Exercice 1

1.
```
def tobin(n):
    L = []
    if n == 0:
        return [0]
    q = n
    while q > 0:
        q, r = q//2, q%2
        L.append(r)
    return L
```

2.
```
def frombin(l):
    n = 0
    puiss = 1
    for k in range(len(l)):
        n += l[k] * puiss
        puiss *= 2
    return n
```

3.
```
def xorb(a, b):
    assert (a==0 or a==1) and (b==0 or b==1)
    if a == b:
        return 0
    else:
        return 1
```

4. On a en base 2 (la barre de surlignage indiquant la base 2) :

$$1 = \overline{1}, 3 = \overline{11}, 5 = \overline{101}, 7 = \overline{111}$$

et l'associativité de l'opération $\oplus$ donne : $1 \oplus 3 \oplus 5 \oplus 7 = 0$.
Puisque $4 = \overline{100}$, on a : $1 \oplus 3 \oplus 4 \oplus 7 = 1$.

5.
```
def complete(a, b):
    if len(a) < len(b):
        c = [k for k in a]
        for _ in range(len(b)-len(a)):
            c.append(0)
        return c, b
    if len(b) < len(a):
        c = [k for k in len(b)]
        for _ in range(len(a)-len(b)):
            c.append(0)
        return c, a
```

6.
```
def xor(n, m):
    ln, lm = complete(tobin(n), tobin(m))
    l = []
    for i in range(len(ln)):
        l.append(xorb(ln[i], lm[i]))
    return frombin(l)
```

7. On a $1 \oplus 3 \oplus 5 \oplus 7 = 0$ donc la configuration $(1,3,5,7)$ n'est pas favorable.

On a $1 \oplus 3 \oplus 4 \oplus 7 = 1 \neq 0$ donc la configuration $(1,3,4,7)$ est favorable.

Pour obtenir une configuration défavorable, on peut :

- prendre l'allumette du tas 1 : $0 \oplus 3 \oplus 4 \oplus 7 = 0$;
- prendre une allumette du tas 2 : $1 \oplus 2 \oplus 4 \oplus 7 = 0$ mais $1 \oplus 1 \oplus 4 \oplus 7 \neq 0$ et $1 \oplus 0 \oplus 4 \oplus 7 \neq 0$;
- prendre une allumette du tas 4 : $1 \oplus 3 \oplus 4 \oplus 6 = 0$ mais aucune autre possibilité ne donne 0.

8. Le fait de modifier l'un des entiers change donc au moins l'une des composantes en base 2 (l'un des "bits") donc la somme $\oplus$ de cette composante change également et la configuration devient favorable (somme $\oplus$ non nulle).

9. Comme suggéré, on écrit $X = \sum_{k=0}^{\infty} X_k 2^k$ et l'on note k_0 le plus grand indice i tel que $X_i \neq 0$ (un tel indice existe puisque l'on est dans le cas où $X \neq 0$).

Par définition de cet indice k_0, il existe au moins l'un des entiers x_i dont la composante k_0 soit égale à 1 (il en existe même un nombre impair). Notons x_{i_0} cet élément.

L'idée est de remplacer x_{i_0} par un entier tel que la somme de chaque composante (pour $i \in [\![0, k_0]\!]$) soit nulle : ce sera nécessairement un entier inférieur puisque cela impose de changer la composante k_0 (qui correspond à la plus grande puissance de 2).

Pour chaque entier $i \in [\![0, k_0 - 1]\!]$, si la composante de X est 0 alors on ne change pas celle de x_{i_0} et sinon on la change. Ainsi :

- si $X_i = 0$ et $(x_{i_0})_i = 0$ alors $(y_{i_0})_i = 0$;
- si $X_i = 0$ et $(x_{i_0})_i = 1$ alors $(y_{i_0})_i = 1$;
- si $X_i = 1$ et $(x_{i_0})_i = 0$ alors $(y_{i_0})_i = 1$;
- si $X_i = 1$ et $(x_{i_0})_i = 1$ alors $(y_{i_0})_i = 0$.

On a donc $(y_{i_0})_i = \mathrm{X}_i \oplus (x_{i_0})_i$.
Donc il faut et il suffit que $z = \mathrm{X} \oplus x_{i_0}$.

Par exemple, la configuration $(3,6,8)$ est favorable puisque :

$$3 = \overline{11}, 6 = \overline{110} \text{ et } 8 = \overline{1000}$$

donc $3 \oplus 6 \oplus 8 = \overline{1101} \neq 0$. On remplace le 8 par $\overline{101} = 5$ (*i.e.* on prend 3 allumettes dans ce tas).

10. Si c'est à A de jouer avec une configuration favorable alors on peut placer (question 11) l'adversaire en situation défavorable et quoi qu'il joue (question 10), A se retrouvera à nouveau en situation favorable.

Puisqu'une configuration défavorable ne permet pas de gagner (il reste au moins deux tas), A a donc une stratégie gagnante : toujours mettre l'adversaire en configuration défavorable.

11. Si c'est à A de jouer avec une configuration défavorable alors il n'y a pas de stratégie gagnante (surtout si l'adversaire connaît la stratégie). Dans ce cas A peut gagner si son adversaire lui laisse plus tard une configuration favorable !

12.

```
def coup_suivant(N):
    p = len(N)
    somme = xor(N[0], N[1])
    for k in range(2, p):
        somme = xor(somme, N[k])
    if somme == 0: # défavorable : on enlève 1 quelque part
        i = 0
        attente = True
        while i < p and attente:
            if N[i]>0:
                N[i] -= 1
                attente = False
            i += 1
    else:           # favorable
        t = tobin(somme)
        for i in range(len(t)):    # recherche de k0
            if t[i] == 1:
                k0 = i
        i0 = 0
        for i in range(1, p):      # recherche de i0
            if N[i] > N[i0]:
                i0 = i
        N[i0] = xor(somme, N[i0]) # changement de x_i0 en z
```

Exercice 2

1. Table `serie` : `id_serie` est clé primaire.

   Table `saison` : `id_sais` est clé primaire.

   Table `episode` : `id_epis` est clé primaire.

   Table `plateforme` : `id_pf` est clé primaire.

   Table `diffusion` : une clé primaire est obtenue en considérant le quadruplet `(id_sais, id_pf, pays_diff, annee_diff)`.

2.
```sql
SELECT * FROM serie
  WHERE annee_crea_serie = 2022 AND pays_serie = "FR"
```

3.
```sql
SELECT COUNT(*) FROM serie
  WHERE annee_crea_serie = 2022 AND pays_serie = "FR"
```

4.
```sql
SELECT e.titre_epis
  FROM saison s JOIN episode e ON s.id_sais = e.is_sais
  WHERE e.id_sais = 123 AND s.num_sais = 1
  ORDER BY e.num_epis ASC
```

5.
```sql
SELECT s.num_sais, SUM(e.duree_epis)
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN episode e ON s.id_sais = e.id_sais
  WHERE sr.titre_serie = "GAME OF THRONES"
  GROUP BY s.num_sais
  ORDER BY s.num_sais ASC
```

6.
```sql
SELECT s.titre_serie, SUM(e.duree_epis) as duree
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN episode e ON s.id_sais = e.id_sais
  GROUP BY s.titre_serie
  ORDER BY duree DESC
  LIMIT 2
```

7.
```sql
SELECT DISTINCT sr.titre_serie
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN diffusion d ON s.id_sais = d.id_sais
  WHERE d.annee_diff = 2023 AND d.pays_diff = "USA"
```

8.
```sql
(SELECT DISTINCT sr.titre_serie
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN diffusion d ON s.id_sais = d.id_sais
  WHERE d.annee_diff = 2023 AND d.pays_diff = "USA")

EXCEPT

(SELECT DISTINCT sr.titre_serie
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN diffusion d ON s.id_sais = d.id_sais
  WHERE d.annee_diff = 2023 AND d.pays_diff = "FR")
```

9.
```sql
SELECT sr.titre_serie, COUNT(s.id_sais) as nb
  FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
      JOIN diffusion d ON s.id_sais = d.id_sais
      JOIN plateforme p ON d.id_pf = p.id_pf
  WHERE d.annee_diff = 2023 AND d.pays_diff = "FR"
          AND p.nom_pf = "NFX"
  GROUP BY sr.id_serie
  HAVING nb > 1
```

Exercice 3

1.

```
def distance(a, b):
    s = 0
    for j in range(4):
        s += (a[j] - b[j])**2
    return s
```

2.

```
def ToutesDistances(new):
    t = []
    for k in range(len(eleves)):
        c = (distance(new[1:], eleves[k][1:-1]), eleves[k][-1])
        t.append(c)
    return t
```

3. Tout d'abord, la fonction de tri évoquée peut par exemple être programmée ainsi :

```
def tri(l):
    for i in range(len(l)):
        i_min = i
        for j in range(i+1, len(l)):
            if l[j][0] < l[i_min][0]:
                i_min = j
        l[i_min], l[i] = l[i], l[i_min]
```

ce qui donne par exemple :

```
>>> a = [(5, 'baz'), (3, 'foo'), (9, 'bar')]
>>> tri(a)
>>> a
[(3, 'foo'), (5, 'baz'), (9, 'bar')]
```

En ce qui concerne désormais la question posée :

```
def MaisonMajoritaire(new):
    t = ToutesDistances(new)
    tri(t)
    t = t[:11]
    d = {}
    for a in t:
        if a[1] in d:
            d[a] += 1
        else:
            d[a] = 1
    val_max = 0
    for a in d:
        if d[a] > val_max:
            mais_max = a
            val_max = d[a]
    return mais_max[1]
```