Numérique et Sciences Informatiques
Chapitre II - Les bases

Classe de première

I. Définitions

Nous utilisons dans la vie courante le système décimal, où on peut écrire n'importe quel nombre entier comme une succession de chiffres dont chacun est pris dans l'ensemble des dix chiffres 0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 (avec un signe positif ou négatif), comme par exemple le nombre $a = 107$.
Dans cette écriture 107, le chiffre 1 par exemple ne vaut pas 1 mais il est « pondéré » par la valeur 100 c'est-à-dire qu'il vaut 1×100 (c'est ce qu'on appelle une numérotation de position) et on a tous appris à l'école élémentaire que cette numérotation de position repose sur des « groupements successifs par paquets de 10 des 107 unités présentes », c'est-à-dire en clair que $107 = 1 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$
Cette représentation à l'aide de puissances successives de 10 s'appelle une écriture du nombre a en base 10 (système décimal).
Mais, pour représenter des nombres dans un ordinateur ou, plus généralement, en électronique numérique, il est souvent plus adapté d'exprimer les nombres dans d'autres bases de numération, comme par exemple la base 2 (système binaire) ou la base 16 (système hexadécimal).

I.1. La base 2

Définition

La base 2, ou système binaire, ne possède que 2 chiffres 0 et 1.

I.1.a. Du binaire vers le système décimal

Propriété

Soit un nombre du système binaire $\overline{x_n x_{n-1} ... x_2 x_1 x_0}$ où les x_i valent 0 ou 1.
Sa représentation dans le système décimal est

$$x = \sum_{i=0}^{n} x_i \times 2^i = x_n \times 2^n + x_{n-1} \times 2^{n-1} + ... + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

Exemple

On considère le nombre binaire 10110.
$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 16 + 0 + 4 + 2 + 0 = 22_{10}$

Remarque

Il faut donc connaître ses puissances de 2.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
2^i	1	2	4	8	16	32	64	128	256	512	1024	2048	4096

I.1.b. Du système décimal vers le binaire

Propriété

Tout nombre entier positif x peut s'écrire :
$x = \sum_{i=0}^{n} x_i \times 2^i = x_n \times 2^n + x_{n-1} \times 2^{n-1} + ... + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$ où les x_i valent 0 ou 1.
Ainsi, tout nombre entier positif x s'écrit en base 2 : $\overline{x_n x_{n-1} ... x_2 x_1 x_0}$

Exemple

$13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
Ainsi 13 s'écrit en binaire 1101.
On écrit $13_{10} = 1101_2$

Remarque

Pour passer du système décimal au système binaire, on peut aussi faire les divisions successives par 2 des quotients (la première division étant celle du nombre de départ par 2) jusqu'à obtenir un quotient nul. Les restes ainsi trouvés donnent l'écriture en base 2 du nombre de départ lorsqu'on les écrit dans le sens inverse d'obtention.

Exemple

On considère le nombre 13 du système décimal.
$13 \div 2 = Q6R1$
$6 \div 2 = Q3R0$
$3 \div 2 = Q1R1$
$1 \div 2 = Q0R1$
Ainsi la représentation binaire du nombre 13 est 1101 (les restes dans l'ordre inverse d'obtention).

I.2. La base 16

La base 2 est peu pratique à écrire car un nombre est représenté en base 2 avec beaucoup de chiffres.
En informatique, on utilise souvent la base 16 qui nécessite 4 fois moins de chiffres que le système binaire pour représenter un même nombre.

Définition

La base 16, ou système hexadécimal, possède 16 chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e et f.

Remarque

Il faut connaître la conversion des différents chiffres dans les différentes bases.

système décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
système hexadécimal	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
système binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

I.2.a. De l'hexadécimal vers le système décimal

Propriété

Soit un nombre du système hexadécimal $\overline{x_n x_{n-1} ... x_2 x_1 x_0}$ où les x_i sont des chiffres du système hexadécimal.
En appelant y_i la représentation dans le système décimal des chiffres x_i du système hexadécimal, sa représentation dans le système décimal est

$$y = \sum_{i=0}^{n} y_i \times 16^i = y_n \times 16^n + y_{n-1} \times 16^{n-1} + ... + y_2 \times 16^2 + y_1 \times 16^1 + y_0 \times 16^0$$

Exemple

On considère le nombre hexadécimal $a5d$.
On rappelle que $a_{16} = 10_{10}$, $5_{16} = 5_{10}$ et $d_{16} = 13_{10}$.
$a5d_{16} = 10 \times 16^2 + 5 \times 16^1 + 13 \times 16^0 = 10 \times 256 + 5 \times 16 + 13 \times 1 = 2560 + 80 + 13 = 2653_{10}$

I.2.b. Du système décimal vers l'hexadécimal

Propriété

Tout nombre entier positif x peut s'écrire :
$x = \sum_{i=0}^{n} y_i \times 16^i = y_n \times 16^n + y_{n-1} \times 16^{n-1} + ... + y_2 \times 16^2 + y_1 \times 16^1 + y_0 \times 16^0$ où les x_i sont des nombres compris entre 0 et 15 inclus.
En appelant x_i la représentation dans le système hexadécimal des nombres y_i du système décimal, le nombre entier positif x s'écrit en base 16 : $\overline{x_n x_{n-1} ... x_2 x_1 x_0}$

Exemple

$713 = 2 \times 256 + 12 \times 16 + 9 \times 1 = 2 \times 16^2 + 12 \times 16^1 + 9 \times 16^0$
On rappelle que $2_{10} = 2_{16}$, $12_{10} = c_{16}$ et $9_{10} = 9_{16}$.
Ainsi 713 s'écrit en hexadécimal $2c9$.
On écrit $713_{10} = 2c9_{16}$

Remarque

Pour passer du système décimal au système hexadécimal, on peut aussi faire les divisions successives par 16 des quotients (la première division étant celle du nombre de départ par 16) jusqu'à obtenir un quotient nul. Les restes ainsi trouvés donnent l'écriture en base 16 du nombre de départ lorsqu'on les écrit dans le sens inverse d'obtention.

Exemple

On considère le nombre 713 du système décimal.
$713 \div 16 = Q44R9$
$44 \div 16 = Q2R12$
$2 \div 16 = Q0R2$
On rappelle que $2_{10} = 2_{16}$, $12_{10} = c_{16}$ et $9_{10} = 9_{16}$.
Ainsi la représentation binaire du nombre 713 est $2c9$ (les restes dans l'ordre inverse d'obtention).

I.2.c. De l'hexadécimal vers le binaire

Propriété

Pour convertir de la base 16 en base 2, il suffit de convertir en binaire chaque chiffre hexadécimal

Exemple

On considère le nombre $ae3_{16}$ du système hexadécimal.
La représentation binaire du nombre $ae3_{16}$ est 101011100011.

I.2.d. Du binaire vers l'hexadécimal

Propriété

Pour convertir de la base 2 en base 16, il suffit de :

- faire des groupes de 4 bits en partant de la droite
- compléter éventuellement par des 0 le groupe le plus à gauche
- convertir en hexadécimal, groupe par groupe

Exemple

On considère le nombre 111010 du système binaire.
$110010 = 0011\ 1010$
Or $0011_2 = 3_{16}$ et $1010_2 = a_{16}$.
Donc $111010_2 = 3a_{16}$

I.3. Exercices

I.3.a. Exercice 1

Donner la représentation binaire :

- 43_{10}
- 125_{10}
- 27_{16}
- ae_{16}

I.3.b. Exercice 2

Donner la représentation hexadécimale :

- 43_{10}
- 125_{10}
- 10100110_2
- 11000001101_2

I.3.c. Exercice 3

Donner la représentation décimale :

- 43_{16}
- $1ae5_{16}$
- 10010110_2
- 11001011101_2

II. Lien avec l'informatique

II.1. Le bit

Le bit est la quantité élémentaire d'information (bit pour Binary digIT). Il ne possède que 2 états : 0 ou 1.
Ces valeurs, selon le contexte, peuvent correspondre à :

- 0 ou 1 (numérique)
- faux ou vrai (logique)
- fermé ou ouvert (interrupteurs)
- nord ou sud (magnétique)
- noir ou blanc (optique)
- Absence ou présence de trou (carte perforée)
- Pile ou face
- Pair ou impair

II.2. L'octet

Pour des raisons pratiques, les bits sont regroupés par paquets adjacents pour représenter de l'information : un octet (byte) est constitué de 8 bits (il peut représenter 256 valeurs). Historiquement cela correspondait au codage d'un caractère. L'octet est choisi comme unité pour représenter les capacités mémoire.

Voici la représentation d'un octet, écrit en base 2.

1	1	0	1	0	1	1	0
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Si l'on numérote les bits de 0 à 7, le bit numéro 0 est appelé bit de poids faible et le bit numéro 7 est appelé bit de poids fort. Dans notre exemple, le bit de poids forts est donc $b_7 = 1$ et celui de poids faible est $b_0 = 0$.
Un octet peut aussi être représenté par un nombre hexadécimal composé de 2 chiffres : dans notre exemple, il serait représenté par $D6$

II.3. Les unités

Ordre de grandeur	Système International		Préfixes binaires	
	Unité	Valeur	Unité	Valeur
10^3	kilobit (kb)	10^3 bits	kibibit (Kibit)	2^{10} bits
10^6	mégabit (Mb)	10^6 bits	mébibit (Mibit)	2^{20} bits
10^9	gigabit (Gb)	10^9 bits	gibibit (Gibit)	2^{30} bits
10^{12}	térabit (Tb)	10^{12} bits	tébibit (Tibit)	2^{40} bits
10^{15}	pétabit (Pb)	10^{15} bits	pébibit (Pibit)	2^{50} bits
10^{18}	exabit (Eb)	10^{18} bits	exbibit (Eibit)	2^{60} bits
10^{21}	zettabit (Zb)	10^{21} bits	zébibit (Zibit)	2^{70} bits
10^{24}	yottabit (Yb)	10^{24} bits	yobibit (Yibit)	2^{80} bits

Ordre de grandeur	Système International		Préfixes binaires	
	Unité	Valeur	Unité	Valeur
10^3	kilooctet (ko)	10^3 octets	kibibit (Kio)	2^{10} octets
10^6	mégaoctet (Mo)	10^6 octets	mébibit (Mio)	2^{20} octets
10^9	gigaoctet (Go)	10^9 octets	gibibit (Gio)	2^{30} octets
10^{12}	téraoctet (To)	10^{12} octets	tébibit (Tio)	2^{40} octets
10^{15}	pétaoctet (Po)	10^{15} octets	pébibit (Pio)	2^{50} octets
10^{18}	exaoctet (Eo)	10^{18} v	exbibit (Eio)	2^{60} octets
10^{21}	zettaoctet (Zo)	10^{21} octets	zébibit (Zio)	2^{70} octets
10^{24}	yottaoctet (Yo)	10^{24} octets	yobibit (Yio)	2^{80} octets

II.4. Exercices

II.4.a. Exercice 1 - Taille d'une image

Soit une image A4 en 600 ppp (point par pouces) codée en 24 bits (3 octets).

1. Quelle est le poids de cette image en Mo ?
2. Une image JPEG de 35 Mpixels a une taille de 14 Mio. Que peut-on en conclure ?

On rappelle que :

- *Le format A4 correspond à une feuille de* 21 *cm sur* $29,7$ *cm.*
- 1 *pouce* $\approx 2,54$ *cm*

II.4.b. Exercice 2 - Taille d'un disque audio

Le format CD audio est le suivant : stéréo, échantillonnage 44,1 kHz, quantification en 16 bits (ce qui veut dire qu'une information prend un poids de 16 bits).

1. Quelle est la taille d'un CD audio de 74 min ?
2. Un CD en Mp3 a une taille de 60 Mio. Que peut-on en conclure ?

II.4.c. Exercice 3 - Taille d'un film muet

Un film muet dure 90 minutes. Son format est 720 x 576 pixels (qualité DVD) codé en 16 bits.

1. Quelle est la taille de ce film en Go ?
2. Un DVD a une capacité de $4,7$ à 17 Go et un Blu-ray (meilleure qualité que le DVD) de 25 à 50 Go. Que peut-on conclure ?

II.4.d. Exercice 4 - Un autre exemple

La BNF pour son dépôt légal de musique, a une collection de 350000 disques microsillons et 150000 CD.
On peut supposer que cela représente environ l'équivalent de 500000 CD d'une heure.

1. Quel est le volume de donnée nécessaire pour stocker toute la musique éditée en France en utilisant la compression MP3. Vous exprimerez ce volume en Tébioctets.
2. Aujourd'hui 2 Tio coutent 50€. Combien cela coûterait-il d'avoir toute la musique de la BNF ?
3. Un ordinateur actuel a un disque dur de 4 Tio. On suppose que la capacité des disques durs double tous les 9 mois. Dans combien d'années pourrez vous avoir toute la musique éditée en France sur votre ordinateur ?

III. Représentation des entiers

III.1. Les entiers non signés

Les entiers non signés sont les entiers naturels des mathématiques. Ils sont codés généralement sur 8, 16, 32 ou 64 bits.

Si on considère un mot de n bits, on peut y représenter les entiers non signés de 0 à $2^n - 1$. Ainsi on a le tableau suivant :

Nombre de bits	Intervalle
8	$[0; 2^8 - 1] = [0; 255]$
16	$[0; 2^{16} - 1] = [0; 65\,535]$
24	$[0; 2^{24} - 1] = [0; 4\,294\,967\,295]$
32	$[0; 2^{32} - 1] = [0; 18\,446\,744\,073\,709\,551\,615]$

Propriété

Pour trouver la représentation binaire d'un entier non signé, il suffit de convertir en binaire ledit nombre entier.

Exemple

Le nombre 34 est codé sur 8 bits par :
$34_{10} = 0010\,0010_2$

III.2. Les entiers signés

Les entiers signés sont les entiers relatifs des mathématiques. Ils sont aussi codés généralement sur 8, 16, 32 ou 64 bits.

Propriété

Pour représenter des entiers signés, on utilise le **complément à 2**, c'est-à-dire, pour un mot de taille n, on code un entier x par :

- x si $x \geqslant 0$
- $2^n + x$ si $x < 0$

Donc si on considère un mot de n bits, on peut y représenter les entiers non signés de -2^{n-1} à $2^{n-1} - 1$. Ainsi on a le tableau suivant :

Nombre de bits	Intervalle
8	$[-2^7, 2^7 - 1] = [-128, 127]$
16	$[-2^{15}, 2^{15} - 1] = [-32\,768, 32\,767]$
24	$[-2^{31}, 2^{31} - 1] = [-2\,147\,483\,648, 2\,147\,483\,647]$
32	$[-2^{63}, 2^{63} - 1] = [-9\,223\,372\,036\,854\,775\,808, 9\,223\,372\,036\,854\,775\,807]$

Exemple

Le nombre -34 est codé sur 8 bits par :
$2^8 - 34 = 256 - 34 = 222_{10} = 1101\,1110_2$

Remarque

Une autre façon de calculer le complément à 2 est d'inverser les bits et d'ajouter 1.

Exemple

$34_{10} = 0010\,0010_2$
Inversons les bits : $1101\,1101_2$
Ajoutons 1 : $1101\,1110_2$
Donc -34 est codé sur 8 bits par $1101\,1110_2$

Remarque

La somme de deux entiers écrit respectivement sur n et p bits (avec $n \leqslant p$) sera écrit sur p ou $p+1$ bits.
Le produit de deux entiers écrit respectivement sur n et p bits sera écrit sur $n+p-1$ ou $n+p$ bits.

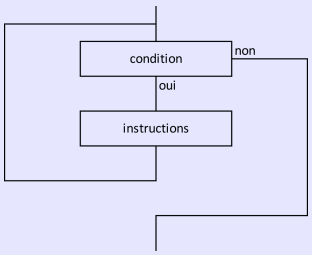
IV. Notions de Python

IV.1. Les boucles non bornées

Lorsqu'on veut répéter une série d'instructions mais qu'on ne sait pas quand/si cela peut s'arrêter, on utilise une boucle non bornée, ou boucle Tant que, ou boucle while.

Définition

Une boucle Tant que est constituée d'une condition et d'une séquence d'instructions.
Lors de l'exécution de la boucle, la condition est évaluée. Si cette condition est vraie, la séquence d'instruction est alors évaluée.
La boucle est répétée jusqu'à ce que la condition soit fausse.



En Python, on écrit :

```
while condition:
    instructions
```

On peut l'écrire en français :
Tant que la condition est vérifiée, on exécute les instructions.

Exemple

```
i = 5
while i <= 12:
    print(i)
    i = i + 1
```

Ce script affichera les nombres de 5 à 12 inclus, les uns en-dessous des autres.

Remarque

Si la condition porte sur une variable, il faut absolument qu'elle soit initialisée avant la boucle.
De plus, la variable en question doit s'approcher du cas d'arrêt à chaque entrée dans la boucle, sous peine de créer une boucle qui se répète à l'infini.

IV.2. Représentation des entiers

La fonction `bin(nb)` permet de donner la représentation binaire du nombre entier `nb` du système décimal.
La fonction `hex(nb)` permet de donner la représentation hexadécimale du nombre entier `nb` du système décimal.
La fonction `int(repr, base)` permet de donner la représentation entière dans le système décimal du nombre `repr` représenté dans la base `base`.
Python représente tous les nombres dans le système décimal. Ainsi, si on écrit un nombre binaire fans la console, le résultat sera ce nombre représenté dans le système décimal.

Exemple

```
>>> bin(35)
'0b100011'
>>> hex(35)
'0x23'
>>> int('0b100011', 2)
35
>>> int('100011', 2)
35
>>> 0b100011
35
>>> int('0x23', 16)
35
>>> int('23', 16)
35
>>> 0x23
35
```

Remarque

La fonction `bin(nb)` attend un entier en paramètre et renvoie une chaîne de caractères (voir chapitre III)
La fonction `int(repr, base)` attend une chaîne de caractères `repr` et un entier `base` supérieur ou égal à 2 et renvoie un entier.

Remarque

Dans un langage où les entiers sont de taille fixe (par exemple sur 32 bits), ajouter 1 à $2^{31}-1$ donnera un nombre négatif (et cela donnera -2^{31} si les nombres sont représentés en complément à 2).

Il n'est pas possible d'illustrer cela sous Python, ou alors de manière très détournée, car les entiers peuvent être arbitrairement grand (l'unique limite étant la mémoire disponible sur la machine).

IV.3. Exercices

IV.3.a. Exercice 1

En utilisant les fonctions `bin()`, `hex()` et `int()`, créer les fonctions :

1. `dec_to_bin(nb: int) -> str` qui donne la représentation binaire d'un nombre du système décimal.
2. `dec_to_hex(nb: int) -> str` qui donne la représentation hexadécimale d'un nombre du système décimal.
3. `bin_to_dec(nb: str) -> int` qui donne la représentation dans le système décimal d'un nombre binaire.
4. `hex_to_dec(nb: str) -> int` qui donne la représentation dans le système décimal d'un nombre du système hexadécimal.
5. `bin_to_hex(nb: str) -> str` qui donne la représentation hexadécimale d'un nombre binaire.
6. `hex_to_bin(nb: str) -> str` qui donne la représentation binaire d'un nombre hexadécimal.

IV.3.b. Exercice 2

Sans utiliser les fonctions `bin()`, `hex()` et `int()`, créer les fonctions :

1. Une fonction `dec_to_bin(nb: int) -> int` qui donne la représentation binaire d'un nombre du système décimal.
2. Une fonction `bin_to_dec(nb: int) -> int` qui donne la représentation hexadécimale d'un nombre du système décimal.

Deux fonctions sont données dans le fichier Python :

- `chiffre(nb: int, n: int) -> int` *qui renvoie le n-ième chiffre de nb en partant de la droite (emplacement 0)*
- `placer(nb: int, chif: int, n: int) -> int` *qui place le chiffre chif à l'emplacement n du nombre nb, quitte à supprimer l'ancien chiffre puis renvoie le nouveau nombre.*

IV.3.c. Exercice 3

On s'intéresse, dans cet exercice, à l'addition binaire de deux nombres binaires.

1. Poser l'addition $0011\,0101 + 0110\,0111$.
2. Quel opérateur binaire renvoie l'unité de la somme de deux nombres binaires ?
3. Soit a et b les opérandes d'une addition binaire à un chiffre et r une retenue éventuelle d'une opération précédente. Déterminer une expression de la nouvelle retenue en fonction de a, b et r.
4. Créer une fonction `somme(nb1: int, nb2: int) -> int` qui renvoie la représentation binaire de la somme de deux nombres binaires.

Trois fonctions sont données dans le fichier Python :

- `chiffre(nb: int, n: int) -> int` *qui renvoie le n-ième chiffre de nb en partant de la droite (emplacement 0)*
- `placer(nb: int, chif: int, n: int) -> int` *qui place le chiffre chif à l'emplacement n du nombre nb, quitte à supprimer l'ancien chiffre puis renvoie le nouveau nombre.*
- `nombre_chiffres(nb: int) -> int` *qui renvoie le nombre de chiffres du nombre mis en paramètre.*