Dictionnaires et tuples

Objectifs

- découvrir les fonctions de hachage
- savoir manipuler les dictionnaires
- savoir manipuler les tuples
- comprendre la notion de retour multiple de fonction

Connaissances et savoir-faire à maîtriser

1) Les fonctions de hachage

En informatique, une fonction de hachage est une fonction qui calcule l'**empreinte alphanumérique** (*hash* en anglais) d'une donnée. La signature d'une telle fonction est la suivante :

```
hachage(data)
    |- ENTREES : data est de n'importe quel type : chaîne, nombre, fichier...
    |- SORTIES : une chaîne de caractères de longueur constante
```

Le hash obtenu en sortie possède trois propriétés fondamentales :

- toute modification aussi infime soit-elle de la donnée en entrée génère en sortie une empreinte radicalement différente ;
- deux données distinctes ne peuvent générées la même empreinte : autrement dit, une fonction de hachage évite les **collisions** ;
- il est impossible de reconstituer la donnée à partir de son hash : une fonction de hachage de qualité est dite **à sens unique**.

Pour en savoir plus : https://fr.wikipedia.org/wiki/Fonction_de_hachage

```
import hashlib
print(hashlib.md5(b"Hello world!").hexdigest()) # 86fb269d190d2c85f6e0468ceca42a20
print(hashlib.md5(b"Hello world").hexdigest())  # 3e25960a79dbc69b674cd4ec67a72c62
```

Remarque 1 : on constate que le hash est complètement différent suivant qu'il y ait ou non le point d'exclamation.

*Remarque 2 : le caractère b qui précède la chaîne est **indispensable** et indique à Python de transformer la donnée en **octets**(bytes en anglais). On peut aussi utiliser la méthode de chaîne* `str.encode()` .

Remarque 3 : c'est ce mécanisme qui est utilisé pour crypter les mots de passe lors des phases d'authentification.

2) Les dictionnaires

a. Définition

Un dictionnaire (ou *table de hachage*) est un ensemble de données stockées sous la forme de **clé/-valeur**. C'est une structure de données qui diffère des listes dans le sens où on accède à une valeur par sa clé et non par son index. On retiendra simplement que le codage (ou implémentation) d'un dictionnaire se base sur le hachage des clés, ce qui assure au développeur de pouvoir accèder aux valeurs de manière performante en terme de rapidité et d'efficacité.

b. Opérations sur les dictionnaires

En tant que structure de données **mutables** (donc modifiables comme le sont les listes), les dictionnaires supportent les opérations d'ajout, de modification et de suppression d'éléments :

```
dico = {"Alice" : 28, "Carol" : 25} # dico = {} pour un dictionnaire vide
dico["Bob"] = 20 # ajout d'un couple clé/valeur
dico["Alice"] = 22 # modification d'un couple clé/valeur
del(dico["Alice"]) # suppression d'un couple clé/valeur
print(dico.get("Carol")) # affiche 25
print("Alice" in dico) # affiche False
```

c. Parcours d'un dictionnaire

Il est possible de récupérer chaque couple clé/valeur lors du parcours d'un dictionnaire :

```
for k, v in dico.items():
    print("{} : {} ans".format(k, v))
```

Mais il est également possible de récupérer uniquement les clés ou uniquement les valeurs :

```
for k in dico.keys():
    print("Prénom : {}".format(k))
for v in dico.values():
    print("Age : {} ans".format(v))
```

https://docs.python.org/fr/3/library/stdtypes.html?highlight=split#dict

3) Les tuples

Les tuples sont des listes de données **non mutables**, c'est-à-dire non modifiables au cours de l'exécution du programme. Comme pour les listes, les données stockées dans un tuple peuvent être ou non de même nature. On le définit en utilisant des parenthèses et on accède à ses éléments comme pour les listes en utilisant les crochets :

```
rgb = (100, 20, 256)
print("PIXEL: R={} G={} B={}".format(rgb[0], rgb[1], rgb[2]))
# affiche PIXEL R=100 G=20 B=256
infos = "Alexandre", 10 # on peut omettre les parenthèses
print("Nom : {} - Age : {}".format(infos[0], infos[1]))
# affiche Nom : Alexandre - Age : 10
```

Remarque 4 *: les tuples offrent un moyen commode pour une fonction de retourner plusieurs valeurs.*

```
def minmax(liste):
    ...
    return (valMin, valMax) # on peut omettre les parenthèses
```

Exercices à réaliser

L'objectif de ce TP est un peu plus ambitieux puisqu'il vise à coder un mini-système d'authentification. Le système aura les fonctionnalités suivantes :

- affichage du hash d'une chaîne de caractères ;
- affichage des comptes utilisateurs ;
- création d'un compte ;
- phase d'identification dit aussi de *login*.

Ces diverses fonctionnalités sont à coder en réalisant les deux exercices qui suivent :

Exercice 1

Ouvrir le fichier `TP-dicos-tuples-exo1.py`, lire les consignes et répondre aux questions suivantes :

1. Écrire une fonction :

   ```
   afficher_menu()
       |- ENTREES : rien
       |- SORTIES : un entier indiquant le choix de l'utilisateur
   ```

   qui affiche le menu suivant :

   ```
   ********************** MENU **********************
   * 1 - Afficher le hash d'une chaîne              *
   * 2 - Afficher les comptes utilisateurs          *
   * 3 - Créer un compte                            *
   * 4 - S'identifier                               *
   * 5 - Quitter                                    *
   * Votre choix :
   ```

   Pour imprimer ce menu "en continu", l'idée est d'utiliser une *boucle infinie*, c'est à dire une boucle WHILE dont la condition d'entrée repose sur un drapeau (ici appelé `prog_is_runnins`) positionné à `True` en début de programme, et qu'il faudra fixer à `False` lorsque l'utilisateur décidera de quitter l'application. Pensez également à contrôler la validité du choix utilisateur (entre 1 et 5).

2. Écrire une fonction `hacher(pmot)` puis coder la fonctionnalité n° 1 du menu.

   ```
   hacher(pmot)
       |- ENTREES : pmot est de type str
       |- SORTIES : le hash md5 de pmot
   ```

   Il existe plusieurs algorithmes de hachage (md5, sha256, ...) qui génèrent chacun des hash plus ou moins robuste. Voir la documentation du module hashlib pour plus d'informations : https://docs.python.org/fr/3.8/library/hashlib.html?highlight=hashlib#module-hashlib.

3. Écrire une fonction `saisir_user_password()` qui demande à l'utilisateur de saisir son nom utilisateur et son mot de passe et dont la signature est la suivante :

   ```
   saisir_user_password()
       |- ENTREES : rien
       |- SORTIES : un tuple de str de la forme (user, password)
   ```

   Attention : *le mot de passe stocké dans ce tuple devra avoir été haché avant d'être retourné. En effet, celui-ci va par la suite être stocké et l'intérêt d'un tel système, c'est justement d'éviter de stocker les mots de passe "en clair".*

Exercice 2

Cet exercice vise à coder les fonctionnalités n°2, 3 et 4 du menu.
Pour cela, ouvrir le fichier `TP-dicos-tuples-exo2.py` et y copier-coller votre code ou le corrigé de l'exercice précédent.

1. Écrire une fonction `afficher_comptes(pdico)` qui affiche tous les couples user/password présents dans le dictionnaire.
2. Écrire une fonction `creer_compte(pdico)` qui ajoute au dictionnaire un élément user/password demandé à l'utilisateur et où password est un mot de passe qui a été haché avant d'être ajouté. Coder alors la fonctionnalité numéro 3 du menu.
3. Écrire une fonction `identifier(pdico)` qui permet d'authentifier l'utilisateur et de coder la fonctionnalité numéro 4 du menu. On pourra écrire pour cela deux fonctions dont les signatures sont les suivantes :

```
verifier_user(pdico, puser)
    |- ENTREES : un dictionnaire et une chaîne
    |- SORTIES : un booléen True ou False indiquant si puser est dans pdico

verifier_password(pdico, puser, ppassword)
    |- ENTREES : un dictionnaire et deux chaînes puser et ppassword
    |- SORTIES : un booléen indiquant si le mot de passe est correct
```

À ce stade, on dispose d'une première version simplifié du système qui possède les fonctionnalités fondamentales. Toutefois, vous pouvez si vous le souhaitez enrichir ces fonctionnalités pour en faire un projet de fin d'année. Voici quelques pistes d'améliorations possibles :

- maintenir une liste des personnes connectés/déconnectés ;
- permettre à un utilisateur connecté de pouvoir modifier son mot de passe ;
- créer un compte super-utilisateur (en informatique, on l'appelle le compte **root**) ayant le droit de supprimer un compte ;
- gérer plus proprement les erreurs de choix notamment quant l'utilisateur saisit une lettre et non un nombre, ce qui provoque un arrêt brutal de l'application. On appelle cela **la gestion des exceptions** qui fera l'objet d'un prochain TP.

Compétences à valider

Dictionnaires et tuples		☀	🌤	🌧	⛈
Dictionnaires	Je sais définir une fonction de hachage				
	Je sais définir un dictionnaire				
	Je sais ajouter, modifier, supprimer un élément de dictionnaire				
	Je sais parcourir un dictionnaire				
Tuples	Je sais définir un tuple				
	J'ai compris la différence entre un tuple et une liste				
	Je sais définir une fonction qui retourne plusieurs valeurs				