

## TP : Théorie des Arbres

Groupe ZASAR : ZAMI Christophe, SARMEZAN Noémi

### Question 2 :

Les bibliothèques Python sont utilisées pour modéliser les données souhaitées sous forme de graphe. Le code va construire le graphe en lui ajoutant ses nœuds et ses arrêtes grâce à des fonctions.

Python possède de nombreux Framework différents et propose un large choix d'options pour ceux-ci. Parmi eux, on retrouve :

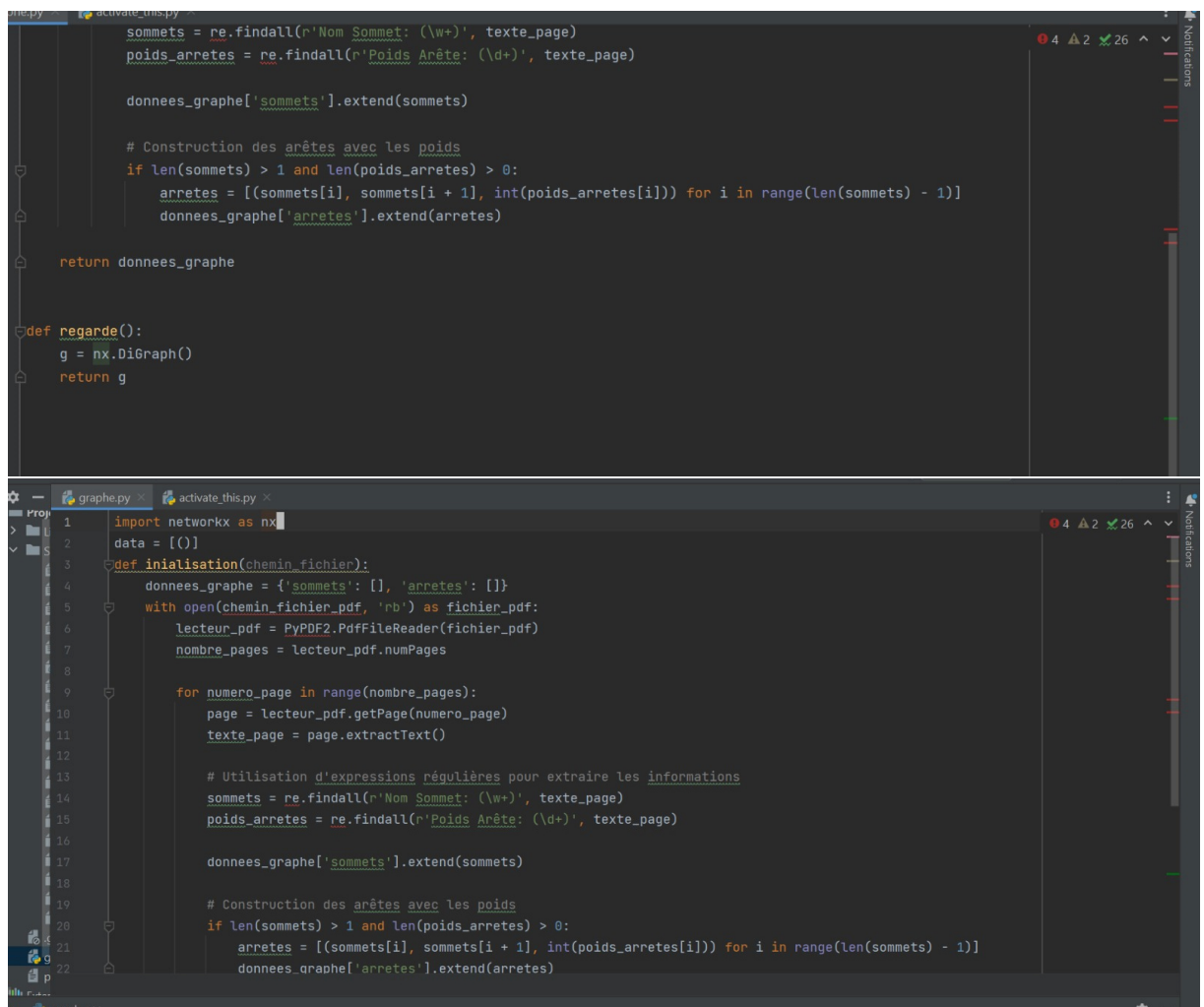
- ❖ **Tkinter** : Une bibliothèque pour l'interface graphique utilisateur en Open Source. La prise en main est très simple et il à la spécificité d'être préinstallé en Python. Cependant, Tkinter est une ancienne bibliothèque qui présente des limites.
- ❖ **Kivy** : A été conçu pour la création d'interfaces utilisateurs avec des applications multi-touch. Elle permet aux concepteurs de coder et de déployer sur plusieurs plates-formes. Cependant le prise en main est un peu compliquée et nécessite du temps.
- ❖ Nous avons trouvé ces deux bibliothèques Python mais avons préféré utiliser **NetworkX** pour sa simplicité d'installation et les options utiles qu'elle propose pour une utilisation large dans le domaine des graphes. La version 2.2 de NetworkX permet de créer, manipuler et analyser les réseaux de façon optimal. Elle dispose aussi d'algorithmes classiques de théorie des graphes tel que Dijkstra, PageRank et SImRank...

#### *A) Installation de la bibliothèque NetworkX sur PyCharm*

```
C:\Users\chris\OneDrive\Documents\Universiter\L3\semestre 1\Projet rinaldo> pip install Networkx
Collecting Networkx
  Using cached networkx-3.2.1-py3-none-any.whl (1.6 MB)
Installing collected packages: Networkx
Successfully installed Networkx-3.2.1

[notice] A new release of pip available: 22.3 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
C:\Users\chris\OneDrive\Documents\Universiter\L3\semestre 1\Projet rinaldo>
```

## B) Chargement du graphe



```
def regarde():
    g = nx.DiGraph()
    return g

sommets = re.findall(r'Nom Sommet: (\w+)', texte_page)
poids_arretes = re.findall(r'Poids Arête: (\d+)', texte_page)

donnees_graphe['sommets'].extend(sommets)

# Construction des arêtes avec les poids
if len(sommets) > 1 and len(poids_arretes) > 0:
    arretes = [(sommets[i], sommets[i + 1], int(poids_arretes[i])) for i in range(len(sommets) - 1)]
    donnees_graphe['arretes'].extend(arretes)

return donnees_graphe

def initialise(chemin_fichier):
    donnees_graphe = {'sommets': [], 'arretes': []}
    with open(chemin_fichier_pdf, 'rb') as fichier_pdf:
        lecteur_pdf = PyPDF2.PdfFileReader(fichier_pdf)
        nombre_pages = lecteur_pdf.numPages

        for numero_page in range(nombre_pages):
            page = lecteur_pdf.getPage(numero_page)
            texte_page = page.extractText()

            # Utilisation d'expressions régulières pour extraire les informations
            sommets = re.findall(r'Nom Sommet: (\w+)', texte_page)
            poids_arretes = re.findall(r'Poids Arête: (\d+)', texte_page)

            donnees_graphe['sommets'].extend(sommets)

            # Construction des arêtes avec les poids
            if len(sommets) > 1 and len(poids_arretes) > 0:
                arretes = [(sommets[i], sommets[i + 1], int(poids_arretes[i])) for i in range(len(sommets) - 1)]
                donnees_graphe['arretes'].extend(arretes)
```

## C) Parcourir le graphe

Ce code Python réalise un parcours en profondeur (DFS) d'un graphe créé à l'aide de la bibliothèque NetworkX. En spécifiant un nœud de départ, le code explore récursivement les voisins de chaque nœud, affichant le nom du nœud en cours de visite. Cette approche permet de parcourir le graphe de manière profonde avant de revenir en arrière, assurant ainsi que tous les chemins sont explorés en profondeur avant d'explorer d'autres branches.

## D) Utilisation de la bibliothèque

### I- Création du graphe

- ❖ Premièrement, après avoir importé NetworkX, on crée le graphe à l'aide de la fonction : **'nx.DiGraph'** (graphe orienté).

- ❖ On déclare le nombre de sommets de ce graphe avec la commande « **nomGraphe.add\_nodes\_from(range(6))** »
- ❖ Ensuite, on y ajoute les nœuds et les arrêtes en utilisant « **nomGraphe.add\_edge('Moule', 'Petit-Canal')** » pour les liens (arrêtes) et « **nomGraphe.node('Moule', 'Petit-Canal')** » pour les nœuds avec lien.
- ❖ Pour ajouter un poids à un lien on utilise « **nomGraphe.add\_edge('Moule', 'Petit-Canal', poids = 6)** »

## II- Visualisation du graphe

Pour afficher le graphe il faut utiliser la commande « **nx.draw\_networkx(nomGraphe)** »