- Epreuve pratique -
- Tle NSI -
Récursivité :

Exercice 1 :

L'objectif de cet exercice est d'écrire deux fonctions récursives `dec_to_bin` et `bin_to_dec` assurant respectivement la conversion de l'écriture décimale d'un nombre entier vers son écriture en binaire et, réciproquement, la conversion de l'écriture en binaire d'un nombre vers son écriture décimale.

Dans cet exercice, on s'interdit l'usage des fonctions Python `bin` et `int`.

On rappelle sur l'exemple ci-dessous une façon d'obtenir l'écriture en binaire du nombre 25 :

$25 = 1 + 2 \times 12$

$= 1 + 2 (0 + 2 \times 6)$

$= 1 + 2 (0 + 2 (0 + 2 \times 3))$

$= 1 + 2 (0 + 2 (0 + 2 \times (1 + 2 \times 1)))$

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

L'écriture en binaire de 25 est donc 11001.

0n rappelle également que :
`a // 2` renvoie le quotient de la division euclidienne de `a` par `2`.
`a % 2` renvoie le reste dans la division euclidienne de `a` par `2`.

On indique enfin qu'en Python si `mot = "informatique"` alors :
`mot[-1]` renvoie `'e'`, c'est-à-dire le dernier caractère de la chaîne de caractères `mot`.
`mot[:-1]` renvoie `'informatiqu'` , c'est-à-dire l'ensemble de la chaîne de caractères `mot` privée de son dernier caractère.

Compléter, puis tester, les codes de deux fonctions de la page suivante.

On précise que la fonction récursive `dec_to_bin` prend en paramètre un nombre entier et renvoie une chaîne de caractères contenant l'écriture en binaire du nombre passé en paramètre.

Exemple :

```
>>> dec_to_bin(25)
'11001'
```

La fonction récursive bin_to_dec prend en paramètre une chaîne de caractères représentant l'écriture d'un nombre en binaire et renvoie l'écriture décimale de ce nombre.

```
>>> bin_to_dec('101010')
42
```

```
def dec_to_bin(nb_dec):
    ...

def bin_to_dec(nb_bin):
    if nb_bin == "0":
        return 0
    elif ...:
        return 1
    else:
        if nb_bin[-1] == '0':
            bit_droit = 0
        else:
            bit_droit = ...
        return ... * ...(nb_bin[:-1]) + ...
```