Exercice 1 :

1. Écrire une fonction **`recherche`** qui prend en paramètre un tableau de nombres entiers **`tab`**, et qui renvoie la liste (éventuellement vide) des couples d'entiers consécutifs successifs qu'il peut y avoir dans **`tab`**.
2. Proposer un jeu de tests permettant de vérifier plusieurs cas de figures.

Exercice 2 :

— On appelle « mot » une chaîne de caractères composée avec des caractères choisis parmi les 26 lettres minuscules ou majuscules de l'alphabet.

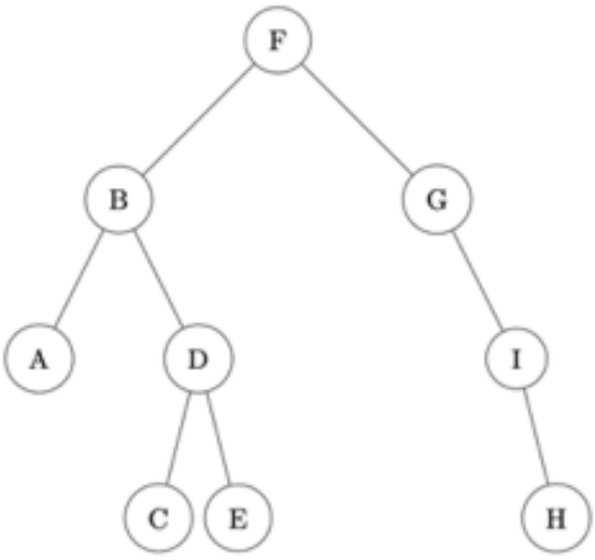
— On appelle « phrase » une chaîne de caractères : composée avec un ou plusieurs « mots » séparés entre eux par un seul caractère espace ' ', se finissant :

  — soit par un point '.' qui est alors collé au dernier mot,

  — soit par un point d'exclamation '!' ou d'interrogation '?' qui est alors séparé du dernier mot par un seul caractère espace ' '.

1. Construire plusieurs exemples de phrase et observer le lien entre le nombre d'espaces et le nombre de mots.
2. Écrire la fonction **`nombre_de_mots(phrase: str) → int`** qui renvoie le nombre de mots dans **`phrase`**.

Exercice 3 : Un arbre binaire de caractères est stocké sous la forme d'un dictionnaire où les clefs sont les caractères des nœuds de l'arbre et les valeurs, pour chaque clef, la liste des caractères des fils gauche et droit du nœud.



1. Construire le dictionnaire **`arbre`** représentant l'arbre binaire ci-dessus.
2. Écrire la fonction récursive **`taille(arbre: dict, lettre: str) → int`** qui renvoie la taille de l'**`arbre`**.
3. Écrire à nouveau la fonction récursive **`taille(arbre: dict, lettre: str) → int`** en prenant en compte les 4 cas suivants :

  — les deux fils du nœud sont '',

  — le fils gauche seulement est '',

  — le fils droit seulement est '',

  — aucun des deux fils n'est ''.

Christophe Viroulaud

Exercice 4 :

1. Écrire la fonction `fusion` prenant en paramètres deux tableaux non vides `tab1` et `tab2` d'entiers, chacun dans l'ordre croissant, renvoyant un tableau trié dans l'ordre croissant de l'ensemble des valeurs de `tab1` et `tab2`.
2. Établir un jeu de tests permettant de vérifier la correction de la fonction.

Exercice 5 : On définit une classe gérant une adresse IPv4. On rappelle qu'une adresse IPv4 est une adresse de longueur 4 octets, notée en décimale à point, en séparant chacun des octets par un point. On considère un réseau privé avec une plage d'adresses IP de 192.168.0.0 à 192.168.0.255. On considère que les adresses IP saisies sont valides. Les adresses IP 192.168.0.0 et 192.168.0.255 sont des adresses réservées.

1. Compléter le code de la classe `Adresse` dans le fichier `adresseip.py`
2. Créer plusieurs instances de la classe qui permettent de vérifier la correction des différentes méthodes.

Exercice 6 : Les chiffres romains sont un système ancien d'écriture des nombres. Les chiffres romains sont : I, V, X, L, C, D, et M. Ces symboles représentent respectivement 1, 5, 10, 50, 100, 500, et 1000 en base dix.
Lorsque deux caractères successifs sont tels que :

— le caractère placé à gauche possède une valeur supérieure ou égale à celui de droite, le nombre s'obtient en additionnant le caractère de gauche à la valeur de la chaîne située à droite. Ainsi, "XVI" est le nombre 16 car X + VI = 10 + 6.

— le caractère placé à gauche possède une valeur strictement inférieure à celui de droite, le nombre s'obtient en retranchant le caractère de gauche à la valeur de la chaîne située à droite. Ainsi, "CDIII" est le nombre 403 car DIII – C = 503 – 100.

On dispose d'un dictionnaire `dico`, à compléter, où les clés sont les caractères apparaissant dans l'écriture en chiffres romains et où les valeurs sont les nombres entiers associés en écriture décimale.
Compléter la fonction récursive `rom_to_dec(nombre: str)` → `int` du fichier `romain.py`, qui renvoie l'écriture décimale du paramètre `nombre` écrit en chiffres romains

Christophe Viroulaud