EXERCICE 1 (7 points)

Cet exercice porte sur les bases de données et le langage SQL.

On considère une gestion simplifiée des voyages dans l'espace. La base de données utilisée est constituée de quatre relations nommées `Astronaute`, `Fusee`, `Equipe` et `Vol`. Voici le contenu des tables `Astronaute`, `Fusee`, `Equipe` et `Vol`.

Les clés primaires sont soulignées et les clefs étrangères sont précédées d'un `#` :

Astronaute				
id_astronaute	**nom**	**prenom**	**nationalite**	**nb_vols**
1	'PESQUET'	'Thomas'	'français'	2
2	'AMSTRONG'	'Neil'	'américain'	8
3	'MAURER'	'Mathias'	'allemand'	1
4	'MCARTHUR'	'Megan'	'américain'	5

Fusee			
id_fusee	**modele**	**constructeur**	**nb_places**
1	'Falcon 9'	'SpaceX'	6
2	'Starship'	'SpaceX'	100
3	'Soyouz'	'TsSKB Progress'	2
4	'SLS'	'Boeing'	6

Equipe	
id_vol	**#id_astronaute**
1	1
1	2
1	3
2	1
2	3
3	1
3	2
3	4
4	2
4	4

Vol		
id_vol	**#id_fusee**	**Date**
1	1	'12/09/2022'
2	4	'25/10/2022'
3	3	'18/11/2022'
4	2	'23/12/2022'

On pourra utiliser les mots clés suivants : `COUNT, FROM, INSERT, INTO, JOIN, ON, ORDER BY, SELECT, VALUES, WHERE.`

- Le mot clé `COUNT` permet de récupérer le nombre d'enregistrements issu de la requête.

Par exemple, la requête suivante renvoie la valeur 4.

```
SELECT COUNT (*) FROM Astronaute;
```

- Le mot clé `ORDER BY` permet de trier les éléments par ordre alphabétique. Par exemple, la requête suivante :

```
SELECT modele FROM Fusee ORDER BY modele;
```

renvoie la table

'Falcon 9'
'SLS'
'Soyouz'
'Starship'

1. On s'intéresse ici à la notion de clés primaire et étrangère.

a. Donner la définition d'une clé primaire.

b. Dans la table `Astronaute`, la clé primaire est `id_astronaute`. Expliquer pourquoi cette requête SQL renvoie une erreur :

```
INSERT INTO Astronaute
VALUES (3, 'HAIGNERE', 'Claudie', 'français', 3);
```

c. Le schéma relationnel de la table `Astronaute` est :
`Astronaute (id astronaute : INT, nom : TEXT, prenom : TEXT, nationalite : TEXT, nb_vols : INT)`

Écrire le schéma relationnel de la table `Fusee` en précisant le domaine de chaque attribut.

2. On s'intéresse ici à la récupération d'informations issues de la base de données.

a. Écrire le résultat que la requête suivante renvoie :

```
SELECT COUNT(*)
FROM Fusee
WHERE constructeur = 'SpaceX';
```

b. Écrire une requête SQL qui renvoie le modèle et le constructeur des fusées ayant au moins quatre places.

c. Écrire une requête SQL qui renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom.

3.

a. Recopier et compléter les requêtes SQL suivantes permettant d'ajouter un cinquième vol avec la fusée 'Soyouz' le 12/04/2023 avec l'équipage composé de PESQUET Thomas et MCARTHUR Megan. On ne s'intéresse pas ici à la mise à jour qui suivra.

```
INSERT INTO Vol VALUES( ..... );
INSERT INTO Equipe VALUES( ..... );
INSERT INTO …… VALUES( ..... );
```

b. Écrire une requête SQL permettant d'obtenir le nom et le prénom des astronautes ayant décollé le '25/10/2022'.

EXERCICE 2 (7 points)

Cet exercice porte sur les files

On considère une structure de données file que l'on représentera par des éléments en ligne, l'élément à droite étant la tête de la file et l'élément à gauche étant la queue de la file.

On appellera `f1` la file suivante :

	'bac'	'nsi'	'2023'	'file'	

On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

- `creer_file()` : renvoie une file vide ;
- `est_vide(f)` : renvoie `True` si la file `f` est vide et `False` sinon ;
- `enfiler(f, e)` : ajoute l'élément `e` à la queue de la file `f` ;
- `defiler(f)` : renvoie l'élément situé à la tête de la file `f` et le retire de la file.

Les trois questions suivantes sont indépendantes.

a. Donner le résultat renvoyé après l'appel de la fonction `est_vide(f1)`.

b. Représenter la file `f1` après l'exécution du code `defiler(f1)`.

c. Représenter la file `f2` après l'exécution du code suivant :

```
1  f2 = creer_file()
2  liste = ['castor', 'python', 'poule']
3  for elt in liste:
4        enfiler(f2, elt)
```

1. Recopier et compléter les lignes 4, 6 et 7 de la fonction `longueur` qui prend en paramètre une file `f` et qui renvoie le nombre d'éléments qu'elle contient. Après un appel à la fonction, la file `f` doit retrouver son état d'origine.

```
1   def longueur(f):
2       resultat = 0
3       g = creer_file()
4       while ... :
5           elt = defiler(f)
6           resultat = ...
7           enfiler(... , ...)
8       while not(est_vide(g)):
9           enfiler(f, defiler(g))
10      return resultat
```

2. Un site impose à ses clients des critères sur leur mot de passe. Pour cela il utilise la fonction `est_valide` qui prend en paramètre une chaine de caractères `mot` et qui retourne `True` si `mot` correspond aux critères et `False` sinon.

```
1  def est_valide(mot):
2      if len(mot) < 8:
3          return False
4      for c in mot:
5          if c in ['!', '#', '@', ';', ':']:
6              return True
7      return False
```

Parmi les mots de passe suivants, recopier celui qui sera validé par cette fonction.

A - `'best@'`

B - `'paptap23'`

C - `'2!@59fgds'`

3. Le tableau suivant montre, sur deux exemples, l'évolution d'une file `f3` après l'exécution de l'instruction `ajouter_mot(f3, 'super')` :

	état initial de `f3`	**état de f3 après l'instruction ajouter_mot(f3, 'super')**
Exemple 1	`'bac'` `'nsi'` `'2023'`	`'super'` `'bac'` `'nsi'`
Exemple 2	`'test'` `'info'`	`'super'` `'test'` `'info'`

Écrire le code de cette fonction `ajouter_mot` qui prend en paramètres une file `f` (qui a au plus 3 éléments) et une chaine de caractères valide `mdp`. Cette fonction met à jour la file de stockage `f` des mots de passe en y ajoutant `mdp` et en défilant, si nécessaire, pour avoir au maximum trois éléments dans cette file.

On pourra utiliser la fonction `longueur` de la question 1.

4. Pour intensifier sa sécurité, le site stocke les trois derniers mots de passe dans une file et interdit au client lorsqu'il change son mot de passe d'utiliser l'un des mots de passe stockés dans cette file.

Recopier et compléter les lignes 7 et 8 de la fonction `mot_file` :

- qui prend en paramètres une file `f` et `mdp` de type chaine de caractères ;

- qui renvoie `True` si le mot de passe est un élément de la file `f` et `False` sinon.

Après un appel à cette fonction, la file `f` doit retrouver son état d'origine.

```
1   def mot_file(f, mdp):
2       g = creer_file()
3       present = False
4       while not(est_vide(f)):
5           elt = defiler(f)
6           enfiler(g, elt)
7           if ...:
8               present = ...
9       while not(est_vide(g)):
10          enfiler(f, defiler(g))
11      return present
```

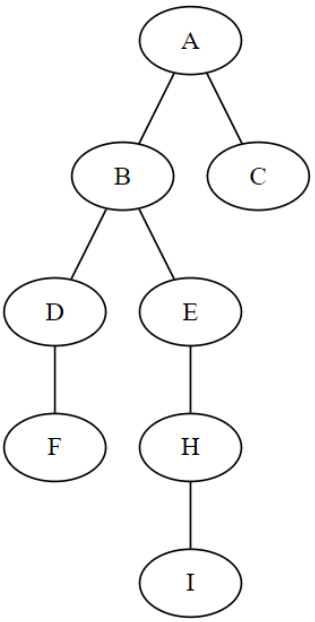
5. Écrire une fonction `modification` qui prend en paramètres une file `f` et une chaine de caractères `nv_mdp`. Si le mot de passe `nv_mdp` répond bien aux **deux** exigences des questions 2 et 4, alors elle modifie la file des mots de passe stockés et renvoie `True`. Dans le cas contraire, elle renvoie `False`.
On pourra utiliser les fonctions `mot_file`, `est_valide` et `ajouter_mot`.

EXERCICE 3 (6 points)

Cet exercice porte sur les arbres

1. Quelles sont les principales caractéristiques d'un arbre ?
2. Donner la définition de la **taille** d'un arbre.
3. Donner la définition d'un arbre **équilibré**.

4. On considère l'arbre suivant



Donner la taille et la hauteur de cet arbre.

5. On nomme h la hauteur d'un arbre binaire.

Donner en fonction de h :

a) La taille minimale d'un arbre binaire.

b) La taille maximale d'un arbre binaire.

c) Comment nomme-t-on un arbre binaire de taille maximale ?

Cette partie traite des arbres binaires de recherche (ABR)

6. Donner la définition d'un arbre binaire de recherche (ABR).
7. Dessiner tous les arbres binaires de recherche formés des noeuds 1, 2 et 3.
8. Où se trouve le minimum d'un arbre binaire de recherche

On dispose d'un arbre binaire de recherche A représenté par la liste [racine(A),SAG(A),SAD(A)] où :

- racine(A) est la fonction qui retourne la racine de l'arbre A;
- SAG(A) est la fonction qui retourne le sous arbre gauche de l'arbre A;
- SAD(A) est la fonction qui retourne le sous arbre droit de l'arbre A.
- est_vide(A) est une fonction qui retourne True si l'arbre A est vide et False sinon.

9. Ecrire sur votre feuille la fonction récursive Python **minimum(A)** qui retourne le minimum de l'arbre binaire de recherche A.

EXERCICE 1 (7 points)

Cet exercice porte sur les bases de données et le langage SQL.

On considère une gestion simplifiée des voyages dans l'espace. La base de données utilisée est constituée de quatre relations nommées `Astronaute`, `Fusee`, `Equipe` et `Vol`. Voici le contenu des tables `Astronaute`, `Fusee`, `Equipe` et `Vol`.

Les clés primaires sont soulignées et les clefs étrangères sont précédées d'un `#` :

Astronaute				
id_astronaute	**nom**	**prenom**	**nationalite**	**nb_vols**
1	'PESQUET'	'Thomas'	'français'	2
2	'AMSTRONG'	'Neil'	'américain'	8
3	'MAURER'	'Mathias'	'allemand'	1
4	'MCARTHUR'	'Megan'	'américain'	5

Fusee			
id_fusee	**modele**	**constructeur**	**nb_places**
1	'Falcon 9'	'SpaceX'	6
2	'Starship'	'SpaceX'	100
3	'Soyouz'	'TsSKB Progress'	2
4	'SLS'	'Boeing'	6

Equipe	
id_vol	**#id_astronaute**
1	1
1	2
1	3
2	1
2	3
3	1
3	2
3	4
4	2
4	4

Vol		
id_vol	**#id_fusee**	**Date**
1	1	'12/09/2022'
2	4	'25/10/2022'
3	3	'18/11/2022'
4	2	'23/12/2022'

On pourra utiliser les mots clés suivants : `COUNT, FROM, INSERT, INTO, JOIN, ON, ORDER BY, SELECT, VALUES, WHERE.`

- Le mot clé `COUNT` permet de récupérer le nombre d'enregistrements issu de la requête.

Par exemple, la requête suivante renvoie la valeur 4.

```
SELECT COUNT (*) FROM Astronaute;
```

- Le mot clé `ORDER BY` permet de trier les éléments par ordre alphabétique. Par exemple, la requête suivante :

```
SELECT modele FROM Fusee ORDER BY modele;
```

renvoie la table

'Falcon 9'
'SLS'
'Soyouz'
'Starship'

1. On s'intéresse ici à la notion de clés primaire et étrangère.

a. Donner la définition d'une clé primaire.

Une clef primaire est un identifiant unique d'une ligne d'une table d'une base de données relationnelle.

b. Dans la table `Astronaute`, la clé primaire est `id_astronaute`. Expliquer pourquoi cette requête SQL renvoie une erreur :

```
INSERT INTO Astronaute
VALUES (3, 'HAIGNERE', 'Claudie', 'français', 3);
```

Un enregistrement est déjà présent dans la table Fusee avec la clef primaire id_astronaute ayant pour valeur 3.
Il n'est donc pas possible d'insérer dans cette table un nouvel enregistrement avec la même clef ayant pour valeur 3.

c. Le schéma relationnel de la table `Astronaute` est :
`Astronaute (id_astronaute : INT, nom : TEXT, prenom : TEXT, nationalite : TEXT, nb_vols : INT)`

Écrire le schéma relationnel de la table `Fusee` en précisant le domaine de chaque attribut.

`Fusee (id_fusee : INT, modele : TEXT, constructeur : TEXT, nb_places : INT)`

2. On s'intéresse ici à la récupération d'informations issues de la base de données.

a. Écrire le résultat que la requête suivante renvoie :

```
SELECT COUNT(*)
FROM Fusee
WHERE constructeur = 'SpaceX';
```

La requête renvoie le nombre 2.

CORRECTION

Elle renvoie le nombre de fusées dont le constructeur est Space X.

b. Écrire une requête SQL qui renvoie le modèle et le constructeur des fusées ayant au moins quatre places.

```
SELECT modele, constructeur
From Fusee
WHERE nb_places >= 4
```

c. Écrire une requête SQL qui renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom.

```
SELECT nom,prenom
FROM Astronaute
ORDER BY nom
```

3.

a. Recopier et compléter les requêtes SQL suivantes permettant d'ajouter un cinquième vol avec la fusée 'Soyouz' le 12/04/2023 avec l'équipage composé de PESQUET Thomas et MCARTHUR Megan. On ne s'intéresse pas ici à la mise à jour qui suivra.

```
INSERT INTO Vol VALUES(5,3,'12/04/2023');
INSERT INTO Equipe VALUES(5,1)
INSERT INTO Equipe VALUES(5,4)
```

b. Écrire une requête SQL permettant d'obtenir le nom et le prénom des astronautes ayant décollé le '25/10/2022'.

```
SELECT nom,prenom

FROM Astronaute,Vol,Equipe

WHERE vol.id_vol = equipe.id_vol AND Astronaute.id_astronaute = Equipe.id_astronaute

AND date = '25/10/2022'

Ou

SELECT nom,prenom

FROM Astronaute

JOIN Equipe ON Astronaute.id_astronaute = Equipe.id_astronau6te

JOIN VOL ON vol.id_vol = equipe.id_vol and date = '25/10/2022'
```

EXERCICE 2 (7 points)

Cet exercice porte sur les files

On considère une structure de données file que l'on représentera par des éléments en ligne, l'élément à droite étant la tête de la file et l'élément à gauche étant la queue de la file.

On appellera `f1` la file suivante :

	'bac'	'nsi'	'2023'	'file'	

On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

- `creer_file()` : renvoie une file vide ;
- `est_vide(f)` : renvoie `True` si la file `f` est vide et `False` sinon ;
- `enfiler(f, e)` : ajoute l'élément `e` à la queue de la file `f` ;
- `defiler(f)` : renvoie l'élément situé à la tête de la file `f` et le retire de la file.

1.

Les trois questions suivantes sont indépendantes.

a. Donner le résultat renvoyé après l'appel de la fonction `est_vide(f1)`.

est_vide(f) retourne False car la file f n'est pas vide.

b. Représenter la file `f1` après l'exécution du code `defiler(f1)`.

Après l'exécution defiler(f1) la file f1 est :

	'bac'	'nsi'	'2023'	

c. Représenter la file `f2` après l'exécution du code suivant :

```
1  f2 = creer_file()
2  liste = ['castor', 'python', 'poule']
3  for elt in liste:
4        enfiler(f2, elt)
```

après l'exécution du code précédent, la file f2 est :

'poule'	'python'	'castor'

2. Recopier et compléter les lignes 4, 6 et 7 de la fonction `longueur` qui prend en paramètre une file `f` et qui renvoie le nombre d'éléments qu'elle contient. Après un appel à la fonction, la file `f` doit retrouver son état d'origine.

CORRECTION

```
1   def longueur(f):
2       resultat = 0
3       g = creer_file()
4       while not(est_vide(f)):
5           elt = defiler(f)
6           resultat = resultat + 1
7           enfiler(g , elt)
8       while not(est_vide(g)):
9           enfiler(f, defiler(g))
10      return resultat
```

3. Un site impose à ses clients des critères sur leur mot de passe. Pour cela il utilise la fonction `est_valide` qui prend en paramètre une chaine de caractères `mot` et qui retourne `True` si `mot` correspond aux critères et `False` sinon.

```
1   def est_valide(mot):
2       if len(mot) < 8:
3           return False
4       for c in mot:
5           if c in ['!', '#', '@', ';', ':']:
6               return True
7       return False
```

Parmi les mots de passe suivants, recopier celui qui sera validé par cette fonction.

a) - `'best@'`

b) - `'paptap23'`

c) - `'2!@59fgds'`

Le mot de passe validé par la fonction es_valide(mot) est '2!@59fgds'

4. Le tableau suivant montre, sur deux exemples, l'évolution d'une file `f3` après l'exécution de l'instruction `ajouter_mot(f3, 'super')` :

	état initial de `f3`	**état de f3 après l'instruction ajouter_mot(f3, 'super')**
Exemple 1	`'bac'` `'nsi'` `'2023'`	`'super'` `'bac'` `'nsi'`
Exemple 2	`'test'` `'info'`	`'super'` `'test'` `'info'`

Écrire le code de cette fonction `ajouter_mot` qui prend en paramètres une file `f` (qui a au plus 3 éléments) et une chaine de caractères valide `mdp`. Cette fonction met à jour la file de stockage `f` des mots de passe en y ajoutant `mdp` et en défilant, si nécessaire, pour avoir au maximum trois éléments dans cette file.

On pourra utiliser la fonction `longueur` de la question 1.

```
def ajouter_mot(f,mdp):
    enfiler(f,mdp)
    while longueur(f)>3:
        defiler(f)
```

5. Pour intensifier sa sécurité, le site stocke les trois derniers mots de passe dans une file et interdit au client lorsqu'il change son mot de passe d'utiliser l'un des mots de passe stockés dans cette file.

Recopier et compléter les lignes 7 et 8 de la fonction `mot_file` :

- qui prend en paramètres une file `f` et `mdp` de type chaine de caractères ;
- qui renvoie `True` si le mot de passe est un élément de la file `f` et `False` sinon.

Après un appel à cette fonction, la file `f` doit retrouver son état d'origine.

```
1   def mot_file(f, mdp):
2       g = creer_file()
3       present = False
4       while not(est_vide(f)):
5           elt = defiler(f)
6           enfiler(g, elt)
7           if mdp == elt:
8               present = True
9       while not(est_vide(g)):
10          enfiler(f, defiler(g))
11      return present
```

6. Écrire une fonction `modification` qui prend en paramètres une file `f` et une chaine de caractères `nv_mdp`. Si le mot de passe `nv_mdp` répond bien aux **deux** exigences des questions 2 et 4, alors elle modifie la file des mots de passe stockés et renvoie `True`. Dans le cas contraire, elle renvoie `False`.

On pourra utiliser les fonctions `mot_file`, `est_valide` et `ajouter_mot`.

```
def modification(f,nv_mdp):
    if est_valide(nv_mdp) and not(mot_file(f,nv_mdp)):
        ajouter_mot(f,nv_mdp)
        return True
    return False
```

EXERCICE 3 (6 points)

Cet exercice porte sur les arbres

1. Quelles sont les principales caractéristiques d'un arbre ?

   - Sa **taille** : le nombre de nœuds
   - Son **arité** : le nombre maximal d'enfants qu'un nœud peut avoir
   - Sa **hauteur** : le nombre de nœuds moins 1 de la branche la plus longue.

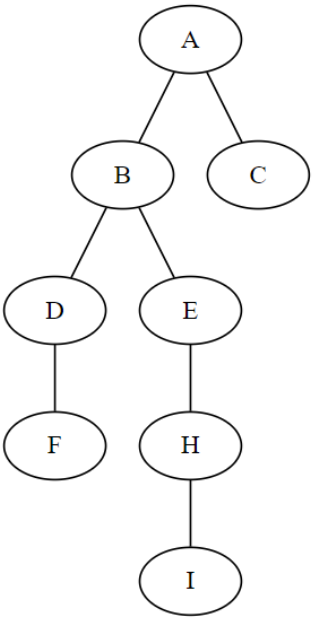
2. Donner la définition de la **taille** d'un arbre.

   Taille : le nombre de nœuds de l'arbre

3. Donner la définition d'un arbre **équilibré**.

   Un arbre équilibré est un arbre binaire de recherche qui maintient une profondeur équilibrée entre ses branches. La différence de longueur entre sa branche la plus longue et la branche la moins longue est au plus égale à 1.)

4. On considère l'arbre suivant



   Donner la taille et la hauteur de cet arbre.

   Taille = 8

   Hauteur = 4

5. On nomme h la hauteur d'un arbre binaire.

Donner en fonction de h :

a) La taille minimale d'un arbre binaire : h + 1 (arbre dégénéré réduit à une branche)

b) La taille maximale d'un arbre binaire. $2^{(h+1)} - 1$

c) Comment nomme-t-on un arbre binaire de taille maximale ? : un arbre complet

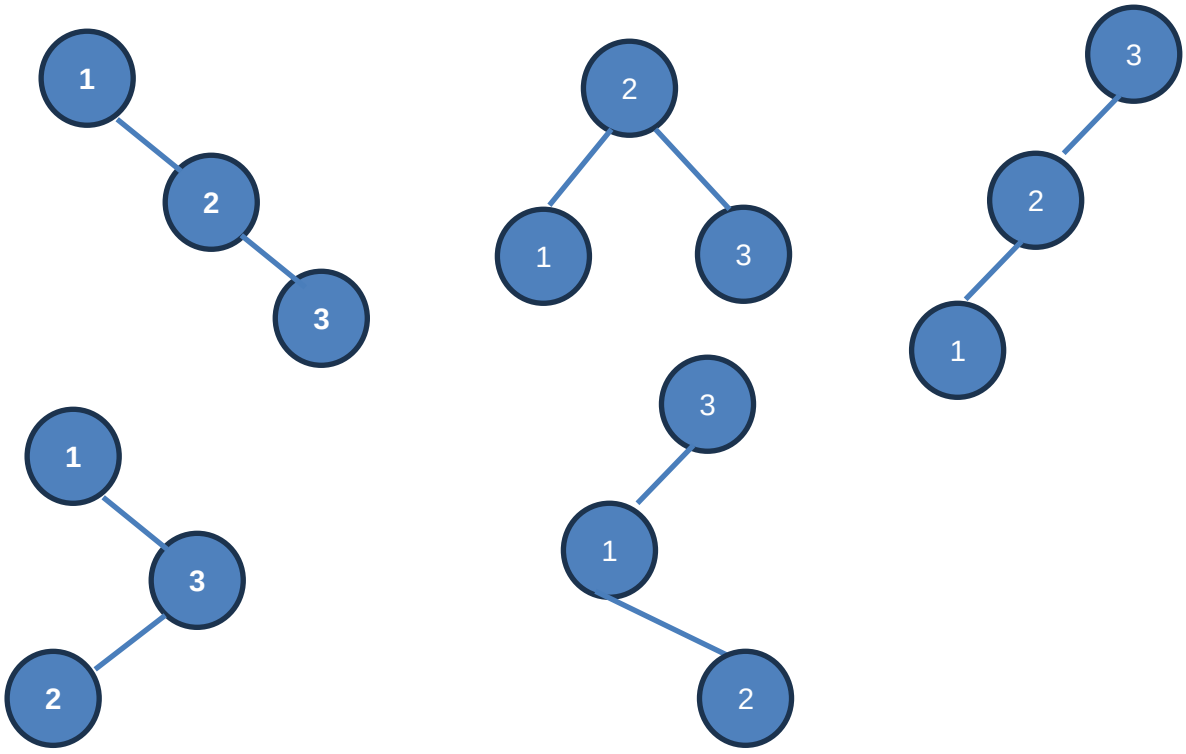
Cette partie traite des arbres binaires de recherche (ABR)

6. Donner la définition d'un arbre binaire de recherche (ABR).

Un arbre binaire de recherche est un arbre défini comme suit :

- Les étiquettes des nœuds sont appelées des clés.
- Les clés de tous les nœuds d'un sous-arbre gauche d'un nœud X sont inférieures ou égales à la clé de X.
- Les clés de tous les nœuds d'un sous-arbre droit d'un nœud X sont strictement supérieures à la clé de X.

7. Dessiner tous les arbres binaires de recherche formés des noeuds 1, 2 et 3.



8. Où se trouve le minimum d'un arbre binaire de recherche

C'est la feuille de la branche la plus à gauche de l'arbre.

9. On dispose d'un arbre binaire de recherche A représenté par la liste [racine(A),SAG(A),SAD(A)] où :

- racine(A) est la fonction qui retourne la racine de l'arbre A;
- SAG(A) est la fonction qui retourne le sous arbre gauche de l'arbre A;
- SAD(A) est la fonction qui retourne le sous arbre droit de l'arbre A.
- est_vide(A) est une fonction qui retourne True si l'arbre A est vide et False sinon.

Ecrire sur votre feuille la fonction récursive Python **minimum(A)** qui retourne le minimum de l'arbre binaire de recherche A.

```
def minimum(A) :
    if est_vide(SAG(A)) and est_vide(SAD(A)) :
        return racine(A)
    return minimum(SAG(A))
```