Binaire II

On continue notre exploration du monde des 0 *et des* 1.

Activité 1 (Palindromes)**.**

Objectifs : trouver des palindromes en écriture binaire et en écriture décimale.

En français un palindrome est un mot (ou une phrase) qui se lit dans les deux sens, par exemple « **RADAR** » ou « **ENGAGE LE JEU QUE JE LE GAGNE** ». Dans cette activité, un ***palindrome*** sera une liste, qui a les mêmes éléments lorsqu'on la parcourt de gauche à droite ou de droite à gauche.
Exemples :

- `[1,0,1,0,1]` est un palindrome (avec une écriture binaire),
- `[2,9,4,4,9,2]` est un palindrome (avec une écriture décimale).

1. Programme une fonction `est_palindrome(liste)` qui teste si une liste est un palindrome ou pas.
   Indications. Tu peux comparer les éléments en position i et $p-1-i$ ou bien utiliser `list(reversed(liste))`.
2. On cherche des entiers n tels que leur écriture binaire soit un palindrome. Par exemple l'écriture binaire de $n=27$ est le palindrome `[1,1,0,1,1]`. C'est le dixième entier n ayant cette propriété.
   Quel est le millième entier $n \geqslant 0$ dont l'écriture binaire est un palindrome ?
3. Quel est le millième entier $n \geqslant 0$ dont l'écriture décimale est un palindrome ?
   Par exemple les décimales de $n=909$ forment le palindrome `[9,0,9]`. C'est le centième entier n ayant cette propriété.
4. Un entier n est un ***bi-palindrome*** si son écriture binaire *et* son écriture décimales sont des palindromes. Par exemple $n=585$ a une écriture décimale qui est un palindrome et son écriture binaire `[1,0,0,1,0,0,1,0,0,1]` aussi. C'est le dixième entier n ayant cette propriété.
   Quel est le vingtième entier $n \geqslant 0$ a être un bi-palindrome ?

Cours 1 (Opérations logiques)**.**
On considère que 0 représente le « faux » et 1 le « vrai ».

- Avec l'opération logique « OU », le résultat est vrai dès que l'un au moins des deux termes est vrai. Cela s'écrit :
  — 0 OU 0 = 0
  — 0 OU 1 = 1
  — 1 OU 0 = 1
  — 1 OU 1 = 1

- Avec l'opération logique « ET », le résultat est vrai uniquement lorsque les deux termes sont vrais. Cela s'écrit :
  - 0 ET 0 = 0
  - 0 ET 1 = 0
  - 1 ET 0 = 0
  - 1 ET 1 = 1
- L'opération logique « NON », échange vrai et faux :
  - NON 0 = 1
  - NON 1 = 0
- Pour des nombres en écriture binaire, ces opérations s'étendent *bits* à *bits*, c'est-à-dire chiffre par chiffre (en commençant par les chiffres de droite) comme on poserait une addition (sans retenue).
  Par exemple :

	ET		OU
	1.0.1.1.0		1.0.0.1.0
ET	1.1.0.1.0	OU	0.0.1.1.0
	1.0.0.1.0		1.0.1.1.0

  Si les deux écritures n'ont pas le même nombre de *bits*, on rajoute des 0 non significatifs à gauche (exemple de 1.0.0.1.0 OU 1.1.0 sur la figure de droite).

Activité 2 (Opérations logiques).

Objectifs : programmer les principales opérations logiques.

1. (a) Programme une fonction `NON()` qui correspond à la négation pour une liste donnée. Par exemple `NON([1,1,0,1])` renvoie `[0,0,1,0]`.
   (b) Programme une fonction `OUeg()` qui correspond au « OU » avec en entrée deux listes qui ont la même longueur. Par exemple, avec `liste1 = [1,0,1,0,1,0,1]` et `liste2 = [1,0,0,1,0,0,1]`, la fonction renvoie `[1,0,1,1,1,0,1]`.
   (c) Même travail avec `ETeg()` pour deux listes de longueurs égales.
2. Écris une fonction `ajouter_zeros(liste,p)` qui rajoute des zéros au début de la liste afin d'obtenir un liste de longueur p. Exemple : si `liste = [1,0,1,1]` et `p = 8`, alors la fonction renvoie `[0,0,0,0,1,0,1,1]`.
3. Écris deux fonctions `OU()` et `ET()` qui correspondent aux opérations logiques « OU » et « ET », mais avec deux listes qui n'ont pas nécessairement la même longueur.
   Exemple :
   - `liste1 = [1,1,1,0,1]` et `liste2 = [1,1,0]`,
   - il faut considérer que `liste2` est équivalente à la liste `liste2bis = [0,0,1,1,0]` de même longueur que `liste1`,
   - donc `OU(liste1,liste2)` renvoie `[1,1,1,1,1]`,
   - puis `ET(liste1,liste2)` renvoie `[0,0,1,0,0]` (ou bien `[1,0,0]` selon ton choix).

   Indications : tu peux reprendre le contenu de tes fonctions `OUeg` et `ETeg`, ou bien tu peux d'abord ajouter des zéros à la liste la plus courte.

Activité 3 (Lois de Morgan).

Objectifs : générer toutes les listes possibles de 0 *et* 1 *afin de vérifier une proposition.*

1. **Première méthode : utiliser l'écriture binaire.**

   On souhaite générer toutes les listes possibles de 0 et de 1 d'une taille p donnée. Voici comment faire :
   - Un entier n parcourt tous les entiers de 0 à $2^p - 1$.
   - Pour chacun de ces entiers n, on calcule son écriture binaire (sous la forme d'une liste).
   - On rajoute (si besoin) des 0 en début de liste, afin d'obtenir une liste de longueur p.

   Exemple : avec $n = 36$, son écriture binaire est `[1,0,0,1,0,0]`. Si on veut une liste de $p = 8$ *bits*, on rajoute deux 0 : `[0,0,1,0,0,1,0,0]`.

2. **Seconde méthode (optionnelle) : un algorithme récursif.**

   On souhaite de nouveau générer toutes les listes possibles de 0 et de 1 d'une taille donnée. On adopte la procédure suivante : si on sait trouver toutes les listes de taille $p-1$, alors pour obtenir toutes les listes de taille p, il suffit de rajouter un 0 en début de chacune des listes de taille $p-1$, puis de recommencer en rajoutant un 1 en début de chacune des listes de taille $p-1$.

   Par exemple : il y a 4 listes de longueur 2 : `[0, 0]`, `[0, 1]`, `[1, 0]`, `[1, 1]`. J'en déduis les 8 listes de longueur 3 :
   - 4 listes en rajoutant un 0 devant : `[0, 0, 0]`, `[0, 0, 1]`, `[0, 1, 0]`, `[0, 1, 1]`,
   - 4 listes en rajoutant un 1 devant : `[1, 0, 0]`, `[1, 0, 1]`, `[1, 1, 0]`, `[1, 1, 1]`.

   Cela donne l'algorithme suivant, qui est un algorithme récursif (car la fonction s'appelle elle-même).

   > **Algorithme.**
   > Usage : `tous_les_binaires(p)`
   > Entrée : un entier $p > 0$
   > Sortie : la liste de toutes les listes possibles de 0 et de 1 de longueur p
   > - Si $p = 1$ renvoyer la liste `[ [0], [1] ]`.
   > - Si $p \geqslant 2$, alors :
   >   — obtenir toutes les listes de taille `p-1` par l'appel `tous_les_binaires(p-1)`
   >   — pour chaque élément de cette liste, construire deux nouveaux éléments :
   >   — d'une part ajouter 0 en début de cet élément ;
   >   — d'autre part ajouter 1 en début de cet élément ;
   >   — ajouter ensuite ces deux éléments à la liste des listes de taille p.
   > - Renvoyer la liste des listes de taille p.

3. **Les lois de Morgan.**

   Les lois de Morgan affirment que pour des booléens (vrai/faux) ou des *bits* (1/0), on a toujours les égalités :

   $$\text{NON}(b_1 \text{ OU } b_2) = \text{NON}(b_1) \text{ ET } \text{NON}(b_2), \qquad \text{NON}(b_1 \text{ ET } b_2) = \text{NON}(b_1) \text{ OU } \text{NON}(b_2).$$

   Vérifie expérimentalement que ces égalités sont encore vraies pour n'importe quelles listes ℓ_1 et ℓ_2 d'exactement 8 *bits*.