GRAPHICS PROGRAMMING, LAB 2

MORE PROCESSING

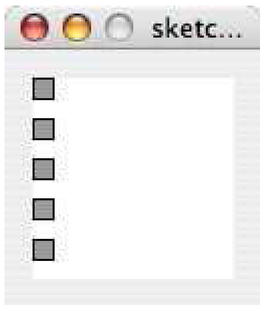
NAME: ____

Introduction:
Up until now, the Processing sketches you have created have been static, i.e., they do not change by themselves. You have used keyboard input to let the user control changes in the display. Today, you'll learn:
(1) how the user can employ the mouse to control changes in the display, and
(2) how to make things change in the display by themselves—through animation.

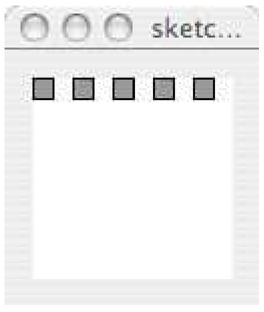
We'll start with a few exercises that use "iteration", a structural programming element that was discussed in lecture 2.2.

1. Iteration in Processing

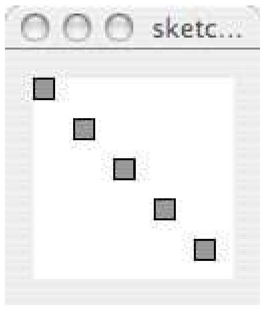
1.1. Use a for loop to create each of the following three sketches. Each sketch contains 5 filled rectangles of size 10 × 10 pixels.



(a)



(b)



(c)

Hint 1: Add the for loop(s) inside the setup() function.

```
void setup() {



}
void draw() {
}
```

Hint 2: Create two variables called xPos and yPos, at the very top of the program, and use those variables in the rect() function. Each time through the loop, make xPos and/or yPos bigger.

1.2. Use a while loop to create three sketches like those shown above (in part 1.1), but in the setup() function, call the Processing function size() to increase the size of the display window to 654 × 321 pixels. Like this:

```
void setup() {
        size( 654, 321 );



}
void draw() {
}
```

Hint1: the Processing variables width and height store the width and height of the display window, respectively.

Hint2: (xPos < width ***&&*** *Ypos < height)*

1.3. Challenge exercises (in case you haven't been challenged yet):

1) Create another sketch similar to sketch (c) above (1.1), but instead of having the trail of squares go diagonally from upper left to lower right, make them go from lower left to upper right.

2) Fill the display window with a checkerboard pattern.
Hint: Use two loops, one inside the other.

3) Modify the checkerboard so that squares are drawn in alternating colors, e.g., red then blue, then red, then blue, etc.

2. Mouse control with Processing

Hint: Refer to the example sketches from Monday's class.

2.1. Draw a circle of radius 10 pixels in the middle of the display window and let the user reposition it with the mouse, as follows: when the user **moves** the mouse, redraw the circle so that its center is at mouseX and mouseY.
Hint: Use the mouseMoved() function.

2.2. Modify your previous sketch so that the circle is only repositioned when the user **clicks** with the mouse.

2.3. Modify your first sketch so that the circle is only reposition when the user **drags** the mouse, i.e., moves the mouse with the button pressed.

2.4. Modify any of the three sketches above so that the mouse position determines the upper left corner of the circle rather than the center of the circle.
Hint: look at the Processing function called ellipseMode().

3. Animation with Processing

The basic principle behind animation is like that of an old-fashioned flip book. If you don't know what a flip book is, you can see a sample here:
http://www.flippies.com/flipbooks-gallery/

In Processing, the idea is that your program will draw an object in the display window, wait a fraction of a second or so, and then clear the display and draw the object again, in a slightly different place. This will make it look like the object is moving across the display.

As discussed in lecture 2.2 the draw() function is a special function in Processing. The draw() function is basically an endless loop. Whenever draw() reaches the end of its statements, it starts back over again at the top.

3.1. Enter this sample code below into Processing. Run it and see what happens.

```
// This is a comment. Below are two variables, x and y.
int x = 0;
int y = 50;

// Below is the setup function. It is called only once.
void setup() {
        background( #000000 );
}

// Below is the draw() function. It is called repeatedly.
void draw() {
        background( #000000 );
        ellipse( x, y, 40, 40 );
        x = x + 1;
        if ( x > width ) {
                x = 0;
        }
}
```

3.2. Modify the code to make the ellipse move vertically instead of horizontally.
3.3. Modify the code again to make the ellipse start with a small width and height (e.g., instead of 40, 40, start with 1,1) and grow larger each time draw() is called. Figure out what you want to do when the ellipse has grown to be too big to fit inside the display window. For example, the ellipse could gradually shrink back to the smallest size and

then grow again. Or it could snap back to the smallest size and grow again. Or it could turn into a small rectangle and grow. Or it could explode...

3.4. Challenge exercise (if you haven't been challenged yet): start the ellipse in the upper left corner of the display and animate it so that it traces the edge of the display window. First it moves horizontally across the top of the window until it reaches the upper right corner. Then it moves vertically down the window until it reaches the lower right corner, and so forth. When it reaches the beginning, it should start all over again.