



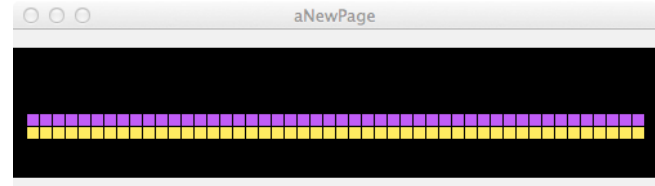
## Lab Exercise 8: Recursion

### Goal

The objective is to understand how recursion works. The programming in this exercise is easy. The goal is to imagine how the computer follows the instructions.

### Warm Up

In lecture we wrote a small recursive function `husky()` to draw purple and gold boxes using the “draw one, and if space remains, do it again” technique. Use that same idea to draw this sequence of 10x10 box pairs, leaving 10 pixels at each side of the window. Check the lecture if you have questions.



### Overview

You are given a recursive program that produces the following (Figure 1)



and you will improve it so that it produces the following (Figure 2).



Then, you will explain in your own words what’s happening, so look carefully at the code.

**Step 1: Start.** Grab the Recursive Code posted with this lab page. Also, open a word processing document.

**Step 2: Color the boxes.** The first step is to modify the `paint()` function so that it colors each level differently. You choose the colors, but they must all be different, and consistent by level – no random colors this time! The `paint()` function, which takes one argument, will be just a series of if-statements, like we’ve seen in the `keyPressed()` function. For example, it uses

```
if (level == 0) {
    return color(0);
}
```

and a separate if-statement for each level (up to, say, 7). Remember, copy/paste are your friends for programming similar sequences of code. *Check that each level has one color.*

**Change A.** Next, **replace** the `//Change A goes here` line in the `box( )` function with the following code

```
else {  
    fill(0,200,0);  
    ellipse(xdist, 240, 10, 20); // draw leaf  
}
```

This will draw green leaves. *Check your code and notice where they are drawn.*

In your document write a sentence or two to finish “The leaf drawn in Change A indicates visually the point in the recursive code where ...” The sentences are to say what’s happening in the code.

**Change B.** Next, “comment out” the first recursive call to `box( )`, and run the program. What you see is a diagonal row of colored boxes. This change neutralizes the tree-drawing ability of the recursive function; notice how the original `box( )` works: For a tree of  $x$  levels, it draws a tree of  $x-1$  levels, draws the root of the tree, and then draws another  $x-1$  level tree. The two shorter trees are created recursively by the same logic. When you removed the recursive call, what you were left with was code that just draws the roots of those trees, but not the trees themselves.

**Analysis.** Using your knowledge of recursion, and your observations/information from the two changes above, answer these questions: [Give answers in your document.]

1. The Processing program that drew the gray diagram (Figure 1) calls the `box( )` function how many times in total? \_\_\_\_\_
2. What does the basis case do in the revised (Change A) program?
3. The box pointed at by the arrow is what color in the colored picture? \_\_\_\_\_
4. If the box pointed at by the arrow was drawn as a result of the call, `box(level)`, what was the value of `level` in the call? \_\_\_\_\_
5. Referring to the function that made the call of question 3, what color box did *it* draw (as shown in the colored picture)? \_\_\_\_\_
6. Referring to the function that made the call of question 3, which of the two `box( )` calls was it? \_\_\_\_\_
7. As explained in the lecture, functions are often suspended in recursive programs while they perform recursive calls. At the moment that the `rect( )` function was called to draw the rectangle referred to in question 2, what `box( )` functions (if any) were suspended besides `box(6)`? \_\_\_\_\_

**Wrap Up:** You have taken a simple recursive program, and by adding lines and colors, you have tagged so that you can follow its behavior. This allows you to see how the recursive function works.

**Turn In:** Paste your `.pde` code for `husky( )` and `paint( )` into your document, save it and submit it to the class dropbox.