Art, Randomness, and Functions
Midterm

Step up your game!
## Common Issues

<table>
<thead>
<tr>
<th>No.</th>
<th>Correct Group Responses</th>
<th>Point Biserial</th>
<th>Correct Answer</th>
<th>Response Frequencies - * indicates correct answer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Upper 27%</td>
<td>Lower 27%</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>73.68%</td>
<td>84.31%</td>
<td>49.02%</td>
<td>0.37</td>
</tr>
<tr>
<td>2</td>
<td>58.95%</td>
<td>82.35%</td>
<td>41.18%</td>
<td>0.37</td>
</tr>
<tr>
<td>3</td>
<td>93.16%</td>
<td>96.08%</td>
<td>86.27%</td>
<td>0.14</td>
</tr>
<tr>
<td>4</td>
<td>92.11%</td>
<td>100.00%</td>
<td>76.47%</td>
<td>0.38</td>
</tr>
<tr>
<td>5</td>
<td>84.74%</td>
<td>94.12%</td>
<td>68.63%</td>
<td>0.34</td>
</tr>
<tr>
<td>6</td>
<td>87.89%</td>
<td>96.08%</td>
<td>70.59%</td>
<td>0.43</td>
</tr>
<tr>
<td>7</td>
<td>90.00%</td>
<td>100.00%</td>
<td>80.39%</td>
<td>0.28</td>
</tr>
<tr>
<td>8</td>
<td>93.68%</td>
<td>98.04%</td>
<td>84.31%</td>
<td>0.37</td>
</tr>
<tr>
<td>9</td>
<td>86.84%</td>
<td>98.04%</td>
<td>74.51%</td>
<td>0.34</td>
</tr>
<tr>
<td>10</td>
<td>15.79%</td>
<td>19.61%</td>
<td>11.76%</td>
<td>0.15</td>
</tr>
</tbody>
</table>
Final stage!

Read your peer reviews on CrowdGrader

Vote your reviews as helpful or unhelpful

- This affects other people's grades, so take it seriously
- Not a vote on whether you agree with the review

Votes due by Thursday before class
Creativity in Processing

- Homework 7
- Due next Tuesday at 5pm
- Get a programming partner

- Go crazy!
Assignment. Write four programs to do whatever you want (but don’t copy the examples above), and try to make them clever or interesting or cute or have some property that would interest a viewer. You should try to use those you have learned in your former Processing homework, because those are the basics and one goal of this assignment is to practice the basics. But, if you need some other feature of processing that you find in the reference page, go ahead and use it. The goal is creativity … but don’t spend forever on it either.
Why pair programming?

- Having a partner gives you a helper
- Many people find computers *more fun* when working with others
- Learning to work with computers is similar to learning a *foreign language*
  - lots of new words, codes
  - easier to learn if you have to “converse” in the language
- Working with a partner develops *teamwork skills*
- *Real-life programming* is almost always done in teams
- *Employers* are looking for team players
Processing as Art
If we generate it with Processing, is it art?

Piet Mondrian (1872-1944)

Jackson Pollock (1912-1956)

Josef Albers (1888-1976)
True “computer art” would be art created by a computer, not just art created by a person using a computer

- Could computers have aesthetic sensibilities?

Piet Mondrian (1872-1944)

http://en.wikipedia.org/wiki/Piet_Mondrian

- Cubist
- Could we generate this in Processing?
A little easier...

void draw() {
    background(128);
    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() {
    r = random(0, 255);
    g = random(0, 255);
    b = random(0, 255);
    outer = color(r, g, b);
    r = random(0, 255);
    g = random(0, 255);
    b = random(0, 255);
    inner = color(r, g, b);
}
Random Numbers

- Random numbers are really random number sequences
  - No matter how many numbers you know in the sequence, you can't predict the next one
  - Non-random: 2, 4, 6, 8, 10, ...
- Computers can't actually be random, they are completely predictable
  - But they can produce sequences that pass all the tests of randomness
  - Pseudo-random numbers (everyone just drops the pseudo-)

- In Processing:
  
  random(<smallest number>, <largest number>)
  
  - Get back a number (unpredictable) between the two limits, including end points
- To generate a random number between 0 and 255, write \texttt{random(0, 255)}
- To generate a random number between 0 and 1, write \texttt{random(0, 1)}
The function lerpColor()
- Generates intermediate colors between two colors
- You give it two colors
  - Gold and purple
  - It finds mixes between those colors
  - Also give it a fraction (like .33 or .66) between 0 and 1
    - specifies the mix ratio
`lerpColor()`

color left = color(255, 210, 10);
color right = color(255, 0, 255);
color mix;

void setup() {
    size(400, 200);
    background(0);
    noStroke();
    fill(left);
    rect(0, 0, 100, 200);
    mix = lerpColor(left, right, 0.33);
    fill(mix);
    rect(100, 0, 100, 200);
    mix = lerpColor(left, right, 0.66);
    fill(mix);
    rect(200, 0, 100, 200);
    fill(right);
    rect(300, 0, 100, 200);
}

`lerp` is short for *linear interpolation*
Remember how colors work...

Colors are sets of Red Green Blue values from 0 to 255

11111111 11010010 00001010 11111111 00000000 11111111
Mixing Colors

Colors are sets of Red Green Blue values from 0 to 255

255  210  10

255  105  122  Halfway color (0.5 mix)

255  0   255
Review: Crazy Rectangles

void setup() {  
  size(600, 600);  
}

void draw() {  
  fill(255 - abs(mouseX - pmouseX));  
  rect(mouseX, mouseY, mouseX, mouseY);  
}

Remember:

System variables
mouseX mouseY
pmouseX pmouseY
width height
frameCount
float r, g, b;
color newColor;

void setup() {
  size(400, 400);
  background(0);
  frameRate(20);
}

void draw() {
  r = random(0, 255);
  g = random(0, 255);
  b = random(0, 255);
  newColor = color(r, g, b);
  fill(newColor);
  rect(mouseX, mouseY, mouseX, mouseY);
}
float r, g, b;
color newColor, oldColor;

void setup() {
    size(400, 400);
    background(0);
    frameRate(20);
}

void draw() {
    r = random(0, 255);
    g = random(0, 255);
    b = random(0, 255);
    newColor = color(r, g, b);
    newColor = lerpColor(newColor, oldColor, 0.8);
    fill(newColor);
    rect(mouseX, mouseY, mouseX, mouseY);
    oldColor = newColor;
}
```cpp
int ballX, ballDia = 50;
void setup() {
    size(400, 400);
    ballX = -ballDia/2;
}

void draw() {
    background(120);
    if (ballX > width+ballDia/2) {
        ballX = -ballDia/2;
    }
    ellipse(ballX, height/2, ballDia, ballDia);
    ballX = ballX + 1;
}
```
The jumping ball...

A. Ball moves across, jumping back to left edge and changing color as soon as it touches the right edge.

B. Ball moves across as one color until it moves off the right edge then changes color and moves back in from the left edge.

C. Ball moves across until half way off the right edge (showing just half a circle) then changes color and jumps to the left side.

D. Ball moves across then disappears and doesn't come back.

```java
void draw() {
    background(0); // try removing
    if (ballX > width+ballDia/2) {
        r = random(0, 255);
        g = random(0, 255);
        b = random(0, 255);
        newColor = color(r, g, b);
        ballX = -ballDia/2;
    }
    fill(newColor);
    ellipse(ballX, height/2, ballDia, ballDia);
    ballX = ballX + 1;
}
```
Pointilism
(and other image manipulation)

- [Link to Tutorial](http://processing.org/tutorials/pixels/)
Tints, Transparency, more...
Jackson Pollock

- [link](http://en.wikipedia.org/wiki/Jackson_pollock)
- Abstract expressionist
void draw()
{
    //create random coordinate seeds
    float Xrandom = random(-2,2);
    float Yrandom = random(-2,2);

    //change line thickness
    if(tick > 2)
    {
        float newthick = sqrt(pow(mouseX-mouseX, 2) + pow(mouseY-mouseY, 2));

        if(newthick > thickness && thickness < 10)
        {
            thickness += 1;
        }
        if(newthick < thickness && thickness > 1)
        {
            thickness -= 1;
        }
    
    }
Repeating with Variation
- **for loops**
  - We've seen this before!
- **Functions**
  - We've used functions, defined a few like `draw()` ourselves

Let's review for loops quickly
- Repeating commands is powerful
  - Lightbot 2.0 used recursion
    - function calls itself
  - Symbolic Lightbot used a prefix number
    - 4:Step means do Step Step Step Step
  - Processing uses a **for** loop

```java
void setup() {
    size(500, 200);
    background(0);
    fill(255);
    for(int i = 0; i < 16; i++) {
        ellipse(100 + 25*i, 100, 15, 15);
    }
}
```
Here are the parts of a for loop (all are required)

```java
for (int i = 0; i < 16; i++) {
    // stuff to be repeated
    ellipse(100 + 25*i, 100, 15, 15);
}
```

Result: we get 16 copies of the ellipse Pac-Man pills!
What does this code display?

A. Lots of little pills going off the left and right side of the screen
B. Rows of little pills repeating with every mouse click
C. A row of little pills that change color with every mouse click
D. Pulsing images of circles changing colors and growing every frame
Functions
Processing Reference

- http://processing.org/learning/basics/functions.html

Functions.

The `drawTarget()` function makes it easy to draw many distinct targets. Each call to `drawTarget()` specifies the position, size, and number of rings for each target.

```java
void setup() {
  size(640, 360);
  background(0);
  noStroke();
  noLoop();
}
```
Instructions formed from simpler actions

- Check out this screenshot
- It is partway through an instruction ... it's beacon is lit, but the tile is still blue
- To the programmer the instruction is one step (monolithic)
- To the agent each instruction is a series of steps
Collecting the operations together and giving them a name is **functional abstraction**

- The group of operations performs some function but we ignore the details
- Only need to refer to the name we gave it
- This is **AMAZINGLY** powerful
- Why?
  - We can forget about the operations
  - Only thing about what function they do
  - We'll talk more about this...
Example: Abstracting your morning

Get dressed

Dress bottom half
- Put on socks

Dress top half
- Put on pants
- Put on shirt
Functions abstract by packaging computation

- F1() packages actions (e.g. “process a riser”)
The Function becomes a Concept

- Because F1() “processes a riser”, you can think of the task as:
  - Process a riser   F1
  - Move to next riser
  - Process a riser   F1
  - Move to next riser
  - Process a riser   F1

- With F1() as a concept, we simplified the programming to 9 steps instead of 21 steps
Calling functions

What happens if we comment out the second call?

```c
void draw() {
    drawTarget(width*0.25, height*0.4, 200, 4);
    //drawTarget(width*0.5, height*0.5, 300, 10);
    drawTarget(width*0.75, height*0.3, 120, 6);
}
```
void draw() {
    drawTarget(width*0.25, height*0.4, 200, 4);
    //drawTarget(width*0.5, height*0.5, 300, 10);
    drawTarget(width*0.75, height*0.3, 120, 6);
}
 Functions

- Here is the form of a function definition

  `<return type> <name> ( <param list> ) {`
  `  <body>`
  `}

- Example:

  ```cpp
  void draw_box(int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
  }
  
  color pinkish() {
    return color(255, 200, 180);
  }
  ```
Return types

- Functions that do something but do not return a value
  - Use `void` as `<return type>`
- Functions that return a value must specify the type
  ```
  void draw_box(int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
  }
  
  color pinkish() {
    return color(255, 200, 180);
  }
  ```
Parameters

- Parameters are the values used as input to the function
  - Functions don't have to use parameters
    - Still need parentheses even if no parameters
- Type of each parameter must be given

```cpp
void draw_box(int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
}

color pinkish() {
    return color(255, 200, 180);
}
```
Parameters

- Customize each function call to a specific situation
- They are the input to the function
- Terminology:
  - **Parameters** are the names of the values used inside the function body
  - **Arguments** are the values from outside to be used for each of the parameters
Arguments become Parameters

- If the **definition** has \( n \) parameters
  - The **call** needs \( n \) arguments

- The parameters and arguments correspond

```java
void draw() {
    drawTarget(width*0.25, height*0.4, 200, 4);
    drawTarget(width*0.5, height*0.5, 300, 10);
    drawTarget(width*0.75, height*0.3, 120, 6);
}

void drawTarget(float xloc, float yloc, int size, int num) {
    float grayvalues = 255/num;
    float steps = size/num;
    for (int i = 0; i < num; i++) {
        fill(i*grayvalues);
        ellipse(xloc, yloc, size - i*steps, size - i*steps);
    }
}
```
A function returns its value with the **return** statement

- Value following the **return** is the result
- Function is done when it reaches **return** or the end of the body

```c
void draw_box(int x_pos, int y_pos) {
    rect(x_pos, y_pos, 20, 20);
}
```

```c
color pinkish() {
    return color(255, 200, 180);
}
```
End