TP 15 : Représentation des entiers naturels

Représentation d'une écriture binaire

Dans ce TP une écriture binaire sera toujours représentée comme une liste python contenant des *booléens*. On rappelle qu'il n'existe que deux valeurs booléennes :

- La valeur `True` qu'on va utiliser pour représenter le chiffre 1
- La valeur `False` qu'on va utiliser pour représenter le chiffre 0

Comme vu en cours, dans une écriture binaire $b_n \dots b_1 b_0$, à chaque chiffre correspond un *indice* i et une puissance de deux 2^i ; et quand on écrit sur feuille ou au tableau, on écrit les chiffres de plus grand indice à droite, et ceux de plus faible indice à gauche. Cependant, pour éviter des erreurs de programmation, dans ce TP on va faire correspondre les indices des chiffres dans une écriture binaire avec les indices des listes python qu'on va utiliser pour les représenter. **Dans ce TP, donc, les chiffres de faible indice seront à gauche de la liste, et les chiffres d'indice élevé seront à droite de la liste**, à l'envers d'une représentation manuscrite ou textuelle.

Par exemple,

- Le nombre 8 dont l'écriture binaire est 1000 sera représentée par la liste python `[False, False, False, True]`
- Le nombre 42 dont l'écriture binaire est 101010 sera représenté par la liste python `[False, True, False, True, False, True]`

Exercice 1 (Affichage d'une écriture binaire)

Écrire une fonction `affiche` qui prend en entrée une écriture binaire (une liste de booléens) et qui l'affiche de la même façon qu'on l'aurait écrite à la main (avec des 0 et des 1, avec les indices plus forts à droite et les plus faibles à gauche).

L'entête et le commentaire de documentation de votre fonction peuvent par exemple être les suivants :

```
def affiche(ecriture):
    """
    Affiche une écriture binaire.

    >>> affiche([False, False, False, True])
    1000

    >>> affiche([False, True, False, True, False, True])
    101010
    """
    # À faire: écrire votre code ici
```

Exercice 2 (Décodage binaire)

Écrire une fonction `decodage_bin` qui prend en entrée une écriture binaire et renvoie l'entier correspondant.
Exemple d'entête et commentaire de documentation :

```
def decodage_bin(ecriture):
    """
    Décodage d'écritures binaires
    (de nombres positifs).

    Entrée
    -------
      Une écriture binaire.

    Sortie
    ------
      L'entier positif correspondant.

    >>> decodage_bin([False, False, False, True])
    8

    >>> decodage_bin([False, True, False, True, False, True])
    42
    """
```

Exercice 3 (Encodage binaire)

1. Écrire une fonction `encodage_bin` qui prend en entrée un nombre et renvoie son écriture binaire.
   Exemple d'entête et commentaire de documentation :

```
def encodage_bin(n):
    """
    Encodage en binaire
    (de nombres positifs, sans limite du nombre de chiffres).

    Entrée
    -------
      Un entier naturel.

    Sortie
    ------
      L'écriture binaire de cet entier.
      Le chiffre de plus fort indice est toujours 1,
      sauf si l'entier encodé est 0, dans lequel cas
      l'écriture est forcément [False].

    >>> encodage_bin(8)
    [False, False, False, True]

    >>> encodage_bin(42)
    [False, True, False, True, False, True]
    """
```

2. Faites en tout au moins 2 version différentes de votre fonction `encodage_bin` de telle sorte que
   — Au moins l'une des versions utilise la méthode 4 du cours (encodage « de gauche à droite »).
   — Au moins l'une des versions utilise la méthode 9 du cours (encodage « de droite à gauche »).
   — Au moins l'une des versions est récursive.
   Remarque 1 : Les listes python ont une méthode `reverse` qui inverse l'ordre des éléments dans une liste, en place. Par exemple :

```
l = [1, 2, 3]
# Après cette affectation l contient [1, 2, 3]
l.reverse()
# Après cet appel de méthode, l contient [3, 2, 1]
```

   Remarque 2 : Il devrait être plus simple de faire une version récursive avec la méthode 9 qu'avec la méthode 4... mais en soi tout est possible.

Exercice 4 (Entiers non-signés de taille fixe)

Dans cet exercice, comme dans la partie 2 du cours, on va limiter le nombre maximum de chiffres d'une écriture binaire.

— Écrire une fonction `encoder_fixe` qui prend en arguments un entier `n` à encoder et un nombre `k` de chiffres binaires à utiliser, et qui renvoie l'encodage sur k chiffres binaires de n.
Le résultat sera de taille exactement k. Tout chiffre non-nécessaire sera fixé à 0 (donc à `False` dans notre représentation).

— Est-il nécessaire d'écrire une fonction `decoder_fixe` (et si non pourquoi) ?

Exercice 5 (Addition de taille fixe)

Écrire une fonction `addition` qui prend en entrée deux écriture binaires de même taille et renvoie leur somme.
Votre fonction vérifiera que les deux écritures en entrée ont bien la même taille. Le résultat aura la même taille que les entrées.

Exercice 6 (Bonus : complexités)

Quelle est la complexité des différentes fonctions que vous avez codé pendant ce TP ? (Y compris de la version récursive de `encoder_bin`)