Exercice 1 (6 points)

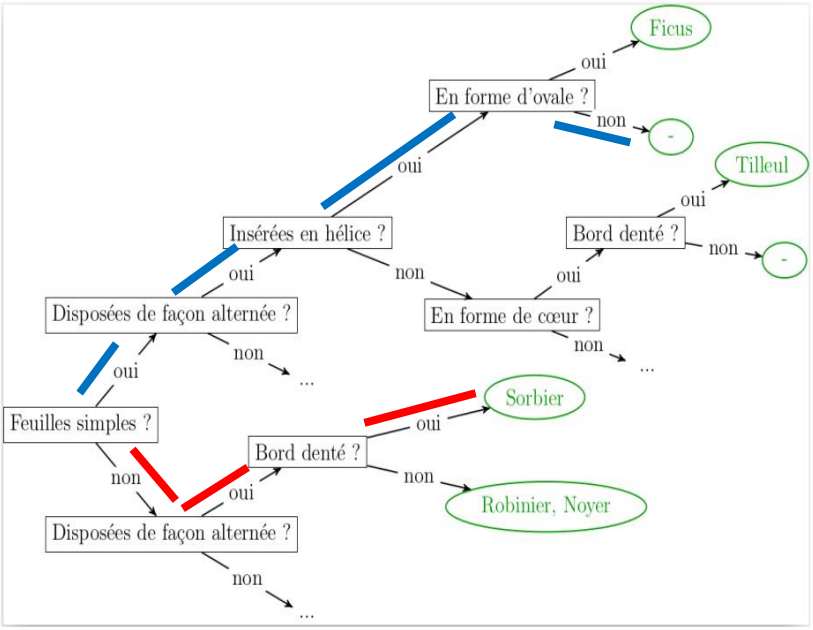
Cet exercice porte sur les arbres binaires, la récursivité et la programmation orientée objet

1. On observe un végétal dont les *folia* sont **complexes (non simples)**, disposées de **façon alternée** et **à bord denté**. D'après l'arbre de décision de la figure 1, peut-on identifier ce végétal ? Si oui, quel est-il ?

On peut identifier le sorbier.

2. On observe un végétal dont les *folia* sont **simples**, disposées de **façon alternée**, insérées **en hélice** et ne sont **pas de forme d'ovale**. D'après l'arbre de décision de la figure 1, peut-on identifier ce végétal ? Si oui, quel est-il ?

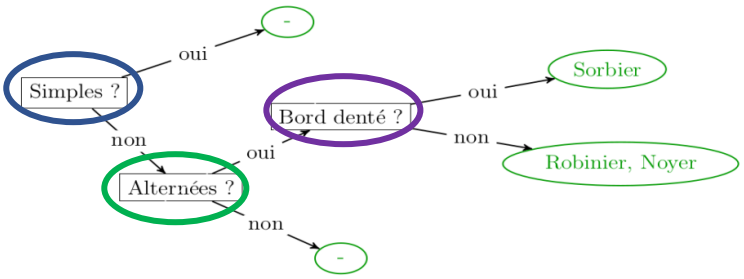
Non on ne peut pas identifier ce végétal.



3. Écrire en langage Python le code permettant de construire l'arbre de décision de la *figure 2* ci-dessous et de l'affecter à une variable nommée `arbre_2`.

```
1 class Noeud:
2     def __init__(self, question, sioui, sinon):
3           self.question = question
4           self.sioui = sioui
5           self.sinon = sinon

1 class Feuille_resultat:
2     def __init__(self, vegetaux):
3           self.vegetaux = vegetaux
```



Solution :

```
arbre_2 = Noeud( "Simples ?" ,
                  Feuille_resultat([]),
                  Noeud( "Alternées ?" ,
                         Noeud( "Bord denté ?",
                                Feuille_resultat(["Sorbier"]),
                                Feuille_resultat (["Robinier", "Noyer"])
                              ),
                  Feuille_resultat([]))
                )
```

4. Écrire le code de la méthode `est_resultat` pour la classe `Nœud`.

```
1 class Noeud:
2     def __init__(self, question, sioui, sinon):
3           self.question = question
4           self.sioui = sioui
5           self.sinon = sinon
6     def est_resultat(self):
7           return False
```

5. Écrire le code de la méthode `est_resultat` pour la classe `Feuille_resultat`.

```
1 class Feuille_resultat:
2     def __init__(self, vegetaux):
3           self.vegetaux = vegetaux
4     def est_resultat(self):
5           return True
```

On souhaite connaître le nombre de végétaux identifiables par un arbre de décision.

6. Écrire le code de la méthode `nb_vegetaux` pour la classe `Feuille_resultat`.

```
1 class Feuille_resultat:
2     def __init__(self, vegetaux):
3           self.vegetaux = vegetaux
4     def est_resultat(self):
5           return True
6     def nb_vegetaux(self):
7           return len(self.vegetaux)
```

7. Écrire le code de la méthode `nb_vegetaux` pour la classe `Noeud`, qui prend en compte tous les végétaux identifiables à partir de ce nœud.

Additionne récursivement le nombre de végétaux dans les sous-arbres sioui et sinon.

```
1 class Noeud:
2     def __init__(self, question, sioui, sinon):
3           self.question = question
4           self.sioui = sioui
5           self.sinon = sinon
6     def est_resultat(self):
7           return False
8     def nb_vegetaux(self):
9           return self.sioui.nb_vegetaux() + self.sinon.nb_vegetaux()
```

On souhaite enfin écrire une méthode `liste_questions` pour chacune des classes `Noeud` et `Feuille_resultat` afin d'obtenir la liste des questions présentes dans un arbre de décision. L'ordre des éléments dans cette liste n'a pas d'importance, de plus elle peut contenir des doublons. On remarque que :

- si `f` est un objet de la classe `Feuille_resultat`, alors `f.liste_questions()` est la liste vide ;
- le résultat de l'appel `arbre_2.liste_questions()` est la liste `['Simples?', 'Alternées? ', 'Bord denté ?']` (ou une liste avec les mêmes éléments mais dans un ordre différent).

8. Écrire le code de la méthode `liste_questions` pour la classe `Feuille_resultat`

```
1 class Feuille_resultat:
2     def __init__(self, vegetaux):
3           self.vegetaux = vegetaux
4     def est_resultat(self):
5           return True
6     def nb_vegetaux(self):
7           return len(self.vegetaux)
8     def liste_questions(self):
9        return []
```

9. Écrire le code de la méthode `liste_questions` pour la classe `Noeud`, qui prend en compte toutes les questions accessibles à partir de ce nœud. On rappelle que l'opérateur `+` en Python permet de concaténer des listes, par exemple la valeur de l'expression `[1,2]+[3,4,5]` est la liste `[1,2,3,4,5]`.

```
1 class Noeud:
2     def __init__(self, question, sioui, sinon):
3           self.question = question
4           self.sioui = sioui
5           self.sinon = sinon
6     def est_resultat(self):
7           return False
8     def nb_vegetaux(self):
9           return self.sioui.nb_vegetaux() + self.sinon.nb_vegetaux()
10    def liste_questions(self):
            return [self.question] +
                    self.sioui.liste_questions() +
                    self.sinon.liste_questions()
```

Pour représenter les caractéristiques des *folia* d'un végétal, on utilise un dictionnaire. Les clés du dictionnaire sont les questions de l'arbre de décision et les valeurs sont `True` ou `False` selon la réponse.

Par exemple, le dictionnaire décrivant les *folia* du sorbier pour l'arbre de décision de la figure 2 est le dictionnaire `folia_sorbier` défini ci-dessous.

```
1 folia_sorbier = {
2     'Simples ?': False,
3     'Alternées ?': True,
4     'Bord denté ?': True
5 }
```

En revanche le dictionnaire `folia_tilleul` ci-dessous qui décrit (partiellement) les *folia* du tilleul n'est pas adapté pour l'arbre de décision de la figure 1 car des données sont manquantes. Par exemple, la question `'Feuilles simples ?'` n'est pas une clé de ce dictionnaire alors que c'est une question présente dans l'arbre.

```
1 folia_tilleul = {
2     'En forme d'ovale ?': False,
3     'Disposées de façon alternée ?': True,
4     'Bord denté ?': True
5      }
```

10. Écrire une fonction `est_bien_renseigne` qui prend en paramètres :

– un dictionnaire `dico_vegetal` qui donne les caractéristiques des *folia* d'un végétal,

– un arbre de décision représenté par un objet `arbre` de la classe `Feuille_resultat` ou de la classe `Noeud`,

et qui renvoie `True` si toutes les questions présentes dans `arbre` sont des clés de `dico_ vegetal`.

```
def est_bien_renseigne(dico_vegetal, arbre):
    # Récupère toutes les questions de l'arbre (peut contenir des doublons)
    questions = arbre.liste_questions()

    # Vérifie que chaque question de l'arbre est bien une clé du
    dictionnaire
    for question in questions:
        if question not in dico_vegetal:
            return False
    return True
```

11. Écrire une fonction `identifier_vegetaux` qui prend en paramètres :

– un dictionnaire `dico_vegetal` qui donne les caractéristiques des *folia* d'un végétal,

– un arbre de décision représenté par un objet `arbre` de la classe `Feuille_resultat` ou de la classe `Noeud`,

et qui renvoie la liste, éventuellement vide, des noms des végétaux dont les *folia* correspondent aux caractéristiques du dictionnaire.

Par exemple l'appel `identifier_vegetaux(arbre_2,folia_sorbier)` devra renvoyer la liste `['Sorbier']`.

On suppose que toutes les questions de l'arbre de décision `arbre` apparaissent comme des clés dans le dictionnaire `dico_vegetal`.

Il faut parcourir **l'arbre de décision** selon les réponses `(True/False)` présentes dans un dictionnaire `dico_vegetal`, pour **obtenir la liste de végétaux** correspondant aux caractéristiques fournies.

```
def identifier_vegetaux(dico_vegetal, arbre):
    if arbre.est_resultat():
        return arbre.vegetaux
    else:
        reponse = dico_vegetal[arbre.question]
        if reponse == True:
            return identifier_vegetaux(dico_vegetal, arbre.sioui)
        else:
            return identifier_vegetaux(dico_vegetal, arbre.sinon)
```