TDS: AP.

1. Non, car la valeur renvoyée n'est pas la valeur du dernier appel récursif de la fonct°. b est un variant de bezout.

2.
```
bezout(51, 42) (-4, 5)                                   51, 42
  ↳ bezout(42, 9) (-1, 4)
      ↳ bezout(9, 6) (1, -1)
          ↳ bezout(6, 3) (0, 1)
              ↳ bezout(2, 0) -> (1, 0)
```

3.
```
def repr_hexadecimale(nombre : int) -> str:
    if nombre < 16:
        return CHIFFRES[nombre]
    else:
        return repr_hexadecimale(nombre // 16) + CHIFFRES[nombre % 16]
```

4.
```
def nb_occurrences(chaine : str) -> dict[str, int]:
    Res = dict()
    for char in chaine:
        if char in Res:
            res[char] += 1
        else: Res[char] = 1
    Return Res
```

5.
```
def premiere_occurrence(chaine : str) -> dict[str, int]:
    Res = dict()
    for i, char in enumerate(chaine)
        if char not in Res:
            Res[char] = i
    Return Res.
```

6.
```
def premier_parmi(cara: set, dico: dict) -> str:
    Res = ""
    indice_min = max(dico)+1.
    for char in cara:
        if indice_min > dico[char]:
            Res = char.
            indice_min = dico[char]
    Return Res.
```

7.
```
def inverse_occurrences(dico: dict) -> dict:
    Res = {}
    for cle in dico:
        if dico[cle] in Res:
            Res[dico[cle]].add(cle)
        else:
            Res[dico[cle]] = {cle}
    Return Res .
```

8.
```
def tri_occurrences(chaine: str) -> list[str]:
    dico = nb_occurrences(chaine)
    occurrences = inverse_occurrence(dico)
    indices = premiere_occurrences(chaine)
    Res = [].
    while len(occurrences) > 0:
        maxi = max(occurrences)
        carac = premier_parmi(occurrences[maxi], indices)
        Res.append(carac)
        occurrences[maxi].Remove(carac)
        if len(occurrences[maxi]) == 0:
            del occurrences[maxi]          -> Return Res.
```

```
9. class Matrice:
       def __init__(self, nb_lignes, nb_colonnes, coefficients):
           self.nb_lignes = nb_lignes
           self.nb_colonnes = nb_colonnes
           self.coefficients = coefficients.

       def __repr__(self)->str:
          return f"Matrice({self.nb_lignes}, {self.nb_colonnes},
                           {self.coefficients}"

       def __eq__(self, other):
           return self.__repr__() == other.__repr__()

       def __add__(self, other):  -> self.coefficients...
          res = {(i,j) : self.get((i,j), 0) + other.get((i,j), 0) for i in range(self.nb_lignes)
                                                                   for j in range(self.nb_colonnes)}.
          return Matrice(self.nb_lignes, self.nb_colonnes, res).

       def ligne(self, ind:int)-> list[int]:
          return [self.coefficients.get((ind, j), 0) for j in range(nb_colonnes)].

10. def __getitem__(self, prop):
     if self.pere == None:
        return self.propriete.get(prop, " ")
     else:
        self.pere.__getattr__(prop).
```