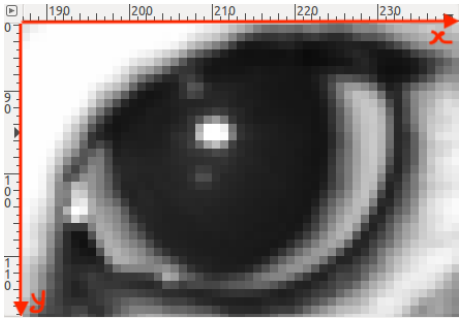
Représentation des entiers naturels

Les **machines actuelles** (ordinateurs, tablettes, smartphones..) traitent l'**information** (images, sons, textes, etc...) sous la forme de bits 0 ou 1, parce que (pour simplifier) le 0 est associé à une tension électrique en-dessous d'un certain seuil et le 1 est associé à une tension électrique au dessus de ce seuil.

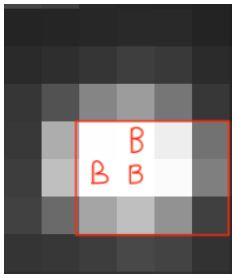
Par exemple une image en nuances de gris comme celle du chat ci-dessus est composée de pixels (picture element) visibles lorsqu'on a fait un zoom (voir ci-dessous)avec l'aide d'un éditeur d'images comme GIMP



L'intensité de gris pour chaque pixel est un entier entre 0 et 255. 0 correspond au noir et 255 au blanc.

On observe la présence de pixels "blancs" à l'intérieur de l'oeil gauche du chat autour du pixel (209,95) (on repère les pixels suivant les axes du repère en rouge)

Les 3 pixels "blancs" sont visibles sur le zoom suivant :



En utilisant la bibliothèque PIL de Python , on fait afficher les intensités des pixels de coordonnées (209,94), (210,94), (209,95) et (210,95)

On obtient :

Ces nombres sont pour les humains, les machines travaillent plutôt avec des 0 et des 1

```
254 255 238 114
255 255 251 127
165 191 146 64
```

```
11111110 11111111 11101110 1110010
11111111 11111111 11111011 1111111
10100101 10111111 10010010 1000000
```

254 et 11111110 représente la même intensité, 254 pour les humains et 11111110 pour les machines
Comment interpréter 11111110 ?

On écrit plutôt $(254)_{10} = (11111110)_2$ à la fois pour dire que c'est la même intensité mais aussi pour préciser la base avec laquelle on calcule
Dans la base 10 (décimale) $(254)_{10} = 4 \times 10^0 + 5 \times 10^1 + 2 \times 10^2$
On dit que 4 est le chiffre des unités, 5 celui des dizaines 10^1 et 2 celui des centaines 10^2
Au lieu de procéder avec des puissances de 10 on va procéder avec des puissances de 2, cependant cette fois ci nous aurons comme chiffres que 0 et 1
Ce qui signifie qu'en base 2 ou binaire :

$(11111110)_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$

Les bits sont regroupés par paquets de 8, appelé **octet**

1 Représentation d'un entier naturel en binaire

Par quel **algorithme** peut on écrire un **n'importe quel entier naturel donné** en binaire ?

Un algorithme est une méthode opérationnelle permettant de résoudre tous les cas particuliers d'un problème donné.

On peut écrire un premier algorithme pour écrire un entier n donné en binaire sous la forme d'une recette de cuisine en Français :

Pour écrire n sous la forme d'une somme de puissance de 2, procéder ainsi : **Tant qu'on peut le faire** , retrancher de n la plus grande puissance de 2 possible contenue dans n et recommencer le processus avec le reste

Exécutons cet algorithme avec un tableau sur plusieurs cas particuliers

1. Pour $n = 10$

n	10	2	0
2^k	$8 = 2^3$	$2 = 2^1$	FIN

D'où le résultat $10 = 2^3 + 2^1 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (1010)_2$

2. Pour $n = 13$

n	13	5	1	0
2^k	$8 = 2^3$	$4 = 2^2$	$1 = 2^0$	FIN

D'où le résultat $13 = 2^3 + 2^2 + 2^0 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (1101)_2$

Si on veut traduire cet algorithme en utilisant un **langage** de programmation comme Python, pour le faire exécuter par une machine,il nous faut expliciter certains passages de cet algorithme.

(Voir le cours sur la boucle while)

On remarque que l'écriture binaire d'un nombre pair comme 10 se termine par 0 alors que celle d'un nombre impair se termine par 1, autrement dit

On divise 13 par 2 et le reste est le dernier chiffre de l'écriture binaire

$13 = 2 \times 6 + 1$

On recommence avec 6 en divisant 6 par 2, le reste obtenu 0, est l'avant dernier chiffre de l'écriture binaire, en effet

$6 = 2 \times 3 + 0$ donc $13 = 2 \times (2 \times 3 + 0) + 1$ donc $13 = 2^2 \times 3 + 2^1 \times 0 + 1$ et on continue ainsi **jusqu'à ce que le quotient soit nul**

$13 = 2 \times 6 + \underline{1}$

$6 = 2 \times 3 + \underline{0}$

$3 = 2 \times 1 + \underline{1}$

$1 = 2 \times 0 + \underline{1}$

d'où un deuxième algorithme :

Pour écrire n en binaire, procéder ainsi : **Tant qu'on peut le faire (tant que n n'est pas nul)** , diviser n par 2 (division euclidienne) et recommencer avec le quotient obtenu. La suite des restes obtenus est l'écriture en binaire de n dans l'ordre inverse.

2 Information-Algorithme-Langage-Machine

On a vu précédemment les quatre grandes idées qui structurent l'Informatique

1. **Information :** Nombres entiers, à virgule, caractères, images, vidéos, sons.
2. **Algorithme :** Beaucoup : sur les nombres, les caractères, les images, etc...
3. **Langage :** Langage de programmation comme Python, langage formel (HTML)
4. **Machine :** Ordinateurs, smartphones, box, microcontrôleurs, robots, commutateurs, routeurs, iot, etc...

3 Base $b \leqslant 10$

Au lieu de représenter un entier en binaire on aimerait le représenter dans une autre base $b \leqslant 10$

On admet le résultat suivant

Théorème 1. *Tout entier naturel $n \geqslant 1$ peut se décomposer dans la base $2 \leqslant b \leqslant 10$ sous la forme $n = c_0 + c_1 \times b^1 + c_2 \times b^2 + ... + c_m \times b^m = (c_m...c_1c_0)_b$*

où $c_i \in \{0, 1, ..., b-1\}$ et m l'unique entier tel que $b^m \leqslant n < b^{m+1}$

L'écriture de n $(c_m...c_1c_0)_b$ comporte $m+1$ chiffres

Concrètement

On veut représenter 45 en base 4 :

n	45	13	1	0
4^k	$32 = 2 \times 4^2$	$12 = 3 \times 4^1$	$1 = 1 \times 4^0$	FIN

Donc $45 = 2 \times 4^2 + 3 \times 4^1 + 1 \times 4^0 = (231)_4$
Autrement
$45 = 4 \times 11 + 1$
$11 = 4 \times 2 + 3$
$2 = 4 \times 0 + 2$
Donc $45 = (231)_4$

4 Hexadécimal

Il existe des outils pour "voir" les images numériques ou les programmes informatiques tel qu'ils sont dans la machine ou presque.
Presque car on préfère utiliser une expression des nombres, intermédiaire entre le binaire (machine) et le décimal(humain), appelée **la notation hexadécimale** (on verra pourquoi après)
Ainsi voici un morceau de l'image ci-dessus en hexadécimal :

```
00000000: 50 32 0A 23 20 43 52 45 41 54 4F 52 3A 20 47 49
00000010: 4D 50 20 50 4E 4D 20 46 69 6C 74 65 72 20 56 65
00000020: 72 73 69 6F 6E 20 31 2E 31 0A 33 32 30 20 32 34
00000030: 30 0A 32 35 35 0A 31 0A 31 0A 31 0A 31 0A 31 0A
00000040: 32 0A 32 0A 33 0A 33 0A 32 0A 32 0A 32 0A 32 0A
00000050: 31 0A 31 0A 31 0A 31 0A 32 0A 33 0A 33 0A 36 0A
00000060: 31 30 0A 31 36 0A 32 31 0A 34 30 0A 38 30 0A 31
00000070: 31 35 0A 31 32 31 0A 31 31 38 0A 31 32 30 0A 31
00000080: 32 31 0A 31 32 33 0A 31 32 34 0A 31 32 33 0A 31
00000090: 32 33 0A 31 32 39 0A 31 33 30 0A 31 32 38 0A 31
000000A0: 33 30 0A 31 33 38 0A 31 34 32 0A 31 33 36 0A 31
000000B0: 32 33 0A 31 32 35 0A 31 33 35 0A 31 33 39 0A 31
000000C0: 33 37 0A 31 34 32 0A 31 33 39 0A 31 33 36 0A 31
000000D0: 34 35 0A 31 34 32 0A 31 33 38 0A 31 34 30 0A 31
000000E0: 34 33 0A 31 34 31 0A 31 33 37 0A 31 34 30 0A 31
000000F0: 33 39 0A 31 32 37 0A 31 31 33 0A 39 39 0A 31 30

00000100: 30 0A 31 31 37 0A 31 32 35 0A 31 32 33 0A 31 32
00000110: 31 0A 31 33 33 0A 31 33 39 0A 31 33 34 0A 31 32
00000120: 30 0A 31 31 30 0A 31 31 34 0A 31 31 31 0A 31 31
00000130: 31 0A 31 30 31 0A 38 39 0A 31 31 31 0A 31 32 36
00000140: 0A 31 32 38 0A 31 33 38 0A 31 33 35 0A 31 33 31
00000150: 0A 31 32 38 0A 31 34 35 0A 31 37 30 0A 31 38 35
00000160: 0A 31 37 35 0A 31 35 37 0A 31 33 34 0A 31 31 32
00000170: 0A 39 35 0A 38 39 0A 31 31 32 0A 39 36 0A 34 35
00000180: 0A 33 33 0A 32 36 0A 32 32 0A 32 34 0A 31 37 0A
00000190: 39 0A 31 33 0A 32 39 0A 32 35 0A 31 35 0A 31 31
000001A0: 0A 36 0A 36 0A 36 0A 35 0A 36 0A 31 30 0A 39 0A
```

Un **octet** est composé de 8 bits

$(c_7c_6c_5c_4c_3c_2c_1c_0)_2 = c_7\times 2^7+c_6\times 2^6+...+c_3\times 2^3+...+c_0 = 2^4(\underline{c_7 \times 2^3 + c_6 \times 2^2 + ... + c_4})+ \underline{c_3 \times 2^3 + ... + c_0}$

Or les quantités soulignées sont comprises entre 0 et 15 d'où le théorème

Théorème 2. *Tout octet est traduit avec 2 symboles en base 16 où les chiffres sont 0 jusqu'à 9 puis A,B,C,D,E et F*
Avec $A_{16} = 10_{10}$, $B_{16} = 11_{10}$, $C_{16} = 12_{10}$, $D_{16} = 13_{10}$, $E_{16} = 14_{10}$ *et* $F_{16} = 15_{10}$

Pour écrire un nombre entier en base seize on peut passer par le binaire, comme précisé ci-dessus.

Cependant les algorithmes vus précédemment sont encore valables :

$2024 = 16 \times 126 + 8$

$126 = 16 \times 7 + 14$ or 14 est symbolisé par e

$7 = 16 \times 0 + 7$

Donc $2024 = 0x7e8$

En Informatique la représentation d'un nombre en hexadécimal est précédé de 0x

Exemples

1. $254 = (1111\ 1110)_2 = (FE)_{16} = 0xfe$
2. Le code html d'une couleur se fait sur 3 octets notés en hexadécimal précédé du symbole #. Ainsi le code #ffff00 code une couleur jaune.
3. Une adresse IPv6 est composée de 8 groupes de seize bits (2 octets) en hexadécimal séparés par le symbole :(en tout 128 bits)
   par exemple 2001 :0db8 :0bad :0cafe :0000 :0000 :0000 :00ab est une adresse IPv6 valide.

Exercices

Ex 1

1. Décomposer en base 2 :
   10, 15, 16, 32, 127, 255 et 2024
2. Que valent $(1010)_2$, $(111010)_2$, $(10101110)_2$ en base 10 ?
3. Que valent $(11)_2$, $(111)_2$, $(1111)_2$, $(11111111)_2$? Quelle est la logique ?

Ex 2

1. Calculer $(1010)_2 + (11)_2$ et vérifier la compatibilité avec la même opération en base 10
2. Calculer $(1110)_2 + (11)_2$ et vérifier la compatibilité avec la même opération en base 10

Ex 3

Décomposer en base 8 :
16, 31,64 et 2019

Ex 4

En quel sens comprenez vous cette phrase :
"Il n'y a que 10 types de personnes au monde, ceux qui comprennent le binaire et ceux qui ne le comprennent pas"

Ex 5

Pourquoi les informaticiens confondent Noël et Halloween ?
Vérifier que Dec 25 = Oct 31

Ex 6

Classer par ordre croissant de capacité :
2000 octets, 15 Ko, 4 Go, 1,5 Mo, 1 To, 2 millions de bits, 2 000 000 bytes

Ex 7

Convertir les entiers suivants en hexadécimal : 9,10,15,16,31 ,32, 255,256 et 2020

Ex 8

Que vaut 0xfff en base 10 ?

Ex 9

If only DEAD people understand hexadecimal, how many people understand hexadecimal ?

Ex 10

1. Voici une série d'octets séparés par / en binaire les traduire en hexadécimal :
   1000 0011 / 0001 1010/1111 1111/1111 1100/0101 1010
2. Voici une série d'octets séparés par / en hexadécimal les traduire en binaire :
   0xff / 0xaa / 0xbe/ 0x1a/0xa1

Ex 11

Coder les couleurs RVB suivantes en couleurs html

1. (127 ;127 ;127)
2. (255 ;63 ;10)
3. (255,255,255)

Ex 12

Coder les couleurs html en RVB

1. aabbcc
2. 00ff00
3. 999a9b

Ex 13

L'adresse MAC (Media Access Control) caractérise de manière unique une carte réseau. Elle est constituée de 6 octets séparés par le symbole.

Rechercher les adresses MAC de votre machine (Ethernet et wi-fi)

Ex 14

Parmi les adresses suivantes lesquelles sont des adresses IPv6 valides

1. 2002 :1002 :4321 :0000 :0000 :0000 :abcd
2. 2001 :0db8 :900d :cafe :0000 :0000 :0000 :1200
3. 2001 :0db8 :good :cafe :0000 :0000 :0000 :1200

Ex 15

1. Aller sur ce site `http://g6.asso.fr`. Que voyez vous en haut à droite de la page ? Utilisez vous ipv6 ?
2. Rechercher l'information dans cette page `http://livre.g6.asso.fr/index.php/AdressageBis-Fondamentaux` vous permettant de décrypter une adresse comme 2001 :db8 :900d : :1