Exercice 1 : Vérification d'une adresse IP avec fonctions dédiées

Écrivez un programme Python qui vérifie si une adresse IP saisie par l'utilisateur est une adresse IP valide au format IPv4. Une adresse IP est constituée de quatre nombres séparés par des points, chacun étant compris entre 0 et 255 (par exemple, 192.168.1.1).

Le programme doit être structuré avec des fonctions pour chaque étape du traitement :

Fonction de validation de segment :

Créez une fonction **est_segment_valide**(segment) qui prend en paramètre un segment (c'est-à-dire chaque partie de l'adresse séparée par des points) et retourne True si le segment est :

- Un entier compris entre 0 et 255.
- Composé uniquement de chiffres.
- La fonction doit retourner False si le segment ne remplit pas ces critères.

Fonction de validation de l'adresse IP :

Créez une fonction **est_ip_valide**(ip) qui prend en paramètre une adresse IP sous forme de chaîne de caractères et utilise la fonction **est_segment_valide** pour vérifier chaque segment.

Cette fonction doit :

- Diviser l'adresse IP en segments en utilisant le point (.) comme séparateur.
- Vérifier qu'il y a exactement 4 segments.
- Utiliser est_segment_valide pour valider chaque segment.
- La fonction retourne True si tous les segments sont valides, sinon elle retourne False.

Fonction principale :

Créez une fonction **saisir_ip**() pour gérer l'interaction avec l'utilisateur.

Cette fonction doit :

- Demander à l'utilisateur de saisir une adresse IP.
- Utiliser la fonction est_ip_valide pour vérifier la validité de l'adresse.
- Afficher "Adresse IP valide" si l'adresse est correcte et "Adresse IP invalide" dans le cas contraire.
- Redemander la saisie tant que l'utilisateur n'a pas entré une adresse IP valide.

Exercice 2 : Analyse de Texte pour la Cybersécurité dédiées

Dans le domaine de la cybersécurité, l'analyse de texte peut être cruciale pour identifier des mots-clés, des longueurs de mots, ou même détecter des termes suspects dans un fichier de log. Vous allez écrire un programme Python qui demande à l'utilisateur de saisir un texte, puis applique différents traitements pour obtenir des informations sur ce texte.

L'objectif de cet exercice est de structurer le code de manière modulaire en créant chaque fonction dans un fichier séparé. Chaque fichier fonctionnera comme un module que vous importerez dans le programme principal.

Spécifications du Programme :

1. **Fonction de nettoyage du texte :**

   - Créez un fichier nommé **nettoyer_texte.py**.
   - Dans ce fichier, définissez une fonction **nettoyer_texte**(texte) qui prend en paramètre une chaîne de caractères et renvoie le texte en minuscules sans signes de ponctuation. Cela permet de rendre l'analyse plus uniforme.

2. **Fonction de comptage de mots :**

   - Créez un fichier nommé **compter_mots.py**.
   - Dans ce fichier, définissez une fonction **compter_mots**(texte) qui prend en paramètre le texte nettoyé et retourne le nombre de mots qu'il contient. Un mot est défini comme une séquence de caractères séparée par des espaces.

3. **Fonction de détection de mots suspects :**

   - Créez un fichier nommé **detecter_mot_suspect.py**.
   - Dans ce fichier, définissez une fonction **detecter_mot_suspect**(texte, mot_suspect) qui prend en paramètre le texte nettoyé et un mot spécifique à rechercher (comme "attaque" ou "virus"). La fonction doit retourner True si le mot est trouvé dans le texte et False sinon.

4. **Fonction principale :**

   - Créez un fichier nommé **main.py** qui servira de programme principal.
   - Dans ce fichier, importez les fonctions des modules précédents (nettoyer_texte, compter_mots, et detecter_mot_suspect).
   - Dans main.py, créez une fonction **analyser_texte**() qui :
     - Demande à l'utilisateur de saisir un texte.
     - Nettoie le texte en appelant **nettoyer_texte**.
     - Affiche le nombre de mots en appelant **compter_mots**.
     - Demande à l'utilisateur de saisir un mot à rechercher, puis utilise **detecter_mot_suspect** pour indiquer si ce mot est présent dans le texte.