1 Palindromes binaires (*Olympiades Lyon 2017*)

Partie A

Dans notre système décimal habituel (en base 10), l'écriture du nombre 53 est liée à la décomposition :

$$53 = 5 \times 10^1 + 3 \times 10^0$$

On peut changer de base et par exemple tout nombre entier naturel s'écrit de manière unique comme somme de puissances de 2.
Ainsi on a :

$$53 = 2^5 + 2^4 + 2^2 + 2^0 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

On dit que le nombre 53 s'écrit 110101 dans le système binaire (en base 2) et on écrira :

$$53 = (110101)_2$$

Le système binaire est très utilisé en informatique, le composant de base des ordinateurs étant le transistor qui fonctionne comme un interrupteur à deux états : le courant passe (état 1) ou ne passe pas (état 0).

1. Écrire dans le système binaire le nombre 135.
2. Déterminer à quel nombre correspond $(101011)_2$.
3. Écrire en Python une fonction `dec2bin(nombre)`, qui implémente l'algorithme suivant de conversion de l'écriture en base 10 d'un nombre vers son écriture en base 2 :

```
Fonction dec2bin(N)
    Si N = 0
        Retourner liste contenant 0
    Sinon
        CHIFFRES ⟵ liste vide
        PUISSANCE ⟵ 1
        Tant que PUISSANCE ⩽ N
            PUISSANCE ⟵ PUISSANCE × 2
        Fin Tant Que
        Tant que PUISSANCE > 1
            PUISSANCE ⟵ PUISSANCE / 2
            Si PUISSANCE ⩽ N
                N ⟵ N − PUISSANCE
                Ajouter 1 à la fin de CHIFFRES
            Sinon
                Ajouter 0 à la fin de CHIFFRES
            Fin Si
        Fin Tant Que
        Retourner CHIFFRES
```

On pourra vérifier que :

```
In [7]: dec2bin(53)
Out[7]: [1, 1, 0, 1, 0, 1]

In [8]: dec2bin(2)
Out[8]: [1, 0]
```

4. Soit n un entier naturel, démontrer que $2^n - 1 = (\underbrace{111\ldots11}_{n \text{ fois}})_2$.

Partie B

On a $17 = (10001)_2$ et $165 = (10100101)_2$.
Un nombre est un palindrome binaire si son écriture en base 2 se lit indifféremment de gauche à droite et de droite à gauche. Ce nombre ne commence jamais par zéro.
Ainsi 17 et 165 sont des palindromes binaires. Remarquons que ce ne sont pas des palindromes en base 10.

1. **a.** 2017 est-il un palindrome binaire ?

b. Trouver le prochain palindrome binaire après 2017.

c. Écrire une fonction `est_palindrome(chiffres)` qui détermine si la liste des chiffres d'un nombre est un palindrome.

d. Écrire une fonction `listePalindromeBmax(bmax)` qui, pour les nombres inférieurs ou égaux à `bmax`, retourne la liste de toutes les listes de chiffres en base 2 des palindromes binaires.

```
In [14]: listePalindromeBmax(15)
Out[14]: [[1], [1, 1], [1, 0, 1], [1, 1, 1], [1, 0, 0, 1], [1, 1, 1,
      1]]
```

e. Déterminer tous les palindromes binaires compris entre 1 et 129.

2. On s'intéresse à l'écriture binaire des nombres entiers naturels.

a. Combien y a-t-il de palindromes binaires à 3, 4, 5, 6 et 7 chiffres en écriture binaire ? Les écrire.

b. Pour tout entier $n \geqslant 1$, on note $P(n)$ le nombre de palindromes binaires à n chiffres.
Déterminer $P(n)$ si n est pair puis si n est impair.

3. On cherche le nombre $F(2^n)$ de palindromes binaires strictement inférieurs au nombre 2^n avec n un entier naturel non nul.

a. Déterminer $F(2^5)$ et $F(2^6)$.

b. Montrer que le nombre de palindromes binaires inférieurs à 2^n est :

- $F(2^n) = 2^{\frac{n+2}{2}} - 2$ si n est pair.
- $F(2^n) = 3 \times 2^{\frac{n-1}{2}} - 2$ si n est impair.

c. Soit x un nombre entier naturel tel que $x > 5$.
On note $F(x)$ le nombre de palindromes binaires strictement inférieurs à x.
On appelle n le nombre entier tel que $2^n \leqslant x < 2^{n+1}$.

- Justifier que $F(2^n) \leqslant F(x) < F(2^{n+1})$.
- Montrer que $\sqrt{x} < F(x) < 3\sqrt{x}$.

2 Changements de base

Partie A : deux algorithmes

Bloc-Note 1 *Conversion en base 2*

Pour écrire les entiers naturels en base deux, on utilise deux chiffres : 0 et 1. Si on se donne un entier naturel n, on imagine qu'il représente n objets et on les groupe par paquets de deux, puis on groupe ces paquets en paquets de deux paquets, ...Ainsi, on fait une succession de divisions par 2, jusqu'à obtenir un quotient égal à 0 et l'écriture en base 2 de n est la liste des restes successifs obtenus (0 ou 1, appelés bits) mais dans l'ordre inverse (le premier bit obtenu est le bit de poids faible et le dernier celui de poids fort).
Avec cet *algorithme des divisions en cascades* voici l'exemple de la conversion du nombre décimal 87 en base 2 :

Etape	N (dividende)	Q (Quotient)	R (Reste)
$87 = 2 \times 43 + 1$	87	43	1
$43 = 2 \times 21 + 1$	43	21	1
$21 = 2 \times 10 + 1$	21	10	1
$10 = 5 \times 2 + 0$	10	5	0
$5 = 2 \times 2 + 1$	5	2	1
$2 = 1 \times 2 + 0$	2	1	0
$1 = 0 \times 2 + 1$	1	0	1

On peut écrire :

$$87 = 2 \times (2 \times (2 \times (2 \times (2 \times 1 + 0) + 1) + 0) + 1) + 1) + 1$$
$$87 = 2^6 \times 1 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1$$

Ainsi l'écriture de 87 en base 2 est 1010111.

En Python, la primitive `bin` retourne l'écriture binaire d'un en entier en écriture décimale, sous la forme d'une chaîne de caractères préfixée par '0b'.

```
>>> bin(87)
'0b1010111'
>>> bits = list(map(int, bin(87)[2:]))
[1, 0, 1, 0, 1, 1, 1]
>>> n = 0
```

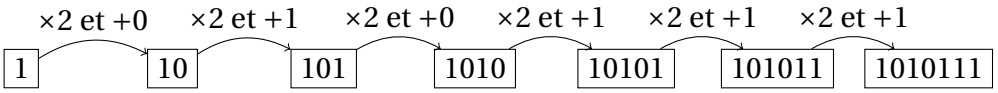
Pour passer d'un tableau contenant l'écriture binaire d'un entier à son écriture décimale, il faut d'abord déterminer si les bits sont rangés dans le tableau par poids décroissant ou croissant. Le sens habituel, choisi pour la fonction `bin` par exemple, est de commencer à l'index 0 du tableau par le bit de poids fort, puis de faire décroître le poids lorsque l'index augmente. Ensuite on peut procéder selon deux algorithmes :

☞ On multiplie chaque bit par la puissance de 2 correspondant à sa position dans le tableau et on somme : à partir de l'écriture binaire `[1, 0, 1, 0, 1, 1, 1]` on retrouve ainsi l'écriture décimale 87. Le premier bit du tableau étant le bit de poids fort, si on appelle ℓ la longueur du tableau, il correspond à $2^{\ell-1}$. Ici $\ell = 7$.

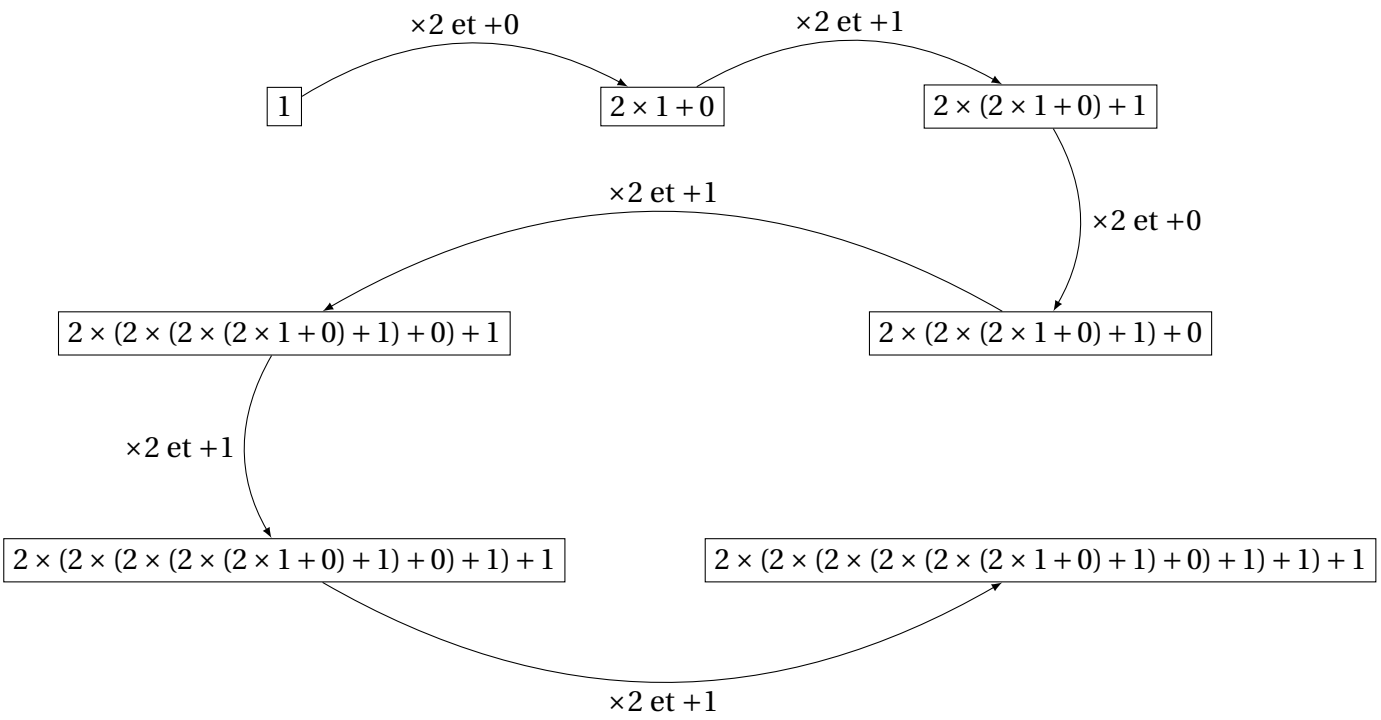
$$87 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

```
>>> for i in range(len(bits)):
...     n += bits[i]*2**(len(bits) - 1 - i)
...
>>> n
87
```

☞ On lit les bits de gauche à droite par poids décroissant, à chaque étape on décale tous les bits du dernier nombre lu vers la gauche et on rajoute le nouveau bit comme bit d'unité. En écriture binaire, un décalage vers la gauche équivaut à une multiplication par 2 :



Il suffit de traduire cet algorithme pour retrouver progressivement l'écriture décimale :



On reconstruit ainsi dans l'autre sens l'expression traduisant la conversion en binaire de 87 par *l'algorithme des divisions en cascades*. On peut lire l'écriture binaire sur la partie droite et la plus grande puissance de 2 inférieure au nombre dans la partie gauche.

Cet algorithme coûte moins de multiplications que le précédent, on n'a pas besoin de calculer les puissances de 2 successives. Il présente donc une meilleure *complexité temporelle*. C'est la réciproque de *l'algorithme des divisions en cascades*.

Ces deux algorithmes de conversion d'écriture de la base 2 en la base 10, peuvent être utilisés pour l'évaluation de polynômes ou pour la conversion de la base 10 vers une base différente de 2. Le premier est un algorithme naif, le second s'appelle *l'algorithme d'Horner*.

Partie B : des applications

1. Compléter le code de la fonction `nombre2chiffres(nombre, base)` qui prend en argument un nombre écrit en base 10 et une base et qui retourne la liste des chiffres de ce nombre dans la base choisie, en implémentant *l'algorithme des division en cascade*. La liste retournée doit contenir les chiffres par poids décroissant.

```
def nombre2chiffres(nombre, base):
    chiffres = [nombre % base]
    nombre = nombre // base
    ........................
    ........................
    ........................
    ........................
    return chiffres[::-1]
```

2. Écrire une fonction `chiffres2nombre(nombre, base)` réciproque de la précédente qui retourne l'écriture en base 10 d'un nombre à partir de la liste de ses chiffres dans une base fixée.

3. Résoudre le problème n° 2 du projet Euler.

   A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is $9009 = 91 \times 99$.

   Find the largest palindrome made from the product of two 3-digit numbers.

4. Résoudre le problème n° 16 du projet Euler.

   $2^{15} = 32768$ *and the sum of its digits is* $3+2+7+6+8 = 26$.

   What is the sum of the digits of the number 2^{1000} ***?***

5. Résoudre le problème n° 36 du projet Euler.

   The decimal number, $585 = (1001001001)_2$ *(binary), is palindromic in both bases.*

   Find the sum of all numbers, less than one million, which are palindromic in base 10 ***and base*** 2**.**

   (Please note that the palindromic number, in either base, may not include leading zeros.)

3 Nombres de Lychrel

Résoudre le problème n° 55 du projet Euler.

If we take 47*, reverse and add,* $47 + 74 = 121$*, which is palindromic.*

Not all numbers produce palindromes so quickly. For example,

$349 + 943 = 1292$,

$1292 + 2921 = 4213$

$4213 + 3124 = 7337$

That is, 349 *took three iterations to arrive at a palindrome.*

Although no one has proved it yet, it is thought that some numbers, like 196, *never produce a palindrome. A number that never forms a palindrome through the reverse and add process is called a Lychrel number.*

Due to the theoretical nature of these numbers, and for the purpose of this problem, we shall assume that a number is Lychrel until proven otherwise.

In addition you are given that for every number below ten-thousand, it will either (i) become a palindrome in less than fifty iterations, or, (ii) no one, with all the computing power that exists, has managed so far to map it to a palindrome.

In fact, 10677 *is the first number to be shown to require over fifty iterations before producing a palindrome :* 4668731596684224866951378664 *(53 iterations, 28-digits).*

Surprisingly, there are palindromic numbers that are themselves Lychrel numbers ; the first example is 4994.

How many Lychrel numbers are there below ten-thousand ?