ICN_processing_cours_1

December 14, 2016

In [3]:
```
%%javascript
IPython.Cell.options_default.cm_config.lineNumbers = true
```

```
<IPython.core.display.Javascript object>
```

In [6]:
```
# Charge ma feuille de style pour nbviewer
from IPython.core.display import display,HTML
from  urllib.request import urlopen
# import urllib.request, urllib.parse, urllib.error

url='https://github.com/debimax/cours-debimax/raw/master/documents/custom.c
with urlopen(url) as response:
    styles = response.read().decode("utf8")
styles="<style>\n{}\n</style>".format(styles)
HTML(styles)
```

Out[6]:
```
<IPython.core.display.HTML object>
```

In [7]:
```
html_template = """
<script type="text/javascript" src="https://raw.githubusercontent.com/debim
<script type="text/javascript">
  var processingCode = `{0}`;
  var myCanvas = document.getElementById("canvas`{1}`");
  var jsCode = Processing.compile(processingCode);
  var processingInstance = new Processing(myCanvas, jsCode);
 </script>
<canvas id="canvas`{1}`"> </canvas>
"""
```

In [8]:
```
#js = "<script>alert('Hello World!');</script>"
#display(HTML(js))
```

1 Processing

Ouvrez processing.

1.1 I bases

1.1.1 1) L'interface

L'interface d'utilisation de Processing est composée de deux fenêtres distinctes : la fenêtre principale dans laquelle vous allez créer votre projet et la fenêtre de visualisation dans laquelle vos créations (dessins, animations, vidéos) apparaissent.

On trouve plus précisément les éléments suivants dans l'interface :

1. Barre d'actions
2. Barre d'onglets
3. Zone d'édition (pour y saisir votre programme)
4. Console (destinée aux tests et messages d'erreur)
5. Fenêtre de visualisation (espace de dessin)
6. Liste déroulante pour les modes

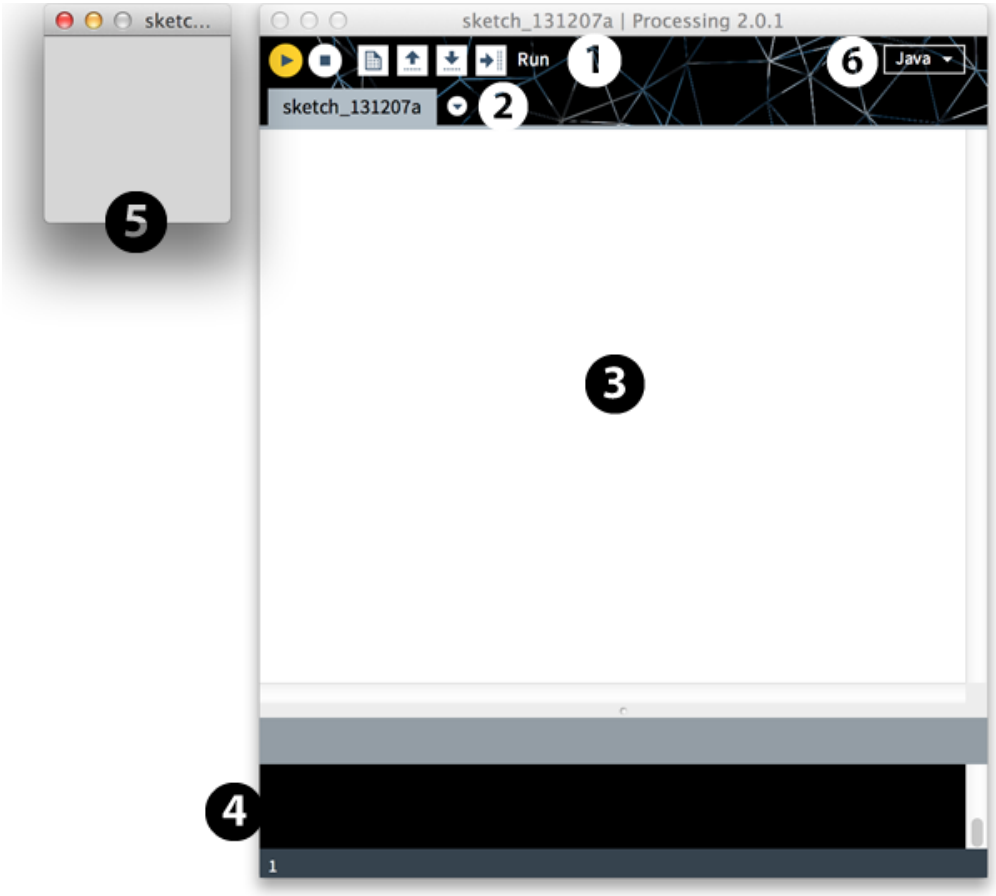


image processing

1.1.2 2) Hello World

On ne va pas y échapper, quand on apprend un langage, le premier programme étudié est le programme qui affiche "Hello World!".

a. avec java Ouvrez processing (mode java) et tapez

```
text("Hello World!", 10, 20);
```

Vous devriez voir s'afficher la petite fenêtre de visualisation par défaut de 100 x 100 pixels, avec le mot << Hello World! >> écrit en blanc :

```
In [9]: processing_code = """
        text("Hello World!", 10, 20);
        """
        html_code = html_template.format(processing_code,"I2a")
        HTML(html_code)

Out[9]: <IPython.core.display.HTML object>
```

b. avec P5.js Ouvrez un nouveau script mode p5.js
Écrire alors

```
function setup() {
    createCanvas(100,100);
}

function draw() {
    text("Hello World!", 10, 20);
}
```

Tester en cliquant sur *run*. C'est cette fois le navigateur qui s'ouvre pour montrer l'image avec le texte.

c. avec python Ouvrez un nouveau script mode python
Écrire alors

```
text("Hello World!", 10, 20)
```

Chacun de ces langages ont des points forts et des faiblesses. Nous étudierons cette année javascript (avec p5.js) et python.

```
In [10]: code = """
         text("Hello World!", 10, 20)
         """
         HTML(html_template.format(code,"I2b"))

Out[10]: <IPython.core.display.HTML object>
```

1.2 II) Espace de dessin

1.2.1 1) fenêtre

L'espace de dessin constitue l'espace de représentation proprement dit. Cette fenêtre de visualisation affichera vos réalisations dans Processing en 2 ou 3 dimensions.

Cet espace est créé par l'instruction size() qui prend deux arguments : ***size(largeur,hauteur);***.

Par exemple, dans la fenêtre d'édition du logiciel Processing, saisissez la commande suivante :

```
size(200, 20)
```

affichera une fenêtre de 200 pixels sur 20pixels.

```
In [11]: code = """
         size(200, 20)
         """
         HTML(html_template.format(code,"II1"))
```

```
Out[11]: <IPython.core.display.HTML object>
```

Au sein d'un programme, on peut connaître à tout moment la taille de l'espace de dessin utilisé au moyen des mots-clés ***width*** et ***height***.
Ces instructions sont très utiles lorsque l'on souhaite notamment dessiner des formes qui puissent s'adapter ultérieurement aux éventuels changements de dimension de la fenêtre de visualisation.

```
In [12]: code = """
         size(300, 50)
         text("largeur:  "+width,10,25)
         text("hauteur: "+height,170,25)
         """
         HTML(html_template.format(code,"II1b"))
```

```
Out[12]: <IPython.core.display.HTML object>
```

1.2.2 2) Le texte

Nous venont donc de voir comment afficher un texte dans un dessin.

Il est possible de modifier ce texte comme - La taille avec la fonction ***textSize()*** - L'alignement du taxte avec ***textAlign()***
On peut aligner au centre (CENTER), à gauche (LEFT) ou à droite (RIGHT). - La fonte du texte. Par exemple on peut utiliser les fontes TrueType (.ttf) et OpenType (.otf).

```
size(300, 100)
textAlign(LEFT)
text("gauche\nga",0,25)              # \n pour un retour à la ligne
textAlign(CENTER)
textSize(14)
text("centre\nce",150,25)
textAlign(RIGHT)
textSize(20)
text("droite\ndr",300,25)
```

alignement-texte

1.2.3 3) La couleur

a. Définition des couleurs Dessiner une image à l'écran, c'est changer la couleur des pixels. Les pixels sont des petites zones, le plus souvent carrées, qui possèdent une couleur.

Chaque couleur se définit par trois canaux qui sont le ***rouge, le vert et le bleu***. On peut aussi utiliser un quatrième canal que l'on appelle canal alpha qui correspond à la transparence.

C'est le codage ***RBV*** (rouge, vert , bleu) ou ***RGB***, (Red, Green, Blue)

2^8=256 qui s'écrit **#FF** en base 16 (hexadécimal). On utilise donc trois (ou quatre) **nombres entiers entre 0 et 256**.

couleur	code RVB	code hexadécimal
Noire	(0,0,0)	#000000
Rouge	(250,0,0)	#FF0000
Vert	(0,250,0)	#00FF00
Bleu	(0,0,250)	#0000FF
Jaune	(250,250,0)	#FFFF00
Violet	(250,0,250)	#FF00FF
cyan	(0,250,250)	#00FFFF
Blanc	(250,250,250)	#00FFFF

Je vous conseille pour obtenir le code d'une couleur d'utiliser le selecteur de couleur (outil->Selecteur de couleur)

On peut aussi utiliser un quatrème code qui est la transparence (canal alpha).
- *(256,0,0,128)* correspond à la couleur rouge avec une transparence de moitié. S'il y a un objet dessous on le vera un peu. - *(256,0,0,0)* correspond à la couleur rouge avec une transparence totale donc on ne voit pas l'objet à l'écran.

```
In [13]: code = """
         size(450, 100)
         background(0,0,256);
         textSize(20)
         fill(256,0,0)
         text("Un texte de couleur rouge",10,30)
         fill(256,0,0,100)
         text("Un texte de couleur rouge",10,70)
         """
         HTML(html_template.format(code,'II3a'))
```

```
Out[13]: <IPython.core.display.HTML object>
```

b) La couleur de fond On peut changer la couleur de fond en appelant la méthode ***background()***.

```
background(0, 256, 256);
```



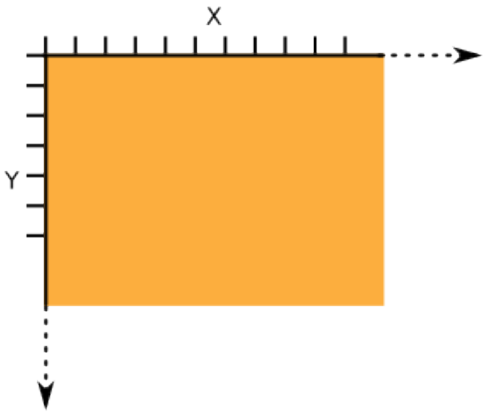
```
In [14]: code = """
         background(0,250,250);
         """
```

1.2.4 4 Les axes

Quand on travaille en 2 dimensions (2D), on utilise deux axes de coordonnées *x* et *y* correspondant respectivement à la largeur (axe horizontal) et à la hauteur (axe vertical) d'une situation.

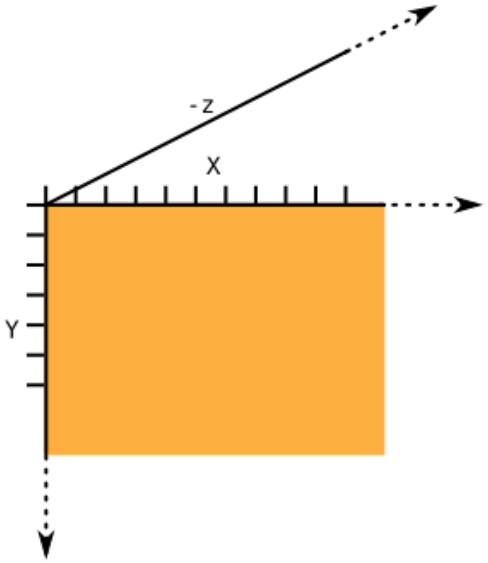
Par convention de la mesure de l'espace, le coin en haut à gauche correspond aux valeurs *x=0* et *y=0*. Les valeurs *x* sont croissantes vers la droite et les valeurs y sont croissantes vers le bas, contrairement à notre habitude du plan cartésien.

Ces valeurs *x* et *y* peuvent s'étendre théoriquement à l'infini, même si, en réalité, les contraintes de la taille de votre fenêtre vont délimiter la taille maximale d'une surface de création visible. C'est donc dans cet espace que nous allons dessiner.



repère2D

Quand on travaille en 3 dimensions (3D), en plus des deux axes de coordonnées, on a un troisième axe de coordonnées Z, exprimant la profondeur :



repère2D

Dans ce cas précis, on utilise la commande size avec un troisième paramètre indiquant que l'on travaille dans un espace en 3D

```
size(100, 100, P3D);
```

1.2.5 5 Les formes

Beaucoup de formes prédéfinies sont fournies par Processing. En voici les principales :

5.a. Le point Pour commencer à dessiner, nous allons partir d'un point. Sur l'écran, un point est l'équivalent d'un pixel localisé dans la fenêtre de visualisation par deux axes de coordonnées, x et y correspondant respectivement à la largeur (axe horizontal) et à la hauteur (axe vertical) de l'espace de dessin. En suivant ce principe, la création d'un point dans Processing s'effectue à l'aide de l'instruction point(x,y). Dans cet exemple, le point est très petit. Il est placé au centre de la fenêtre de visualisation.

```
point(50, 50);
```

```
In [15]: code = """
         point(50,50)
         """
         HTML(html_template.format(code,'II5a'))
```

```
Out[15]: <IPython.core.display.HTML object>
```

Notez que le cadre de la fenêtre de visualisation (l'espace de dessin) a une dimension de 100x100, ce qui explique que le point soit situé en plein milieu. Si on le dessinait en dehors du cadre (hors champ), avec par exemple l'instruction size(150,150), on ne le verrait pas.

5.b. La ligne Par définition, une ligne (AB) est constituée par une infinité de points entre un point de départ A et un point d'arrivée B. Pour la construire, nous allons nous intéresser uniquement aux coordonnées x et y de A et de B.

Ainsi, si par exemple dans la fenêtre par défaut, le point A se situe dans la région en bas à gauche de votre fenêtre, et que le point B se situe en haut à droite, les instructions suivantes, peuvent dessiner cette ligne sous la forme ***line(xA,yA,xB,yB)*** :

```
line(15, 90, 95, 10);
```

```
In [16]: code = "line(15, 90, 95, 10);"
         HTML(html_template.format(code,'II5b'))
```

```
Out[16]: <IPython.core.display.HTML object>
```

5.c. Le rectangle Un rectangle se dessine par quatre valeurs en faisant l'appel de ***rect(x,y,largeur,hauteur)***.
La première paire de valeurs x et y, par défaut (mode CORNER) correspond au coin supérieur gauche du rectangle, à l'instar du point.
En revanche la seconde paire de valeurs ne va pas se référer à la position du coin inférieur droit, mais à la largeur (sur l'axe des x, horizontal) et à la hauteur (sur l'axe des y, vertical) de ce rectangle.

```
rect(10, 10, 80, 40);
```

```
In [17]: code = "rect(10, 10, 80, 60);"
         HTML(html_template.format(code,'II5ci'))
```

```
Out[17]: <IPython.core.display.HTML object>
```

Pour que la première paire de valeurs corresponde au centre (le croisement des deux diagonales aux coordonnées 50, 50) du rectangle, il faut utiliser le mode CENTER comme suit :

```
rectMode(CENTER);
rect(50, 50, 80, 40);
```

```
In [18]: code = """
         rectMode(CENTER);
         rect(50, 50, 80, 40);
         """
         HTML(html_template.format(code,'II5cii'))
```

```
Out[18]: <IPython.core.display.HTML object>
```

1.2.6 5.d. L'ellipse

Comme pour le rectangle, l'ellipse se construit sous les modes CENTER (par défaut), et CORNER. Ainsi l'instruction suivante produit un cercle dont les coordonnées du centre sont les deux premières valeurs entre parenthèses. La troisième valeur correspond à la grandeur du diamètre sur l'axe horizontal (x) et la quatrième à la grandeur du diamètre sur l'axe vertical : notez que si les 3e et 4e valeurs sont identiques, on dessine un cercle et dans le cas contraire, une ellipse quelconque :

```
ellipse(50, 50, 80, 80);
```

Amusez-vous à faire varier les 3e et 4e valeurs et observez-en les résultats.

```
In [19]: code = """
         ellipse(50, 50, 80, 80);
         """
         HTML(html_template.format(code,'II5d'))
```

```
Out[19]: <IPython.core.display.HTML object>
```

1.2.7 5.e. Le triangle

Le triangle est un plan constitué de trois points. L'invocation de triangle(x1,y1,x2,y2,x3,y3) définit les trois points de ce triangle :

```
triangle(10, 90, 50, 10, 90, 90);
```

```
In [20]: code = """
         triangle(10, 90, 50, 10, 90, 90);
         """
         HTML(html_template.format(code,'II5e'))
```

```
Out[20]: <IPython.core.display.HTML object>
```

1.2.8 5.f. L'arc

Un arc ou section de cercle, peut se dessiner avec l'appel de ***arc(x, y, largeur, hauteur, début, fin)***, où la paire x, y définit le centre du cercle, la seconde paire ses dimensions et la troisième paire, le début et la fin de l'angle d'arc en radians :

```
arc(50, 50, 90, 90, 0, PI);
```

```
In [21]: code = """
         arc(50, 50, 90, 90, 0, PI);
         """
         HTML(html_template.format(code,'II5f'))
```

```
Out[21]: <IPython.core.display.HTML object>
```

Pour les angles on utilise les angles en radian, on verra en mathématiques à la fin de seconde pourquoi on utilise des angles en degré plustôt qu'en degrés.

- On tourne dans le sens des aiguilles d'une montre, (c'est l'inverse en mathématique)
- On peut avec la fonction ***radians()*** utiliser les angles en degré puis les convertir.

```
In [22]: code = """
         size(480, 120)
         arc(90, 60, 80, 80, 0, radians(90))
         arc(190, 60, 80, 80, 0, radians(270))
         arc(290, 60, 80, 80, radians(180), radians(450))
         arc(390, 60, 80, 80, radians(45), radians(225))
         """
         HTML(html_template.format(code,'II5f2'))
```

```
Out[22]: <IPython.core.display.HTML object>
```

1.2.9 5.g. Le quadrilatère

Le quadrilatère se construit en spécifiant quatre paires de coordonnées x et y sous la forme ***quad(x1,y1,x2,y2,x3,y3,x4,y4)*** dans le sens horaire :

```
quad(10, 10, 30, 15, 90, 80, 20, 80);
```

```
In [23]: code = """
         quad(10, 10, 30, 15, 90, 80, 20, 80);
         """
         HTML(html_template.format(code,'II5g'))
```

```
Out[23]: <IPython.core.display.HTML object>
```

1.2.10 5h. Courbe

Une courbe se dessine à l'aide de ***curve(x1, y1, x2, y2, x3, y3, x4, y4)***, où *x1* et *y1* définissent le premier point de contrôle, *x4* et *y4* le second point de contrôle , *x2* et *y2* le point de départ de la courbe et *x3, y3* le point d'arrivée de la courbe :

```
curve(0, 300, 10, 60, 90, 60, 200, 100);
```

```
In [24]: code = """
         curve(0, 300, 10, 60, 90, 60, 200, 100);
         """
         HTML(html_template.format(code,'II4h'))
```

```
Out[24]: <IPython.core.display.HTML object>
```

1.2.11 5.i. Courbe Bézier

La fonction ***bezier*** permet de dessiner une courbe de Bésier. Si le coeur vous en dit, vous en apprendrez plus sur les courbes de Bézier ici (je vous préviens, c'est très compliqué).

La courbe de type Bézier se construit à l'aide de ***bezier(x1,y1,x2,y2,x3,y3,x4,y4)***

```
bezier(10, 10, 70, 30, 30, 70, 90, 90);
```

```
In [25]: code = """
         bezier(10, 10, 70, 30, 30, 70, 90, 90);
         """
         HTML(html_template.format(code,'II5i'))
```

```
Out[25]: <IPython.core.display.HTML object>
```

1.2.12 5.j. Courbe lissée

L'appel de ***curveVertex()*** dessine plusieurs paires de coordonnées x et y, entre deux points de contrôle, sous la forme ***curveVertex(point de contrôle initial,xN,yN,xN,yN,xN,yN, point de contrôle final)*** ce qui permet de construire des courbes lissées :

```
beginShape();
curveVertex(0, 100);
curveVertex(10, 90);
curveVertex(25, 70);
curveVertex(50, 10);
curveVertex(75, 70);
curveVertex(90, 90);
curveVertex(100, 100);
endShape();
```

```
In [26]: code = """
         beginShape();
         curveVertex(0, 100);
         curveVertex(10, 90);
```

```
curveVertex(25, 70);
curveVertex(50, 10);
curveVertex(75, 70);
curveVertex(90, 90);
curveVertex(100, 100);
endShape();
"""
HTML(html_template.format(code,'II5j'))
```

```
Out[26]: <IPython.core.display.HTML object>
```

1.2.13 5.k. Formes libres

Plusieurs formes libres peuvent être dessinés par une succession de points en utilisant la suite d'instructions beginShape(),vertex(x,y),...,endShape(). Chaque point se construit par ses coordonnées x et y. La fonction CLOSE dans endShape(CLOSE) indique que la figure sera fermée, c'est-à-dire que le dernier point sera relié au premier, comme dans l'exemple ci-dessous de dessin d'un hexagone :

```
beginShape();
vertex(50, 10);
vertex(85, 30);
vertex(85, 70);
vertex(50, 90);
vertex(15, 70);
vertex(15, 30);
endShape(CLOSE);
```

```
In [27]: code = """
         beginShape();
         vertex(50, 10);
         vertex(85, 30);
         vertex(85, 70);
         vertex(50, 90);
         vertex(15, 70);
         vertex(15, 30);
         endShape(CLOSE);
         """
         HTML(html_template.format(code,'II5k'))
```

```
Out[27]: <IPython.core.display.HTML object>
```

1.2.14 5.l. Contours

Vous avez remarqué que jusqu'à présent, toutes les figures données en exemple comportent un contour, ainsi qu'une surface de remplissage.

Il est possible de modifier le contour.

- Si vous voulez rendre invisible le contour, utilisez ***noStroke()*** - On peut aussi dessiner le contour en couleur avec la fonction ***stroke()*** - On peut changer l'épaisseur des traits avec ***strokeWeight()***

```
noStroke()                  # On elève le contour
stroke(256,0,0)             # La couleur du contour est rouge
strokeWeight(4)             # L'épaisseur est 4
```

Par exemple

```
In [28]: code = """
         size(450, 120)
         ellipse(50, 50, 80, 80)
         noStroke()
         ellipse(150, 50, 80, 80)
         stroke(256,0,0)
         ellipse(240, 50, 80, 80)
         strokeWeight(4)
         ellipse(340, 50, 80, 80)
         """
         HTML(html_template.format(code,'II5l'))

Out[28]: <IPython.core.display.HTML object>
```

1.2.15 5.m. Remplissage

De la même manière, il est possible de modifier la surface de remplissage des formes.

- Si vous voulez rendre invisible la surface, utilisez ***noFill()***
- Si on veux dessiner la surface en couleur on utilise la fonction ***fill()***

```
noFill()                    # On enlève la surface de remplissage.
stroke(256,0,0)             # La couleur de la surface est rouge
```

Par défaut, l'arrière-fond de la fenêtre de visualisation (l'espace de dessin) est gris neutre, les contours des figures sont noirs, et la surface de remplissage est blanche. Vous apprendrez au prochain chapitre comment modifier les couleurs à votre convenance.

```
In [29]: code = """
         size(450, 120)
         ellipse(50, 50, 80, 80)
         noFill()
         ellipse(150, 50, 80, 80)
         fill(256,0,0)
         ellipse(240, 50, 80, 80)
         fill(0,256,0)
         ellipse(340, 50, 80, 80)
         """
         HTML(html_template.format(code,'II5m'))

Out[29]: <IPython.core.display.HTML object>
```

1.3 III Exercices:

exercice 1: Créer à l'aide de processing un bonhomme pacman.

1.4 IV Pour aller plus loin

1.4.1 1 Primitives 3D

Les formes prédéfinies disponibles en 3 dimensions (primitives 3D) peuvent être réalisées de manière simple en appelant size(x, y, P3D) au début de notre sketch puis en employant en fonction de vos besoins les instructions sphere(taille) et box(longueur, largeur, profondeur). Il est également possible de produire des effets d'éclairage sur nos formes tridimensionnelles à l'aide de ***lights()***.

a La sphère

```
size(100, 100, P3D);
noStroke();             # On enlève le contour
lights();               # éclairer l'objet 3D
translate(50, 50, 0);   # translation
sphere(28);
```

```
In [30]: code = """
         size(100, 100, P3D);
         noStroke();
         lights();
         translate(50,50,0);
         sphere(28);
         """
         HTML(html_template.format(code,'IV1a'))
```

```
Out[30]: <IPython.core.display.HTML object>
```

b La boîte

```
size(100, 100, P3D);
noStroke();
lights();
translate(50, 50, 0);
rotateY(0.5);   //rotation selon l'axe (yy')
box(40);
```

```
In [31]: code = """
         size(100, 100, P3D);
         noStroke();
         lights();
         translate(50, 50, 0);
         rotateY(0.5);
         box(40);
         """
         HTML(html_template.format(code,'IV1b'))
```

```
Out[31]: <IPython.core.display.HTML object>
```