

# REPRÉSENTATION DE L'INFORMATION

Vendredi 5 janvier

Option Informatique  
Ecole Alsacienne

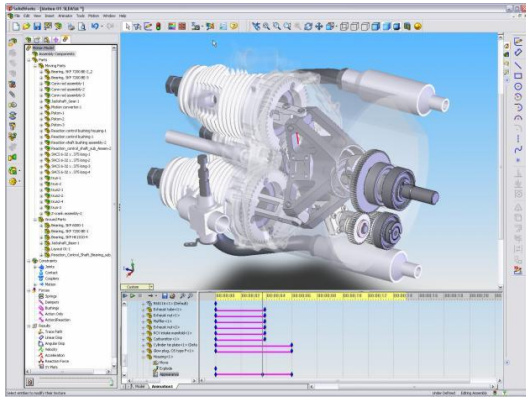
BONNE ANNÉE !

# PLAN

1. Langages informatiques
2. Représenter des nombres
3. Représenter du texte
4. Représenter des images
5. Représenter du son
6. Exercices

# LANGAGES INFORMATIQUES

# UN ORDINATEUR PERMET BEAUCOUP DE CHOSES...



Concevoir et réaliser



Echanger des informations



WIKIPEDIA

Accéder à la connaissance



Créer des œuvres d'art



Faire la poussière



Faire des calculs dans les nuages

# UN ORDINATEUR EST CAPABLE DE BEAUCOUP DE CHOSES...

*Comment obtenir de tels résultats ?*



Avec des 0 et des 1 !

# LANGAGES DE PROGRAMMATION

- On appelle **langage naturel** une langue « normale » utilisée par les êtres humains pour communiquer entre eux.
- Le langage naturel est trop riche et trop complexe pour être compris tel quel par un ordinateur (le traitement du langage naturel est d'ailleurs un vaste domaine de recherche).
- Un **langage de programmation** permet à un être humain de donner des instructions à un ordinateur.
- Un langage de programmation est très codifié : il existe un nombre fini d'instructions et la syntaxe doit être respectée

# LANGAGES DE PROGRAMMATION

- Il existe une grande variété des langages de programmation.
- On distingue notamment les **langages de haut niveau** et les **langages de bas niveau**

Langages de haut niveau	Langages de bas niveau
Plutôt abstrait	Plutôt concret
Peu dépendant de la machine	Dépendant de la machine
Moins de choses à gérer	Plus optimisable

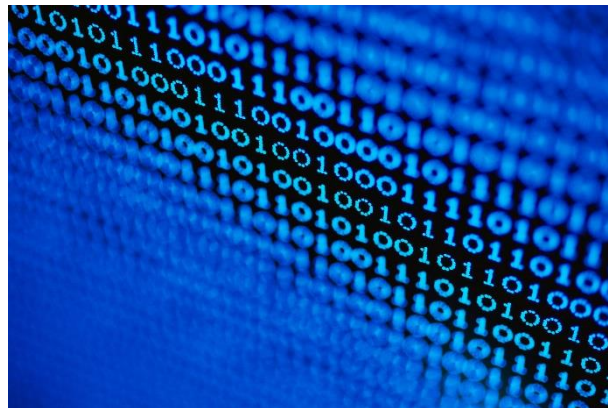
# ASSEMBLEUR

- L'**assembleur** est un langage de très bas niveau
- Il est basé sur :
  - Un petit nombre de variables appelées **registres**
  - Des instructions basiques :
    - Déplacement entre la mémoire et les registres
    - Calculs simples sur les registres (addition, soustraction, comparaison)
    - Modification dans le déroulement du programme (ex : saut conditionnel)
- Exemple :

```
mov    ax, '00'  
mov    di, counter  
mov    cx, digits+cntDigits/2  
cld
```
- On s'en sert encore aujourd'hui pour optimiser certaines tâches

# LANGAGE MACHINE

- Le **langage machine** est le langage utilisé par le processeur d'un ordinateur.
- Ce langage n'est composé que de 0 et de 1 : on dit qu'il est écrit en **binaire**.
- Une telle valeur, 0 ou 1, s'appelle un **bit**
  - Contraction de « binary digit »
  - Jeu de mot sur « bit » (morceau en anglais)



# LANGAGE MACHINE

- Exemple de langage machine (processeur MIPS)

000000 00001 00010 00110 00000 100000

« Ajouter les registres 1 et 2 et placer le résultat dans le registre 6 »

- Remarques :
  - Un programme écrit en langage machine est illisible pour un humain
  - On retrouve la notion de registres présente dans l'assembleur
  - Le langage machine, comme l'assembleur, est dépendant de la machine sur lequel il s'exécute.

# LANGAGES INTERPRÉTÉS ET LANGAGES COMPILÉS

- *Comment un programme est-il traduit en langage machine ?*
- **Langages compilés**
  - La traduction en langage machine est faite « une bonne fois pour toutes »
  - Cette traduction est faite par un programme annexe appelé **compilateur**
  - Au terme de cette compilation, un nouveau fichier est généré
  - Ce nouveau fichier, appelé **exécutable**, est autonome
  - Exemple : C++
- **Langages interprétés**
  - La traduction en langage machine est faite « à la volée »
  - Cette traduction est faite par un programme annexe appelé **interpréteur**
  - Exemple : MATLAB

# LANGAGES INTERPRÉTÉS ET LANGAGES COMPILÉS

- Avantages des langages compilés :
  - L'exécutable n'a pas besoin d'un programme annexe pour être utilisé
  - Généralement plus rapide (la traduction est déjà faite)
  - Le code est "sécurisé" (illisible par un être humain)
- Avantages des langages interprétés :
  - Un seul fichier (intelligible)
  - Les petites modifications sont aisées

# LANGAGES INTERPRÉTÉS ET LANGAGES COMPILÉS

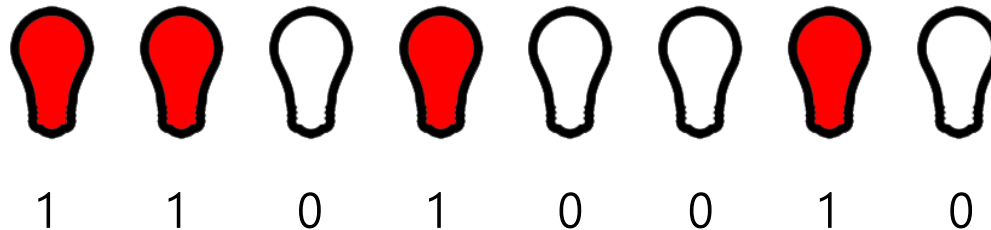
- **Langages "intermédiaires" :**
  - La traduction en langage machine est fait « en deux étapes »
  - Le programme est compilé dans une version "intermédiaire"
    - Non intelligible par l'homme
    - Non exécutable sans un interpréteur
  - Exemple : Applets Java (utilisable uniquement dans un navigateur)
- On appelle **bytecode** le code intermédiaire entre le code source et le langage machine.
  - Chaque instruction est codée en binaire, d'où le mot « bytecode »
  - Un interpréteur de bytecode est souvent appelé « machine virtuelle »

# LANGAGES INTERPRÉTÉS ET LANGAGES COMPILÉS

- Et Python ?
- Python est l'un de ces langages "intermédiaires".
  - Il y a d'abord une passe de compilation vers du bytecode (des fichiers `.pyc` sont générés à partir des fichiers `.py`)
  - Puis ce byte code est interprété
- Remarques
  - La frontière entre "compilation" et "interprétation" peut être floue
  - Pour un même langage, il peut exister plusieurs implémentations

## AU CŒUR DE LA MACHINE...

- Les composants d'un ordinateur contiennent une multitude de petits circuits électriques.
- Chacun de ces circuits n'a que deux états possibles, qu'on a choisis d'appeler 0 et 1.
- On peut également voir ça comme autant de lampes allumées ou éteintes

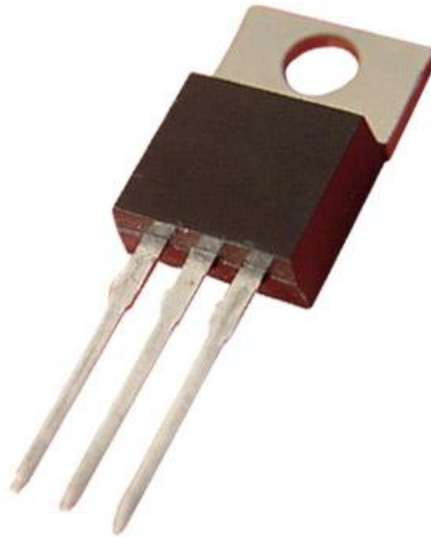


# BITS, BYTES, OCTETS

- Quelle est la différence entre :
  - Un ordinateur avec 4 Gb de RAM
  - Un ordinateur avec 4 GB de RAM
  - Un ordinateur avec 4 Go de RAM
- Un **octet** est une séquence de 8 bits
- Un **byte** est la plus « petite unité adressable d'un ordinateur », c'est-à-dire la plus petite entité avec laquelle on peut interagir.
- Dans la grande majorité des machines, un byte contient 8 bits
- Ainsi, on manipule généralement les bits 8 par 8.

## AU CŒUR DE LA MACHINE...

- Qu'est-ce que c'est ?

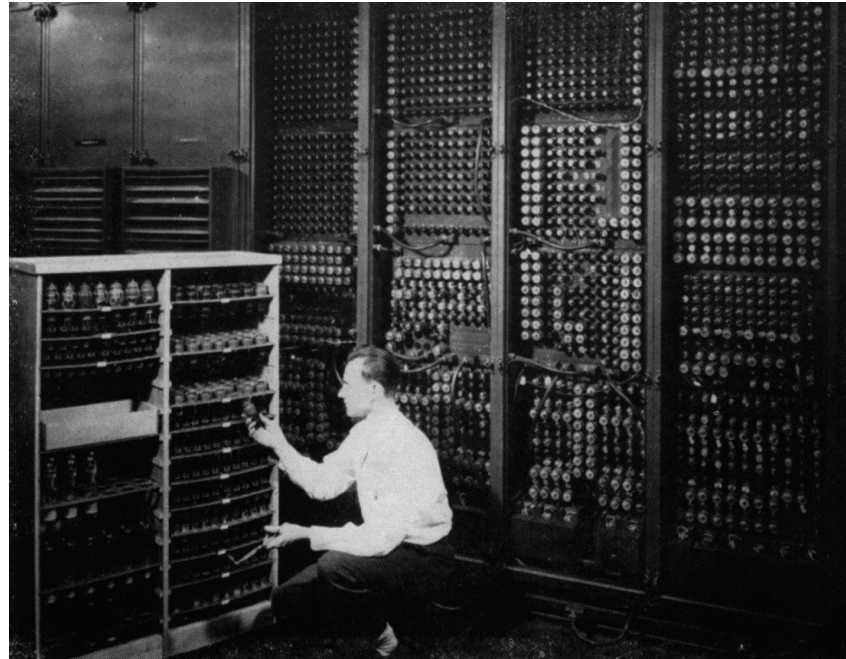


Un transistor !

- Le **transistor** est l'un des composants fondamentaux d'un ordinateur

# PETIT VOYAGE DANS LE TEMPS

- *Qu'est-ce que c'est ?*



- Le premier ordinateur entièrement électronique
  - Date de présentation au public : 1946
  - *Electronic Numerical Integrator Analyser and Computer (ENIAC)*

# POUR L'INSTANT...



Nous allons surtout nous intéresser sur ce qu'il est possible de faire avec des 0 et des 1.

# RÉPRÉSENTER DES NOMBRES

## UN EXERCICE TRÈS DIFFICILE

- Question très difficile :

*Combien font 4 plus 2 ?*

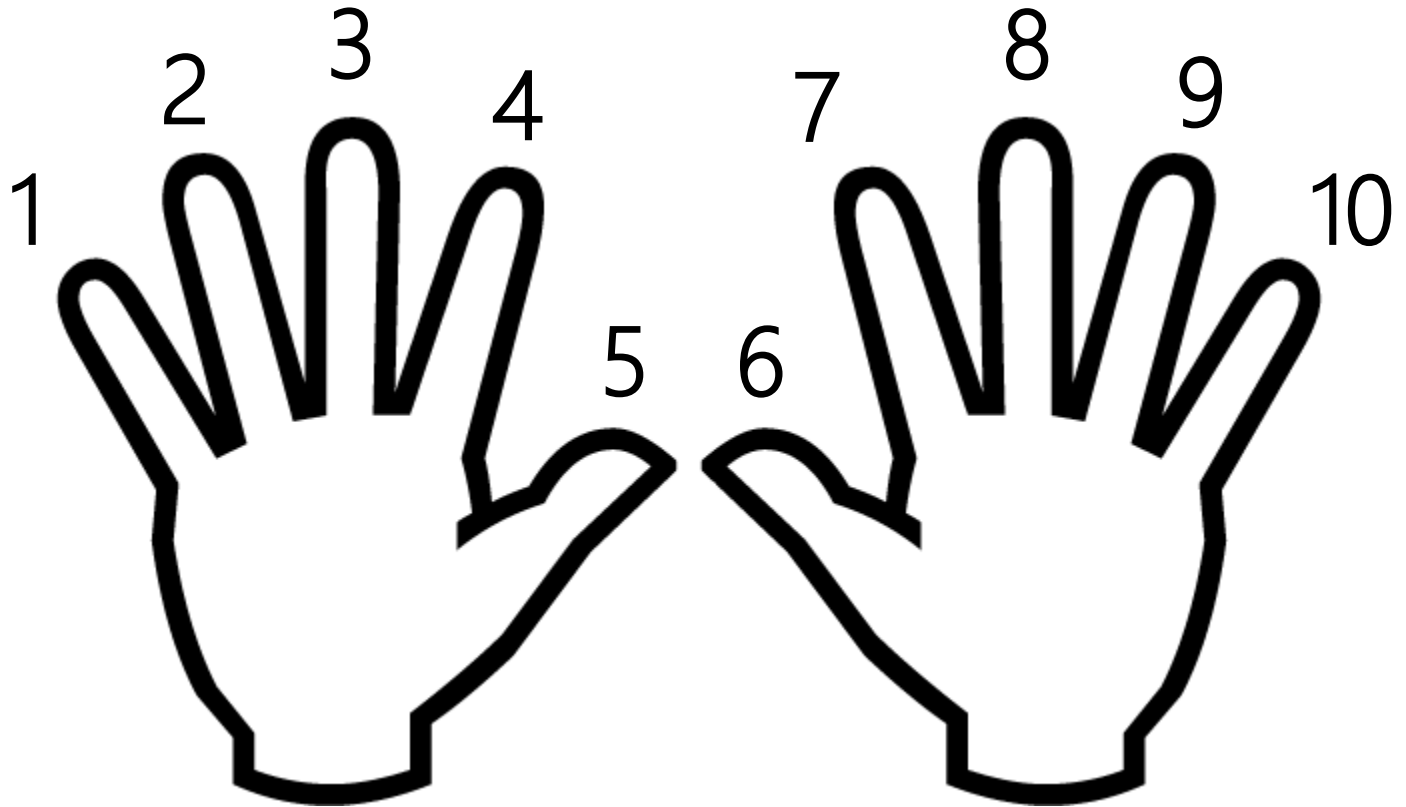
- Test d'agilité :

*Êtes-vous encore capable de faire ce calcul  
en comptant sur vos doigts ?*

- Question encore plus difficile :

*Combien font 6 plus 8 ?*

# UN EXERCICE TRÈS DIFFICILE



*Cela fait des années que vous comptez en base 10*

# ÉCRITURE EN BASE 10

- Sans forcément le savoir, nous utilisons tous les jours des nombres écrits **en base 10**
- En effet, tout nombre entier s'écrit comme une succession de chiffres :

$$x = \underline{x_k x_{k-1} \dots x_2 x_1 x_0}_{10}$$

avec  $x = x_k \times 10^k + x_{k-1} \times 10^{k-1} + \dots + x_2 \times 10^2 + x_1 \times 10^1 + x_0 \times 10^0$

et  $0 \leq x_i < 10$  pour tout  $i$

- Exemple :

$$\underline{7865}_{10} = 7 \times 1000 + 8 \times 100 + 6 \times 10 + 5 \times 1$$

$$\underline{7865}_{10} = 7 \times 10^3 + 8 \times 10^2 + 6 \times 10^1 + 5 \times 10^0$$

# ÉCRITURE EN BASE 2

- D'autres types d'écriture existent, notamment l'écriture en base 2
- En effet, tout nombre s'écrit également

$$x = \underline{x_n x_{n-1} \dots x_2 x_1 x_0}_2$$

avec  $x = x_n \times 2^n + x_{n-1} \times 2^{n-1} + \dots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$

et  $0 \leq x_i < 2$  pour tout  $i$

- Exemple : 101010<sub>2</sub> ?

$$\underline{101010}_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$\underline{101010}_2 = 32 + 8 + 2$$

$$\underline{101010}_2 = 42$$

$$\underline{101010}_2 = \underline{42}_{10}$$

## ECRITURE EN BASE 2

- Un nombre écrit en base 2 est plus « long » que ce même nombre écrit en base 10 (environ 3,2 fois)
- On appelle
  - **bit de poids faible** le bit associé à  $2^0$
  - **bit de poids fort** le bit associé à la plus grande puissance de 2

$$42 = \underline{101010}_2$$

- On écrit généralement les nombres avec un bit de poids fort égal à 1

$$\underline{000101010}_2 = \underline{101010}_2$$

# ECRITURE EN BASE 2

- **Question 1.** Quelle est la représentation en base 2 des nombres suivants :
  - 11
  - 42
  - 64
  - 255
  - 1000
  
- **Question 2.** Quelle est la représentation en base 10 des nombres suivants :
  - 111<sub>2</sub>
  - 1001<sub>2</sub>
  - 100000<sub>2</sub>
  - 1010101<sub>2</sub>
  - 10100111001<sub>2</sub>

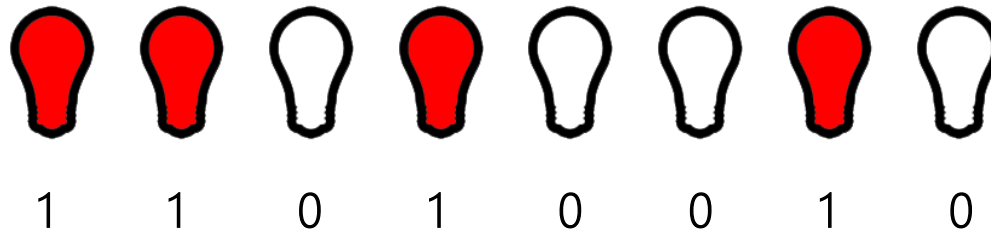
## ECRITURE EN BASE 2

- **Question 3.** On appelle « décalage à gauche » l'opération qui consiste à ajouter un 0 à la fin d'un nombre écrit en binaire.
  - Exemples :  $\underline{11}_2$  devient  $\underline{110}_2$ ,  $\underline{1001}_2$  devient  $\underline{10010}_2$ , etc.
  - A quoi correspond cette opération en base 10 ?
- **Question 4.** On appelle « décalage à droite » l'opération qui consiste à supprimer le bit de poids faible dans un nombre écrit en binaire.
  - Exemples :  $\underline{1010}_2$  devient  $\underline{101}_2$ ,  $\underline{101}_2$  devient  $\underline{10}_2$ , etc.
  - A quoi correspond cette opération en base 10 ?

# REPRÉSENTER UN NOMBRE ENTIER POSITIF

- *Comment représenter un nombre entier positif ou nul avec des 0 et des 1?*
- On utilise son écriture en base 2.

- Exemple :  $210 = \underline{11010010}_2$



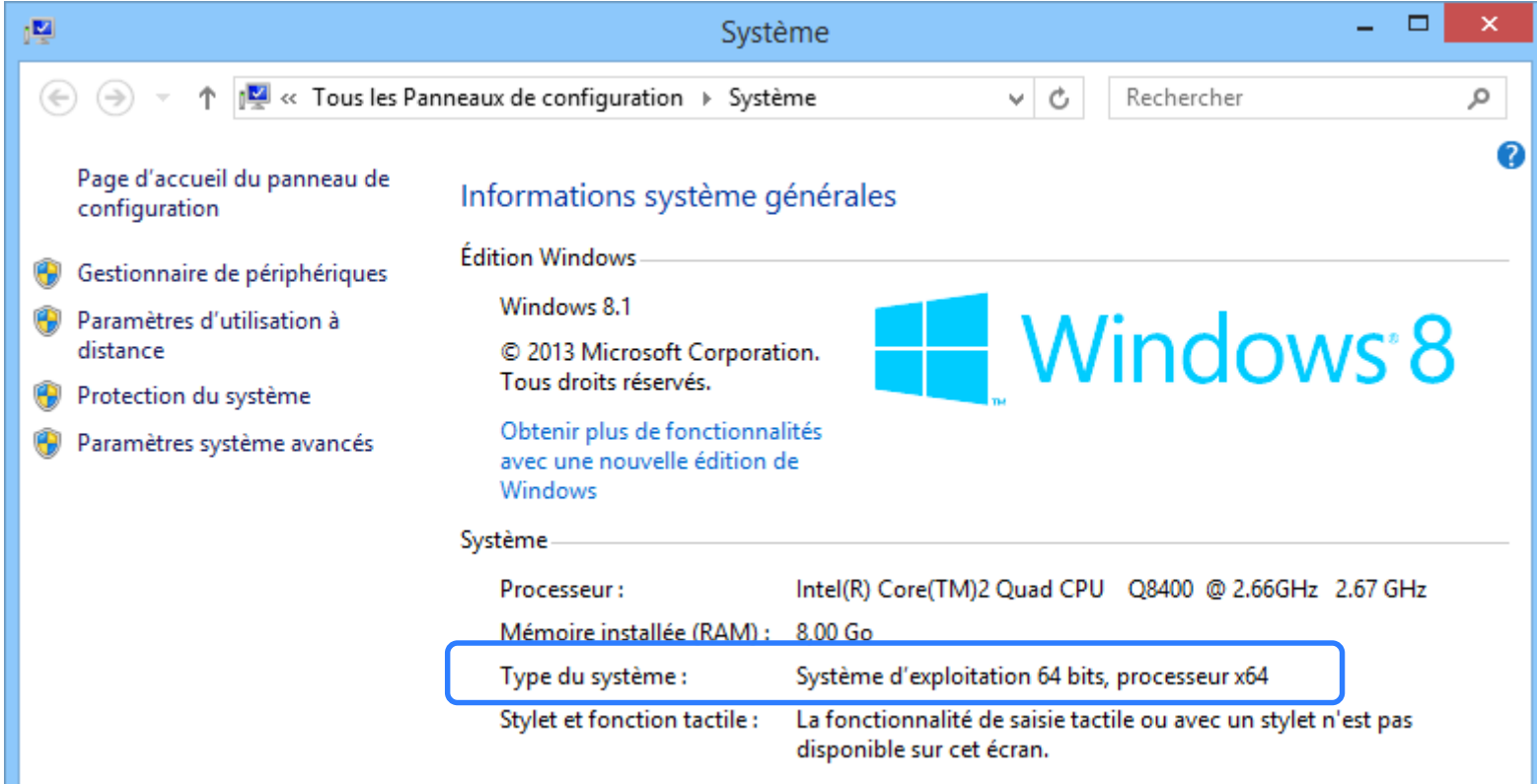
- Questions
  - Est-il possible de représenter n'importe quel entier positif ou nul ?
  - De combien de bits a-t-on besoin ?

# REPRÉSENTER UN NOMBRE ENTIER POSITIF

- **Question 1.** Quel est le plus grand entier positif ou nul qu'on peut représenter avec un octet ?
- **Question 2.** Quel nombre obtient-on si on applique le décalage à gauche sur ce plus grand entier ?
- **Question 3.** Quel nombre obtient-on si on ajoute 1 à ce plus grand entier ?
- **Question 4.** Est-ce que ça vous rappelle quelque chose ?

# ARCHITECTURE 32-BITS ET 64-BITS

- *Quelle est la différence entre une version 32-bits d'un logiciel et une version 64-bits ?*



The screenshot shows the Windows System Control Panel window. The title bar reads "Système". The breadcrumb navigation shows "Tous les Panneaux de configuration > Système". The left sidebar lists navigation options: "Page d'accueil du panneau de configuration", "Gestionnaire de périphériques", "Paramètres d'utilisation à distance", "Protection du système", and "Paramètres système avancés". The main content area is titled "Informations système générales". Under "Édition Windows", it displays "Windows 8.1", "© 2013 Microsoft Corporation. Tous droits réservés.", and a link to "Obtenir plus de fonctionnalités avec une nouvelle édition de Windows". Under "Système", the following information is listed: "Processeur : Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz 2.67 GHz", "Mémoire installée (RAM) : 8.00 Go", "Type du système : Système d'exploitation 64 bits, processeur x64" (highlighted with a red box), and "Stylet et fonction tactile : La fonctionnalité de saisie tactile ou avec un stylet n'est pas disponible sur cet écran."

## LE RETOUR DE LA BASE 2

- **Question 1.** Quelle est la représentation en base 2 des nombres suivants :
  - 18
  - 55
  - 90
  
- **Question 2.** Quelle est la représentation en base 10 des nombres suivants :
  - 10<sub>2</sub>
  - 11011<sub>2</sub>
  - 111000011<sub>2</sub>

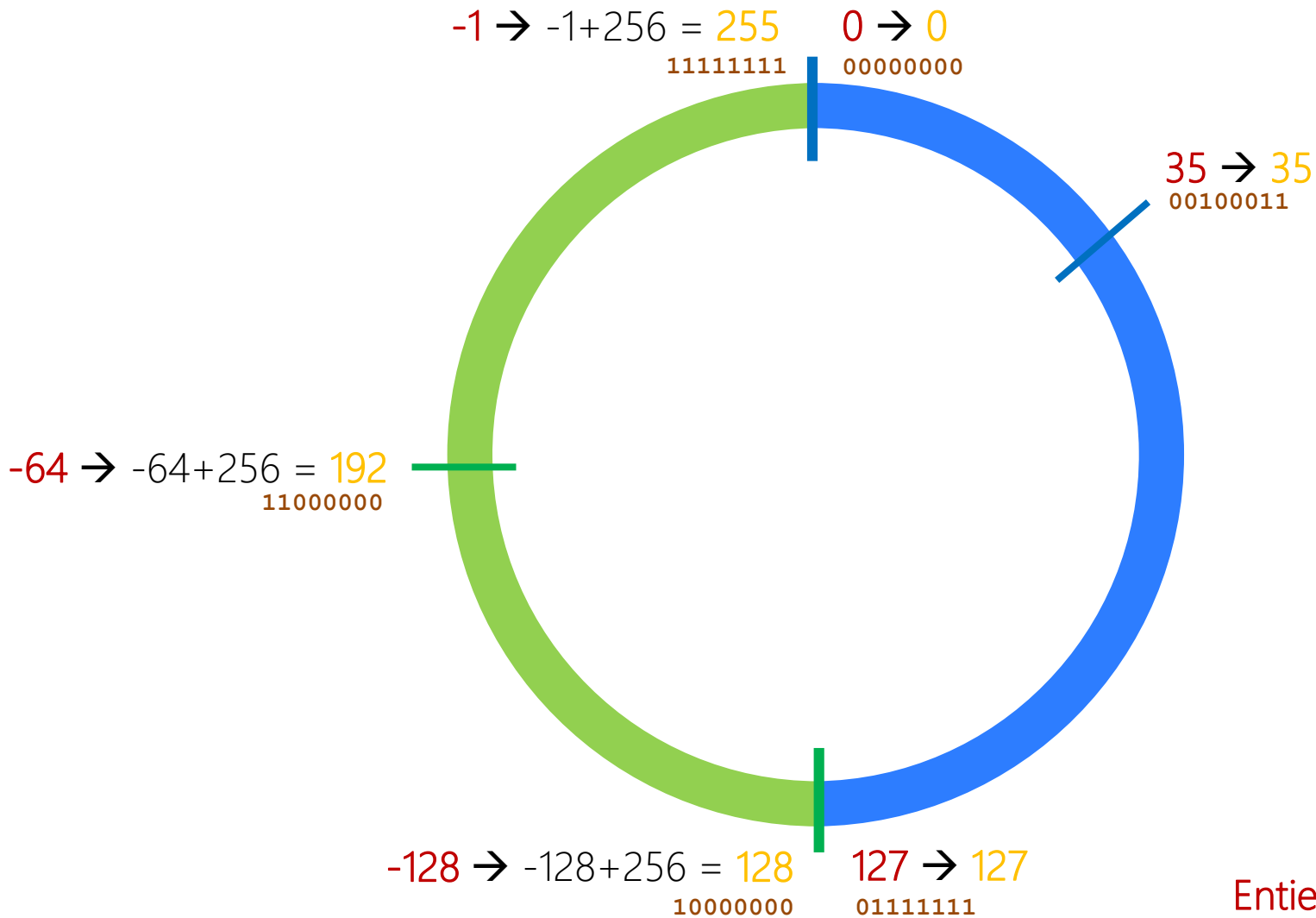
# REPRÉSENTER UN NOMBRE ENTIER SIGNÉ

- *Comment représenter un nombre entier signé (positif ou négatif) avec des 0 et des 1 ?*
- Représentation sur  $n$  bits
  - Si on dispose de  $n$  bits
  - On peut représenter  $2^n$  entiers
    - $2^{n-1}$  entiers positifs
    - $2^{n-1}$  entiers négatifs
  - Comment gère-t-on le **0** ?
- Approche naïve :
  - 1 bit pour le signe
  - le reste pour la valeur absolue du nombre
  - Inconvénient : on aurait ainsi deux zéros (**+0** et **-0**)

# REPRÉSENTER UN NOMBRE ENTIER SIGNÉ

- Représentation **en complément à deux** :
  - Si on dispose de  $n$  bits
  - On peut représenter  $2^n$  entiers
    - $2^{n-1}$  entiers positifs ou nul : entre  $0$  et  $2^{n-1} - 1$
    - $2^{n-1}$  entiers strictement négatifs : entre  $-2^{n-1}$  et  $-1$
  - Représentation :
    - Si  $x$  est positif ou nul, on prend son écriture en base 2
    - Si  $x$  est strictement négatif, on prend l'écriture en base 2 de  $x + 2^n$  ( $x + 2^n$  est alors compris entre  $2^{n-1}$  et  $2^n - 1$ )
- Exemple sur 8 bits
  - $2^8 = 256$  : on va représenter les entiers compris entre  $-128$  et  $127$
  - Si  $0 \leq x \leq 127$ , alors on prend l'écriture en binaire de  $x$
  - Si  $-128 \leq x \leq -1$ , alors on prend l'écriture en binaire de  $x + 256$

# REPRÉSENTER UN NOMBRE ENTIER SIGNÉ



■ ■ ■ ■ ■ Représenter des nombres

Entier signé  
Entier représenté  
Représentation binaire

# REPRÉSENTER UN NOMBRE ENTIER SIGNÉ

- **Question 1.** Quelle est la représentation en complément à 2 (sur 8 bits) des nombres suivants :
  - 29
  - -50
  - -126
  - 130
  
- **Question 2.** Quelle est la représentation en base 10 des nombres suivants (exprimé en complément à 2 sur 8 bits) :
  - 01100011<sub>2</sub>
  - 10010110<sub>2</sub>
  - 11111110<sub>2</sub>

# REPRÉSENTER UN NOMBRE RÉEL

- *Comment représenter un nombre à virgule avec des 0 et des 1 ?*
- Approche naïve
  - Si on dispose de  $n$  bits
  - On utilise  $\frac{n}{2}$  bits pour représenter la partie entière (avant la virgule)
  - On utilise  $\frac{n}{2}$  bits pour représenter la partie décimale (après la virgule)
- Inconvénient : Cette approche ne permet pas de représenter les valeurs très élevées ou très faibles
  - Exemple : Le nombre d'Avogadro vaut environ  $6,02214129 \times 10^{23}$
  - Soit donc environ **602214129000000000000000,0**
  - Pour écrire cette partie entière en base 2, il faut 79 bits :  
11111110000110000101111110010110101010010011000111100000000000000000  
00000000000

# REPRÉSENTER UN NOMBRE RÉEL

- Notation scientifique :  $s m 2^n$ 
  - $s$  : signe (+ ou -)
  - $n$  : **exposant**, entier naturel
  - $m$  : **mantisse**, nombre réel compris entre 1 (inclus) et 2 (exclu)
- Exemple en 64-bits :
  - Signe : 1 bits
    - 0 pour le signe + et 1 pour le signe -
  - Exposant : 11 bits
    - Compris entre **-1022** et **1023**
    - Si l'exposant est  $n$ , on représente  **$n + 1023$**  (compris entre **1** et **2046**)
    - Les valeurs **0** et **2047** sont réservées à des valeurs exceptionnelles
  - Mantisse : 52 bits
    - toujours un seul chiffre avant la virgule
    - ce chiffre est toujours **1**
    - donc il n'est pas nécessaire de le représenter
    - on ne représente que les chiffres après la virgule





# REPRÉSENTER LE TEMPS

- *Comment représenter un instant  $t$  avec des 0 et des 1 ?*
- On appelle **timestamp** une valeur représentant une date et une heure
- Représentations classiques :
  - **POSIX** : Nombre de secondes depuis le 1<sup>er</sup> janvier 1970 00:00:00 UTC
    - Exemple : le 06/01/2014 à 18:00:00 est représenté par 1389027600
  - **DateTime** : Stockage « indépendant » des différents composants :
    - Année + Mois (17 bits)
    - Jour (5 bits)
    - Heure (5 bits)
    - Minute (6 bits)
    - Secondes (6 bits)

# LE BUG DE L'AN 2038

- La norme POSIX est très répandue
- Un timestamp est généralement représenté par un entier signé, donc sur un logiciel 32-bits, la valeur maximale est  $2^{31} - 1 = 2147483647 = \underline{01111111111111111111111111111111}_2$
- Ce nombre sera atteint  $2^{31} - 1$  secondes après le 1<sup>er</sup> janvier 1970 à minuit, soit donc le 19 janvier 2038 à 03:14:07.
- La seconde suivante, le timestamp « bouclera » et vaudra  $\underline{10000000000000000000000000000000}_2 = -2147483648 = -2^{31}$
- Pour un tel logiciel, nous serons donc  $2^{31}$  secondes avant le 1<sup>er</sup> janvier 1970 à minuit, soit donc le 13 décembre 1901 à 20:45:52

REPRÉSENTER DU TEXTE

# CARACTÈRES ET TEXTES

- Le type de données **caractère** englobe différents types de symboles :
  - Des lettres (majuscules ou minuscules, accentuées ou non, etc.)
  - Des chiffres (dans différentes langages)
  - Des signes de ponctuation
  - Des symboles de mise en page (saut de ligne, tabulation, etc.)
  - Des opérations spéciales (sonnerie, effacement, etc.)
- Ce type de données est généralement noté **char**.
- Un **texte** est constitué d'une succession de caractères
  - On parle de **chaîne de caractère**
  - Le type correspondant est généralement nommé **string**

# REPRÉSENTER UN CARACTÈRE

- *Comment représenter un caractère avec des 0 et des 1 ?*
- Idée :
  - On sait comment représenter des nombres entiers
  - On associe à chaque caractère un entier
  - On représente ce caractère avec l'écriture en base 2 de l'entier correspondant
- On appelle **encodage** une telle association entre un jeu de caractères et leurs « codes » entiers.
- Exemple :
  - Principe : « Chaque lettre est représentée par sa place dans l'alphabet »
  - Application : HELLO → 8 5 12 12 15

# LE CODE ASCII

- **ASCII** : American Standard Code for Information Interchange

- 128 caractères :

- 26 majuscules
- 26 minuscules
- 10 chiffres
- 32 symboles !«# \$%&' ( ) \*+, - . / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~
- 1 signe d'espace
- 33 symboles de mise en page ou de contrôle

- 8 bits par caractère

- 7 auraient suffi
- Le bit de point fort vaut toujours 0

■■■■■ Représenter du texte

PDF: fr en v · d · m	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
002	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

# LE CODE ASCII

- Exemple : Informatique en ASCII
  - Texte : HELLO WORLD !
  - Décimal : 72 101 108 108 111 32 119 111 114 108 100  
32 33
  - Binaire : 01001000 01100101 01101100 01101100  
01101111 00100000 01110111 01101111 01110010  
01101100 01100100 00100000 00100001
- Limitations :
  - Pas de caractères accentués, de cédilles, de œ, etc.
  - Uniquement l'alphabet latin

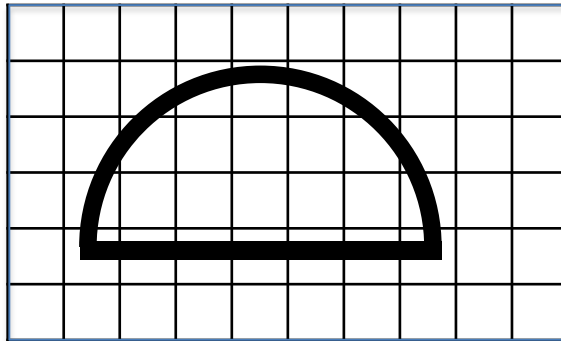
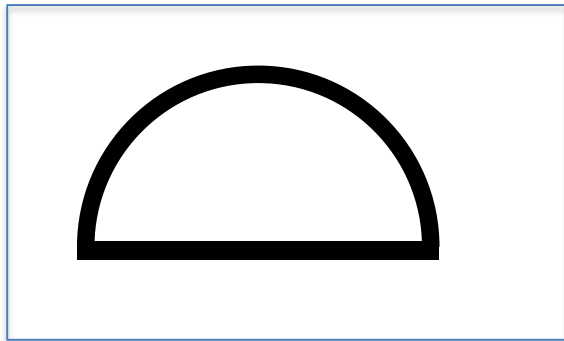
# VERS UN FORMAT UNIVERSEL

- Un format universel : l'**Unicode**
- Près de 245 000 caractères
- Plusieurs déclinaisons :
  - UTF-32 : chaque caractère se code sur 32 bits
  - UTF-8 : chaque caractère se code sur 8, 16, 32 ou 64 bits, selon sa fréquence
- L'UTF-8 a vocation à devenir le standard, mais ce n'est pas encore le cas

REPRÉSENTER DES IMAGES

# REPRÉSENTER UNE IMAGE EN NOIR ET BLANC

- Partons d'une image en noir et blanc :

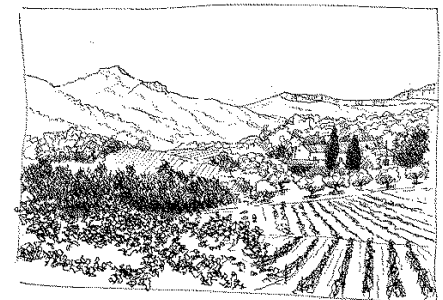
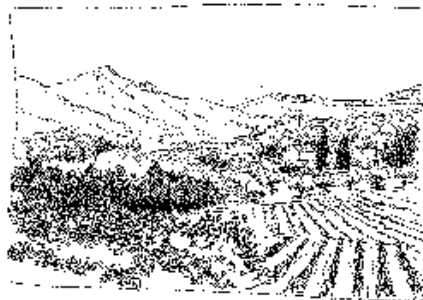


0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	1	1	0	0	0	1	1	0	0
0	1	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

- Pour représenter une telle image, la zone de travail est divisée par un quadrillage
  - Chaque case de ce quadrillage est appelée un **pixel** (picture element)
  - Plus ce quadrillage est fin, plus il y a de pixels
- Chaque pixel qui contient un trait est entièrement noirci
- On en déduit une représentation binaire de cette image
  - Les pixels blancs sont représentés par des 0
  - Les pixels noirs sont représentés par des 1

# REPRÉSENTER UNE IMAGE EN NOIR ET BLANC

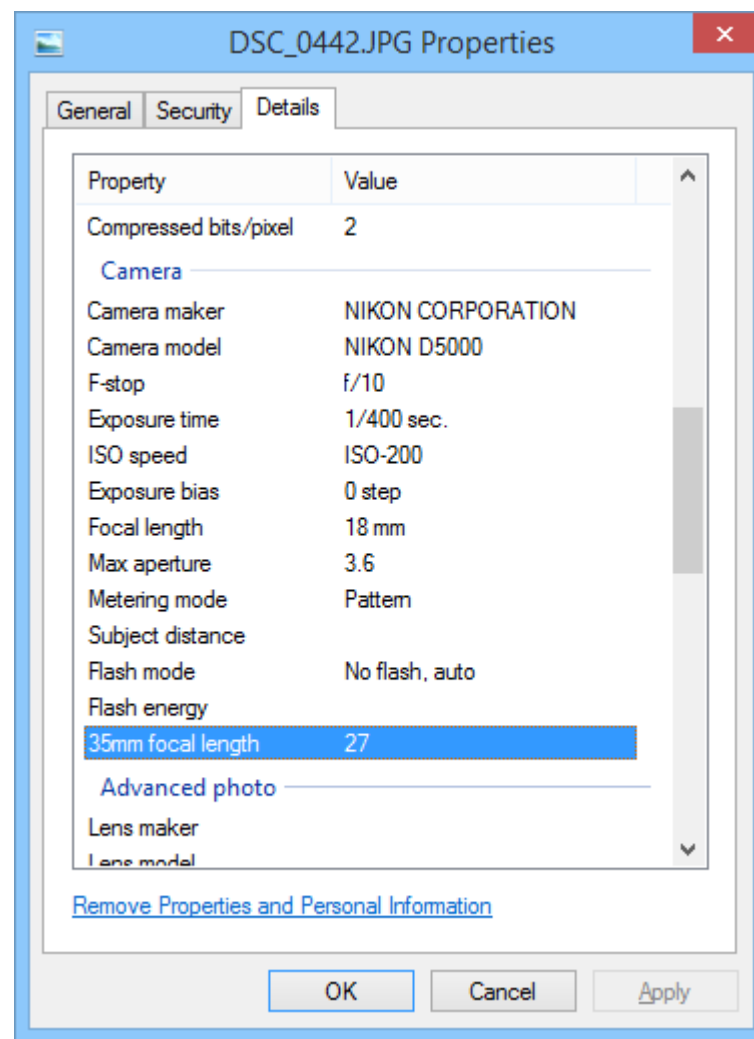
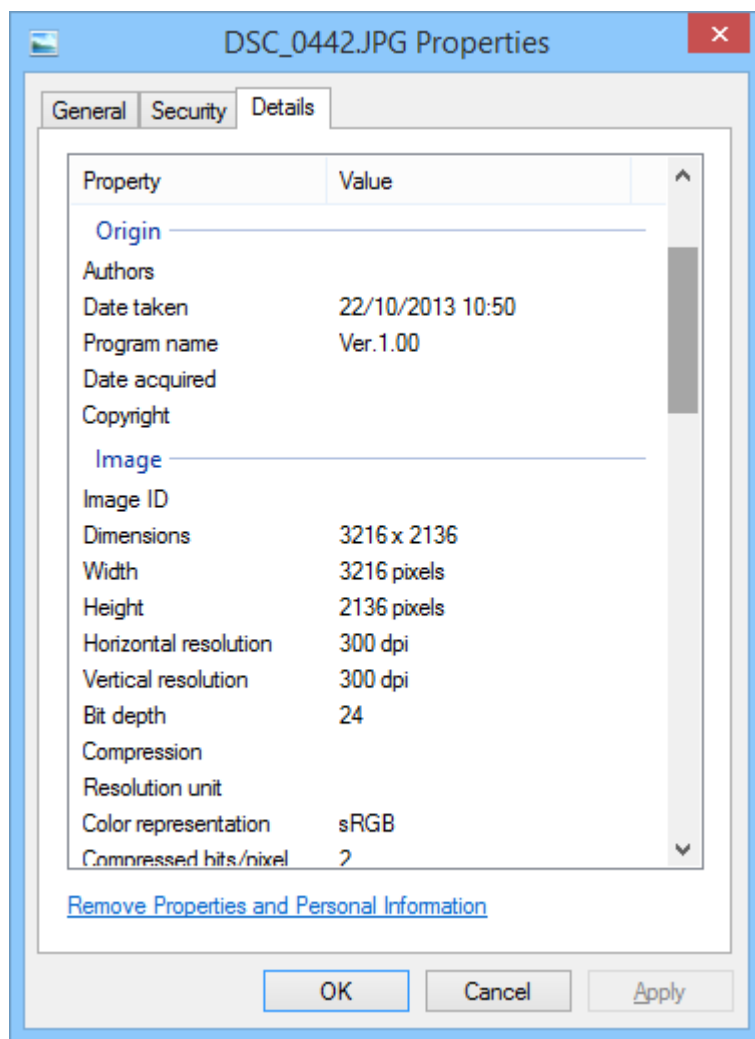
- Une image ainsi représentée en binaire est appelée une **bitmap**.
- Cette méthode est universelle :
  - Toute image en noir et blanc peut-être représentée de la sorte
- Cette méthode est approximative :
  - Il peut y avoir de la perte de données lors de la pixellisation
  - Pour limiter ces pertes, on augmente la résolution du quadrillage



# DONNER DU SENS À UNE SUITE DE 0 ET DE 1

- *Question : Comment savoir qu'une suite de 0 et de 1 correspondant à une image ?*
- Le **format** d'un fichier permet de donner du sens à son contenu
  - Il indique le type de données et la structure utilisée pour le stockage
  - Il est souvent identifié par une extension (ex : `.txt`, `.png`, etc.)
  - Il permet de savoir quel programme utiliser
- La plupart des formats de fichiers utilisent un **entête** pour regrouper au début du fichier les informations nécessaires à son utilisation
- On parle aussi de **métadonnées** pour parler de ces informations supplémentaires
  - Exemple : Une photo prise depuis un appareil photo numérique inclut des données EXIF (dimensions, type d'appareil, date et heure, focale, etc.)

# EXEMPLE DE DONNÉES EXIF





# LE FORMAT PBM : PORTABLE BITMAP

- Le format **Portable BitMap** a été créé sur ce modèle
- Extension `.pbm`
- Contenu
  - Les caractères `P1`, suivi d'un retour à la ligne ou d'un espace
  - La largeur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La hauteur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La liste des pixels, ligne par ligne, de haut en bas et de gauche à droite (les retours à la ligne et les espaces sont ignorés dans cette partie)
- Remarques
  - Aucune ligne ne doit dépasser 70 caractères
  - Les lignes commençant par `#` sont ignorées

# LE FORMAT PBM : PORTABLE BITMAP

- Exemple :

P1

10 6

0000000000

0011111000

0110001100

0100000100

0111111100

0000000000

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	1	1	0	0	0	1	1	0	0
0	1	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

# LE FORMAT PGM : PORTABLE GREYMAP

- Une **image en niveaux de gris** est constituée de différentes nuances allant du noir au blanc.
- Pour représenter une telle image
  - On choisit une valeur maximale  $v_{\max}$  (généralement 255)
  - Chaque pixel est représenté par un entier
    - 0 pour un pixel noir
    - $v_{\max}$  pour un pixel blanc
    - une valeur intermédiaire pour un pixel gris



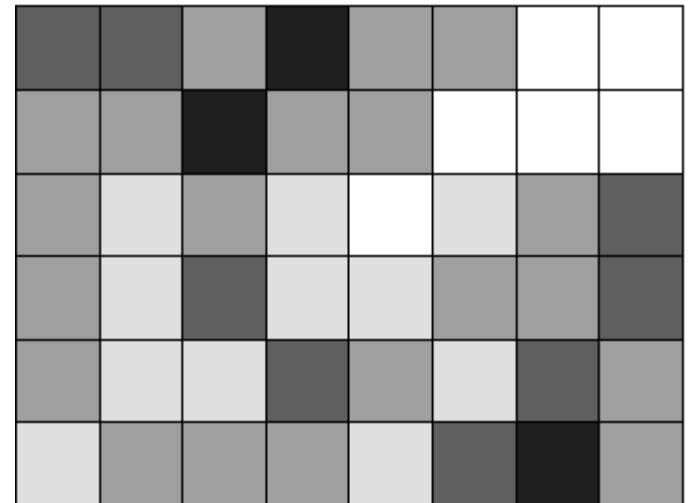
- Le format **Portable GreyMap** est basé sur ce principe
- Extension `.pgm`

# LE FORMAT PGM : PORTABLE GREYMAP

- Contenu
  - Les caractères P2, suivi d'un retour à la ligne ou d'un espace
  - La largeur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La hauteur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La valeur maximale, correspondant au blanc, suivie d'un retour à la ligne ou d'un espace
  - La liste des pixels, ligne par ligne, de haut en bas et de gauche à droite (séparés par des retours à la ligne ou des espaces)

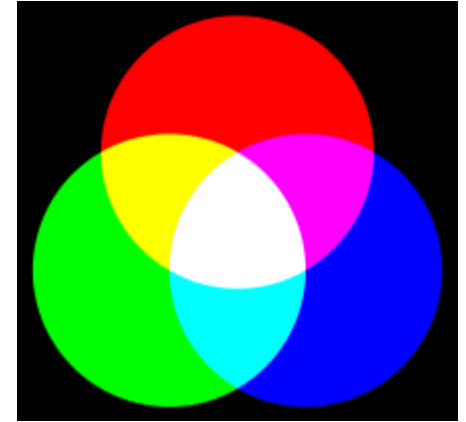
- Exemple

```
P2
8 6
255
 89  89 120  20 120 120 255 255
120 120  20 120 120 255 255 255
120 181 120 181 255 181 120  89
120 181  89 181 181 120 120  89
120 181 181  89 120 181  89 181
181 120 120 120 181  89  29 181
```



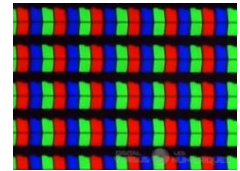
# UN PEU DE COULEUR

- La couleur en informatique
  - Trois composantes principales : rouge, vert, bleu
  - Synthèse « additive »
    - rouge + bleu = magenta
    - rouge + vert = jaune
    - vert + bleu = cyan
    - rouge + vert + bleu = blanc
    - aucune couleur = noir
- La couleur en imprimerie :
  - Trois composantes principales : cyan, jaune, magenta
  - Synthèse « soustractive »
    - cyan + jaune = vert
    - cyan + magenta = bleu (violet)
    - jaune + magenta = rouge (vermillon)
    - cyan + jaune + magenta = noir
    - aucune couleur = blanc



# UN PEU DE COULEUR

- Une couleur est la **somme de ses trois composantes**
  - Pour chaque pixel d'un écran couleur, il y a trois sources de lumières
  - Pour représenter une couleur, on utilise un triplet de valeurs



- Représentation classique : avec des entiers
  - Chaque composante est codée par un entier compris entre 0 et 255
  - Une couleur est représentée par un triplet d'entiers
  - Exemples : (0,0,0) (255,0,0) (0,255,255) (127,127,127) (45,125,255)
  - Remarque : Ces codes sont souvent notés au format hexadécimal  
#000000 #FF0000 #00FFFF #7F7F7F #2D7DFF

- Remarque :
  - Les couleurs peuvent varier d'un écran à l'autre
  - Des outils de calibrage permettent d'ajuster ce rendu

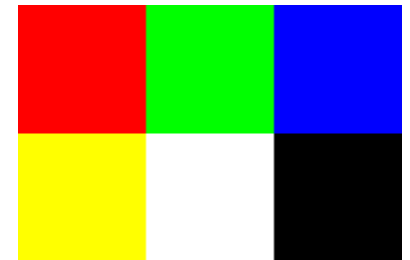


# LE FORMAT PPM : PORTABLE PIXMAP

- Le format **Portable PixMap** est basé sur ce principe
- Extension `.ppm`
- Contenu
  - Les caractères `P3`, suivi d'un retour à la ligne ou d'un espace
  - La largeur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La hauteur de l'image (en base 10), suivie d'un retour à la ligne ou d'un espace
  - La valeur maximale utilisée pour l'intensité des couleurs (généralement 255), suivie d'un retour à la ligne ou d'un espace
  - La liste des trois composantes (rouge, vert, bleu) pour chaque pixel, ligne par ligne, de haut en bas et de gauche à droite, séparées par des retour à la ligne ou des espaces

- Exemple

```
P3
3 2
255
255 0 0 0 255 0 0 0 255
255 255 0 255 255 255 0 0 0
```



# LA COMPOSANTE ALPHA

- Une quatrième composante
  - On utilise dans certains cas une quatrième composante : **alpha**
  - Cette composante alpha correspond à la « transparence » du pixel
  - Plus cette composante est élevée, plus « on voit à travers le pixel »



- Remarques :
  - Ces formats sont universels : on peut représenter tout type d'image
  - Ces formats sont peu efficaces :
    - Une image couleur de 1024x768 est représentée par 2 236 416 entiers
    - Même si cette image est entièrement blanche !
  - Il existe de nombreux autres formats d'image (JPEG, GIF, PNG, etc.)

# EXERCICES SUR LES IMAGES

- **Question 1.** Quel(s) type(s) de fichiers peut-on utiliser pour représenter
  - Un carré noir de 10 pixels de côté ?
  - Un carré gris de 10 pixels de côté ?
  - Un carré bleu de 10 pixels de côté ?
- **Question 2.** Quelle est la taille minimale (en octets) du fichier obtenu dans chaque cas ?

REPRÉSENTER DU SON

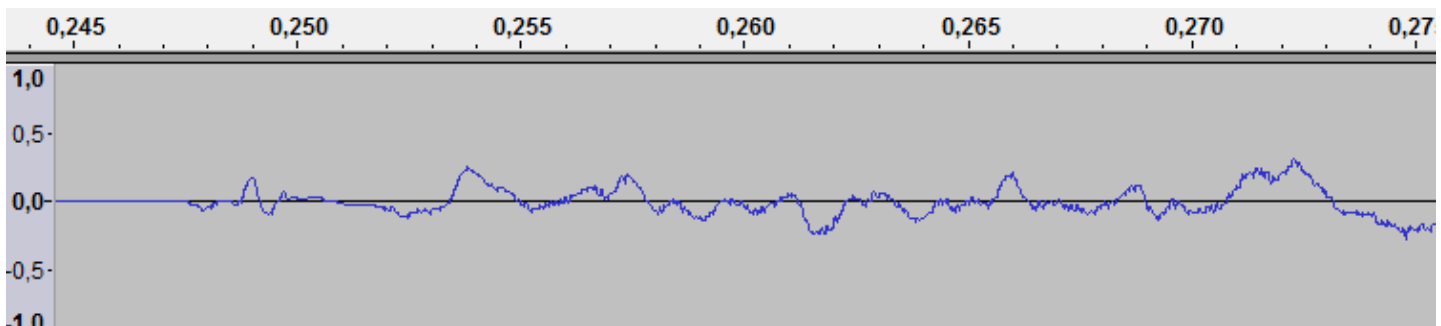
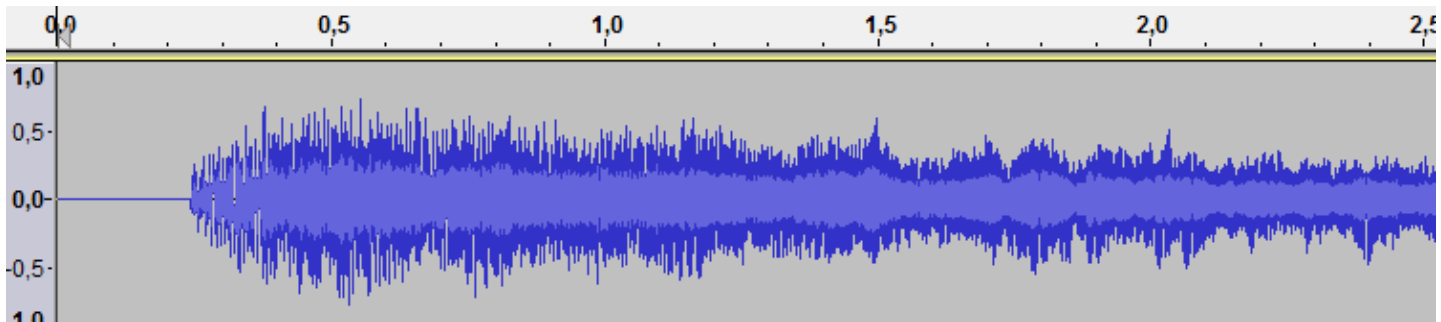
# REPRÉSENTER LE SON

The image shows two systems of musical notation for a piece in G major, 4/4 time. The first system consists of four measures. The right hand (treble clef) plays a melody of quarter notes: G4, A4, B4, G4. The left hand (bass clef) plays a bass line of quarter notes: G2, B1, D2, G2. Chords are indicated above the right hand: G, Gmaj7, C, G, C, G. Dynamics include a forte (f) marking in the first measure and a breath mark (v) in the second. The second system also consists of four measures. The right hand melody is: F4, G4, A4, G4. The left hand bass line is: G2, B1, D2, G2. Chords are indicated: F, D, G, Gmaj7, C, G. Dynamics include a mezzo-forte (mf) marking in the third measure and a breath mark (v) in the first measure.

- *Peut-on faire lire une telle partition à un ordinateur ?*
- *Peut-on représenter n'importe quel son avec une telle partition ?*

# REPRÉSENTER LE SON

- Un peu de physique
  - Un son est une onde mécanique qui se propage dans un milieu
  - Un son est une variation de la pression de l'air au cours du temps
  - On parle également de signal acoustique



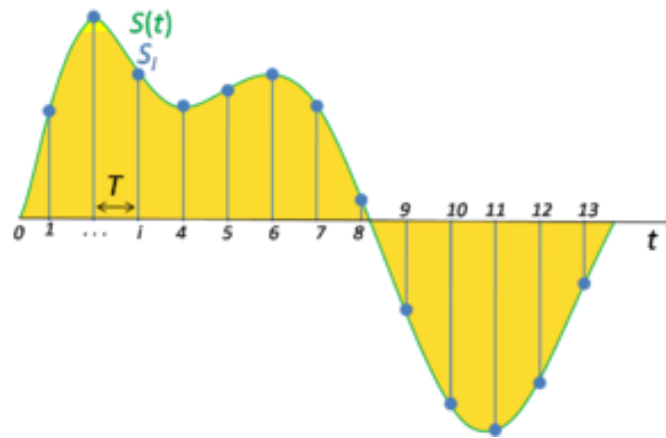
# DE L'AIR ET DE L'ÉLECTRICITÉ



- Capter le son
  - Un microphone permet de passer d'un signal acoustique à un signal électrique
  - Une membrane vibre sous l'effet de la pression acoustique, et un dispositif convertit ces oscillations en signal électrique
- Emettre du son :
  - Une enceinte permet de passer d'un signal électrique à un signal acoustique
  - Une membrane est mise en mouvement par un signal électrique, et fait ainsi varier la pression de l'air
  - Une enceinte contient généralement plusieurs haut-parleurs (pour couvrir l'ensemble du spectre audible)
- Question :
  - *Est-ce suffisant pour représenter un son ?*
  - *Comment stocker un son ?*



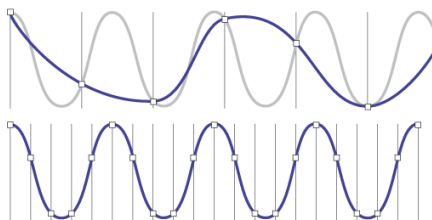
# ECHANTILLONNAGE



- L'**échantillonnage** permet de passer d'une signal continu (avec une infinie de valeurs) à un ensemble de valeurs discrètes
- Principe : On mesure la valeur du signal à intervalle réguliers
- Deux paramètres clés :
  - Fréquence d'échantillonnage
  - Quantification d'échantillonnage

# ECHANTILLONNAGE

- La **fréquence d'échantillonnage** est le nombre de valeurs relevées par seconde.
- Cette fréquence doit être assez élevée pour permettre de reconstituer sans ambiguïté le signal source.

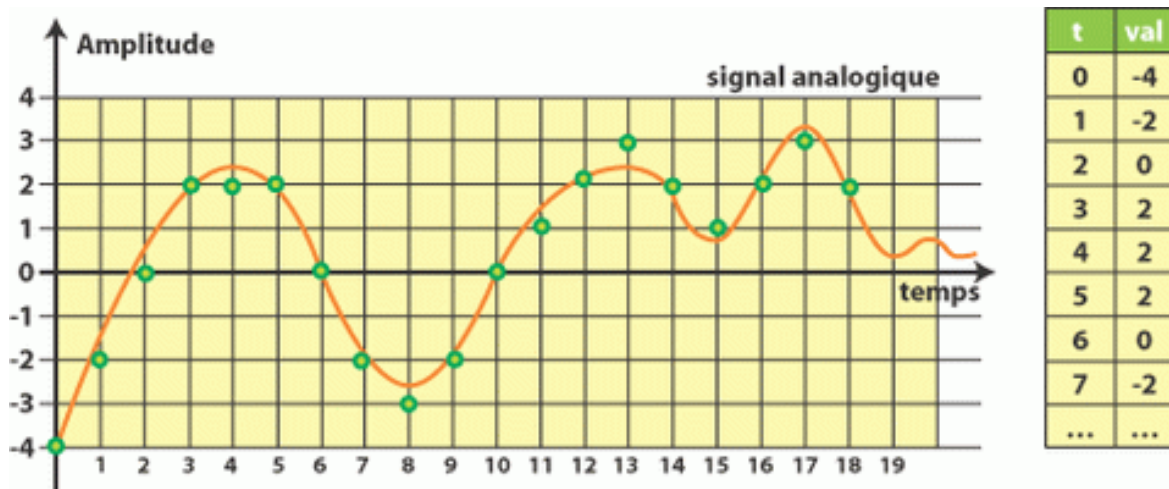


- Fréquence d'échantillonnage classique : 44 000 Hz
  - Fréquence maximale perceptible par l'oreille humaine : 22 000 Hz
  - Théorème d'échantillonnage de Nyquist-Shannon

*La représentation discrète d'un signal par des échantillons régulièrement espacés exige une fréquence d'échantillonnage supérieure au double de la fréquence maximale présente dans ce signal.*

# REPRÉSENTER LE SON

- Il existe théoriquement une infinie de valeurs possibles pour les valeurs mesurées à intervalles réguliers
- En pratique, seule une liste finie de valeurs sont autorisées
- Cette **quantification** est caractérisée par le nombre de bits utilisés pour décrire chaque échantillon (8-bits, 16-bits, etc.)



# REPRÉSENTER LE SON

- Le son peut être représenté via :
  - La fréquence d'échantillonnage
  - La quantification (et son codage)
  - La liste des valeurs échantillonnées au cours du temps
- De nombreux formats audio existent :
  - Plusieurs façons de stocker les données échantillonnées
  - Avec ou sans compression
  - Avec ou sans perte
  - Format public ou secret

# EXERCICES

# EXERCICES

- **Question 1.** Quelle précision perd-on si on divise un nombre à virgule par deux avant de le remultiplier par deux ?
- **Question 2.** Ecrivez une phrase courte en binaire, et donnez la à votre voisin pour qu'il la traduise.
- **Question 3.** Montrez que dans la représentation binaire d'un entier signé, le bit de poids fort vaut 1 pour les entiers strictement négatifs, et 0 pour les entiers positifs ou nuls.

# CONVERSIONS AUTOMATIQUES

- **Question 4.** Imaginez un algorithme
  - prenant en argument un entier positif (ou nul) écrit en base 10
  - renvoyant la représentation binaire de cet entier.
- **Question 5.** Imaginez un algorithme
  - prenant en argument un entier positif (ou nul) écrit en binaire
  - renvoyant la représentation en base 10 de cet entier.
- **Question 6.** Imaginez un algorithme
  - prenant en argument un entier signé écrit en base 10
  - renvoyant la représentation en complément à 2 sur 8 bits de cet entier.
- **Question 7.** Imaginez un algorithme
  - prenant en argument la représentation en complément à 2 sur 8 bits d'un entier signé
  - renvoyant la représentation en base 10 de cet entier

# PROCHAINE SÉANCE

Vendredi 23 janvier

[TD] CONVERSIONS ET ENCODAGES

There are only 10 types  
of people in the world:  
Those who understand binary  
and those who don't.