1-3 ► 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

1-3 ► 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

1-3 ► 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);
}
```

1-3 ► 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);
}
```

1-3 ► 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
}
```

1-3 ► 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

1-3 ► 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);
}
```

1-3 ► 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...)
```

1-3 ► Conditions

Les variables booléennes

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

1-3 ► 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
```

1-3 ► 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;
}
```

1-3 ► 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!

1-3 ► 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);
}
```

1-3 ► 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);
}
```

1-3 ► 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;
```

1-3 ► 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);
}
```

1-3 ► 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);
}
```

1-3 ► 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);
}
```

1-3 ► 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();`

1-3 ► 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);
}
```

1-3 ► 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);
}
```