

## 1 Getting started with Processing

Processing is a “sketch” programming tool designed for use by non-technical people (e.g., artists, designers, musicians). For technical people, it is a handy tool for prototyping applications in Java.

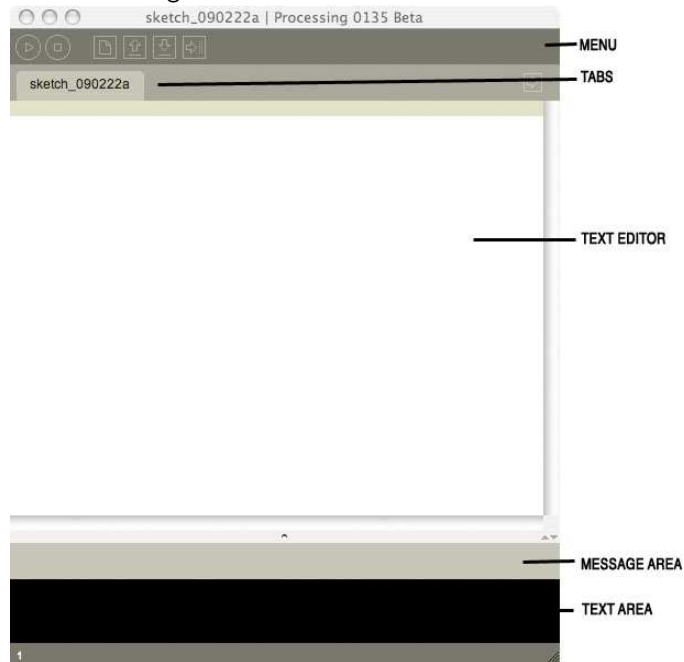
You can do lots of things with Processing. This lab will focus on drawing graphics.

### 1.1 Start up Processing

Double-click the **Processing** icon, which probably looks something like this:









The Processing window looks like this:



Note the annotations on the right in the image above that point out the areas of the window.

Menu buttons:

- |   |               |   |
|---|---------------|---|
|  | <b>run</b>    | compiles the code, opens a display window and runs the program.   |
|  | <b>stop</b>   | terminates a running program.   |
|  | <b>new</b>    | creates a new “sketch” in the current window.   |
|  | <b>open</b>   | provides a menu with options to open files from your “sketchbook”, an example or another a sketch on your computer. |
|  | <b>save</b>   | saves the current sketch with its current name and location.  |
|  | <b>export</b> | exports the current sketch as a <b>Java</b> “applet”.   |

Notes on Menu buttons:

- **run** — Hold down the **shift** key to **Present** instead of run
- **new** — To create a new sketch in its own (new) window, use **File - New**
- **open** — Note that opening a sketch from the toolbar will replace the sketch in the current window. To open a sketch in a new window, use **File - Open**.
- **save** — If you want to give the sketch a different name, use **File - Save As**.

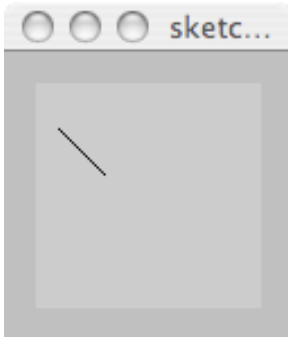
## 1.2 Write your first program: drawing a line

In the **text window**, type the following:

```
line( 10, 20, 30, 40 );
```

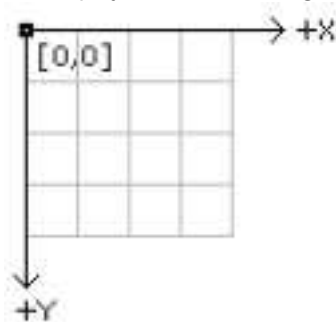
- Note the punctuation—parenthesis, commas and semi-colon.
- Note that the “function” **line** is written in *lower case*. Processing is case-sensitive, so watch the case with each new function introduced.

After you type the above in the text window, click on the **run** button. Processing will open a display window, like this:



What have you done?

The **line()** function takes four “arguments”. These are the endpoints of a line. Imagine that the display window is a piece of graph paper with two axes:  $x$  and  $y$ . The  $x$  axis runs horizontally along the display window, starting with 0 on the left and increasing as you move to the right. The  $y$  axis runs vertically down the display window, starting with 0 on the top and increasing as you move down.



So, you have just drawn a line from  $(x_1, y_1) = (10, 20)$  to  $(x_2, y_2) = (30, 40)$ .

## 1.3 Modify your program

- Try changing the values of the arguments to the **line()** function to different  $(x, y)$  values. Each time you change the values, click on the **run** button to see the effect of what you’ve done.
- Try adding a second **line()** function (put it on another line in the text editor, below the first **line()** function).

## 1.4 Adding color

- The Processing function for drawing in color is called **stroke()**. It takes one argument, a 6-digit *hexademical* number specifying the amount of *red*, *green* and *blue* in the color that should be used for drawing. For example:

```
stroke( #ff0000 );  
line( 10,10,20,20 );
```

*Does this remind you of setting colors in HTML/CSS?*

- Try adding a **stroke()** function call above your call(s) to **line()** in the text editor.
- Check out the command **Tools - Color Selector**. You'll find help for picking cool colors!
- Now try adding multiple **stroke()** calls to your sketch, one before each **line()** call, each one with a different color. This way, each line you draw will be a different color.
- Try drawing a green square using one call to **stroke()** and four calls to **line()**.  
*Hint: plan out your code ahead of time by drawing a square on paper and figuring out what the coordinates of each of the four corners should be.*
- Try drawing a square with each side a different color. You will need four calls to **stroke()** and four calls to **line()**.

## 1.5 Learning new functions

Look up the syntax of each of the following functions on the Processing on-line reference page:

<http://www.processing.org/reference/index.html>

Try putting each command in your sketch and see how they work.

- **rect()**
- **ellipse()**
- **fill()**
- **triangle()**
- **strokeWeight()**

## 1.6 Exporting an applet

Try clicking on the **export** button or selecting **File - Export**.

This will create a Java "applet" out of your sketch. A Java applet is embedded in an HTML file. With the export command, Processing saves the Java applet as well as an HTML called **index.html**. Both are placed in a subfolder called **applet** in the sketch's folder inside your sketchbook. Processing will open this subfolder. Click on **index.html** to view the Java applet version of your sketch.

## 2 Making your program interactive

In the first section (above), you learned how to write programs that only had **output**. This means the programs display stuff. In this section, you will learn how to write programs that take **input**, which means that you can have the program respond to things the user/viewer does.

As above, remember to be mindful of upper and lower case, as well as punctuation.

2.1 Create a new sketch and enter the following code in your text window.

```
void setup() {  
    background( #ff0000 );  
}  
  
void keyPressed() {  
    background( #0000ff );  
}  
  
void draw() {  
}
```

Click on the **run** button, and then when Processing opens the display window, click anywhere in the display window and then click on any key. The color of the display window should change from red to blue.

The code in the **setup()** function runs as soon as the sketch starts.

The code in the **keyPressed()** function runs as soon as the user presses a key. Note that you (the user) have to click in the display window to give it *focus*, so that the sketch will recognize (i.e., be “listening”) when a key is pressed.

2.2 The above program demonstrates input from the keyboard, when *any* key is pressed. Now try entering and running the code below, which responds differently when different keys are pressed.

*Note:* The program will respond to R, G, B, and W.

Don't forget to click in the display window, to give it focus, before pressing any keys.

```
void setup() {  
    background( #000000 );  
}  
  
void keyPressed() {  
    if ( key == 'R' ) {  
        background( #ff0000 );  
    }  
    else if ( key == 'G' ) {  
        background( #00ff00 );  
    }  
    else if ( key == 'B' ) {  
        background( #0000ff );  
    }  
    else if ( key == 'W' ) {  
        background( #ffffff );  
    }  
}  
  
void draw() {  
}
```

After you have run the program, go back and look at the code. You will see the words **if** and **else**. These are called *control structures*, and they control the flow of the code. If the user presses the R key, one thing

happens (what is it?); otherwise, if the user presses the G key, something else happens (what is it?); and so on. The **if...else** is called a *conditional* control structure, because it specifies what the program should do under conditions specified by the programmer.

- Try changing the code by modifying what happens when the user presses R. Note that whatever code you add/change, all functions have to be contained within curly brackets ({ and }). Currently, only the statement `background( #ff0000 );` is in between the curly brackets. If you add more lines of code, keep them between the same curly brackets.
- Try modifying the code so that it responds to both uppercase and lowercase input (e.g., R and r).
- Now try changing the code by adding another condition of your own, when another key is pressed (other than R, G, B, or W).

### 3 Programming challenge

Combine what you have learned so far to make a program that draws different shapes depending on what letters the user types.

For example, your program could draw a line if the user types L and a circle if the user types C.

Or you could draw more complicated graphics, like drawing a house if the user types H or a bridge if the user types B.

Have fun!

### 4 On-line references

Processing

- Processing web site: <http://www.processing.org/>
- getting started tutorial: <http://www.processing.org/learning/gettingstarted/>
- drawing tutorial: <http://www.processing.org/learning/drawing/>
- color tutorial: <http://www.processing.org/learning/color>
- reference: <http://www.processing.org/reference/index.html>

Processing for mobile devices:

- <http://mobile.processing.org/>