**Compte rendu – Arbre**

**I – Introduction**

Notre projet consiste à partir d’une liste de noms en texte (.txt), de créer un arbre binaire de recherche puis avec cet arbre créer 3 fichiers texte (.txt) différents dont dans chacun d’eux les éléments de l’arbre seront affichés d’une manière précise. D’abord sous le parcours d’affichage préfixe puis infixe et pour finir postfixe.

**Plan :**

**1 – Explication des codes dans ressources**

1. Arbre Binaire
2. L’insertion des valeurs dans l’arbre

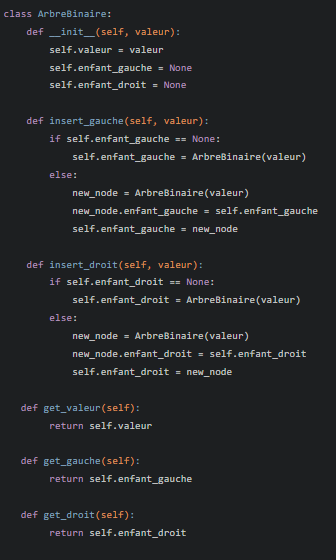
**2 – Explication des codes personnels**

1. Création de l’arbre
2. Les parcours d’affichage (préfixe, infixe, postfixe)
3. Création des fichiers .txt

**3 – Conclusion**

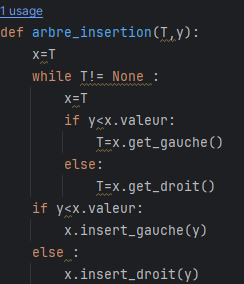
**Ressources**

* Liste de noms (.txt)
* Python (Arbre, insertion)

**1 – Explication des codes dans ressources**

a) Ce code définit une classe **ArbreBinaire** qui permet de créer et manipuler un arbre binaire.

* La méthode **\_\_init\_\_** est le constructeur de la classe. Elle initialise un noeud de l'arbre avec une valeur donnée et initialise les enfants gauche et droit à **None**.
* La méthode **insert\_gauche** insère un nouveau noeud avec une valeur donnée comme enfant gauche du noeud actuel. Si le noeud a déjà un enfant gauche, le nouveau noeud devient l'enfant gauche du noeud actuel et l'ancien enfant gauche devient l'enfant gauche du nouveau noeud.
* La méthode **insert\_droit** est similaire à **insert\_gauche**, mais pour l'insertion d'un enfant droit.
* Les méthodes **get\_valeur**, **get\_gauche** et **get\_droit** retournent la valeur du noeud, son enfant gauche et son enfant droit.

b) Ce code définit une fonction **arbre\_insertion** qui prend en paramètres un arbre binaire de recherche **T** et une valeur **y** à insérer dans cet arbre.

Il commence par initialiser une variable **x** avec la racine de l'arbre **T**.

Ensuite, il exécute une boucle **while** qui parcourt l'arbre en comparant la valeur **y** avec la valeur du nœud actuel de l'arbre **T**.

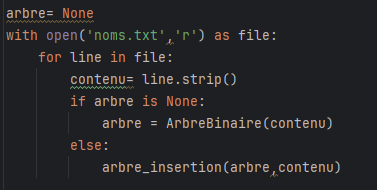
Si **y** est inférieur à la valeur du nœud actuel (**y < x.valeur**), alors il met à jour **T** avec le sous-arbre gauche du nœud actuel (**T=x.get\_gauche()**).

Sinon, il met à jour **T** avec le sous-arbre droit du nœud actuel (**T=x.get\_droit()**).

À chaque itération, il met également à jour la variable **x** avec le nœud actuel, ce qui permet de suivre le nœud parent du nœud courant.

Une fois que la boucle **while** atteint un nœud **None**, c'est-à-dire une feuille de l'arbre où la valeur **y** doit être insérée, le code insère **y** comme un nouveau nœud à gauche ou à droite de **x**, selon la comparaison de **y** avec **x.valeur**.

**2 – Explication des codes personnels**

a) Ce code lit les noms des patients à partir d'un fichier texte nommé **noms.txt** et construit un arbre binaire contenant ces noms. Voici ce que chaque partie du code fait :

**arbre = None**: Initialise la variable **arbre** à **None**. Cela signifie qu'au début, il n'y a pas d'arbre binaire construit.

**with open('noms.txt', 'r') as file:** : Ouvre le fichier **noms.txt** en mode lecture (**'r'**). La déclaration **with** est utilisée pour s'assurer que le fichier est correctement fermé une fois que le bloc est exécuté.

**for line in file:** : Itère sur chaque ligne du fichier **noms.txt**.

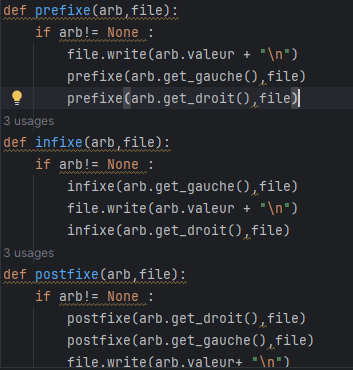
**contenu = line.strip()**: Supprime les espaces blancs en début et fin de chaque ligne et stocke le contenu de la ligne dans la variable **contenu**.

**if arbre is None:** : Vérifie si l'arbre est vide (c'est-à-dire si **arbre** est **None**). Si c'est le cas, cela signifie que c'est la première entrée et nous devons créer un nouvel arbre binaire.

**arbre = ArbreBinaire(contenu)**: Crée un nouvel arbre binaire avec le contenu de la première ligne du fichier **noms.txt**.

**else:** : Si l'arbre n'est pas vide, cela signifie que nous devons ajouter un nouveau nœud à l'arbre existant.

**arbre\_insertion(arbre, contenu)**: Insère le contenu de la ligne actuelle dans l'arbre binaire existant en utilisant une fonction arbre\_insertion vu précedemment.

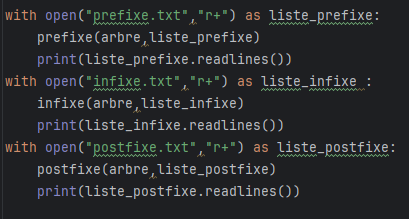
 b) Ce code définit trois fonctions **prefixe**, **infixe** et **postfixe** pour effectuer les parcours préfixe, infixe et postfixe d'un arbre binaire.

Chacune de ces fonctions prend deux arguments :

1. **arb**: C'est le nœud courant de l'arbre que nous examinons.
2. **file**: C'est le fichier dans lequel les valeurs des nœuds de l'arbre seront écrites.

Voici ce que chaque fonction fait :

* **prefixe**: Cette fonction effectue un parcours préfixe de l'arbre. Elle commence par écrire la valeur du nœud courant dans le fichier, puis elle appelle récursivement la fonction **prefixe** sur le sous-arbre gauche, puis sur le sous-arbre droit.
* **infixe**: Cette fonction effectue un parcours infixe de l'arbre. Elle commence par appeler récursivement la fonction **infixe** sur le sous-arbre gauche, puis elle écrit la valeur du nœud courant dans le fichier, puis elle appelle récursivement la fonction **infixe** sur le sous-arbre droit.
* **postfixe**: Cette fonction effectue un parcours postfixe de l'arbre. Elle commence par appeler récursivement la fonction **postfixe** sur le sous-arbre droit, puis sur le sous-arbre gauche, et enfin elle écrit la valeur du nœud courant dans le fichier.

Chaque fonction vérifie d'abord si le nœud courant est différent de **None**, ce qui est la condition d'arrêt de la récursion. Si le nœud est différent de **None**, la fonction effectue les opérations spécifiques au parcours demandé (préfixe, infixe ou postfixe) et appelle récursivement la même fonction sur les sous-arbres gauche et droit.

c) Ce code fait :

1. Il ouvre trois fichiers texte en mode lecture/écriture (**"r+"**) : **"prefixe.txt"**, **"infixe.txt"** et **"postfixe.txt"**.
2. Pour chaque fichier ouvert, il appelle une fonction correspondante (**prefixe**, **infixe** et **postfixe**) en lui passant l'arbre et le fichier comme arguments. Ces fonctions sont supposées écrire les noms des patients selon le parcours préfixe, infixe et postfixe de l'arbre dans les fichiers respectifs.
3. Après avoir appelé chaque fonction, il lit et affiche le contenu de chaque fichier texte à l'aide de la méthode **readlines()**.

**3 – Conclusion**

**Ce projet nous a permis de voir l’utilité d’un arbre binaire mais aussi la technique et les connaissances pour réaliser ce genre de mise en situation concrète. Ce projet ne fut point facile mais avec la hargne, nous avons pu le réaliser et ainsi croît notre expertise dans ce domaine.**