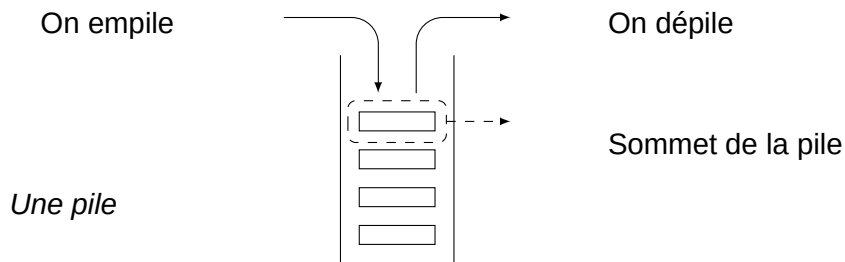


Exercice 1 (E1S1)

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti » :



On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

Structure de données abstraite : Pile
Utilise : Éléments, Booléen
Opérations : <ul style="list-style-type: none">— <code>creer_pile_vide</code> : $\emptyset \rightarrow$ Pile <code>creer_pile_vide()</code> renvoie une pile vide— <code>est_vide</code> : Pile \rightarrow Booléen <code>est_vide(pile)</code> renvoie True si pile est vide, False sinon— <code>empiler</code> : Pile, Éléments \rightarrow Rien <code>empiler(pile, element)</code> ajoute element au sommet de la pile— <code>depiler</code> : Pile \rightarrow Éléments <code>depiler(pile)</code> renvoie l'élément au sommet de la pile en le retirant de la pile

Question 1 On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut) :

4
2
5
8

Quel sera le contenu de la pile Q après exécution de la suite d'instructions suivante?

```
1   Q = creer_pile_vide()
2   while not est_vide(P):
3       empiler(Q, depiler(P))
```

Question 2

1. On appelle *hauteur* d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile P et renvoie sa hauteur. Après appel de cette fonction, la pile P doit avoir retrouvé son état d'origine.

Exemple : si P est la pile de la question 1 : `hauteur_pile(P) = 4`.

Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les ??? par les bonnes instructions.

```

1      def hauteur_pile(P):
2          Q = creer_pile_vide()
3          n = 0
4          while not(est_vide(P)):
5              ???
6              x = depiler(P)
7              empiler(Q,x)
8          while not(est_vide(Q)):
9              ???
10         empiler(P, x)
11         return ???

```

2. Créer une fonction `max_pile` ayant pour paramètres une pile `P` et un entier `i`. Cette fonction renvoie la position `j` de l'élément maximum parmi les `i` derniers éléments empilés de la pile `P`. Après appel de cette fonction, la pile `P` devra avoir retrouvé son état d'origine. La position du sommet de la pile est 1.

Exemple : si `P` est la pile de la question 1 : `max_pile(P, 2) = 1`

Question 3 Créer une fonction `retourner` ayant pour paramètres une pile `P` et un entier `j`. Cette fonction inverse l'ordre des `j` derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

Exemple : si `P` est la pile de la question 1(a), après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

5
2
4
8

Question 4 L'objectif de cette question est de trier une pile de crêpes.

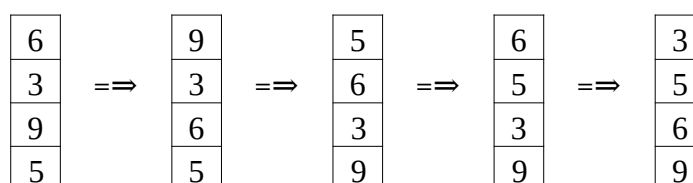
On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

Le principe est le suivant :

- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :



fin

Créer la fonction `tri_crepes` ayant pour paramètre une pile `P`. Cette fonction trie la pile `P` selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile `P` est.

6
14
12
5
8

, après l'appel de `tri_crepes(P)`, la pile `P` devient

5
7
8
12
14

Exercice 2 (E5S3)

Une méthode simple pour gérer l'ordonnancement des processus est d'exécuter les processus en une seule fois et dans leur ordre d'arrivée.

1. Parmi les propositions suivantes, quelle est la structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out) ?
 - a) liste
 - b) dictionnaire
 - c) pile
 - d) file
2. On choisit de stocker les données des processus en attente à l'aide d'une liste Python `lst`. On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`. Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.

On choisit maintenant d'implémenter une file `file` à l'aide d'un couple `(p1,p2)` où `p1` et `p2` sont des piles. Ainsi `file[0]` et `file[1]` sont respectivement les piles `p1` et `p2`.

Pour enfiler un nouvel élément `elt` dans `file`, on l'empile dans `p1`.

Pour défiler `file`, deux cas se présentent.

- La pile `p2` n'est pas vide : on dépile `p2`.
- La pile `p2` est vide : on dépile les éléments de `p1` en les empilant dans `p2` jusqu'à ce que `p1` soit vide, puis on dépile `p2`.

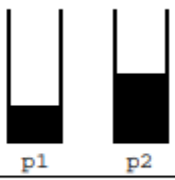
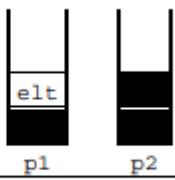
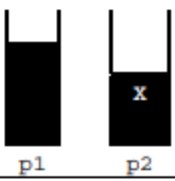
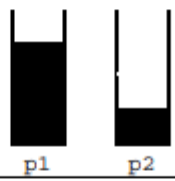
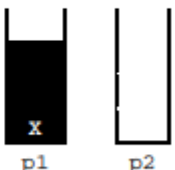
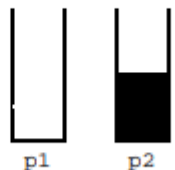
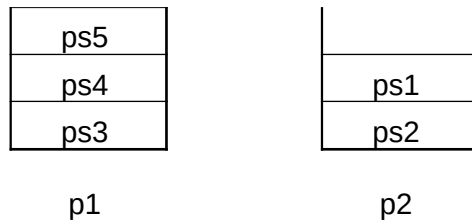
	État de la file avant	État de la file après
<code>enfiler(file, elt)</code>		
<code>defiler(file)</code> cas où <code>p2</code> n'est pas vide		
<code>defiler(file)</code> cas où <code>p2</code> est vide		

Illustration du fonctionnement des fonctions `enfiler` et `defiler`.

3. On considère la situation représentée ci-dessous.



On exécute la séquence d'instructions suivante :

enfiler(file,ps6)

defiler(file)

defiler(file)

defiler(file)

enfiler(file,ps7)

Représenter le contenu final des deux piles à la suite de ces instructions.

4. On dispose des fonctions :

- empiler(p,elt) qui empile l'élément elt dans la pile p,
- depiler(p) qui renvoie le sommet de la pile p si p n'est pas vide et le supprime,
- pile_vide(p) qui renvoie True si la pile p est vide, False si la pile p n'est pas vide.

- a. Écrire en Python une fonction est_vide(f) qui prend en argument un couple de piles f et qui renvoie True si la file représentée par f est vide, False sinon.
- b. Écrire en Python une fonction enfiler(f,elt) qui prend en arguments un couple de piles f et un élément elt et qui ajoute elt en queue de la file représentée par f.
- c. Écrire en Python une fonction defiler(f) qui prend en argument un couple de piles f et qui renvoie l'élément en tête de la file représentée par f en le retirant.

Exercice 3 (E1S6)

On crée une classe Pile qui modélise la structure d'une pile d'entiers.

Le constructeur de la classe initialise une pile vide.

La définition de cette classe sans l'implémentation de ses méthodes est donnée ci-dessous.

class Pile:

```
def __init__(self):
    """Initialise la pile comme une pile vide."""

def est_vide(self):
    """Renvoie True si la liste est vide, False sinon."""

def empiler(self, e):
    """Ajoute l'élément e sur le sommet de la pile,
    ne renvoie rien."""

def depiler(self):
    """Retire l'élément au sommet de la pile et le renvoie."""

def nb_elements(self):
    """Renvoie le nombre d'éléments de la pile. """

def afficher(self):
    """Affiche de gauche à droite les éléments de la pile, du fond de la pile vers son sommet. Le
    sommet est alors l'élément affiché le plus à droite. Les éléments sont séparés par une virgule.
    Si la pile est vide la méthode affiche « pile vide »."""
```

Seules les méthodes de la classe ci-dessus doivent être utilisées pour manipuler les objets Pile.

1.

- a.** Écrire une suite d'instructions permettant de créer une instance de la classe Pile affectée à une variable pile1 contenant les éléments 7, 5 et 2 insérés dans cet ordre.

Ainsi, à l'issue de ces instructions, l'instruction pile1.afficher() produit l'affichage : 7, 5, 2.

- b.** Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler()
pile1.empiler(5)
pile1.empiler(element1)
pile1.afficher()
```

2. On donne la fonction mystere suivante :

```
def mystere(pile, element):
    pile2 = Pile()
    nb_elements = pile.nb_elements()
    for i in range(nb_elements):
        elem = pile.depiler()
        pile2.empiler(elem)
        if elem == element:
            return pile2
    return pile2
```

a) Dans chacun des quatre cas suivants, quel est l'affichage obtenu dans la console ?

- Cas n°1

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 2).afficher()
```
- Cas n°2

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 9).afficher()
```
- Cas n°3

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 3).afficher()
```
- Cas n°4

```
>>>pile.est_vide()
True
>>>mystere(pile, 3).afficher()
```

b) Expliquer ce que permet d'obtenir la fonction mystere.

3. Écrire une fonction `etendre(pile1, pile2)` qui prend en arguments deux objets Pile appelés `pile1` et `pile2` et qui modifie `pile1` en lui ajoutant les éléments de `pile2` rangés dans l'ordre inverse. Cette fonction ne renvoie rien.

On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile1.afficher()
7, 5, 2, 3
>>>pile2.afficher()
1, 3, 4
>>>etendre(pile1, pile2)
>>>pile1.afficher()
7, 5, 2, 3, 4, 3, 1
>>>pile2.est_vide()
True
```

4. Écrire une fonction `supprime_toutes_occurences(pile, element)` qui prend en arguments un objet Pile appelé `pile` et un élément `element` et supprime tous les éléments `element` de `pile`.

On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile.afficher()
7, 5, 2, 3, 5
>>>supprime_toutes_occurences (pile, 5)
>>>pile.afficher()
7, 2, 3
```

Exercice 4 (E1S6)

Cryptage selon le « Code de César »

Dans cet exercice, on étudie une méthode de chiffrement de chaînes de caractères alphabétiques. Pour des raisons historiques, cette méthode de chiffrement est appelée "code de César". On considère que les messages ne contiennent que les lettres capitales de l'alphabet "ABCDEFGHIJKLMNOPQRSTUVWXYZ" et la méthode de chiffrement utilise un nombre entier fixé appelé la clé de chiffrement.

1. Soit la classe CodeCesar définie ci-dessous :

```
class CodeCesar:

    def __init__(self, cle):
        self.cle = cle
        self.alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    def decale(self, lettre):
        num1 = self.alphabet.find(lettre)
        num2 = num1+self.cle
        if num2 >= 26:
            num2 = num2-26
        if num2 < 0:
            num2 = num2+26
        nouvelle_lettre = self.alphabet[num2]
        return nouvelle_lettre
```

On rappelle que la méthode str.find(lettre) renvoie l'indice (index) de la lettre dans la chaîne de caractères str

Représenter le résultat d'exécution du code Python suivant :

```
code1 = CodeCesar(3)
print(code1.decale('A'))
print(code1.decale('X'))
```

3. La méthode de chiffrement du « code César » consiste à décaler les lettres du message dans l'alphabet d'un nombre de rangs fixé par la clé. Par exemple, avec la clé 3, toutes les lettres sont décalées de 3 rangs vers la droite : le A devient le D, le B devient le E, etc.

Ajouter une méthode cryptage(self, texte) dans la classe CodeCesar définie à la question précédente, qui reçoit en paramètre une chaîne de caractères (le message à crypter) et qui retourne une chaîne de caractères (le message crypté).

Cette méthode cryptage(self, texte) doit crypter la chaîne texte avec la clé de l'objet de la classe CodeCesar qui a été instancié.

Exemple :

```
>>> code1 = CodeCesar(3)
>>> code1.cryptage("NSI")
'QVL'
```


4. Écrire un programme qui :

- demande de saisir la clé de chiffrement
- crée un objet de classe CodeCesar
- demande de saisir le texte à chiffrer
- affiche le texte chiffré en appelant la méthode cryptage

4. On ajoute la méthode transforme(texte) à la classe CodeCesar :

```
def transforme(self, texte):  
    self.cle = -self.cle  
    message = self.cryptage(texte)  
    self.cle = -self.cle  
    return message
```

On exécute la ligne suivante : `print(CodeCesar(10).transforme("PSX"))`

Que va-t-il s'afficher ? Expliquer votre réponse.

Exercice 5 (E2S6)

Une ville souhaite gérer son parc de vélos en location partagée. L'ensemble de la flotte de vélos est stocké dans une table de données représentée en langage Python par un dictionnaire contenant des associations de type `id_velo : dict_velo` où `id_velo` est un nombre entier compris entre 1 et 199 qui correspond à l'identifiant unique du vélo et `dict_velo` est un dictionnaire dont les clés sont : "type", "etat", "station".

Les valeurs associées aux clés "type", "etat", "station" de `dict_velo` sont de type chaînes de caractères ou nombre entier :

- "type" : chaîne de caractères qui peut prendre la valeur "electrique" ou "classique"
- "etat" : nombre entier qui peut prendre la valeur 1 si le vélo est disponible, 0 si le vélo est en déplacement, -1 si le vélo est en panne
- "station" : chaînes de caractères qui identifie la station où est garé le vélo.

Dans le cas où le vélo est en déplacement ou en panne, "station" correspond à celle où il a été dernièrement stationné.

Voici un extrait de la table de données :

```
flotte = {  
    12 : {"type" : "electrique", "etat" : 1, "station" : "Prefecture"},  
    80 : {"type" : "classique", "etat" : 0, "station" : "Saint-Leu"},  
    45 : {"type" : "classique", "etat" : 1, "station" : "Baraban"},  
    41 : {"type" : "classique", "etat" : -1, "station" : "Citadelle"},  
    26 : {"type" : "classique", "etat" : 1, "station" : "Coliseum"},  
    28 : {"type" : "electrique", "etat" : 0, "station" : "Coliseum"},  
    74 : {"type" : "electrique", "etat" : 1, "station" : "Jacobins"},  
    13 : {"type" : "classique", "etat" : 0, "station" : "Citadelle"},  
    83 : {"type" : "classique", "etat" : -1, "station" : "Saint-Leu"},  
    22 : {"type" : "electrique", "etat" : -1, "station" : "Joffre"}  
}
```

`flotte` étant une variable globale du programme.

Toutes les questions de cet exercice se réfèrent à l'extrait de la table `flotte` fourni cidessus. L'annexe 1 présente un rappel sur les dictionnaires en langage Python.

1.

- a.a. Que renvoie l'instruction `flotte[26]` ?
- a.b. Que renvoie l'instruction `flotte[80]["etat"]` ?
- a.c. Que renvoie l'instruction `flotte[99]["etat"]` ?

2. Voici le script d'une fonction :

```
def proposition(choix):  
    for v in flotte:  
        if flotte[v]["type"] == choix and flotte[v]["etat"] == 1:  
            return flotte[v]["station"]
```

- a) Quelles sont les valeurs possibles de la variable `choix` ?
- b) Expliquer ce que renvoie la fonction lorsque l'on choisit comme paramètre l'une des valeurs possibles de la variable `choix`.

3.

- 3.a. Écrire un script en langage Python qui affiche les identifiants (`id_velo`) de tous les vélos disponibles à la station "Citadelle".
- 3.b. Écrire un script en langage Python qui permet d'afficher l'identifiant (`id_velo`) et la station de tous les vélos électriques qui ne sont pas en panne.

4. On dispose d'une table de données des positions GPS de toutes les stations, dont un extrait est donné ci-dessous. Cette table est stockée sous forme d'un dictionnaire.

Chaque élément du dictionnaire est du type:
'nom de la station' : (latitude, longitude)

```
stations = {  
    'Prefecture' : (49.8905, 2.2967) ,  
    'Saint-Leu' : (49.8982, 2.3017),  
    'Coliseum' : (49.8942, 2.2874),  
    'Jacobins' : (49.8912, 2.3016)  
}
```

On admet que l'on dispose d'une fonction `distance(p1, p2)` permettant de renvoyer la distance en mètres entre deux positions données par leurs coordonnées GPS (latitude et longitude).

Cette fonction prend en paramètre deux tuples représentant les coordonnées des deux positions GPS et renvoie un nombre entier représentant cette distance en mètres.

Par exemple, `distance((49.8905, 2.2967), (49.8912, 2.3016))` renvoie 9591

Écrire une fonction qui prend en paramètre les coordonnées GPS de l'utilisateur sous forme d'un tuple et qui renvoie, pour chaque station située à moins de 800 mètres de l'utilisateur :

- le nom de la station ;
- la distance entre l'utilisateur et la station ;
- les identifiants des vélos disponibles dans cette station.

Une station où aucun vélo n'est disponible ne doit pas être affichée.

Annexe 1

Action	Instruction et syntaxe
Créer un dictionnaire vide	<code>dico={}</code>
Obtenir un élément d'un dictionnaire existant à partir de sa clé renvoie une erreur si cle n'existe pas dans le dictionnaire	<code>dico[cle]</code>
Modifier la valeur d'un élément d'un dictionnaire à partir de sa clé	<code>dico[cle]=nouvelle_valeur</code>
Ajouter un élément dans un dictionnaire existant	<code>dico[nouvelle_cle]=valeur</code>
Supprimer et obtenir un élément d'un dictionnaire à partir de sa clé	<code>dico.pop(cle)</code>
Tester l'appartenance d'un élément à un dictionnaire (renvoie un booléen)	<code>cle in dico</code>
Objet itérable contenant les clés. Ce n'est pas un objet de type list	<code>dico.keys()</code>
Objet itérable contenant les valeurs. Ce n'est pas un objet de type list	<code>dico.values()</code>
Objet itérable contenant les couples (clé,valeur)	<code>dico.items()</code>
Afficher les associations cle:valeur du dictionnaire dico	<pre>for cle in dico: print(cle, dico[cle])</pre>

Exercice 6 (E1S7)

On cherche à obtenir un mélange d'une liste comportant un nombre **pair** d'éléments. Dans cet exercice, on notera N le nombre d'éléments de la liste à mélanger.

La méthode de mélange utilisée dans cette partie est inspirée d'un mélange de jeux de cartes :

- On sépare la liste en deux piles :
 - à gauche, la première pile contient les $N/2$ premiers éléments de la liste ;
 - à droite, la deuxième pile contient les $N/2$ derniers éléments de la liste.
- On crée une liste vide.
- On prend alors le sommet de la pile de gauche et on le met en début de liste.
- On prend ensuite le sommet de la pile de droite que l'on ajoute à la liste et ainsi de suite jusqu'à ce que les piles soient vides.

Par exemple, si on applique cette méthode de mélange à la liste `['V','D','R','3','7','10']`, on obtient pour le partage de la liste en 2 piles :

Pile gauche	Pile droite
'R'	'10'
'D'	'7'
'V'	'3'

La nouvelle liste à la fin du mélange sera donc `['R','10','D','7','V','3']`.

1. Que devient la liste `['7','8','9','10','V','D','R','A']` si on lui applique cette méthode de mélange ?

On considère que l'on dispose de la structure de données de type pile, munie des seules instructions suivantes :

p = Pile() : crée une pile vide nommée **p**

p.est_vide() : renvoie Vrai si la liste est vide, Faux sinon

p.empiler(e) : ajoute l'élément **e** dans la pile

e = p.depiler() : retire le dernier élément ajouté dans la pile et le retourne (et l'affecte à la variable **e**)

p2 = p.copier() : renvoie une copie de la pile **p** sans modifier la pile **p** et l'affecte à une nouvelle pile **p2**

2. Recopier et compléter le code de la fonction suivante qui transforme une liste en pile.

```
def liste_vers_pile(L):  
    """prend en paramètre une liste et renvoie une pile"""  
    N = len(L)  
    p_temp = Pile()  
    for i in range(N):  
        .....  
  
    return .....
```

3. On considère la fonction suivante qui partage une liste en deux piles. Lors de sa mise au point et pour aider au débogage, des appels à la fonction **affichage_pile** ont été insérés. La fonction **affichage_pile(p)** affiche la pile **p** à l'écran verticalement sous la forme suivante :

dernier élément empilé
...
...
premier élément empilé

```

def partage(L):
    N = len(L)
    p_gauche = Pile()
    p_droite = Pile()
    for i in range(N/2):
        p_gauche.empile(L[i])
    for i in range(N/2, N):
        p_droite.empile(L[i])
    affichage_pile(p_gauche)
    affichage_pile(p_droite)
    return p_gauche, p_droite

```

Quels affichages obtient-on à l'écran lors de l'exécution de l'instruction : `partage([1,2,3,4,5,6])` ?

4.

4.a Dans un cas général et en vous appuyant sur une séquence de schémas, **expliquer** en quelques lignes comment fusionner deux piles **p_gauche** et **p_droite** pour former une liste **L** en alternant un à un les éléments de la pile **p_gauche** et de la pile **p_droite**.

4.b. **Écrire** une fonction **fusion(p1,p2)** qui renvoie une liste construite à partir des deux piles **p1** et **p2**.

5. **Compléter** la dernière ligne du code de la fonction **affichage_pile** pour qu'elle fonctionne de manière récursive.

```

def affichage_pile(p):
    p_temp = p.copier()
    if p_temp.est_vide():
        print('____')
    else:
        elt = p_temp.depiler()
        print('| ', elt, ' |')
    ...

```

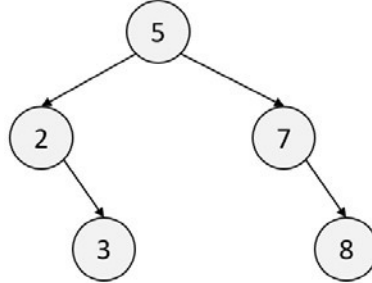
ligne à compléter

Exercice 7 (E3S9)

Cet exercice traite principalement du thème « algorithmique, langages et programmation » et en particulier les arbres binaires de recherche. La première partie aborde les arbres en mode débranché via l'application d'un algorithme sur un exemple. La suivante porte sur la programmation orientée objet. La dernière partie fait le lien avec les algorithmes de tri.

Partie A : Étude d'un exemple

Considérons l'arbre binaire de recherche ci-dessous.



- 2 Indiquer quelle valeur a le nœud racine et quels sont les fils de ce nœud.
- 3 Indiquer quels sont les nœuds de la branche qui se termine par la feuille qui a pour valeur 3.
- 4 Dessiner l'arbre obtenu après l'ajout de la valeur 6.

Partie B : Implémentation en Python

Voici un extrait d'une implémentation en Python d'une classe modélisant un arbre binaire de recherche.

```
class ABR:
    """Implémentation d'un arbre binaire de recherche (ABR)"""
    def __init__(self, valeur=None):
        self.valeur = valeur
        self.fg = None
        self.fd = None

    def estVide(self):
        return self.valeur == None

    def insererElement(self, e):
        if self.estVide():
            self.valeur = e
        else:
            if e < self.valeur:
                if self.fg:
                    self.fg.insererElement(e)
                else:
                    self.fg = ABR(e)
            if e > self.valeur:
                if self.fd:
                    self.fd.insererElement(e)
                else:
                    self.fd = ABR(e)
```

- 1 Expliquer le rôle de la fonction `__init__`.

Dans cette implémentation, expliquer ce qui se passe si on ajoute un élément déjà présent dans l'arbre.

- 2 Recopier et compléter les lignes de code surlignées ci-dessous permettant de créer l'arbre de la partie A.

```
arbre = ABR(.....)
arbre.insererElement(2)
arbre.insererElement(.....)
arbre.insererElement(7)
arbre.insererElement(.....)
```

Partie C : Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

1. Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.
2. Comparer la complexité de cette méthode de tri avec celle du tri par insertion ou du tri par sélection.

Exercice 8 (E2S10)

Les Aventuriers du Rail© est un jeu de société dans lequel les joueurs doivent construire des lignes de chemin de fer entre différentes villes d'un pays.

La carte des liaisons possibles dans la région Occitanie est donnée en **annexe 1**. Dans l'**annexe 2**, les liaisons possédées par le joueur 1 sont en noir, et celles du joueur 2 en blanc. Les liaisons en gris sont encore en jeu.

Codages des structures de données utilisées :

- **Liste des liaisons d'un joueur** : Toutes les liaisons directes (sans ville intermédiaire) construites par un joueur seront enregistrées dans une variable de type "**tableau de tableaux**".

Le joueur 1 possède les lignes directes "Toulouse-Muret", "Toulouse-Montauban", "Gaillac-St Sulpice" et "Muret-Pamiers" dans l'**annexe 2** ["Gaillac","St Sulpice"], de mémorisées dans la variable ci-contre.]

```
liaisonsJoueur1 = [  
  ["Toulouse","Muret"],  
  ["Toulouse","Montauban"], (liaisons indiquées en noir  
  l'exercice 2). Ces liaisons sont ["Muret","Pamiers"]
```

Remarque : Seules les liaisons directes existent, par exemple ["Toulouse","Muret"] ou ["Muret","Toulouse"]. Par contre, le tableau ["Toulouse","Mazamet"] n'existe pas, puisque la ligne Toulouse-Mazamet passe par Castres.

- **Dictionnaire associé à un joueur** : On code la liste des villes et des trajets possédée par un joueur en utilisant un **dictionnaire de tableaux**. Chaque **clef de ce dictionnaire** est une ville de départ, et chaque **valeur** est un **tableau** contenant les villes d'arrivée possibles en fonction des liaisons possédées par le joueur.

Le dictionnaire de tableaux du joueur 1 est ["Toulouse"],

```
DictJoueur1 = {  
  "Toulouse":["Muret","Montauban"], donné ci-contre : "Montauban":  
    "Gaillac":["St Sulpice"],  
    "St Sulpice":["Gaillac"],  
    "Muret":  
      ["Toulouse","Pamiers"],  
      "Pamiers":["Muret"]}
```

- a) Expliquer pourquoi la liste des liaisons suivante n'est pas valide :
tableauliaisons = [["Toulouse","Auch"], ["Luchon","Muret"], ["Quillan","Limoux"]]
- b) Cette question concerne le joueur n°2 (*Rappel : les liaisons possédées par le joueur n°2 sont représentées par un rectangle blanc dans l'annexe 2*).
- c) Donner le tableau liaisonsJoueur2, des liaisons possédées par le joueur n°2.
- d) Recopier et compléter le dictionnaire suivant, associé au joueur n°2 :
- ```
DictJoueur2 = {
 "Toulouse":["Castres","Castelnaudary"],
 ...
}
```

3. À partir du tableau de tableaux contenant les liaisons d'un joueur, on souhaite construire le dictionnaire correspondant au joueur. Une première proposition a abouti à la fonction **construireDict** ci-dessous.



```

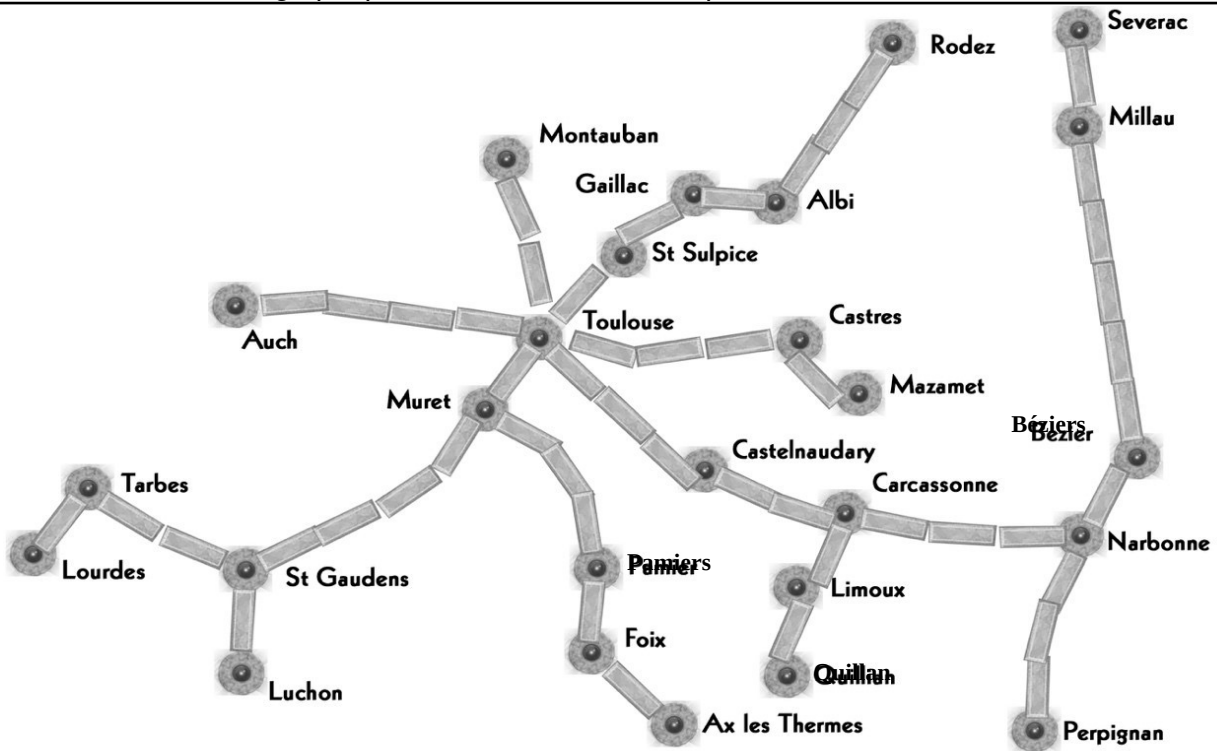
1 def construireDict(listeLiaisons):
2 """
3 listeLiaisons est un tableau de tableaux représentant la
4 liste des liaisons d'un joueur comme décrit dans le problème
5 """
6 Dict={}
7 for liaison in listeLiaisons :
8 villeA = liaison[0]
9 villeB = liaison[1]
10 if not villeA in Dict.keys() :
11 Dict[villeA]=[villeB]
12 else :
13 destinationsA = Dict[villeA]
14 if not villeB in destinationsA :
15 destinationsA.append(villeB)
16 return Dict

```

- Écrire sur votre copie un assert dans la fonction **construireDict** qui permet de vérifier que la **listeLiaisons** n'est pas vide.
- Sur votre copie, donner le résultat de cette fonction ayant comme argument la variable **liaisonsJoueur1** donnée dans l'énoncé et expliquer en quoi cette fonction ne répond que partiellement à la demande.
- La fonction construireDict, définie ci-dessus, est donc partiellement inexacte. Compléter la fonction construireDict pour qu'elle génère bien l'ensemble du dictionnaire de tableaux correspondant à la liste de liaisons données en argument. À l'aide des numéros de lignes, on précisera où est inséré ce code.

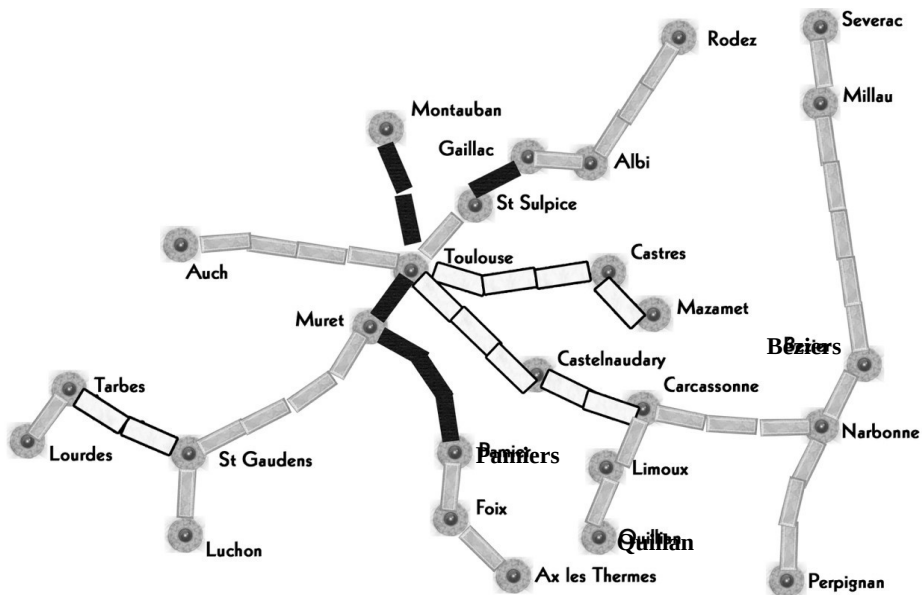
## Annexe 1

graphique 1 : Extrait des liaisons possibles en Occitanie



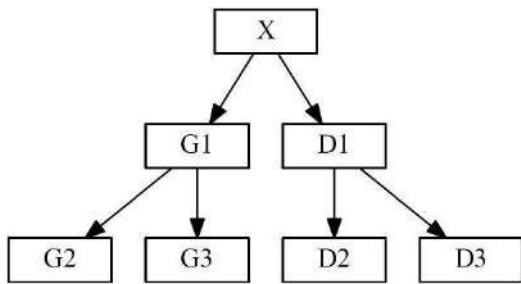
## Annexe 2

Liaisons possédées par le joueur 1 (en noir) et le joueur 2 (en blanc)



## Exercice 9 (E3S6)

Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit).



X est un nœud, sa valeur est X.valeur

G1 est le fils gauche de X, noté X.fils\_gauche

D1 est le fils droit de X, noté X.fils\_droit

Un arbre binaire de recherche est ordonné de la manière suivante :

Pour chaque nœud X,

- les valeurs de tous les nœuds du sous-arbre gauche sont strictement inférieures à la valeur du nœud X
- les valeurs de tous les nœuds du sous-arbre droit sont supérieures ou égales à la valeur du nœud X

Ainsi, par exemple, toutes les valeurs des nœuds G1, G2 et G3 sont strictement inférieures à la valeur du nœud X et toutes les valeurs des nœuds D1, D2 et D3 sont supérieures ou égales à la valeur du nœud X.

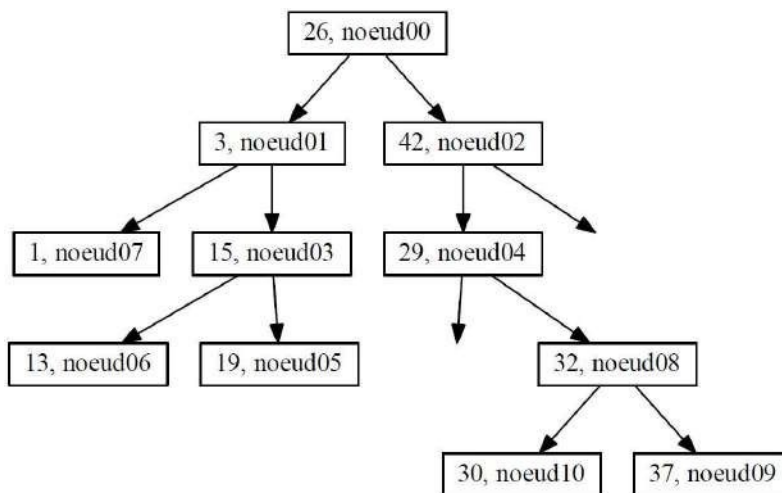
Voici un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les valeurs :

[26, 3, 42, 15, 29, 19, 13, 1, 32, 37, 30]

L'étiquette d'un nœud indique la valeur du nœud suivie du nom du nœud.

Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous.

'29, noeud04' signifie que le nœud nommé noeud04 possède la valeur 29.



- On insère la valeur 25 dans l'arbre, dans un nouveau nœud nommé noeud11. Recopier l'arbre binaire de recherche étudié et placer la valeur 25 sur cet arbre en coloriant en rouge le chemin parcouru. Préciser sous quel nœud la valeur 25 sera insérée et si elle est insérée en fils gauche ou en fils droit, et expliquer toutes les étapes de la décision.
- Préciser toutes les valeurs entières que l'on peut stocker dans le nœud fils gauche du nœud04 (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche ?

- Voici un algorithme récursif permettant de parcourir et d'afficher les valeurs de l'arbre :

Parcours(A) # A est un arbre binaire de recherche

Afficher(A.valeur)

Parcours(A.fils\_gauche)

Parcours(A.fils\_droit)

- Écrire la liste de toutes les valeurs dans l'ordre où elles seront affichées.
  - Choisir le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : Préfixe, Suffixe ou Infixe
- 
- En vous inspirant de l'algorithme précédent, écrire un algorithme Parcours2 permettant de parcourir et d'afficher les valeurs de l'arbre A dans l'ordre croissant.