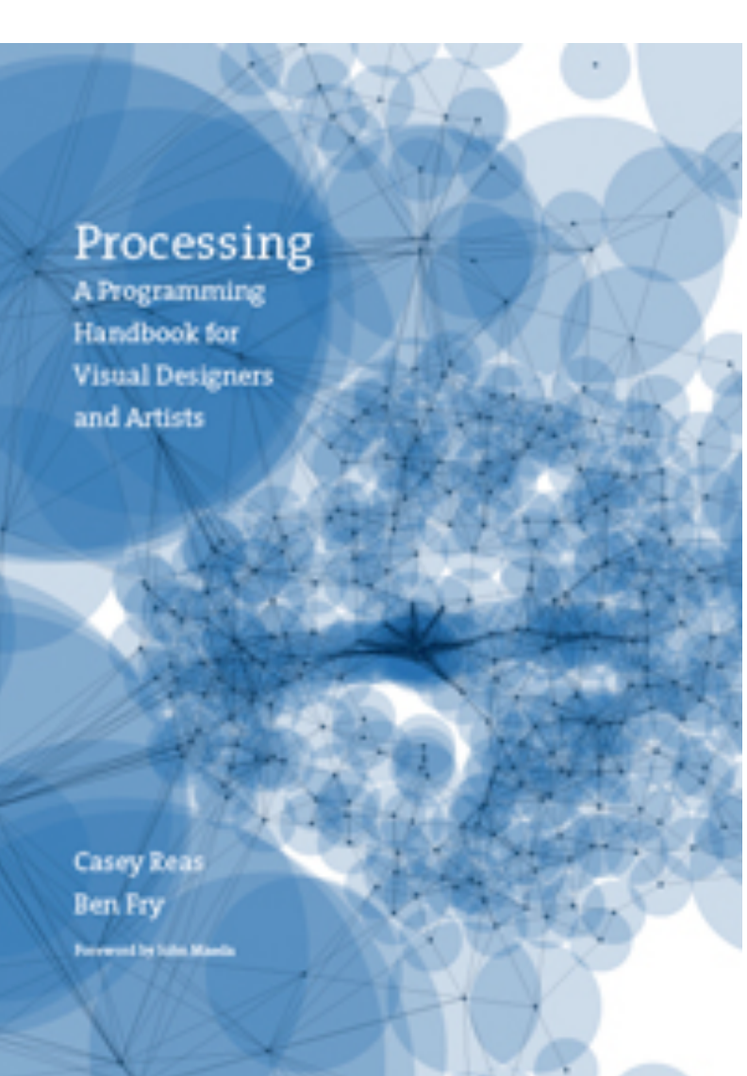
Programmation par Objets - Printemps 2010
Frédéric Mallet & Jean-Paul Roy
L1 Info - Faculté des Sciences

http://deptinfo.unice.fr/~roy

Université Nice SOPHIA ANTIPOLIS

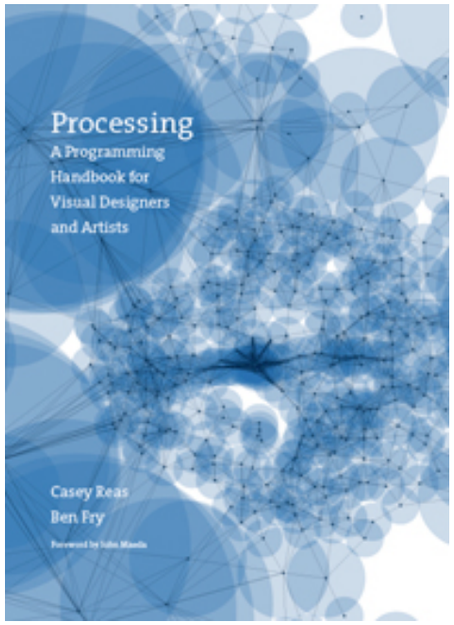
Introduction au système Processing



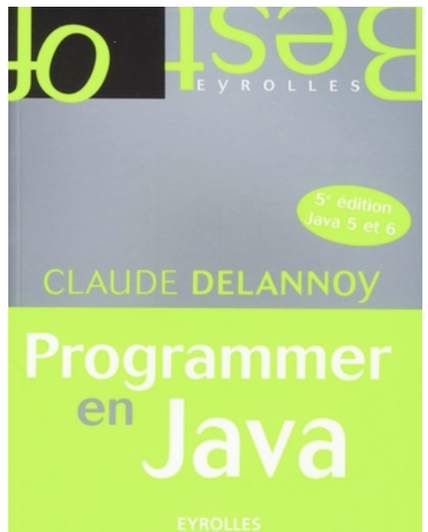
pages 9-34, 43-47

deptinfo.unice.fr/~roy

- **Processing. *A Programming Handbook for Visual Designers and Artists*** par Reas & Fry (MIT Press, 2007). Logiciel téléchargeable sur :
  www.processing.org



- **Programmer en Java** par Delannoy (Eyrolles, 2008)



- **Pour ce cours-ci, les séances 02-03 de l'Ecole d'Art d'Aix en Provence :**
  www.ecole-art-aix.fr/rubrique81.html

ECOLE SUPERIEURE D'ART D'AIX-EN-PROVENCE

Ressources sur cet enseignement

- Les livres cités à la page précédente, entre autres. Achat non obligatoire. Mais il est très important de **LIRE DES LIVRES** !

- Le langage support est **Java**, avec une introduction au système **Processing**, basé sur Java.

- La **page Web du cours**, que vous enregistrez dans vos signets :

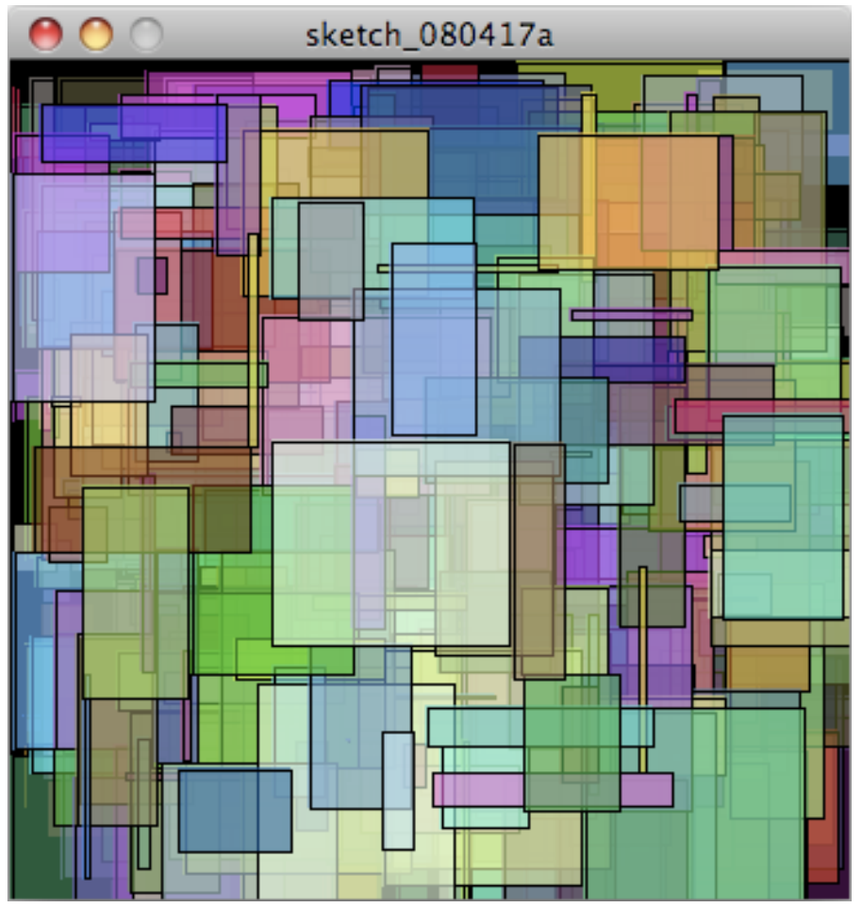
`http://deptinfo.unice.fr/~roy`

- La documentation exhaustive de **l'API Java** chez *Sun Microsystems*™ :

`http://java.sun.com/javase/6/docs/api/`

- Les logiciels à télécharger : voir la page Web du cours...

Le système Processing



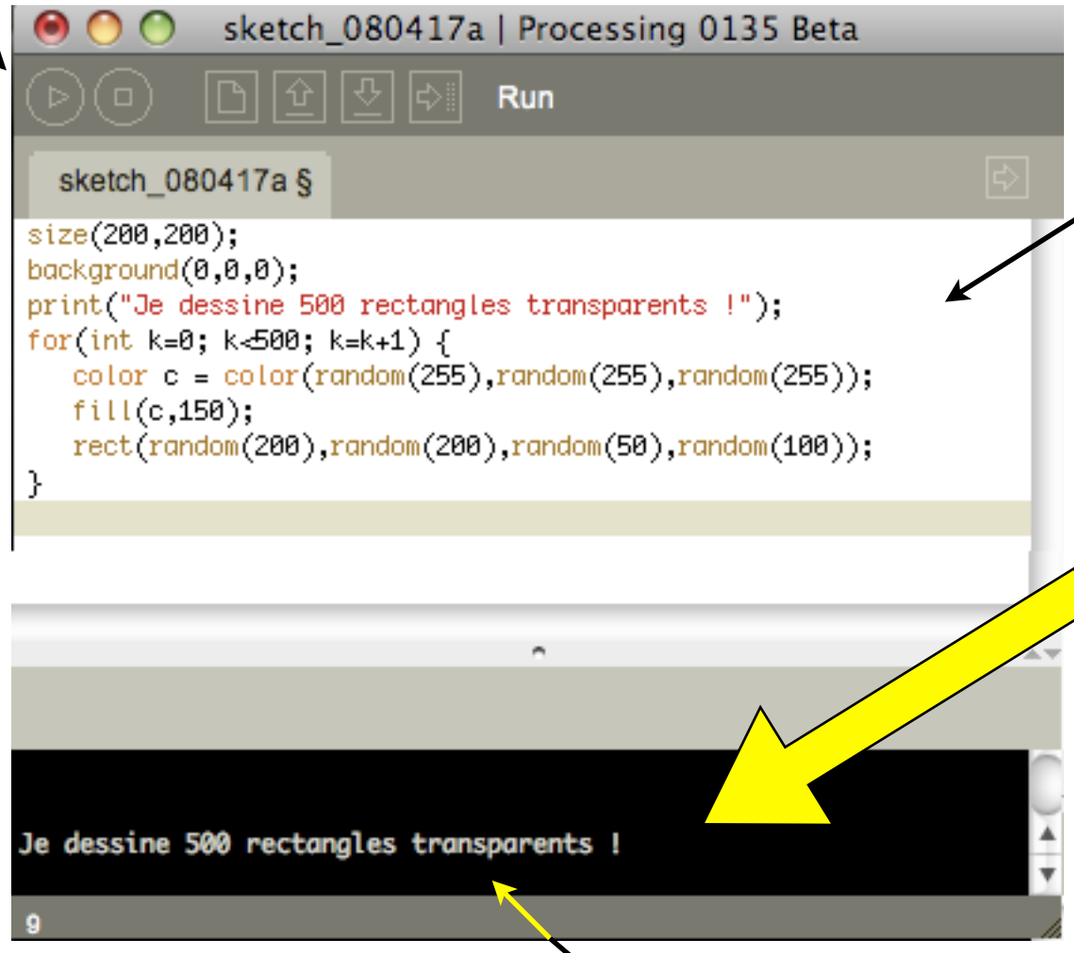
- Créé au MIT (Boston, USA) pour introduire les concepts de base de la programmation dans le contexte des arts visuels (artistes, Web).
- Idée : immersion rapide dans les programmes graphiques, l'Art Numérique et la simulation d'expériences physiques !...
- Syntaxe compatible avec celle de Java que nous étudierons plus tard. Processing est plus ludique et mieux adapté aux débutants.

```
size(350, 350);
background(0, 0, 0);
print("Je dessine 500 rectangles transparents !");
for (int k=0; k<500; k=k+1) {
  fill(random(255), random(255), random(255), 150);
  rect(random(350), random(350), random(100), random(100));
}
```

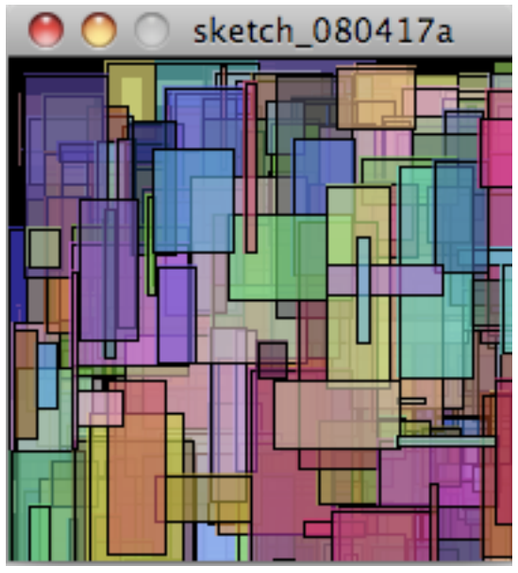
Présentation de l'environnement Processing

Exécuter (Run)

Editer son programme



Run



Résultat graphique (un ***sketch***)

Résultats textuels

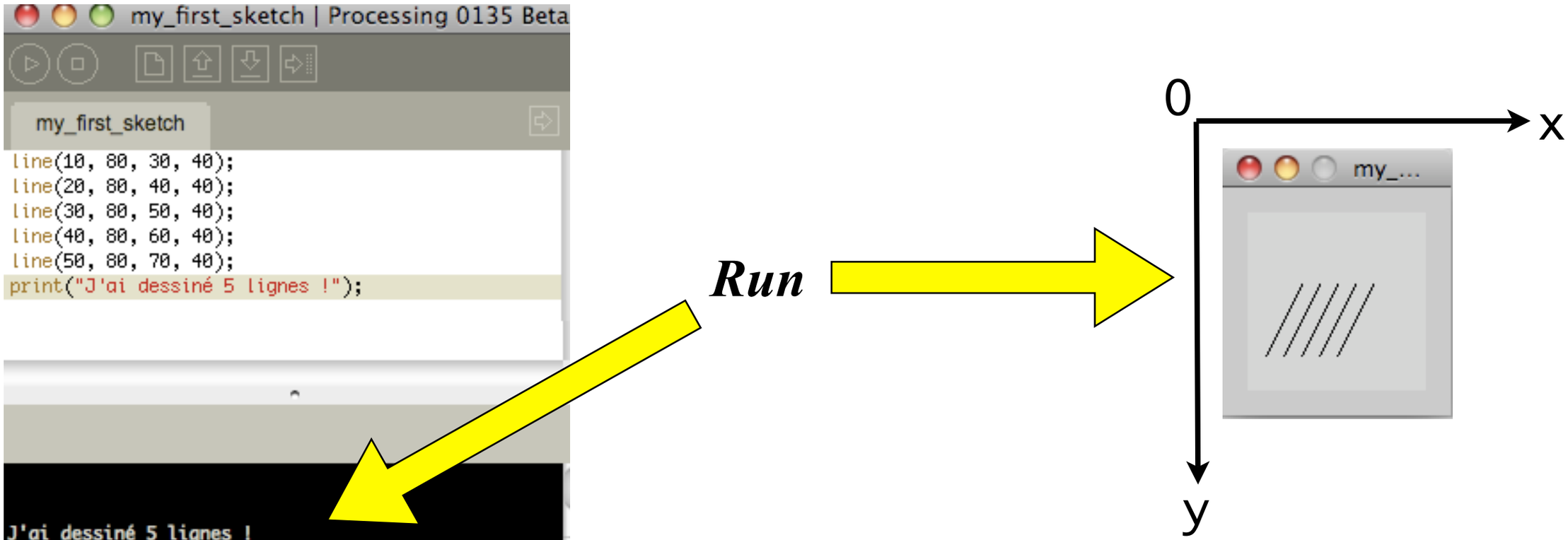
Mon premier programme Processing

- Un **programme élémentaire** est une suite d'instructions.

Programmation impérative !

- Une **instruction** demande à l'ordinateur d'effectuer une action, par exemple afficher du texte, ou dessiner un objet.

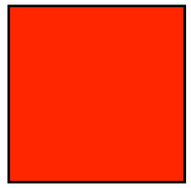
```
line(10,80,30,40);

print("Hello !");
```



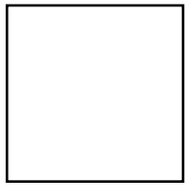
La couleur du fond de la fenêtre graphique

- La **couleur du fond** est grise par défaut, mais elle est réglable avec la **méthode** background(r,v,b) où r, v, b sont des **arguments** entiers de [0,255] représentant le niveau de rouge, de vert et de bleu.

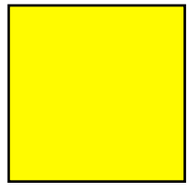
background(0,0,0);



background(255,0,0);



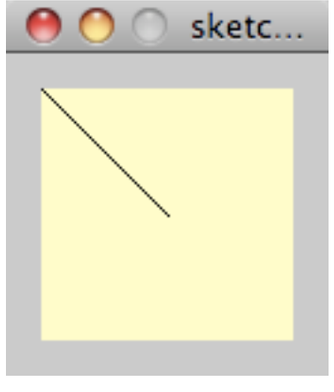
background(255,255,255);



background(255,255,0);

- La fenêtre graphique possède des **champs** width et height qui fournissent ses dimensions [100 x 100 par défaut].

```
background(255,255,180);          // jaune pâle
print("largeur = " + width + " et hauteur = " + height);
line(0,0,width/2,height/2);
```

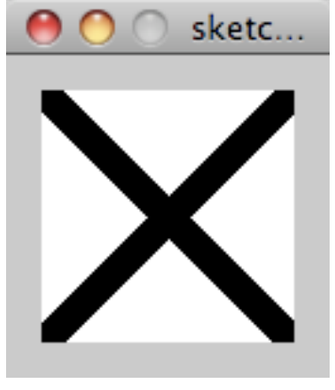



```
largeur = 100 et hauteur = 100
```

L'épaisseur du crayon

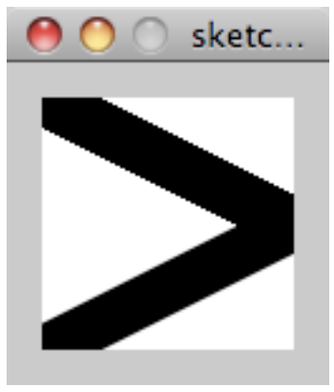
• L'**épaisseur du crayon** est de 1 pixel par défaut, mais on peut la faire varier [raisonnablement] avec la méthode `strokeWeight(n)` pour la mettre à n pixels :

```
background(255,255,255);
strokeWeight(10);
line(0,0,width,height);
line(0,height,width,0);
```



• Dans certaines lignes obliques, un désagréable **effet d'escalier** peut être supprimé avec la méthode `smooth()` qui va *lisser* la courbe.

```
background(255,255,255);
strokeWeight(20);
line(0,0,width,height/2);          // avec escaliers
smooth();
line(0,height,width,height/2);     // lisse !
```



Quelques formes géométriques

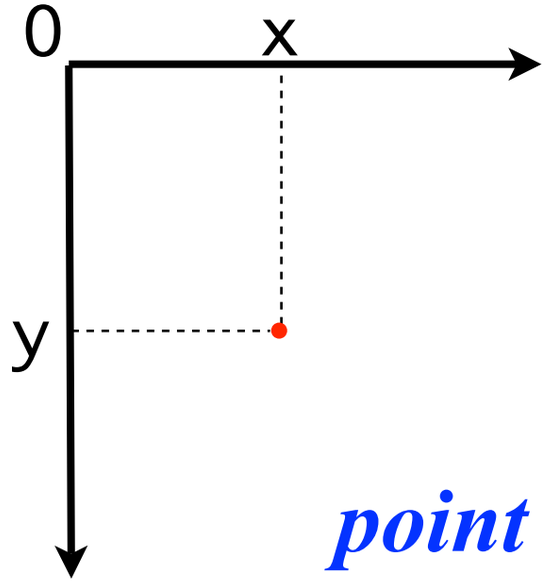
• Bien qu'il soit possible d'en définir d'autres, le langage Processing nous offre certaines formes géométriques prédéfinies.

Le POINT : `point(x,y)`

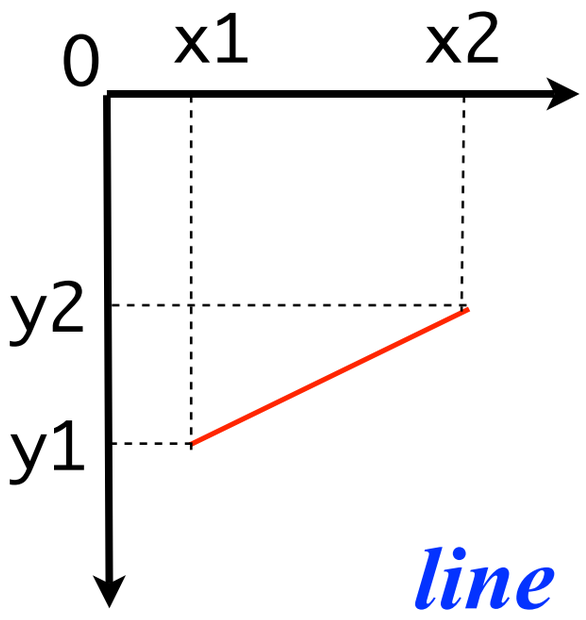
Le SEGMENT : `line(x1,y1,x2,y2)`

Le RECTANGLE : `rect(x,y,largeur,hauteur)`

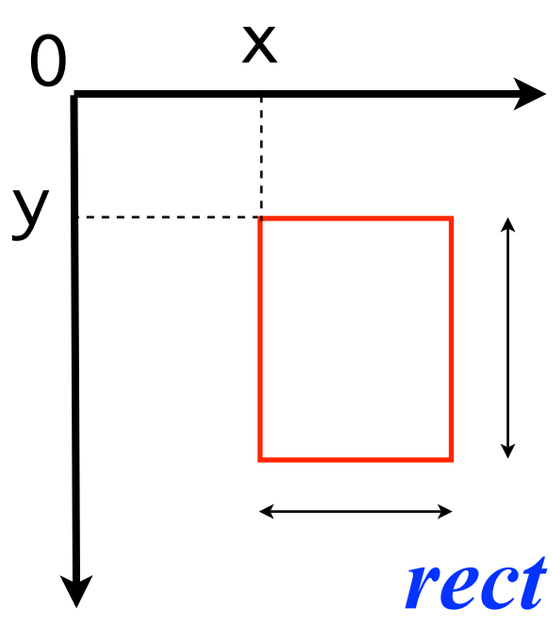
L'ELLIPSE : `ellipse(x,y,largeur,hauteur)`



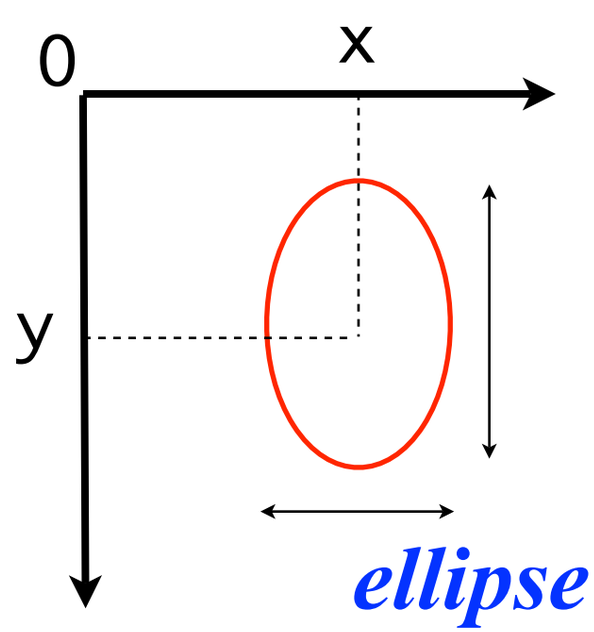
point



line

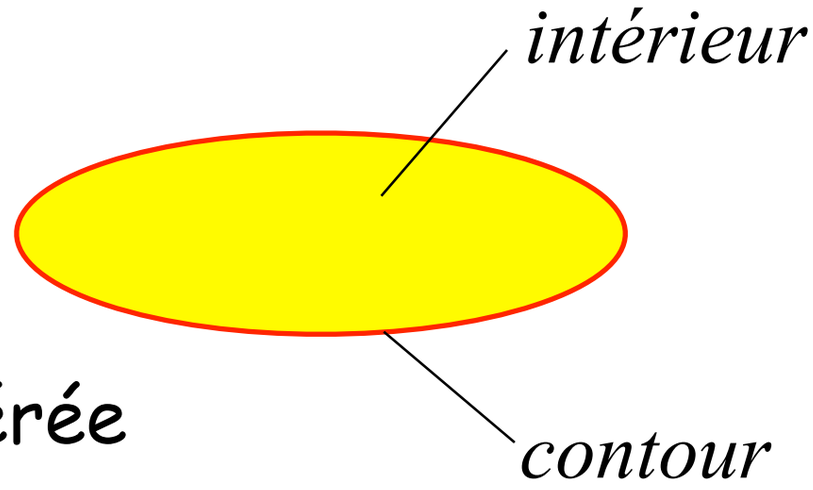


rect



ellipse

La couleur de dessin

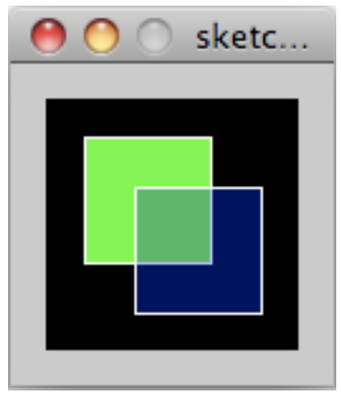


- Une forme a un **intérieur** et un **contour**.

- La **couleur du contour** [celle du *crayon*] est gérée par la fonction `stroke(r,v,b)` où r, v, b ∈ [0,255].

- La **couleur de remplissage** [celle de la *brosse*] est gérée par la fonction `fill(r,v,b)`.

- Un quatrième argument optionnel, dans [0,255], gère la **transparence** : 255 = opaque, 0 = transparente.

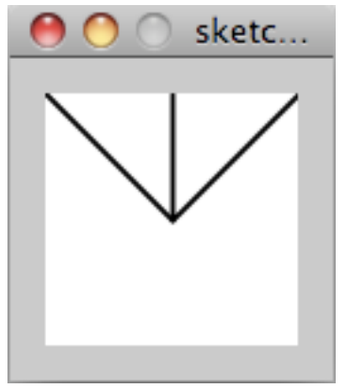



```
background(0,0,0);
stroke(255,255,255);    // contour blanc
fill(0,255,0);          //intérieur vert
rect(15,15,50,50);
fill(0,0,255,80);       //intérieur bleu + transparence:80
rect(35,35,50,50);
```

- Le programmeur **évite les sous-entendus** dans un programme. Par exemple, il ne supposera pas que la fenêtre est de 100 x 100 :

```
background(255,255,255);
strokeWeight(2);
smooth();
line(50,50,0,0);
line(50,50,50,0);
line(50,50,100,0);
```

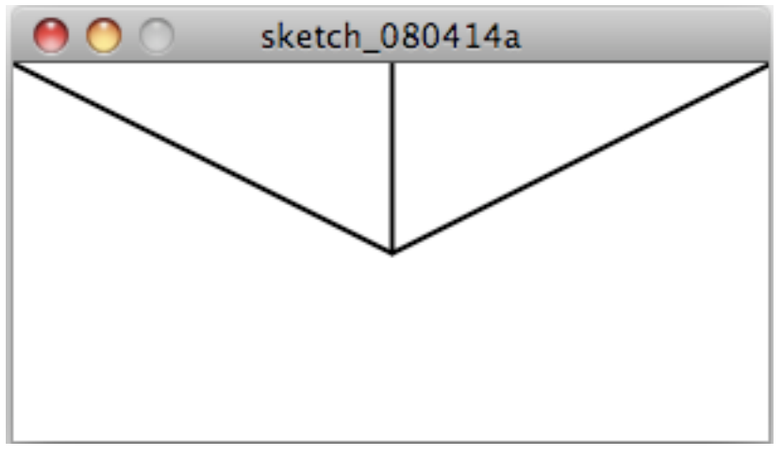
Bad !



- Si j'opte pour une fenêtre 300 x 150, que dois-je changer ?...

```
background(255,255,255);
size(300,150);
strokeWeight(2);
smooth();
line(width/2,height/2,0,0);
line(width/2,height/2,width/2,0);
line(width/2,height/2,width,0);
```

Better !



Peut-on éviter de calculer plusieurs fois width/2, height/2 ?

Utiliser des variables

• Une **variable** est un mot contenant des lettres et des chiffres, mais débutant par une lettre minuscule. **Une variable permet de mémoriser des valeurs de calculs intermédiaires.** Par exemple :

```
x      x31      vitesse      centreDeRotation
```

• Ce ne doit bien entendu pas être un **mot réservé du langage**, comme `print` ou `lineto` !

• Nos premières variables seront des **entiers exacts** [comme -203] ou des **nombres réels approchés** [comme 55.1046].

• L'ensemble des valeurs possibles pour une variable se nomme un **type**. Par exemple, `int` est le type *entier*, et `float` un type *réel approché*.

• Un ordinateur ne peut connaître TOUS les nombres. Par exemple, les entiers de type `int` varient dans $[-2^{31},2^{31}-1]$.

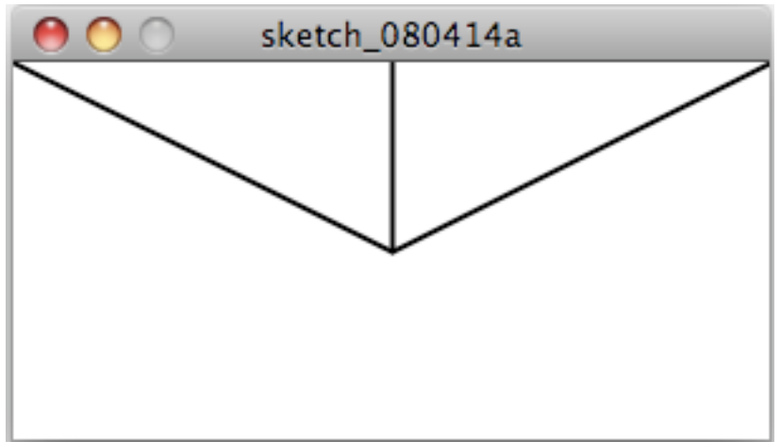
(32 bits)

```
[-2147483648, 2147483647]
```

- Pour **éviter les recalculs**, on utilise des variables :

```
background(255,255,255);
size(300,150);
strokeWeight(2);
smooth();
int x = width/2;
int y = height/2;
line(x,y,0,0);
line(x,y,x,0);
line(x,y,width,0);
```

Good !



- L'instruction `int x = 8;` **déclare** la variable x comme étant de type entier `int`, et **initialise** sa valeur à 8.

- On pouvait aussi simplement déclarer la variable par `int x;` puis l'initialiser plus tard par `x = 8;`

- Au moment de son utilisation dans une expression, une variable doit être à la fois déclarée et initialisée ! Elle ne peut pas changer de type !

L'instruction d'affectation

- L'opérateur = permet de **changer la valeur d'une variable** :

variable = expression;

- Exemples :

```
x = 3;          x = y + 1;
```

```
x = x + 2;
```

*une **affectation** : x devient égal à x+2*

```
float hypo = sqrt(x * x + y * y);

projx = r * cos(PI/5);        // la constante PI = 3.14159...
```

- Le signe = de Processing/Java n'est **pas celui des maths** ! Dans l'instruction `x = x + 2;` , ne simplifiez pas par x !

- Attention, **à gauche d'une affectation**, doit se trouver une variable, jamais un calcul :

~~`x + 2 = x;`~~

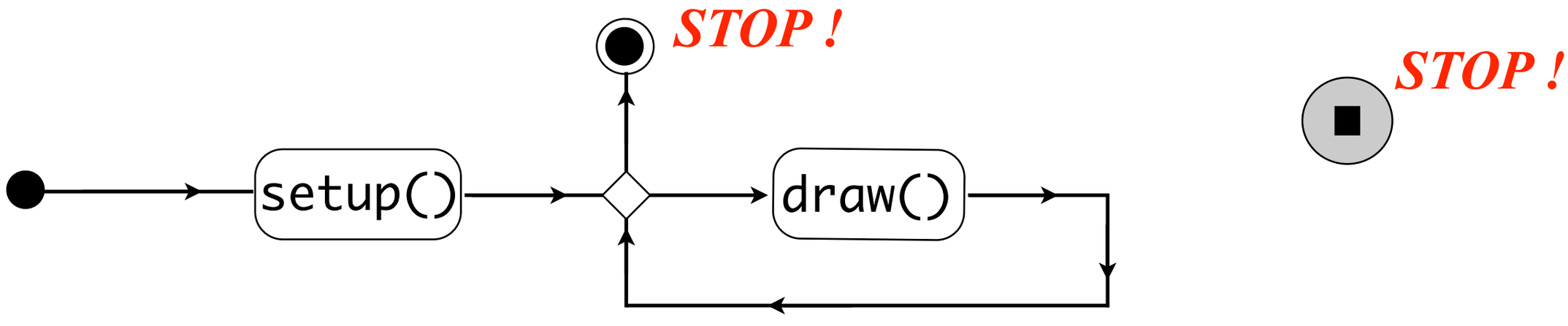
Structure d'une animation en Processing

- Le système Processing fournit au programmeur un mécanisme analogue aux ***GIF animées*** dans les pages Web ! Une GIF animée n'est autre qu'une suite d'images, que l'on feuillette tous les 1/20 seconde par ex.

- Les fonctions `setup()` et `draw()` permettent d'**animer un sketch**.

- La fonction `setup()` est invoquée une seule fois par Processing au lancement du programme.

- La fonction `draw()` dessine une image qui va être affichée. La fonction `draw()` est ré-invoquée en boucle dès qu'elle termine, provoquant une suite d'images perçues comme une animation !

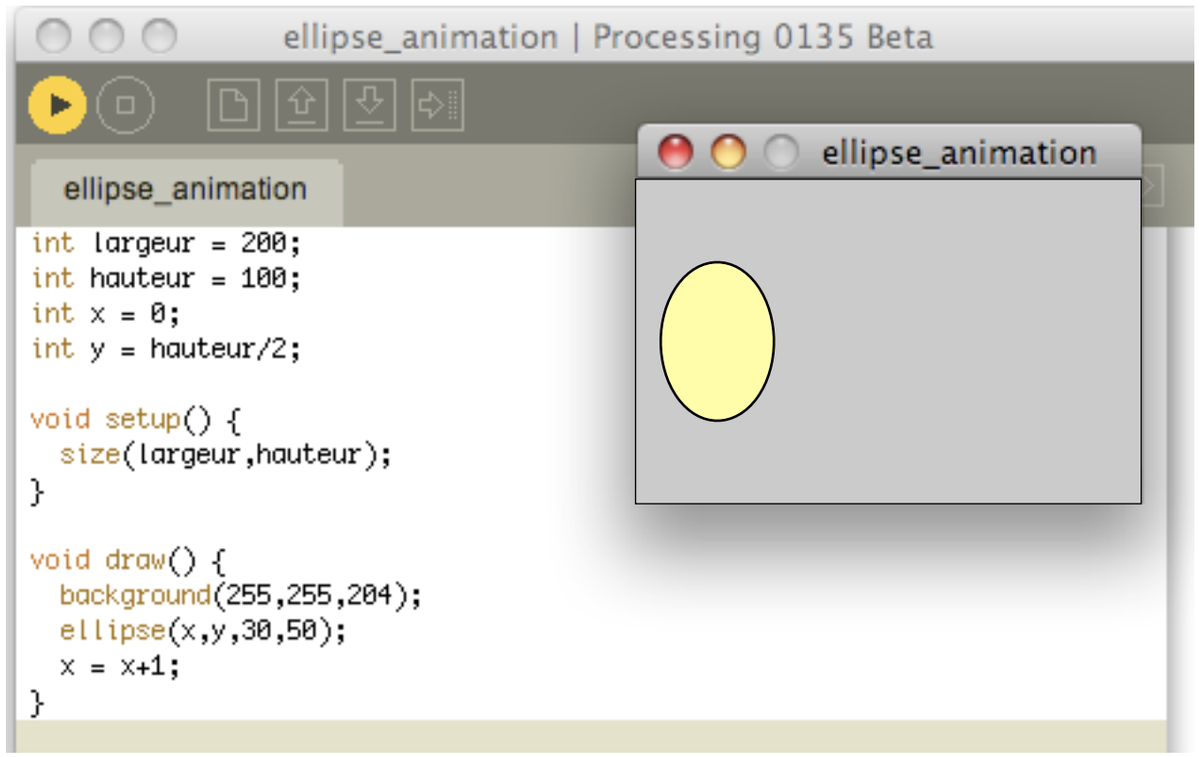


Un exemple d'animation : une ellipse en mouvement

- La méthode `ellipse(x,y,width,height)` dessine une **ellipse** de centre `(x,y)`, de largeur `width` et de hauteur `height`. Si `width` est égal à `height`, on obtient un cercle...

- L'**animation** est obtenue en faisant avancer le centre de l'ellipse d'un pixel vers la droite à chaque image par l'instruction `x = x + 1;`

```
int largeur = 200;
int hauteur = 100;
int x = 0;
int y = hauteur/2;

void setup() {
  size(largeur,hauteur);
}

void draw() {
  background(255,255,204);
  ellipse(x,y,30,50);
  x = x+1;
}
```



Comment faire pour arrêter l'ellipse au bord de l'écran ?

Eléments linguistiques

Nos premiers types de données

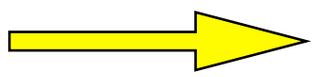
• Nous en verrons beaucoup d'autres. Pour l'instant nos variables pourront contenir :

- des **entiers** [type `int`], comme `4589201` ou `-5643`

- des **nombres réels approchés** [type `float`], comme `56.70554`. Notez que si un *float* est attendu, un *int* sera accepté et automatiquement converti en *float*. Mais 2.0 n'est pas 2 !

- des **booléens** [type `boolean`]. Il n'y a que deux valeurs booléennes : `true` et `false`. Une expression comme `x + 2 > y` est vraie ou fausse : elle vaut `true` ou `false`.

```
int x = 5;
int y = 8;
boolean b = (x + 2 > y);
print("b = " + b);
```



```
b = false
```

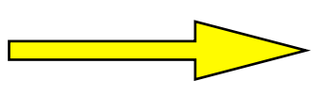
*Les **booléens** vont nous permettre de poser des conditions.*

Expressions et instructions

• Les **expressions** sont formées de données et d'opérateurs sur ces données. **Une expression a une valeur**.

• Par exemple, si x est un entier, alors (x + 2) / 3 est une expression. Si de plus je sais que x vaut 6, la valeur de cette expression est l'entier 2.

```
int x = 6;
int y = (x + 2) / 3;
print("y = " + y);
```



```
y = 2
```

• Les **instructions** sont formées de données et d'opérateurs sur ces données. **Une instruction n'a pas de valeur**.

• Par exemple, `int x = 6;` ou encore `print("y = " + y);` sont deux instructions. Notez le point-virgule qui termine une telle instruction.

• En Java, on dit parfois qu'une expression a pour valeur `void`, pour dire qu'elle n'a pas de valeur !

Les fonctions et leur signature

- Une **méthode** est un mécanisme logiciel qui à un ou plusieurs arguments associe une [seule] valeur.

- Les matheux utilisent une **flèche** :

```
pow : float x float → float
print : String → void
line : int x int x int x int → void
```

$$void = \emptyset$$

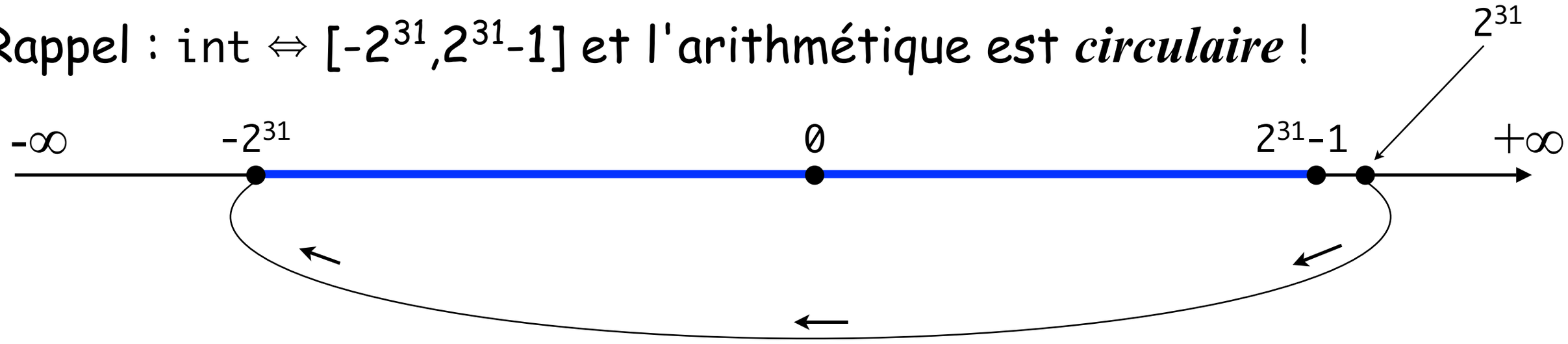
- Les programmeurs Java utilisent une **signature** :

```
float pow(float x, float y)
void print(String str)
void line(int x1, int y1, int x2, int y2)
```

- Une fonction $f : \mathbb{Z} \times \mathbb{R} \rightarrow \mathbb{R}$ sera représentée en Java par une méthode dont la signature est : `float f(int x, float y)`

Fonctions arithmétiques dans les entiers [int]

- Rappel : `int` ⇔ [-2^{31},$2^{31}-1$] et l'arithmétique est ***circulaire*** !



- Opérations primitives :

`a + b`	`a - b`	`a * b`	`a / b`	`a % b`
			quotient entier	*reste de la division*

- Priorités d'opérateurs : `3 + 4 * 5` est lu comme `3 + (4 * 5)`

les moins prioritaires		*les plus prioritaires*		
+	-	*	/	%

- Si les opérateurs ont même priorité, calculs de gauche à droite !

Fonctions arithmétiques dans les réels approchés [float]

- Il n'y a qu'un nombre **fini** de réels approchés ! Donc il existe ε > 0 tel que l'intervalle]0, ε[soit vide ! C'est ***l'epsilon de la machine***.

- Un calcul entre nombres approchés donnera un résultat approché !

- Opérations primitives :

`a + b`	`a - b`	`a * b`	`a / b`	`a % b`
			quotient approché	*reste approché*
`cos(a)`	`sin(a)`	`tan(a)`	`log(a)`	`exp(a)`
`abs(a)`	`sqrt(a)`	`pow(a,b)`	`floor(a)`	*etc.*

- Si une opération attend un nombre approché, et qu'on lui fournit un nombre entier exact, ce dernier sera converti *automatiquement* en nombre approché. Par exemple `sqrt(2)` est lu comme `sqrt(2.0)`

Mots de bits (binaire = base 2)

1	1	0	1	1	0	0	1

un octet

- 1 **bit** contient 0 ou 1

- 8 bits peuvent représenter $2^8 = 256$ valeurs : un **octet**

- Un **mot-mémoire** sur une machine actuelle dont les adresses varient sur 32 bits est donc constitué de 4 octets.

- On peut ainsi représenter tous les ensembles FINIS de valeurs :

Il suffit de prendre un nombre suffisant de bits !

- Pour les ensembles INFINIS de valeurs, on se conserve qu'un (grand) sous-ensemble fini !

ENTIERS EXACTS				REELS INEXACTS	
byte	short	int	long	float	double
8 bits	*16 bits*	*32 bits*	*64 bits*	*32 bits*	*64 bits*

Les textes : String

• Les **chaînes de caractères** (mot, portion de phrase) sont les éléments du type String.

```
"Elle ajoute 243 !"        "2009"        "rouge"
```

```
print("rouge");
```

• L'opérateur + permet de calculer la **concaténation** de deux chaînes de caractères :

```
print("rou" + "ge") ;
print("hauteur=" + height);
print(2 + 3 + "4");
print(2 + "3" + 4);
```

calcul de gauche à droite et conversion automatique en chaîne !