

# An Introduction to Processing

---

Formatting shapes

Lecturer: Caio Fonseca



Waterford Institute *of* Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics  
<http://www.wit.ie/>

# Topics list

---

- Filling shapes with colour.
- Formatting the shape outline.
- Adding comments to your code.

# fill() - Syntax

---

**fill (r, g, b)**

r = red colour (a number between 0 and 255 inclusive)

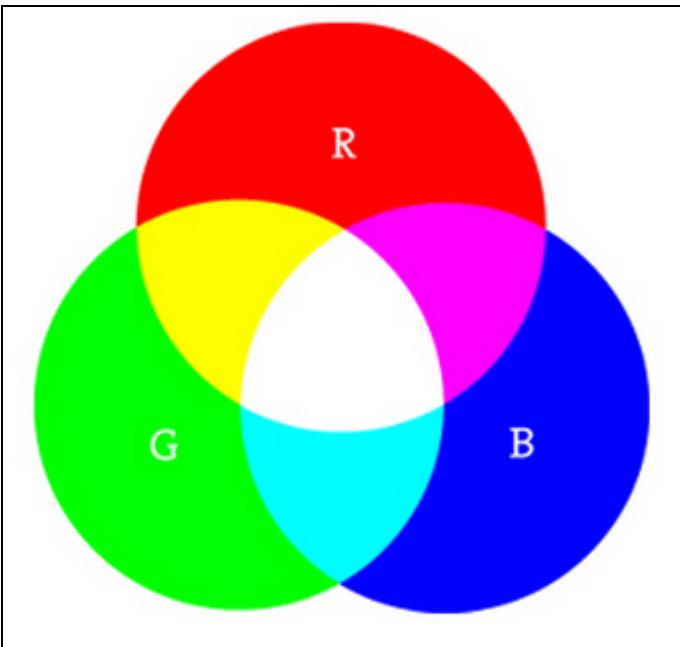
g = green colour (a number between 0 and 255 inclusive)

b = blue colour (a number between 0 and 255 inclusive)

- fills shapes with a chosen colour.
- can use the RGB colours to select a colour.
- all shapes drawn after the **fill** function is called, will be filled with the chosen colour.

# A Recap of RGB Colours

---



“As with grayscale, the individual color elements are expressed as ranges from 0 (none of that color) to 255 (as much as possible), and they are listed in the order R, G, and B.”

Digital colours are made by mixing the three primary colours of light (red, green, and blue).

# fill()

P lab01a\_solution\_step08 | Processing 2.2.1

File Edit Sketch Tools Help

Java

lab01a\_solution\_step08

```
size(400,300);
background(190,240,245);

rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);
ellipse(200,100,20,60);
ellipse(250,130,25,25);
```

lab01a\_solution\_step08

Starting code...

# fill()

P lab01a\_solution\_step08 | Processing 2.2.1

File Edit Sketch Tools Help

lab01a\_solution\_step08

Java

```
size(400,300);
background(190,240,245);

fill(100,150,70);

rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);
ellipse(200,100,20,60);
ellipse(250,130,25,25);
```

All shapes filled with dark green...

lab01a\_solution\_step08

The image shows a Processing sketch window titled "lab01a\_solution\_step08". The code in the editor defines a background color of 190, 240, 245. It then uses the fill() function with parameters 100, 150, 70 to set the fill color for all subsequent shapes. The sketch contains a large green rectangle with a diagonal line through it, and two smaller ellipses, one light green and one white, positioned above and to the right of the rectangle.

# fill()

P lab01a\_solution\_step08 | Processing 2.2.1

File Edit Sketch Tools Help

lab01a\_solution\_step08

```
size(400,300);
background(190,240,245);

fill(100,150,70);

rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

fill(200,250,70);

ellipse(200,100,20,60);
ellipse(250,130,25,25);
```

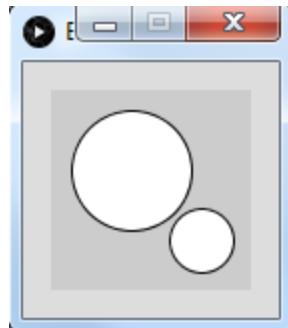
lab01a\_solution\_step08

The sketch displays a large green rectangle with a diagonal line through it. Two light green ellipses are positioned on the right side of the rectangle. A small dark green square is located at the bottom left corner of the green rectangle.

Rectangles filled with dark green...  
Ellipses filled with light green...  
Order of statements matter!!!

# fill() – More Examples

---



Starting code...

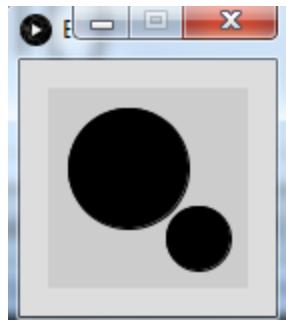
A screenshot of the Processing IDE showing a sketch titled "Ex\_19". The code in the editor is:

```
1 ellipse(40, 40, 60, 60); // Large circle
2 ellipse(75, 75, 32, 32); // Small circle
```

The code consists of two calls to the `ellipse` function. The first call creates a large circle at coordinates (40, 40) with a diameter of 60 pixels. The second call creates a small circle at coordinates (75, 75) with a diameter of 32 pixels. The code is written in Java syntax, as indicated by the "Java" dropdown menu in the top right of the IDE.

# fill() – More Examples

noFill()



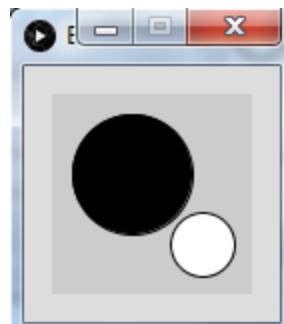
Circles filled with  
black

A screenshot of the Processing IDE. The title bar says "Ex\_19 | Processing 3.0b6". The code editor window is titled "Ex\_19" and contains the following Java code:

```
1 fill(0);
2 ellipse(40, 40, 60, 60); // Large circle
3 ellipse(75, 75, 32, 32); // Small circle
4
5
6
7
8
9
10
11
12
13
14
15
```

The code uses the fill() function to set the fill color to black (represented by the value 0) before drawing two ellipses. The first ellipse has a bounding box of [40, 40, 60, 60] and the second has a bounding box of [75, 75, 32, 32]. The Processing interface also shows tabs for "Console" and "Errors".

# fill() – More Examples



The screenshot shows the Processing IDE interface with the title bar "Ex\_19 | Processing 3.0b6". The code editor displays the following Java code:

```
1 fill(0);
2 ellipse(40, 40, 60, 60); // Large circle
3 fill(255);
4 ellipse(75, 75, 32, 32); // Small circle
5
6
7
8
9
10
11
12
13
14
15
16
```

The code uses the `fill()` function to set the fill color before drawing ellipses with the `ellipse()` function. The first ellipse is filled black and has a bounding box of (40, 40, 60, 60). The second ellipse is filled white and has a bounding box of (75, 75, 32, 32).

First Circle filled with black...  
Second Circle filled with white...  
**Order of statements matter!!!**

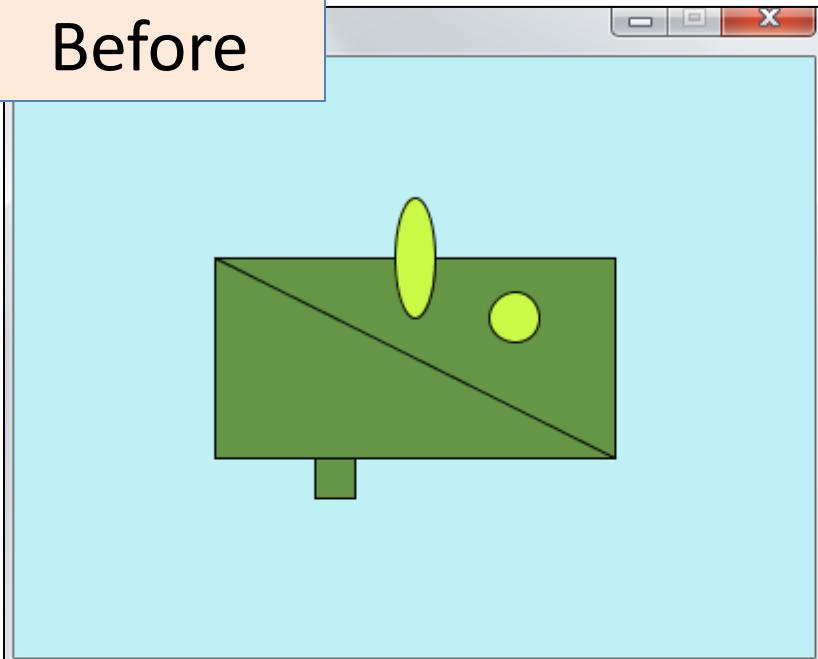
# Topics List

---

- Filling shapes with colour.
- Formatting the shape outline.
- Adding comments to your code.

# Changing the outline (i.e. stroke)

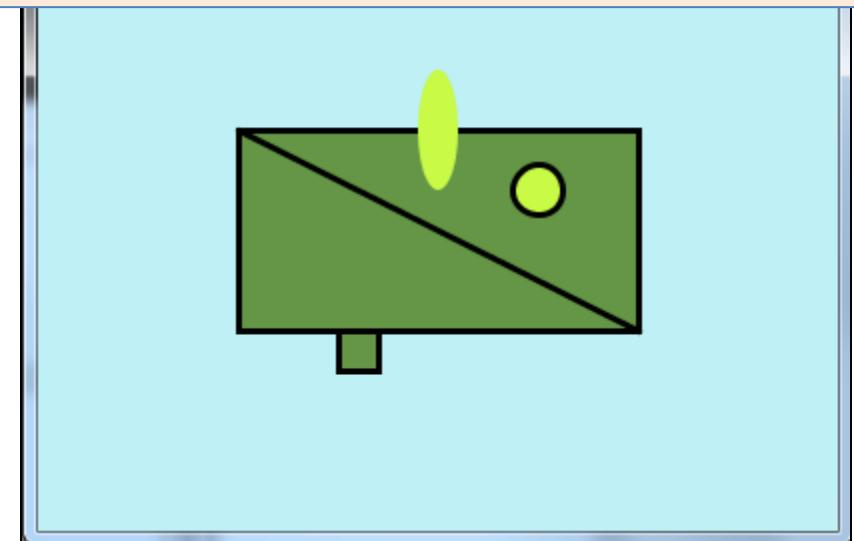
Before



After (changes):

- The oval has no border; all other shapes do.
- The outline is heavier.

We will now make those changes



# noStroke() - syntax

---

```
noStroke();
```

```
//no parameters defined for this function.
```

- A **stroke** is the outline of a shape.
- The noStroke() function disables the outline on shapes that are drawn after the function is called.
- All shapes drawn after the **noStroke** function is called, will have no outline.

# noStroke()

The screenshot shows the Processing IDE interface. On the left is the code editor window with the title "lab01a\_solution\_step08 | Processing 2.2.1". The code is as follows:

```
size(400,300);
background(190,240,245);

fill(100,150,70);

rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

fill(200,250,70);

noStroke();
ellipse(200,100,20,60);
ellipse(250,130,25,25);
```

A red oval highlights the line `noStroke();`. To the right is the preview window titled "lab01a\_solution\_step08" showing the output of the code. The sketch features a large green rectangle with a diagonal black line. Inside the rectangle are two yellow ovals: one at the top center and one at the bottom right. A small yellow square is located at the bottom edge of the rectangle.

✓ We have no border on the oval shape.  
✗ But now our circle also has no border.

# stroke() - syntax

---

**stroke (r, g, b)**

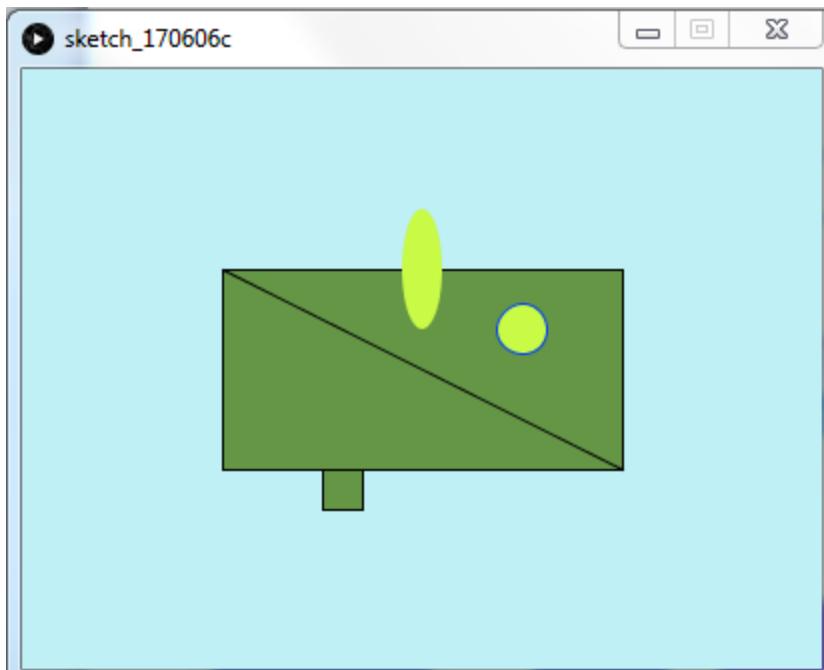
r = red colour (a number between 0 and 255 inclusive)

g = green colour (a number between 0 and 255 inclusive)

b = blue colour (a number between 0 and 255 inclusive)

- The stroke() function enables the outline on all shapes that are drawn after the function is called.
- When you call stroke(), you need to specify a colour.

# stroke()



✓ Our circle  
now has a blue  
border.

The Processing IDE window shows the code for the sketch:

```
background(190,240,245);
fill(100,150,70);
rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);
fill(200,250,70);
noStroke();
ellipse(200,100,20,60);
stroke(0,61,245);
ellipse(250,130,25,25);
```

The line `stroke(0,61,245);` is highlighted in light blue, indicating it is the current line of code being executed or selected.

# stroke()

P lab01a\_solution\_step08 | Processing 2.2.1

File Edit Sketch Tools Help

Java

lab01a\_solution\_step08

```
size(400,300);
background(190,240,245);

fill(100,150,70);

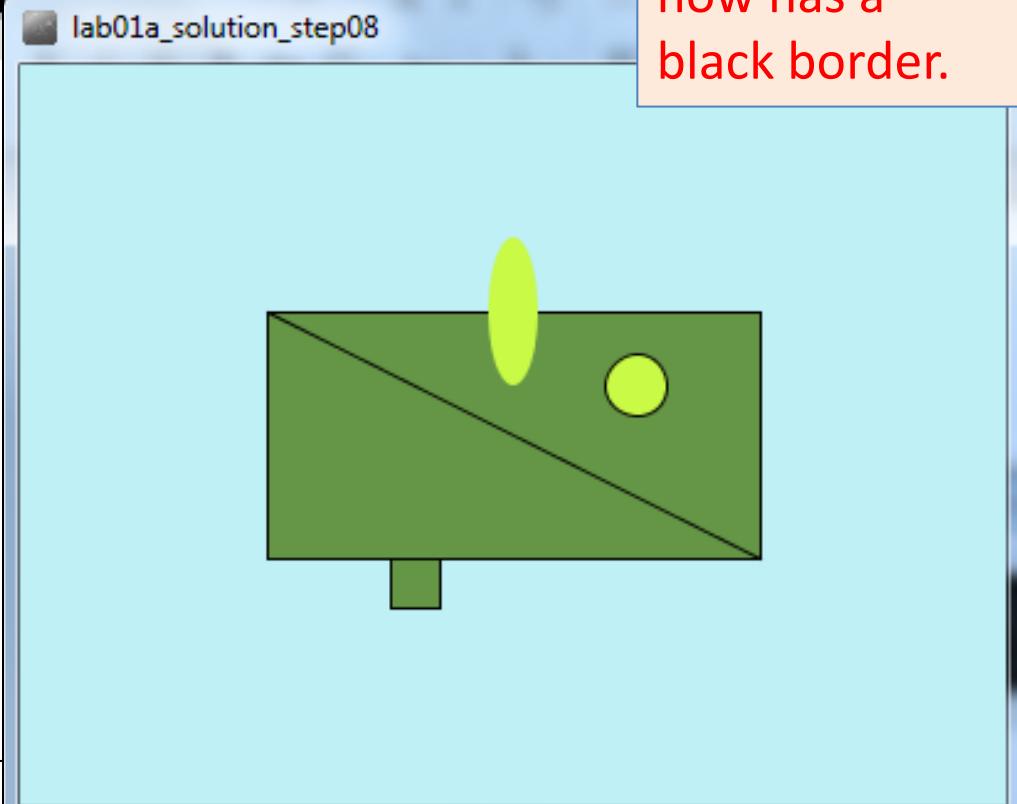
rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

fill(200,250,70);

noStroke();
ellipse(200,100,20,60);

stroke(0,0,0);
ellipse(250,130,25,25);
```

✓ Our circle now has a black border.



The Processing sketch window shows a green rectangle with a diagonal line through it. A small green ellipse is located near the top right, and a larger yellow-green ellipse with a black border is at the bottom right. The code in the editor includes a circled 'stroke(0,0,0)' line.

# strokeWeight() - Syntax

---

`strokeWeight (pixels)`

`pixels` = thickness of the outline measures in pixels.

- The `strokeWeight()` function allows you to choose the thickness of a line/outline on shapes.
- The chosen thickness will apply to all lines/shapes that are drawn after the function is called.
- The thickness is specified in pixels.
- The default thickness is 1 pixel.

# strokeWeight()

P lab01a\_solution\_step08 | Processing 2.2.1

File Edit Sketch Tools Help

Stop Java

lab01a\_solution\_step08

```
size(400,300);
background(190,240,245);
strokeWeight(3);

fill(100,150,70);

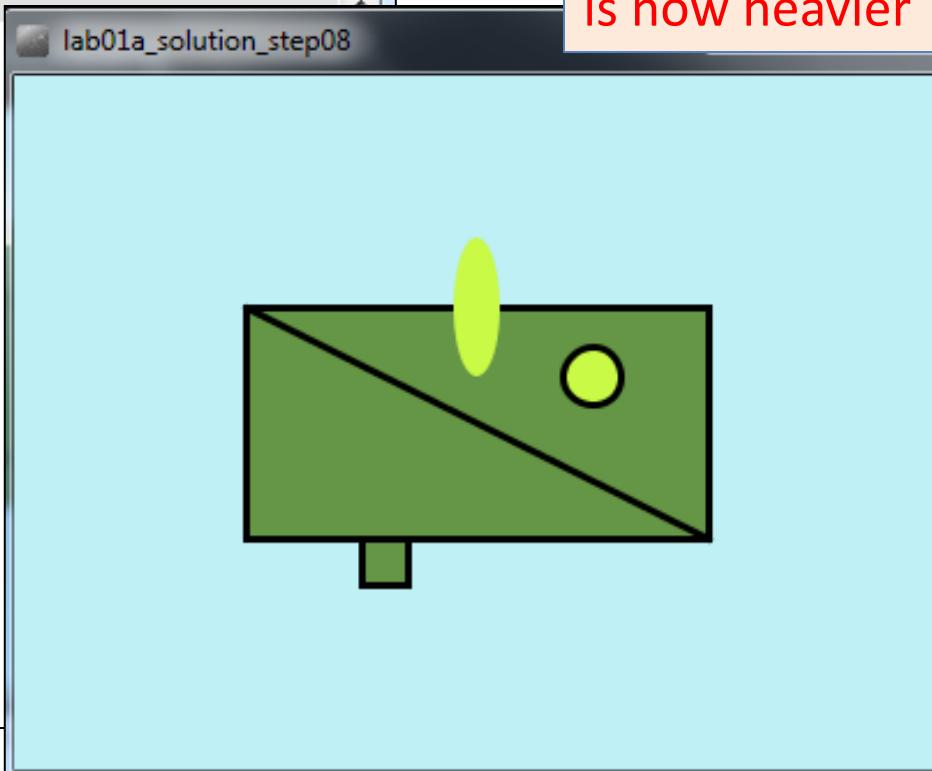
rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

fill(200,250,70);

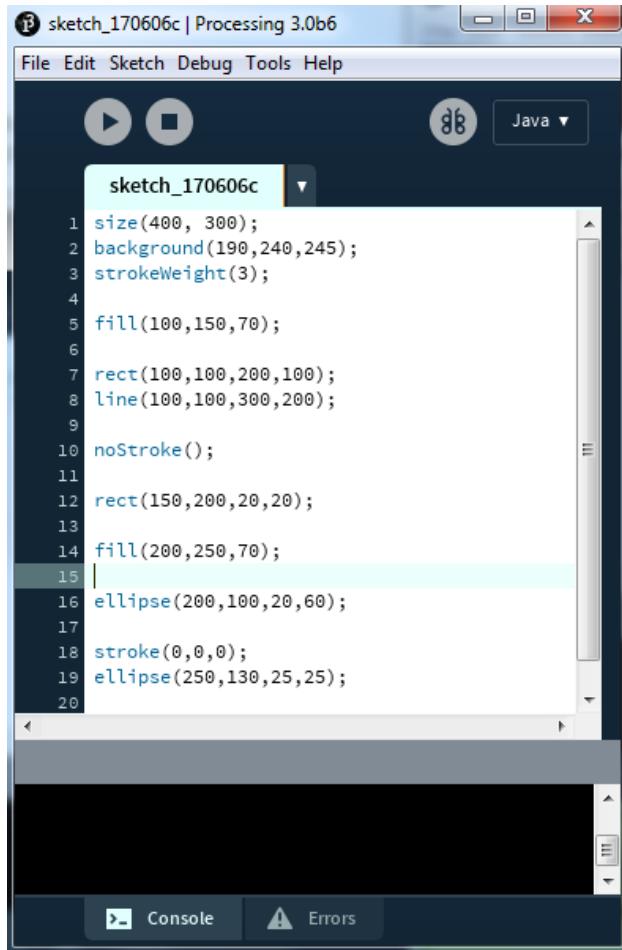
noStroke();
ellipse(200,100,20,60);

stroke(0,0,0);
ellipse(250,130,25,25);
```

✓ Our outline  
is now heavier



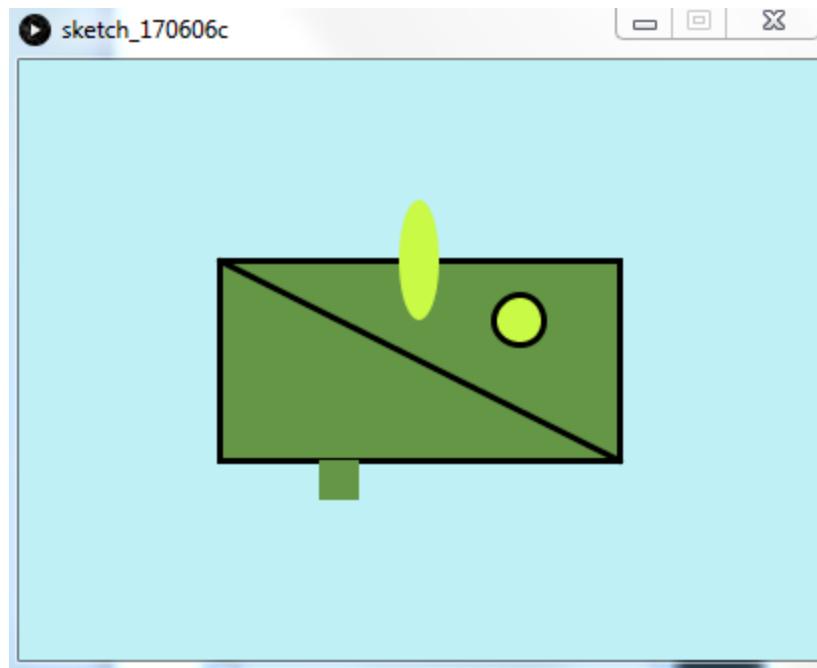
# strokeWeight()



The screenshot shows the Processing 3.0b6 IDE interface. The title bar says "sketch\_170606c | Processing 3.0b6". The menu bar includes File, Edit, Sketch, Debug, Tools, and Help. The toolbar has play and stop buttons, and a Java dropdown. The code editor window displays the following sketch code:

```
1 size(400, 300);
2 background(190,240,245);
3 strokeWeight(3);
4
5 fill(100,150,70);
6
7 rect(100,100,200,100);
8 line(100,100,300,200);
9
10 noStroke();
11
12 rect(150,200,20,20);
13
14 fill(200,250,70);
15
16 ellipse(200,100,20,60);
17
18 stroke(0,0,0);
19 ellipse(250,130,25,25);
20
```

The code uses strokeWeight(3) for the main rectangle and line, resulting in a thicker black outline. The other shapes (noStroke() rectangle and ellipse) have thin outlines.



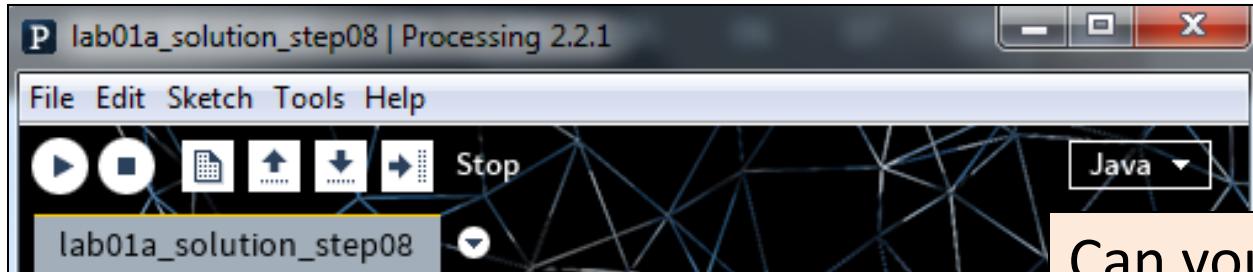
✓ Our outline  
is heavier but  
noStroke on 2  
shapes

# Topics list

---

- Filling shapes with colour.
- Formatting the shape outline.
- Adding comments to your code.

# Code so far...



```
P lab01a_solution_step08 | Processing 2.2.1
File Edit Sketch Tools Help
Stop Java
lab01a_solution_step08

size(400,300);
background(190,240,245);
strokeWeight(3);

fill(100,150,70);

rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

fill(200,250,70);

noStroke();
ellipse(200,100,20,60);

stroke(0,0,0);
ellipse(250,130,25,25);
```

Can you tell, from looking at the code, what RGB colours you have chosen?

- We can leave notes for ourselves and others in our code.
- This is called **commenting your code.**

# Commenting your code...

---

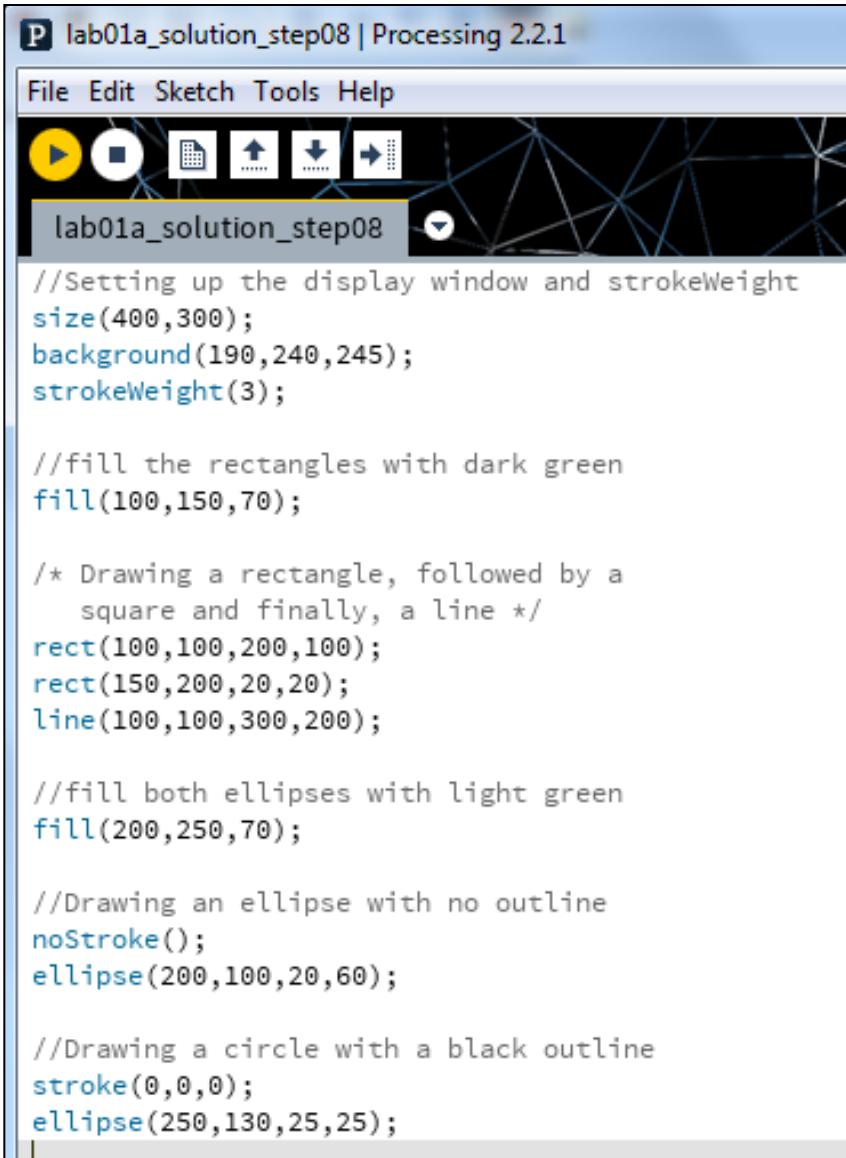
// This is a comment.

// Anything typed after the two slashes

// up to the end of the line, is ignored by Java.

/\* This is a longer comment. As you can span  
more than one line with this comment style, it  
can be quite handy. \*/

# Code so far...with commenting



The screenshot shows the Processing 2.2.1 interface with a sketch titled "lab01a\_solution\_step08". The sketch displays a dark blue background with a complex network of light blue lines forming a star-like geometric pattern. The code area contains the following pseudocode:

```
P lab01a_solution_step08 | Processing 2.2.1
File Edit Sketch Tools Help
lab01a_solution_step08
//Setting up the display window and strokeWeight
size(400,300);
background(190,240,245);
strokeWeight(3);

//fill the rectangles with dark green
fill(100,150,70);

/* Drawing a rectangle, followed by a
   square and finally, a line */
rect(100,100,200,100);
rect(150,200,20,20);
line(100,100,300,200);

//fill both ellipses with light green
fill(200,250,70);

//Drawing an ellipse with no outline
noStroke();
ellipse(200,100,20,60);

//Drawing a circle with a black outline
stroke(0,0,0);
ellipse(250,130,25,25);
```

We have commented our code with explanations of what is happening.

This makes our code easier to read, understand and maintain.

It is considered best practice to comment your code.

Comments do not affect your code at all.

# Questions?

---





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

Produced  
by:

Dr. Siobhán Drohan  
Mairead Meagher  
Sinéad Walsh



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics  
<http://www.wit.ie/>