Design Programming

DECO1012 & DECO2011

Rob Saunders

Rob Saunders

web: http://www.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

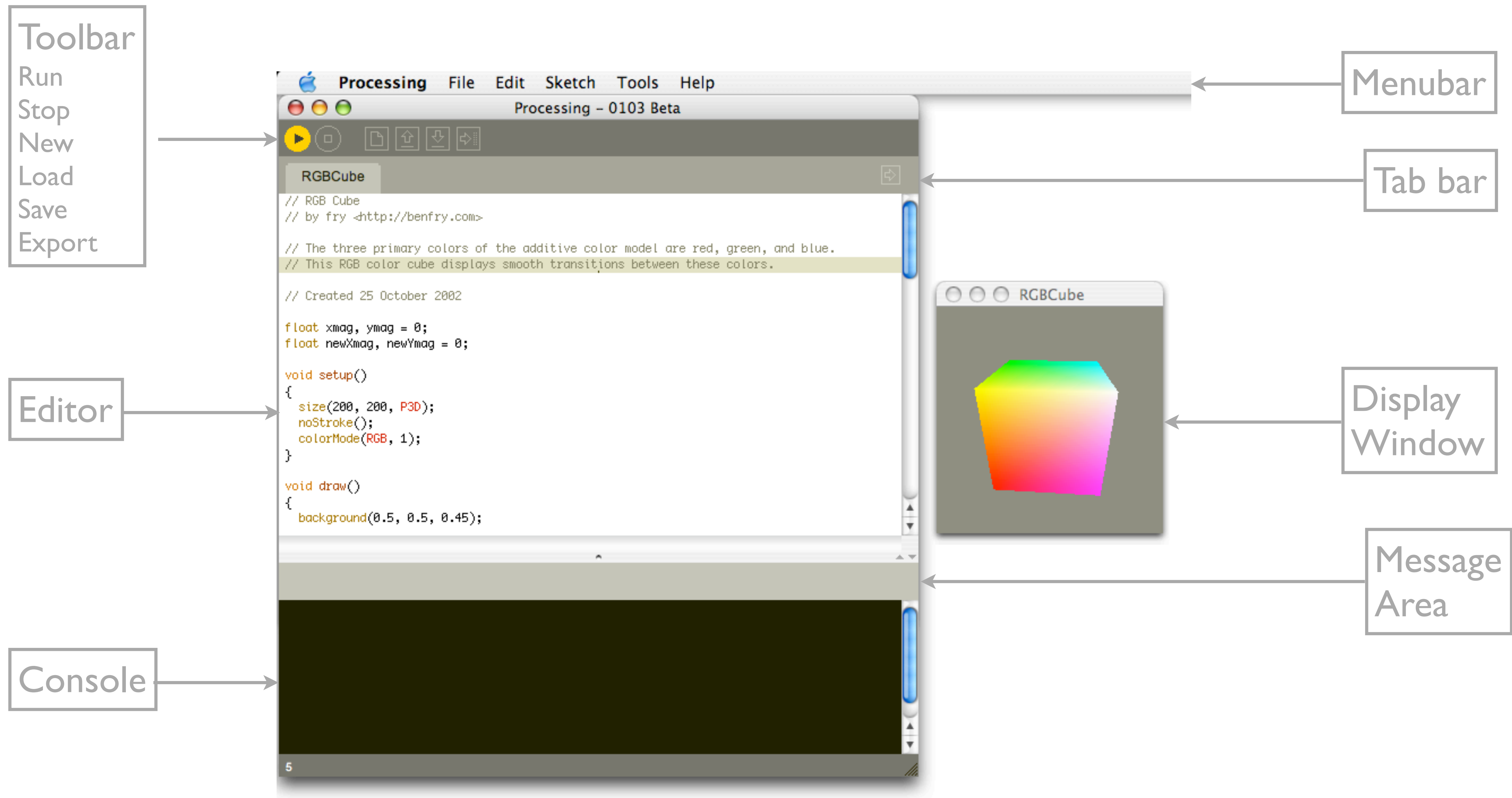
office: Room 274, Wilkinson Building

Introduction to Processing

What is Processing?

- Processing is a programming language and a development environment designed for creative people like you!
- The Processing programming language is similar to the popular programming language Java, but is easier to learn
- The Processing Development Environment (PDE) has been designed to be very simple to use to create small programs

The Processing Development Environment



Sketches

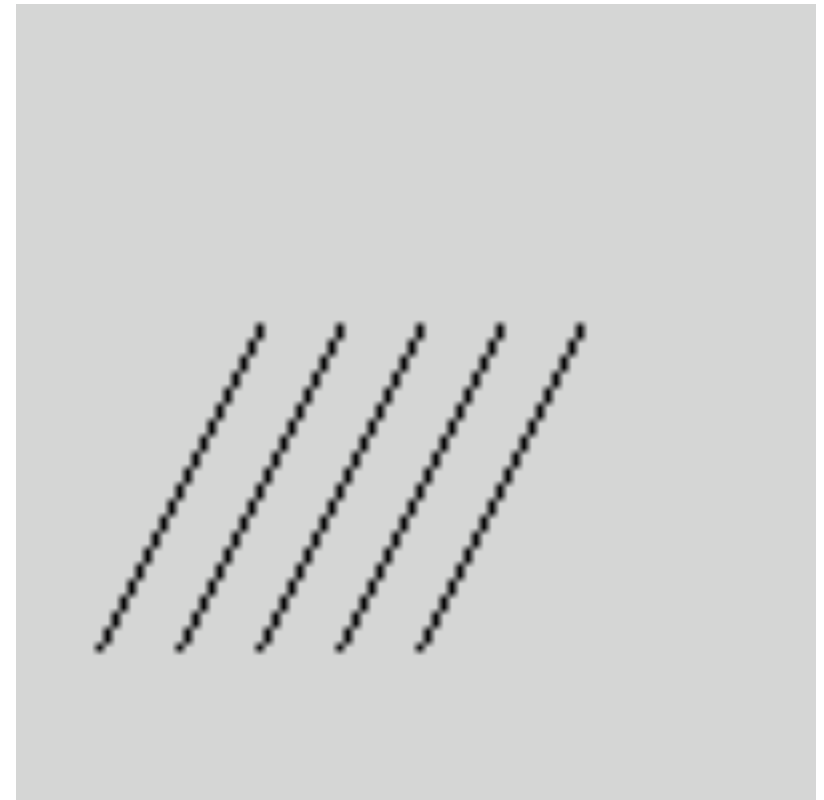
- Processing refers to a program and its associated data files as a “sketch”
- All sketches can be accessed from the File menu item “File > Sketchbook”
- All new sketches get an automatically generated name, e.g. “sketch_060308a”
- You can rename a sketch using the arrow button in the tab bar and choosing “Rename”

Publishing Sketches

- Select menu item “File > Export to web” or click on the export icon in the toolbar
- Processing will generate a number of files in a new folder called “applet” that include a simple webpage, the compiled program and data files, and the program’s source code
- The sketch can be published by simply uploading the applet directory to a website

Developing a Sketch

```
line(10, 80, 30, 40);
line(20, 80, 40, 40);
line(30, 80, 50, 40);
line(40, 80, 60, 40);
line(50, 80, 70, 40);
```



```
background(0);
stroke(255);
strokeWeight(5);
smooth();
line(10, 80, 30, 40);
line(20, 80, 40, 40);
line(30, 80, 50, 40);
line(40, 80, 60, 40);
line(50, 80, 70, 40);
```

```
background(0);
stroke(255);
smooth();
strokeWeight(1);
line(10, 80, 30, 40);
strokeWeight(2);
line(20, 80, 40, 40);
strokeWeight(3);
line(30, 80, 50, 40);
strokeWeight(4);
line(40, 80, 60, 40);
strokeWeight(5);
line(50, 80, 70, 40);
```

```
int x = 10;
int y = 70;

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

void draw() {
  background(0);
  stroke(255);
  strokeWeight(1);
  line(x+10, y, x+30, y-40);
  strokeWeight(2);
  line(x+20, y, x+40, y-40);
  strokeWeight(3);
  line(x+30, y, x+50, y-40);
  strokeWeight(4);
  line(x+40, y, x+60, y-40);
  strokeWeight(5);
  line(x+50, y, x+70, y-40);
  x = x + 1;
  if (x > 100) { x = -80; }
}
```

```
int x = 10;
int y = 70;

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

void draw() {
  background(0);
  stroke(255);
  strokeWeight(1);
  line(x+10, y, x+30, y-40);
  strokeWeight(2);
  line(x+20, y, x+40, y-40);
  strokeWeight(3);
  line(x+30, y, x+50, y-40);
  strokeWeight(4);
  line(x+40, y, x+60, y-40);
  strokeWeight(5);
  line(x+50, y, x+70, y-40);
  x = x + 1;
  if (x > 100) { x = -80; }
}
```

Elements of Code

Writing Program Code

- The text that we write to develop programs is called “source code” or simply “code”
- Writing code is different from writing text for humans because we have to obey strict rules about the form of code, so that a computer can compile it and execute it
  - Compiling source code turns it into a program that is easier for the computer to execute but even more difficult for humans to write

Comments

- Comments are ignored by a computer but are important for people
  - Because code often requires a lot of effort to read and understand, it is important that you document your code as you write it
  - Comments allow you to leave notes about your code so that (a) other people can understand what it does, and (b) you can understand what your code does when you come back to it

Comments

```
// Two forward slashes are used to denote a comment.

// All text on the same line is a part of the comment.

// There must be no spaces between the slashes. For example,
// the code "/ /" is not a comment and will cause an error

size(200, 200); // Comments can be added after some code

/*
  If you want to have a comment that is many lines long,
  you may prefer to use the syntax for a multiline comment

  A forward slash followed by an asterisk allows the
  comment to continue until the opposite
*/
```

Using Functions

- Functions allow you to draw shapes, set colors, calculate numbers, and to execute many other types of actions.
  - A function's name is usually a lowercase word followed by parentheses.
- The comma-separated elements between the parentheses are called parameters, and they affect the way the function works.
  - Some functions have no parameters and others have many.

Using Functions

```
// The size function has two parameters. The first sets the
// width of the display window and the second sets the height

size(200, 200);

// This version of the background function has one parameter.
// It sets the grey value for the background of the display
// window in the range of 0 (black) to 255 (white)

background(102);

// This version of the background function has three
// parameters. It sets the colour value for the background of
// the display window using RGB (or HSB) colour components

background(127, 191, 15);
```

Expressions

- An expression is like a phrase in English
  - An expression always has a value, determined by evaluating its contents.
  - An expression can be as basic as a single number or as complex as a long mathematical function.

Expressions

Expression	Value
`5`	`5`
`122.3 + 3.1`	`125.4`
`((3 + 2) * -10) + 1`	`-49`
`“jack” + “ & ” + “jill”`	`“jack & jill”`
`6 > 3`	`true`
`54 < 50`	`false`

Statements

- A set of expressions create a statement, the programming equivalent of a sentence.
  - Every statement ends with the terminator, the programming equivalent of a full stop / period.
- In the Processing language, the statement terminator is a semicolon.
  - It is important to remember to use a semicolon at the end of every statement.

Statements

```
// There are different types of statements. A statement can
// define a variable, assign a variable, run a function, or
// construct an object.

size(200, 200);  // Runs the size() function

int x;           // Declares a new variable x

x = 102;         // Assigns the value 102 to the variable x

background(x);   // Runs the background() function 1-03

PImage img = new PImage(); // Constructs a PImage object
```

Comment

```
// Create a 300 x 400 window
size(300, 400);
background(0);
```

Function

Statement terminator

Statement

Parameter

Case Sensitivity

- The Processing language is case sensitive.
  - It differentiates between uppercase and lowercase characters; writing “`Size`” when you mean to write “`size`” creates an error.
- You must be careful to match the case of functions and variables in your code.

Whitespace

- In many programming languages, including Processing, you can use as much space as you want between code elements.
  - Using space to arrange your code neatly can greatly help reading and fixing your programs.

Console

- When software runs, the computer performs operations at a rate too fast to perceive with human eyes.
- It is important to understand what is happening inside the machine, the functions `print()` and `println()` can be used to display data in the console area while a program is running.

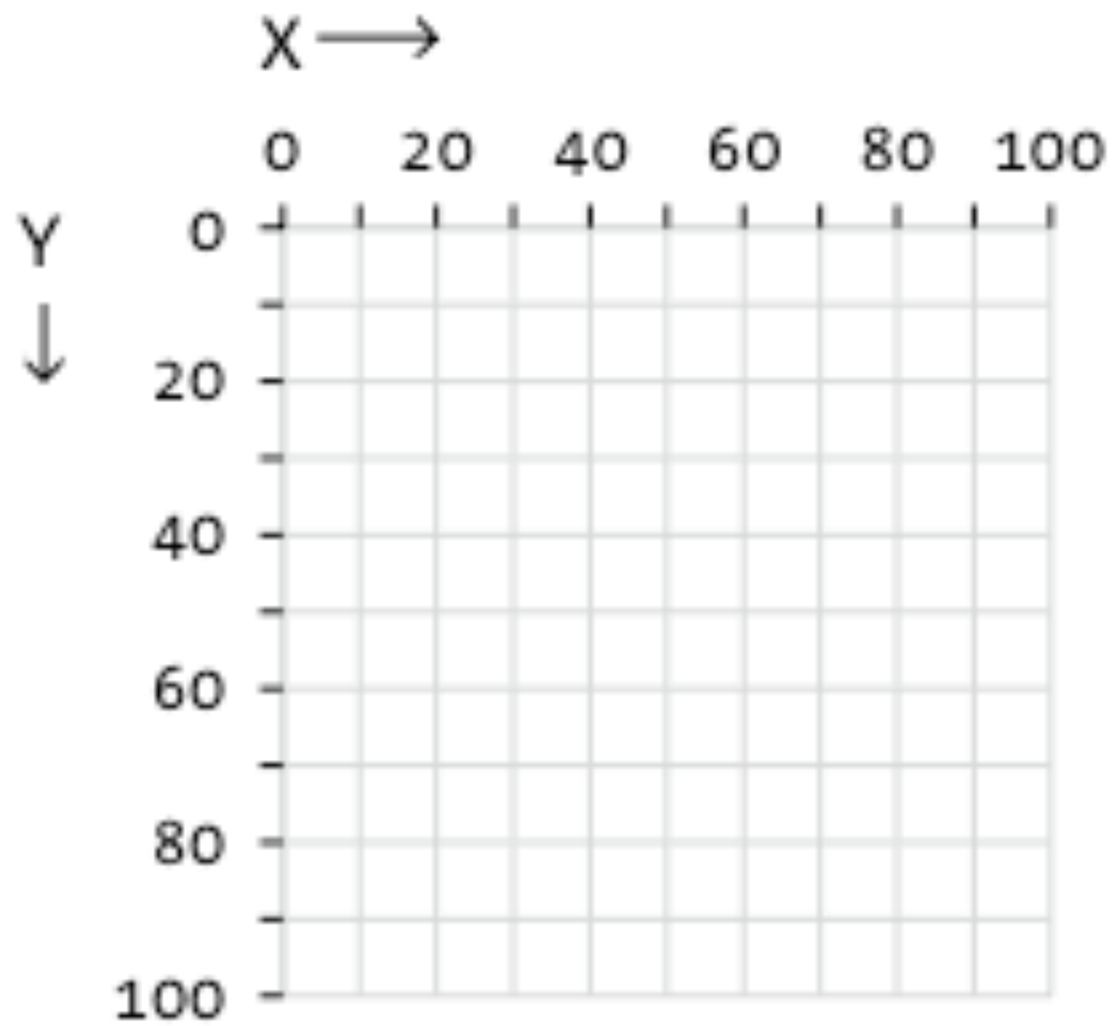
Drawing with Code

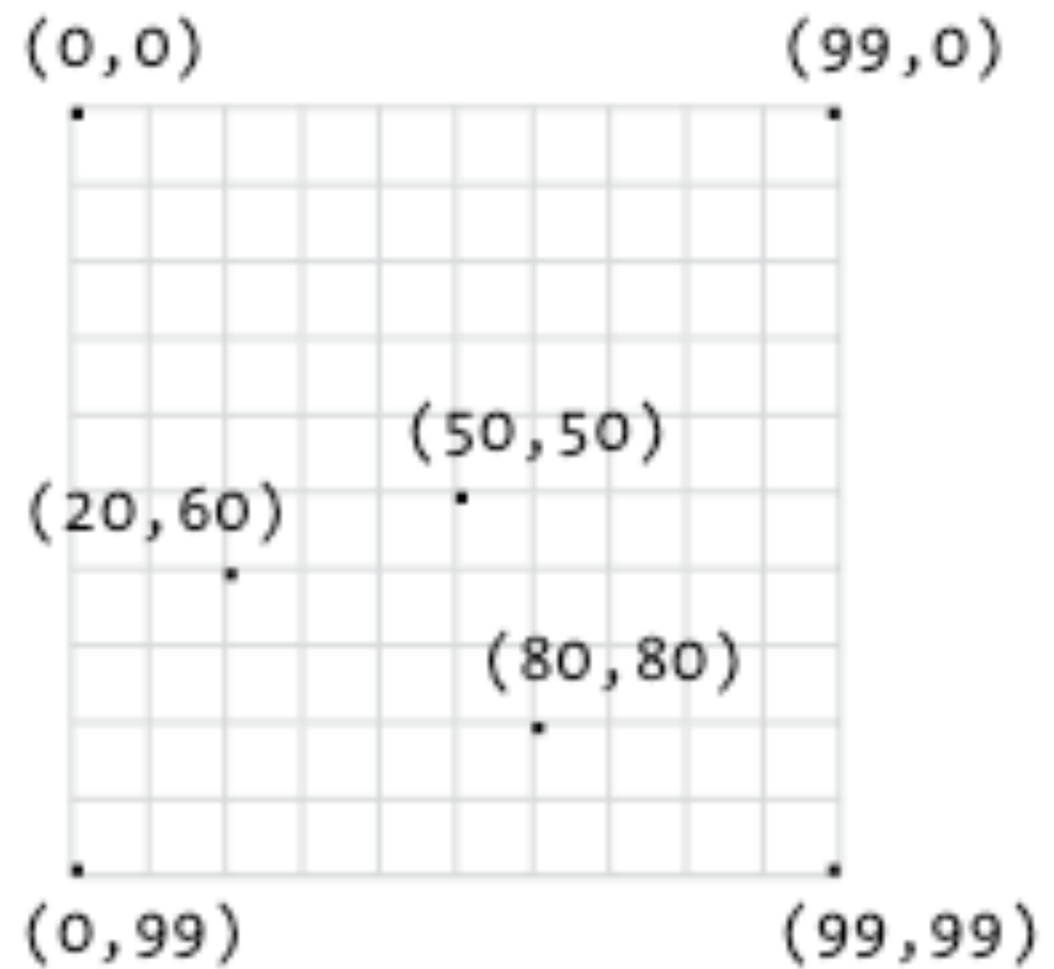
Creating a Canvas

- A computer screen is a grid of small light elements called pixels.
  - Modern computer screens have between 1 and 2 million pixels.
- Processing draws using pixels contained within its display window.
  - The size of the display window is controlled with the `size()` function.

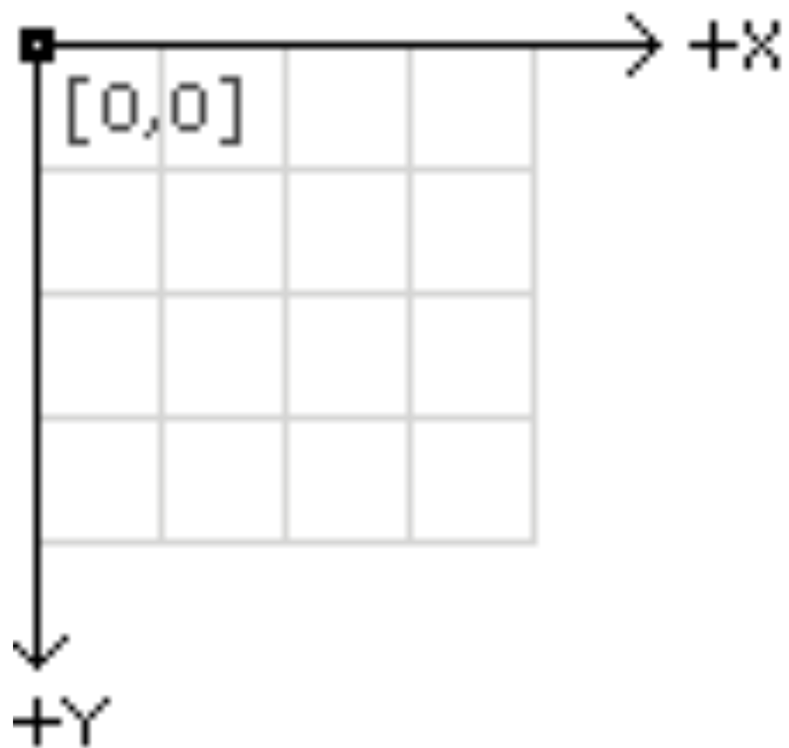
```
size(width, height)
```

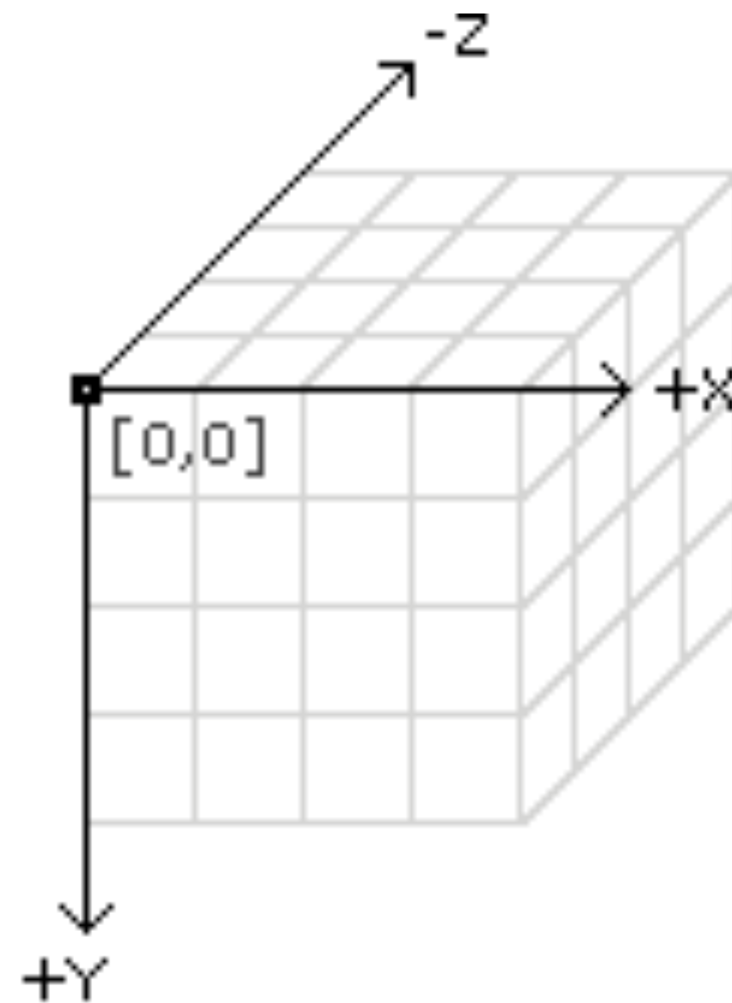
Coordinates





Coordinates





2D coordinates = (x, y)

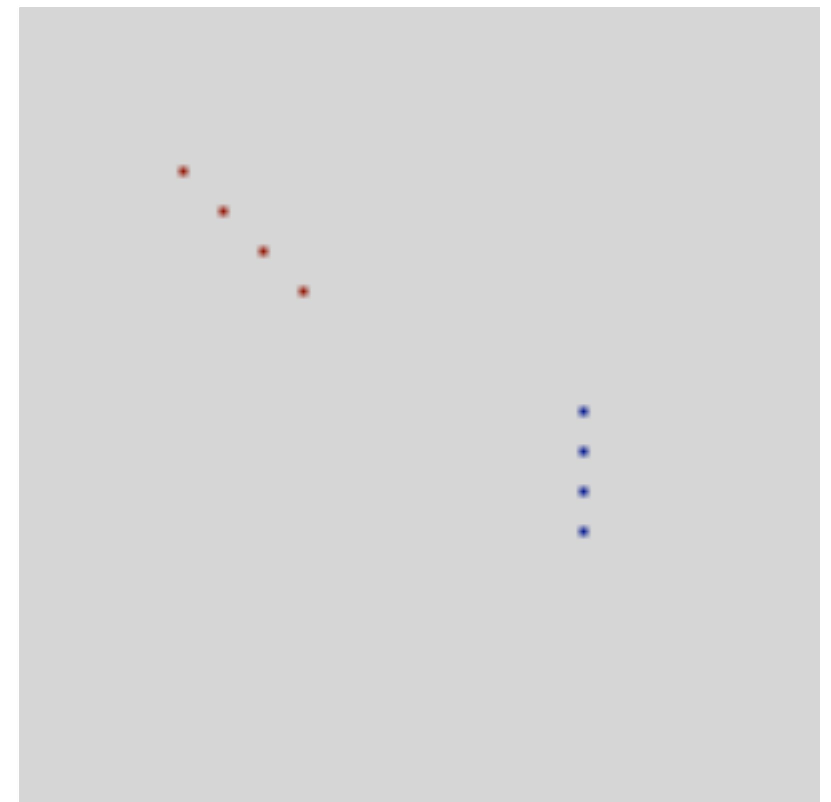
3D coordinates = (x, y, z)

Drawing Points

A single pixel can be drawn using the `point()` function.

```
stroke(32, 0, 0);
point(20, 20);
point(25, 25);
point(30, 30);
point(35, 35);

stroke(0, 0, 32);
point(70, 50);
point(70, 55);
point(70, 60);
point(70, 65);
```

point(x, y)
function

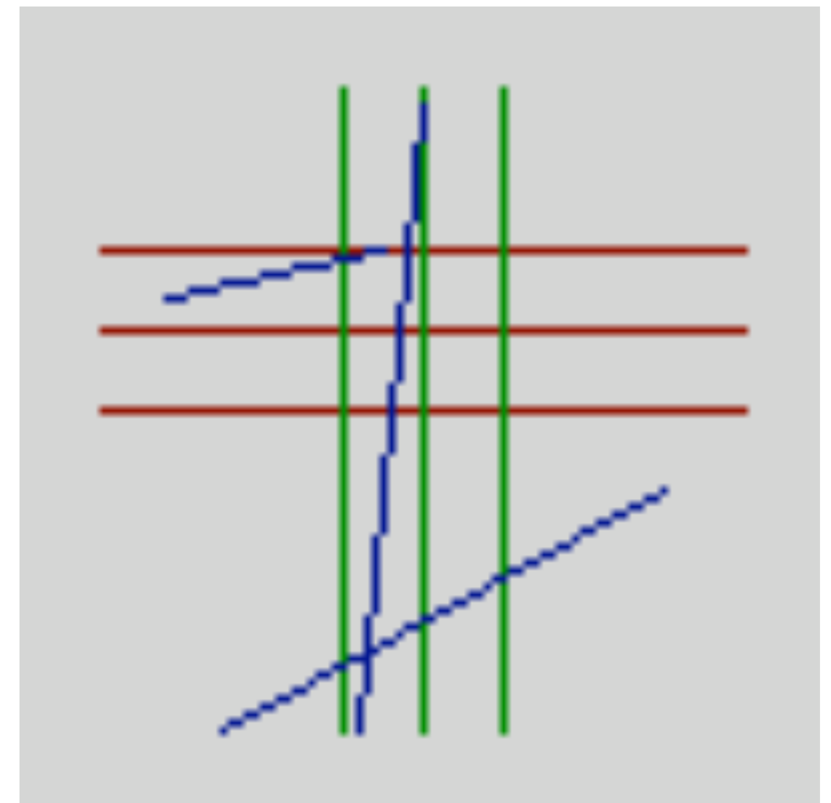


Drawing Lines

Lines are drawn using the `line()` function.

```
stroke(127, 0, 0);
line(10, 30, 90, 30);
line(10, 40, 90, 40);
line(10, 50, 90, 50);

stroke(0, 127, 0);
line(40, 10, 40, 90);
line(50, 10, 50, 90);
line(60, 10, 60, 90);

stroke(0, 0, 127);
line(25, 90, 80, 60);
line(50, 12, 42, 90);
line(45, 30, 18, 36);
```

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

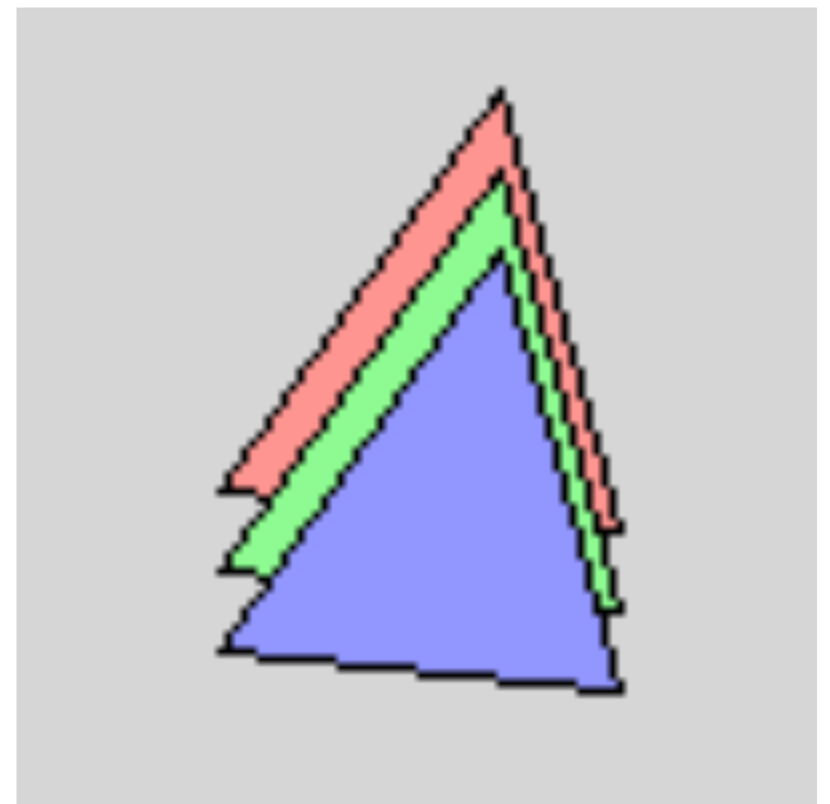


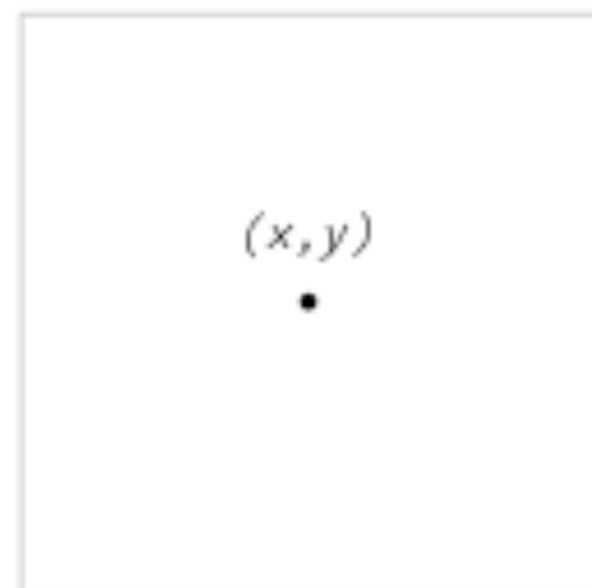
Drawing Triangles

Triangles are drawn using the `triangle()` function.

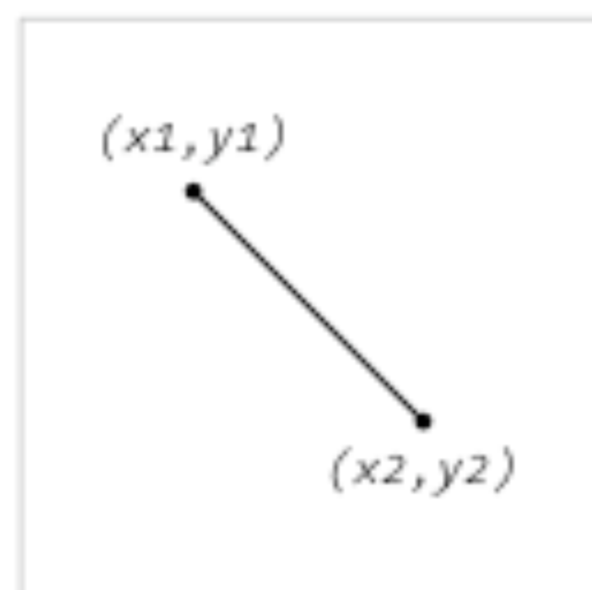
```
triangle(x1, y1, x2, y2, x3, y3)
```

```
fill(255, 127, 127);
triangle(60, 10, 25, 60, 75, 65);

fill(127, 255, 127);
triangle(60, 20, 25, 70, 75, 75);

fill(127, 127, 255);
triangle(60, 30, 25, 80, 75, 85);
```

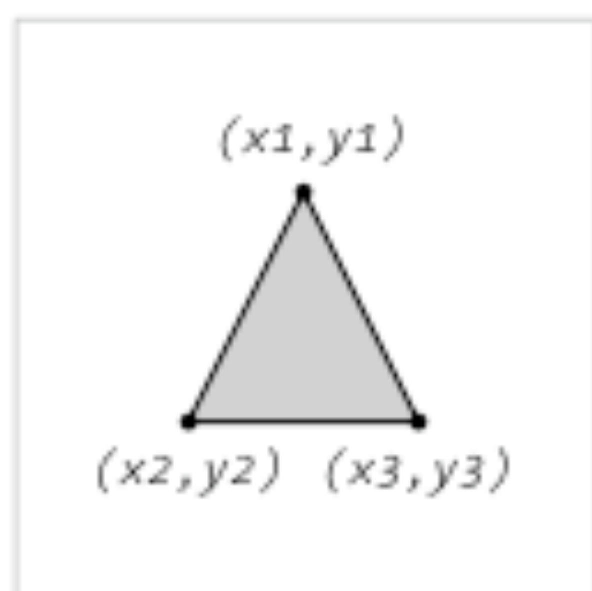




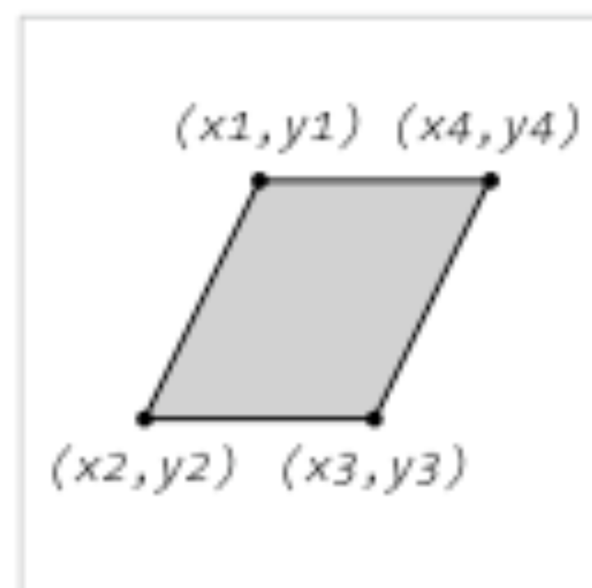

```
point(x, y)
```



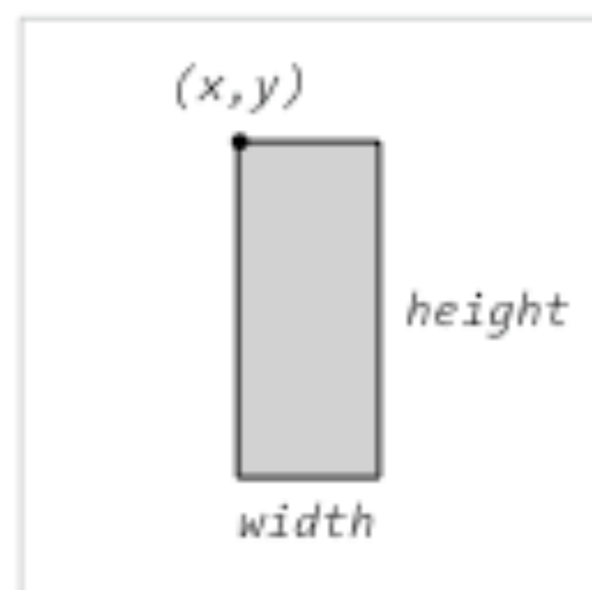

```
line(x1, y1, x2, y2)
```



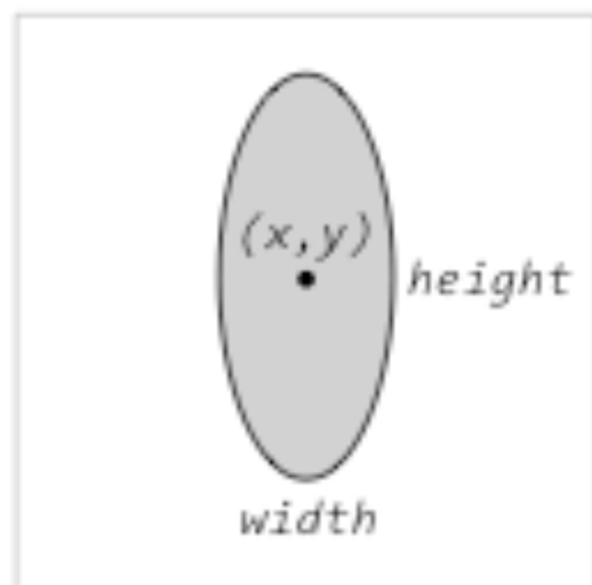

```
triangle(x1, y1, x2, y2, x3, y3)
```



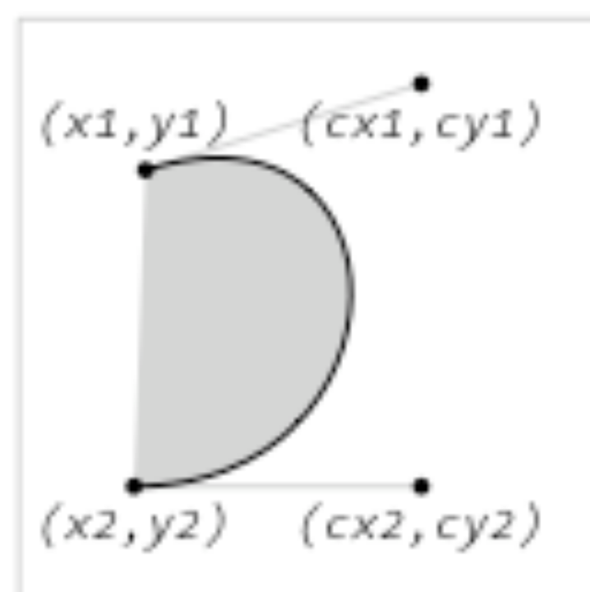

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```




```
rect(x, y, width, height)
```




```
ellipse(x, y, width, height)
```



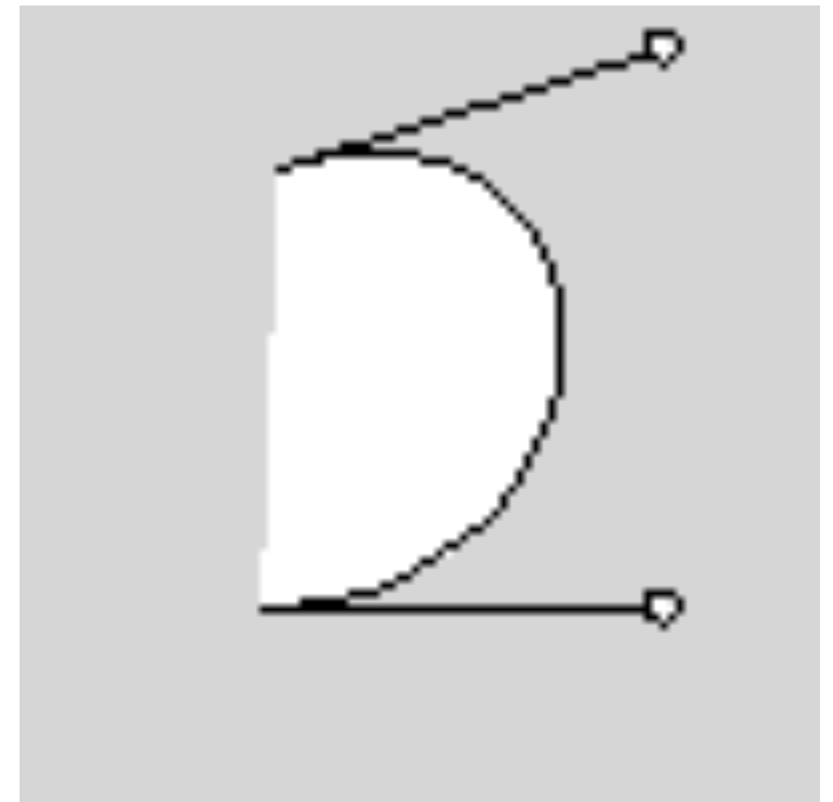

```
bezier(x1, y1, cx1, cy1, cx2, cy2, x2, y2)
```

Bezier Curves

Bezier curves are drawn by specifying start, end and two control points using the `bezier()` function.

```
bezier(x1,y1,cx1,cy1,cx2,cy2,x2,y2)
```

```
bezier(32,20,80,5,80,75,30,75);

// Draw the control points
line(32, 20, 80, 5);
ellipse(80, 5,4,4);
line(80,75,30,75);
ellipse(80, 75,4,4);
```

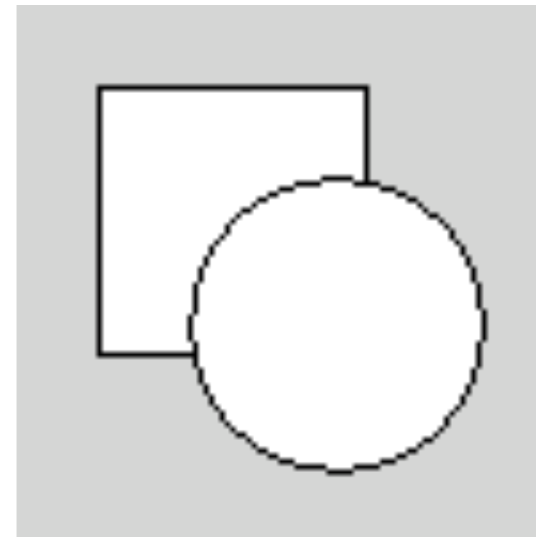


Notice that Bezier curves are filled.

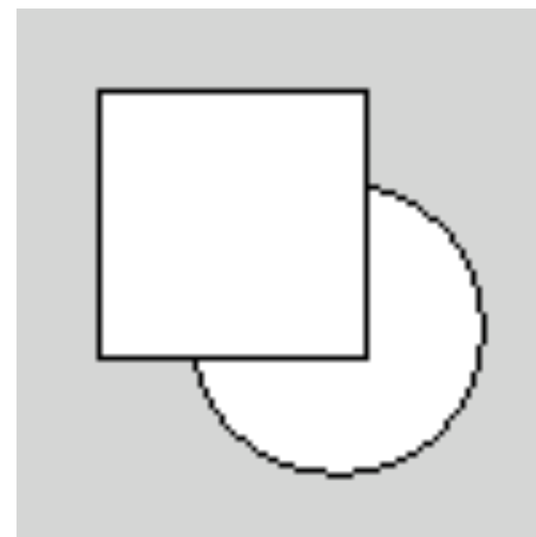
Drawing Order

The order that drawing commands appear in your code affects how shapes overlap on screen.

```
rect(15,15,50,50);
ellipse(60, 60,55,55);
```



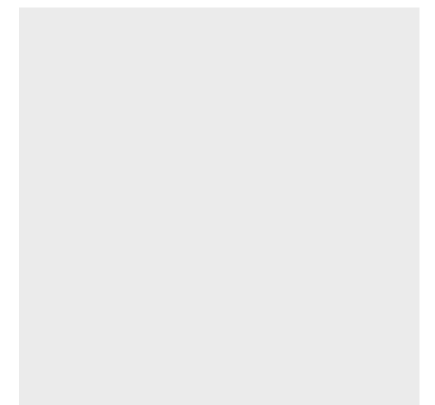
```
ellipse(60, 60,55,55);
rect(15,15,50,50);
```



Greyscale Values

We can specify greyscale colour values for drawing and filling using a single number between 0 and 255.

```
background(0);

background(124);

background(230);
```

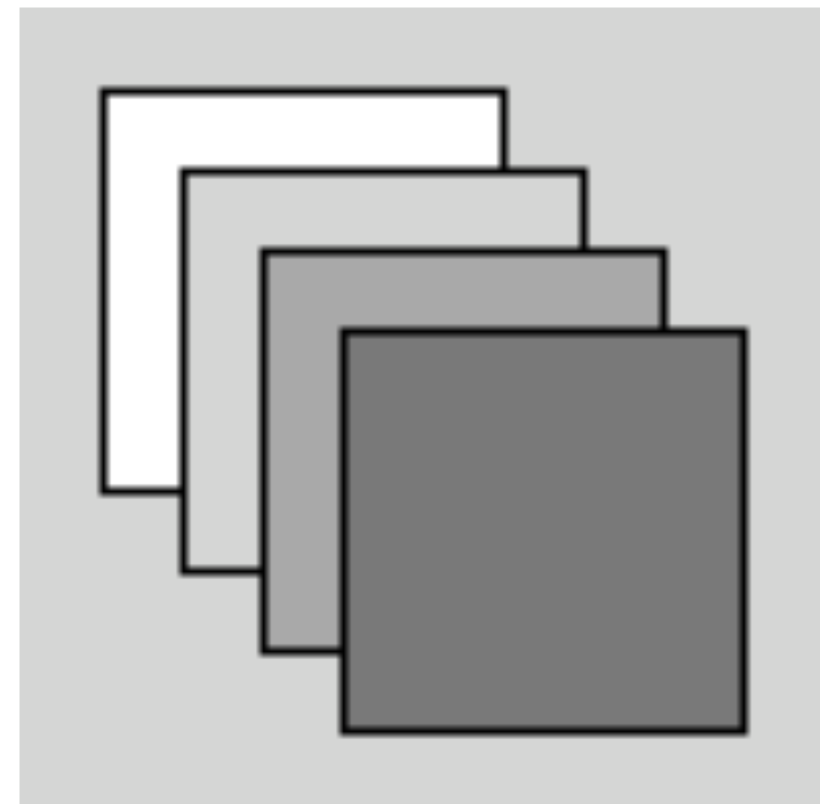


Filling Shapes

The colour that closed shapes, like rectangles, are filled with can be controlled using the `fill()` function.

`fill(g)`

```
rect(10,10,50,50);

fill(204);
rect(20, 20, 50,50);

fill(153);
rect(30,30,50,50);

fill(102);
rect(40,40,50,50);
```

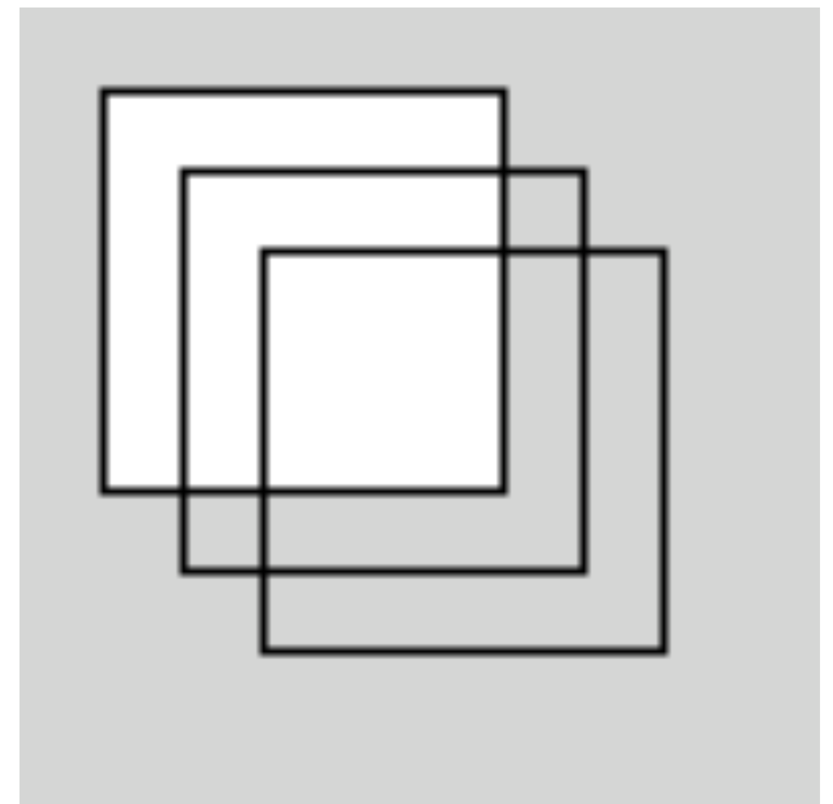


Filling Shapes

The filling of closed shapes can be turned off using the `noFill()` function.

`noFill()`

```
rect(10,10,50,50);

noFill(); // Disable fill

rect(20, 20, 50,50);
rect(30,30,50,50);
```



Stroking Shapes

The colour used to draw the outline of shapes is controlled using the `stroke()` function.

```
background(255);

rect(10,10,50,50);

stroke(102);
rect(20, 20, 50,50);

stroke(153);
rect(30,30,50,50);

stroke(204);
rect(40,40,50,50);
```

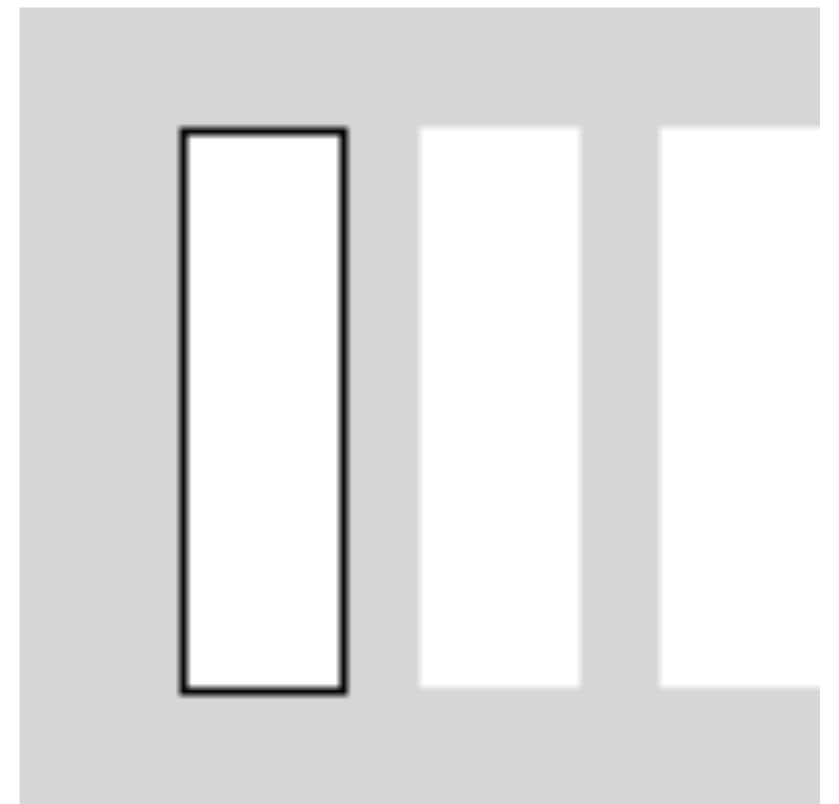
`stroke(g)`

Stroking Shapes

The outline of shapes can be disabled using the `noStroke()` function.

noStroke()

```
rect(20,15, 20, 70);

noStroke(); // Disable stroke

rect(50,15, 20, 70);
rect(80,15, 20, 70);
```

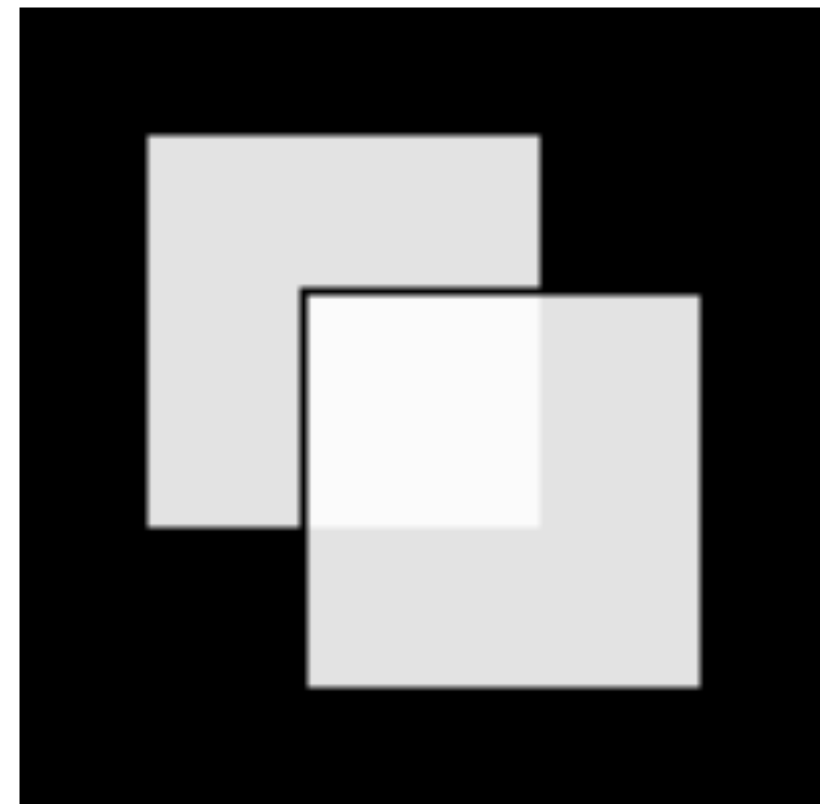


Opacity

The function `fill()` can take a second parameter that controls the opacity (or alpha channel) of the drawing.

`fill(g, a)`

```
background(0);

fill(255, 220);

rect(15, 15, 50, 50);
rect(35, 35, 50, 50);
```

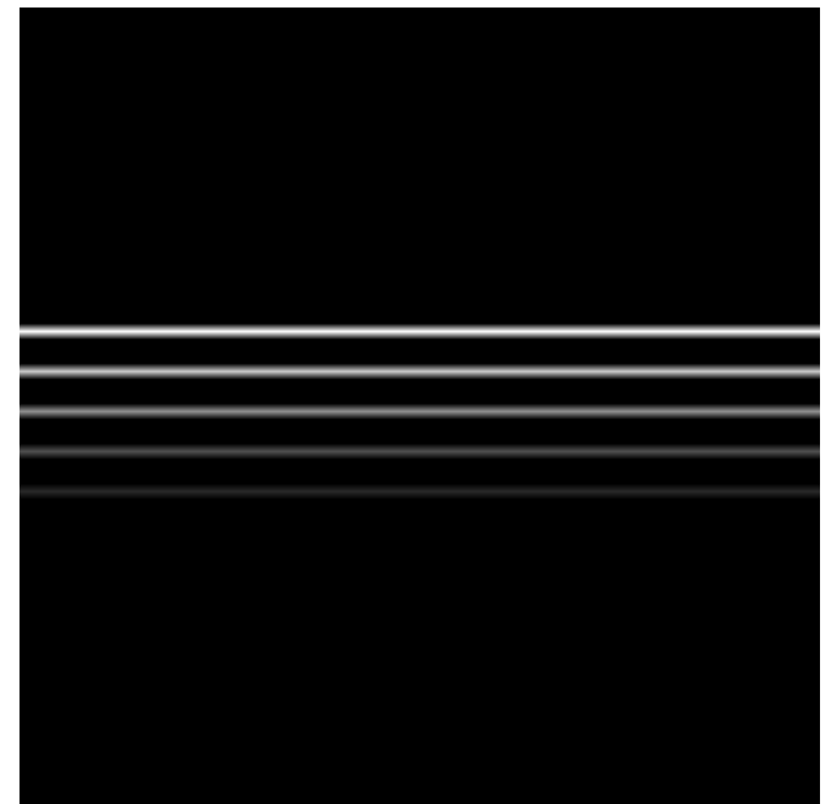


Opacity

The `stroke()` function can also take a second parameter to control opacity.

`stroke(g, a)`

```
background(0);
stroke(255, 255);
line(0, 40, 100, 40);
stroke(255, 191);
line(0, 45, 100, 45);
stroke(255, 127);
line(0, 50, 100, 50);
stroke(255, 63);
line(0, 55, 100, 55);
stroke(255, 31);
line(0, 60, 100, 60);
```

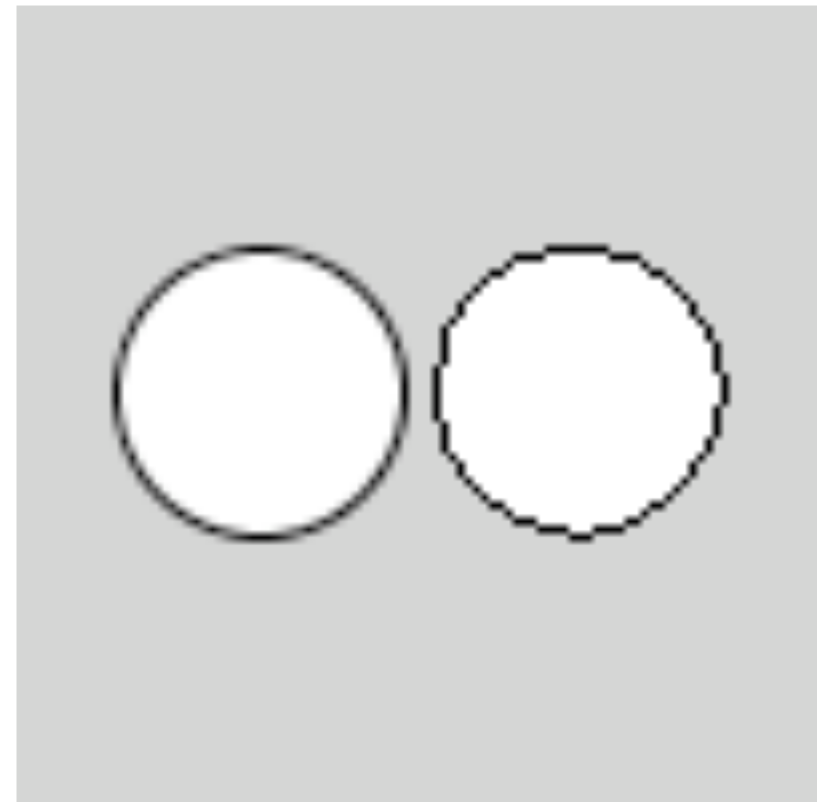


Smooth Drawing

Processing supports the drawing of shapes with anti-aliasing to smooth out jagged edges produced by pixels.

`smooth() / noSmooth()`

```
smooth();
ellipse(30, 48,36,36);

noSmooth();
ellipse(70, 48,36,36);
```



Line Attributes

The appearance of lines can be controlled using the functions `strokeWeight()`, `strokeCap()` and `strokeJoin()`

`strokeWeight(t)`

```
smooth();

line(20, 20, 80, 20);

strokeWeight(6);
line(20,40, 80, 40);

strokeWeight(18);
line(20,70, 80, 70);
```

Capping Lines

The function `strokeCap()` controls the shape of the ends of lines, useful for thick lines.

`strokeCap(TYPE)`

```
smooth();
strokeWeight(12);

strokeCap(ROUND);
line(20,30, 80, 30);

strokeCap(SQUARE);
line(20,50, 80, 50);

strokeCap(PROJECT);
line(20,70, 80, 70);
```

Joining Lines

The function `strokeJoin()` controls how lines are joined at the corners of shapes.

`strokeJoin(TYPE)`

```
smooth();
strokeWeight(12);

strokeJoin(BEVEL);
rect(12,33,15,33);

strokeJoin(MITER);
rect(42,33,15,33);

strokeJoin(ROUND);
rect(72,33,15,33);
```

Drawing Modes

The way that functions like **`rect()`** and **`ellipse()`** draw shapes can be affected by changing the drawing mode.

```
smooth();
noStroke();

fill(126);
ellipseMode(RADIUS);
ellipse(33,33,60,60);

fill(255);
ellipseMode(CORNER);
ellipse(33,33,60,60);

fill(0);
ellipseMode(CORNERS);
ellipse(33,33,60,60);
```

`ellipseMode(MODE)`

Lab Exercises

- Install Processing on a lab machine’s Data drive
- Write comments in Processing to describe a program you would like to create
- Write a program to create a 640 x 480 pixel window with a black background
- Use `print()` and `println()` to write some text to the console area in Processing

Lab Exercises

- Create a composition by carefully positioning one line and one ellipse
- Modify the code for exercise 1 to change the fill, stroke and background values
- Create a visual knot using only Bezier curves