1-4 ► Advanced 1

Dessiner des formes complexes:
beginShape()
endShape()
vertex()
curveVertex
bezierVertex()

- Pour dessiner des formes différentes des primitives de bases, nous allons utiliser des séries de point que Processing reliera. Chaque point est défini par ses coordonnées x et y:
vertex(x, y)

- Il faut d'abord utiliser la fonction beginShape(), puis spécifier les différents points qui compose-ront la forme, et clôturer la forme par endShape(). Un triangle comptera trois points, un quadrilatère quatre, etc...

- Pour fermer la forme, il faut utiliser le paramètre CLOSE pour endShape():
endShape(CLOSE)

1-4 ► Advanced 1

```
fill(0);
noStroke();
smooth();
beginShape();
vertex(10,0);
vertex(100,30);
vertex(90,70);
vertex(100,70);
vertex(10,90);
vertex(50,40);
enShape();
```

- On peut relier les points dans différents ordres, et ainsi créer différentes sortes de formes.
Voir la doc pour l'ordre des points:

```
beginShape(TRIANGLES)
beginShape(TRIANGLE_STRIP)
beginShape(TRIANGLE_FAN)
beginShape(QUADS)
beginShape(QUAD_STRIP)
```

1-4 ► Advanced 1

- On peut aussi relier les points par des courbes:
curveVertex(x,y)
ou des courbes de Bezier avec c1 et c2 pour les coordonnées des poignées de contrôle :
bezierVertex(cx1, cy1, cx2, cy2, x, y)

- On pourra également dessiner des courbes à l'aide de fonctions mathématiques:
sq()
sqrt()
pow()
norm()
lerp()
map()

1-4 ► Advanced 1

Travailler les images
PImage() > la classe qui gère les bitmaps
loadImage() > charger une image depuis le dossier «data»
image() > afficher l'image > image(name, x, y) ou image(name, x, y, width, height)

tint() > teinte l'image, fonctionne comme fill() ou stroke()
noTint()

```
PImage img;
img = loadImage("arch.jpg");
tint(102); // teinte en gris
image(img,0,0);
noTint(); //Désactive la teinte
image(img, 50, 0);
```

- Pour une image transparente, utiliser la teinte blanche et une valeur d'alpha: tint(255,102)

1-4 ► Advanced 1

Accéder directement aux pixels des images

- On peut accéder directement à la couleur de chaque pixel qui compose une image pour la réutiliser, ou travailler sur le dessin spécifique de cette image.

Pour récupérer la couleur d'un pixel précis: PImage.get(x, y)

```
PImage trees;
void setup(){
	size(100,100);
	noStroke();
	trees = loadImage("arch.jpg");
}
void draw(){
	image(trees, 0, 0);
	color c = get(mouseX, mouseY);
	fill(c);
	rect(50,0,50,100);
}
```

1-4 ► Advanced 1

Il suffit d'inclure get() dans une boucle for pour récupérer la couleur d'une zone de l'image:

```
PImage trees;
int y = 0;
void setup(){
	size(100,100);
	trees = loadImage("arch.jpg");
}
void draw(){
	image(trees, 0, 0);
	y = constrain(mouseY, 0, 99);
	for (int i =0; i<49; i++) {
		color c = get(i,y);
		stroke(c);
		line(i+50,0,i+50,100);
	}
	stroke(255);
	line(0, y, 49 , y);
}
```

1-4 ► Advanced 1

- On peut aussi travailler les pixels de l'image chargée, ou ceux du sketch par l'intermédiaire d'un tableau qui stocke la valeur des couleurs. Pour cela, il faut encadrer pixels[ ] (le tableau qui stocke tout cela) par les fonctions loadPixels() et updatePixels(), si les pixels ont été modifiés, comme avec beginShape() et endShape().

```
void setup() {
 size(100, 100);
}

void draw() {
// Constrain to not exceed the boundary of the array
  int mx = constrain(mouseX, 0, 99);
  int my = constrain(mouseY, 0, 99);
  loadPixels();
  pixels[my*width + mx] = color(0);
  updatePixels();
}
```

1-4 ► Advanced 1

- Afficher un pixel sur deux d'une image

```
PImage arch = loadImage(«arch.jpg»);
int count = arch.width * arch.height;
arch.loadPixels();
loadPixels();
for (int i = 0; i < count; i += 2) {
  pixels[i] = arch.pixels[i];
}
updatePixels();
```

1-4 ► Advanced 1

- Quantifier les pixels rouges d'une ligne d'une image, en récupérant uniquement sa composante rouge par la fonction red()

```
PImage arch;

void setup() {
  size(100, 100);
  arch = loadImage(«arch.jpg»);
  arch.loadPixels();
}

void draw() {
  background(204);
  int my = constrain(mouseY, 0, 99);
  for (int i = 0; i < arch.height; i++) {
    color c = arch.pixels[my*width + i]; // Get a pixel
    float r = red(c); // Get the red value
    line(i, 0, i, height / 2 + r / 6);
  }
}
```

1-4 ► Advanced 1

- Il existe également des fonctions permettant de filtrer > filter() , mélanger > blend() , copier > copy() , ou masquer une image > mask()

Se reporter au exemples Books>Processing Handbooks>Units31-42>39 Image 4
pour voir l'utilisation de ces fonctions et à la doc pour voir tous les paramètres qu'elles acceptent.

```
smooth();
strokeWeight(5);
noFill();
line(0, 30, 100, 60);
filter(BLUR, 3);
line(0, 50, 100, 80);
```

1-4 ► Advanced 1

- Le système de coordonnées x et y où l'origine est dans le coin en haut à gauche de l'écran peut être modifié pour faciliter le dessin et la transformation des formes.

translate() > va déplacer l'origine pour les dessins suivants

rotate() > effectue une rotation des axes x et y avant le dessin (en radians, pas en degrés)

scale() > modifie l'échelle de dessin

Ces 3 modifications sont cumulatives, et peuvent être utilisées conjointement.

Pour modifier temporairement les coordonnées de dessin, utiliser:
pushMatrix() qui stocke les valeurs actuelles de transformation, puis
translate(), rotate(), scale(), et enfin
popMatrix() pour revenir à l'origine initiale.