

TD 24.2 – Codage

Conversions numériques : binaire et hexadécimal

1 / Conversion décimal → binaire

En cours sur la numération, nous avons vu l'algorithme suivant pour passer de décimal vers binaire :

1. Regarder combien de bits sont nécessaires pour écrire K . On prend N bits tels que $2^{N-1} - 1 < K \leq 2^N - 1$
2. Écrire la base des puissances de 2 de droite à gauche : $L_puiss_2 = [2^{N-1}; 2^{N-2}; \dots; 4; 2; 1]$
3. Soit $x = K$. # (x désignera le reste pour la suite)

On lit la liste précédente de gauche à droite. Pour chaque élément 2^i de la liste L_puiss_2

si $x \geq 2^i$

alors on écrit 1 sous le 2^i correspondant, et x est remplacé par son ancienne valeur moins 2^i .

Sinon,

on écrit 0 et x inchangé.

(Dans les deux cas, on passe ensuite à l'élément suivant 2^{i-1})

Q1 – 📁 Quelle valeur maximale peut être codée en binaire naturel à l'aide de N bits ?

Q2 – 📁 Écrire une fonction `bits_necessaires( $N\_decimal$ )` prenant en argument une valeur entière décimale $N_decimal$ et renvoyant un entier (`int`) correspondant au nombre de bits nécessaires pour coder ce nombre.

Algorithme : On s'interdira l'usage d'un logarithme : on utilisera plutôt une boucle `while` pour ce problème : on cherche donc N tel que : $2^{N-1} - 1 < N_decimal \leq 2^N - 1$. La condition du `while` ne devra compter qu'une seule inégalité.

Contrainte supplémentaire : on considèrera en début de fonction l'éventualité où $N_decimal$ vaut 0, auquel cas la fonction renverra 1.

Tests :

```
print(bits_necessaires(0))      # doit afficher 1
print(bits_necessaires(1))      # doit afficher 1
print(bits_necessaires(128))    # doit afficher 8
print(bits_necessaires(255))    # doit afficher 8
```

Q3 – 📁 Écrire une fonction `base_2( $N\_bits$ )` prenant en argument un nombre entier N_bits (positif non nul) et renvoyant la liste des puissances de 2 : $[2^{N_bits-1}; 2^{N_bits-2}; \dots; 4; 2; 1]$ comptant N_bits éléments.

Test :

```
print(base_2(8))                # doit afficher [128, 64, 32, 16, 8, 4, 2, 1]
```

Q4 – 📁 Écrire une fonction `decim_binaire( $N\_decimal$ )` prenant en argument un nombre décimal $N_decimal$ positif, et renvoyant le codage de ce nombre en binaire sous forme de liste de 0 et de 1 (on s'appuiera sur les deux fonctions précédentes, et sur l'algorithme décrit en haut de page).

Tests :

```
print(decim_binaire(0))          # doit afficher [0]
print(decim_binaire(2))          # doit afficher [1,0]
print(decim_binaire(127))        # doit afficher [1, 1, 1, 1, 1, 1, 1]
print(decim_binaire(128))        # doit afficher [1, 0, 0, 0, 0, 0, 0, 0]
print(decim_binaire(255))        # doit afficher [1, 1, 1, 1, 1, 1, 1, 1]
print(decim_binaire(201))        # doit afficher [1, 1, 0, 0, 1, 0, 0, 1]
```

Q5 – 📁 Écrire une fonction `binaire_decimal( $L\_binaire$ )` prenant en argument une liste de 0 et de 1, et renvoyant la valeur décimale correspondante $\sum_{i=0}^{N-1} L[N-1-i] \times 2^i$. (N désignant ici la taille de la liste)

Tests :

```
print(decim_binaire([1, 0, 1, 0]))      # doit afficher 10
print(decim_binaire([1, 1, 0, 0, 1, 0, 0, 1])) # doit afficher 201
```

2 / Conversion binaire → hexadécimal

Q5 – Écrire une fonction `digit_hexa(N)` prenant en argument un nombre N (compris entre 0 et 15) et renvoyant son codage en string hexadécimal (*str* qui sera de taille 1).

Rappel : en hexadécimal, $10 \Leftrightarrow "A"$; $11 \Leftrightarrow "B"$; ... ; $14 \Leftrightarrow "E"$; $15 \Leftrightarrow "F"$.

Contrainte : on impose la trame suivante (à compléter)

```
def digit_hexa(N):
    if ...                # N s'écrit pareil en décimal et hexadécimal
        return ...
    else :                 # sinon : il faut faire appel aux lettres
        L = ["A", "B", "C", "D", "E", "F"]
        return ...
```

Tests :

```
print(digit_hexa(0)) # doit afficher "0" (l'afficheur cache les guillemets)
print(digit_hexa(8)) # doit afficher "8" (l'afficheur cache les guillemets)
print(digit_hexa(10)) # doit afficher "A" (l'afficheur cache les guillemets)
print(digit_hexa(15)) # doit afficher "F" (l'afficheur cache les guillemets)
```

On veut écrire une fonction `decoupe_4(L)` prenant en argument une liste, de taille quelconque N , et qui renverra une liste de listes toutes de taille 4 : l'idée est de découper une liste de valeurs binaires par lots de 4 bits.

Exemple simple : `decoupe_4([1,1,0,0,0,0,1,0])` # renverra `[[1,1,0,0],[0,0,1,0]]`

Dans le cas général, N n'est cependant pas un multiple de 4. Dans ce cas, la fonction complètera la liste L avec des zéro à gauche, jusqu'à ce que la liste soit de taille multiple de 4. De la sorte :

Exemples :

```
decoupe_4([1,1,1,0,1,0]) # renverra [[0,0,1,1],[1,0,1,0]]
decoupe_4([1,1,1,1,0])   # renverra [[0,0,0,1],[1,1,1,0]]
```

On donne la trame suivante à copier-coller et compléter :

```
def decoupe_4(L):
    N = len(L)
    if ... # s'il y a besoin de compléter (donc si N n'est pas multiple de 4)
        N_bits_a_compléter = ... # nombre de 0 à compléter à gauche.
        L = ... # on complète avec ce nombre de 0 à gauche. Aide : [0]*2 donne [0,0]

    L_sortie = []
    for k in range(... # combien y aura-t-il d'éléments dans la liste de sortie ?
        L_sortie.append(... # on « range » la kième liste de taille 4
    return L_sortie
```

Q7 – Écrire la fonction `decoupe_4(L)` ainsi décrite, et la tester, notamment sur les exemples donnés.

Pour passer d'un nombre binaire à une écriture hexadécimale, on peut regrouper les bits 4 par 4 (puisque 4 bits permettent d'écrire une valeur décimale entre 0 et 15, c'est-à-dire un seul digit en hexadécimal), puis interpréter la valeur décimale de chaque lot de 4 bits, pour enfin lui affecter une seule valeur hexadécimale.

Exemple :

En binaire (bits regroupés 4 par 4) :	1010	0001	1111	0101
Conversion de chaque lot de 4 en décimal :	10	1	15	5
Interprétation de chacun en hexadécimal (ici <i>str</i>) :	"A"	"1"	"F"	"5"

Q8 – En vous aidant des **trois** questions précédentes, écrire la fonction `bin_hexa(L)` qui prend en argument une liste de bits (valeur 0 ou 1) correspondant à l'écriture binaire d'un nombre, et qui renvoie **un seul str** correspondant à l'écriture hexadécimale de ce même nombre.

Tests :

```
print(bin_hexa([1,0,1,0,0,0,0,1,1,1,1,0,1,0,1])) # doit afficher "A1F5"
print(bin_hexa([1,1,1,1,0])) # doit afficher "1E"
```

Q9 –  En vous appuyant sur les fonctions nécessaires dans ce qui précède, écrire une fonction `decim_hexa(N)` qui prend en argument un nombre (décimal) N et renvoie son écriture en hexadécimal.

Aide : cette fonction peut tenir en 2 lignes (`def decim_hexa(N) :`
`return ...`).

Tests :

```
print(decim_hexa(201))           # doit afficher "C9"
print(decim_hexa(789463950))     # doit afficher "2F0E438E"
```

Remarque : on voit bien ici que l'écriture hexadécimale (8 digits 2F0E 438E) est plus « compacte » que l'écriture décimale (9 digits 789463950), cet écart de nombre de digits nécessaires se creusant quand les valeurs décrites sont vraiment élevées.

3 / Vérification : conversion hexadécimal → décimal

Dans cette dernière partie, on suppose qu'on veut vérifier nos résultats des parties précédentes. À l'aide des parties 1 et 2 nous avons pu convertir un nombre décimal en hexadécimal (en passant par une écriture binaire). Il s'agit ici de faire la conversion inverse (plus facile), c'est-à-dire repasser le nombre hexadécimal en décimal, pour vérifier qu'on « retombe » bien sur le nombre initial.

Q10 –  Écrire une fonction `hexa_decim_1digit(H)` prenant en argument un caractère (c'est-à-dire un *str* de taille 1) hexadécimal (qui peut donc valoir "0", "1", ... "9", "A", "B", ... "F") et qui renvoie la valeur décimale (*int*) correspondante.

Trame : la trame suivante est imposée, à copier-coller et compléter.

```
def hexa_decim_1digit(H):
    val_possibles = "0123456789ABCDEF"
    for k in range(...):          # on va chercher l'index de H dans val_possibles
        if ...:
            return ...
```

Tests :

```
print(hexa_decim_1digit("0"))     # doit afficher 0
print(hexa_decim_1digit("8"))     # doit afficher 8
print(hexa_decim_1digit("A"))     # doit afficher 10
print(hexa_decim_1digit("F"))     # doit afficher 15
```

On rappelle que si $H = (H_i)_{0 \leq i \leq N-1}$ désigne un nombre hexadécimal, on a :

$$H = H_0 \times 16^{N-1} + H_1 \times 16^{N-2} + \dots + H_{N-2} \times 16 + H_{N-1} \times 1 = \sum_{i=0}^{N-1} H_i \times 16^{N-1-i}$$

Q11 –  En vous appuyant sur la fonction précédente qui permet d'interpréter les valeurs des H_i , écrire une fonction `hexa_decim(H)` prenant en argument un hexadécimal (c'est-à-dire un *str* de taille quelconque) et qui renvoie la valeur décimale (*int*) correspondante.

Tests :

```
print(hexa_decim("C9"))           # on doit retrouver 201
print(hexa_decim("2F0E438E"))     # on doit retrouver 789463950
```