Écriture d'un entier positif dans une base b >= 2

Système décimal ou base 10

Dans le système décimal que l'on utilise tous les jours, les nombres sont écrit à l'aide des 10 chiffres bien connus : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

C'est la position d'un chiffre dans un nombre qui indique son importance, ou **poids**, dans ce nombre. Par exemple, le nombre qui s'écrit 2083 est égal à 2 milliers plus 8 dizaines plus 3 unités. Il n'est pas égal au nombre 8320, même s'il s'écrit avec les mêmes chiffres 0, 2, 8 et 3.

Tout nombre entier naturel peut s'écrire comme combinaison linéaire de puissance de 10. Par exemple, les chiffres du nombre 2083 correspondent à :

chiffre	**2**	**0**	**8**	**3**
i	3	2	1	0
10^i	10^3	10^2	10^1	10^0
combinaison	2×10^3	0×10^2	8×10^1	3×10^0

$$2083 = 2 \times 10^3 + 0 \times 10^2 + 8 \times 10^1 + 3 \times 10^0$$

De manière générale, un nombre n qui s'écrit dans le système décimal avec p chiffres $d_{p-1}d_{p-2}\dots d_2d_1d_0$ (chaque d_i est un chiffre valant entre 0 et 9)[1] est égal à :

$$n = d_{p-1} \times 10^{p-1} + d_{p-2} \times 10^{p-2} + \dots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$$

ou encore avec la formule mathématique d'une somme de 0 à $p-1$: $n = \sum_{i=0}^{p-1} d_i \times 10^i$

Noter qu'on peut écrire 10 nombres allant de 0 à 9 avec 1 seul chiffre, 100 nombres allant de 0 à 99 avec 2 chiffres, 1000 nombres allant de 0 à 999 avec 3 chiffres, ... 10^p nombres allant de 0 à $10^p - 1$ avec p chiffres.

Quand on ajoute un chiffre 0 à droite d'un nombre n, tous les chiffres sont décalés vers la droite, les puissances de 10 correspondantes sont augmentées d'une unité, donc le nombre n est multiplié par 10.

Réciproquement, on peut écrire chacun des chiffres d'un nombre décimal n qui s'écrit dans le système décimal avec p chiffres $d_{p-1}d_{p-2}\dots d_2d_1d_0$ par un algorithme simple qui consiste à effectuer une succession de divisions entières par 10 :

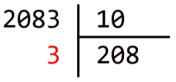
L'opérateur de division entière `//` et l'opération modulo `%` utilisés avec des entiers (de type `int`) donnent respectivement le quotient et le reste d'une division euclidienne : si `a` et `b` sont des entiers tels que $a = b \times q + r$, alors `a // b` renvoie q et `a % b` renvoie r.

- Le reste de la division entière de n par 10, `n % 10` en Python, renvoie d_0. Cela permet d'obtenir le dernier chiffre de l'écriture décimale de n.
- Le quotient de la division entière de n par 10, `n // 10` en Python, renvoie $d_{p-1}d_{p-2}\dots d_2b_1$. On remplace n par ce nombre pour trouver les autres chiffres.

Il suffit alors de répéter l'opération jusqu'à ce que n soit égal à 0, on aura bien obtenu tous les chiffres de l'écriture décimale de n.

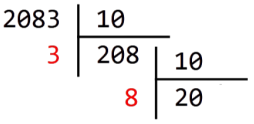
Prenons l'exemple du nombre $n = 2083$, le reste de la division entière par 10 est 3 et le quotient 208.

```
>>> 2083 % 10
3
>>> 2083 // 10
208
```



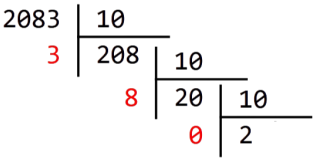
On a déjà trouvé le dernier chiffre : 3. Continuons avec 208. Le reste de la division entière de 208 par 10 est 8 et le quotient 20.

```
>>> 208 % 10
8
>>> 208 // 10
20
```



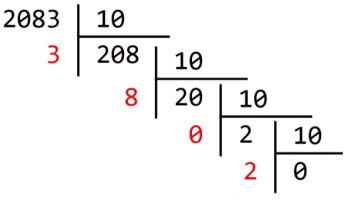
On obtient le 8. Continuons avec 20. Le reste de la division entière de 20 par 10 est 0 et le quotient 2.

```
>>> 20 % 10
0
>>> 20 // 10
2
```



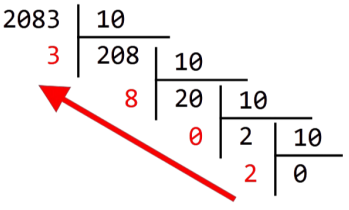
On obtient le 0. Continuons avec 2. Le reste de la division entière de 2 par 10 est 2 et le quotient 0.

```
>>> 2 % 10
2
>>> 2 // 10
0
```



On a obtenu le dernier chiffre 2. Le quotient est 0, inutile de continuer les divisions, tous les chiffres ont été trouvés.

Noter qu'on a obtenu les chiffres de l'écriture décimale de 2083, mais ⚠️ de gauche à droite, il faut donc les lire de droite à gauche pour retrouver 2083.



On peut traduire cet algorithme en Python, par exemple pour écrire une fonction `etoile` qui renvoie une chaîne de caractère composées de tous les chiffres d'un nombre entier `n` suivis d'une étoile.

```
def etoile(n):
    n_etoile = ''
    while n > 0:
```

```
        n_etoile =  str(n % 10) + '*' + n_etoile
        n = n // 10
    return n_etoile

>>> etoile(2083)
'2*0*8*3*'
```

On note que le cas ou `n` est égal à `0`, la fonction n'entre pas dans la boucle `while` et renvoie une chaîne vide. On peut traiter le cas séparément en ajoutant les lignes :

```
def etoile(n):
    if n == 0:
        return '0*'
    n_etoile = ''
    while n > 0:
        ...
```

Exercice corrigé

Un nombre harshad, ou nombre de Niven, est un entier naturel qui est divisible par la somme de ses chiffres dans une base donnée [...]. En base dix, les vingt premiers nombres harshad strictement supérieurs à 10 sont (suite A005349 de l'OEIS) : 12, 18, 20, 21, 24, 27, 30, 36, 40, 42, 45, 48, 50, 54, 60, 63, 70, 72, 80 et 81. Source : https://fr.wikipedia.org/wiki/Nombre_harshad.

Écrire un programme demande un nombre entier et affiche s'il est un nombre de harshad ou pas.

Réponse

Système binaire ou base 2

Cours

En binaire, ou base 2, les seuls chiffres utilisés pour écrire des nombres sont 0 et 1, aussi appelés « bits » pour *binary digits*, ou « chiffres binaires » en français.

8 bits forment un **octet**.

Par exemple on peut écrire 1101, que l'on note aussi 1101_2, pour indiquer qu'il est écrit en binaire.

Il convient également de ne pas lire ces nombres comme on lirait des nombres décimaux. Ainsi, 1101_2 ne se dit pas « mille cent un » mais plutôt « un un zéro un ».

Comme dans le système décimal, c'est la position qui indique le poids de chaque bit dans un nombre. Mais en binaire, c'est une combinaison linéaire de puissances de 2. Par exemple, les bits du nombre 1101_2 correspondent à :

bits	**1**	**1**	**0**	**1**
i	3	2	1	0

bits	2^3	2^2	2^1	2^0
combinaison	$1 \times 2^3 = 8$	$1 \times 2^2 = 4$	$0 \times 2^1 = 0$	$1 \times 2^0 = 1$

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$

Noter le $\ldots_{10}$ pour indiquer que 13 est un nombre en base 10.

Cours

De manière générale, un nombre n qui s'écrit dans le système binaire avec p bits $b_{p-1}b_{p-2}\ldots b_2b_1b_0$ (chaque b_i est un bit valant 0 ou 1) a une valeur décimale égale à :

$n = b_{p-1} \times 2^{p-1} + b_{p-2} \times 2^{p-2} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$

ou encore avec la formule mathématique d'une somme de 0 à $p-1$: $n = \sum_{i=0}^{p-1} b_i \times 2^i$

Écrire un nombre binaire en décimal

La formule précédente permet d'écrire facilement un nombre binaire en décimal. Il suffit de multiplier chaque bit par la puissance de 2 correspondante et de faire la somme des valeurs obtenues.

Énumérons les premiers nombres binaires et quelques autres :

binaire	combinaison	décimal
0	0	0
1	1×2^0	1
10	1×2^1	2
11	$1 \times 2^1 + 1 \times 2^0$	3
100	1×2^2	4
101	$1 \times 2^2 + 1 \times 2^0$	5
110	$1 \times 2^2 + 1 \times 2^1$	6
111	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	7
1000	1×2^3	8
$1\,0000$	1×2^4	16

binaire	combinaison	décimal
10 0000	1×2^5	32
100 0000	1×2^6	64
1000 0000	1×2^7	128
1111 1111	$1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + \ldots 1 \times 2^0$	255

Exercice corrigé

Calculer la valeur décimale des nombres binaires suivants :

- 11010
- 10101
- 11100110

Réponse

Traduisons cela en Python. Pour plus de simplicité, on peut parcourir le nombre binaire de gauche à droite.

```
def bin_to_dec(n):
    """ str -> int
    Renvoie l'écriture décimale de la chaine de caractère n représentant un nombre binaire
    """
    dec = 0
    for i in range(len(n)):
        dec = dec + int(n[-i-1]) * 2**i
    return dec
```

En Python, les nombres binaires s'écrivent avec le préfixe `0b` et l'équivalent en décimal est affiché automatique dans la console :

```
>>> 0b1101
13
```

Écrire un nombre décimal en binaire

Il existe plusieurs méthodes pour écrire un nombre décimal en binaire :

- Trouver les puissances de 2 est la méthode la plus simple qui aide à comprendre la structure du binaire, mais elle est peu utilisée en pratique.
- Effectuer des divisions successives par 2 est pratique pour un calcul algorithmique.
- Utiliser une fonction Python.

Trouver les puissances de 2

On peut faire l'opération inverse de l'écriture d'un nombre binaire en décimal en essayant de retrouver la combinaison linéaire de puissances de 2 d'un nombre binaire.

Cherchons pas exemple, l'écriture binaire du nombre 13_{10}. Il faut remplir le tableau des puissances de 2 suivant :

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	?	?	?	?	?

On pourrait commencer par remplir le tableau à droite, du côté des petites puissances de 2 : $2^0 = 1$

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	?	?	?	?	1

Puis on continue : $2^0 + 2^1 = 1 + 2 = 3$

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	?	?	?	1	1

Et encore : $2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7$

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	?	?	1	1	1

Mais quand on arrive là, on est coincé ! La puissance suivante est $2^3 = 8$, c'est trop grand pour obtenir 13 et toutes les autres puissances seront encore plus grande.

Il faut donc procéder dans l'autre sens, en partant de la gauche, du côté des grandes puissances[2].

$2^4 = 16$, c'est plus grand que 13, on ne peut pas prendre le bit correspondant, on met 0 !

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	0	?	?	?	?

$2^3 = 8$, c'est plus petit que 13, on met 1 pour le bit correspondant. Il reste $13 - 8 = 5$ à trouver.

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	0	1	?	?	?

$2^2 = 4$, c'est plus petit que 5, on met 1 pour le bit correspondant. Il reste $5 - 4 = 1$ à trouver.

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	0	1	1	?	?

$2^1 = 2$, c'est plus grand que 1, on ne peut pas prendre le bit correspondant, on met 0.

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	0	1	1	0	?

$2^0 = 1$, c'est le dernier bit que l'on cherchait, on met 1.

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
binaire	0	1	1	0	1

On a bien trouvé l'écriture binaire de 13_{10}, c'est 1101_2.

Voilà l'algorithme que l'on a suivi :

- On repère la plus grande puissance de 2 plus petite ou égale au nombre.

- On soustrait, puis on continue avec le reste.
- On met 1 si la puissance est utilisée, 0 sinon.

Exercice corrigé

Compléter le tableau pour écrire 22_{10} et 29_{10} en binaire :

i	4	3	2	1	0
2^i	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
22_{10}					
29_{10}					

Réponse

C'est une méthode simple et intuitive, mais en pratique elle est inefficace et peu utilisée, en particulier pour les grands nombres, car elle consiste à calculer et garder en mémoire de nombreuses puissances de 2 inutiles.

Effectuer des divisions successives par 2

De la même manière qu'on a utilisée précédemment pour trouver les chiffres en base 10 d'un nombre par une succession de divisions entières par 10, on peut écrire un nombre décimal n sous sa forme binaire $b_{p-1}b_{p-2}\dots b_2b_1b_0$ en effectuant des divisions entières par 2 :

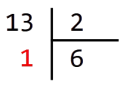
- Le reste de la division entière de n par 2, `n % 2` en Python, renvoie b_0. Cela permet d'obtenir le dernier bit de l'écriture binaire de n.
- Le quotient de la division entière de n par 2, `n // 2` en Python, renvoie $b_{p-1}b_{p-2}\dots b_2b_1$. On remplace n par ce nombre pour trouver les autres bits.

Il suffit alors de répéter l'opération jusqu'à ce que n soit égal à 0, on aura bien obtenu tous les bits de l'écriture binaire de n.

⚠️ Attention, on obtient les bits de la gauche vers la droite.

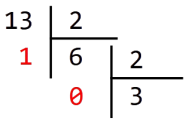
Prenons l'exemple de $n_{10} = 13$, le reste de la division entière par 2 est 1 et le quotient 6.

```
>>> 13 % 2
1
>>> 13 // 2
6
```



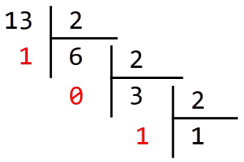
On a déjà trouvé le dernier bit : 1. Continuons avec 6. Le reste de la division entière de 6 par 2 est 0 et le quotient 3.

```
>>> 6 % 2
0
>>> 6 // 2
3
```



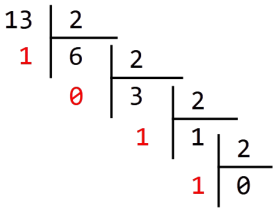
On obtient le bit 0. Continuons avec 3. Le reste de la division entière de 3 par 2 est 1 et le quotient 1.

```
>>> 3 % 2
1
>>> 3 // 2
1
```



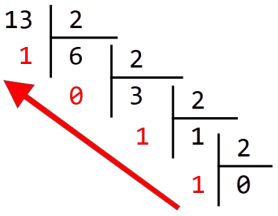
On obtient le bit 1. Continuons avec 1. Le reste de la division entière de 1 par 2 est 1 et le quotient 0.

```
>>> 1 % 2
1
>>> 1 // 2
0
```



On a obtenu le dernier bit, c'est encore 1. Le quotient est 0, inutile de continuer les divisions, tous les bits ont été trouvés.

On a obtenu les bits de l'écriture binaire de 13_{10}, mais ⚠️ de gauche à droite, il faut donc les lire de droite à gauche pour trouver 1101_2.



Exercice corrigé

Écrire les nombres suivants en binaire :

- 178
- 761

Réponse

On peut traduire cet algorithme en Python, par exemple pour écrire une fonction `dec_to_bin` qui renvoie l'écriture binaire d'un nombre entier `n` :

```
def dec_to_bin(n):
    """ int -> str
    Renvoie l'écriture binaire de l'entier n
    """
    bin = ''
    while n > 0:
        bin =  str(n % 2) + bin
        n = n // 2
    return bin
```

```
>>> bin_to_dec(13)
'1101'
```

⚠️ Il faut écrire `bin =  str(n % 2) + bin` et non pas `bin =  bin + str(n % 2)`, c''est un bug classique, car le premier bit trouvé est b_0 et le dernier est b_{p-1}.

On note que le cas ou `n` est égal à `0`, la fonction n'entre pas dans la boucle `while` et renvoie une chaîne vide. On peut traiter le cas séparément au début de la fonction :

```
def dec_to_bin(n):
    """ int -> str
    Renvoie l'écriture binaire de l'entier n
    """
    if n == 0:
        return '0'
    bin = ''
    while n > 0:
        bin =  str(n % 2) + bin
        n = n // 2
    return bin
```

Fonction Python `bin`

En Python, il existe bien sûr une fonction `bin` qui permet d'écrire un nombre décimal en binaire, mais il est souvent interdit de l'utiliser dans le cadre de NSI :

```
>>> bin(13)
'0b1101'
```

Opérations

Les quatre opérations de base (addition, soustraction, multiplication et division) restent exactement les mêmes qu'en écriture décimale, mais sont plus simples parce qu'il n'y a que les deux chiffres 0 et 1.

- Addition

  On additionne bit à bit en prenant soin d'aligner les bits de même poids à droite et on calcule de la droite vers la gauche en passant la retenue si nécessaire.

  Les additions bit à bit sont simples :

  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 10 ⚠️ passer la retenue de 1 à droite

  Exemple : Calculer $101101 + 1100$

  On aligne d'abord à gauche les bits des deux nombres l'un au dessus de l'autre. Puis on commence par ajouter les deux bits les plus à droite, $1 + 0 = 1$, puis on continue vers la gauche, $0 + 0 = 0$. On arrive à $1 + 1$, ce qui fait 10, on garde le 0 et on ajoute une retenue de 1 à gauche. On ajoute les bits suivants, avec cette retenue on a $1 + 1 + 1$, ce qui fait 11, on garde le 1 et on ajoute une nouvelle retenue de 1 encore à gauche. Les deux derniers bits ne sont que sur le premier nombre, on fait comme si c'était des 0 sur le second pour terminer

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ \ \ \ } \\ 101101 \\ +\ \ 1100 \\ \hline 111001 \end{array}$$

l'addition. On trouve le résultat final, $101101 + 1100 = 111011$, c'est l'équivalent binaire de $45 + 12 = 57$ en décimal.

Exercice corrigé

Calculer les additions suivantes en binaire :

- $10101 + 1011$
- $11101 + 11011$
- $11010 + 11010$

Réponse

- Soustraction

  Comme pour l'addition, on soustrait bit à bit en prenant soin d'aligner les bits de même poids à droite et on calcule de la droite vers la gauche en passant la retenue si nécessaire.

  Les soustractions bit à bit sont simples :

  - 0 - 0 = 0
  - 1 - 0 = 1
  - 1 - 1 = 0
  - 0 - 1. ⚠️ Il faut calculer 10 - 1 = 1 et passer la retenue de 1 à droite

  Exemple : Calculer $110001 - 11100$

  On aligne d'abord à gauche les bits des deux nombres le plus grand au dessus de l'autre. Puis on commence par soustraire les deux bits les plus à droite, $1 - 0 = 1$, et on remonte vers la gauche, $0 - 0 = 0$. On arrive à $0 - 1$, on garde le 0 et on ajoute la retenue de 1 au chiffre du bas à droite. Ici on applique la méthode française (ou méthode « par compensation »)[3] qui consiste à ajouter la retenue à la gauche du nombre en bas : $11 + 1 = 100$. On continue la soustraction en remplaçant ces deux bits 11 par 100. On obtient $0 - 0 = 0$, puis $1 - 0 = 1$, et enfin $1 - 1 = 0$. On trouve le résultat final, $110001 - 11100 = 010101$, ou 10101 en omettant le 0 inutile à gauche, c'est l'équivalent binaire de $49 - 28 = 21$ en décimal.

  ```
    110001
  -  11100
  ---------
    010101
  ```

  Exercice corrigé

  Calculer les additions suivantes en binaire :

  - $10101 - 1001$
  - $10000 - 1$
  - $1101001 - 1011011$

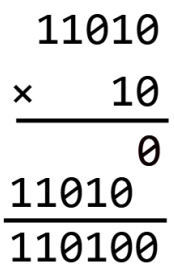
  Réponse

- Multiplication

On a fait dans un exercice précédant la multiplication d'un nombre binaire par 2 en l'ajoutant à lui-même :
$11010 + 11010 = 110100$

On aurait pu aussi poser cette multiplication bit à bit comme pour une multiplication dans le système décimal, en notant que 2 s'écrit 10 en binaire :

On commence par multiplier 11010 par le 0 de 10 et on écrit le résultat, 0, sur la première ligne. Puis on multiplie 11010 par le 1 de 10 et on écrit le résultat, 11010, sur la seconde ligne en décalant d'un bit vers la gauche. On fait ensuite la somme des deux lignes et on obtient le résultat final 110100.



On constate que multiplier un nombre binaire par 2 ($= 10_2$ en binaire) se fait en ajoutant un zéro à droite.

En effet, si on reprend la décomposition en puissances de 2 de 11010_2 (ou 26_{10}) :

$11010_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1$

quand on le multiplie par 2, toutes les puissances augmentent d'une unité. On obtient bien le nombre binaire :

$2 \times (1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1) = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^2 = 110100_2$

qui s'écrit 52_{10} en décimal.

Cours

Quand on ajoute un bit 0 à droite d'un nombre, tous les bits sont décalés vers la droite, les puissances de 2 correspondantes sont augmentées d'une unité, donc le nombre est multiplié par 2.

De la même façon, on peut multiplier deux nombres binaires ensembles.

Exercice corrigé

Calculer les additions suivantes en binaire :

- 1101×11
- 10101×101
- 1000×1000

Réponse

Hexadécimal ou base 16

Le système hexadécimal, ou base 16, est souvent utilisé en informatique, par exemple dans les adresses IPv6 ou pour définir les couleurs dans un fichier CSS, car il offre un bon compromis entre le système binaire utilisé par les machines tout en restant lisible pour les informaticiens.

Dans le système hexadécimal, ou en base 16, on utilise 16 symboles, appelés chiffres hexadécimaux : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. Voilà leur équivalent décimal et binaire :

hexadécimal	décimal	binaire	hexadécimal	décimal	binaire
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Cours

En hexadécimal, ou base 16, les nombres s'écrivent avec 10 chiffres et 6 lettres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F, appelés chiffres hexadécimaux.

Par exemple on peut écrire $B0D5$, que l'on note aussi $B0D5_{16}$ pour indiquer qu'il est écrit en hexadécimal.

Comme dans les systèmes décimal et binaire, c'est la position qui indique le poids de chaque chiffre hexadécimal dans un nombre. Mais en hexadécimal, c'est une combinaison linéaire de puissances de 16. Par exemple, les chiffres du nombre $B0D5_{16}$ correspondent à :

hexadec.	B	0	D	5
i	3	2	1	0
16^i	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 16$
combinaison	$11 \times 16^3 = 450564$	$0 \times 16^2 = 0$	$13 \times 16^1 = 208$	$5 \times 16^0 = 5$

$$B0D5_{16} = 11 \times 16^3 + 0 \times 16^2 + 13 \times 16^1 + 5 \times 16^0 = 4526910$$

Cours

De manière générale, un nombre n qui s'écrit dans le système hexadécimal avec p chiffres hexadécimaux $h_{p-1}h_{p-2}\dots h_2h_1h_0$ (chaque h_i est un chiffre hexadécimal valant 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E ou F) a une valeur décimale égale à :

$$n = h_{p-1} \times 16^{p-1} + h_{p-2} \times 16^{p-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

ou encore avec la formule mathématique d'une somme de 0 à $p-1$: $n = \sum_{i=0}^{p-1} h_i \times 16^i$

Écrire un nombre hexadécimal en décimal

La formule précédente permet d'écrire facilement un nombre hexadécimal en décimal. Il suffit de multiplier chaque chiffre hexadécimal par la puissance de 16 correspondante et de faire la somme des valeurs obtenues.

!! question "Exercice corrigé" Calculer la valeur décimale des nombres hexadécimaux suivants :

```
- A4C
- 59
- 2BF0E
```

Réponse

La traduction en Python est la même que pour le binaire, seulement il faut remplacer les lettres hexadécimales par leur valeur équivalente, par exemple en utilisant un dictionnaire Python.

```
def hex_to_dec(n):
    """ str -> int
    Renvoie l'écriture décimale de la chaine de caractère n représentant un nombre hexadécimal
    """
    valeur_hex = {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
        'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15}

    dec = 0
    for i in range(len(n)):
        dec = dec + valeur_hex[n[-i-1]] * 16**i
    return dec
```

En Python, les nombres hexadécimaux s'écrivent avec le préfixe `0x` (les lettres en minuscules ou majuscules) et l'équivalent en décimal est automatique affiché dans la console :

```
>>> 0xb0d5
45269
```

Écrire un nombre décimal en hexadécimal

Comme pour le binaire, on peut écrire un nombre décimal n sous sa forme hexadécimale, $h_{p-1}h_{p-2}\dots h_2h_1h_0$, par une suite de divisions entières par 16 :

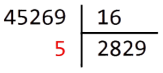
- Le reste de la division entière de n par 16, `n % 16` en Python, renvoie h_0. Cela permet d'obtenir le dernier chiffre de l'écriture hexadécimale de n.
- Le quotient de la division entière de n par 16, `n // 16` en Python, renvoie $h_{p-1}h_{p-2}\dots h_2h_1$. On remplace n par ce nombre pour trouver les autres chiffres hexadécimaux.

Il suffit alors de répéter l'opération jusqu'à ce que n soit égal à 0, on aura bien obtenu tous les chiffres de l'écriture hexadécimale de n.

⚠️ Attention, on obtient les bits de la gauche vers la droite.

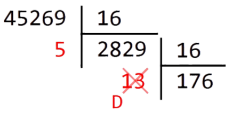
Prenons l'exemple de $n_{10} = 45269$, le reste de la division entière par 16 est 5 et le quotient 2829.

```
>>> 45269 % 16
5
>>> 45269 // 16
2829
```



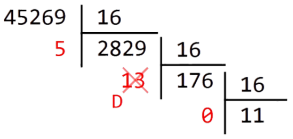
On a déjà trouvé le dernier chiffre hexadécimal : 5. Continuons avec 2829. Le reste de la division entière de 2829 par 16 est 13 et le quotient 1763.

```
>>> 2829 % 16
13
>>> 2829 // 16
176
```



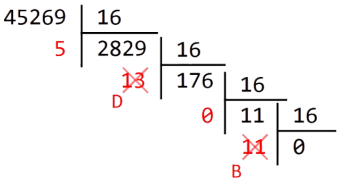
⚠️ Attention, ici le chiffre 13 est en décimal, il faut écrire D en hexadécimal !
Continuons avec 176. Le reste de la division entière de 176 par 16 est 0 et le quotient 11.

```
>>> 176 % 16
0
>>> 176 // 16
11
```



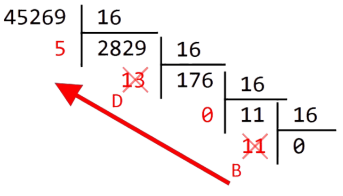
On obtient le chiffre hexadécimal 0. Continuons avec 11. Le reste de la division entière de 11 par 16 est 11 et le quotient 0.

```
>>> 11 % 16
11
>>> 11 // 16
0
```



On a obtenu le dernier chiffre hexadécimal, c'est B (11 en décimal). Le quotient est 0, inutile de continuer les divisions, tous les chiffres hexadécimaux ont été trouvés.

On a obtenu les chiffres hexadécimaux de l'écriture binaire de 45269_{10}, mais ⚠️ de gauche à droite, il faut donc les lire de droite à gauche pour trouver $B0D5_{16}$.



Exercice corrigé

Écrire les nombres suivants en hexadécimal :

- 142
- 3121309

Réponse

On peut traduire cet algorithme en Python, par exemple pour écrire une fonction `dec_to_hex` qui renvoie l'écriture hexadécimale d'un nombre entier `n` :

```
def dec_to_hex(n):
    """ int -> str
    Renvoie l'écriture hexadécimale de l'entier n
    """
    chiffre_hex = {0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9:
'9',
        10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F'}

    if n == 0:
        return '0'
    hex = ''
    while n > 0:
        hex = chiffre_hex[n % 16] + hex
        n = n // 16
    return hex


>>> dec_to_hex(45269)
'B0D5'
```

En Python, la fonction `hex` permet d'écrire un nombre décimal en binaire :

```
>>> hex(45269)
'0xb0d5'
```

Écrire un nombre binaire en hexadécimal et réciproquement

Chaque chiffre hexadécimal correspond à 4 bits en binaire, car $2^4 = 16$, ce qui rend l'hexadécimal particulièrement utile en informatique offrant une écriture plus compacte et des conversions très simples. Avec un peu d'habitude, la conversion de binaire en hexadécimal se fait donc tout simplement en lisant les bits quatre par quatre, et inversement on passe de l'hexadécimal au binaire en remplaçant chaque chiffre hexadécimal par les 4 bits équivalents.

Par exemple un octet, sur 8 bits, s'écrit simplement avec 2 chiffres hexadécimaux : On peut écrire directement $E8_{16}$ en binaire, c'est $1110\ 1000_2$, 1110 pour E et 1000 pour 8; et inversement, $110\ 1101_2$ peut s'écrire $6D$ en hexadécimal, 6 pour 0110 et D pour 1101.

Opérations

Les quatre opérations de base (addition, soustraction, multiplication et division) restent exactement les mêmes qu'en écritures décimale et binaire, mais cette fois ci sont un peu plus compliquées avec les 16 chiffres hexadécimaux.

- Addition

  On additionne les chiffres hexadécimaux en prenant soin d'aligner les chiffres de même poids à droite et on calcule de la droite vers la gauche en passant la retenue si nécessaire.

  Exemple : Calculer $CDE8B3 + FC1A25$

  On aligne d'abord à gauche les chiffres hexadécimaux des deux nombres l'un au dessus de l'autre. Puis on commence par ajouter les deux chiffres les plus à droite, $3 + 5 = 8$, puis on continue vers la gauche, $B + 2 = D$ (car $11 + 2 = 13$ en décimal).

  ```
    CDE8B3
  + FC1A25
  --------
        D8
  ```

  On arrive à $8 + A$. On peut calculer l'équivalent en décimal, $8 + 10 = 18$, qui s'écrit 12 en hexadécimal. On peut faire la conversion avec `18 // 16 = 1` et `18 % 16 = 2` ou tout simplement en faisant `18-16` pour trouver le chiffre des unités. On écrit le 2 et on ajoute une retenue de 1 à gauche.

  ```
     1
    CDE8B3
  + FC1A25
  --------
       2D8
  ```

  On ajoute les chiffres hexadécimaux suivants, avec cette retenue doit calculer $1 + E + 1$, c'est l'équivalent en décimal de $1 + 14 + 1 = 16$, qui s'écrit 10 en hexadécimal. On garde le 0 et on ajoute une nouvelle retenue de 1 encore à gauche.

  ```
    1 1
    CDE8B3
  + FC1A25
  --------
      02D8
  ```

  On arrive à $1 + D + C$, équivalent en décimal de $1 + 13 + 12 = 26$, qui s'écrit $1A$ en hexadécimal. On écrit le A et on ajoute une nouvelle retenue de 1 encore à gauche

  ```
    1 1 1
    CDE8B3
  + FC1A25
  --------
     A02D8
  ```

  Finalement, l'addition des derniers chiffres $1 + C + F = 1C$, ou 28 en décimal, permet d'obtenir le résultat final, $CDE8B3 + FC1A25 = 1CA02D8$, c'est l'équivalent hexadécimal de $13\ 494\ 451 + 16\ 521\ 765 = 30\ 016\ 216$ en décimal.

  ```
    1 1 1
    CDE8B3
  + FC1A25
  --------
    1CA02D8
  ```

  Exercice corrigé

  Calculer les additions suivantes en binaire :

  - $4A3 + E1C$
  - $7D4F + E6B8$
  - $9A4F3 + 7B8E7$

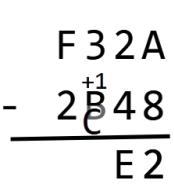
  Réponse

- Soustraction

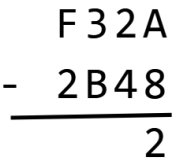
  Comme pour l'addition, on soustrait les chiffres hexadécimaux en prenant soin d'aligner ceux de même poids à droite et on calcule de la droite vers la gauche en passant la retenue si nécessaire.

  Exemple : Calculer $F32A - 2B48$

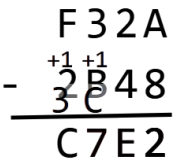
  On aligne d'abord à droite les chiffres hexadécimaux des deux nombres le plus grand au dessus de l'autre. Puis on commence par soustraire les deux chiffres les plus à droite, $A - 8 = 2$ (car $10 - 8 = 2$ en décimal), et on remonte vers la gauche.



On arrive à $2 - 4$. 2 est plus grand que 4, il faut calculer $12 - 4$ et ajouter une retenue de 1 au chiffre du bas à droite. ⚠️ Attention au bug classique, en hexadécimal $12 - 4 = E$ (car $12_{16} = 18_{10}$ et $18 - 4 = 14$ en décimal) et pas 6 ! On garde le E et on ajoute la retenue de 1 au chiffre du bas à droite : $B + 1 = C$.



On continue la soustraction en remplaçant B par C. Calculons $13 - C0$. Une fois de plus, il faut calculer $13 - C$ et ajouter une retenue de 1 au chiffre du bas à droite. ⚠️ Attention encore, en hexadécimal $13 - C = 7$ (car $13_{16} = 19_{10}$, $C_{16} = 12_{10}$ et $19 - 12 = 7$ en décimal) !



Il ne reste plus qu'à soustraire les deux derniers chiffres en remplaçant 2 par 3 : $F - 3 = C$ (car $15 - 3 = 12$ en décimal) pour obtenir le résultat final, $F32A - 2B48 = C7E2$, c'est l'équivalent hexadécimal de $62\,250 - 11\,080 = 51\,170$ en décimal.

Exercice corrigé

Calculer les additions suivantes en binaire :

- $8AD - 272$
- $ED4F - A1B2$
- $9A4F3 - 7B8E7$

Réponse

Autres bases

On peut généraliser l'écriture de nombre en base 2, 10 et 16 vue précédemment à une base dans un nombre entier b ($b \geq 2$) qui utilise des puissances successives de b.

En base b, on utilise b symboles, appelés chiffres, de valeurs allant de 0 à $b - 1$:

- Pour les bases usuelles jusqu'à dix inclus, on utilise généralement les chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.
- Pour les bases entre 11 et 36, on utilise ces mêmes 10 chiffres et les chiffres suivants sont les lettres de l'alphabet en capitales dans l'ordre de A à Z. Par exemple, pour la base 16, les chiffres utilisés sont : 0, 1, 2, 3... 8, 9, A, B, C, D, E, F. Pour la base 36, on utilise les chiffres : 0,1, 2, 3...8, 9, A, B, C...X,Y, Z.
- Pour les bases entre 37 et 62, on peut utiliser les 10 chiffres de 0 à 9, les lettres capitales, puis les lettres minuscules. Par exemple, pour la base 62, on peut utiliser les chiffres : 0,1, 2, 3...8, 9, A, B, C...X,Y, Z, a, b, c,...x, y, z.
- Et pour n'importe quelle base, on peut utiliser une notation sans lettres en séparant les chiffres par des points virgules. Par exemple, en base soixante, on peut utiliser les chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,..., 59. Dans ce cas on écrira par exemple 5112_{10} s'écrit en base 60 $(1;25;12)_{60}$ (= $1 \times 60^2 + 25 \times 60^1 + 12 \times 60^0 = 5112$).

Cours

Un nombre n qui s'écrit en base b ($b \geq 2$) avec p chiffres $a_{p-1}a_{p-2}\dots a_2a_1a_0$ (chaque a_i est un chiffre de la base tel que : $0 \quad a_i \quad b-1$) a une valeur décimale égale à :

$n = a_{p-1} \times b^{p-1} + a_{p-2} \times b^{p-2} + \dots + a_2 \times b^2 + a_1 \times b^1 + a_0 \times b^0$

ou encore avec la formule mathématique d'une somme de 0 à $p-1$: $n = \sum_{i=0}^{p-1} a_i \times b^i$

On peut écrire les b premiers nombres entre 0 et $b-1$ avec 1 seul chiffre, b^2 nombres allant de 0 à b^2-1 avec 2 chiffres, ... b^p nombres allant de 0 à b^p-1 avec p chiffres.

Écrire un nombre en base b en décimal et réciproquement

De la même La formule précédente permet d'écrire facilement un nombre hexadécimal en décimal. Il suffit de multiplier chaque chiffre hexadécimal par la puissance de 16 correspondante et de faire la somme des valeurs obtenues.

Comme pour le binaire et l'hexadécimal, on peut écrire un nombre en base b en décimal en calculant la formule avec les puissances de b et retrouver l'écriture en base b d'un nombre décimal par une suite de divisions entières par b.

Par exemple, on peut facilement calculer la valeur décimale de 6103 en base 7 :
$6103_7 = 6 \times 7^3 + 1 \times 7^2 + 3 \times 7^0 = 2058 + 49 + 3 = 2110_{10}$

Réciproquement, on peut écrire 64_{10} en base 3 par une succession de divisions par 3 :

```
>>> 64 % 3
1
>>> 64 // 3
21
>>> 21 % 3
0
>>> 21 // 3
7
>>> 7 % 3
1
>>> 7 // 3
2
>>> 2 % 3
2
>>> 2 // 3
0
```

En utilisant les restes de divisions successives par 3, en remontant à partir du dernier calculé jusqu'au premier, on obtient $64_{10} = 2101_3$

En Python, la fonction `int()` permet de convertir une chaine de caractère en précisant la base avec le paramètre par mot clé `base`.

Exemple 45 en base 6 est égal à 29 en base 10 ($4 \times 6 + 5$) :

```
>>> int('6103', base=7)
2110
```

1. Dans l'écriture $d_{p-1}d_{p-2}\dots d_2d_1d_0$, le chiffre d_{p-1} est dit de « poids fort » et d_0 de « poids faible ». On a l'habitude d'écrire les nombres en partant du poids le plus fort à gauche jusqu'au poids le plus faible à droite, cette représentation est appelée « gros boutisme », ou « *big endian* », mais certains systèmes d'exploitation utilisent la convention inverse, appelée « petit boutisme », ou « *little endian* ». ↩
2. Ce type d'algorithme appartient à la famille des « algorithmes gloutons ». ↩
3. Il existe une autre méthode, appelée méthode anglo-saxonne « par emprunt » ou « par cassage », qui consiste à soustraire la retenue du nombre du haut. ↩