NSI 1ère - Données

Hexadécimal

qkzk

Hexadécimal

Hexadécimal

Les nombres en binaires sont longs. On utilise souvent la base 16 pour les manipuler plus facilement.

Chiffre hexadécimaux

On utilise 16 chiffres :

Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

16 chiffres : 0 1 2 4 5 6 7 8 9 A B C D E F

Convertir un binaire en hexa est facile. Chaque paquet de 4 bits donne un chiffre hexa :

$$1010\ 0011\ 1011\ 1100_2 = A3BC_{16}$$

Notations

Maths	Python	CSS
$A3BC_{16}$	`0xA3BC`	`#A3BC`

Pour aller vite on peut utiliser une table

$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

De l'hexadécimal vers le décimal

Pour convertir $4D5_{16}$ de l'hexa. vers le décimal,
on commence par le dernier chiffre :

- 5×16^0 et on recule :
- 13×16^1 (D correspond au nombre 13)
- 4×16^2

$$4D5_{16} = 5 \times 16^0 + 13 \times 16^1 + 4 \times 16^2 = 1\ 237_{10}$$

Du décimal vers l'hexadécimal

- Divisions entières successives par **16** *jusqu'à trouver 0.*
  Les **restes** donnent les chiffres dans l'ordre **inverse**

$$959 = 59 \times 16 + 15 \longrightarrow F$$

$$59 = 3 \times 16 + 11 \longrightarrow B$$

$$3 = 0 \times 16 + 3 \longrightarrow 3$$

$$959_{10} = 3BF_{16}$$

Python

```
>>> int('3BF', 16)
959
>>> hex(959)
'0x3bf'
>>> 0xA3BC # c'est un entier pas une chaîne !!!
41916
>>> 0xa3bc # majuscule ou minuscule
41916
```

Représenter facilement des octets ?

Pour la machine, l'unité de stockage la plus petite n'est pas le bit mais l'octet.

Comment représenter facilement un octet ?

256 octets possibles... 256 symboles ? Difficile...

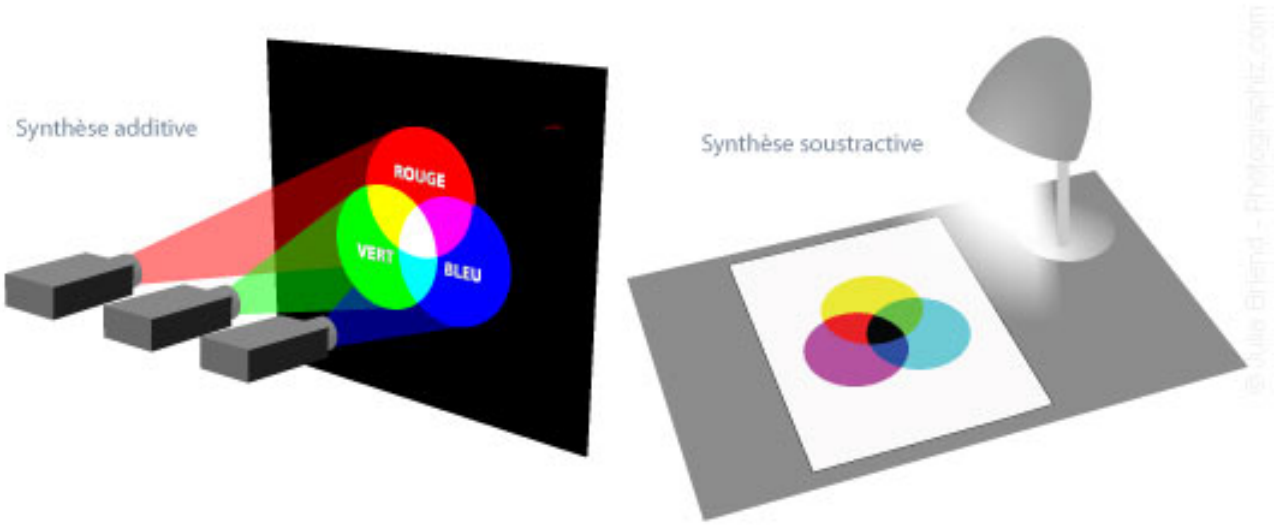
Mais ! $256 = 16^2$ on peut utiliser 2 symboles en base 16.

Un octet est représenté par 2 chiffres hexadécimaux

Les couleurs

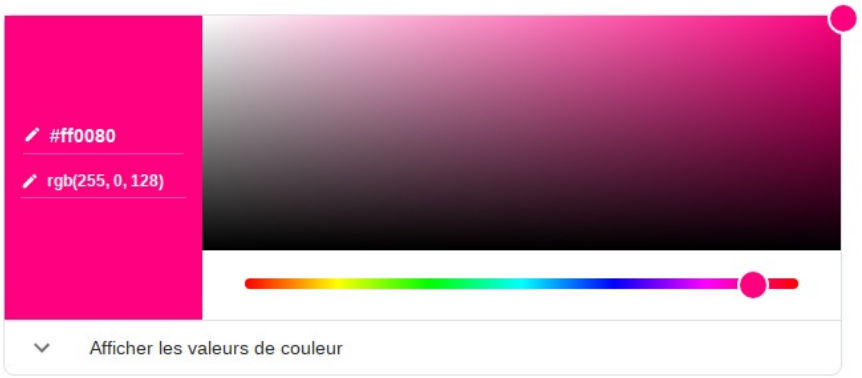
En informatique on distingue

- les couleurs à l'écran : synthèse additive
- les couleurs imprimées : synthèse soustractive



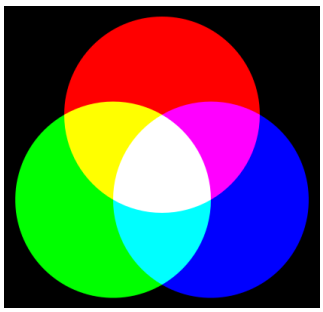
Synthèse additive

- En **synthèse additive** on utilise 256 niveaux de couleur pour les composantes Rouge, Vert et Bleu.
- Chaque niveau de couleur est codé sur un octet.
- `#FF0080` : FF rouge à fond, 00 pas de vert, 80 bleu à moitié : un joli rose, noté parfois : `rgb(255, 0, 128)`



Quelques exemples

blanc	#FFFFFF	noir	#000000
rouge	#FF0000	jaune	#FFFF00
vert	#00FF00	cyan	#00FFFF
bleu	#0000FF	magenta	#FF00FF



Synthèse soustractive

- En **synthèse soustractive** on utilise souvent **CMJN** :
  cyan, magenta, jaune et noir.
- Le niveau de noir permet d'économiser les encres et améliore le rendu.

On a développé de nombreuses méthodes.

Le contenu d'un fichier

Un fichier en machine n'est pas toujours lisible directement.

Ouvrir une image avec un lecteur de texte produit un résultat décevant.
Comment lire facilement les octets qui la constituent ?

```
$ hexdump img/ff0080.jpg | head
0000000 d8ff e0ff 1000 464a 4649 0100 0101 4800
0000010 4800 0000 e2ff 8823 4349 5f43 5250 464f
0000020 4c49 0045 0101 0000 7823 636c 736d 1002
0000030 0000 6e6d 7274 4752 2042 5958 205a df07
0000040 0b00 0a00 0c00 1200 3800 6361 7073 6e2a
0000050 7869 0000 0000 0000 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0000 0000 d6f6 0100
0000070 0000 0000 2dd3 636c 736d 0000 0000 0000
0000080 0000 0000 0000 0000 0000 0000 0000 0000
*
```

Que fait la commande ?

```
hexdump img/ff0080.jpg | head
```

- `hexdump` : affiche les octets d'un fichier sous forme hexadécimale
- `img/ff0080.jpg` : l'image avec la couleur rose vue plus tôt
- `| head` : ne garder que le début du fichier

Comment lire le résultat ?

- Première colonne : position dans le fichier

  ```
  0000000
  0000010   <--- Cette ligne commence à l'octet x10
  0000020
  0000030
  ```

- `0000020 4c49 0045` :

Position	x20	x21	x22	x23
Contenu	x4c	x49	x00	x45
Contenu	76	73	0	69

Le contenu de mon image .jpg est donc UN NOMBRE, encodé en binaire, que la machine interprète à l'aide d'un programme et affiche à l'écran.