Workshop __ Beaux arts de Besançon _ 2 / 3 / 4 mars 2010

PROGRAMME DU WORKSHOP

2 MARS

9H/12H

Présentation du logiciel Processing
Qui fait quoi avec Processing?
Hello World : premier sketch

14H/18H

Inside Processing : amener l’interaction
Notions fondamentales de programmation
- Variables
- Conditions
- Boucles

PROGRAMME DU WORKSHOP

3 MARS

9H/12H

Créer des fonctions personnalisées
Typos / images
Bibliothèques => à quoi servent-elles, comment les utiliser?
Arduino

14H/18H

Créer un sketch s’intégrant au projet Processing Monsters

PROGRAMME DU WORKSHOP

4 MARS

9H/12H

Sources: où chercher du code Processing
Mise en application sur un projet

14H/18H

Finalisation des projets
Exportation en applet web pour mise en ligne

Présentation de Processing

- À la fois un **language de programmation** et un **IDE,**
un logiciel pour développer des programmes.

- Aussi appelé **P5** ou **Proce55ing** , pour plus de commidité lors des recherches sur le net.

- Logiciel libre, gratuit, open source, multiplateforme.

Présentation de Processing

- Développé **par des artistes, pour des artistes**.
Utilisés par des étudiants, des artistes, des graphistes pro, des chercheurs.

- Philosophie: **la programmation pour les «non ingénieurs»**,
un outil pour la création numérique, gratuit, open source => collaboratif.

- Utilisé pour l'apprentissage des **bases de la programmation**,
le **prototypage rapide** de programmes, mais aussi comme **outil de production**.

- Développé dans un **esprit de simplification de la programmation**, tout en offrant
aux utilisateurs avancés la possibilité de développer des librairies ou des méthodes plus complexes.

Présentation de Processing

- On peut quasiment tout faire...

- Les domaines de prédilection pour l'utilisation:
création d'images fixes ou animées de processus qui peuvent être interactifs.
Mais les nombreuses bibliothèques disponibles permettent d'étendre considérablement les possibilités.

- Ces bibliothèques sont le fruit d'une grande communauté d'utilisateurs, très dynamique.

Présentation de Processing

- Développé depuis 2001 par **Casey Reas et Ben Fry**, alors étudiants au Aesthetics and Computation Group du MIT Media Lab sous la direction de **John Maeda** (http://www.maedastudio.com).

- Processing est une évolution de Design by Numbers, un projet de Maeda.

- http://reas.com/ et http://benfry.com/

Présentation de Processing

- techniquement, c’est une simplification du language **JAVA**.

- peut être utilisé comme un point d’entrée vers d’autres languages plus complexes, mais peut se suffire à lui-même.

Qui fait quoi, avec Processing ?

www.processing.org

exploration de la rubrique exhibition

- **COP15 generative identity** ► logo génératif
- **Mud Tub** ► interface homme/machine en argile
- **Platonic Solids** ► images 3D
- **Body Navigation** ► installation pour danseurs
- **News Knitter** ► broderie générative
- **Nervous System, Play + Learn** ► design
- **Similar Diversity** ► visualisation de données
- **PixelRoller** ► installation/performance
- **Shadow Monsters** ► installation
- **Flight Patterns** ► visualisation de données / expérimentation

- **Flight 404 (Robert Hodgin)** ► vidéos, installations • http://roberthodgin.com/
- **Toxi (Karsten Schmidt)** ► graphisme, installation, sculpture, video • http://postspectacular.com/

Extensions de Processing

- Différents projets sont étroitement associés à Processing

Arduino ► contrôleur open hardware permettant de relier l'ordinateur au monde physique (capteurs de température, de pression, boutons, écrans lcd...)
Wiring ► la même chose, plus puissant
P5 Mobile ► Processing sur téléphone mobiles
portages dans d'autres languages ou sur d'autres supports. (parfois au stade très expérimental)
Androïd
JavaScript
AS3
Scala
Ruby
et même une tentative de portage vers **l'iphone**...

- Exemples de réalisations avec Arduino: http://www.arduino.cc/playground/Projects/ArduinoUsers

Hello World

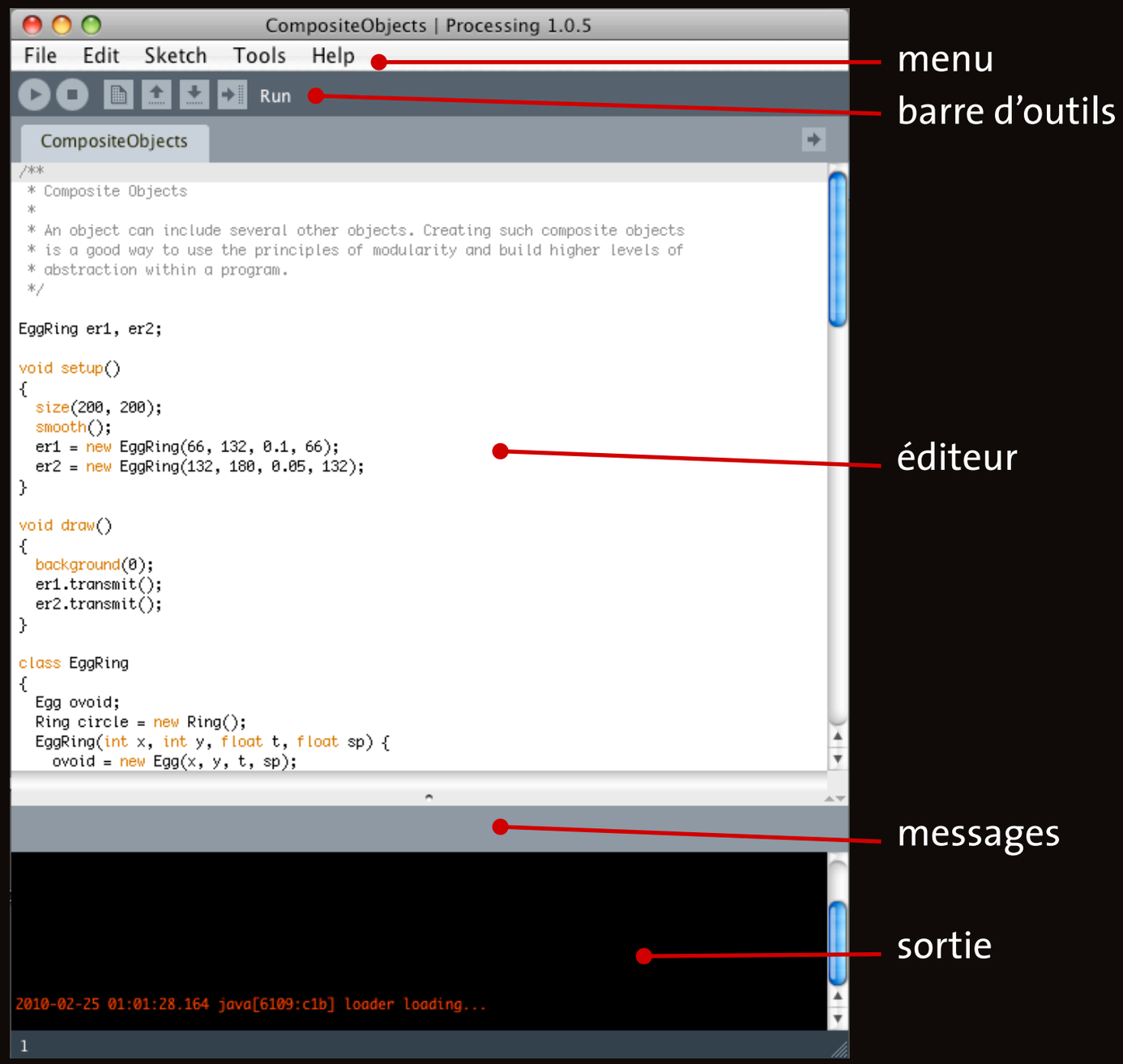
- installons Processing http://processing.org/download/

- ouvrons (enfin) Processing !

- l'interface est réduite au stricte nécessaire.

Hello World

tour de l'interface



Hello World

- un programme, dans P5, est appelé un **sketch**

- **notre premier sketch: hello world!** (pour d'autres languages, le hello world affiche du texte, comme P5 est orienté image, on va afficher une ligne).

```
line(15, 25, 70, 90);
```

- Cliquez sur 

Hello World

- changeons la taille et les couleurs:

```
size(400, 400);
background(192, 64, 0);
stroke(255);
line(150, 25, 270, 350);
```

- Testons, puis ajoutons quelques commentaires:

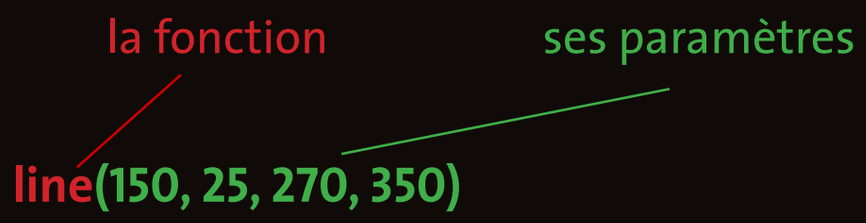
```
// taille de l'image
size(400, 400);
//couleur du fond
background(192, 64, 0);
//couleur des lignes
stroke(255);
//tracer une ligne
line(150, 25, 270, 350);
```

Hello World

```
//
/* */
```

commentaires ignorés par l’ordinateur. Indispensables pour s’y retrover dans ses sketchs.

Hello World



les fonctions nous permettent d’exécuter des actions (dessiner des formes, définir des couleurs, calculer des nombres,...). Une fonction est en général écrite en bas de casse.
Les infos séparées par une virgule sont les paramètres de cette fonction.

Hello World

; (le point virgule)

Le point virgule est un séparateur, il permet de faire comprendre à l'ordinateur que l'on passe d'une ligne d'instruction à l'autre.
L'oubli d'un point virgule déclenchera l'apparition d'un message d'erreur...

Hello World

- Attention, sensibilité à la casse.

- Les espaces n’ont pas d’importance.

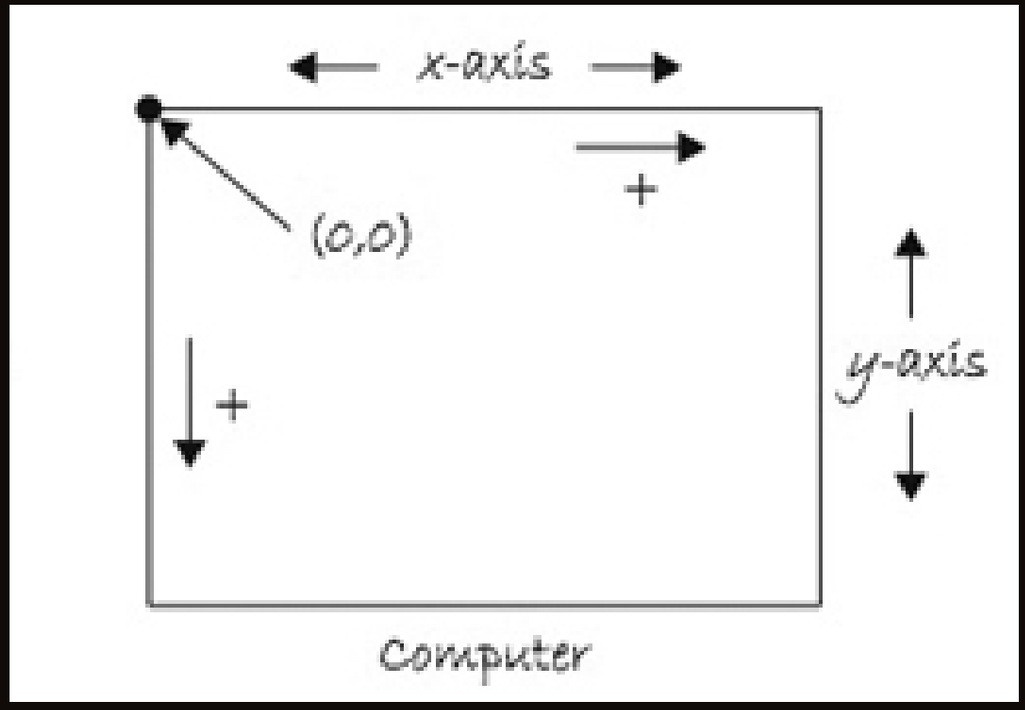
- La console nous permet de comprendre ce qui se passe pendant l’exécution du programme, grâce aux fonctions:

```
print(  )
println(  )
```

Hello World

Se repérer dans l'espace 2D

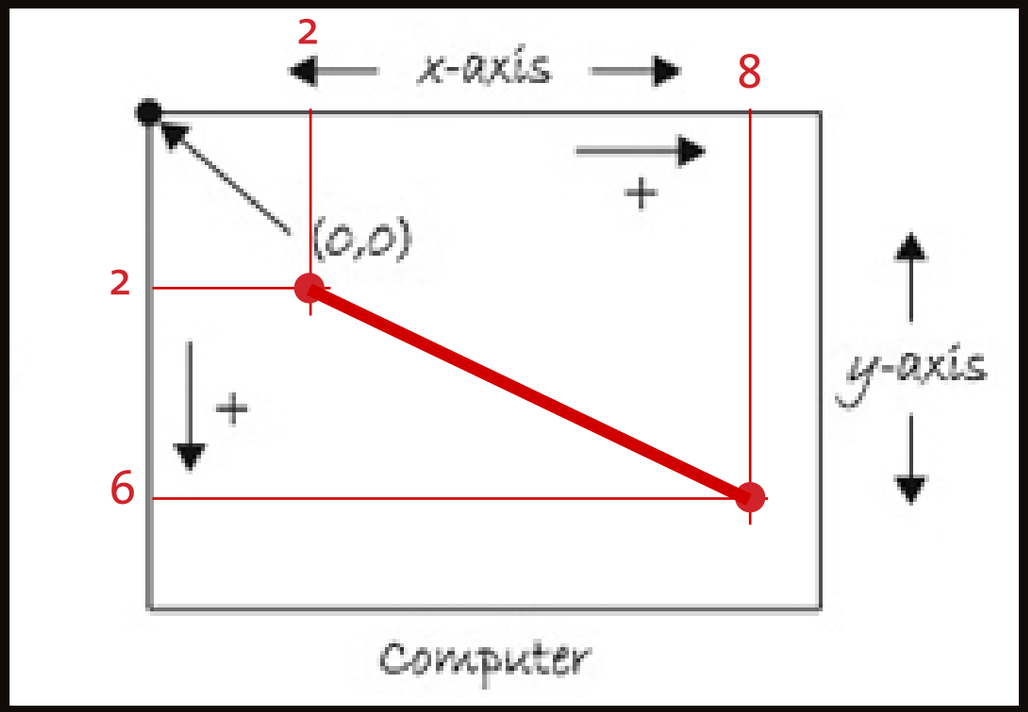
- la plus petite unité d'un écran => le pixel
c'est donc l'unité qui nous servira à définir des points dans l'espace de l'écran,
selon le schéma ci-dessous:



Hello World

Se repérer dans l'espace 2D

- pour tracer une ligne, il faut donc relier deux points:

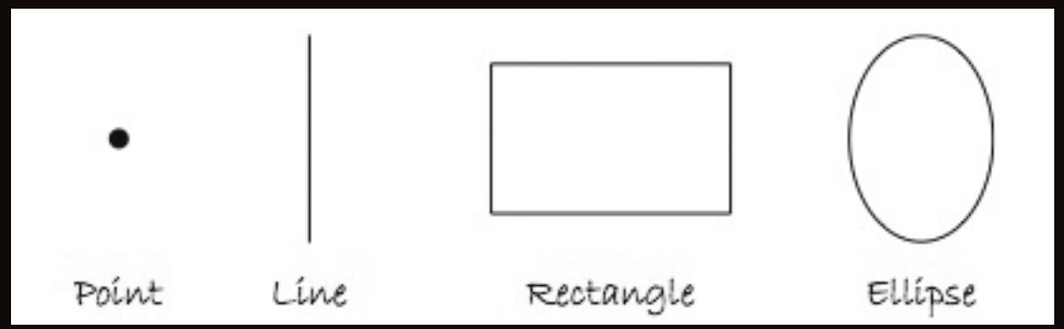


- en code, ça donne `line (2,2,8,6);`

Hello World

Se repérer dans l'espace 2D

- les primitives de base disponibles:



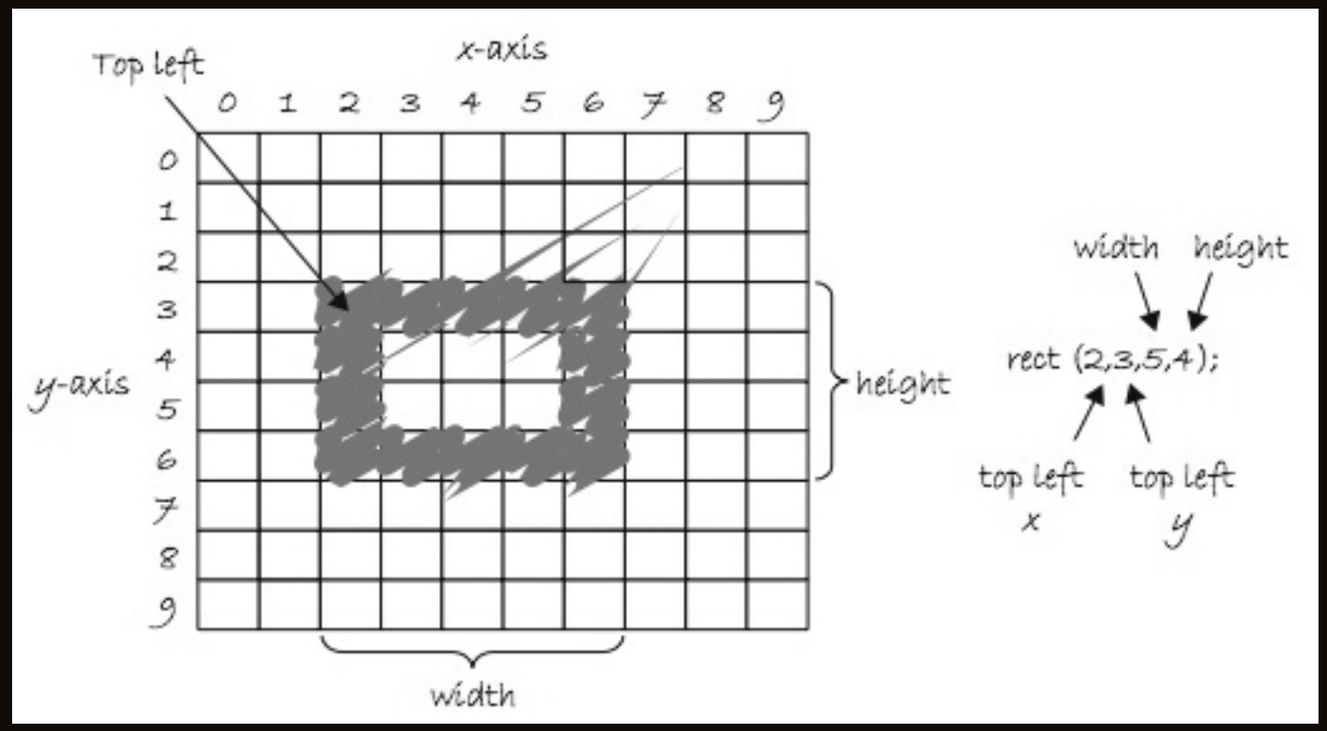
- il en existe d'autres...

Hello World

Se repérer dans l'espace 2D

- les modes de dessin du rectangle

- le mode par défaut CORNER: on définit la position du coin supérieur gauche, puis la largeur, puis la hauteur.

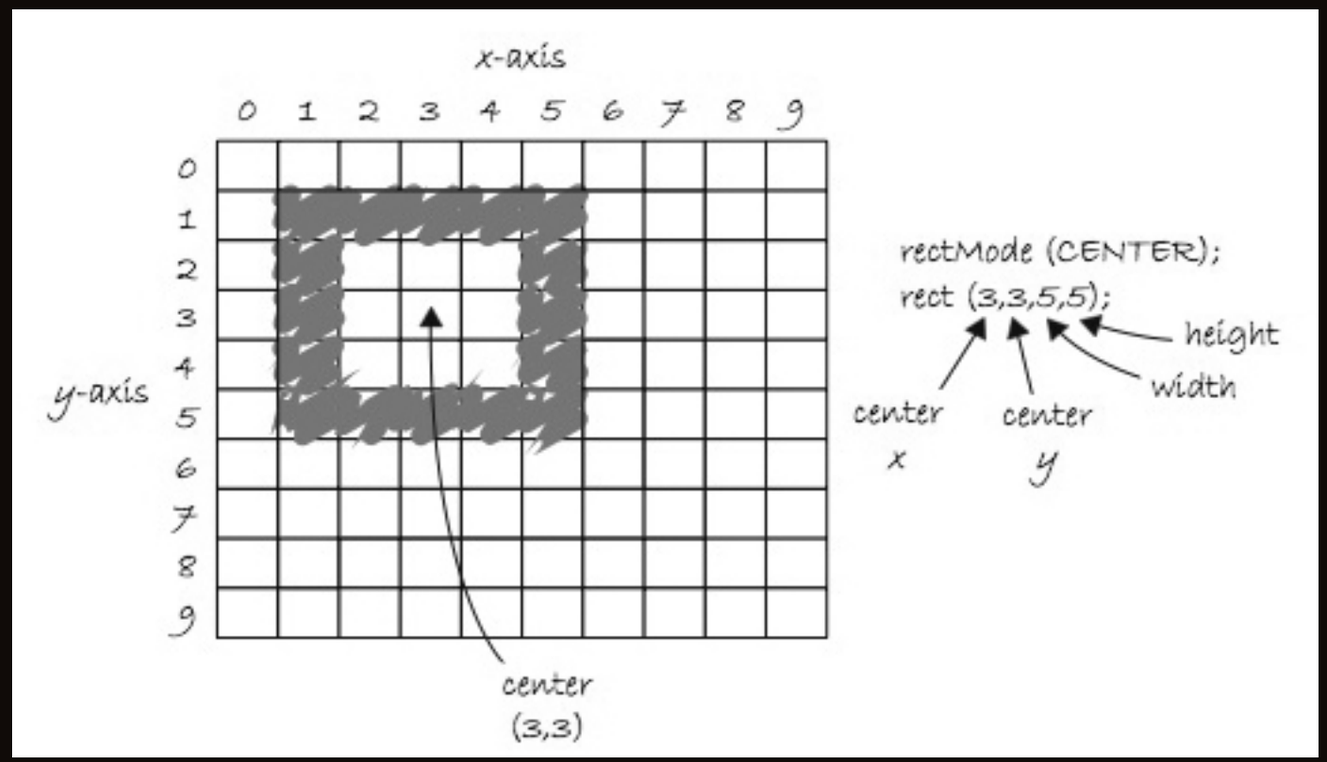


Hello World

Se repérer dans l'espace 2D

- les modes de dessin du rectangle

- le mode CENTER: on définit le centre, puis la largeur, puis la hauteur.

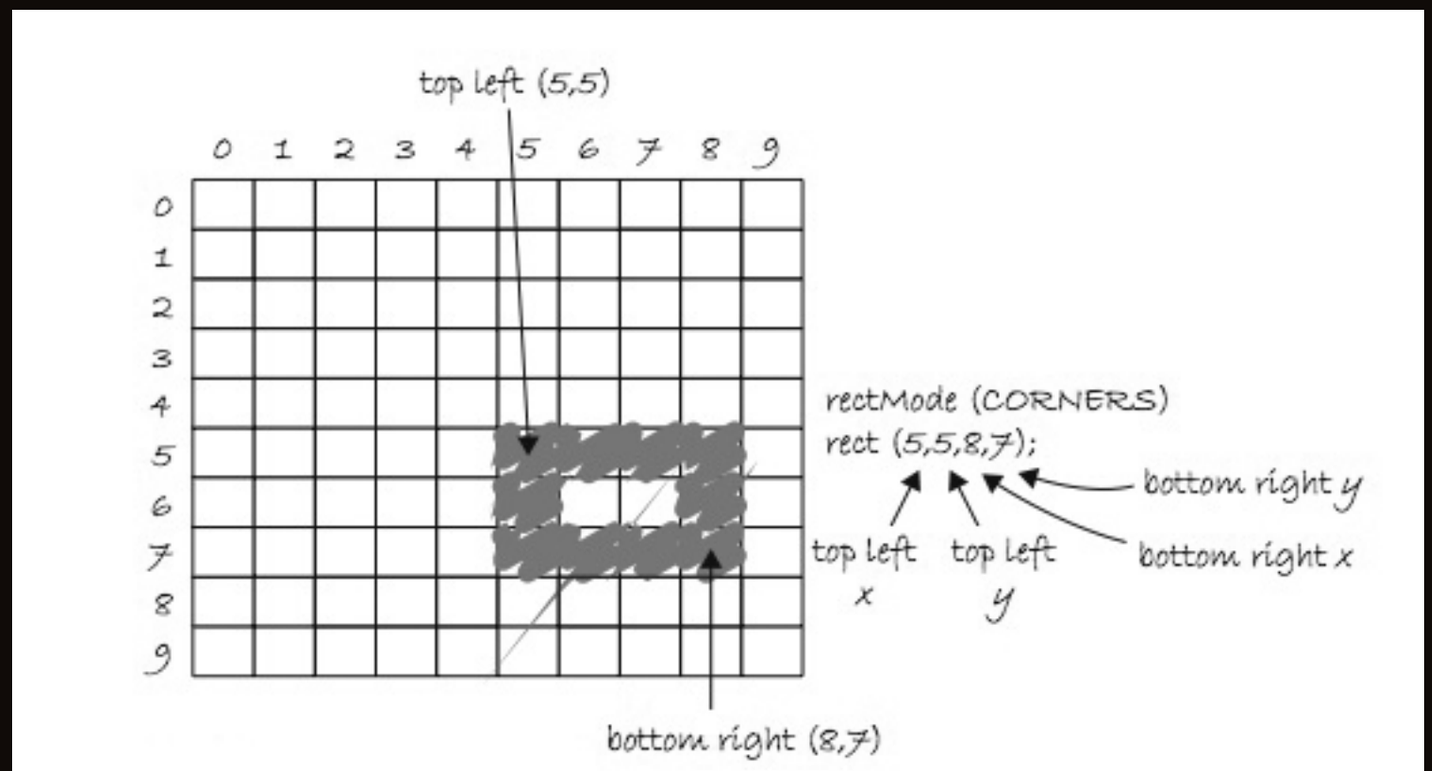


Hello World

Se repérer dans l'espace 2D

- les modes de dessin du rectangle

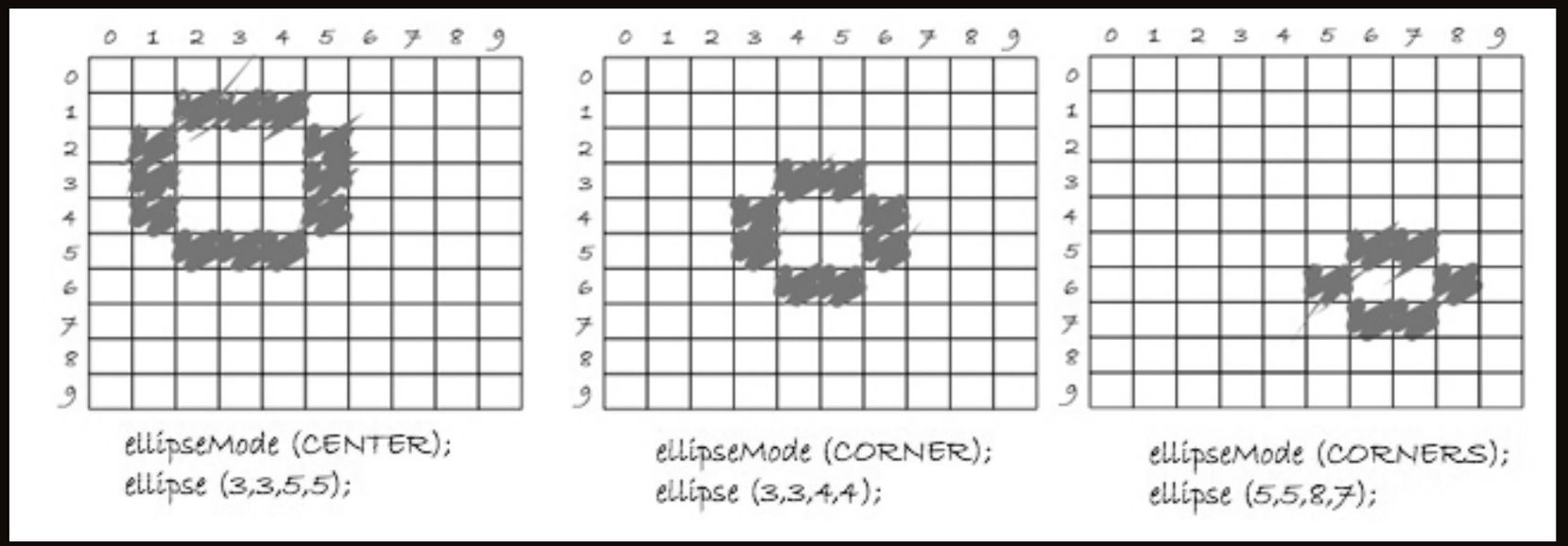
- le mode CORNERS: on définit le coin supérieur gauche, puis le coin inférieur droit.



Hello World

Se repérer dans l'espace 2D

- les modes de dessin de l'ellipse sont les mêmes que pour le rectangle, sauf que le mode CENTER est celui par défaut.



Hello World

Se repérer dans l'espace 2D

- Comment savoir tout cela ?
-> Il faut aller voir dans la référence des fonctions de Processing,
sur le site ou dans le menu du logiciel.

Hello World

Changer les couleurs

- les niveaux de gris sont définis sur 256 niveaux.
0 = noir
255 = blanc

- chaque forme que nous avons dessiné peut avoir deux paramètres «visuels»
stroke() -> le contour
fill() -> le remplissage

- on ajoute le paramètre de couleur entre parenthèses, fill(210);

Hello World

Changer les couleurs

- les modes de dessin se définissent AVANT de dessiner la forme:

```
//fond gris
size(250,250);
background (125);

// contour noir et fond blanc
stroke(0);
fill(255);
rect(20,20,40,80);

// contour blanc et fond noir
stroke(255);
fill(0);
rect(80,20,40,80);
```

Hello World

Changer les couleurs

- pas de contours: `noStroke();`

- pas de remplissage: `noFill();`

- évidemment, attention de ne pas utiliser les deux en même temps! Rien ne serait affiché.

Hello World

Changer les couleurs

- le mode RVB
-> chaque couleur, en mode RVB, est une addition de rouge, de vert et de bleu, dans des proportions différentes.
-> comme en niveaux de gris, il y a 256 variations possibles pour chaque couleur.

Un rouge lumineux: `fill (255,0,0);`
Un rouge sombre: `fill (40,20,16);`
Un jaune: `fill (255,255,0);`

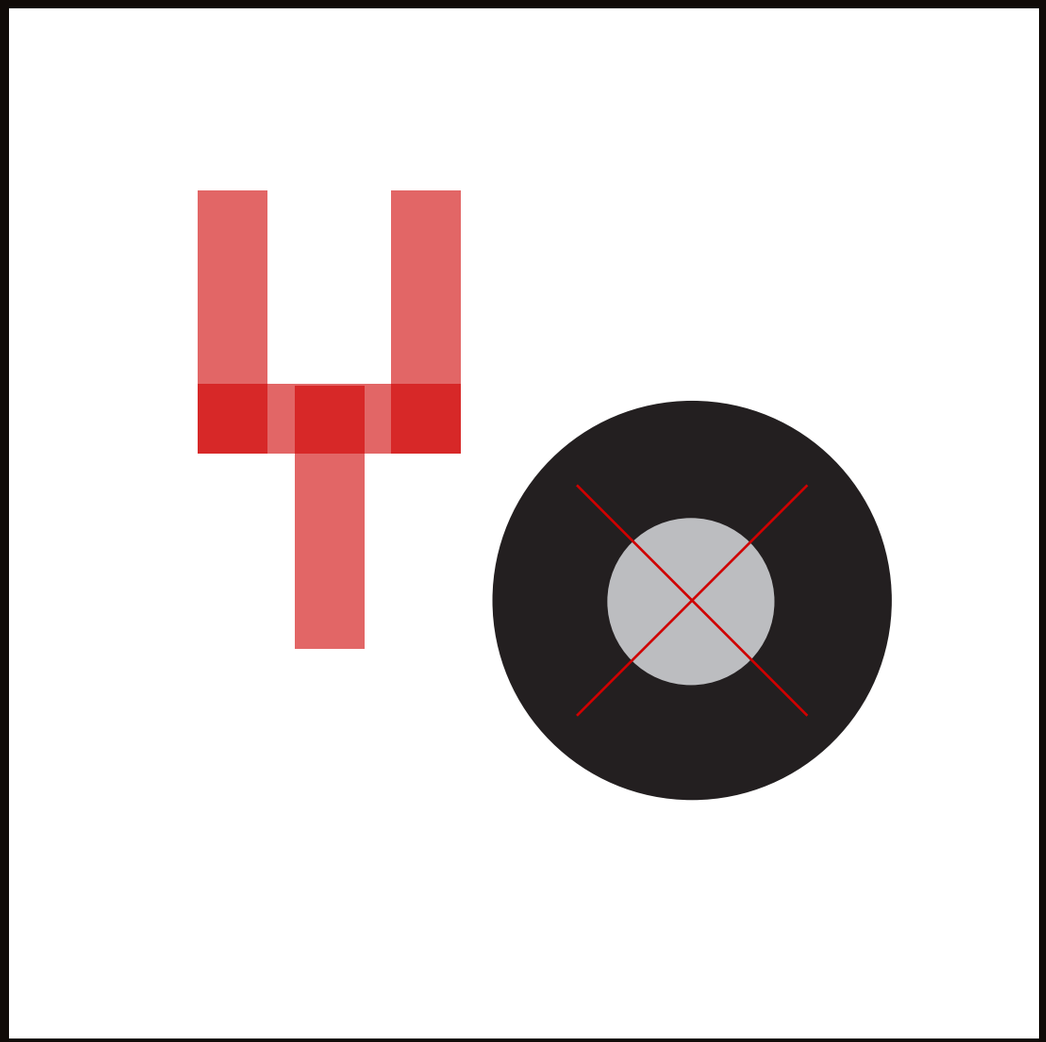
- un sélecteur de couleur dans Processing nous facilitera le choix des couleurs.

- un quatrième paramètre peut être ajouté, il définira une valeur d'alpha, de transparence
0 -> transparence totale , 127 -> transparence à 50% , 255-> opacité totale
`fill (125,20,125,200);`

Hello World

Exercice

- reproduire ce message de paix, ou un truc ressemblant:



Hello World

Publier son sketch

- lorsque le sketch est terminé, on peut désormais le publier,

-> soit comme un **applet**, un programme qui pourra être lu dans un navigateur (menu File>Export)

-> soit comme une application autonome, que l'on pourra lancer depuis le bureau (menu File>Export Application)

Interaction

- qui dit interaction, dit ajout d'une **dimension temporelle**

- intéressons nous donc au flux d'un programme tel que nous allons le créer avec P5

- prenons l'exemple d'un jeu vidéo

    -> le jeu commence: vous donnez un nom à votre personnage, le score commence à 0, et vous commencez au niveau 1. C'est l'**initialisation** du programme.

    -> une fois que ces conditions sont initialisées, vous commencez à jouer. L'ordinateur vérifie à chaque instant la position de la souris, calcule le comportement approprié des joueurs, et met à jour l'affichage à l'écran. Ce cycle de calculs et de dessins tourne en boucle, idéalement 30 fois par seconde, ou plus, pour une animation fluide. C'est la phase de **dessin** du programme.

Interaction

- Résumons :

Étape 1 - Définition des conditions de départ pour le programme (1 fois).

Étape 2 - Faire quelquechose encore, et encore, et encore, et encore... jusqu'à ce que le programme quitte.

Si je pars faire un footing:

Étape 1 - Mettre ses baskets et son short. On ne fait cela qu'une fois...

Étape 2 - Un pied devant l'autre, puis l'autre devant celui d'avant, et ainsi de suite...

Interaction

- Dans Processing, nous allons séparer ces deux étapes à l'aide de **blocs de code**.

- un bloc de code, c'est simplement du code inclus dans des accolades.

```
{
un bloc de code;
on peut en mettre autant que l'on veut;
tout ça est regroupé dans ces accolades;
}
```

on peut les imbriquer

```
{
un bloc de code;
      {
      un autre bloc de code à l'intérieur;
      }
}
```

Interaction

```
void setup () {
      // l'initialisation
      // étape 1
      // étape 2
      // étape 3
}
```

effectué une seule fois

lorsque l'éxecution du setup() est terminée, P5 passe au draw()

```
void draw () {
      // le code qui va se répéter
      // étape 1
      // étape 2
      // étape 3
}
```

boucle jusqu'à ce qu'on quitte le programme

Interaction

Mis en application

Nous voulons attacher un carré noir de 10 pixels de côté au curseur de notre souris.
En bon français informatique, nous voulons que les coordonnées X et Y du centre de notre carré soient toujours égales à celles de notre curseur.
Quand je dis TOUJOURS, je signifie que cette information doit être mise à jour dans le draw().

J'aurai donc, dans le draw(), une ligne qui créera un carré selon cette formule:

```
rectMode(CENTER);
rect( la position X de la souris, la position Y de la souris, 10,10);
```

Il se trouve qu'en language Processing, la position de la souris est renvoyée par les fonctions `mouseX` et `mouseY`.

Interaction

```
void setup(){
	size(200,200);
}

void draw(){
	background (255);

	noStroke();
	fill(0);
	rectMode(CENTER);
	rect (mouseX, mouseY, 10,10);

}
```

Faisons maintenant un test en déplaçant le background() dans le setup().

Interaction

- Reprenons notre YO en lui faisant suivre les mouvements de la souris...

- Arrangeons nous, maintenant, pour que les couleurs changent en fonction de la position de la souris... (idée: travailler dans un sketch de 255 pixels de côtés...)

Interaction

- Nous avons vu que `mouseX` et `mouseY` renvoyaient les valeurs X et Y de notre curseur.

- Nous avons également à disposition les fonctions `pmouseX` et `pmouseY` .
Elle renvoient la position de la souris **à l'image précédente**.

- avec ces deux nouvelles fonctions, nous pouvons facilement créer une application de dessin...

```
void setup(){
	size (200,200);
	background (255);
	//lissage des graphiques
	smooth();
}

void draw(){
	stroke(0);
	line (pmouseX, pmouseY, mouseX, mouseY);
}
```

Interaction

Détecter les clic de souris et les touches du clavier

- nous aurons besoin de deux nouvelles fonctions, qui détecterons ces **événements:**
mousePressed() -> détecte les clics de souris
keyPressed() -> détecte un appui sur une touche

- ce que nous allons mettre entre les accolades sera executé une fois et une seule, à chaque fois que l'évenement sera exécuté.

Interaction

```
void setup(){
	size (200,200);
	background (255);
}

void draw(){
}

void mousePressed(){
	stroke(0);
	fill(175);
	rectMode(CENTER);
	rect (mouseX, mouseY, 16,16);
}

void keyPressed(){
	background (255);
}
```

Interaction

- une dernière fonction avant de clore ce chapitre:
frameRate();

- cette fonction, à poser dans le setup(), sert à définir **une vitesse de rafraîchissement**.
- la boucle draw() sera donc executée autant de fois par secondes que le chiffre indiqué entre parenthèses.

- c'est une valeur maximum. Si vous essayez de dessiner un million de triangles à chaque image, elle mettra beaucoup plus de temps que prévu pour être dessinée.

- si frameRate() n'est pas défini, sa valeur par défaut est de 60 images par secondes.

```
void setup(){
	size (200,200);
	smooth();
	frameRate (30);
}
```

VARIABLES

Qu'est ce qu'une variable?

- Une variable, c'est un panier. Vous mettez quelque chose à l'intérieur, vous l'amenez quelque part, vous vous servez de ce que vous avez mis à l'intérieur.

- Une variable, c'est une consigne atomatique. Vous stockez quelque chose à l'intérieur, et vous venez le chercher quand vous en avez besoin.

- Une variable, c'est un post-it, il est noté dessus «je suis une variable, écrivez une information sur moi».

Daniel Shiffman

VARIABLES

Qu'est ce qu'une variable?

- L'ordinateur a une mémoire. Il va donc pouvoir retenir les informations dont on va avoir besoin. Une variable, c'est un nom qui pointera vers un espace de la mémoire de l'ordinateur, là où la variable sera stockée.

- Une variable pourra contenir toutes les informations dont nous aurons besoin pour dessiner des formes, par exemple: couleur, taille, position...

- Une variable est une référence textuelle a une valeur, qui pourra changer au fur et à mesure du programme.

Ex: lors d'un jeu de scrabble entre Jane et Billy.
Au début du jeu, les variables scoreBilly = 0 et scoreJane = 0.
À la fin du jeu, scoreBilly = 124 et scoreJane = 136.

- Dans cette exemple, la variable scoreBilly a un **NOM** (scoreBilly) et une **VALEUR**, évolutive. Cette valeur est d'un certain **TYPE** (un nombre).

VARIABLES

Déclarer et initialiser des variables

- Les variables sont déclarées en donnant d'abord le **TYPE**, puis son **NOM**
(c'est à dire que l'on réserve une partie de la mémoire de l'ordinateur,
et on lui donne un nom pour la retrouver).

- Le nom des variables doit commencer par une lettre. Il peut contenir un chiffre, mais pas en premier.

- Pas de ponctuation, pas de caractères spéciaux, sauf l'underscore _
Une convention de nommage fait que l'on commence toujours les variables par une minuscule,
et on sépare les mots avec des majuscules.
maSuperBelleVariable

VARIABLES

Déclarer et initialiser des variables

- Le type est la sorte de données qui va être stockée dans la variable. Cela peut être un nombre entier, un décimal, un ou des caractères.

Les types que l'on utilisera le plus souvent:

• Nombre entiers: 0, 1, 2, -125 ...
ils sont de type «integers» : on notera int dans Processing

• Nombres décimaux: 3,14159 ou 2,5 ou -9,95
ils sont de type «floating point values»: float

• Caractères: a, b ou c seront de type char
un mot (plusieurs caractères) sera de type «string», une chaîne de caractères: string

Variables

Déclarer et initialiser des variables

Une fois qu'une variable est déclarée (et seulement une fois que c'est le cas), on peut lui assigner une valeur.

```
int compteur;
compteur = 50;
```

On peut être plus concis:

```
int compteur = 50;
```

Variables

Déclarer et initialiser des variables

Une fois qu'une variable est déclarée (et seulement une fois que c'est le cas), on peut lui assigner une valeur.

```
int compteur;
compteur = 50;
```

On peut être plus concis:

```
int compteur = 50;
```

VARIABLES

Exemples de déclarations de variables

```
int compteur = 0;
char lettre = ‘a’;
boolean happy = false;
float x = 4.0;
y = x +5.2;
float z = x*y + 15.0;
```

VARIABLES

Exemples de déclarations de variables

Commençons par cet exemple simple:

```
void setup(){
      size (200,200);
}

void draw(){
      background(255);
      stroke (0);
      fill(175);
      ellipse (100, 100,50,50);

}
```

VARIABLES

Exemples de déclarations de variables

Nous déclarons deux variables qui viennent remplacer les valeurs que nous avions posées.

```
int circleX = 100;
int circleY = 100;

void setup(){
	size (200,200);
}

void draw(){
	background(255);
	stroke (0);
	fill(175);
	ellipse (circleX, circleY,50,50);

}
```

VARIABLES

Exemples de déclarations de variables

Faisons maintenant varier une des valeurs au fur et à mesure du draw().

```
int circleX = 0;
int circleY = 100;

void setup(){
	size (200,200);
}

void draw(){
	background(255);
	stroke (0);
	fill(175);
	ellipse (circleX, circleY,50,50);
	circleX = circleX + 1;
}
```

VARIABLES

Exemples de déclarations de variables

Que modifier dans le code précédent pour que le disque ne se déplace plus mais grossisse?

VARIABLES

Les variables système

Ce sont des variables prédéfinies qui n'on pas besoin d'être déclarées avant d'être utilisées.
Les plus courantes:

mouseX et mouseY
width -> la largeur de la fenêtre du sketch
height -> la hauteur de la fenêtre du sketch
frameCount -> le nombre d'images affichées par Processing depuis le début du sketch
frameRate - > fréquence de rafraichissement
screen.width -> la largeur totale de l'écran
screen.height -> la hauteur totale de l'écran
key -> la dernière touche pressée du clavier
keyPressed -> true ou false? une touche est-elle pressée?
mousePressed -> true ou false? un bouton de la souris est il pressé?
mouseButton -> quel boutton de la souris est pressé?

VARIABLES

Les variables système

```
void setup(){
	size (200,200);
}

void draw(){
	background(100);
	stroke (255);
	fill(175);
	rectMode (CENTER);
	rect (width/2, height/2, mouseX+10, mouseY+10);
}

void keyPressed(){
	println(key);
}
```

VARIABLES

Aléatoire

Nous allons maintenant modifier nos variables avec des chiffre aléatoires.
Commençons par écrire un script simple, nous ajouterons de nouvelles fonctionnalités ensuite:

```
float r = 100;
float g = 150;
float b = 200;
float a = 200;

float diam = 20;
float x = 100;
float y = 100;

void setup(){
        size (200,200);
        background (255);
        smooth();
}
(...la suite...)
```

VARIABLES

Aléatoire

```
void draw(){
        noStroke();
        fill (r,g,b,a);
        ellipse(x,y,diam,diam);
}
```

Vous voyez que tout, ici, est stocké dans des variables...
La fonction random() est un fonction spéciale: elle renvoie une valeur.
Elle renvoie une valeur float au hasard, entre deux nombres définis.

```
float w = random(1,100);
```

w peut être égal à 63.0, ou 12.0, 25.0...

Pour convertir un float en int:

```
int w = int(random(1,100));
```

VARIABLES

Aléatoire

Modifions donc le code précédent:

```
float r;
float g;
float b;
float a;

float diam;
float x;
float y;

void setup(){
        size (200,200);
        background (255);
        smooth();
}
(...la suite...)
```

VARIABLES

Aléatoire

```
void draw(){
        r = random(0,255);
        g = random(0,255);
        b = random(0,255);
        a = random(0,255);
        diam = random (0,20);
        x = random(width);
        y = random(height);
        noStroke();
        fill (r,g,b,a);
        ellipse(x,y,diam,diam);
}
```

Conditions

Les expressions booléennes

- Dans le monde de l'informatique, il n'existe qu'une seule sorte de test: le lest **booléen** - vrai ou faux. Une expression booléenne (définie par le mathématicien George Boole) est une expression qui ne peut avoir comme valeur que **vrai** ou **faux**.

J'ai faim -> vrai
La programmation me fait peur -> faux
Ce workshop est hilarant -> faux

En informatique, on va pouvoir tester des relations entre des chiffres:

15 est plus grand que 20 -> faux
5 égal 5 -> vrai
32 est plus petit ou égal à 33 -> vrai

Conditions

Les expressions booléennes

- Nous allons voir comment tester la valeur de variables et éxecuter des actions différentes suivant le résultat.

`x>20` -> dépendra de la valeur de x
`y==5` -> dépendra de la valeur de y
`z<=33` -> dépendra de la valeur de z

Pour cela, nous pourrons utiliser les opératuers suivants:

>	plus grand
<	plus petit
>=	plus grand ou égal
<=	plus petit ou égal
==	egalité
!=	inégalité

Conditions

If, Else, Else If

- Grâce aux opérations booléennes, nous allons pouvoir définir des conditions selon lesquelles certaines actions seront éxecutées, ou non.
La syntaxe est la suivante:

```
if (condition){
      action à éxecuter
}
```

Transcrivons:
Si la souris est dans la partie gauche de l'écran, dessine un rectangle sur la partie gauche du sketch.

```
if (mouseX < width/2){
      fill(255);
      rect (0,0, width/2, height);
}
```

Conditions

If, Else, Else If

- Si la condition n’est pas remplie, nous pouvons ajouter une nouvelle action à éxecuter.
(si ..... fais ceci, sinon, fais cela)

Si la souris est dans la partie gauche de l’écran, le fond est blanc, sinon, le fond est noir.

```
if (mouseX < width/2){
        background (255);
} else {
        background (0);
}
```

Conditions

If, Else, Else If

- Enfin, nous pouvons tester de multiples conditions avec else if

```
if (condition 1 = true){
        // code à éxecuter si la condition 1 est vérifiée
} else if (condition 2 = true){
        // code à éxecuter si la condition 2 est vérifiée
} if (condition 3 = true){
        // code à éxecuter si la condition 3 est vérifiée
} else {
        // code à éxecuter si aucune condition n'est vérifiée
}
```

Conditions

Opérateurs logiques

- Parfois, nous voudrons tester plusieurs conditions en même temps.

Si j'ai 40° de fièvre OU si j'ai le bras cassé, je vais voir le docteur.
Si je me fais piquer par une abeille ET que je suis allergique, je vais voir le docteur.

- Nous aurons souvent besoin de ces opérateurs logiques en programmation:

Si le curseur est en bas de l'écran ET à droite de l'écran, dessiner un carré en bas et à droite de l'écran.

&& -> ET logique
|| -> OU logique
nous avons aussi le N'EST PAS (NOT):
! -> NOT logique

Conditions

Mise en pratique de toutes ces notions

```
void setup(){
	size (200,200);
}
void draw(){
	background(255);
	stroke(0);
	line (100,0,100,200);
	line(0,100,200,100);
	noStroke();
	fill(0);

	if (mouseX <100 && mouseY < 100){
		rect(0,0,100,100);
	} else if (mouseX >100 && mouseY < 100){
		rect(100,0,100,100);
	} else if (mouseX <100 && mouseY > 100){
		rect(0,100,100,100);
	} else if (mouseX >100 && mouseY > 100){
		rect(100,100,100,100);
	}
```

Conditions

Les variables booléennes

- Une variable booléenne est une variable qui ne peut être que vraie ou fausse (true/false).
Cela va être très utile, pour, par exemple, créer un bouton commutateur.

```
boolean bouton = false;

void setup(){
	size (200,200);
}

void draw(){
	background (130);
	if (bouton == true){
		noStroke();
		fill(255);
		rect (50,50,100,100);
	} else {
		noStroke();
		fill(0);
		rect (50,50,100,100);
	}
}  (...la suite...)
```

Conditions

Les variables booléennes

```
void mousePressed(){
	if (bouton == false){
		bouton = true;
	} else {
		bouton = false;
	}
}
```

Boucles

Itération (?????)

- Une itération, c'est un processus répétitif, qui se répète encore et encore, suivant des règles définies.

- imaginons que nous voulions dessiner 100 lignes, espacées de 2 pixels. Nous pouvons réaliser cela en écrvivant 100 lignes de code. Avec des variables, nous n'avons plus à calculer, mais nous aurions 200 lignes de code:

```
int y = 80;
int x = 50;
int espace = 2;
int taille = 30;

line (x,y,x, y+taille);

x = x+espace;
line (x,y,x,y+taille); // et cela 100 fois de suite
```

Boucles

- Plutôt que de dessiner 100 fois une ligne, il serait plus facile de dessiner une fois 100 lignes!
- Ce problème sera résolu grâce aux boucles. La syntaxe est très proche des conditions.
Il existe 3 sorte de boucles, **while**, **for**, **do-while**. Do-while n'est quasiement jamais utilisée.

- Comme les conditions, while utilise une condition booléenne comme test. Si le test est true, les instructions sont executées. S'il est faux, on passe à la suite. La différence avec la condition est que les instructions dans le bloc while continuent jusqu'à ce que le test soit false.

```
size (200,200);
int y = 20;
int x = 0;
int espace = 2;
int taille = 160;
int end = 200;

stroke(0);
while (x<=end){
      line (x,y,x,y+taille);
      x = x+espace;
}
```

Boucles

Autre exemple

```
size (200,200);

int y = 10;

while (y < height){
      rect (50,y,100,10);
      y = y +20;
}
```

Attention aux boucles infinies, lorsque la condition ne se réalise jamais!

Boucles

- La boucle for est assez proche de la boucle while, mais elle fonctionne, elle,
en incrémentant une valeur.

- Il y a trois étapes pour démarrer une boucle for:
    - L'initialisation. Une variable est déclarée, elle sera utilisée uniquement à l'intérieur de la boucle.
    - Le test booléen. La boucle s'éxecutera tant que ce test renverra true.
    - L'itération. Un événement qui se déclenchera à chaque boucle.
    Par exemple: augmenter ma variable de 1.

```
for (int i=0 ; i < 10 ; i = i+1){
    println (i);
}
```

Boucles

- Réécrivons l'exemple précédent dans une boucle for:

```
size (200,200);
int y = 20;
int x = 0;
int espace = 2;
int taille = 160;
int end = 200;

stroke(0);
for (int i =0; i < 200; i = i+espace){
        line (x+i, y , x+i ,y+taille);
}
```

Boucles

- Raccourcis de programmation pour incrémenter ou décrémenter

```
X++;          ->          X = X+1;

X--;          ->          X = X-1;

X+= 2;        ->          X = X+2;

X*=3;         ->          X = X*3;
```

Fonctions

- il est impossible de faire tenir tout le code d'un long programme à l'intérieur de la seule fonction draw().

- pour organiser, simplifier, découper le code, nous allons pouvoir créer des fonctions personnalisées.

- fonctionnement identique aux fonctions prédéfinies draw() ou setup().

- cela va ressembler à cela:

```
typeRetourné functionName (arguments) {
    //corps de la fonction;
}

void drawBlackCircle() {
    fill(0);
    ellipse(50,50,20,20);
}
```

Fonctions

Exemple sans fonctions:

```
int x = 0;
int speed = 1;

void setup(){
	size (200,200);
	smooth();
}

void draw(){
	background (255);
	x = x+speed;
	if (( x>width ) || (x<0)) {
		speed = speed*-1;
	}
	stroke (0);
	fill (175);
	ellipse (x,100,32,32);
}
```

Fonctions

Exemple avec fonctions:

```
int x = 0;
int speed = 1;

void setup(){
	size (200,200);
	smooth();
}

void draw(){
	background (255);
	move();
	bounce();
	display();
}


void move(){
	x = x+speed;
}

void bounce(){
	if (( x>width ) || (x<0)) {
		speed = speed*-1;
	}
}

void display() {
	stroke (0);
	fill (175);
	ellipse (x,100,32,32);
}
```

Du texte et des images

- le texte, dans Processing, est une variable de type String.

- on utilise les guillemets pour définir une variable string

- pour afficher du texte:
    - Tools>Create font .
    - Déclarer un objet de type PFont.
    - Charger la typo avec `loadFont();`
    - Spécifier la taille du texte avec `textFont();`
    - spécifier une couleur
    - Appeler la fonction `text();`

Du texte et des images

```
PFont f;

void setup(){
	size (200,200);
	f = loadFont (“ArialMT-16.vlw“);
}

	void draw(){
		background (255);
		textFont (f,16);
		fill(0);
		text (“wow, quelle string“, 10, 100);
	}
```

Du texte et des images

- une image, dans Processing, est une variable de type PImage.

- Les fichiers doivent être stockés dans le dossier data du sketch.
P5 accepte le GIF, le JPG, le TGA, le PNG.

```
PImage img;

void setup(){
	size (320,240);
	img = loadImage (“monimage.jpg“);
}

void draw(){
	background (0);
	image (img, 0,0);
}
```