Exercice 1 :

1. — préfixe : × - 12 8 + 7 9
   — infixe : 12 - 8 × 7 + 9
   — postfixe : 12 8 - 7 9 + ×
2. 64
3. Parcours infixe

Exercice 2 :

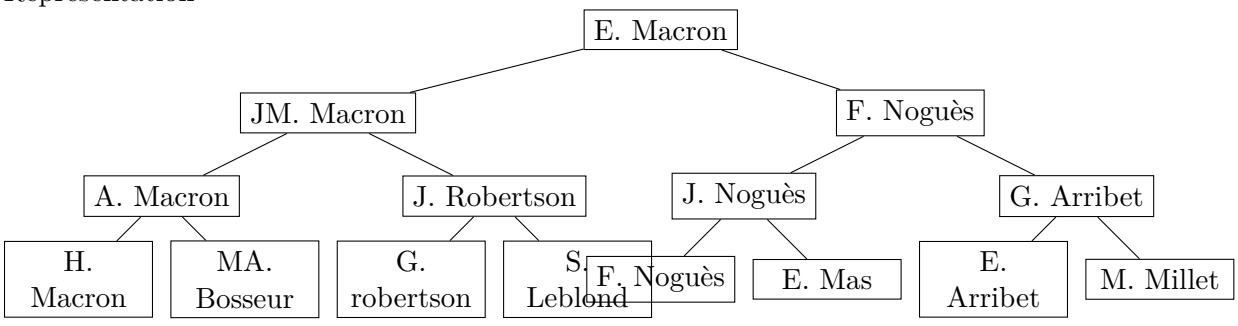
1. — en largeur : 1 2 3 4 5 6 7 8 9 10 11 12 13
   — préfixe : 1 2 4 8 5 3 6 9 10 12 13 7 11
   — infixe : 4 8 2 5 1 9 6 12 10 13 3 11 7
   — postfixe : 8 4 5 2 9 12 13 10 6 11 7 3 1
2. La hauteur est 4.
3. Cet arbre est équilibré car la hauteur de chaque sous-arbre gauche diffère au plus de 1 de chaque sous-arbre droit.
4. Cet arbre n'est pas complet car tous les niveaux ne sont pas remplis.

Exercice 3 :

1. Le numéro 17 est une femme (indice impair). Son père a pour indice 34 et sa mère 35. Son enfant a pour indice 8.
2. Quatrième génération : $2^4 = 16$ personnes (la numérotation commence à 1).
3. Chaque niveau i contient 2^{i-1} ascendants (la numérotation commence à 1). La somme de tous les niveaux correspond à la somme de termes d'une suite géométrique de raison 2 et de premier terme 1.

$$2^0 + 2^1 + 2^2 + 2^3 + 2^4 = \sum_{k=0}^{4} 2^k = \frac{1 - 2^{4+1}}{1 - 2} = 31$$

4. Représentation



5. Ouvrir le fichier *arbre-genealogique.py*.
6. Les parents

```
1 def get_parents(tab: list, id_enfant: int)->tuple:
2     # vérifie si on est encore dans le tableau
3     assert 2*id_enfant < len(tab), "Nous ne sommes pas remontés
          aussi loin."
4     return(tab[2*id_enfant], tab[2*id_enfant+1])
```

7. Pour parcourir le sous-arbre gauche en premier il faut empiler d'abord l'indice impair.

Christophe Viroulaud

```python
def ascendant_homme(tab: list, hommes: list)->list:
    p = []
    p.append(1)
    while len(p) > 0:
        en_cours = p.pop()
        if en_cours < len(tab):
            # enregistre ascendant hommes
            if en_cours%2 == 0 and en_cours > 1:
                hommes.append(tab[en_cours])

            p.append(2*en_cours+1)
            p.append(2*en_cours)
    return hommes
```

8. Dans ce cas c'est l'indice pair qui est utilisé en premier dans les appels.

```python
def ascendant_homme_rec(tab: list, hommes: list, en_cours: int = 1)
   ->list:
    if en_cours < len(tab):
        # enregistre ascendant hommes
        if en_cours%2 == 0 and en_cours > 1:
            hommes.append(tab[en_cours])

        ascendant_homme_rec(tab, hommes, 2*en_cours)
        ascendant_homme_rec(tab, hommes, 2*en_cours+1)
    return hommes
```

Exercice 4 :

1. Class

```python
class Arbre_binaire:

    def __init__(self, h: int)->None:
        """
        Initialise un tableau correspondant àla hauteur 'h' de l'
           arbre
        Le premier noeud est 'r'
        """
        self.hauteur = h
        self.arbre= [None for _ in range(2**(h+1) - 1)]
        self.arbre[0] = "r"
```

2. Insertion

```python
    def inserer(self, pere: str, fils_g: str, fils_d: str)->None:
        """
        insère les fils gauche et droit du noeud père
        """
        i_pere = self.arbre.index(pere)
        #vérification de sortie de l'arbre
        assert 2*i_pere < len(self.arbre), "Ce noeud n'a pas de fils"
        self.arbre[2*i_pere + 1] = fils_g
```


Christophe Viroulaud

```
9           self.arbre[2*i_pere + 2] = fils_d
```

3. Création

```
1  arbre_car = Arbre_binaire(4)
2  arbre_car.inserer("r", "a", "b")
3  arbre_car.inserer("a", "c", "d")
4  arbre_car.inserer("c", "g", "h")
5  arbre_car.inserer("d", "i", "j")
6  arbre_car.inserer("j", "l", None)
7  arbre_car.inserer("b", "e", "f")
8  arbre_car.inserer("e", "k", None)
```

4. Préfixe

```
1      def prefixe(self, parcours: list, i: int = 0)->list:
2          if i >= len(self.arbre) or self.arbre[i] is None:
3              return
4          else:
5              parcours.append(self.arbre[i])
6              self.prefixe(parcours, 2*i+1)
7              self.prefixe(parcours, 2*i+2)
8          return parcours
```

5. Infixe et postfixe

```
1      def infixe(self, parcours: list, i: int = 0)->list:
2          if i >= len(self.arbre) or self.arbre[i] is None:
3              return
4          else:
5              self.infixe(parcours, 2*i+1)
6              parcours.append(self.arbre[i])
7              self.infixe(parcours, 2*i+2)
8          return parcours
```

```
1      def postfixe(self, parcours: list, i: int = 0)->list:
2          if i >= len(self.arbre) or self.arbre[i] is None:
3              return
4          else:
5              self.postfixe(parcours, 2*i+1)
6              self.postfixe(parcours, 2*i+2)
7              parcours.append(self.arbre[i])
8          return parcours
```

6. Variante

```
1      def prefixe2(self, i: int = 0)->list:
2          """ création de la liste au fur et àmesure des appels"""
3          if i >= len(self.arbre) or self.arbre[i] is None:
4              return []
5          else:
6              return [self.arbre[i]] + self.prefixe2(2*i+1) + self.
                   prefixe2(2*i+2)
```


Christophe Viroulaud