



Homework 6: Homage to a Square

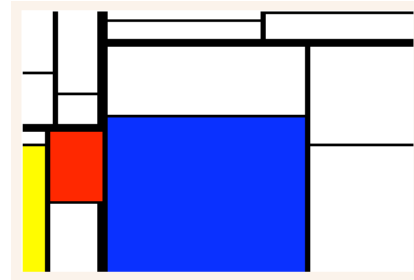
Goal: The purpose of this is to practice with Processing, and to lay the foundation for our future discussion of Artificial Intelligence, an important branch of computer science. You will gain some experience with Processing and learn three new CS concepts: randomness, variables and data types.

Art In A Click

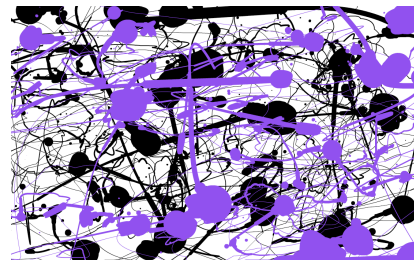
Computer art, that is, art generated by computers, not art created by people using computers, leads to some fun Web sites. We will try two of these before getting started on our own computer art project.

Instructions: Click around on the Web to familiarize yourself with various artists' work. (You will want to save a screen shot of your work in Steps 1 & 2 to be turned in later.)

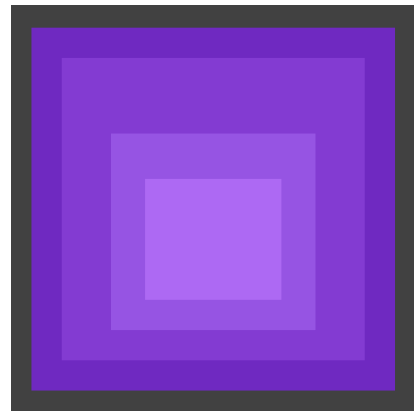
Step 1: Perform a Google search on "piet mondrian" and then click the Images link at the top of the returned hits list. Scroll through the images to get an idea of his work. He was a cubist and created pictures that look like the one at right. This, however, was a computer generated "Mondrian" ... try creating some yourself at <http://www.netlabs.net/hp/richieb/java/Mondrian.htm>. Take a screenshot of one of your creations



Step 2: Perform a Google search on "jackson pollock" and then click on the Images link at the top of the returned hits list. Scroll through the images to get an idea of his work. He was an abstract expressionist and created paintings that look like the one at right. This, however, was a computer generated "Pollock" ... try creating some yourself at <http://www.jacksonpollock.org>. Take a screenshot of one of your creations



Step 3: Perform a Google search on "josef albers" and then click on the Images link at the top of the returned hits list. Scroll through the images to get an idea of his work. He was an abstract painter, and created works that look like the one at right. This, however, was a computer generated "Albers" ... try creating some yourself at ... Oops! First you'll have to write the *Albers In A Click* program!



Albers In A Click

The program skeleton is shown at right.

Here's how it works: It starts by defining some variables (explained below). In the `setup()` it initializes the canvas, and then it picks two random colors – these are the outer color of the Albers image and the inner color. In the `draw()` component, the program picks two “in between” colors using a math function called interpolation (`lerpColor`). We don't worry about how that works. Then it draws the rectangles, outer to inner. Finally, `mousePressed()` picks two more random colors, which will be redrawn.

There are five places in the skeleton where you will have to make modifications. The first defines some `float` variables. The other places need a random color, but to do that you will need other lines of code. (Note that the code you add for the random coloring will be pretty much the same in each of the four places.) If you need help figuring out what to add check the resources for additional information.

To Turn In

Put your name on your two screen shots and on your `.pde` file, and submit to the class drop box.

```
hommageInAClick.pde
float //Need declarations
color outer, inner, mid_outer, mid_inner;

void setup( ) {
  size(600,600);
  background(51);
  noStroke();

  outer = //Need a random outer color
  inner = //Need a random inner color
}

void draw( ) {
  mid_outer = lerpColor(outer, inner, .33);
  mid_inner = lerpColor(outer, inner, .66);
  fill(outer);
  rect(60, 60, 480, 480);
  fill(mid_outer);
  rect(100, 100, 400, 400);
  fill(mid_inner);
  rect(165, 200, 270, 260);
  fill(inner);
  rect(210, 260, 180, 160);
}

void mousePressed( ) {
  outer = //Need a new random outer color
  inner = //Need a new random inner color
}
```

Homework 6 Resource: Randomness

Random numbers should be called *random number sequences*, because the definition requires that no matter how many numbers you already know in the sequence, it's not possible to predict the next one. A non-random sequence is 2, 4, 6, 8, 10, ... Computers cannot produce random numbers (because computers are completely predictable), but they can produce a sequence of numbers that passes all of the tests for randomness. These are called *pseudo-random numbers*, but everyone drops the "pseudo" part.

To generate a random number in Processing we write `random(<smallest possible number>, <largest possible number>)`. What we get back is some number – we can't predict which – between the two limits, including the end points. So, to generate a random number between 0 and 255, write `random(0, 255)`. To generate a number between 0 and 1, write `random(0, 1)`.

Homework 6 Resource: Variables and Their Data Types

Variables are names that can have values. For example, file names like `term_paper.doc` are variables and the name's value is the file; in this case, a Word document. The point about variables – pay close attention – is that they *vary*. For example, every time you edit the `term_paper.doc` file and save it, you've changed the value of the variable. This is pretty natural. The other thing about variables is that they name a certain type of data; in this case the name `term_paper.doc` names a Word document, which we know by the “.doc” part. So, the basics are: Variables are names, they name values, they name a specific kind of data value, and the values can be changed.

In programming, things are pretty much the same. Variables are names made out of letters and numbers and underscores. The kind of information that variables name – the *data type* is what computer people call it – is not expressed by the dot file extension mechanism, but instead is declared using a program instruction. So, if I want three variables that will name integer values, that is, whole numbers, then I write

```
int x, y, z;
```

The term `int` is short for integer and the `x`, `y` and `z` are the variables; I could have used any other names, too. If I want variables that are decimal numbers, like 3.14, I write

```
float a, b, c;
```

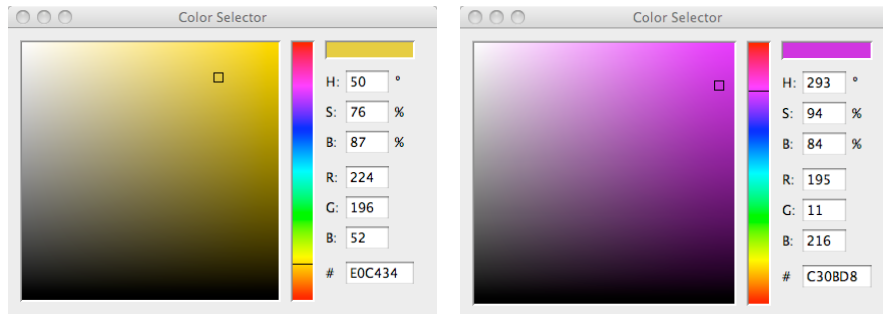
The term `float` is very strange – it's short for *floating point* – but we don't need to remember that at all; we will use `float`, however. These two programming commands are called *declarations*, because they declare what type of data values the variables will name. Notice that I have only specified what type of values they WILL have ... at the moment, they don't have values.

As one more example, I mention the `COLOR` data type, as in

```
color husky_gold, husky_purple;
```

Homework 6 Resource: Random Colors in Processing

Unlike ints and floats, the color data type is formed from three values corresponding to the three RGB values. (There's actually a fourth value, but let's forget that for now.) If I think Husky Gold is defined by a red value of 224, a green value of 196, and a blue value of 52, which I got from the Color Selector under the Processing Tools menu,



I write

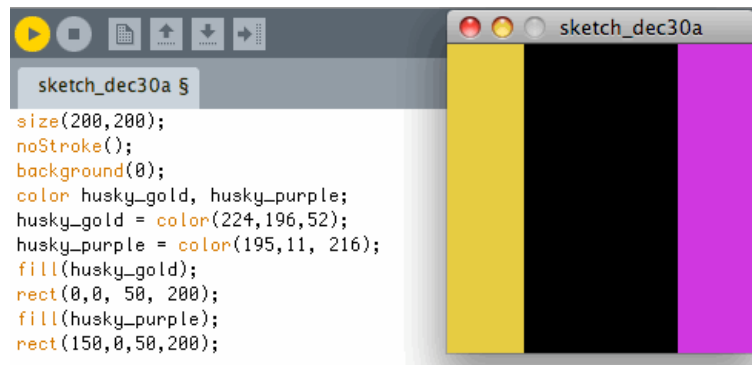
```
husky_gold = color(224, 196, 52);
```

which says to the computer to pack up the three values into whatever form the color data type has and make the result the value of the color variable `husky_gold`. Having the variable with a color value allows me to write

```
fill(husky_gold);
```

to get a gold color filled in shapes. See the program at right.

What's happening is the function `color()` packs the three values together so that we can treat them as a single thing. (We don't need a function to pack integers or floats together because they are already just a "single thing.")



Combining the ideas of randomness and data types, I generate a random color with,

```
float r, g, b;  
color any_color;
```

```
r = random(0, 255);  
g = random(0, 255);  
b = random(0, 255);
```

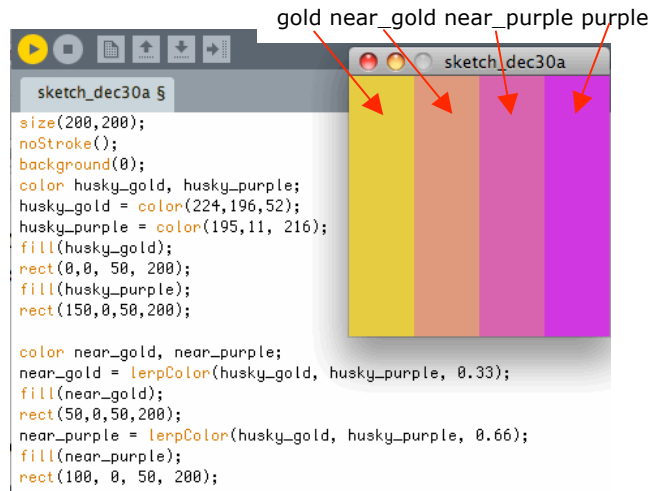
```
any_color = color(r, g, b);
```



The first two lines declare the variables, defining three float variables and one color variable. The next three lines randomly choose values for the RGB positions. Finally, the last line packages the three values into a color. When I used this for a background, I got (by dumb luck) a color pretty close to husky gold! Each time I run this program, I get some other background color.

A Color Tool

There is a very interesting function called `lerpColor()`. It uses a mathematical idea (that we don't need to know about) called *linear interpolation* to pick intermediate colors. What we do is give it two colors, say `husky_gold` and `husky_purple`, and it finds a color in between. Where in between? We also give it a fraction between 0 and 1 that tells. So, look at my program at right, modified from the Husky colors above. I have filled in the black region in the middle with two colors between gold and purple. On the left I picked a color "close" to gold, which is 1/3 of the way between gold and purple, and on the right, I picked a color "close" to purple, which is 2/3 of the way between the two. It's kind of attractive!



```
size(200,200);
noStroke();
background(0);
color husky_gold, husky_purple;
husky_gold = color(224,196,52);
husky_purple = color(195,11, 216);
fill(husky_gold);
rect(0,0, 50, 200);
fill(husky_purple);
rect(150,0,50,200);

color near_gold, near_purple;
near_gold = lerpColor(husky_gold, husky_purple, 0.33);
fill(near_gold);
rect(50,0,50,200);
near_purple = lerpColor(husky_gold, husky_purple, 0.66);
fill(near_purple);
rect(100, 0, 50, 200);
```