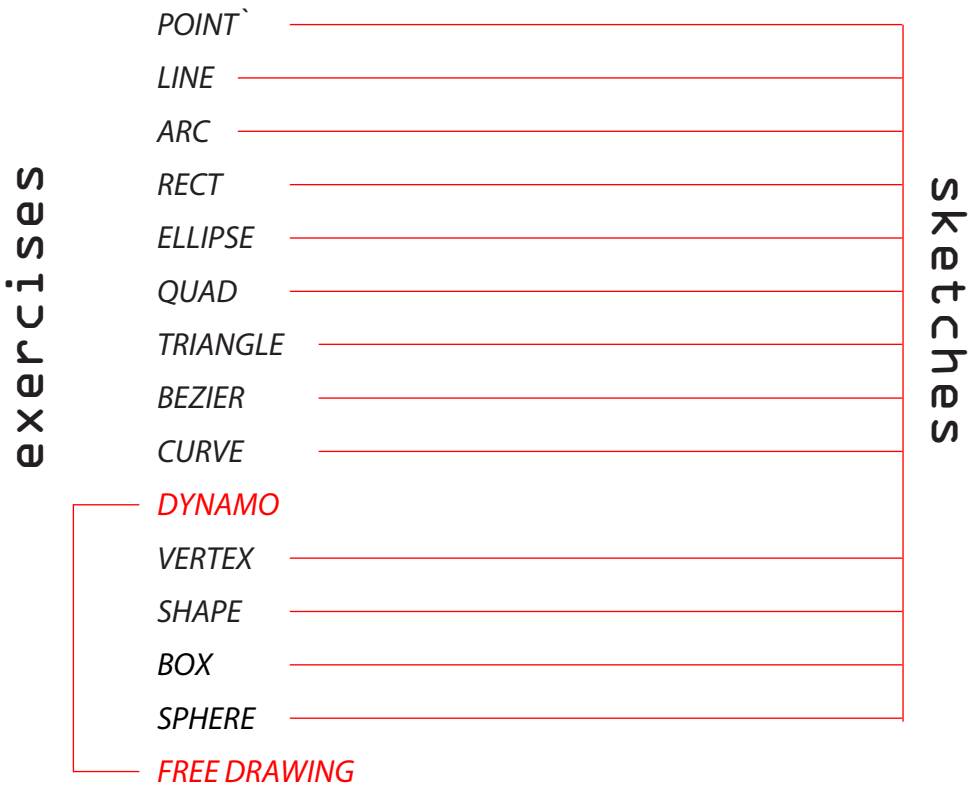

# DIGITAL DRAWING
## FROM THE TOOL TO THE SYSTEM

creative
scripting
m e a l s

01


exercises
POINT`
LINE
ARC
RECT
ELLIPSE
QUAD
TRIANGLE
BEZIER
CURVE
DYNAMO
VERTEX
SHAPE
BOX
SPHERE
FREE DRAWING
sketches

# Drawing Geometry

## *CREATE SHAPE*

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()
sphereDetail()

## *CREATE VERTEX*

beginContour()
beginShape()
bezierVertex()
curveVertex()
endContour()
endShape()
quadraticVertex()
vertex()

## *ATTRIBUTES*

ellipseMode()
noSmooth()
rectMode()
smooth()
strokeCap()
strokeJoin()
strokeWeight()

Constants
HALF_PI
PI
QUARTER_PI
TAU
TWO_PI

## *COLOR*

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

# point()

http://processing.org/reference/point_.html

*SYNTAX*

```
point(x, y)
point(x, y, z)
```

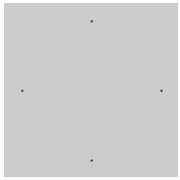
*DESCRIPTION*

Draws a point, a coordinate in space at the dimension of one pixel. The first parameter is the horizontal value for the point, the second value is the vertical value for the point, and the optional third value is the depth value. Drawing this shape in 3D with the z parameter requires the P3D parameter in combination with size() as shown in the above example.

*PARAMETRES*

| | |
|---|---|
| x | float: x-coordinate of the point |
| y | float: y-coordinate of the point |
| z | float: z-coordinate of the point |

*CODE*



```
noSmooth(); // remove anti-aliased edges filter

// create the points of a triangle
// Syntax point(x,y);
point(10,50);
point(50,10);
point(90,50);
point(50,90);
```

# line()

http://processing.org/reference/line_.html

*SYNTAX*

```
line(x1, y1, x2, y2)
line(x1, y1, z1, x2, y2, z2)
```

*DESCRIPTION*

Draws a line (a direct path between two points) to the screen. The version of line() with four parameters draws the line in 2D. To color a line, use the stroke() function. A line cannot be filled, therefore the fill() function will not affect the color of a line. 2D lines are drawn with a width of one pixel by default, but this can be changed with the strokeWeight() function. The version with six parameters allows the line to be placed anywhere within XYZ space. Drawing this shape in 3D with the z parameter requires the P3D parameter in combination with size() as shown in the above example.

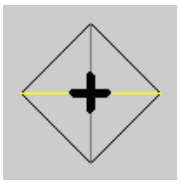
*PARAMETRES*

| | |
|---|---|
| x1 | float: x-coordinate of the first point |
| y1 | float: y-coordinate of the first point |
| x2 | float: x-coordinate of the second point |
| y2 | float: y-coordinate of the second point |
| z1 | float: z-coordinate of the first point |
| z2 | float: z-coordinate of the second point |

*PROPERTIES*

stroke()
strokeWeight()

*CODE*



```
noSmooth(); // remove anti-aliased edges filter

//NO COLOR, NO WEIGHT
// line(x1, y1, x2, y2);
line(10,50, 50,10);
line(50,10, 90,50);
line(90,50, 50,90);
line(50,90, 10,50);

// APPLIED COLOR
// storke(GrayscalePattern);
stroke(122); // Gray Color
line(50,10, 50,90);
// stroke(Red, Green, Blue);
stroke(255,255,0); // Yellow Color
line(10,50, 90,50);

// APPLIED WEIGHT
// strokeWeight(pixel);
strokeWeight(5);  // Default
stroke(0); // Black Color
line(50,40, 50,60);
line(40,50, 60,50);
```

# arc()

http://processing.org/reference/arc_.html

*SYNTAX*

arc(a, b, c, d, start, stop)
arc(a, b, c, d, start, stop, mode)

*DESCRIPTION*

Draws an arc to the screen. Arcs are drawn along the outer edge of an ellipse defined by the a, b, c, and d parameters. The origin of the arc's ellipse may be changed with the ellipseMode() function. Use the start and stop parameters to specify the angles (in radians) at which to draw the arc. There are three ways to draw an arc; the rendering technique used is defined by the optional seventh paramter. The default mode is OPEN, and the other options are CHORD and PIE. Each is shown in the above examples.

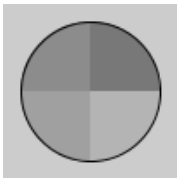
*PARAMETRES*

| | |
|---|---|
| a | float: x-coordinate of the arc's ellipse |
| b | float: y-coordinate of the arc's ellipse |
| c | float: width of the arc's ellipse by default |
| d | float: height of the arc's ellipse by default |
| start | float: angle to start the arc, specified in radians |
| stop | float: angle to stop the arc, specified in radians |
| mode | Allows to close the shape |

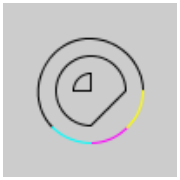
*PROPERTIES*

stroke()
strokeWeight()

*CODE*



```
smooth(); // remove anti-aliased edges filter
// COLORED ARC
fill(180);
arc(50, 50, 80, 80, 0, HALF_PI);
fill(160);
arc(50, 50, 80, 80, HALF_PI, PI);
fill(140);
arc(50, 50, 80, 80, PI, PI+HALF_PI);
fill(120);
arc(50, 50, 80, 80, PI+HALF_PI, TWO_PI);
```



```
smooth(); // remove anti-aliased edges filter
// NO COLOR, MODES
noFill();
arc(50, 50, 60, 60, HALF_PI+QUARTER_PI, TWO_PI);
arc(50, 50, 40, 40, HALF_PI, TWO_PI, CHORD);
arc(50, 50, 20, 20, PI, PI+HALF_PI, PIE);

// STROKE
stroke(255,0,255);
arc(50, 50, 60, 60, QUARTER_PI, HALF_PI);
stroke(255,255,0);
arc(50, 50, 60, 60, 0, QUARTER_PI);
stroke(0,255,255);
arc(50, 50, 60, 60, HALF_PI, HALF_PI+QUARTER_PI);
```

# rect()

http://processing.org/reference/rect_.html

*SYNTAX*

```
rect(a, b, c, d)
rect(a, b, c, d, r)
rect(a, b, c, d, tl, tr, br, bl)
```

*DESCRIPTION*

Draws a rectangle to the screen. A rectangle is a four-sided shape with every angle at ninety degrees. By default, the first two parameters set the location of the upper-left corner, the third sets the width, and the fourth sets the height. The way these parameters are interpreted, however, may be changed with the rectMode() function.

To draw a rounded rectangle, add a fifth parameter, which is used as the radius value for all four corners.

To use a different radius value for each corner, include eight parameters. When using eight parameters, the latter four set the radius of the arc at each corner separately, starting with the top-left corner and moving clockwise around the rectangle.

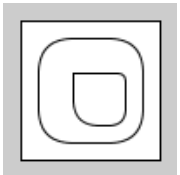
*PARAMETRES*

| | |
|---|---|
| a | float: x-coordinate of the rectangle by default |
| b | float: y-coordinate of the rectangle by default |
| c | float: width of the rectangle by default |
| d | float: height of the rectangle by default |
| r | float: radii for all four corners |
| tl | float: radius for top-left corner |
| tr | float: radius for top-right corner |
| br | float: radius for bottom-right corner |
| bl | float: radius for bottom-left corner |

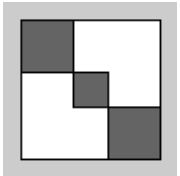
*PROPERTIES*

rectMode()

*CODE*



```
smooth(); // remove anti-aliased edges filter
rect(10, 10, 80, 80);
rect(20, 20, 60, 60, 20);
rect(40, 40, 30, 30, 0, 5, 10, 15);
```



```
fill(255);  // Set fill to white
rectMode(CENTER);  // Set rectMode to CORNERS
rect(50, 50, 80, 80);  // rect(centerX, centerY, width, height);

fill(100);  // Fill with gray color the others
rectMode(CORNERS);  // Set rectMode to CORNERS
rect(10, 10, 40, 40);  // rect(topL_X, topL_Y, bottomR_X, bottomR_Y);

rectMode(RADIUS);  // Set rectMode to CORNERS
rect(50, 50, 10, 10);  // rect(centerX, centerY, width/2, width/2);

rectMode(CORNER);  // Default rectMode is CORNER
rect(60, 60, 30, 30);  // rect(topL_X, topL_Y, width, height);
```

# ellipse()

http://processing.org/reference/ellipse_.html

*SYNTAX*

ellipse(a, b, c, d)

*DESCRIPTION*

Draws an ellipse (oval) to the screen. An ellipse with equal width and height is a circle. By default, the first two parameters set the location, and the third and fourth parameters set the shape's width and height. The origin may be changed with the ellipseMode() function.

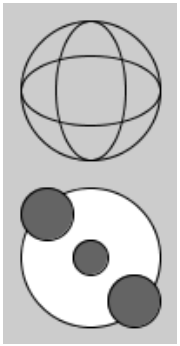
*PARAMETRES*

| | |
|---|---|
| a | float: x-coordinate of the ellipse |
| b | float: y-coordinate of the ellipse |
| c | float: width of the ellipse by default |
| d | float: height of the ellipse by default |

*PROPERTIES*

ellipseMode()
arc()

*CODE*



```
smooth();
noFill();
ellipse(width/2, height/2, 80, 80);
ellipse(width/2, height/2, 40, 80);
ellipse(width/2, height/2, 80, 40);
```

```
fill(255);  // Set fill to white
ellipseMode(CENTER);  // Set rectMode to CORNERS
// ellipse(centerX, centerY, width, height);
ellipse(50, 50, 80, 80);  // Draw gray rect using CORNERS mode

fill(100);  // Set fill to gray
ellipseMode(CORNERS);  // Set rectMode to CORNERS
// ellipse(topLeftX, topLeftY, bottomRightX, bottomRightY);
ellipse(10, 10, 40, 40);  // Draw gray rect using CORNERS mode

ellipseMode(RADIUS);  // Set rectMode to CORNERS
// ellipse(centerX, centerY, width/2, width/2);
ellipse(50, 50, 10, 10);  // Draw gray rect using CORNERS mode

ellipseMode(CORNER);  // Default rectMode is CORNER
// ellipse(topLeftX, topLeftY, width, height);
ellipse(60, 60, 30, 30);  // Draw white rect using CORNER mode
```

# triangle()

http://processing.org/reference/triangle_.html

*SYNTAX*

```
triangle(x1, y1, x2, y2, x3, y3)
```

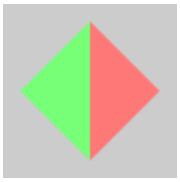
*DESCRIPTION*

A triangle is a plane created by connecting three points. The first two arguments specify the first point, the middle two arguments specify the second point, and the last two arguments specify the third point.

*PARAMETRES*

| | |
|---|---|
| x1 | float: x-coordinate of the first point |
| y1 | float: y-coordinate of the first point |
| x2 | float: x-coordinate of the second point |
| y2 | float: y-coordinate of the second point |
| x3 | float: x-coordinate of the third point |
| y3 | float: y-coordinate of the third point |

*CODE*



```
smooth();
noStroke();
fill(255,120,120);
triangle(width/2, 10, 90, height/2, width/2, 90);
fill(120,255,120);
triangle(width/2, 10, 10, height/2, width/2, 90);
```

# quad()

http://processing.org/reference/quad_.html

*SYNTAX*

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```

*DESCRIPTION*

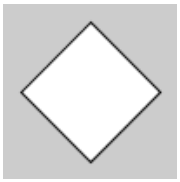
A quad is a quadrilateral, a four sided polygon. It is similar to a rectangle, but the angles between its edges are not constrained to ninety degrees. The first pair of parameters (x1,y1) sets the first vertex and the subsequent pairs should proceed clockwise or counter-clockwise around the defined shape.

*PARAMETRES*

| | |
|---|---|
| x1 | float: x-coordinate of the first corner |
| y1 | float: y-coordinate of the first corner |
| x2 | float: x-coordinate of the second corner |
| y2 | float: y-coordinate of the second corner |
| x3 | float: x-coordinate of the third corner |
| y3 | float: y-coordinate of the third corner |
| x4 | float: x-coordinate of the fourth corner |
| y4 | float: y-coordinate of the fourth corner |

*CODE*

```
smooth();
quad(width/2, 10, 90, height/2, width/2, 90, 10, height/2);
```



# bezier()

http://processing.org/reference/bezier_.html

*SYNTAX*

```
bezier(x1, y1, x2, y2, x3, y3, x4, y4)
bezier(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)
```

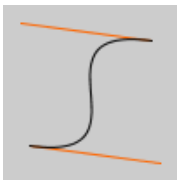
*DESCRIPTION*

Draws a Bezier curve on the screen. These curves are defined by a series of anchor and control points. The first two parameters specify the first anchor point and the last two parameters specify the other anchor point. The middle parameters specify the control points which define the shape of the curve. Bezier curves were developed by French engineer Pierre Bezier. Using the 3D version requires rendering with P3D (see the Environment reference for more information).

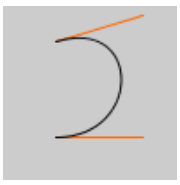
*PARAMETRES*

| | |
|---|---|
| x1 | float: coordinates for the first anchor point |
| y1 | float: coordinates for the first anchor point |
| z1 | float: coordinates for the first anchor point |
| x2 | float: coordinates for the first control point |
| y2 | float: coordinates for the first control point |
| z2 | float: coordinates for the first control point |
| x3 | float: coordinates for the second control point |
| y3 | float: coordinates for the second control point |
| z3 | float: coordinates for the second control point |
| x4 | float: coordinates for the second anchor point |
| y4 | float: coordinates for the second anchor point |
| z4 | float: coordinates for the second anchor point |

*CODE*



```
noFill();
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
```



```
noFill();
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20,  80, 5,  80, 75,  30, 75);
```

# curve()

http://processing.org/reference/curve_.html

*SYNTAX*

```
curve(x1, y1, x2, y2, x3, y3, x4, y4)
curve(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)
```

*DESCRIPTION*

Draws a curved line on the screen. The first and second parameters specify the beginning control point and the last two parameters specify the ending control point. The middle parameters specify the start and stop of the curve. Longer curves can be created by putting a series of curve() functions together or using curveVertex(). An additional function called curveTightness() provides control for the visual quality of the curve. The curve() function is an implementation of Catmull-Rom splines. Using the 3D version requires rendering with P3D (see the Environment reference for more information).

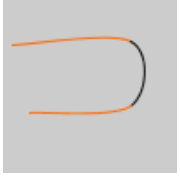
*PARAMETRES*

| | |
|---|---|
| x1 | float: coordinates for the beginning control point |
| y1 | float: coordinates for the beginning control point |
| x2 | float: coordinates for the first point |
| y2 | float: coordinates for the first point |
| x3 | float: coordinates for the second point |
| y3 | float: coordinates for the second point |
| x4 | float: coordinates for the ending control point |
| y4 | float: coordinates for the ending control point |
| z1 | float: coordinates for the beginning control point |
| z2 | float: coordinates for the first point |
| z3 | float: coordinates for the second point |
| z4 | float: coordinates for the ending control point |

*PROPERTIES*

curveVertex()
curveTightness()

*CODE*



```
noFill();
stroke(255, 102, 0);
curve(5, 26, 5, 26, 73, 24, 73, 61);
stroke(0);
curve(5, 26, 73, 24, 73, 61, 15, 65);
stroke(255, 102, 0);
curve(73, 24, 73, 61, 15, 65, 15, 65);
```

# DYNAMO

*DESCRIPTION*

Using scripting, draw then DYNAMO character logo.
Use the following sketch in order to complete your project.
Try to keep the code simple.

*CODE*

```
// DYNAMO

size(600,100); // window size
background(200, 255, 255);
stroke(200);
for (int i=1; i<7; i++) {
line(100*i,0,100*i,100);
}

/*
INSRUCTIONS
fit your text inside the given space
keep 10 points distance from top, bottom and lines
*/

// add your code below
```

# vertex()

http://processing.org/reference/vertex_.html

*SYNTAX*

```
vertex(x, y)
vertex(x, y, z)
vertex(v)
vertex(x, y, u, v)
vertex(x, y, z, u, v)
```

*DESCRIPTION*

All shapes are constructed by connecting a series of vertices. vertex() is used to specify the vertex coordinates for points, lines, triangles, quads, and polygons. It is used exclusively within the beginShape() and endShape() functions.

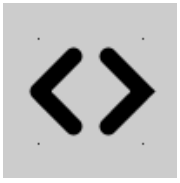
*PARAMETRES*

| | |
|---|---|
| v | float[]: vertex parameters, as a float array of length VERTEX_FIELD_COUNT |
| x | float: x-coordinate of the vertex |
| y | float: y-coordinate of the vertex |
| z | float: z-coordinate of the vertex |
| u | float: horizontal coordinate for the texture mapping |
| v | float: vertical coordinate for the texture mapping |

*PROPERTIES*

```
beginShape()
endShape()
```

*CODE*



```
beginShape(POINTS);
vertex(20, 20);
vertex(80, 20);
vertex(80, 80);
vertex(20, 80);
endShape();

noFill();
strokeWeight(10);
beginShape();
vertex(60, 30);
vertex(80, 50);
vertex(60, 70);
endShape();

noFill();
strokeWeight(10);
strokeJoin(ROUND);
beginShape();
vertex(40, 30);
vertex(20, 50);
vertex(40, 70);
endShape();
```

# curveVertex()

http://processing.org/reference/curveVertex_.html

*SYNTAX*

curveVertex(x, y)
curveVertex(x, y, z)

*DESCRIPTION*

Specifies vertex coordinates for curves. This function may only be used between beginShape() and endShape() and only when there is no MODE parameter specified to beginShape(). The first and last points in a series of curveVertex() lines will be used to guide the beginning and end of a the curve. A minimum of four points is required to draw a tiny curve between the second and third points. Adding a fifth point with curveVertex() will draw the curve between the second, third, and fourth points. The curveVertex() function is an implementation of Catmull-Rom splines. Using the 3D version requires rendering with P3D (see the Environment reference for more information).

*PARAMETRES*

| | |
|---|---|
| x | float: the x-coordinate of the vertex |
| y | float: the y-coordinate of the vertex |
| z | float: the z-coordinate of the vertex |

*PROPERTIES*

beginShape()
endShape()

*CODE*

```
noFill();
beginShape();
curveVertex(84,  91);
curveVertex(84,  91);
curveVertex(68,  19);
curveVertex(21,  17);
curveVertex(32, 100);
curveVertex(32, 100);
endShape()
```

# box()

http://processing.org/reference/box_.html

*SYNTAX*

box(size)
box(w, h, d)

*DESCRIPTION*

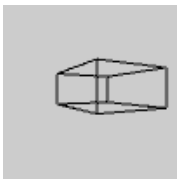
A box is an extruded rectangle. A box with equal dimensions on all sides is a cube.

*PARAMETRES*

| | |
|---|---|
| size | float: dimension of the box in all dimensions (creates a cube) |
| w | float: dimension of the box in the x-dimension |
| h | float: dimension of the box in the y-dimension |
| d | float: dimension of the box in the z-dimension |

*CODE*

```
size(100, 100, P3D);
translate(58, 48, 0);
rotateY(0.5);
noFill();
box(40, 20, 50);
```



# sphere()

http://processing.org/reference/sphere_.html

*SYNTAX*

sphere(r)

*DESCRIPTION*

A sphere is a hollow ball made from tessellated triangles.

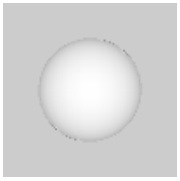
*PARAMETRES*

r float: the radius of the sphere

*PROPERTIES*

stroke()
strokeWeight()

*CODE*



```
size(100, 100, P3D);
noStroke();
lights();
translate(50, 50, 0);
sphere(28);
```