computer graphics

Steps of the lesson

- Open the link https://processing.org/ (type the keyword – processing in the browser)
- Go to the menu –Learn –>Tutorials->Hello processing: https://hello.processing.org/ and try the tutorial and make the example there
- Go to Download - https://processing.org/download and download the program
- Make a sketch with 2d primitives function (use for help (Documentation->Reference))

INSTRUCTIONS FOR USER

- ✓ Start by visiting **https://processing.org/download** and selecting the Mac, Windows, or Linux version, depending on what machine you have.
- ✓ Installation on **Windows**, you'll have a **.zip file**. Double-click it, and drag the folder inside to a location on your hard disk. It could be Program Files or simply the desktop, but the important thing is for the processing folder to be pulled out of that .zip file. Then double-click **processing.exe** to start.

INSTRUCTIONS FOR USER

- ✓Installation on **the Mac OS X version** is also a **.zip file**. Double-click it and drag the Processing icon to the Applications folder. If you're using someone else's machine and can't modify the Applications folder, just drag the application to the desktop. Then double-click the Processing icon to start.

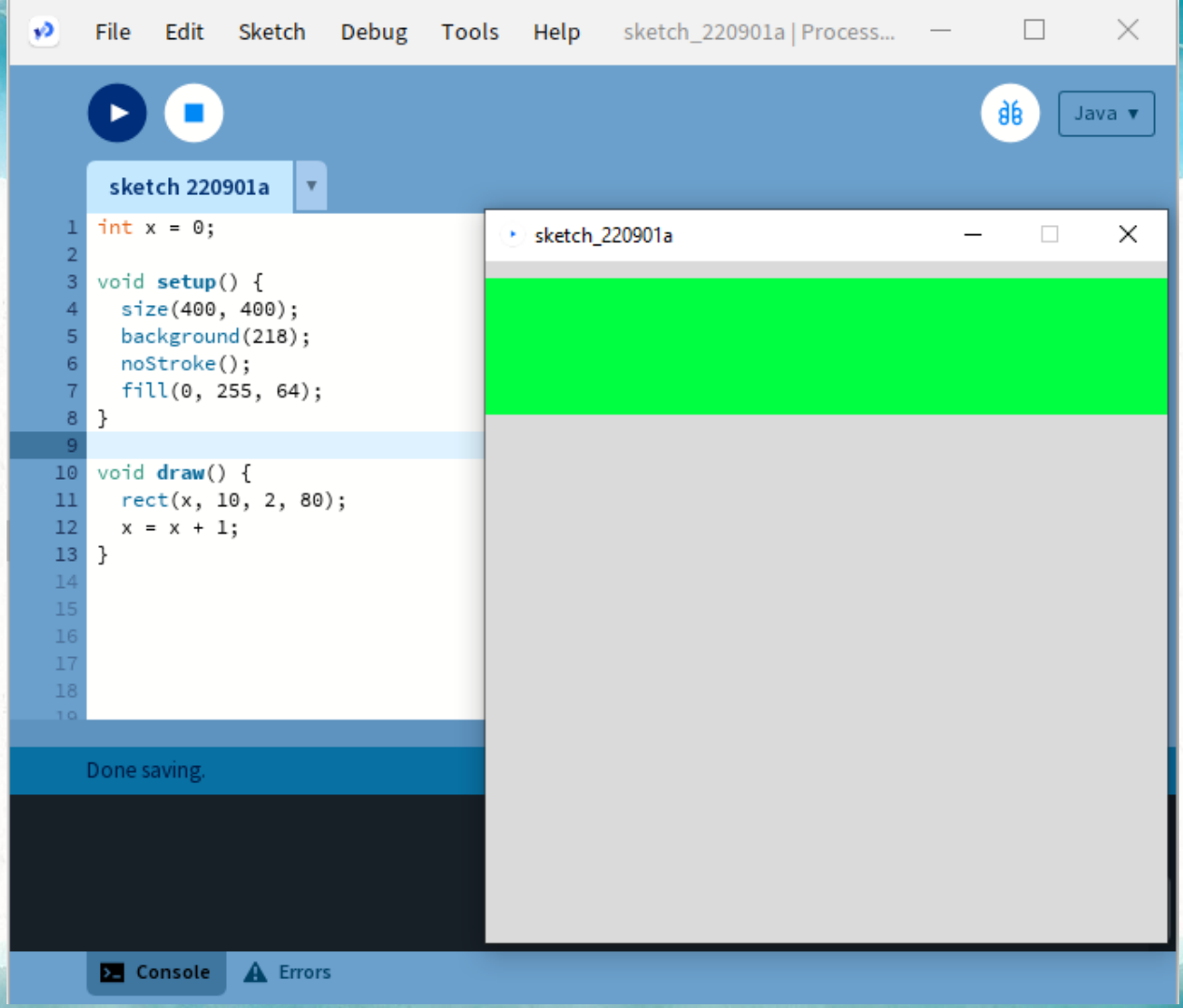
INSTRUCTIONS FOR USE

- ✓ Installation on the **Linux** version is a **.tar.gz file**, which should be familiar to most Linux users. Download the file to your home directory, then open a terminal window, and type:

tar xvfz processing-xxxx.tgz

(Replace xxxx with the rest of the file's name, which is the version number.) This will create a folder named processing-2.0 or something similar. Then change to that directory: cd processing-xxxx

and run it: **./processing**


File
Edit
Sketch
Debug
Tools
Help
sketch_220901a | Process...
Java
sketch 220901a
1 int x = 0;
2
3 void setup() {
4 size(400, 400);
5 background(218);
6 noStroke();
7 fill(0, 255, 64);
8 }
9
10 void draw() {
11 rect(x, 10, 2, 80);
12 x = x + 1;
13 }
sketch_220901a
Done saving.
Console
Errors

Primitive datatype

`boolean`	Datatype for the Boolean values `true` and `false`
`byte`	Datatype for bytes, 8 bits of information storing numerical values from 127 to -128
`char`	Datatype for characters, typographic symbols such as A, d, and $
`color`	Datatype for storing color values
`double`	Datatype for floating-point numbers larger than those that can be stored in a `float`
`float`	Data type for floating-point numbers, e
`int`	Datatype for integers, numbers without a decimal point
`long`	Datatype for large integers

Structure function – setup()

- The setup() function is run once, when the program starts.
- It's used to define initial environment properties such as screen size and to load media such as images and fonts as the program starts.
- There can only be one setup() function for each program and it shouldn't be called again after its initial execution.

Structure function – setup()

- If the sketch is a different dimension than the default, the **size()** function or **fullScreen()** function must be the first line in **setup()**.

Note: Variables declared within setup() are not accessible within other functions, including **draw().**

Structure function – draw()

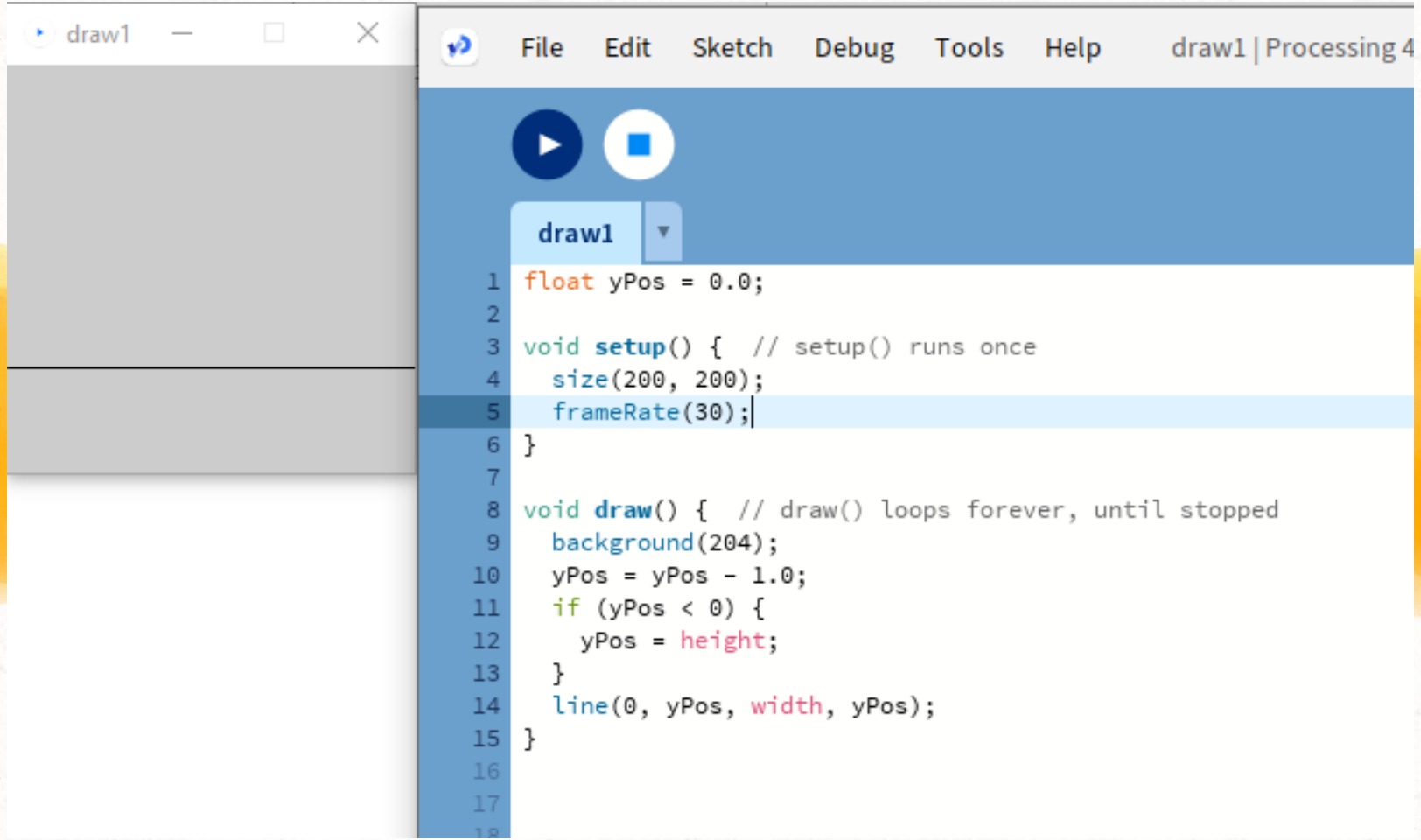
- Called directly after **setup()**, the **draw()** function ***continuously*** executes the lines of code contained inside its block until the program is stopped or **noLoop()** is called.
- **draw()** is called automatically and should never be called explicitly. All Processing programs update the screen at the end of draw(), never earlier.

Structure function – draw()

- To stop the code inside of **draw()** from running continuously, use **noLoop()**, **redraw()** and **loop()**.
- If **noLoop()** is used to stop the code in **draw()** from running, then **redraw()** will cause the code inside **draw()** to run a single time, and **loop()** will cause the code inside **draw()** to resume running continuously.

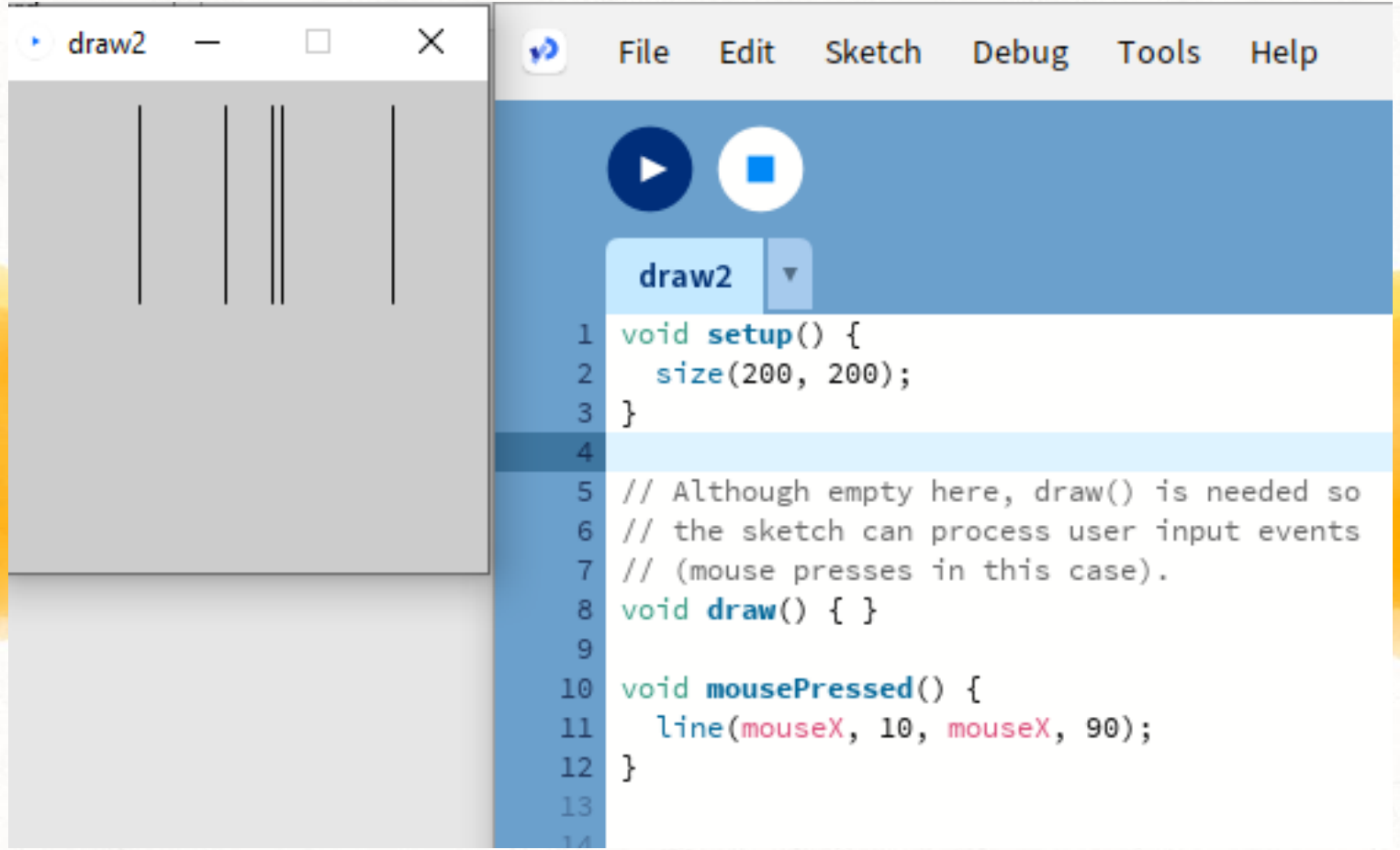
Structure function – draw()

- The number of times **draw()** executes in each second may be controlled with the **frameRate()** function.
- It is common to call **background()** near the beginning of the **draw()** loop to clear the contents of the window. Since pixels drawn to the window are cumulative, omitting **background()** may result in unintended results.


draw1
File
Edit
Sketch
Debug
Tools
Help
draw1 | Processing 4
draw1
1 float yPos = 0.0;
2
3 void setup() { // setup() runs once
4 size(200, 200);
5 frameRate(30);
6 }
7
8 void draw() { // draw() loops forever, until stopped
9 background(204);
10 yPos = yPos - 1.0;
11 if (yPos < 0) {
12 yPos = height;
13 }
14 line(0, yPos, width, yPos);
15 }
16
17

Structure function – draw()

- There can only be one **draw()** function for each sketch, and **draw()** must exist if you want the code to run continuously, or to process events such as **mousePressed()**.
- Sometimes, you might have an empty call to **draw()** in your program.


draw2
File
Edit
Sketch
Debug
Tools
Help
draw2
1 void setup() {
2 size(200, 200);
3 }
4
5 // Although empty here, draw() is needed so
6 // the sketch can process user input events
7 // (mouse presses in this case).
8 void draw() { }
9
10 void mousePressed() {
11 line(mouseX, 10, mouseX, 90);
12 }
13

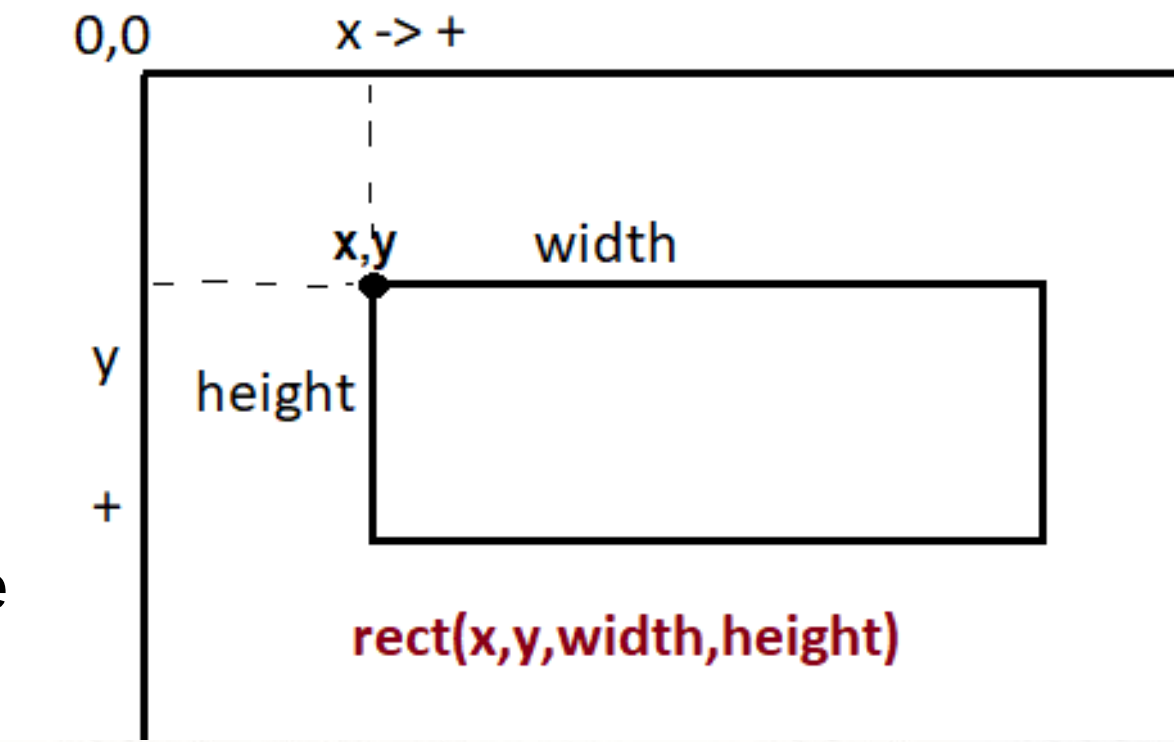
Environment function – size()

- Exemple: size(200, 100);

  size(150, 200, P3D); // Specify P3D renderer
- Defines the dimension of the display window width and height in units of pixels.
- In a program that has the **setup()** function, the **size()** function must be the first line of code inside **setup()**, and the **setup()** function must appear in the code tab with the same name as your sketch folder.

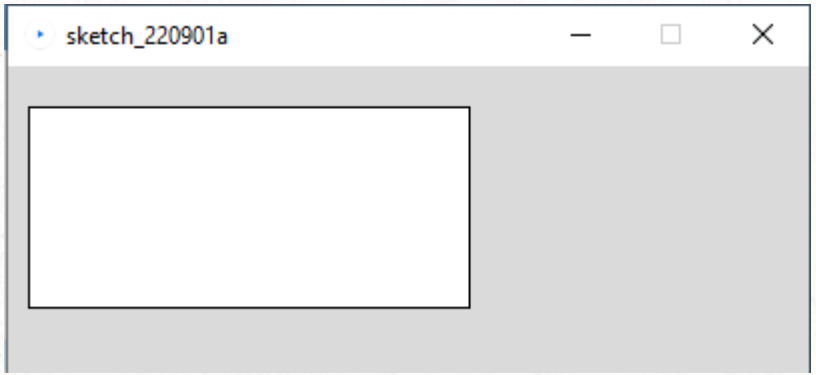
2D primitives function – rect()

- the first two parameters set the location of the upper-left corner,
- the third sets the width,
- and the fourth sets the height.



rect(x,y,width,heigth) – exemple:

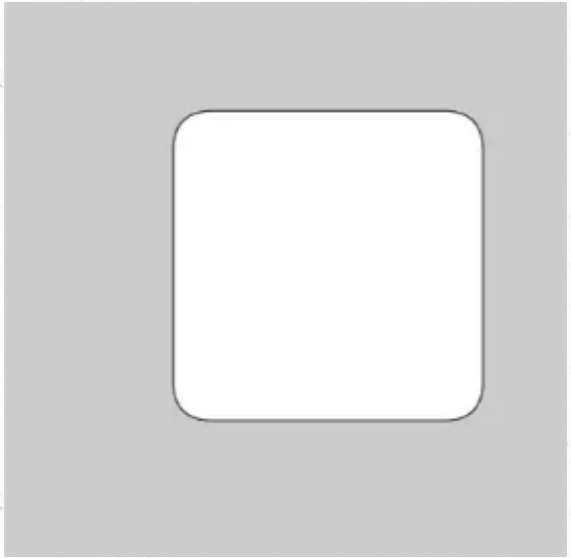
```
size(400, 400);
rect(10, 20, 220, 100);
```



rect(x,y,width,height,radii 4 corners)

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

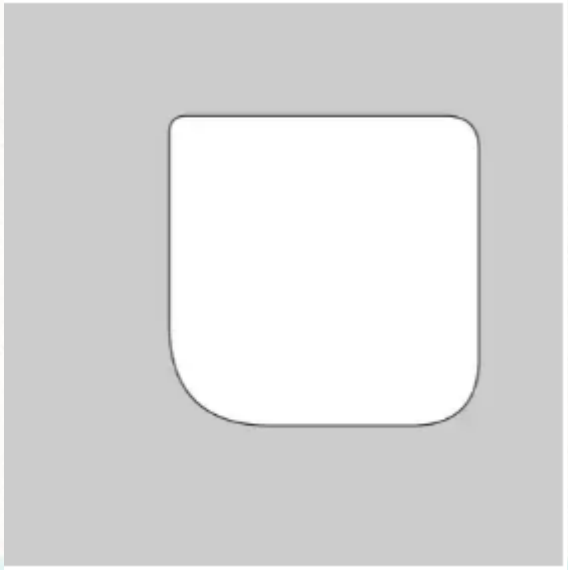
```
size(400, 400);
rect(120, 80, 220, 220, 28);
```



rect(x,y,width,height,tl,tr,bl,br)

- To use a different radius value for each corner, include eight parameters.

```
size(400, 400);
rect(120, 80, 220, 220, 12, 24, 48, 72);
```

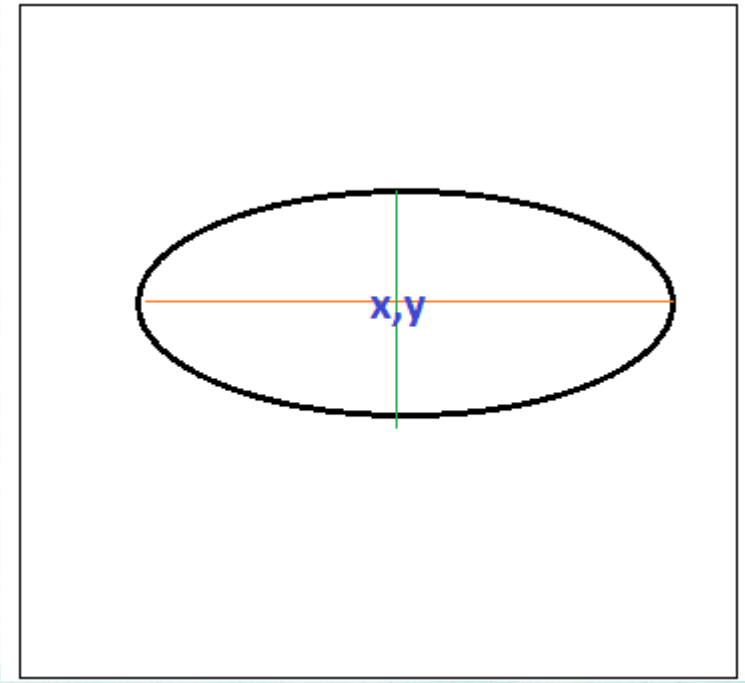


- **tl**=radius for top-left corner
- **tr**=radius for top-right corner
- **bl**=radius for bottom-left corner
- **br**=radius for bottom-right corner

2D primitives function – ellipse()

- 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.

ellipse(x,y,width,height)

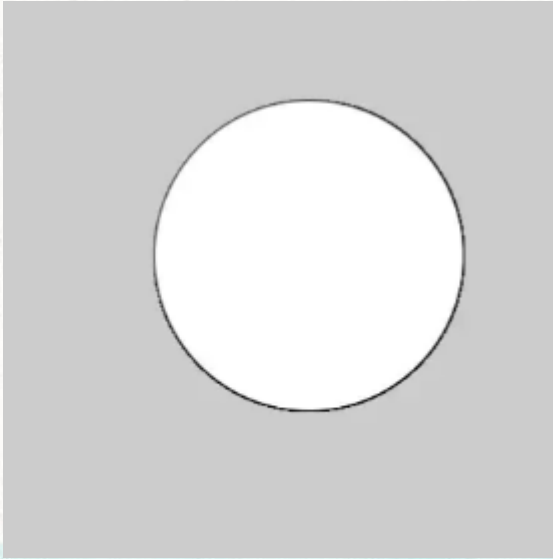


2D primitives function – circle()

- Draws **a circle** to the screen.
- the *first two parameters set the location of the center*,
- **extent**=sets the shape's *width and height* by default.
- The origin may be changed with the ellipseMode() function.

circle(x, y, extent)

circle(224, 184, 220);

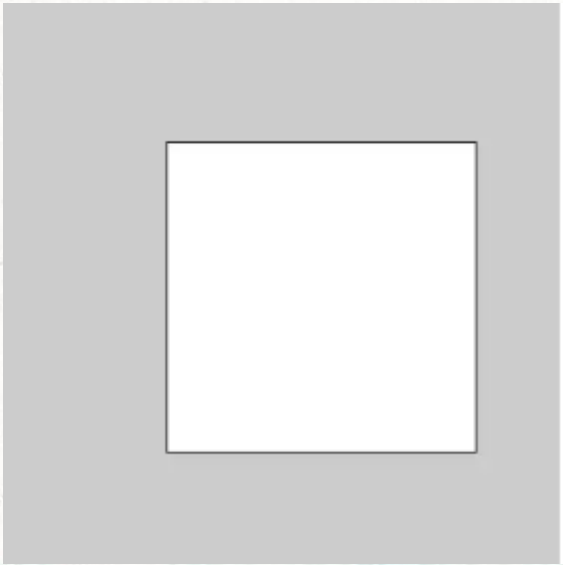


2D primitives function – square()

- Draws a square to the screen.
- **extent**= set width and height of the rectangle by default.
- The origin may be changed with the rectMode() function.

square(x, y, extent)

square(120, 100, 220);

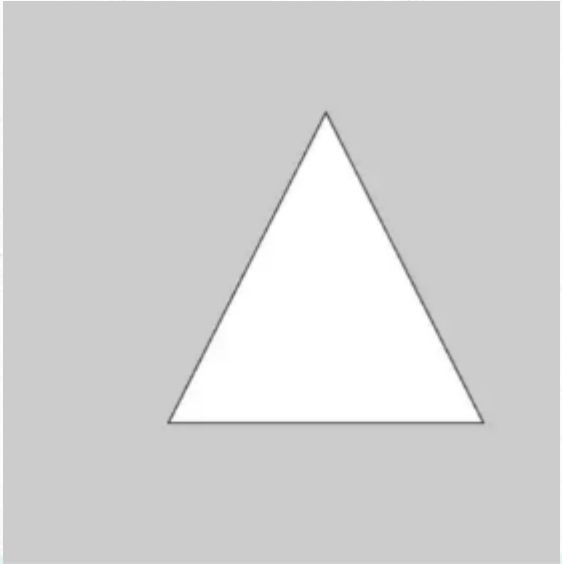


2D primitives function – triangle()

- 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.

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

triangle(120, 300, 232, 80, 344, 300);

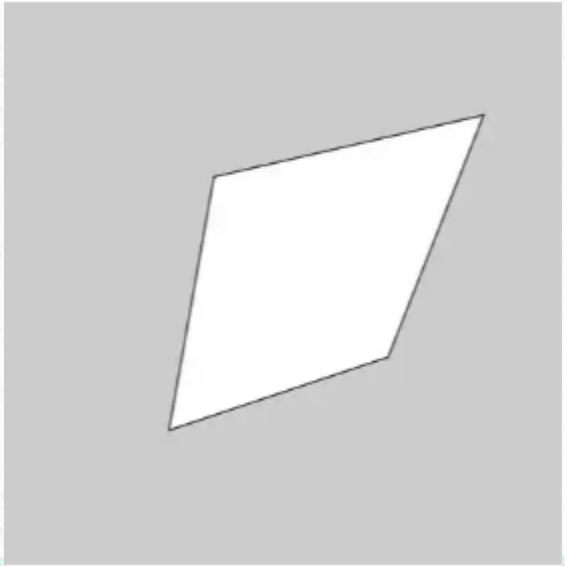


2D primitives function – quad()

- 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.

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

```
quad(152, 124, 344, 80, 276, 252, 120, 304);
```



2D primitives function – line()

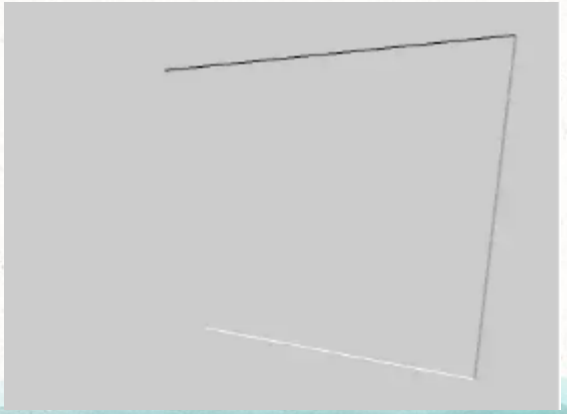
line(x1, y1, x2, y2)

- 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.

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

- 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.

```
size(400, 400, P3D);
line(120, 80, 0, 340, 80, 60);
stroke(126);
line(340, 80, 60, 340, 300, 0);
stroke(255);
 line(340, 300, 0, 120, 300, -200);
```



dOCUMENTATION

- https://processing.org/reference
- DANIEL SHIFFMAN “Learning Processing” http://learningprocessing.com/
- DANIEL SHIFFMAN “The Nature of Code”

https://natureofcode.com/book/introduction/