

Processing | LESSON 2

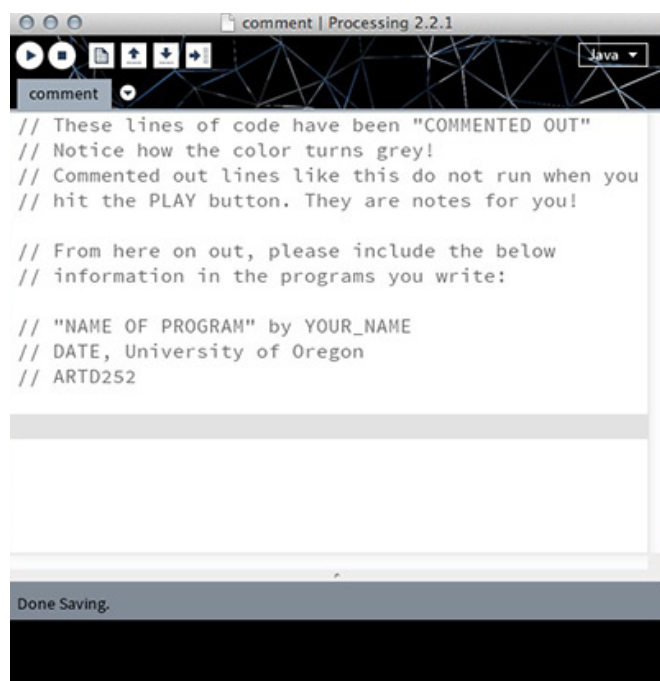
More advanced options with the processing language...

STEP 1: Commenting

Writing Code is a personal process and everyone does it differently. Because of this, sometimes it is hard to decipher what other people are doing in their code-writing process, or sometimes you forget what YOU YOURSELF were thinking when you wrote some code. How do we get around this?

ENTER...COMMENTING

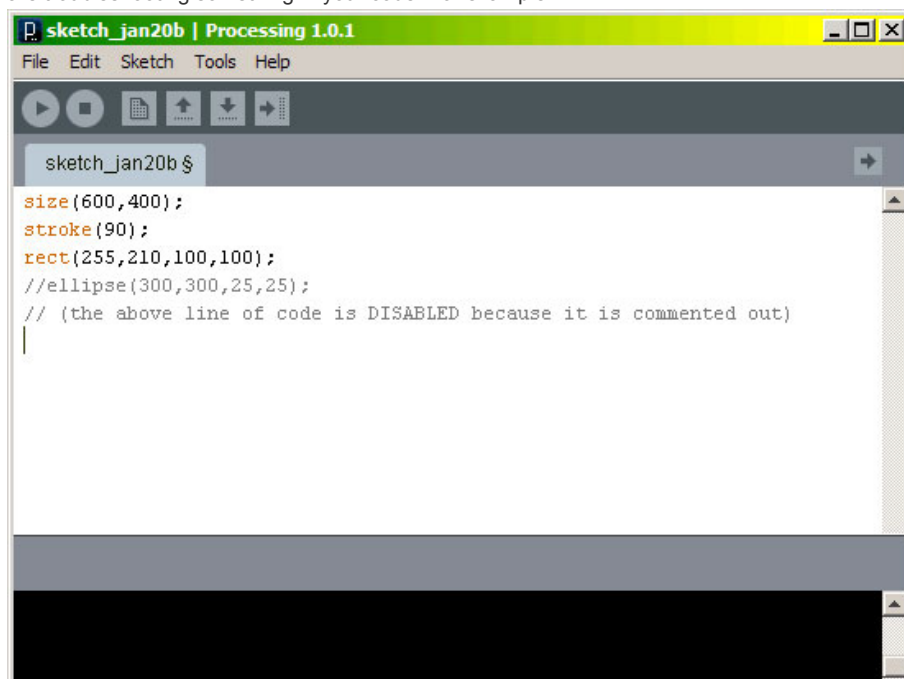
Commenting is a way to leave notes for yourself or for others when you are writing a program. Commented lines of code do not run in the actual program once it compiles...so leaving comments for yourself doesn't slow things down at all or ruin any of your work. Think of commenting as a handy way to leave as many notes to yourself or others who might someday see your source code. To comment a single line of text you add two `//` symbols before the line. For example:



If you want to comment out large sections of text. There is a common programming technique of starting a comment block, adding all your text, then ending a comment block. It works by beginning a block with `/*` and ending with `*/`:



A VERY common practice is to comment out lines of code if they are causing poor results or if you are troubleshooting something in your code. For example:



STEP 2: Data Types and Declaring Variables

When we declare variable we want to tell Processing what type of information those variables will store. Think how a program would need treat a name such as "Sarah" (called a STRING datatype in processing) differently from numeric information such as a temperature reading like 98.6 (called a FLOAT data type in processing).

Common **data types**:

int - whole numbers, positive or negative. [See More on reference page.](#)
int age = 26;

float - decimal, positive or negative. [See More on reference page.](#)
float sensorValue = 2.751;

String - text as opposed to numbers. [See More on reference page.](#)
String currentName= "Grizelda";

boolean - true or false (1 = true, 0 = false). [See More on reference page.](#)
boolean mode1 = false;

char - a single character such as a letter or a symbol. [See More on reference page.](#)
char letter = 'A';

When you declare a variable, first write the **DATA TYPE**, then the **NAME OF THE VARIABLE** and then you can either leave the value unspecified (and define it later) or define it there on the spot by adding an equal signs and assigning it a value.

For example. If you know you only need whole numbers for a value, an integer (or int) is most appropriate:

int age;
OR
int age = 30;

If you know that you will need decimal based or fractional numbers, it is more appropriate to use a floating point number of float.

float weight = 116.52;

If you know you will be displaying text, then a String is most appropriate.

String label1 = "Great Visuals Require Work";

STEP 3: Putting it to use

Now we will look at a little Sketch that uses a float value to change the rotation of an ellipse with every frame. The code looks like this:

```
//*****

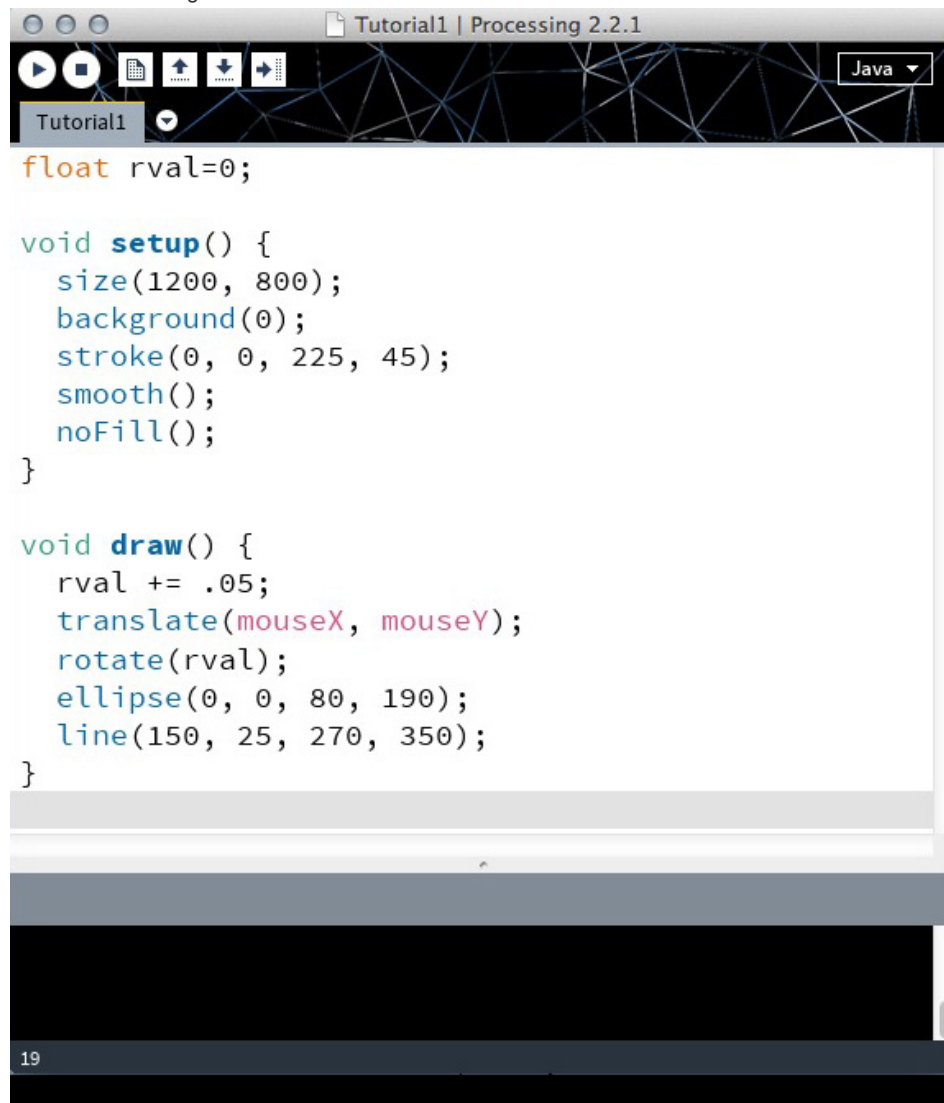
float rval=0;

void setup(){
  size(1200,800);
  background(0);
  stroke(0,0,225,45);
  smooth();
  noFill();
}
```

```
void draw(){
  rval += .05;
  translate(mouseX,mouseY);
  rotate(rval);
  ellipse(0,0,80,190);
  line(150, 25, 270, 350);
}
```

```
//*****
```

or in the Processing window:



Try running the program to see how it looks and then start to see if you can figure out what it happening on your own. If you encounter a command that you don't recognize (the blue text lines means the word is a built-in command of the language). By RIGHT CLICKING on the selected text and by choosing 'Find In Reference' you can learn a surprising amount about this language.

New code to learn about:

translate();

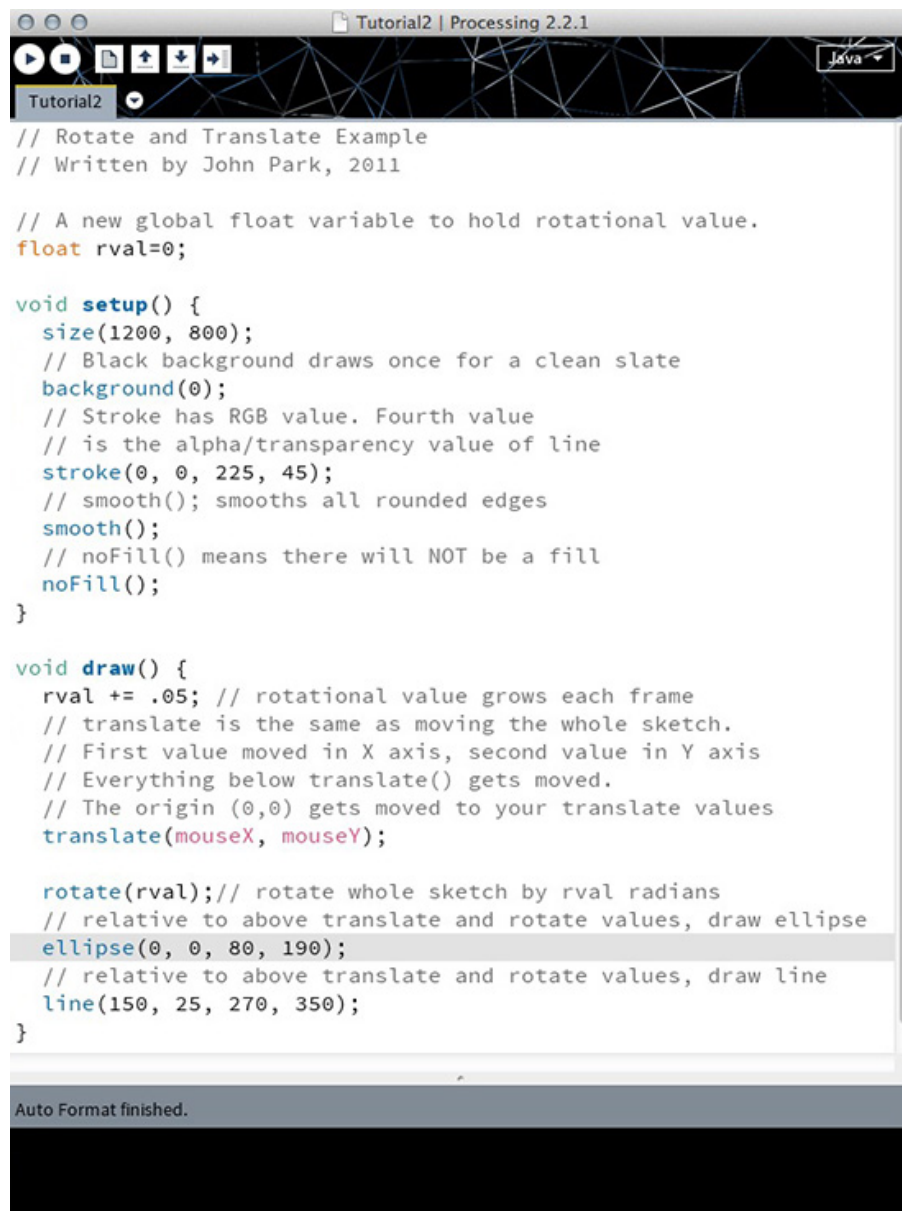
translate is the same as "move" or "slide". By default processing is looking for two values...the first is the X-axis movement, or a LEFT/RIGHT slide. The second value is a Y-axis movement, or UP/DOWN slide. The translate command moves the entire sketch canvas and all of its content that follow this command by the designated amount. All code above a translate() are NOT AFFECTED. All code after the translate() are affected. Values are in pixels.

rotate();

rotate(); rotates the entire sketch canvas and all of its content that follows the rotate() call. There are two important catches. Processing always rotates content around the canvas origin, or 0,0 point. This is usually at the top left of sketch. The second important note is that processing rotates in terms of radians, not degrees. There are 2π radians in a circle (6.2831 radians in 360 degrees). If you want to automatically convert degrees to radians, use the **radians()** command. For example **rotate(radians(45));** will convert 45 degrees to radians, then rotate by that amount.

By combining the translate() with the rotate() in the below sketch, we are moving the origin point (originally 0,0) to where the mouse is located. Then we are rotating around that point.

Below is an explanation via commented code.



```
// Rotate and Translate Example
// Written by John Park, 2011

// A new global float variable to hold rotational value.
float rval=0;

void setup() {
  size(1200, 800);
  // Black background draws once for a clean slate
  background(0);
  // Stroke has RGB value. Fourth value
  // is the alpha/transparency value of line
  stroke(0, 0, 225, 45);
  // smooth(); smooths all rounded edges
  smooth();
  // noFill() means there will NOT be a fill
  noFill();
}

void draw() {
  rval += .05; // rotational value grows each frame
  // translate is the same as moving the whole sketch.
  // First value moved in X axis, second value in Y axis
  // Everything below translate() gets moved.
  // The origin (0,0) gets moved to your translate values
  translate(mouseX, mouseY);

  rotate(rval); // rotate whole sketch by rval radians
  // relative to above translate and rotate values, draw ellipse
  ellipse(0, 0, 80, 190);
  // relative to above translate and rotate values, draw line
  line(150, 25, 270, 350);
}

Auto Format finished.
```

Try taking these skills and adding other shapes inside the draw function.

Other built-in shapes can be found in the [REFERENCE](#) page for Processing under '2D Primitives'