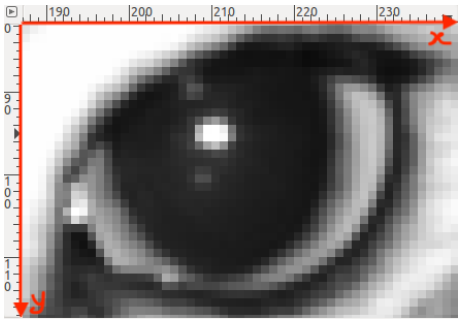
Représentation des entiers naturels dans une base b

Les machines actuelles (ordinateurs, tablettes, smartphones..) traitent les données (images, sons, textes) 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 en 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)

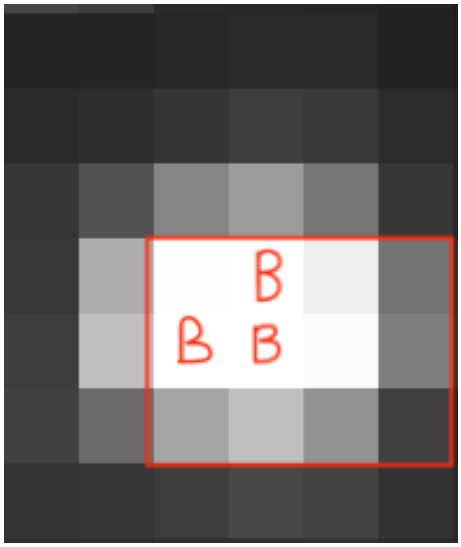
Nous pouvons observer cela en utilisant la bibliothèque PIL de Python , on fait afficher les intensités des pixels de coordonnées (209,94), (210,94), (209,95) et (210,95)

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

Ensuite à l'aide d'un programme Python on fait afficher les intensités des pixels du rectangle rouge

```
from PIL import Image

#on charge l'image du chat
image1 = Image.open("/Users/vallon/Documents/chat.png")

for y in range(94,97):
    print()
    for x in range(209,213):
```

```
>>> from PIL import Image

>>> image = Image.open("/Users/vallon/Documents/chat.png")

>>> image.getpixel((209,94))
254

>>> image.getpixel((210,94))
255

>>> image.getpixel((209,95))
255

>>> image.getpixel((210,95))
255

>>> image.getpixel((207,91))
36
```



```
print(image1.getpixel((x,y)),end=' ')
```

On obtient :

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

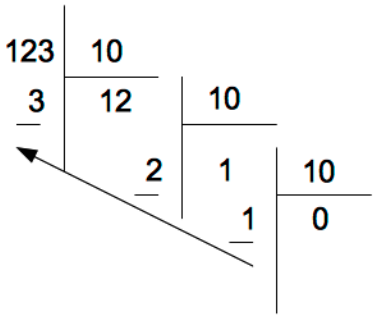
Ces nombres sont pour les humains, les machines travaillent plutôt avec des 0 et des 1

```
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
$(254)_{10} = 4 \times 10^0 + 5 \times 10^1 + 2 \times 10^2$
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 que $(11111110)_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$
Coment obtient on les chiffres ?Par quel algorithme les obtient on ? On pose la division

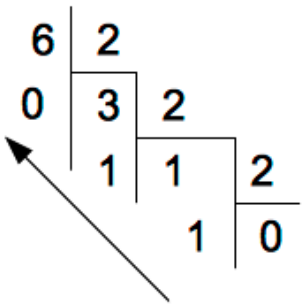


euclidienne de 123 par 10 puis celle de 12 (le quotient de la division précédente) par 10 puis celle de 1 (le quotient de la division précédente) par 10 et puisque le quotient de la dernière division est nul on arrête l'algorithme
Les chiffres de la décomposition de 123 en base 10 sont les **restes**

1. Le premier reste obtenu est le chiffre des unités

2. Le dernier donne le nombre de la plus grande puissance de 10 contenue dans le nombre donné

Décomposons 6 maintenant en puissances de 2 On observe le même phénomène les



chiffres sont les restes....

1 Représentation d'un entier naturel dans une base $b \leqslant 10$

Voici l'algorithme en base $b \leqslant 10$

Algorithme 1 : décomposition d'un entier n en base b $\leqslant 10$

Données : Un entier naturel n, une base b $\leqslant 10$
Résultat : les chiffres de n à la volée dans l'ordre chiffre des unités vers chiffre

```
1  début
2      nombre ← n
3      base ← b
4      quotient ← nombre
5      tant que quotient ≠ 0 faire
6          chiffre ← nombre % base
7          afficher chiffre
8          quotient ← nombre // base
9          afficher reste
10     fin
11 fin
```

En Python

```
nombre = int(input("Entrez un entier naturel "))
base     = int(input("Entrez une base b <= 10 "))
quotient = nombre

while quotient > 0:
    chiffre = quotient \% base
    print(chiffre,end = " ")
    quotient = quotient // base
```

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
```

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

Preuve
Soit $m \geqslant 1$ l'unique entier tel que $b^{m-1} \leqslant n < b^m$. On peut donc diviser n par b , $m-1$ fois
La division euclidienne de n par b donne $n = ba_1 + r_1$ avec $0 \leqslant r_1 < b$

On recommence et $a_1 = ba_2 + r_2$ avec $0 \leqslant r_2 < b$ et $a_2 < a_1$ et $n = b(ba_2 + r_2) + r_1 = b^2a_2 + r_2 \times b + r_1$

Soit $a_2 = 0$ et $n = r_2 \times b + r_1$ est l'écriture de n soit on recommence et $a_2 = ba_3 + r_3$ avec $0 \leqslant r_3 < b$ donc $n = b^3a_3 + r_3 \times b^2 + r_2 \times b + r_1$ et $a_3 < a_2 < a_1$

Et puisqu'on ne peut pas avoir dans $\mathbb{N}$ une suite d'entiers strictement décroissante forcément si on divise m fois on obtient $a_m = 0$

A la fin $n = r_m b^{m-1} + ... + r_3 \times b^2 + r_2 \times b + r_1$

Exemple

$13 = 2 \times 6 + 1$

$6 = 2 \times 3 + 0$

$3 = 2 \times 1 + 1$

$1 = 2 \times 0 + 1$

Donc $13 = (1101)_2$

2 Hexadécimal

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* $F_{16} = 15_{10}$

Exemple

$254 = (1111\ 1110)_2 = (FE)_{16} = 0$xfe

Exercices

Ex 1

1. Décomposer en base 2 :
   10, 15, 16, 32, 127, 255 et 2019
2. Que vaut $(1010)_2$ en base 10 ? Que vaut $(101010)_2$ puis $(10101010)_2$en base 10. 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

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

Ex 5

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 6

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

Ex 7

Que vaut 0xfff en base 10 ?

Ex 8

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

Ex 9

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 10

Coder les couleurs RVB suivantes en couleurs html

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

Ex 11

Coder les couleurs html en RVB

1. aabbcc
2. 00ff00
3. 999a9b