Introduction to Programming

The aim of these notes is to introduce you to the basic knowledge of programming in the Processing environment that can be downloaded from the official website www.processing.org

Introduction

Once the Processing Integrated Development Environment (IDE) is started, you will be presented with something similar to Figure 1 below.

Figure 1: The Processing IDE

The code is written in the blank white space and is executed by pressing the 'play' button on the top left of the window. Any errors or messages from the compiler will be displayed in the grey strip beneath the code window.

Anatomy of a Program in Processing

Primary Methods

Methods are blocks of code that the compiler executes. They allow us to code in a modular manner and thus making code more reusable. While there can be differences, methods can also be referred to as functions.

In processing, there are 2 methods that we, mainly, will be using to write the code within. These are the `setup()` and `draw()` methods. In the setup() method we enter code that sets up the sketch such as the size of the screen/canvas and the colour of the background.

On the other hand, the draw() method is generally used to write code that creates the main artefacts of the program and includes the main program logic. This will become more evident in the examples that will be provided at a later stage.

Functions

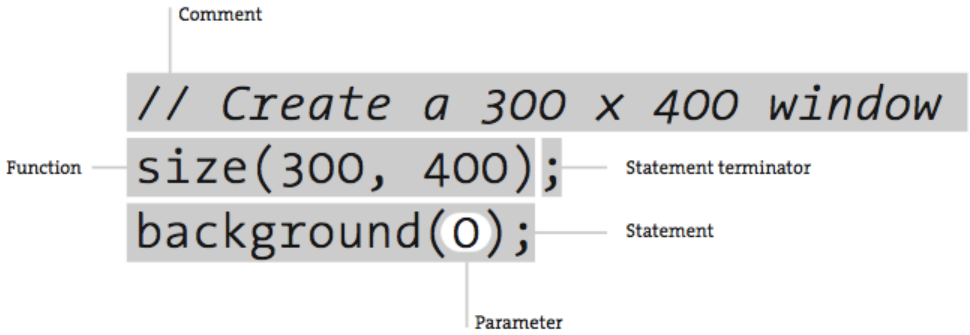


Figure 2: The main components of a program (Source: Raes & Fry)

Functions are references to segments of code that perform a specific task. This allows code to be encapsulated and easily referenced in other parts of the program without seeing the code it actually performs.

For example, the function **`line(0,0,9,9);`** draws a line between the coordinates (0,0) and (9,9) as parameters. In the libraries of the language, this

function would have the code that actually takes the stroke and stroke weight and draws the respective line. At the end of the day, functions make it easier to code since they allow us to focus on the bigger picture.

Every function has to be followed by a terminator that is simply represented by a semi-colon symbol **;** that tells the compiler that the function ends there and another might follow.

Comments

As the name implies, comments are text that the compiler (the engine that converts the code we write into machine code) ignores. Therefore, it is advised that programmers add comments when they are coding to record/document the rationale behind coding decisions taken in the programming approach. This makes the code more readable when other developers look at the code and contribute to it.

A secondary use of comments is 'hiding' of certain lines of code from the compiler. Consider the example below:

```
line(0,0,9,9);
//strokeWeight(10);
ellipse(10,10,40,40);
```

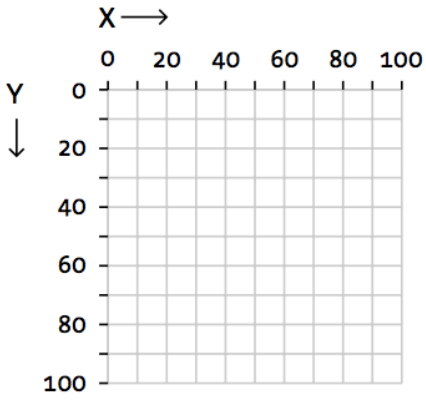
The code above would draw a line and an ellipse. There is a line of code in the middle that stroke weight to 10 but since it is commented by the **//** characters, it would be ignored by the compiler and would not be visible in the final output of the program.

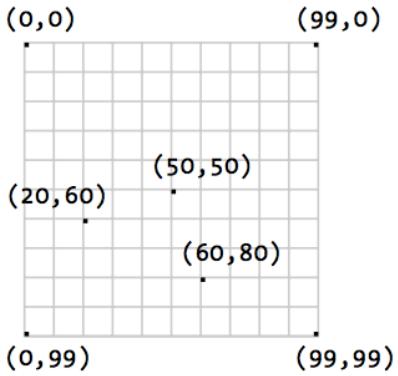
Drawing in Processing

Coordinates in Processing

Coordinates are pairs of numbers that represent positions on a grid. In Processing, the numbers growing horizontally from left to right are known as the

x-axis or x-values. On the other hand, the numbers growing vertically from top to bottom are known as the y-axis or y-values.





Basic Drawing Functions

- `point(x,y);`
  - Draws a point at coordinate (x,y).
- `line(x1, y1, x2, y2);`
  - Draws a line from coordinate (x1,y2) to (x2, y2).
- `ellipse(x1, y1, width, height);`
  - Draws an ellipse of width and height specified as parameters with origin (x1,y1).
  - NB: A circle is drawn by simply specifying equal width and height .
- triangle(x1, y1, x2, y2, x3, y3);
  - Draws a triangle between these 3 points.
- quad(x1, y1, x2, y2, x3, y3, x4,y4);
  - Draws a quadrilateral between these 4 points.
- rect(x1,y1, width, height);
  - draws a rectangle with top left corner at (x1,y1) and respective width and height.

Colours

In most examples we will be using the RGB colour space to assign pixel values. The Red, Green and Blue values will be given as parameters to functions that

would affect pixel values such as for example **`background(255,0,0);`** would set the background to red while **`stroke(0,0,255);`** would draw the forthcoming graphics in blue.

The pixel values are given in the range between 0 and 255 and therefore there are 256 shades for each colour element. One can therefore find the RGB value desired and set the parameters accordingly to get the desired effect such as for example **`stroke(100,22,255);`** would give a colour to outlines of subsequent shapes with the RGB values passed as parameters. On the other hand, the function **`fill(0,0,255)`** would fill the subsequent functions with the colour composed of the RGB components passed as parameters.

The reason behind the range between 0 and 255 is that the most common *pixel format* is the *byte image*, where this value is stored as an 8-bit integer (formal name for a whole number). A byte is equivalent to 8-bits where every bit is either 1 or 0. The different possible values of an 8 digit number where every digit is either 1 or 0 is equal to 2^8 that is equal to 256 therefore giving a range of possible values from 0 to 255.

The Processing functions that take a value of colour can also accept different parameters such as hex values and so on. On the other hand, if only 255 is passed as a parameter, the colour would be set to white while on the other hand, a value of 0 would return a black colour.

The function StrokeWeight(10) takes a number as a parameter and it specifies the weight/thickness of subsequents shapes drawn.
