Implémentation des structures de données en Python

TP1	**Implémentation d'un arbre binaire par tuple de tuple (sans utiliser les objets)**	**3 séances**
TP2	**Implémentation d'un arbre binaire en Programmation Orientée Objet (POO)**	**2 séances**
TP3	**Implémentation des structures de données linéaires en POO**	**3 séances**

TP1 Implémentation d'un arbre binaire par tuple de tuple (sans utiliser les objets)

Séance 1/3

- définition **récursive** des arbres binaires
- implémentation d'un **nœud avec un tuple** (racine,fils gauche,fils droit)
- l'arbre vide est représenté par **None**
- codage d'un arbre binaire avec des **tuples de tuples**
- par exemple
  arbre=(2,(1,(0,None,None),None),(5,(4,(3,None,None),None),(12,(9,(8,(6,None,(7,None,None)),None),(10,None,(11,None,None))),(13,None,(14,None,(18,(17,(15,None,(16,None,None)),None),(19,None,None)))))))
- affichage du nombre de **nœuds**, du nombre de **feuilles**, de la liste de feuilles et de la **hauteur** de l'arbre binaire
- **appel récursif des fonctions** pour visiter simplement tout l'arbre
- tracé graphique de l'arbre binaire en utilisant la bibliothèque **matplotlib**

TP1 Implémentation d'un arbre binaire par tuple de tuple (sans utiliser les objets)

Séance 2/3

- implémentation d'un arbre **binaire de recherche** (ABR) avec des tuples de tuples
- **recherche récursive** d'une valeur dans l'ABR
- recherche du **minimum** (on va à fond à gauche) et du **maximum** (on va à fond à droite) dans l'ABR
- modification de l'ABR : **ajout** et **suppression** d'une valeur
- génération d'un **ABR aléatoire** à partir d'une liste aléatoire
- implémentation des différents types de **parcours** d'un arbre
- le parcours en **largeur** (parcours niveau par niveau, utilisant une file)
- distinction entre les 3 parcours en profondeur :
  - le parcours **préfixe** : R,G,D
  - le parcours **infixe** : G,R,D (donne une liste triée des nœuds de l'ABR)
  - le parcours **suffixe** : G,D,R
- utilisation d'un ABR pour **trier une liste** aléatoire : on range la liste dans un ABR puis on donne son parcours **infixe**

TP1 Éléments de syntaxe Python utilisés

- **Affectation multiple :**
- a,b,c=4,5,6 en une seule ligne remplaçant les 3 lignes suivantes :
- a=4
  b=5
  c=6
- Génération d'une **liste en compréhension** : liste=[2*k for k in range(8)] donne [0, 2, 4, 6, 8, 10, 12, 14] et remplace les 3 lignes suivantes :
- liste=[]
  for k in range(8):
  liste.append(2*k)
- Utilisation du mot clé **None** pour définir une variable sans valeur (symbolise ici l'arbre nul)
- Renvoie direct d'une **valeur booléenne** : return t==None remplace les 4 lignes suivantes :
- if t==None:
  return True
  else:
  return False
- Utilisation du mot clé **raise** pour générer une exception avec un message personnalisé :
  raise Exception('Arbre vide')
- Utilisation des **triples doubles quottes** pour commenter plusieurs lignes dans le programme
- Utilisation du module **queue** pour créer une file (utilisé ici pour le parcours en largeur)
- Utilisation d'un **tuple** pour renvoyer plusieurs valeurs à la fois à la sortie d'une fonction : return (a,b,c)

TP1 Implémentation d'un arbre binaire par tuple de tuple (sans utiliser les objets)

Séance 3/3

- codage d'un texte en binaire en utilisant les **codes ASCII** des caractères : ord()
- **décodage** d'un message binaire pour retrouver les codes ASCII : chr()
- notion de **code-préfixe** (aucun mot du code préfixe ne peut se prolonger pour donner un autre mot du code)
- **codage de Huffman** pour la compression de données : il génère pour chaque caractère un code-préfixe à taille variable
- utilisation **d'une file de priorité** (file auto-triée lors de l'ajout d'un élément) avec le module **heapq** de Python
- estimation du **taux de compression** obtenu avec le codage de Huffman par rapport au codage ASCII

TP2 Implémentation d'un arbre binaire en Programmation Orientée Objet (POO)

Séance 1/2

- Création d'une classe **Arbre** possédants les méthodes **__init__(self, val)**, **ajout_gauche(self, val)**, **ajout_droit(self, val)**, **taille(self)** et **hauteur(self)**
- Ajout d'une méthode **vide(self)** renvoyant **True** si l'arbre est vide
- Distinction entre les **fonctions externes** à la classe et les **méthodes** internes à la classe
- Distinction entre l'arbre **a** (liste de liste) et l'arbre **b** (définit par les objets)

TP2 Implémentation d'un arbre binaire en Programmation Orientée Objet (POO)

Séance 2/2

- Création de 3 **fonctions** externes à la classe **Arbre** prenant en paramètre un objet "arbre" de classe Arbre et affichant chacune un parcours en profondeur sur une ligne dans la console :
  - **parcours_prefixe(arbre)**
  - **parcours_infixe(arbre)**
  - **parcours_suffixe(arbre)**
- Enrichissement de la classe **Arbre** en créant 3 nouvelles **méthodes** qui affichent chacune un parcours en profondeur de l'arbre lui-même sur une ligne dans la console :
  - **parcours_prefixe(self)**
  - **parcours_infixe(self)**
  - **parcours_suffixe(self)**

TP3 — Implémentation des structures de données linéaires en POO

Séance 1/3

- rappel des bases de la POO : **classes**, **attributs**, **méthodes**, **instance**, **constructeur**, etc.
- rappel sur les structures de données (**linéaire** ou non linéaire, **homogène** ou non, **statique** ou **dynamique**)
- création et test de la classe **Perso** (avec nom, prénom, statut, date de naissance et nationalité)
- test de la méthode **info()**
- ajout des méthodes **age()** et **majeur()** à la classe **Perso**
- ajout de l'attribut **classe** et de la méthode **est_eleve_de()** à la classe **Perso**

TP3	Implémentation des structures de données linéaires en POO	Séance 2/3

- création d'une classe **Tableau** avec les méthodes **insert** et **supprime**
- création d'une classe **Pile**
- création d'une classe **File**
- création d'une classe **Maillon** pour implémenter une **liste chaînée**
- les programmes de base à compléter sont disponibles sur l'ENT nsi.gecif.net

TP3	Implémentation des structures de données linéaires en POO	Séance 3/3

- Terminer le **TP3**
- Terminer le **TP2**
- Terminer le **TP1**
- Faire les exercices suivants de l'épreuve écrite de NSI avec validation dans Python:
- BAC 2025 **Sujet 1** Exercice 1
- BAC 2025 **Sujet 2** Exercices 1 et 2
- BAC 2025 **Sujet 3** Exercice 2
- BAC 2025 **Sujet 4** Exercice 2