En cybersécurité, le salage est une technique essentielle pour protéger les mots de passe stockés en base de données. Les fonctions (pseudo-)aléatoires interviennent à une étape cruciale : la création d'une donnée unique et imprévisible pour chaque utilisateur.

Pourquoi l'aléatoire est-il indispensable ?

Sans sel, deux utilisateurs ayant le même mot de passe (ex: 123456) auraient le même "hash" (empreinte numérique) dans la base de données. Un attaquant pourrait alors utiliser des Rainbow Tables (tables de correspondance pré-calculées) pour retrouver les mots de passe instantanément.
Le but du sel est de rendre chaque hash unique. Pour cela, le sel doit être :

- Unique pour chaque utilisateur.
- Imprévisible pour l'attaquant.

Pourquoi ne pas simplement utiliser une fonction "Random" classique ?

Si vous utilisez une fonction pseudo-aléatoire non sécurisée, un attaquant peut "remonter" l'état interne du générateur après avoir observé quelques échantillons de sels. Une fois l'état découvert, il peut prédire les sels des futurs utilisateurs ou même deviner ceux des comptes créés récemment, rendant le salage totalement inutile.

On souhaite écrire une fonction `creer_sel()` qui utilise le module random . cette fonction renvoie une chaîne de 8 caractères correspondant aux caractères ASCII de code décimal entre 33 et 125
cette chaîne est construite de manière aléatoire (en fait pseudo aléatoire)

1. Compléter la fonction creer_sel() ci-dessous

```
import random

def creer_sel():
    '''
     cette fonction renvoie un sel -chaîne de 8 caractères
     correspondant aux  caractères ASCII de code décimal
     entre 33 et 125 cette chaîne est construite de manière aléatoire
      (en fait pseudo aléatoire)
    '''
    sel=''
    for i in range(8):
        sel = ..............................
    return sel
```

Dans la pratique , les mots de passe ne sont pas stockés tels quels dans une base de donnée. On stocke plutôt le haché de mot_de_passe + sel ainsi que le sel (côte à côte)
Ici on supposera que la base de donnée est une simple liste de tuples (haché,sel)

2. Ecrire une fonction `base_donnee` qui construit une base de données (en fait une liste de tuples) à partir d'une liste de mots de passe

```
liste_mot_passe=['iloveu','liverpool','123123','carlos','cookie',\
'sweety','7777777','orange','rainbow','qm$pAx!']

def base_donnee():
    '''
    la fonction renvoie une liste de tuples (haché , sel)
    -haché : haché du mot de passe + sel aléatoire
    '''

    #on prend une liste vide qu'on remplira avec les tuples
    liste_mots_hash=[]
    for mot in liste_mot_passe:
    #on crée le sel
        y = ........................
    #on calcule l'empreinte sur 32 octets (256 bits) du mot de passe+sel
        x = hashlib.sha256((mot + y).encode()).hexdigest()
        #on stocke le haché et le sel dans une liste de tuples
         ..........................
    return liste_mots_hash
```

Pourquoi stocker le hash d'un mot de passe (plus sel) avec le sel dans la base de données.Ceci ne facilite t il pas la tâche de l attaquant?

Réponse: au premier abord, donner le sel à l'attaquant semble revenir à lui donner la "clé" pour ouvrir la porte. Mais en réalité, le sel n'est pas une clé secrète, c'est une barrière de calcul.

Voici pourquoi le stockage du sel en clair avec le hash est nécessaire et pourquoi cela ne facilite pas (vraiment) la tâche de l'attaquant.

Mais, comme le sel est une donnée aléatoire unique, le serveur a besoin de savoir quel sel a été utilisé pour vous au moment de votre inscription.

S'il ne stockait pas le sel, il ne pourrait jamais recalculer le même hash. Le mot de passe serait "perdu" à jamais, même pour le système.
3.Compléter la fonction craquer_password() qui va tester les mots de passe
du fichier (tel "rockyou.txt")

```
def craquer_password(hache_cible, sel, fichier_mdp):
    '''
    hache_cible : le haché qu on cherche à voler dans la base de données
    sel : le sel utilisé pendant le hachage
    fichier_mdp : le fichier contenant des mots de passe à tester
    '''

    with open(fichier_mdp, 'r', encoding='latin-1') as f:
        for ligne in f:
            mot = ligne.strip()
        # On simule le hachage : SHA-256(mot + sel)
            essai = hashlib.sha256((mot + ......).encode()).hexdigest()

           if essai == ......................:
        return f" Mot de passe trouvé : {mot}"
    return "Non trouvé dans la liste."
```

```
# Exemple d'utilisation
base=base_donnee()
#print(base)
cible = base[9][0] # Le hash trouvé en base de données
sel_associe = base[9][1] # Le sel trouvé à côté du hash
print(craquer_password(cible, sel_associe, "rockyou.txt"))
```