Processing #1

TECH MODEL RAILROAD CLUB

1/L'accès aux ordinateurs et à tout ce qui peut nous apprendre qqch sur la marche du monde doit être total et sans restrictions. Appliquez toujours ce principe : Faites le vous même.

2/L'accès à l'information doit être libre.

3/Défiez le pouvoir- défendez la décentralisation.

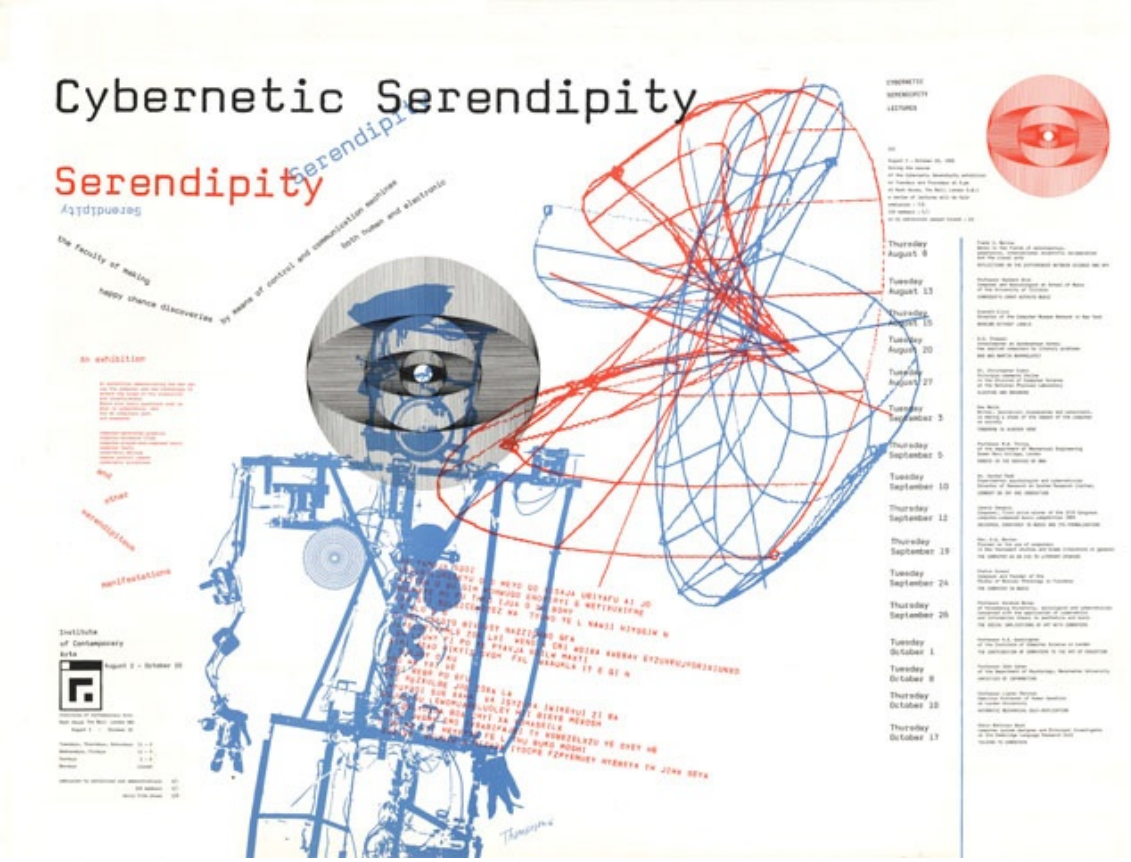
4/Les hackers doivent être jugés sur leurs résultats et non sur des critères fallacieux comme leurs diplômes, leur âge ou leur classe.

5/On peut créer de la beauté et de l'art avec un ordinateur.

6/Les ordinateurs peuvent améliorer votre vie.

L'éthique des Hackers /Steven Levy (http://www.editions-globe.com/lethique-des-hackers/)

1968 _Cybernetic Serendipity /commissaire d'expo Jasia Reichardt



https://www.youtube.com/watch?v=n8TJx8n9UsA
https://www.youtube.com/watch?v=oSwovB28B34
Edward Ihnatowicz
https://www.youtube.com/watch?v=1jDt5unArNk
John Withney
https://www.youtube.com/watch?v=ZrKgyY5aDvA
Jean Tinguely
https://www.youtube.com/watch?v=4-NrTsq6bsg
Saul Bass et John Withney
https://www.youtube.com/watch?time_continue=138&v=-DU0IVmBgsQ

...

Aujourd'hui

Quelques exemples

https://www.youtube.com/watch?v=0G7-Y-QSsRo

http://blog.blprnt.com/blog/blprnt/just-landed-processing-twitter-metacarta-hidden-data

https://vimeo.com/134905817

http://www.lavitrinedetrafik.fr

https://chevalvert.fr/installation/le-temps-suspendu/

http://in-flexions.com/

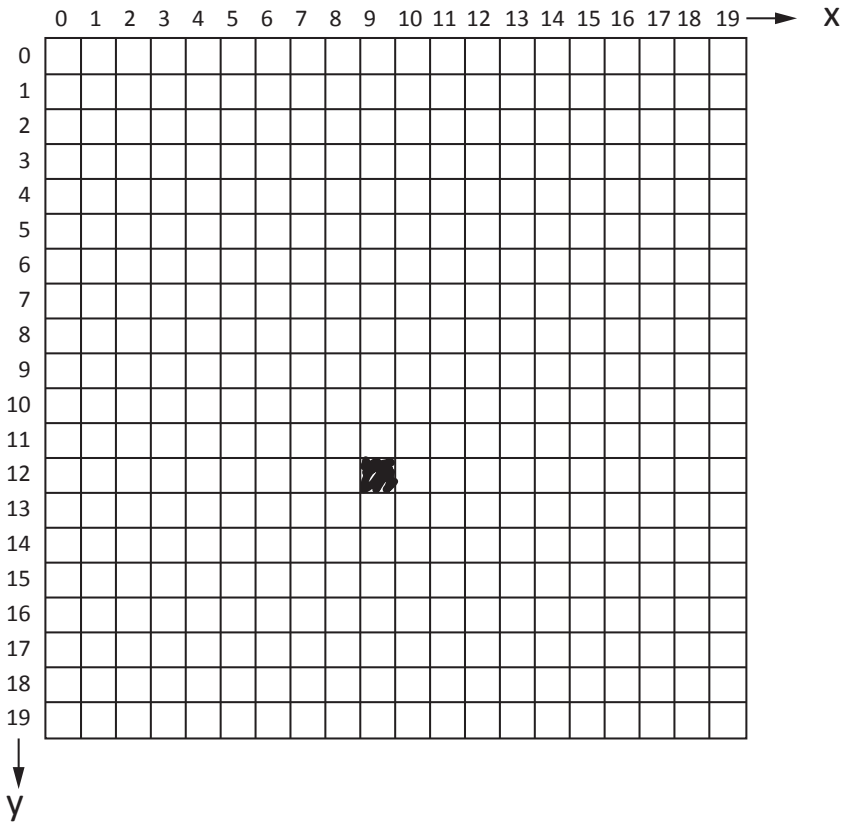
http://www.esono.com/boris/projects/poetry02/

Processing / java
Ben Fry & Casey Reas //createurs de Processing

processing.org // download free

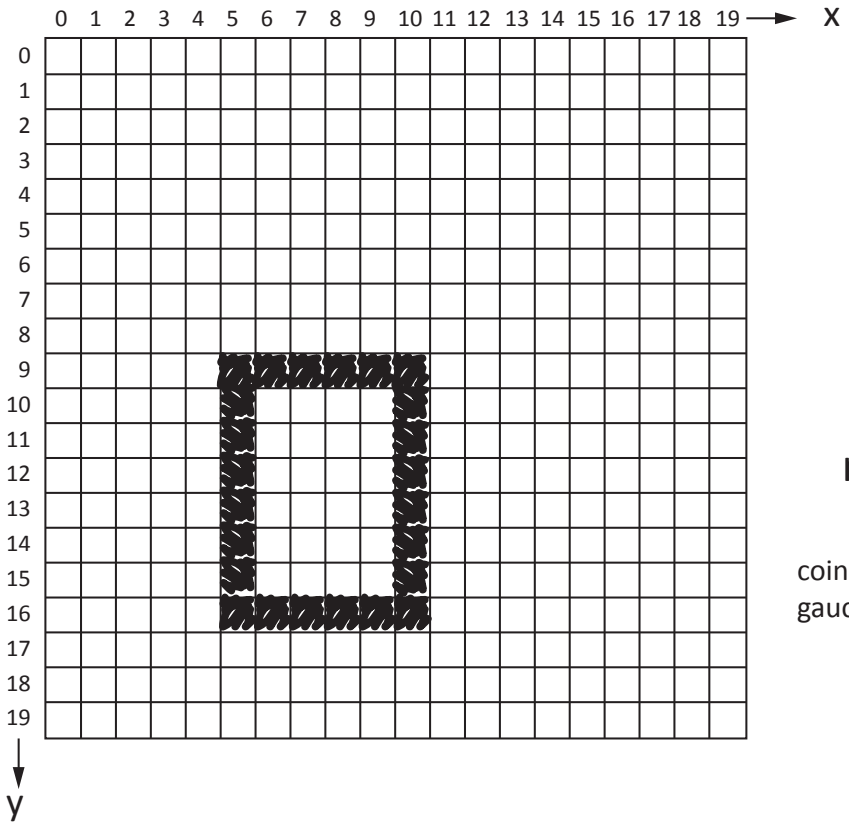
_1 Pixels.

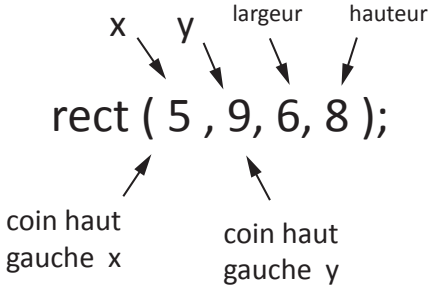
La fenêtre d'éxécution du script s'organise en pixels sur 2 axes (ou 3 si l'on fait de la 3D, mais commençons par la 2D).
Un axe x, un axe y .

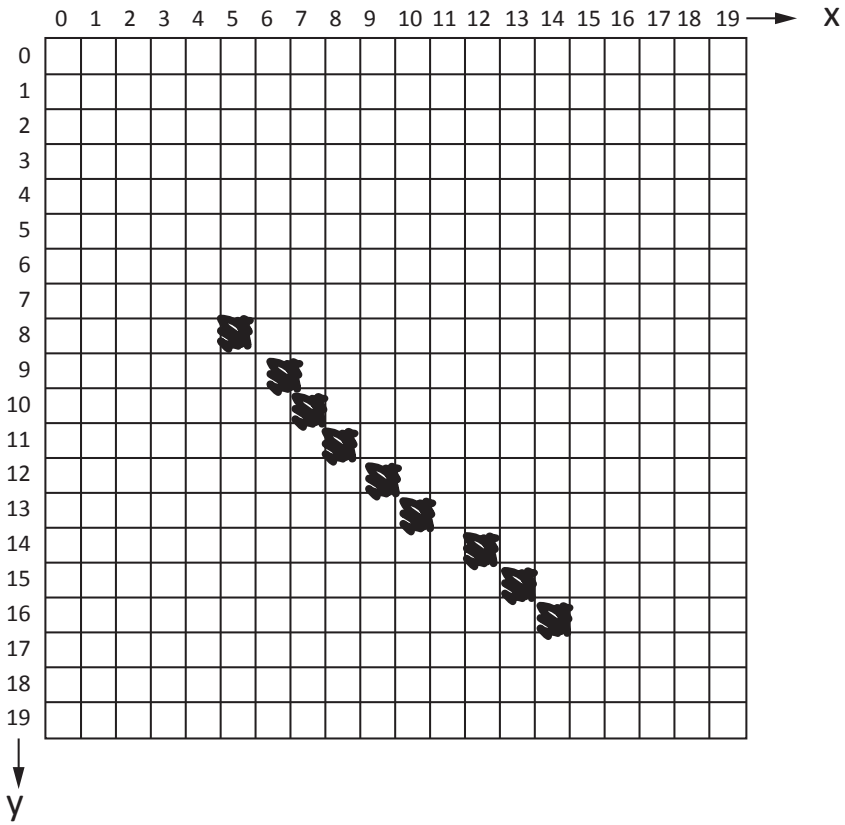

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
x
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
y

x y

```
point( 9 , 12 );
```


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
x
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
y


x
y
largeur
hauteur
rect (5 , 9, 6, 8);
coin haut gauche x
coin haut gauche y

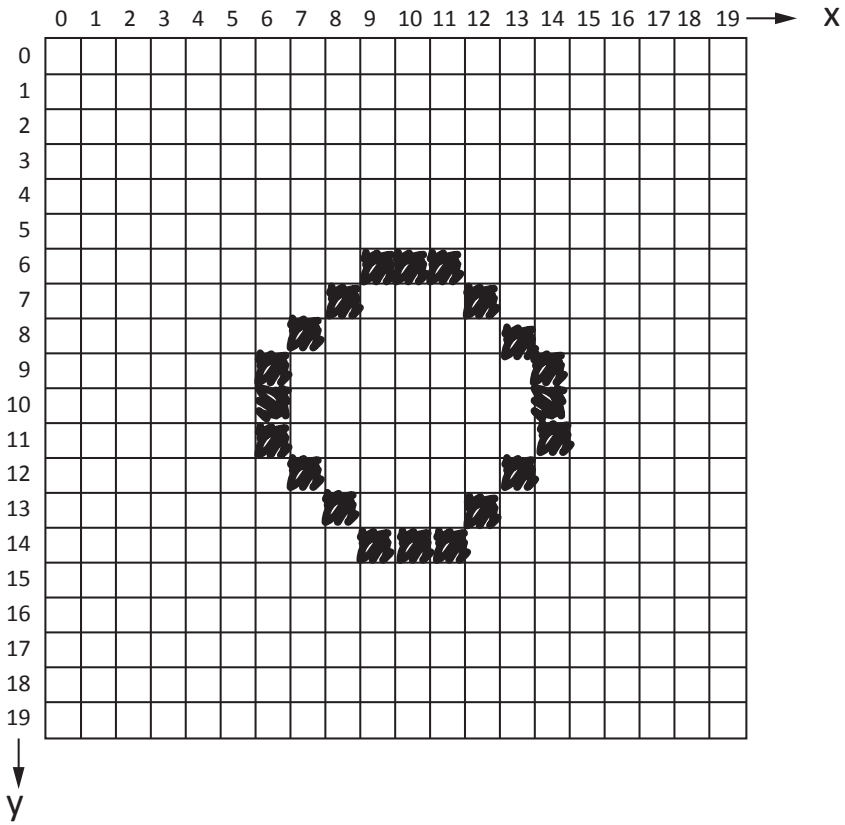


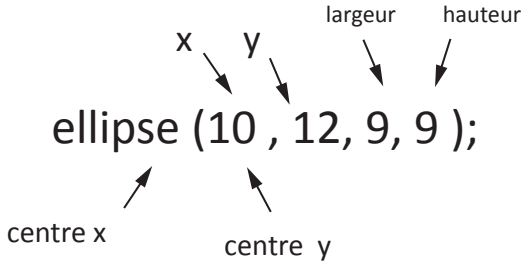
x y x1 y2

```
line (5 , 8, 14, 16 );
```

x, y point de début de ligne

x2,y2 point de fin de ligne

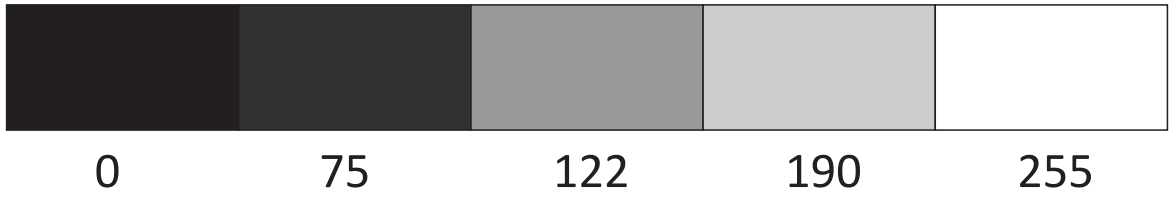




Si largeur et hauteur sont égales
alors on obtient un cercle
sinon une ellipse.

_2 Couleurs et Niveaux de gris.

Pour colorer une forme ou une ligne en niveaux de gris, on utilise une valeur entre 0 et 255 en sachant 0 correspond au noir et 255 au blanc.



Remplir une forme :
fill(niveauxdegris) ;

Donc
fill(0) ; // remplir avec du noir
fill(122) ; //remplir en gris

etc

Pour la couleur du contour :
stroke(niveauxdegris) ;

donc
stroke(0) ; // contour noir
stroke(122) ; //contour gris

Si on veut une forme vide :

```
noFill() ;
```

Si on veut une forme sans contour

```
noStroke() ;
```

Pour augmenter l'épaisseur du contour :

```
strokeWeight( 1 ); //pour une épaisseur de 1 pixel
strokeWeight( 3 ) ; //pour une épaisseur de 3 pixels
```

Fond de fenêtre.

Pour donner une couleur au fond on applique la fonction suivante :

```
background(255) ; // une seule valeur, donc niveaux de gris, ici 255 donc blanc.
```

_3 RGB

Pour appliquer de la couleur, on utilise 3 valeurs : Rouge Vert Bleu entre 0 et 255 .

fill(255,0,0) ; rempli la forme avec du rouge
fill(255,255,0) ; rempli la forme avec du jaune

Idem pour les contours ou le fond.

stroke(100,0,90) ;
background(40,50,100) ;

bref un mélange rouge vert bleu…

_4 TAILLE DE LA FENETRE

Pour agrandir la fenêtre où s'exécute le script : on utilise la fonction size(w,h) ; ou w, width, est la largeur et h, height, la hauteur.
Ex :
size (600,400) ;

Si le script doit prendre tout l'écran :
fullScreen() ;

_5 FLOW

Processing lit et éxécute les tâches dans l'ordre, les unes après les autres.

Pour dessiner des carrés de couleurs différentes successivement il faut à chaque fois redonner une couleur.

Ex :

```
size(400,400);   //taille de la fenêtre
background(255); //fond blanc

fill(255,255,0);  //couleur
rect(0,0,100,100);//forme

fill(0,255,0); // couleur
rect(100,100,100,100); //forme

fill(0,255,255); // couleur
rect(120,200,100,100); //forme


fill(255,0,255); // couleur
rect(230,190,100,100); //forme
```

_6 Bloc de code et dynamique

Dans processing on utilise des blocs de code. Chaque bloc débute par {
Et se termine par }

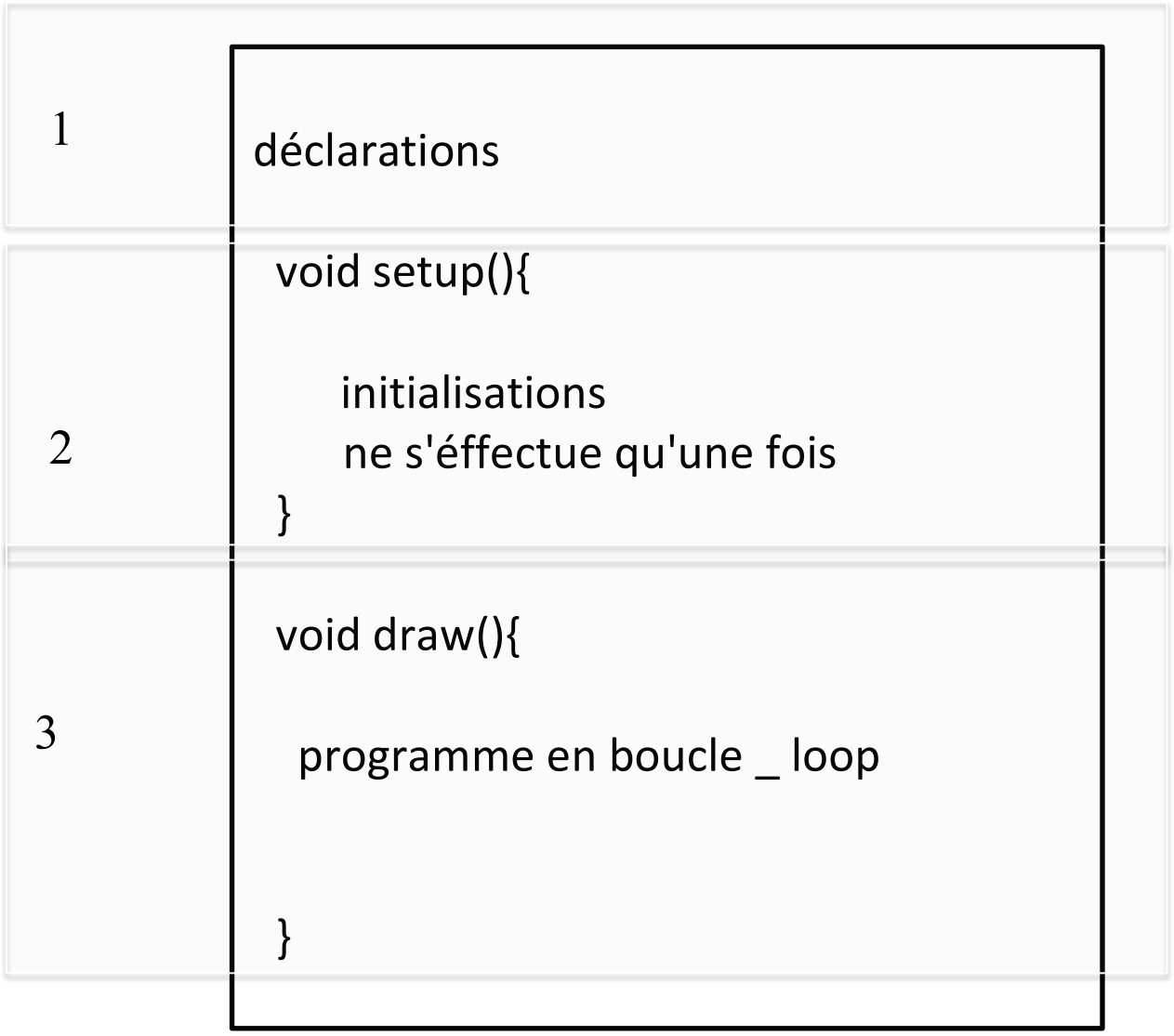
Tout bloc ouvert doit être fermé même si il est dans un autre bloc :

```
{
ceci est un bloc
}

{ ceci est un bloc
        {ceci est un bloc dans un bloc
        }
}
```

Pour que le sketch réagisse ou évolue il faut qu'il soit dynamique, toujours en écriture…
Alors un skecth est en minimum 3 parties :

1



_7 Interactivité

Processing est capable d'intercactivité :

La sourie est un outil simple pour apprendre et tester des variations.

mouseX donne la coordonnées en x de la sourie .

mouseY donne la coordonnées en y de la sourie .

pmouseX est la position de la sourie en X précédent le rafraichissement du background.
pmouseY est la position de la sourie en y précédent le rafraichissement du background.

_8 Mapping

La fonction map est utile pour changer l'échelle d'une valeur.
Elle donnera une valeur float(nombre avec virgule).

Dans :

```
map(mouseX, 0,width,0,255) ;
```

Quand on fait un mapping de mouseX. La valeur originale qui est comprise entre entre 2 valeurs mini et max (ici 0 et la largeur de la fenêtre (width)) , est convertie entre 2 nouvelles valeurs mini et max (ici 0 et 255) .

_9 if

Pour donner des conditions d'éxécution on peut utiliser la fonction if.

```
if (x >100)  {
text( « hop ! », 100,100) ; }
```

On pose une condition : si(x est plus grand que 100) alors écrire « hop ! » aux coordonnées 100,100.

La condition à remplir est entre parenthèses, puis on ouvre un bloc de code (avec l'accolade) on donne ce qui doit être exécuté si la condition est remplie. Et on ferme le bloc de code (avec l'accolade) .

Pour écrire si x est égal à 100 alors écrire hop

```
if (x ==100)  {
text( « hop ! », 100,100) ; }
```

Pour écrire si x est supérieur à 100 alors écrire hop

```
if (x >100)  {
text( « hop ! », 100,100) ; }
```

inferieur s'écrit <
inferieur ou égal s'écrit <=
supérieur ou égal s'écrit >=

Pour écrire si x est différent de y alors écrire hop

```
if (x != y) {
text( « hop ! », 100,100) ; }
```

Pour écrire si x est plus grand que 100 et y est plus grand que 100 alors écrire hop

```
if (x >100 &&  y >100) {
text( « hop ! », 100,100) ; }
```

Pour écrire si x est plus grand que 100 ou x est plus petit que 0 alors écrire hop

```
if (x >100 ||  x <0) {
text( « hop ! », 100,100) ; }
```

_10 HSB

On peut changer le mode de couleurs du programme.
Par défaut les couleurs sont soit en RVB ou en valeurs de gris selon le nombre de valeurs que l'on donne .
fill(R,G,B) ;
fill(gris) ;

Si l'on indique
 colorMode(HSB,100) ;
On change de mode de couleur et les valeurs que l'on donne seront la teinte(H) , la saturation (S), et la brillance(B). Le tout sur une échelle de valeur comprise ici en 0 et 100.
Si on avait écrit :
colorMode(HSB, 400) ;
Les valeurs des teintes saturations et brillances seraient alors comprise entre 0 et 400 .

_11 random()

La fonction random sert à donner une valeur aléatoire.
Pour obtenir un valeur au hazard entre 0 et 100, on dirait :

```
float r= random(100) ;
```

Lorsqu'une seule valeur est indiquée, le choix sera fait entre 0 et cette valeur.
Ou presque ! Ce sera entre 0 et 100 mais sans inclure 100.

Pour choisir entre 2 valeurs on les indique entre les parenthèses. Pour choisir une valeur aléatoire entre 8 et 57 on écrirait

```
float  r = random(2,57) ;
```

ici aussi le 57 n'est pas inclus.

Random donne une valeur float. Pour la transformer en nombre entier on écrirait plutôt :

```
int r = int(random(2,57) );
```

Donc pour obtenir un nombre entier entre 0 et 10 on écrirait :

```
int r = int(random(0,11) );  //11 parce que le random se fera entre 0 et 11 sans inclure 11.
```

_12 La boucle for

Une boucle for est une séquence de répétition.

On l'écrit en 3 parties, initialisation , teste et update.

```
for( initialisation, teste, update){
                    dessine une ligne;
}
```

On initialise une variable qui va se modifier à chaque fois si la condition du test n'est pas remplie.

```
 for( int i=0 ; i<100 ; i++){
                  line( i,0,i,height) ;}
```

On pourrait dire : Tant que i (qui tout d'abord est égal à 0) est inférieur à 100 alors i augmente de 1 et une ligne est dessinée. A chaque boucle, i augmente et l'action entre accolade est répétée.
Quand la condition est remplie le programme passe à la suite.