Classe : terminale	LYCÉE INTERNATIONAL COURS LUMIÈRE
séquence 2	STRUCTURES DE DONNEES : LES DICTIONNAIRES
Mr BANANKO K.	

Objectif :

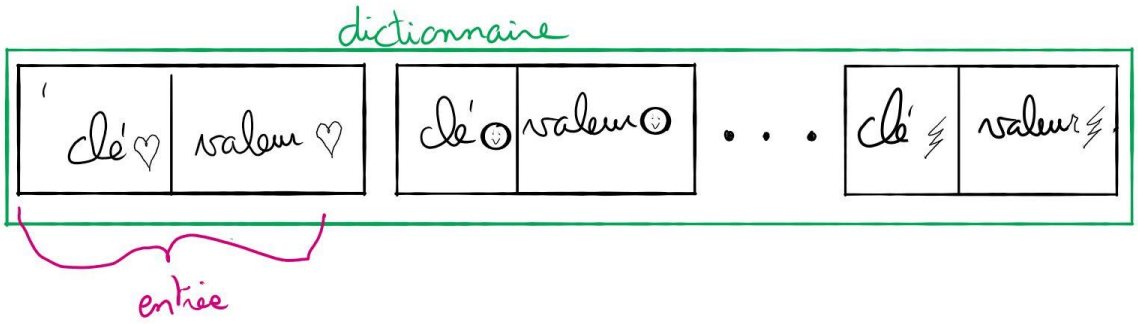
Choisir une structure de données adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.

I- Les dictionnaires

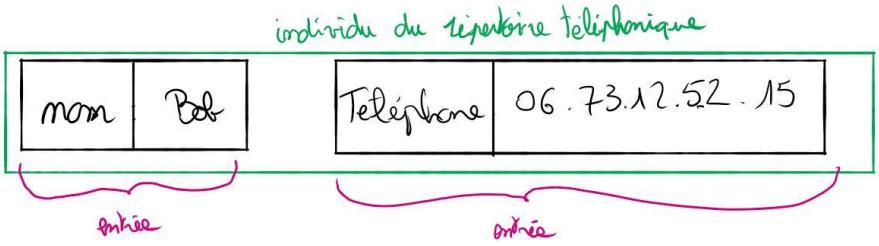
Nous allons maintenant étudier un autre type abstrait de données : les dictionnaires aussi appelés tableau associatif.

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.**

On retrouve une structure qui ressemble, à première vue, beaucoup à un tableau (à chaque élément on associe un indice de position). Mais au lieu d'associer chaque élément à un indice de position, dans un dictionnaire, on associe chaque élément (on parle de valeur dans un dictionnaire) à une clef, on dit qu'un dictionnaire contient des couples clé:valeur (chaque clé est associée à une valeur).



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



Exemples de couples

clé:valeur => prenom:Kevin, nom:Durand, date-naissance:17-05-2005.

prenom, nom et date sont des clés ; Kevin, Durand et 17-05-2005 sont des valeurs.

Voici les opérations que l'on peut effectuer sur le type abstrait dictionnaire :

- **ajout** : on associe une nouvelle valeur à une nouvelle clé
- **modif** : on modifie un couple clé:valeur en remplaçant la valeur courante par une autre valeur (la clé restant identique)
- **suppr** : on supprime une clé (et donc la valeur qui lui est associée)
- **rech** : on recherche une valeur à l'aide de la clé associée à cette valeur.

Exemples :

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 :

- `ajout(D,tel:06060606)` ; le dictionnaire D est maintenant composé des couples suivants : prenom:Kevin, nom:Durand, date-naissance:17-05-2005, tel:06060606
- `modif(D,nom:Dupont)` ; le dictionnaire D est maintenant composé des couples suivants : `prenom:Kevin, nom:Dupont, date-naissance:17-05-2005`
- `suppr(D,date-naissance)` ; le dictionnaire D est maintenant composé des couples suivants : `prenom:Kevin, nom:Durand`
- `rech(D,prenom)` ; la fonction retourne Kevin

Rendez-vous sur le cours de première (sur pronote) pour retravailler l'ensemble des notions concernant les dictionnaires en Python.

- un dictionnaire permet de “stocker” des données
- Chaque élément d'un dictionnaire est composé de 2 parties, on parle de paires

"clé/valeur" (exemple : mon_dico = {"nom": "Durand", "prenom": "Christophe", "date de naissance": "29/02/1981"})

- Pour afficher une “valeur” particulière, on utilise la notation “mon_dico [nom_de_la_clé]”
- Il est possible de parcourir l’ensemble des clés d’un dictionnaire à l’aide d’une boucle for en utilisant “keys()”
- Il est possible de parcourir l’ensemble des valeurs d’un dictionnaire à l’aide d’une boucle for en utilisant “values()”

- Il est possible de parcourir l'ensemble des clés et des valeurs (en même temps) d'un dictionnaire à l'aide d'une boucle for en utilisant "items()"

Activité1 : Soit le programme Python suivant :

```
P = [{"nom":"Turing","prenom":"Alan","age":28},{"nom":"Lovelace","prenom":"Ada","age":27}]
```

Qu'obtient-on si on tape `P[1]['age']` dans une console Python ?

Activité 2 : Soit le programme Python suivant :

```
def ajoute(stock,element,quantite):
  if element in stock:
    stock[element] = stock[element] + quantite
  else:
    stock[element] = quantite


stock = { 'clous': 14, 'vis': 27, 'boulons': 8, 'écrous': 24}
ajoute(stock,'vis',5)
ajoute(stock,'chevilles',3)
```

Quelle est la valeur de la variable stock à la fin de cette exécution ?

Exercice :

Un enseignant de NSI du nom de Zizou stocke les résultats de ses élèves à l'aide d'un dictionnaire :

- les clés sont les noms des élèves,
- les valeurs sont des dictionnaires dont les clés sont les noms des épreuves et les valeurs sont des listes contenant la note obtenue suivie du coefficient de l'épreuve.

Voici le dictionnaire de 3 de ses élèves :

```
resultats = {'Alice': {'DS1' : [15.5, 3],
                       'DM1' : [14.5, 1],
                       'DS2' : [17, 4],
                       'PROJET1' : [18, 5],
                       'DS3' : [16, 4]},
             'Bob':{'DS1' : [6 , 3],
                    'DM1' : [14.5, 1],
                    'DS2' : [8.8, 4],
                    'PROJET1' : [10, 5],
                    'IE1' : [7, 2],
                    'DS3' : [8, 4],
                    'DS4' :[10, 4]},
            'Chaïn':{'DS1' : [13.5, 3],
                    'DM1' : [16, 1],
                    'DS2' : [14.5, 4],
                    'PROJET1' : [16, 5],
                    'IE1' : [16, 2],
                    'DS4' : [19, 4]}}
```

Le problème est que le professeur Zizou n'arrive pas à créer une fonction `moyenne` prenant en paramètre le dictionnaire `resultat` ainsi qu'un `nom` sous forme de chaîne de caractères et qui renvoie la moyenne des notes de l'élève ayant ce nom `nom` dans le cas où son nom apparaît dans le dictionnaire `resultat`.

Proposer une telle fonction `moyenne` afin d'aider cet enseignant. Il en sera surement reconnaissant !

II- Implémentation des dictionnaires

L'implémentation des dictionnaires dans les langages de programmation peut se faire à l'aide des tables de hachage. 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. Voici un texte qui vous permettra de comprendre le principe des fonctions de hachages : c'est quoi le hachage . Pour avoir quelques idées sur le principe des tables de hachages, je vous recommande le visionnage de cette vidéo : wandida : les tables de hachage

Si vous avez visionné la vidéo de wandida, vous avez déjà compris que l'algorithme de recherche dans une table de hachage a une complexité O(1) (le temps de recherche ne dépend pas du nombre d'éléments présents dans la table de hachage), alors que la complexité de l'algorithme de recherche dans un tableau non trié est O(n). Comme l'implémentation des dictionnaires s'appuie sur les tables de hachage, on peut dire que l'algorithme de recherche d'un élément dans un dictionnaire a une complexité O(1) alors que l'algorithme de recherche d'un élément dans un tableau non trié a une complexité O(n).

Python propose une implémentation des dictionnaires, nous avons déjà étudié cette implémentation l'année dernière, n'hésitez pas à vous référer à la ressource proposée l'an passé : les dictionnaires en Python

Complexité

La complexité d'ordre *n* (ou complexité linéaire) est une mesure de la façon dont le temps d'exécution (ou l'espace mémoire utilisé) d'un algorithme augmente proportionnellement à la taille de l'entrée *n*. En d'autres termes, si la taille de l'entrée double, le temps d'exécution (ou l'espace mémoire) nécessaire pour exécuter l'algorithme augmentera également proportionnellement.

Mathématiquement, si $\boldsymbol{T(n)}$ est la fonction qui donne le temps d'exécution en fonction de la taille de l'entrée *n*, alors la complexité d'ordre *n* signifie que $\boldsymbol{T(n)}$ peut être exprimé de manière linéaire, souvent sous la forme $\boldsymbol{T(n)=an+b}$, où *a* et *b* sont des constantes.

En notation big-O, on écrit généralement que la complexité d'ordre *n* est $O(n)$. Cela signifie que la croissance de la complexité de l'algorithme est linéaire par rapport à la taille de l'entrée.

Exemples d'algorithmes avec une complexité d'ordre *n* :

1. **Parcours linéaire d'un tableau :** Si vous devez parcourir chaque élément d'un tableau une fois, le temps nécessaire sera proportionnel au nombre d'éléments dans le tableau.

```
for element in tableau:

  # faire quelque chose avec l'élément
```

2. **Recherche linéaire :** Si vous recherchez un élément dans une liste non triée, vous devrez peut-être parcourir la liste élément par élément jusqu'à ce que vous trouviez celui que vous cherchez.

```
def recherche_lineaire(liste, element):

  for item in liste:

    if item == element:

      return True

  return False
```

La complexité d'ordre *n* est généralement considérée comme relativement efficace, et de nombreux algorithmes ont cette complexité dans le pire des cas. Cependant, il existe également des algorithmes avec des complexités meilleures (comme $O(\log n)$ ou $O(1)$) dans certaines situations.

Exercice :

- Sujet 6 2022 Exercice 3
- Sujet 7 2022 Exercice 2
- Sujet 6 2022 Exercice 1