveur Le Vésinet – Terminale spécialité NSI
D'après un document collectif :de S. GUILLAUD et C. ROBLAIN – Lycée Talma

Regardons ce qui se passe si on essaie de définir une clé de type **list** pour un dictionnaire :

```
>>> dico[[1,2]] = 'ça ne marche pas'
  Traceback (most recent call last):
    File "<pyshell>", line 1, in <module>
  TypeError: unhashable type: 'list'
```

Le type **list** n'est pas pas *hashable*. Mais qu'est-ce que le hachage ?

Compléments (hors programme) : Notion de hachage

La notion de **Hachage** est omniprésente en informatique et est au coeur du fonctionnement des dictionnaires. Le hachage est un mécanisme permettant de transformer la clé en un nombre permettant l'accès à la donnée, un peu à la manière d'un indice dans un tableau.
Les tables de hachages ainsi que les fonctions de hachages qui sont utilisées pour construire les tables de hachages, ne sont pas au programme de NSI. Cependant, l'utilisation des fonctions de hachages est omniprésente en informatique, il serait donc bon, pour votre "culture générale informatique", de connaitre le principe des fonctions de hachages.

a) Définition d'une fonction de hachage :

Une fonction de hachage est une fonction qui va calculer une empreinte unique à partir de la donnée fournie en entrée. Elle doit respecter les règles suivantes :

- La longueur de l'empreinte (valeur retournée par la fonction de hachage) doit être toujours la même, indépendamment de la donnée fournie en entrée.
- Connaissant l'empreinte, il ne doit pas être possible de reconstituer la donnée d'origine.
- Des données différentes doivent donner *dans la mesure du possible* des empreintes différentes.
- Des données identiques doivent donner des empreintes identiques.

b) Quelques utilisations du hachage :

L'utilisation la plus courante est le stockage des mots de passe dans un système informatique un peu sécurisé. En effet, lorsqu'on crée un compte sur un service en ligne, le mot de passe ne **doit pas être stocké en clair**, une empreinte est générée afin que si le service est piraté et que les comptes sont dérobés, il ne soit pas possible de reconstituer le mot de passe à partir de l'empreinte. Voici un exemple de fonctionnement d'une fonction de hachage. Nous utiliserons le hachage **MD5** accessible sous *Linux* au travers de la fonction md5sum.
(Utiliser le lien https://bellard.org/jslinux/vm.html?cpu=riscv64&url=fedora29-riscv-2.cfg&mem=256 pour obtenir un environnement linux).

```
$ echo Lecluse | md5sum
4dc334bdaa9047f6a86a2f002308b945  -

$ echo Lécluse | md5sum
da9586508c85a8fc1f385c28f477cfe3  -

$ echo "Olivier Lecluse" | md5sum
7136fe2a82df094189d74c82f9ed1c6e  -
```

On constate bien sur cet exemple que :

- toutes les empreintes ont la même longueur
- un petit changement dans la valeur d'entrée fournit une empreinte totalement différente.

Exercice 3 : en Python
Taper le programme suivant et coder NSI et nsi. Que remarquez-vous ?

```
import hashlib      # interface pour algorithme de hachage

def hashing(text):
    m = hashlib.md5()   # choix du type de hachage (sha224, md5,...)
    text = text.encode('utf-8')  # avant de hacher le texte, il faut l'encoder
                                 # pour indiquer le type de caractères utilisés
    m.update(text)     # on effectue le type de hachage choisi
    return m.hexdigest()       # on renvoie la valeur du texte après transformation
                               # en hexadécimaux

texte = input("Entrez votre texte : ")
print(hashing(texte))
```

Le Hachage change en fonction des caractères

Exercice 4 :

Une autre utilisation du hachage est la détection de la modification d'un fichier.

1. Créer avec un éditeur de texte un texte de plusieurs paragraphes (vous pouvez générer un contenu quelconque sur internet grâce au site https://fr.lipsum.com/). Sauvegarder ce texte dans un fichier nommé `lorem.txt`.
2. Compléter le programme précédent avec les instructions suivantes et le lancer.

```
T = open('lorem.txt','r')
print(T.read())
print()
print(hashing(T.read()))
```

3. Modifiez très légèrement votre fichier original en modifiant une majuscule par exemple. Recalculer le hash MD5. Que constatez-vous ?

Le MD5 a changé.

Ainsi la fonction de hachage peut mettre en évidence des différences qui seraient invisibles à l'oeil nu.

c) Table de hachage :

Regardez la vidéo ci-dessous sur les tables de hachage (video1 8 :17) :
https://www.youtube.com/watch?v=IhJo8sXLfVw&feature=youtu.be

Ce qui est important à retenir c'est que la recherche dans une table de hachage **est indépendante du nombre d'éléments dans cette table**, l'algorithme de recherche dans une table de hachage a ainsi une complexité O(1). Dans une liste chaînée au contraire, la recherche d'un élément prend un temps **proportionnel** au nombre d'éléments dans la structure, soit une complexité de l'algorithme O(n).

Dans un dictionnaire, les clés sont stockées dans une table de hachage, ce qui explique le fait que le dictionnaire est optimisé pour la recherche sur les clés.

d) Utilisation des dictionnaires en Python :

Vous pouvez à présent regarder la vidéo suivante (vidéo2) afin de vous réviser la manipulation des dictionnaires en python avant de passer aux travaux pratiques.
https://www.youtube.com/watch?v=VnhBoQAgIVs&feature=youtu.be