1+1=3

La révolution des fourmis, 1996, Bernard Werber

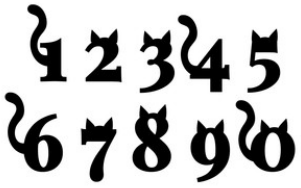
Eins, Zwei, Drei, Vier, Fünf, Sechs, Sieben, Acht

Numbers, 1981, Kraftwerk

4

Représentation des entiers

Extrait du programme



Thème : Types et valeurs de base

Contenus : Écriture d'un entier positif dans une base $b > 2$

Capacités attendus : Passer de la représentation d'une base dans une autre

Commentaires : Les bases 2, 10 et 16 sont privilégiées.

Thème : Types et valeurs de base

Contenus : Représentation binaire d'un entier relatif

Capacités attendus : Évaluer le nombre de bits nécessaires à l'écriture en base 2 d'un entier, de la somme ou du produit de deux nombres entiers.
Utiliser le complément à 2.

Commentaires : Il s'agit de décrire les tailles courantes des entiers (8, 16, 32 ou 64 bits).
Il est possible d'évoquer la représentation des entiers de taille arbitraire de Python..

Vous pouvez retrouver le cours complet à l'adresse suivante :
`https://github.com/NaturelEtChaud/NSI-Premiere/tree/main/04%20Repr%C3%A9sentation%20des%20entiers`

« Il n'y a que 10 sortes de personnes : celles qui savent compter en binaire et les autres ! »

Introduction

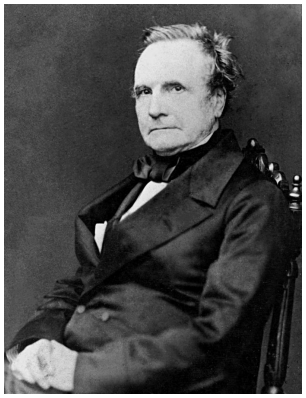
Dans un ordinateur, toutes les informations (nombres, caractères, textes, images, sons, vidéos, ...) sont codées uniquement avec des 0 et 1. C'est ce qu'on appelle un système binaire.
Comme nous avons 10 doigts, nous avons appris à compter dans ce qu'on appelle une base 10, c'est-à-dire avec uniquement dix symboles (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) nous savons écrire tous les nombres. Mais cette méthode n'est pas unique. Ainsi les bretons comptaient en base 5 (5 doigts), les mayas en base 20 (10 doigts + 10 orteils) et les babyloniens en base 60 (mais ça, personne n'a jamais vraiment compris pourquoi ....)

Il est à noter que le chansonnier Bobby Lapointe, féru de mathématiques et d'informatique avait proposé et breveté en 1968 un nouveau système, le système **bibi-binaire**[1], un dérivée du système hexadécimal qui lui permettait de faire de jolis jeux de mots laids.

Dans ce chapitre, nous allons donc commencer par nous intéresser au changement de base en nous concentrant sur la base 2 (mais pas que).
Nous allons ensuite voir comment on peut coder les nombres relatifs. Dans un autre chapitre, nous ferons un rapide survol des nombres décimaux.

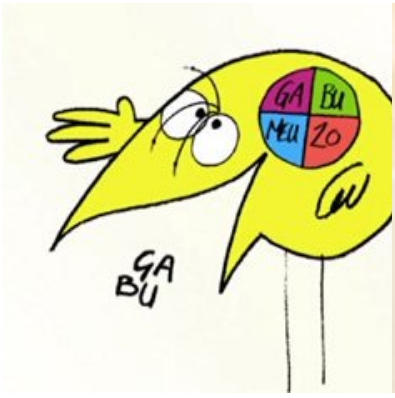
Les exposés en lien avec ce cours

Augusta Ada Byron King, comtesse de Lovelace (10 décembre 1815 - 27 novembre 1852)



Charles Babbage (26 décembre 1791 - 18 octobre 1871)

Sources Wikipédia pour Ada Lovelace et Charles Babbage



Les Shadoks, Ga Bu Zo Meu, 1968-1973

Petit boutisme, Gros boutisme

1. `https://pi.ac3j.fr/boby-lapointe/`
ou encore
`https://fr.wikipedia.org/wiki/Système_bibi-binaire`

I. Codage des entiers naturels

La première compétence à acquérir est celle de savoir compter dans une base quelconque. Dans toute cette partie la lettre b désignera une base.
Lorsque $b = 2$ nous sommes en **base 2** ou **binaire**. Nous devons représenter tous les nombres avec uniquement deux symboles : 0 et 1.
Lorsque $b = 10$, nous sommes en **base 10** ou **base décimale**. Nous devons représenter tous les nombres avec uniquement dix symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Lorsque $b = 16$, nous sommes en **base 16** ou **base hexadécimale**. Nous devons représenter tous les nombres avec uniquement seize symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Voici ce que nous pouvons obtenir :

$b = 2$	10	16
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

$b = 2$	10	16
1 0000	16	10
1 0001	17	11
1 0010	18	12
1 0011	19	13
1 0100	20	14
1 0101	21	15
1 0110	22	16
1 0111	23	17
1 1000	24	18
1 1001	25	19
1 1010	26	1A
1 1011	27	1B
1 1100	28	1C
1 1101	29	1D
1 1110	30	1E
1 1111	31	1F

$b = 2$	10	16
10 0000	32	20
10 0001	33	21
10 0010	34	22
10 0011	35	23
10 0100	36	24
...	...	...
1111 0110	246	F6
1111 0111	247	F7
1111 1000	248	F8
1111 1001	249	F9
1111 1010	250	FA
1111 1011	251	FB
1111 1100	252	FC
1111 1101	253	FD
1111 1110	254	FE
1111 1111	255	FF

Exercice 1 (base 2)
Intéressons nous à la base 2.

1. Comment reconnaît-on un nombre pair ? un nombre impair ? .....
   .....
2. (a) Que pouvez-vous dire des nombres en base décimale 2, 4, 16 ou 32 ? .....
   .....
   (b) Que remarquez-vous de leur écriture en binaire ? .....
   .....
   (c) Quel est l'écriture binaire des nombres décimaux 128, 256 et 1024 ? .....
   .....
   (d) Quel est l'écriture binaire des nombres décimaux 127, 129, 255, 257, 1023 et 1024 ? .....
   .....

Exercice 2 (base 3)
Le système ternaire ou trinaire a été utilisé sur certains ordinateurs. Mon wikipédia me dit :

« En 1958 en union soviétique, l'équipe de Nikolay Brusentsov et Sergei Sobolev à l'université d'État de Moscou a développé un Ordinateur ternaire, le Setun, reposant sur l'utilisation de tôles feuilletées miniatures et de diodes utilisables pour créer un système basé sur une logique à trois états. À l'usage, ces éléments se sont révélés plus rapides, plus fiables, plus durables et moins gourmands en énergie que leurs concurrents binaires (du moins avant que l'URSS n'ait accès aux transistors). Le développement du Setun a pris sept ans, et l'ordinateur a été opérationnel sitôt assemblé. Une cinquantaine de machines ont été produites, mais le programme, considéré comme un caprice d'universitaires dans un pays qui a mis du temps à comprendre l'importance de l'informatique, a rapidement été abandonné, au profit d'ordinateurs binaires plus banals. »
« En 1970, Nikolay Brusentsov conçut une version améliorée de Setun, nommée Setun-70. »
Pour plus d'informations sur le sujet (la photo est issue de ce lien) :
`https://earltcampbell.com/2014/12/29/the-setun-computer/`

Écrire tous les nombres décimaux de 0 à 27 en base 3.

.....
.....
.....
.....

II. Comment convertir un nombre d'une base dans une autre ? ... ou le contraire

Remarque
Pour éviter toutes confusions, nous rajouterons à la fin du nombre, en indice, la base. Ainsi :

- 101_{10} est bien le nombre cent un.
- 101_2 est le nombre binaire 101 correspondant à cinq en décimal.
- 101_3 est le nombre trinaire 101 correspondant à dix en décimal.
- 101_{16} est le nombre en hexadécimal 101 correspondant à ... on verra cela plus tard.

a) D'une base quelconque vers la base décimale

Quelque soit la base choisie, les chiffres ont chacun une signification selon leur position.

Par exemple, pour le nombre 125_{10} en base 10 :

$$125_{10} = 1 \text{ centaine} + 2 \text{ dizaines} + 5 \text{ unités}$$

Mais chaque catégorie (centaine, dizaine, unité) peut s'exprimer comme une puissance de 10 :
centaine= 10^2, dizaine= 10^1, unité= 10^0.
Ainsi, on peut décomposer le nombre 125_{10} :

$$125_{10} = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

Exercice 3 (base 10)
Écrire la décomposition en base 10 des nombres décimaux suivants :

- $256_{10} =$ .....
- $1\ 024_{10} =$ .....
- $7\ 050\ 260_{10} =$ .....

De la même manière,

en base b, chaque position correspond à une puissance de b. On peut ainsi facilement convertir un nombre d'une base b vers la base décimale.Ainsi :

$$vwxyz_b = v \times b^4 + w \times b^3 + x \times b^2 + y \times b^1 + z \times b^0$$

Par exemple, pour le nombre 1101_2 :

$$\begin{aligned} 1101_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 8 + 1 \times 4 + 0 + 1 \times 1 \\ &= 13_{10} \end{aligned}$$

Exercice 4 (base 2)
Convertir les nombres suivants écrits en base 2, dans la base décimale :

- $1010_2 =$ .....
- $1\ 1110_2 =$ .....
- $1000\ 0010_2 =$ .....
- $1111\ 1110_2 =$ .....

Exercice 5 (base 3 - pour le fun i.e. non exigible)
Convertir les nombres suivants écrits en base 3, dans la base décimale :

- $101_3 =$ .....
- $210_3 =$ .....
- $1201_3 =$ .....

Exercice 6 (base 16)
Convertir les nombres suivants écrits en base 16, dans la base décimale :

- $101_{16} =$ .....
- $AF_{16} =$ .....
- $1B_{16} =$ .....
- $E3_{16} =$ .....

b) De la base décimale vers une base quelconque

Convertir un nombre exprimé dans une base décimale vers une base b est par contre beaucoup plus ardue. Mais nous allons y arriver.
Il faut effectuer des divisions euclidiennes successives des quotients précédemment obtenus par b jusqu'à ce que le quotient soit nul. L'écriture est alors obtenue en reprenant tous les restes obtenus au cours des divisions successives ...
- As-tu bien compris ? ? ?
- NON ! ! ! ? ? ? ? Au secours ! ! ! ! !
- Ben alors, petit scarabée, tu n'as pas écouté ? Bon, c'est normal. Voyons plutôt avec un exemple simple. Nous allons écrire 59_{10} *en base 2.*

	$59 \div 2 = 29$ reste 1	son écriture en binaire est $...1_2$
on reprend le quotient précédent 29	$29 \div 2 = 14$ reste 1	son écriture en binaire est $...11_2$
on reprend le quotient précédent 14	$14 \div 2 = 7$ reste 0	son écriture en binaire est $...011_2$
on reprend le quotient précédent 7	$7 \div 2 = 3$ reste 1	son écriture en binaire est $...1011_2$
on reprend le quotient précédent 3	$3 \div 2 = 1$ reste 1	son écriture en binaire est $...1\ 1011_2$
on reprend le quotient précédent 1	$1 \div 2 = \boxed{0}$ reste 1	son écriture en binaire est $...11\ 1011_2$

Comme le dernier quotient est nul, on arrête les divisions. Donc : $59 = 11\ 1011_2$.

A vous maintenant d'essayer :

Exercice 7 (base 2)

Convertir en base binaire, les entiers suivants :

$13_{10} =$ .....

$27_{10} =$ .....

$36_{10} =$ .....

$114_{10} =$ .....

$240_{10} =$ .....

Une fois que l'on a bien compris le principe, on peut le reproduire pour n'importe quelle base. Ainsi, si nous voulons convertir 59 en base 3, nous devrons effectuer des divisions successives par 3 :

	$59 \div 3 = 19$ reste 2	son écriture en trinaire est $...2_3$
on reprend le quotient précédent 19	$19 \div 3 = 6$ reste 1	son écriture en trinaire est $...12_3$
on reprend le quotient précédent 6	$6 \div 3 = 2$ reste 0	son écriture en trinaire est $...012_3$
on reprend le quotient précédent 2	$2 \div 3 = \boxed{0}$ reste 2	son écriture en trinaire est $...2012_3$

Comme le dernier quotient est nul, on arrête les divisions. Donc : $59 = 2012_3$.

Exercice 8 (base 3)

Convertir en base trinaire, les entiers suivants :

$13_{10} =$ .....

$27_{10} =$ .....

$36_{10} =$ .....

$114_{10} =$ .....

$240_{10} =$ .....

Exercice 9 (base 16)

Convertir en base hexadécimale, les entiers suivants :

$13_{10} =$ .....

$27_{10} =$ .....

$36_{10} =$ .....

$114_{10} =$ .....

$240_{10} =$ .....

c) Pour convertir de la base 2 vers la base 16

Il suffit de faire des paquets de 4. Regardons dans l'exemple suivant :

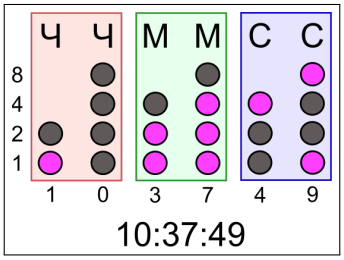
$$\begin{aligned}1011\ 0110_2 &= 1\times 2^7+0\times 2^6+1\times 2^5+1\times 2^4+0\times 2^3+1\times 2^2+1\times 2^1+0\times 2^0\\ &= (1\times 2^3+0\times 2^2+1\times 2^1+1\times 2^0)\times 2^4+(0\times 2^3+1\times 2^2+1\times 2^1+0\times 2^0)\\ &= 1011_2\times 16+0110_2\\ &= B_{16}\times 16+6_{16}\\ &= B6_{16}\end{aligned}$$

Exercice 10 (De la base 2 vers la base 16)
Reprendre les résultats de l'exercice 7 en base 2 pour les convertir directement en base 16.
Comparer avec les résultats obtenus à l'exercice 9.

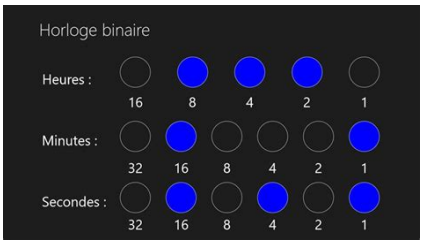
$13_{10} =$ .....
$27_{10} =$ .....
$36_{10} =$ .....
$114_{10} =$ .....
$240_{10} =$ .....

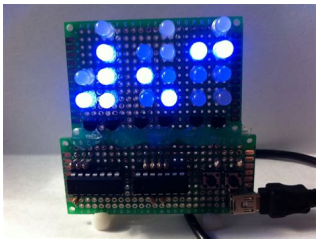
III. Quelques applications

Exercice 11 (L'horloge binaire)
Il existe des horloges à affichage binaire selon le modèle suivant :



Quelle est l'heure affichée sur les horloges suivantes ?





....

Exercice 12 (Adresse MAC)
Chaque carte réseau a une adresse MAC [2], un identifiant unique composé de 6 nombres écrit en base hexadécimale.

1. Convertir tous les nombres de la carte ci-dessus en binaire.
   .....
   .....
   .....
2. Convertir tous les nombres de la carte ci-dessus en base 10.
   .....
   .....
   .....
3. Combien d'adresses MAC différentes peut-on obtenir avec ce système ?
   .....
   .....
4. Les trois premiers octets d'une adresse MAC permettent d'identifier le fabricant de la carte. C'est ce qu'on appelle l'OUI (*Organizationally Unique Identifier*).
   Combien d'adresses MAC différentes disposent chaque fabricant ?
   .....
   .....
   La liste des OUI est donnée à l'adresse suivante : `http://standards-oui.ieee.org/oui.txt`.
   Quel est le fabricant de la carte réseau sur la photo ?
   .....

2. MAC pour *Media Access Control*. Nous en reparlerons dans le chapitre 23.

IV. Représentation en machine

Nous l'avons vu, Charles Babbage a utilisé des cartes perforées pour transmettre des données et programmer sa *machine analytique*.

Source wikipédia

Cet outil est très utile car chaque case ne peut avoir que deux états :
- perforée ;
- non perforée.

Ce qui peut correspondre à :
- faire ;
- ne pas faire.

ou encore :
- vrai ;
- faux.

...

Ce principe a été gardé et IBM a breveté un modèle de carte perforée dès 1928. Ces cartes comportaient 80 colonnes et 12 lignes. Elles étaient stockées par boîte de 2 000. Les cartes étaient perforées « *à la main* » par des opératrices.

Source wikipédia

Archivage de cartons de cartes perforées archivés au service du NARA en 1959.
Chaque carton peut contenir 2 000 cartes.
Source wikipédia

Les ordinateurs ont été équipés d'unités périphériques capables de lire et de perforer ces cartes jusqu'au début des années 1980. Pour plus d'explications, vous pouvez voir la vidéo sur le lien suivant :
`https://www.youtube.com/watch?v=oaVwzYN6BP4`

a) Le bit

L'unité élémentaire utilisée en informatique pour coder l'information est appelée **un `bit`**, (contraction de *binary digit*, chiffre binaire).
Un bit peut prendre deux valeurs : 0 ou 1.
On peut alors coder tout chiffre, tout nombre, toute lettre, tout mot, tout texte, toute image, tout son, toute vidéo en une suite de 0 et de 1. Néanmoins il est nécessaire d'inventer des **encodages** pour représenter ces informations.

b) L'octet

Dans la mémoire d'un ordinateur (RAM, ROM, registres des microprocesseurs, etc) ces chiffres binaires sont regroupés en octet, c'est-à-dire par paquets de 8 bits puis organisés en **mots machine** de 2, 4 ou 8 octets. Par exemple une machine dite de 32 bits est un ordinateur qui manipule directement des mots de 4 octets ($4 \times 8 = 32$ bits) lorsqu'il effectue des opérations. Un octet étant codé sur 8 bits, cela offre $2^8 = 256$ possibilités. Avec un octet on peut donc coder tous les nombres entiers entre 0 et 225.

c) Les unités officielles en informatique

Il est très courant en informatique de mesurer la capacité d'un disque dur, de la RAM d'un ordinateur ou d'un débit de données Internet avec une unité de mesure exprimée comme un multiple d'octets. Ces multiples sont traditionnellement des puissances de 10 et on utilise les préfixes « kilo », « mega », etc. pour les nommer.

Nom	Symbole	Valeur
kilooctet	ko	10^3
mégaoctet	Mo	10^6
gigaoctet	Go	10^9
téraoctet	To	10^{12}
pétaoctet	Po	10^{15}
...		

Historiquement, les multiples utilisés en informatique étaient des puissances de 2. Pour ne pas confondre l'ancienne et la nouvelle notation, on utilise (ou on devrait utiliser !) des symboles différents pour représenter ces multiples.

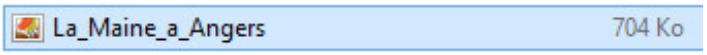
Nom	Symbole	Valeur
kibioctet	Kio	$2^{10} = 1\ 024$ octets
mébioctet	Mio	$2^{20} = 1\ 048\ 576$ octets
gibioctet	Gio	$2^{30} = 1\ 073\ 741\ 824$ octets
tébioctet	Tio	$2^{40} = 1\ 099\ 511\ 627\ 776$ octets
...		

Exercice 13 (Punched card)

1. Sachant qu'une carte perforée *80 colonnes* est composée de 80 colonnes et de 12 lignes, calculer sa capacité mémoire en octet.
   .....
   Peut-on trouver une unité plus adéquate ?
   .....
2. Calculer la capacité mémoire d'une boite de 2 000 cartes perforées en choisissant l'unité la plus adéquate.
   .....
3. Sur ma Raspberry Pi, j'ai une carte microSD de 16 Go.
   Combien me faut-il de boîtes de 2 000 cartes pour avoir la même capacité de stockage ?
   .....

Exercice 14 (Une photo non compressée)

Source : Wikipédia



J'ai une photo *La_Maine_a_Angers* dans un format *BMP* non compressé qui fait 800 pixels de long pour 300 pixels de hauteur, chaque pixel étant codé sur 24 bits, autrement dit chaque pixel étant obtenu comme un mélange de rouge codé sur 1 octet, de vert codé sur 1 octet et de bleu codé sur 1 octet.
Pour calculer la taille mémoire prise par la photo, je fais le calcul suivante :
$800 \times 300 = 240\ 000$ pixels
Chaque pixel faisant 3 octets, on a :
$240\ 000 \times 3 = 720\ 000$ octets $= 720$ ko
Vérification faite, le fichier pèse 704 ko.
Comment expliquer cette différence ?

.....
.....
.....

Exercice 15 (Révisions)

1. Donner la représentation en base 2, sur 8 bits des entiers :
   - $14_{10} =$.....
   - $218_{10} =$.....
   - $42_{10} =$.....
   - $57_{10} =$.....

2. Donner la représentation en base 16 des entiers binaires suivants :
   - $0100\ 1010_2 =$.....
   - $0100\ 0010\ 0001_2 =$.....
   - $1010\ 0100\ 1111\ 0010_2 =$.....

3. Quelle est la valeur en base 10 de l'entier $BEEF_{16}$?
   .....
   .....
   .....

V. Les nombres relatifs

a) Additions en binaire

Pour commencer nous allons voir comment on peut effectuer l'addition de deux nombres écrits en binaire.

On a pour chaque position : $0+0=0$, $1+0=1$, $0+1=1$ et $1+1=10$ autrement dit un 0 avec 1 retenue. Ainsi, pour additionner $13=1101_2$ avec $9=1001_2$, on pose l'opération comme en primaire, en rajoutant des retenues lorsque l'on fait $1+1$:

	1			1		retenues
		1	1	0	1	treize
+		1	0	0	1	neuf
	1	0	1	1	0	vingt-deux

Exercice 16 (Addition binaire)
Effectuer les deux additions binaires suivantes :

					retenues
	1	1	0	0	.....
+	1	1	0	1	.....
					.....

					retenues
	1	1	1	1	.....
+	0	0	1	1	.....
					.....

b) Signer un nombre ?

Pour simplifier l'idée nous n'utiliserons ici que des mots de 4 bits.
La première idée qui vient à l'esprit est d'utiliser un des bits pour donner le signe du nombre :

- 0 est le nombre est positif ;
- 1 est le nombre est négatif.

Par exemple $\boxed{0}111_2$ est le nombre 7 et $\boxed{1}111_2$ est le nombre -7.
Nous obtenons ainsi 8 nombres positifs :
$0000_2=0$ $0001_2=1$ $0010_2=2$ $0011_2=3$ $0100_2=4$ $0101_2=5$ $0110_2=6$ $0111_2=7$
et 8 nombres négatifs :
$1000_2=-0$ $1001_2=-1$ $1010_2=-2$ $1011_2=-3$ $1100_2=-4$ $1101_2=-5$ $1110_2=-6$ $1111_2=-7$
Malheureusement cette idée simple comporte deux problèmes :

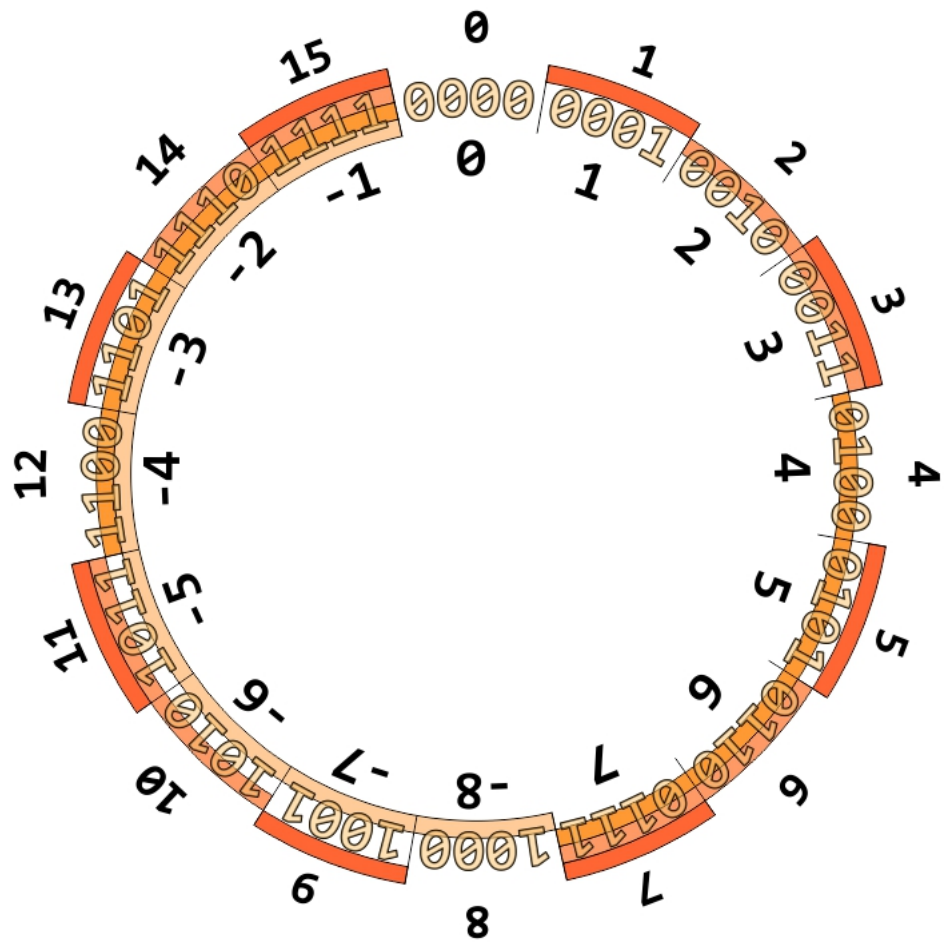
- il y a deux façons différentes de coder 0 : 0000_2 et 1000_2... pourquoi pas...
- si on ajoute 1 à un nombre négatif, on n'augmente pas de 1... c'est beaucoup plus problématique !

		1	1		retenues
	1	0	1	1	-trois
+	0	0	0	1	un
	1	1	0	0	-quatre

- Mais Monsieur, depuis quand $-3+1=-4$?
- Je suis d'accord avec toi, petit scarabée ; ça fait désordre. On a donc choisi une façon différente pour coder les nombres relatifs.

c) Le complément à 2

Il faut avoir une vision circulaire des nombres pour ne plus tomber sur l'écueil du $-3+1=-4$.



Les nombres sont alors codés :

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000_2	1001_2	1010_2	1011_2	1100_2	1101_2	1110_2	1111_2	0000_2	0001_2	0010_2	0011_2	0100_2	0101_2	0110_2	0111_2

On constate que les problèmes que nous avions relevé sont résolus :

- il n'y a plus d'une seule façon de coder $0=0000_2$;
- lorsque l'on prend un entier négatif (autre que -1) et que l'on rajoute 1, le résultat devient correct :

			1		retenues
	1	1	0	1	-trois
+	0	0	0	1	un
	1	1	1	0	-deux

- lorsque l'on effectue l'opération $-1+1$, le résultat est lui aussi correct à condition de bien comprendre que nous effectuons (ici) un calcul sur 4 bits.

1	1	1	1		retenues
	1	1	1	1	-un
+	0	0	0	1	un
1	0	0	0	0	????

En effet, comme nous codons avec seulement 4 bits, le chiffre 1 apparaît sur un cinquième bit. Il est donc perdu. Ainsi $1111_2+0001_2=(1)0000_2=0000_2$.
Nous voilà rassuré.

Problème ! Si nous avons bien compris ce qui vient d'être dit, il nous reste un dernier problème $7+1=-8$!!! Voyons plutôt :

	1	1	1		retenues
	0	1	1	1	sept
+	0	0	0	1	un
	1	0	0	0	-huit

Nous ne pourrons malheureusement pas résoudre ce problème et nous aurons toujours cet effet de bord.

Il faut donc bien avoir conscience qu'il est possible d'additionner deux nombres positifs et d'obtenir un nombre négatifs...
De même, il est possible d'additionner deux nombres négatifs et d'obtenir un nombre positifs...

Conséquences :

- avec 4 bits, on peut coder tous les nombres relatifs entre $-2^{4-1}=-8$ et $2^{4-1}-1=7$;
- avec 1 octet = 8 bits, on peut coder tous les nombres relatifs entre $-2^{8-1}=-128$ et $2^{8-1}-1=127$;
- avec 2 octet = 16 bits, on peut coder tous les nombres relatifs entre $-2^{16-1}=-32\,768$ et $2^{16-1}-1=32\,767$;
- ...
- avec n bits, on peut coder tous les nombres relatifs entre -2^{n-1} et $2^{n-1}-1$.

Avec n bits, on peut coder tous les nombres relatifs entre -2^{n-1} et $2^{n-1}-1$

Exemple (En Java)
Dans le langage Java, le type de variable `int` est un type d'entier relatif stocké sur 4 octets soit 32 bits. On peut donc coder tous les entiers relatifs entre $-2^{32-1}=-2^{31}=-2\,147\,483\,648$ et $2^{32-1}-1=2^{31}-1=2\,147\,483\,647$ ce qui semble suffisant dans un premier temps.
On obtient malgré tout des résultats assez surprenant si on en n'est pas averti :

```
int a = 2147483647;
int b = a+1;
print(b);
```

Résultat : `-2147483648`

- Mais Monsieur, si je peux me permettre, c'est nul votre truc... C'est même pas capable d'additionner 1. Même ma petite sœur c'est le faire... C'est pour dire.

Exemple (Et en Python ? ? ?)
Il est difficile de mettre en évidence ces dépassements avec Python qui définit des entiers de taille arbitraire.

```
a = 2147483647
b = a+1
print(b)
```

Résultat : `2147483648`
Que c'est-il passé ? `a` est codé sur 24 bits car c'est largement suffisant. Ensuite pour éviter tout problème de dépassement, `b` est ensuite codé sur 32 bits... mais un vrai informaticien ne code pas qu'en Python...

d) Comment obtenir un entier négatif avec la méthode du complément à 2 ... et réciproquement

Prenons par exemple le nombre -3 à coder sur 4 bits.

1. On passe en positif : 3
2. On code en binaire : 0011_2
3. On opère une négation de chaque bit (autrement dit un 0 devient un 1 et un 1 devient un 0 : 1100_2
4. On ajoute 1 : 1101_2

On peut alors vérifier que $-3 + 3 = 0$ car on code sur 4 bits et donc le cinquième bit qui apparaît dans notre calcul à la main, disparaît pour la machine :

	1	1	1	1		retenues
		1	1	0	1	-3
+		0	0	1	1	3
	(1)	0	0	0	0	zéro

Inversement, pour décoder un nombre binaire négatif, on effectue les mêmes opérations, dans le sens inverse. Voyons plutôt avec 1001_2 codé sur 4 bits :

1. On opère la négation de chaque bit : 0110_2
2. On ajoute 1 : 0111_2
3. On convertit dans la base 10 : 7
4. On passe au négatif : -7

Exercice 17

Donner la représentation en complément à 2 et sur 8 bits des entiers :

- $-10_{10} =$ .....
- $-128_{10} =$ .....
- $-42_{10} =$ .....
- $97_{10} =$ .....

Exercice 18

Donner en base 10 la valeur des octets signés :

- $1110\ 0111_2 =$ .....
- $1100\ 0001_2 =$ .....

VI. Quelques précédents fâcheux

a) Des bug encore et toujours

α - Le bug de l'an 2000

Source Texte et Image : Wikipédia

Dans les années 1960, la mémoire et l'entreposage des données en informatique étaient coûteux et rares. Cette pénurie de place en mémoire et dans les fichiers a incité les programmeurs à coder les années sur deux chiffres seulement. Autrement dit l'année 1999 était codé 99 et l'année 2000 codé 00.
Le 1er janvier 2000, les machines ont alors cru qu'elles étaient en 1900.
Finalement, aucun problème critique ne s'est produit. Cependant, des sommes considérables, se chiffrant en centaines de milliards de dollars dans le monde, ont dû être dépensées pour prévenir tout incident lors du passage.

β - Le bug du 6 avril 2019

Les boîtiers GPS les moins récents (comme le mien ...) comptent les semaines sur 10 bits, ils repassent alors à zéro toutes les $2^{10} = 1024$ semaines. Ce WNRO ou Week Number Rollover (remise à zéro du compteur des semaines) s'est produit le 6 avril 2019, la précédente remise à zéro s'étant produite en 1999.
`https://www.60millions-mag.com/2019/02/19/gps-le-drole-de-bug-du-6-avril-2019-12463`

γ - Le bug de l'an 2038

Le bug de l'an 2038 est un bug informatique similaire au bug de l'an 2000 qui pourrait perturber le fonctionnement d'un grand nombre de systèmes informatiques le 19 janvier 2038 à 3 h 14 min 8 s, temps universel. Ils afficheront alors la date du 13 décembre 1901. Ce bug concerne potentiellement tous les systèmes d'exploitation et les programmes qui utilisent une représentation des dates en 32 bits.
Source Wikipédia

b) L'explosion d'Ariane 5 en 1996

Source Image : Brèves de maths
http://www.breves-de-maths.fr/meme-les-ordinateurs-font-des-erreurs/

Le premier vol du nouveau lanceur Ariane 5 a eu lieu le 4 juin 1996. Après 30 secondes de vol, le lanceur, alors à une altitude de 3700 m, a soudainement basculé, quitté sa trajectoire, s'est brisé et a explosé. L'échec était dû à la perte totale des informations de guidage et d'attitude, 37 secondes après la mise à feu du moteur Vulcain.
Le Système de Référence Inertiel a calculé une accélération horizontale beaucoup plus grande pour Ariane 5 que pour Ariane 4. Cette valeur flottante sur 64 bits n'a pu être convertie en un entier sur 16 bits.
Comble d'ironie, ce calcul était inutile pour Ariane 5.
Tous les logiciels ont été soigneusement vérifiés avant le lancement d'Ariane 502.
Cette erreur a coûté plus de 500 millions de dollars.
Source Texte : Supinfo
https://www.supinfo.com/articles/single/235-erreur-informatique-plus-couteuse-histoire

VII. Exercices en Python

Exercice 19 (Conversion en binaire)
Écrire une fonction `binaire(n)` qui convertit `n` un nombre écrit en base 10 dans la base 2.

ENTRÉE : Un nombre entier strictement positif en base 10.
SORTIE : Un nombre en base 2 au format chaîne de caractère.
EXEMPLES : `binaire(15)` doit renvoyer `'1111'`.
`binaire(255)` doit renvoyer `'11111111'`.
`binaire(32)` doit renvoyer `'100000'`.

Exercice 20 (Conversion en base b)
Écrire une fonction `conversion(n,b)` qui convertit `n` un nombre écrit en base 10 dans la base `b`.

ENTRÉES : Un nombre entier strictement positif en base 10, une base sous la forme d'un entier strictement positif.
SORTIE : Un nombre en base b au format chaîne de caractère.
EXEMPLES : `conversion(15,2)` doit renvoyer `'1111'`.
`conversion(255,4)` doit renvoyer `'3333'`.
`conversion(666,10)` doit renvoyer `'666'`.
`conversion(666,16)` doit renvoyer `'2910'`.

Exercice 21 (Conversion en base hexadécimale)
Nous constatons qu'il y a un problème avec la base 16. Par exemple `conversion(666,16)` retourne `2910`. Faut-il comprendre 2-9-1-0 ou 2-9-10 ?
Modifier le programme précédent en utilisant les lettres A, B, C, D, E et F si c'est la base 16 qui a été choisie.

ENTRÉES : Un nombre entier strictement positif en base 10, une base sous la forme d'un entier strictement positif.
SORTIE : Un nombre en base b au format chaîne de caractère avec les lettres A, B, C, D, E, F si $b = 16$.
EXEMPLES : `conversion(15,2)` doit renvoyer `'1111'`.
`conversion(255,4)` doit renvoyer `'3333'`.
`conversion(666,10)` doit renvoyer `'666'`.
`conversion(666,16)` doit renvoyer `'29A'`.
`conversion(6666,16)` doit renvoyer `'1A0A'`.

Exercice 22 (Conversion en base hexadécimale - version 2)
Nous constatons que dans notre première version nous enchaînons plusieurs fois `if ... elif ... elif ... elif ...` simplement pour associer 10 à la lettre A, 11 à la lettre B, ..., 15 à la lettre F. Il s'agit juste d'un décalage de un en un à chaque fois. On pourrait essayer d'améliorer le code. Pour cela, nous allons utiliser une chaîne de caractères.
Pour mieux appréhender l'utilisation des chaînes, voici un petit programme :

```
1 def comprendre(texte):
2     print("la longueur du texte est de",len(texte),"caractère(s)")
3 
4     print("le premier caractère est",texte[0])
5     print("le dernier",texte[len(texte)-1])
6 
7     print("Et voici tous les caractères")
8     for i in range(len(texte)):
9         print(texte[i])
```

Une chaîne de caractères peut être vue comme une liste de caractères et on peut avoir accès à chacun grâce à son numéro dans la liste. Mais il faut faire attention car on compte **TOUJOURS** à partir de 0.
Ainsi `texte[0]` est le premier caractère et `texte[len(texte)-1]` est le dernier.

Avec cet exemple, nous obtenons par exemple le résultat suivant :

```
>>> comprendre("Hello World!")
la longueur du texte est de 12 caractère(s)
le premier caractère est H
le dernier !
Et voici tous les caractères
H
e
l
l
o

W
o
r
l
d
!
```

Revenons à notre conversion. Nous partons de la chaîne `'ABCDEF'`.
Le `'A'` est numéroté 0 et on doit l'utiliser lorsque le reste sera égal à 10 ;
le `'B'` est numéroté 1 et on doit l'utiliser lorsque le reste sera égal à 11 ;
le `'C'` est numéroté 2 et on doit l'utiliser lorsque le reste sera égal à 12 ;
...
le `'F'` est numéroté 5 et on doit l'utiliser lorsque le reste sera égal à 15 ;
On constate dont que l'on obtient le numéro du caractère choisi en fonction du reste par la formule :
.....
Modifier alors le programme précédent pour en simplifier l'écriture.

Exercice 23 (Conversion en base Shadok - Exercice non corrigé)
Écrire une fonction `shadok(n)` programme qui convertit `n` un nombre en base 10 vers la base Shadok GaBuZoMeu.

ENTRÉE : Un nombre entier strictement positif en base 10.
SORTIE : La conversion du nombre en base Shadok au format chaîne de caractère uniquement avec les chiffres Ga, Bu, Zo, Meu.
EXEMPLES : `shadok(4)` doit renvoyer `'BuGa'`.
`shadok(15)` doit renvoyer `'MeuMeu'`.
`shadok(255)` doit renvoyer `'MeuMeuMeuMeu'`.
`shadok(666)` doit renvoyer `'ZoZoBuZoZo'`.

Exercice 24 (Conversion en base 10)
Écrire une fonction `baseDecimale(n,b)` programme qui effectue la conversion d'un nombre `n` écrit dans une base `b`≤ 10 dans la base 10.

ENTRÉES : Un nombre entier écrit dans une base $b \leq 10$ et un nombre $b \leq 10$.
SORTIE : La conversion du nombre en base 10.
EXEMPLES : `baseDecimale(10110,2)` doit renvoyer `21`.
`baseDecimale(1210,3)` doit renvoyer `48`.
`baseDecimale(666,10)` doit renvoyer `666`.
`baseDecimale(443,5)` doit renvoyer `123`.
Question subsidiaire : `baseDecimale(666,5)` renvoie le résultat `186` mais ça n'a pas de sens. Pourquoi ?
.....
.....

Exercice 25 (Conversion de la base 16 vers la base 10)
Écrire une fonction `hexaToDecimale(n)` qui effectue la conversion d'un nombre `n` écrit en base 16 dans la base 10.

Entrée : Un nombre entier écrit dans la base 16.
Sortie : La conversion du nombre en base 10.
Exemples : `hexaToDecimale(6)` doit renvoyer `6`.
`hexaToDecimale(42)` doit renvoyer `66`.
`hexaToDecimale(29A)` doit renvoyer `666`.
`hexaToDecimale(1A0A)` doit renvoyer `6666`.

VIII. Corrections des exercices

Correction de l'exercice 19 (Conversion en binaire)

```
def conversion(n):
    '''
    Convertit n en binaire
    '''
    quotient = n
    resultat =""
    while quotient !=0:
        reste = quotient%2
        quotient = quotient//2
        resultat = str(reste) + resultat
    return resultat
```

Correction de l'exercice 20 (Conversion en base *b*)

```
def conversion(n,b):
    '''
    Convertit n en base n
    '''
    quotient = n
    resultat =""
    while quotient !=0:
        reste = quotient%b
        quotient = quotient//b
        resultat = str(reste) + resultat
    return resultat
```

Correction de l'exercice 21 (Conversion en base hexadécimale)

```
def conversion(n,b):
    '''
    Convertit n en base b avec une écriture spécifique pour la base 16
    '''
    quotient = n
    resultat =""
    while quotient !=0:
        reste = quotient%b
        quotient = quotient//b
        if reste == 10:
            reste = 'A'
        elif reste == 11:
            reste ='B'
        elif reste == 12:
            reste ='C'
        elif reste == 13:
            reste ='D'
        elif reste == 14:
            reste ='E'
        elif reste == 15:
            reste ='F'
        resultat = str(reste) + resultat
    return resultat
```

Correction de l'exercice 22 (Conversion en base hexadécimale - version 2)

```
def conversion(n,b):
    '''
    Convertit n en base b avec une écriture spécifique pour la base 16
    '''
    hexa="ABCDEF"
    quotient = n
    resultat =""
    while quotient !=0:
        reste = quotient%b
        quotient = quotient//b
        #dans le cas d'une base hexadecimale
        if b==16 and reste >= 10:
            reste = hexa[reste-10]
        resultat = str(reste) + resultat
    return resultat
```

Correction de l'exercice 23 (Conversion en base Shadok - Exercice non corrigé)

```
def shadok(n):
    '''
    Convertit n dans la base Shadok, Ga Bu Zo Meu
    '''
    gabuzomeu=['Ga','Bu','Zo','Meu']
    quotient = n
    resultat =""
    while quotient !=0:
        reste = quotient%4
        quotient = quotient//4
        reste = gabuzomeu[reste]
        resultat = reste + resultat
    return resultat
```

Correction de l'exercice 24 (Conversion en base 10)

```
def baseDecimale(n,b):
    '''
    Convertit le nombre n écrit en base b<=10 en base décimale
    '''
    nombre = str(n)
    longueur = len(nombre)
    resultat = 0
    for i in range(longueur-1,-1,-1):
        resultat = resultat + int(nombre[i])*b**(longueur-i-1)
    return resultat

#autre version plus efficace
def baseDecimale2(n,b):
    nombre = str(n)
    longueur = len(nombre)
    resultat = 0
    for i in range(longueur-1):
        resultat = b*(resultat + int(nombre[i]))
    resultat = resultat + int(nombre[longueur-1])
    return resultat
```