

Comment sont stockées vos données dans un ordinateur

Yann Strozecki
yann.strozecki@uvsq.fr

Université de Versailles St-Quentin-en-Yvelines

Année universitaire 2020-2021

Programme

Sur 8 cours et 12 tds. Chargé de TD : Loric Duhazé.

Notation prévue : 4 QCMs en TD, 2 contrôles en cours.

- ▶ Utiliser Tkinter (transverse)
- ▶ Représenter les nombres, le texte et les images.
- ▶ Compresser des données.
- ▶ Protéger ses données contre les erreurs et les pirates.
- ▶ Modéliser des systèmes complexes par des automates.
- ▶ Programmer des robots (si retour en présentiel).

La mémoire de l'ordinateur

Dans un ordinateur, la mémoire est séparée du calcul (processeur).
Qu'est-ce qui sert de mémoire à un ordinateur ?

La mémoire de l'ordinateur

Dans un ordinateur, la mémoire est séparée du calcul (processeur).
Qu'est-ce qui sert de mémoire à un ordinateur ?

Qu'y a-t-il dans la mémoire d'un ordinateur ?

La mémoire de l'ordinateur

Dans un ordinateur, la mémoire est séparée du calcul (processeur).
Qu'est-ce qui sert de mémoire à un ordinateur ?

Qu'y a-t-il dans la mémoire d'un ordinateur ?

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

La mémoire de l'ordinateur

Dans un ordinateur, la mémoire est séparée du calcul (processeur).
Qu'est-ce qui sert de mémoire à un ordinateur ?

Qu'y a-t-il dans la mémoire d'un ordinateur ?

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

On va expliquer comment représenter toute information de votre ordinateur comme suite de 0 et de 1.

Où l'on apprend à compter

Notre système de numération compte 10 symboles (chiffres), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. On fabrique des mots avec ces symboles, ce sont les **nombre**s. C'est le système **décimal**.

Où l'on apprend à compter

Notre système de numération compte 10 symboles (chiffres), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. On fabrique des mots avec ces symboles, ce sont les **nombre**s. C'est le système **décimal**.

D'autres systèmes existent, les chiffres romains par exemple : *I, V, X, L, C, D, M*. Ainsi *CDLXXIV* signifie 474.

Où l'on apprend à compter

Notre système de numération compte 10 symboles (chiffres), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. On fabrique des mots avec ces symboles, ce sont les **nombre**s. C'est le système **décimal**.

D'autres systèmes existent, les chiffres romains par exemple : *I, V, X, L, C, D, M*. Ainsi *CDLXXIV* signifie 474.

L'homme des cavernes sait compter : pourtant les chiffres et même l'écriture ne sont pas inventés.

Où l'on apprend à compter

Notre système de numération compte 10 symboles (chiffres), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. On fabrique des mots avec ces symboles, ce sont les **nombre**s. C'est le système **décimal**.




D'autres systèmes existent, les chiffres romains par exemple : *I, V, X, L, C, D, M*. Ainsi *CDLXXIV* signifie 474.

L'homme des cavernes sait compter : pourtant les chiffres et même l'écriture ne sont pas inventés.

On utilise des *cailloux*, on trace des batons, on compte sur ses doigts. C'est un système **unaire**. Problème, c'est **encombrant**!!!





Des nombres plus grands plus petits

Une solution égyptienne :

1	
10	
100	
1000	

Des nombres plus grands plus petits





Une solution égyptienne :

1	
10	
100	
1000	

Système **hybride**, on ajoute les valeurs des symboles pour obtenir le nombre représenté. La position des chiffres n'est pas importante. Le système est plus efficace que le unaire.

Des nombres plus grands plus petits

Une solution égyptienne :

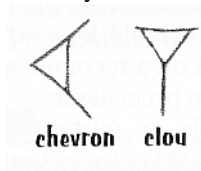
1	
10	
100	
1000	

Système **hybride**, on ajoute les valeurs des symboles pour obtenir le nombre représenté. La position des chiffres n'est pas importante. Le système est plus efficace que le unaire.

Combien de symboles pour représenter 474 ?

Encore plus grands et encore plus petits

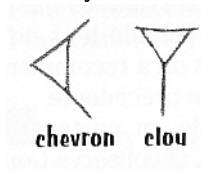
Les babyloniens avait un vrai système à base 60 avec seulement deux symboles :



Le chevron pour 1 et le clou pour 10. Leur position est **importante** ainsi que les espaces de séparation : c'est un système **positionnel**.

Encore plus grands et encore plus petits

Les babyloniens avait un vrai système à base 60 avec seulement deux symboles :

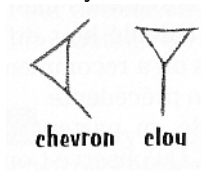


Le chevron pour 1 et le clou pour 10. Leur position est **importante** ainsi que les espaces de séparation : c'est un système **positionnel**.

Représenter 474 dans ce système.

Encore plus grands et encore plus petits

Les babyloniens avait un vrai système à base 60 avec seulement deux symboles :



Le chevron pour 1 et le clou pour 10. Leur position est **importante** ainsi que les espaces de séparation : c'est un système **positionnel**.

Représenter 474 dans ce système.

On utilise encore la base 60 pour les degrés et les heures, pratique car 60 est très divisible.

C'est la base

Un nombre en *base* b , s'écrit

$$a_k a_{k-1} \dots a_1 a_0$$

où $a_i \in [0, b - 1]$, i.e. les a_i sont des chiffres entre 0 et $b - 1$.

Par exemple en base 5, on peut écrire le nombre 2304.

C'est la base

Un nombre en *base* b , s'écrit

$$a_k a_{k-1} \dots a_1 a_0$$

où $a_i \in [0, b - 1]$, i.e. les a_i sont des chiffres entre 0 et $b - 1$.

Par exemple en base 5, on peut écrire le nombre 2304.

Pour préciser qu'on est en base b , on note le nombre

$$(a_k a_{k-1} \dots a_1 a_0)_b.$$

Par exemple $(2304)_5$ et $(329)_{10}$ sont deux manière de noter le même nombre.

Valeur d'un nombre

Si un nombre s'écrit $(a_k a_{k-1} \dots a_1 a_0)_b$, alors sa valeur est

$$a_k * b^k + a_{k-1} * b^{k-1} \dots + a_1 * b^1 + a_0 * b^0$$

Sur l'exemple précédant : $(2304)_5 = 2 * 5^3 + 3 * 5^2 + 4 * 5^0 = 329$.

Calculer la valeur de $(1212)_3$.

... avec des 0 et des 1

Le plus simple : le **système binaire** avec deux symboles, 0 et 1.
Dans un ordinateur courant/absence de courant (processeur),
condensateur chargé ou non (RAM) ou spin positif/spin négatif
(disque dur) ...

Et dans un **cerveau** ?

... avec des 0 et des 1

Le plus simple : le **système binaire** avec deux symboles, 0 et 1.
Dans un ordinateur courant/absence de courant (processeur),
condensateur chargé ou non (RAM) ou spin positif/spin négatif
(disque dur) ...

Et dans un **cerveau** ?

Apprenons à compter (retour en maternelle) : 0, 1, 10, 11, 100,
101 ...

... avec des 0 et des 1

Le plus simple : le **système binaire** avec deux symboles, 0 et 1.
Dans un ordinateur courant/absence de courant (processeur),
condensateur chargé ou non (RAM) ou spin positif/spin négatif
(disque dur) ...

Et dans un **cerveau** ?

Apprenons à compter (retour en maternelle) : 0, 1, 10, 11, 100,
101 ...

Calculer combien vaut le nombre $(1001101)_2$.

... avec des 0 et des 1

Le plus simple : le **système binaire** avec deux symboles, 0 et 1.
Dans un ordinateur courant/absence de courant (processeur),
condensateur chargé ou non (RAM) ou spin positif/spin négatif
(disque dur) ...

Et dans un **cerveau** ?

Apprenons à compter (retour en maternelle) : 0, 1, 10, 11, 100,
101 ...

Calculer combien vaut le nombre $(1001101)_2$.

Comment fait-on pour transformer un nombre en base 10 en un
nombre en base 2 ?

Changement de base

Observation :

- ▶ La division entière d'un nombre par 2 donne un nombre qui a un symbole de moins en base 2.
- ▶ Ce symbole est donné par le reste par la division entière.

Changement de base

Observation :

- ▶ La division entière d'un nombre par 2 donne un nombre qui a un symbole de moins en base 2.
- ▶ Ce symbole est donné par le reste par la division entière.

Justification : $(a_k a_{k-1} \dots a_1 a_0)_2$ vaut $a_k * 2^k + \dots + a_1 * 2 + a_0$.
De plus $a_k * 2^k + \dots + a_1 * 2 + a_0 = 2(a_k * 2^{k-1} + \dots + a_1) + a_0$.

Changement de base

Observation :

- ▶ La division entière d'un nombre par 2 donne un nombre qui a un symbole de moins en base 2.
- ▶ Ce symbole est donné par le reste par la division entière.

Justification : $(a_k a_{k-1} \dots a_1 a_0)_2$ vaut $a_k * 2^k + \dots + a_1 * 2 + a_0$.
De plus $a_k * 2^k + \dots + a_1 * 2 + a_0 = 2(a_k * 2^{k-1} + \dots + a_1) + a_0$.

Algorithme : Tant que le nombre est différent de 0 le diviser par 2 et récupérer le reste. La suite des restes écrits de droite à gauche donne l'écriture en base 2.

Changement de base

Observation :

- ▶ La division entière d'un nombre par 2 donne un nombre qui a un symbole de moins en base 2.
- ▶ Ce symbole est donné par le reste par la division entière.

Justification : $(a_k a_{k-1} \dots a_1 a_0)_2$ vaut $a_k * 2^k + \dots + a_1 * 2 + a_0$.
De plus $a_k * 2^k + \dots + a_1 * 2 + a_0 = 2(a_k * 2^{k-1} + \dots + a_1) + a_0$.

Algorithme : Tant que le nombre est différent de 0 le diviser par 2 et récupérer le reste. La suite des restes écrits de droite à gauche donne l'écriture en base 2.

Calculer la représentation en base 2 de 79.

Représentation informatique

Il existe deux autres bases classiques en informatique :

- ▶ l'**octal**, c'est à dire la base 8 (plus très utilisée),
- ▶ l'**hexadécimal** c'est à dire la base 16 qui utilise les symboles de 0 à 9 puis $A(10)$, $B(11)$, $C(12)$, $D(13)$, $E(14)$, $F(15)$.

Représentation informatique

Il existe deux autres bases classiques en informatique :

- ▶ l'**octal**, c'est à dire la base 8 (plus très utilisée),
- ▶ l'**hexadécimal** c'est à dire la base 16 qui utilise les symboles de 0 à 9 puis $A(10)$, $B(11)$, $C(12)$, $D(13)$, $E(14)$, $F(15)$.

Un chiffre en octal peut se voir comme un nombre de trois chiffres en binaire.

Par exemple $(5)_8 = (101)_2$.

De même $(B)_{16} = (1011)_2$.

Représentation informatique

Il existe deux autres bases classiques en informatique :

- ▶ l'**octal**, c'est à dire la base 8 (plus très utilisée),
- ▶ l'**hexadécimal** c'est à dire la base 16 qui utilise les symboles de 0 à 9 puis $A(10), B(11), C(12), D(13), E(14), F(15)$.

Un chiffre en octal peut se voir comme un nombre de trois chiffres en binaire.

Par exemple $(5)_8 = (101)_2$.

De même $(B)_{16} = (1011)_2$.

On a un entier sur 16 bits en mémoire : $\underbrace{1001}_9 \underbrace{1101}_D \underbrace{0100}_4 \underbrace{0101}_5$.

Attention, l'ordre de lecture a une importance (endiannes) !

Représenter des entiers positifs

Un entier sur votre ordinateur peut être codé sur 32 bits : il a une valeur entre 0 et 4294967295.

En C, cela correspond au type **unsigned int**.

Représenter des entiers positifs

Un entier sur votre ordinateur peut être codé sur 32 bits : il a une valeur entre 0 et 4294967295.

En C, cela correspond au type **unsigned int**.

Que se passe-t-il si vous faites $4294967295 + 1$ en C ?

Représenter des entiers positifs

Un entier sur votre ordinateur peut être codé sur 32 bits : il a une valeur entre 0 et 4294967295.

En C, cela correspond au type **unsigned int**.

Que se passe-t-il si vous faites $4294967295 + 1$ en C ?

0

On utilise une **représentation modulaire**.

Représenter des entiers positifs

Un entier sur votre ordinateur peut être codé sur 32 bits : il a une valeur entre 0 et 4294967295.

En C, cela correspond au type **unsigned int**.

Que se passe-t-il si vous faites $4294967295 + 1$ en C ?

0

On utilise une [représentation modulaire](#).

Les entiers sur 32 bits ne sont pas forcément suffisant, que faire ?

Représenter des entiers positifs

Un entier sur votre ordinateur peut être codé sur 32 bits : il a une valeur entre 0 et 4294967295.

En C, cela correspond au type **unsigned int**.

Que se passe-t-il si vous faites $4294967295 + 1$ en C ?

0

On utilise une [représentation modulaire](#).

Les entiers sur 32 bits ne sont pas forcément suffisant, que faire ?

Les architectures modernes supportent des entiers sur 64 bits qui vont de 0 à 18446744073709551615 (et même des entiers sur 128 bits, 256 bits et 512 bits).

Cela correspond aux types **unsigned long int** et **unsigned long long int** en C.

Taille arbitraire

En Python, les entiers ont une taille arbitraire : ils peuvent être aussi grand qu'on veut. Dans d'autres langages, on a aussi parfois besoin de manipuler des nombres plus grand que 2^{64} .

Taille arbitraire

En Python, les entiers ont une taille arbitraire : ils peuvent être aussi grand qu'on veut. Dans d'autres langages, on a aussi parfois besoin de manipuler des nombres plus grand que 2^{64} .

On peut représenter un nombre plus petit que 2^{128} comme $n = n_1 + 2^{64}n_2$ avec n_1 et n_2 des entiers sur 64 bits. Autrement dit l'entier n s'écrit n_2n_1 et peut se représenter par les deux entiers n_1 et n_2 .

Taille arbitraire

En Python, les entiers ont une taille arbitraire : ils peuvent être aussi grand qu'on veut. Dans d'autres langages, on a aussi parfois besoin de manipuler des nombres plus grand que 2^{64} .

On peut représenter un nombre plus petit que 2^{128} comme $n = n_1 + 2^{64}n_2$ avec n_1 et n_2 des entiers sur 64 bits. Autrement dit l'entier n s'écrit n_2n_1 et peut se représenter par les deux entiers n_1 et n_2 .

En Python, un entier est représenté en interne par une liste d'entiers sur 64 bits.

Limites

Défauts dus à la représentation des entiers de Python.

- ▶ Les entiers nécessitent quatre fois plus de place pour être stockés.
- ▶ Il faut implémenter les opérations de base (plus lent que les opérations processeurs).

Comment faire l'opération $l = n + m$?

Limites

Défauts dus à la représentation des entiers de Python.

- ▶ Les entiers nécessitent quatre fois plus de place pour être stockés.
- ▶ Il faut implémenter les opérations de base (plus lent que les opérations processeurs).

Comment faire l'opération $l = n + m$?

$$l_1 = n_1 + m_1$$

$$\text{Si } l_1 < 2^{64}$$

$$\text{Alors } l_2 = n_2 + m_2$$

Sinon

$$l_1 = l_1 - 2^{64}$$

$$l_2 = n_2 + m_2 + 1$$

Représentation négative

Comment représenter un nombre négatif :

- **Bit de signe** : le nombre -19 s'écrit sur 8 bits 10010011
alors que 19 s'écrit 00100011.

Représentation négative

Comment représenter un nombre négatif :

- **Bit de signe** : le nombre -19 s'écrit sur 8 bits 10010011
alors que 19 s'écrit 00100011.

Problème : 00000000 et 10000000 représentent 0 !

Représentation négative

Comment représenter un nombre négatif :

- ▶ **Bit de signe** : le nombre -19 s'écrit sur 8 bits 10010011
alors que 19 s'écrit 00100011.

Problème : 00000000 et 10000000 représentent 0 !

- ▶ Complément à deux : le nombre a sur b bits s'écrit $2^{b-1} - a$.

Représentation négative

Comment représenter un nombre négatif :

- ▶ **Bit de signe** : le nombre -19 s'écrit sur 8 bits 10010011 alors que 19 s'écrit 00100011.

Problème : 00000000 et 10000000 représentent 0 !

- ▶ Complément à deux : le nombre a sur b bits s'écrit $2^{b-1} - a$.
 - ▶ Le nombre 19 s'écrit comme $2^7 - 19 = 45$ en binaire i.e. 00101101
 - ▶ Le nombre -19 comme $2^7 - (-19) = 83$ i.e. 10010011

La deuxième solution est utilisée dans les processeurs modernes car elle est pratique pour les opérations arithmétiques.

Résumé

En informatique un des problèmes fondamentaux est la représentation des idées et du réel par des 0 et des 1.

C'est ce qu'on appelle l'**encodage** des données. Cet encodage est une convention qui doit être partagée par tous les utilisateurs de la donnée (par exemple little-endian vs big-endian).

Les encodages sont plus ou moins efficaces :

- ▶ pour la taille de la représentation (unaire vs binaire)
- ▶ pour la facilité à faire des opérations dessus
- ▶ pour leur résistance aux erreurs (on verra ça plus tard)

Les images bitmap

Une image est représentée comme une **matrice de pixels** et un **entête** (header) avec des métadonnées.

Un pixel est donné par ses coordonnées (implicites) et sa couleur :

- ▶ noir ou blanc sur 1 bit.
- ▶ en niveau de gris avec 16 ou 256 niveaux d'intensité (4 ou 8 bits).
- ▶ en couleur sur 24 bits.

Les images bitmap

Une image est représentée comme une **matrice de pixels** et un **entête** (header) avec des métadonnées.

Un pixel est donné par ses coordonnées (implicites) et sa couleur :

- ▶ noir ou blanc sur 1 bit.
- ▶ en niveau de gris avec 16 ou 256 niveaux d'intensité (4 ou 8 bits).
- ▶ en couleur sur 24 bits.

Plusieurs manières de rendre la couleur :

- ▶ **Image à palette** ou indexée (GIF ou PNG).

Les images bitmap

Une image est représentée comme une **matrice de pixels** et un **entête** (header) avec des métadonnées.

Un pixel est donné par ses coordonnées (implicites) et sa couleur :

- ▶ noir ou blanc sur 1 bit.
- ▶ en niveau de gris avec 16 ou 256 niveaux d'intensité (4 ou 8 bits).
- ▶ en couleur sur 24 bits.

Plusieurs manières de rendre la couleur :

- ▶ **Image à palette** ou indexée (GIF ou PNG).
- ▶ **Couleurs vraies**, une information de couleur est composée des trois canaux **R**ed, **G**reen et **B**lue (BMP).

Les images bitmap

Une image est représentée comme une **matrice de pixels** et un **entête** (header) avec des métadonnées.

Un pixel est donné par ses coordonnées (implicites) et sa couleur :

- ▶ noir ou blanc sur 1 bit.
- ▶ en niveau de gris avec 16 ou 256 niveaux d'intensité (4 ou 8 bits).
- ▶ en couleur sur 24 bits.

Plusieurs manières de rendre la couleur :

- ▶ **Image à palette** ou indexée (GIF ou PNG).
- ▶ **Couleurs vraies**, une information de couleur est composée des trois canaux **R**ed, **G**reen et **B**lue (BMP).
- ▶ Il y a parfois un quatrième canal pour la transparence (alpha).

Caractéristiques d'une image

Définition (d'une image ou d'un écran) : c'est le nombre de pixels dans l'image. Généralement donné comme un produit qui donne la largeur et la longueur de l'image : 640×480 .

Par extension, la définition d'un écran est la taille de la plus grande image qu'il peut afficher.

Résolution (d'un écran) : Nombre de Pixels par unité de surface, généralement de pouces (inches) carrés. Unité le DPI, par exemple un iphone a un écran 300 DPI (450 DPI pour les modèles les plus haut de gamme).

Caractéristiques d'une image

Définition (d'une image ou d'un écran) : c'est le nombre de pixels dans l'image. Généralement donné comme un produit qui donne la largeur et la longueur de l'image : 640×480 .

Par extension, la définition d'un écran est la taille de la plus grande image qu'il peut afficher.

Résolution (d'un écran) : Nombre de Pixels par unité de surface, généralement de pouces (inches) carrés. Unité le DPI, par exemple un iphone a un écran 300 DPI (450 DPI pour les modèles les plus haut de gamme).



Calculer la résolution de votre écran d'ordinateur, à partir de sa définition et de ses dimensions.

Taille d'une image

La **taille de l'image** en mémoire dépend de la définition et de l'encodage d'un pixel. On néglige les métadonnées qui sont de petites tailles et ne dépendent pas de la définition.

- ▶ Une icône noir et blanc : $32 \times 32 = 1024$ bits ou 128 octets
- ▶ Une image en couleur vraies et 1080p :
 $1920 \times 1080 \times 24 = 49766400$ bits ou 6220800 octets
- ▶ Une image en couleurs vraies et canal de transparence en 4K :
 $4096 \times 2160 \times 32 = 283115520$ bits ou 35389440 octets.
- ▶ Une image prise par un capteur 24MP ?
- ▶ Un film en 4K, de 1h, en 24 images par seconde ?

Opération sur une image bitmap

L'image est donnée par la matrice M .

- **Interpolation** : On agrandit une image (zoom), ici d'un facteur 2 dans chaque dimension.

$$\begin{aligned}M_{zoom}[2i][2j] &= M_{zoom}[2i+1][2j] = M_{zoom}[2i][2j+1] = \\M_{zoom}[2i+1][2j+1] &= M[i][j]\end{aligned}$$

Opération sur une image bitmap

L'image est donnée par la matrice M .

- **Interpolation** : On agrandit une image (zoom), ici d'un facteur 2 dans chaque dimension.

$$M_{zoom}[2i][2j] = M_{zoom}[2i+1][2j] = M_{zoom}[2i][2j+1] = M_{zoom}[2i+1][2j+1] = M[i][j]$$

- **Flou** : Moyenne des pixels proches, utilisé pour cacher les défauts de l'interpolation ou d'une image bruitée.

$$M[i][j] = \frac{M[i-1][j-1] + M[i-1][j] + M[i][j-1] + M[i][j] + \dots}{9}$$

Filtres spéciaux pour conserver les arêtes.

Opération sur une image bitmap

L'image est donnée par la matrice M .

- **Interpolation** : On agrandit une image (zoom), ici d'un facteur 2 dans chaque dimension.

$$M_{zoom}[2i][2j] = M_{zoom}[2i+1][2j] = M_{zoom}[2i][2j+1] = M_{zoom}[2i+1][2j+1] = M[i][j]$$

- **Flou** : Moyenne des pixels proches, utilisé pour cacher les défauts de l'interpolation ou d'une image bruitée.

$$M[i][j] = \frac{M[i-1][j-1] + M[i-1][j] + M[i][j-1] + M[i][j] + \dots}{9}$$

Filtres spéciaux pour conserver les arêtes.

- **Changement de base** : les couleurs sont codées dans un espace à 3 dimensions RGB. On peut passer au codage TSL ou **T**einte, **S**aturation, **L**uminance.

On a aussi Y'UV pour la vidéo. Système plus proche de la perception humaine, bon pour la création et la compression.

Représentation d'une image bitmap en Python

Il existe plusieurs bibliothèques pour manipuler les images qui utilisent différents types pour représenter l'image. Nous utiliserons en TD une version simplifiée :

Un tableau à deux dimensions M (largeur \times hauteur) où $M[i][j]$ est un triplet de trois entiers dans $[0, 255]$ qui représente l'intensité des canaux R, G et B.

Le pixel $M[0][0]$ est le pixel en haut à gauche de l'image.

Représentation d'une image bitmap en Python

Il existe plusieurs bibliothèques pour manipuler les images qui utilisent différents types pour représenter l'image. Nous utiliserons en TD une version simplifiée :

Un tableau à deux dimensions M (largeur \times hauteur) où $M[i][j]$ est un triplet de trois entiers dans $[0, 255]$ qui représente l'intensité des canaux R, G et B.

Le pixel $M[0][0]$ est le pixel en haut à gauche de l'image.

Dessiner l'image suivante :

$[[(0, 0, 255), (255, 0, 0)], [(0, 0, 0), (255, 255, 255)]]$

Images vectorielle

Une **image vectorielle** est un ensemble de fonctions mathématiques décrivant des éléments de l'image :

- ▶ point
- ▶ ligne droites
- ▶ polygnes
- ▶ ellipses
- ▶ couleur remplissant une zone, dégradé ...
- ▶ composition additive, soustractive ...

Le format typique est le svg qu'on peut manipuler avec inkscape ou illustrator. Il n'y a pas de notion de définition : on peut **zoomer et dézoomer** sans perte . Utile pour les polices, les icônes, les logos.

Images vectorielle

Une **image vectorielle** est un ensemble de fonctions mathématiques décrivant des éléments de l'image :

- ▶ point
- ▶ ligne droites
- ▶ polygnes
- ▶ ellipses
- ▶ couleur remplissant une zone, dégradé ...
- ▶ composition additive, soustractive ...

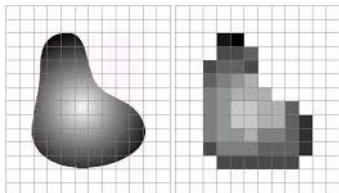
Le format typique est le svg qu'on peut manipuler avec inkscape ou illustrator. Il n'y a pas de notion de définition : on peut **zoomer et dézoomer** sans perte . Utile pour les polices, les icônes, les logos.

Regardons le code d'une image stockée sur le disque dur (en bmp et en svg).

Numérisation d'une image

Il y a trois étapes dans la numérisation d'une image (photo argentique ou numérique, vidéo, scanner 3D...) :

- ▶ Échantillonnage : discrétisation d'un signal continu (ici lumineux et en deux dimensions).
- ▶ La quantification : la valeur en un point est discrétisée (avec plus ou moins de précision).
- ▶ Encodage : représentation par une suite de 0 et de 1

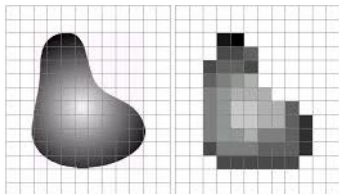


Compromis entre bruit, précision et taille.

Numérisation d'une image

Il y a trois étapes dans la numérisation d'une image (photo argentique ou numérique, vidéo, scanner 3D...) :

- ▶ Échantillonnage : discrétisation d'un signal continu (ici lumineux et en deux dimensions).
- ▶ La quantification : la valeur en un point est discrétisée (avec plus ou moins de précision).
- ▶ Encodage : représentation par une suite de 0 et de 1



Compromis entre bruit, précision et taille.

Même procédé pour le son (échantillonnage temporel).

Rappels sur Tkinter

Tkinter est une interface pour Tk, une librairie de création d'interface graphique.

Le sujet a été traité au premier semestre, voir le cours sur https://github.com/uvsq-info/l1-python/tree/master/cours/07_interface_graphique.

Tkinter permet de créer des fenêtres avec des widgets :

- ▶ bouton
- ▶ label (afficher du texte)
- ▶ canvas (afficher des images et des formes)
- ▶ champs de saisie

La librairie permet aussi de réagir aux entrées utilisateur (déplacement de souris, clic de souris, entrée clavier ...).

Créer une fenêtre

- ▶ Importer la librairie tkinter :

import tkinter as tk

- ▶ Créer la fenêtre :

racine = tk.Tk()

La variable `racine` représente le widget fenêtre et va servir à contenir tous nos autres widgets.

- ▶ Lancement de la boucle d'interaction avec l'utilisateur (mainloop) :

racine.mainloop()

Créer un bouton

Pour créer un bouton, on peut exécuter ce code :

```
bouton = tk.Button(racine, text="affichage 1", font =  
("helvetica", "30"))
```

- ▶ bouton est la *variable* qui représente le widget bouton
- ▶ **tk.Button** est la méthode qui crée le bouton
- ▶ **racine** est le widget à l'intérieur duquel est placé le bouton
- ▶ **text="affichage 1"** permet d'afficher dans le bouton le texte affichage 1 (paramètre optionnel)
- ▶ **font = ("helvetica", "30"))** permet d'afficher le texte avec une police helvetica de taille 30 (paramètre optionnel)

Placer des éléments dans une fenêtre

- ▶ La fenêtre est organisée en grille, chaque zone pouvant afficher un ou plusieurs widgets.
- ▶ La hauteur de chaque ligne (row) est donnée la plus grande hauteur des widgets de la ligne.
- ▶ La largeur de chaque colonne (column) est donnée la plus grande largeur des widgets de la colonne.
- ▶ La case (*column* = 0, *row* = 0) est en haut à gauche.

On utilise la méthode `grid()` pour placer un widget :

- ▶ **`bouton.grid(row=0)`** place le bouton sur la ligne 0.
- ▶ **`bouton.grid(row=1,column=1)`** place le bouton sur la ligne 1 et la colonne 1.
- ▶ **`bouton.grid(row=0, column= 1, colspan = 2)`** place le bouton sur la ligne 0 et sur les colonnes 1 et 2.

Gérer le clic sur un bouton

Pour gérer un clic sur un bouton, il faut définir une fonction qui s'exécutera quand on cliquera sur le bouton.

```
def mafonction() : print("bouton cliqué")
```

Gérer le clic sur un bouton

Pour gérer un clic sur un bouton, il faut définir une fonction qui s'exécutera quand on cliquera sur le bouton.

```
def mafonction() : print("bouton cliqué")
```

On lie la fonction au bouton grâce à l'argument optionnel `command` à la création du bouton.

```
bouton = tk.Button(racine, command = mafonction
```

Gérer le clic sur un bouton

Pour gérer un clic sur un bouton, il faut définir une fonction qui s'exécutera quand on cliquera sur le bouton.

```
def mafonction() : print("bouton cliqué")
```

On lie la fonction au bouton grâce à l'argument optionnel `command` à la création du bouton.

```
bouton = tk.Button(racine, command = mafonction)
```

On peut lier des fonctions à d'autres actions, grâce à la méthode `bind` :

```
def mafonction2() : print("clavier pressé")
```

Gérer le clic sur un bouton

Pour gérer un clic sur un bouton, il faut définir une fonction qui s'exécutera quand on cliquera sur le bouton.

```
def mafonction() : print("bouton cliqué")
```

On lie la fonction au bouton grâce à l'argument optionnel `command` à la création du bouton.

```
bouton = tk.Button(racine, command = mafonction
```

On peut lier des fonctions à d'autres actions, grâce à la méthode `bind` :

```
def mafonction2() : print("clavier pressé")
```

```
racine.bind("<KeyPress-a>",mafonction
```