Controlled Repetition – Loops

The while and for- loops provide a more versatile and controlled way of repeating statements than the built-in, 60 times/second, loop in the draw() function in Processing. In fact, for this lesson, we will suppress the implicit draw() loop by using the noLoop() command in the setup() function of our sketches.

The while-loop

The syntax of a while-loop is given below:

while ( <condition> ) {
    <statements>
}

When execution encounters the while-loop, the <condition> is evaluated. As we discussed in the Conditions lecture notes, the <condition> is a Boolean expression that yields a true/false value. If the <condition> is true then statements are carried out once. After that, the <condition> is evaluated again and the process is repeated.

Here is a simple sketch that shows how to use a while-loop:

void setup() {
    size(500, 500);
    background(255);
    noLoop();
} // setup()

void draw() {
    int i = 1, N = 5;
    while (i <= N) {
        println(i);
        i = i + 1;
    }
} // draw()

The sketch above prints out the values of i each time around in the loop (see image above). Below, we have reproduced the loop and numbered the lines so we can refer to them:

1  int i = 1, N = 5;
2  while (i <= N) {
3      println(i);
4      i = i + 1;
5  }

In line 1 we define two variables i (=1) and N (=5). At line 2, the condition (i <= N) will be true since 1 is less than 5. Then the statements on lines 3 and 4 will be carried out. Consequently, the value of i (at line 3) is printed on the Console and following that, at line 4, it will be incremented to 2. Since we have now reached the end of the loop (line 5), we go back to test the loop condition again (at line 2). Now,
the value of \(i\) is 2, which is 2 is less than 5, so we will repeat lines 3 and 4, printing a 2 and then incrementing the value of \(i\) to 3. This will go on until \(i\) becomes 5 and 5 is printed and then \(i\) is incremented (at line 4). Then, when the condition is tested, it will be false because the value of \(i\), which is now 6) is no longer less than or equal to 5 (the value of \(N\)). At this point, we say that the loop terminates and the execution exits the loop.

Exercise: Modify the sketch above to print out 5 random values in the range [0..1.0).

The for-loop

Another way to express a repetition is to use a for-loop:

```
for ( <initialization>; <condition>; <update> ) {
    <statements>
}
```

When execution reaches the for-loop for the first time, the <initialization> specified is carried out. After that, the <condition> is evaluated. If the <condition> is true, the <statements> are all carried out. Upon reaching the end of the loop, the <update> is executed. Then, the <condition> is tested and the process repeats. Until the condition is false. You can see that in the example below:

```
for (int i = 1; i <= N; i = i + 1) {
    println(i);
}
```

The for-loop defines and initializes \(i\) (to 1). Next, the condition \((i <= N)\) is tested. Since \(i\) is 1, it will be true and the value of \(i\) will be printed. Next, \(i\) will be incremented (specified as <update>), condition tested again, etc. the loop is repeated until \(i\) becomes greater than \(N\) (=5). The for-loop shown above is exactly equivalent to the while-loop in the sketch above.

As you can see, you can interchangeably use while- and for- loops to accomplish a repetitive task. While people have their personal preferences, we are more inclined to use for-loops because they make all the parts of the loop clearly visible and explicit.

Example: The program segment below simulates the tossing of a coin five times and prints out each outcome (see image):

```
int N = 5;
for (int i = 1; i <= N; i = i + 1) {
    if (round(random(1)) == 0) {
        println("Heads");
    }
    else {
        println("Tails");
    }
}
```
Drawing with loops

Now that we have a basic understanding of loops, let us use them to create drawings.

**Sketch: Drawing Circles**

We want to draw several circles in the Drawing Window, going from left to right. The sketch below accomplishes this task:

```java
void setup() {
  size(250, 200);
  background(183, 117, 203);
  noLoop();
} // setup()

void draw() {
  int x = 0, y = height/2, w = 20;
  noStroke();
  fill(118, 203, 117);
  while (x < width) {
    circle(x, y, w);
    x = x + w + 5;
  }
} // draw()
```

**Exercise:** Rewrite the sketch above to use a for-loop.

**Exercise:** Modify the sketch above so it draws circles of increasing width.

**Sketch: Drawing concentric Circles**

```java
void draw() {
  int x = width/2, y = height/2;
  int dist = 10;
  noFill();
  for (int w = 0; w < width; w = w + dist) {
    circle(x, y, w);
  }
} // draw()
```

This time we used a for-loop to do the task.

**Exercise:** Rewrite the for-loop above to use a while-loop.

The variable `dist` is the distance between two circles (it is set to 10 pixels in the above sketch). Change the value of `dist` to 2, or 3 and observe the resulting drawing. You will notice some emergent patterns that are not drawn but are artifacts of the drawing screen. These are called *Moire patterns*. 


**Sketch:** Drawing radial lines

The sketch below, uses translate(), rotate(), and a loop to draw the radial lines shown.

```java
void setup() {
  size(250, 250);
  background(255);
  noLoop();
} // setup()

void draw() {
  int N = 20;
  float angle = 2*PI/N;

  translate(width/2, height/2);
  for (int i=1; i <= N; i = i + 1) {
    line(0, 0, width/2-10, 0);
    rotate(angle);
  }
} // draw()
```

**Exercise:** Study the sketch carefully to make sure you understand every feature that is being used. Try modifying the values of N to see different numbers of radial lines.

**Project: Drawing a Four-Leaf Clover**

We want to draw a four-leaf clover flower as shown on the right.

If you look carefully at the flower, each clover leaf is the same clover leaf shape rotated. Thus, we can break down this sketch into first drawing a single clover leaf and then drawing the entire flower. Let’s us say the command to draw a flower is:

```java
flower(x, y, sz);
```

This command will draw a flower of size, *sz* centered at (*x*, *y*) coordinates. Next, assume we have a command:

```java
clover(x, y, w, h);
```

The **clover()** command will draw a clover leaf with its bottom-left corner anchored at (*x*, *y*) of width *w*, and height, *h*. Thus, we can define the flower function as shown below:

```java
void flower(float x, float y, float sz) {
  int N = 4;
  float angle = 2 * PI/N;
  push();
  translate(x, y);
  ```
for (int i = 1; i <= N; i = i + 1)
    clover(0, 0, sz, sz);
    rotate(angle);
}
pop();
} // flower()

N is 4 because we are drawing a 4-leaf clover. Prior to the loop, we translate to the 
(x, y) points (the center point of the flower), and then draw four clovers, each 
rotated by 90 degrees (angle = 2*PI/N).

Now, our only remaining task is to define the clover() function. To plan the 
drawing of a clover, we decide that a clover leaf will be a closed shape based on 
the curve defined using the anchor points shown in the image.

The bottom-left corner is the anchor point and is the start and end point of the curve. The rest of the 
curve is traced by the two points shown on the top and right edges of the enclosing rectangle (of width, 
w and height, h. With this plan in mind, we present below the clover() function:

void clover(float x, float y, float w, float h) {
    push();
    translate(x, y);
    beginShape();
    curveVertex(0, 0);  // bottom-left anchor point
    curveVertex(0, 0);
    curveVertex(0.25*w, -h);  // point on top edge
    curveVertex(w, -0.25*h);  // point on right edge
    curveVertex(0, 0);
    curveVertex(0, 0);
    endShape();
    pop();
} // clover()

**Exercise:** Complete the sketch to draw the four-leaf clover flower in the center of the screen. The 
Shamrock green color has the RGB values (22, 155, 98).

**Exercise:** Play around with the shape of the clover. Move the points on the top and bottom edges of the 
rectangle to change the shape of the clover into more like a petal. Draw a flower with several petals. i.e., 
make N a parameter of the flower function. Add an alpha value to the fill color and try other colors. 
Make the sketch interactive so it draws a flower wherever the mouse is clicked. Make it so it has a 
random color and size.