Processing: Computational art

While programs like Photoshop and Illustrator allow you to create and modify image using a powerful set of tools, there are times when you may need to perform tasks that existing software programs cannot easily do. In these cases, it may be necessary to create your own software program.

Although you can use a general purpose computer language like C++ or JavaScript, there are several languages specifically designed for artists and designers. One of the most commonly used ones is *Processing*.

Processing programs are called *sketches*. You type the program into the text editing window and then run it with the "Run" button in the Processing toolbar. If there is an error in your code, an error message will appear in the message area below the text editing window. The processing web site (http://www.processing.org) and the help files (in Processing help menu) are good starting points for learning the Processing language. There are also a series of examples listed in Processing's "File" menu.

Your first Processing program

The best way to begin using Processing is to display a simple shape on your computer screen. Type the following example code into the Processing text editor, and the click the Run button.

```
void setup()
{
  size(400, 300);
  background(200, 200, 250);
 }

void draw() {
  noStroke();
  fill(255, 0, 0);
  rect(40, 20, 100, 150);
}
```

Here's what each line does

`void setup()`
Here is where we set up any parts of the sketch that will not change, such as the window size.

`size(400, 300);`
creates a new window on screen, 400 pixels wide and 300 pixels tall

`background(200, 200, 250);`
sets the background to light blue. A single number will be interpreted as a greyscale value, from 0 (black) to 255 (white). Three values (200, 200, 250); will be interpreted as an RGB color

`void draw()`
Instructions in the "draw" loop will run until the program finishes

`noStroke();`
the shape we are drawing will have no stroke

`fill(255, 0, 0);`
the shape we are drawing will have a red fill. A set of 3 numbers separated by commas will be interpreted as an RGB value.

`rect(40, 20, 100, 150);`
draw a rectangle. The numbers stand for X position, Y position, width and height, in pixels. This example will create a rectangle with its top left corner 40 pixels over and 20 pixels down from the top left of the drawing. It will be 100 pixels wide and 150 pixels tall.

There are several other shape drawing commands. The line command, for example, looks like this:

```
line(30, 20, 85, 75);
```

The first 2 numbers are the x and y coordinates of the start of the line, the last two are the x and y coordinates of the end of the line. Check the Reference section of the Help menu for other drawing commands — they are listed under *Shape*.

Using colour

Processing (like other systems design for screen, rather than print) uses RGB values to represent colour. You can use the simple colour picker under Tools->Color Selector or the RGB values from a program such as Photoshop.

Using variables

While in our first example we put the values directly into the statements, for more sophisticated programs you will need to put those values into *variables*. A variable is simply a container that holds a certain type of information.

In Processing, you need to *declare* the type of information you plan on storing in each variable. The only type of variable we will use at this point is the *integer*, which Processing calls an *int*. An int can store any whole number (that is, a number with no decimal points) from 2,147,483,647 to -2,147,483,647.

Instead of writing the line:

```
background(230, 240, 20);
```

If we use a variable, we would write it like this:

```
int myRed = 230;
int myBlue = 240;
int myGreen = 20;
background(myRed, myBlue, myGreen);
```

This creates three new variables, called `myRed, myBlue, myGreen`. Each is an int, meaning that it can store a whole number.

You can use any name you like for your variables, provided they don't contain spaces, punctuation, or begin with a number. In the example above, we then set the value of myRed to be 230. Now, where we had used hard-coded values in the background() statement, we can use the variable instead.

Using variables will be essential when we want to do more sophisticated tasks.

Creating a loop

A loop is a series of statements that will be executed a specified number of times.

For example, we may want to draw several objects on the screen. It would be tedious to have to type in the size, colour and location of each one.

Here is an example of a loop, it will draw a series of lines.

```
for (int theCounter=40; theCounter<80; theCounter=theCounter+5) {
  stroke(0);
  line(30, theCounter, 80, theCounter);
}
```

The for statement is used to create a loop. Within the parentheses, there are three parameters. The first, creates a new variable (in this case called *theCounter*) and assigns a value to it. This will be used as a counter. The second checks if a condition is true. If the condition is true, in this case the variable theCounter is less then 80, is the loop continues. If the condition is false, the loop ends. The final parameter updates the counter, in this case adding 5 to it.

After the { are the statements that will be executed in the loop. You often use the loop counter variable as part of these statements.

Finally, the loop is closed with the }

Random numbers
By introducing random values, you can create sketches with random elements. Create random numbers with the *random* statement.

```
random(low, high);
```

The result of this statement will be a random number between the “low” and “high” number. For example, random (10,20); will generate a random number between 10 and 20.

The random function will create a floating point value (a value with a decimal point) and not an int. To convert the value to an int, use int(random(low, high));

To generate a random number between 1 and 10, use int(random(1, 10));

The following code uses the line statement and the random statement to draw random lines on the screen:

```
void setup() {
int myColour = 100;
size(300, 200);
background(myColour);
}


void draw() {
noLoop(); //this keeps the sketch from drawing forever
          //- remove it and see what happens
for (int theCounter=0; theCounter<10; theCounter=theCounter+1) {
  //generate random values for X and Y of start and end of line
  int lineStartX= int(random(1, 300));
  int lineStartY= int(random(1, 200));
  int lineEndX= int(random(1, 300));
  int lineEndY= int(random(1, 200));
  stroke(0);
  line(lineStartX, lineStartY, lineEndX, lineEndY);
  }
}
```

Note that every time you run this code, you will get a different result.

Next steps:
The program below creates a line that extends from the initial point (150,25) to the location of your mouse.

```
void setup() {
       size(400, 400);
       stroke(255);
     }

     void draw() {
       background(192, 64, 0);
       line(200, 200, mouseX, mouseY);
     }
```

The setup section runs once when the sketch is opened, and contains setup information such as the size of the sketch, colours, etc. The draw section runs as a loop, repeating over and over again. By putting the statement: background(192, 64, 0); in the draw block it fills the screen with the background colour, effectively erasing the screen each time.

Resources:
www.processing.org
www.arduino.cc