Bases et conversions

Représentation de l'information

Introduction

Pour compter, nous utilisons la **base 10** : dix chiffres, de 0 à 9, et un regroupement par paquets de dix (dix unités font une dizaine, dix dizaines une centaine, etc.). C'est un système **positionnel** : la valeur d'un chiffre dépend de sa place dans le nombre. Dans 374, le 7 vaut sept dizaines ; dans 723, il vaut sept centaines.

Autrement dit, 374 est une écriture abrégée de $3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$. Rien n'oblige à grouper par dix : en groupant par deux, par huit ou par seize, on obtient d'autres bases, qui suivent exactement le même principe.

Écrire un nombre dans une base

Dans une base b, on dispose de b chiffres, de 0 à $b-1$. En partant de la droite, le premier chiffre compte les unités, le deuxième les paquets de b, le troisième les paquets de b^2, et ainsi de suite. Le **poids** du chiffre placé au rang k (rangs comptés à partir de 0, depuis la droite) est donc b^k.

On note la base en indice : 101_{10} se lit « cent un », alors que 101_2 vaut $1 \times 4 + 0 \times 2 + 1 \times 1 = 5$. Sans indice, un nombre est écrit en base 10.

— **La base 2 (binaire).** Un circuit électronique distingue facilement deux états, notés 1 et 0. Un chiffre binaire s'appelle un **bit** (*binary digit*) et un groupe de huit bits un **octet**. Toutes les données d'un ordinateur sont finalement des suites de bits.

— **La base 16 (hexadécimal).** Elle donne des suites de bits une écriture quatre fois plus courte. On complète les chiffres 0 à 9 par A, B, C, D, E et F, qui valent 10, 11, 12, 13, 14 et 15.

Les seize premiers entiers dans les trois bases (écritures binaires complétées à quatre chiffres) :

Base 10	0	1	2	3	4	5	6	7
Base 2	0000	0001	0010	0011	0100	0101	0110	0111
Base 16	0	1	2	3	4	5	6	7

Base 10	8	9	10	11	12	13	14	15
Base 2	1000	1001	1010	1011	1100	1101	1110	1111
Base 16	8	9	A	B	C	D	E	F

Du binaire au décimal

Méthode. On écrit sous chaque bit son poids (1, 2, 4, 8, 16… en partant de la droite), puis on additionne les poids des bits qui valent 1.

Exemple. Que vaut 10110110_2 ?

Bit	1	0	1	1	0	1	1	0
Poids	128	64	32	16	8	4	2	1

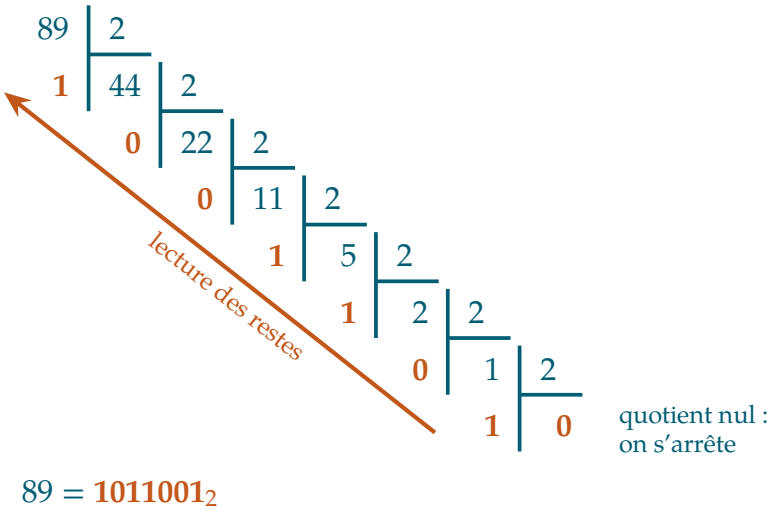
On additionne les poids placés sous un 1 : $128 + 32 + 16 + 4 + 2 = 182$. Donc $10110110_2 = 182$.

À retenir. Sur un octet, le plus petit nombre est $00000000_2 = 0$ et le plus grand $11111111_2 = 255$: un octet peut prendre 256 valeurs.

Du décimal au binaire

Méthode des divisions successives. On divise le nombre par 2 : le reste (0 ou 1) donne le bit des unités. On recommence avec le **quotient**, et l'on s'arrête lorsque le quotient vaut **0**. Le nombre binaire s'obtient en lisant les restes du dernier au premier.

Exemple. Écrivons 89 en binaire.



$89 = \mathbf{1011001}_2$

Les mêmes calculs s'écrivent sous forme de divisions euclidiennes :

Division euclidienne	Quotient	Reste
$89 = 2 \times 44 + 1$	44	**1**
$44 = 2 \times 22 + 0$	22	**0**
$22 = 2 \times 11 + 0$	11	**0**
$11 = 2 \times 5 + 1$	5	**1**
$5 = 2 \times 2 + 1$	2	**1**
$2 = 2 \times 1 + 0$	1	**0**
$1 = 2 \times 0 + 1$	0	**1**

À la dernière ligne, le quotient vaut 0 : on s'arrête. Attention à ne pas s'arrêter une ligne trop tôt, quand le quotient vaut 1 : la division de 1 par 2 fournit encore un reste, le bit de gauche, qui vaut toujours 1.

En lisant les restes de bas en haut : $89 = 1011001_2$, soit 01011001_2 sur un octet. Vérification : $64 + 16 + 8 + 1 = 89$.

Pourquoi cela marche. Chaque division par 2 décale le nombre d'un rang vers la droite : le reste est le bit qui sort, le quotient est le nombre formé par les bits restants. Quand le quotient vaut 0, il ne reste plus aucun bit.

Autre méthode. On retire la plus grande puissance de 2 possible, et on recommence avec ce qui reste : $89 = 64 + 25$, $25 = 16 + 9$, $9 = 8 + 1$. Les bits de poids 64, 16, 8 et 1 valent 1 : on retrouve 1011001_2.

En Python

```
print(bin(89))              # affiche 0b1011001
print(int('1011001', 2))    # affiche 89
print(hex(89))              # affiche 0x59
print(int('59', 16))        # affiche 89
```

Le préfixe `0b` signale une écriture binaire et `0x` une écriture hexadécimale. Le second paramètre de `int` indique la base de la chaîne.

Entre binaire et hexadécimal

Comme $16 = 2^4$, un chiffre hexadécimal correspond exactement à quatre bits.

Du binaire vers l'hexadécimal. On découpe le nombre en paquets de quatre bits **en partant de la droite** (le dernier paquet est complété à gauche par des zéros), puis on remplace chaque paquet par son chiffre hexadécimal. Exemple avec 10110110_2 :

Paquets de 4 bits	1011	0110
Valeur	11	6
Chiffre hexadécimal	B	6

Donc $10110110_2 = B6_{16}$. Vérification : $11 \times 16 + 6 = 182$.

De l'hexadécimal vers le binaire. On remplace chaque chiffre par ses quatre bits. Exemple avec $2C9_{16}$:

Chiffre hexadécimal	2	C	9
Quatre bits	0010	1100	1001

Donc $2C9_{16} = 1011001001_2$.

Du décimal à l'hexadécimal

On divise cette fois par 16. Les restes vont de 0 à 15 ; ceux qui dépassent 9 sont remplacés par une lettre. On s'arrête quand le quotient vaut 0. Exemple avec 2026 :

Division euclidienne	Quotient	Reste
$2026 = 16 \times 126 + 10$	126	**10**, soit **A**
$126 = 16 \times 7 + 14$	7	**14**, soit **E**
$7 = 16 \times 0 + 7$	0	**7**

En lisant les restes de bas en haut : $2026 = 7EA_{16}$. Vérification : $7 \times 256 + 14 \times 16 + 10 = 2026$.

Où rencontre-t-on l'hexadécimal ?

— **Les couleurs du Web** : `#2E86C1` décrit une couleur par trois octets, rouge $2E_{16} = 46$, vert $86_{16} = 134$, bleu $C1_{16} = 193$.

— **Les adresses mémoire**, affichées par exemple sous la forme `0x7ffd3a10`.

— **Les adresses MAC** des cartes réseau, formées de six octets : `3C:52:82:A7:1E:F4`.

Addition en binaire

On pose l'addition comme en base 10, colonne par colonne en partant de la droite ; une retenue apparaît dès que la somme d'une colonne atteint 2 :

— $0 + 0 = 0$, $0 + 1 = 1$;

— $1 + 1 = 2 = 10_2$: on écrit 0 et on retient 1 ;

— $1 + 1 + 1 = 3 = 11_2$: on écrit 1 et on retient 1.

Exemple. Calculons $0110_2 + 0111_2$ (le tableau se remplit de droite à gauche) :

Rang	3	2	1	0
Retenues	1	1		
Premier nombre	0	1	1	0
Second nombre	0	1	1	1
Somme	**1**	**1**	**0**	**1**

Rang 0 : $0 + 1 = 1$. Rang 1 : $1 + 1 = 10_2$, on écrit 0 et on retient 1. Rang 2 : $1 + 1 + 1 = 11_2$, on écrit 1 et on retient 1. Rang 3 : $0 + 0 + 1 = 1$. Le résultat $1101_2 = 13$ correspond bien à $6 + 7$.

Quand les nombres occupent un nombre fixé de bits, une retenue produite par la colonne de gauche n'a plus de place : c'est un **dépassement de capacité**. Sur un octet, $11111111_2 + 1$ donne 00000000_2.

Exercices

Exercice 1 Donner l'écriture décimale des nombres suivants :

1. 1101_2
2. 10010110_2
3. 11111111_2

Exercice 2 Écrire en binaire, sur un octet, les nombres suivants, en détaillant les divisions successives :

1. 37
2. 128
3. 200

Exercice 3 Recopier et compléter le tableau : chaque ligne contient un même nombre écrit dans les trois bases.

Base 2	Base 10	Base 16
1101011010_2		
	1789	
		$B2E7_{16}$

Exercice 4

1. Combien de valeurs différentes peut-on écrire avec 4 bits ? avec 8 bits ? avec n bits ?
2. Quel est le plus grand nombre que l'on peut écrire avec 10 bits ?
3. Combien de bits faut-il au minimum pour écrire le nombre 500 ?

Exercice 5 La couleur `#F4A261` est une teinte orangée. Donner, en décimal puis sur huit bits, la valeur de chacune de ses trois composantes (rouge, vert, bleu).

Exercice 6 Poser les additions suivantes en binaire, puis vérifier en base 10 :

1. $1001_2 + 0101_2$
2. $10111_2 + 01101_2$

Exercice 7 Écrire une fonction `vers_base(n, b)` qui renvoie, sous forme de chaîne de caractères, l'écriture du nombre entier positif `n` dans la base `b` (avec `b` compris entre 2 et 16), en utilisant la méthode des divisions successives. Par exemple, `vers_base(2026, 16)` doit renvoyer `'7EA'`.