Informatique PCSI
TP 11a : représentation des entiers

Exercice 1

1. On utilise la méthode des divisions euclidiennes successives en conservant les restes dans une chaîne de caractères. Il faut être attentif à l'ordre d'écriture.

   La fonction est nommée `base2`.

```
def base2(n):
    if n == 0:
        return "0"
    b = ""
    while n != 0:
        r = n % 2
        n = n // 2
        b = str(r) + b # le reste est ajouté devant la chaîne
    return b
```

2. Il suffit d'une simple boucle. Si $n = (b_n b_{n-1} \dots b_1 b_0)_2$ alors $n = b_n \times 2^n + \dots + b_1 \times 2 + b_0$.
   Une autre manière d'écrire le programme est de commencer par transformer l'expression à calculer : $n = 2 \times (2 \times (\dots 2 \times (2 \times b_n + b_{n-1}) + b_{n-2} \dots) + b_1) + b_0$.
   Dans ce cas, on initialise une variable `n = 0`, puis on parcourt les bits du nombre et à chaque passage dans la boucle, on multiplie la valeur courante de `n` par 2 et on ajoute le bit suivant. Les valeurs successives de `n` sont alors b_n, puis $2 \times b_n + b_{n-1}$, puis $2 \times (2 \times b_n + b_{n-1}) + b_{n-2}$, etc.

```
def base10(b):
    n = 0
    for bit in b:
        n = 2 * n + int(bit)
    return n
```

Exercice 2

```
def bin_kbits(n, k):
    b = str()
    q = n
    while q != 0:
        q, r = q // 2, q % 2
        b = str(r) + b
    b = (k - len(b)) * '0' + b
    return b
```

Exercice 3

1. On peut remarquer le résultat suivant :
   $x_0 \times 256^3 + x_1 \times 256^2 + x_2 \times 256^1 + x_3 \times 256^0 = x_3 + 256(x_2 + 256(x_1 + 256x_0))$.

```
    def decimal_be(b): #  si big endian
        n = len(b)
        d = b[0]
        for i in range(1, n):
            d = b[i] + 256 * d
        return d
```

2.

```
    def decimal_le(b): #  si little endian
        n = len(b)
        d = b[n-1]
        for i in range(1, n):
            d = b[n-1-i] + 256 * d
        return d
```

Exercice 4

Codage d'un entier relatif

Avec un codage sur n bits, on code les nombres positifs r de 0 à $2^{n-1}-1$ par l'écriture en binaire de r et les négatifs r de -2^{n-1} à -1 par l'écriture en binaire de $r+2^n$.

Le programme doit tester si r est négatif.

Il faut penser à ajouter éventuellement des 0 pour obtenir n bits.

```
def relatif(r, n):
    if r < 0:
        r = r + 2 ** n
    if r == 0:
        return n * "0"
    b = ""
    q = r
    while q != 0:
        reste = q % 2
        q = q //2
        b = str(reste) + b
    return (n - len(b)) * "0" + b
```

Pour tester la fonction utiliser des entiers r et n vérifiant : $-2^{n-1} \leq r < 2^{n-1}$.