

Parameters And Arguments

Lawrence Snyder
University of Washington, Seattle

Form of Functions In Processing ...

- We saw functions in Lightbot and in Moonwalk ... how do they look in Processing?
- Recall these components of the function declaration

```
void littleRedBox( ) {  
    fill(255,0,0);  
    rect(100,100,5,5);  
}
```

Form of Functions In Processing ...

- We saw functions in Lightbot and in Moonwalk ... how do they look in Processing?
- Recall these components of the function declaration

declaration

Name starts with letter,
uses letters, numbers
underscore; doesn't
collide

Required; contain
parameters if any

```
void littleRedBox( ) {  
    fill(255,0,0);  
    rect(100,100,5,5);  
}
```

Form of Functions In Processing ...

- We saw functions in Lightbot and in Moonwalk ... how do they look in Processing?
- Recall these components of the function

declaration

Function's return type –
Use 'void' if none

Name starts with letter,
uses letters, numbers
underscore; doesn't
collide

Required; contain
parameters if any

```
void littleRedBox( ) {  
    fill(255,0,0);  
    rect(100,100,5,5);  
}
```

Form of Functions In Processing ...

- We saw functions in Lightbot and in Moonwalk ... how do they look in Processing?
- Recall these components of the function

declaration

Function's return type –
Use 'void' if none

Name starts with letter,
uses letters, numbers
underscore; doesn't
collide

Required; contain
parameters if any

```
void littleRedBox( ) {  
    fill(255,0,0);  
    rect(100,100,5,5);  
}
```

Required; contain
function definition

Form of Functions In Processing ...

- We saw functions in Lightbot and in Moonwalk ... how do they look in Processing?
- Recall these components of the function

declaration

Function's return type –
Use 'void' if none

Name starts with letter,
uses letters, numbers
underscore; doesn't
collide

Required; contain
parameters if any

The definition
implements
the function

```
void littleRedBox( ) {  
    fill(255,0,0);  
    rect(100,100,5,5);  
}
```

Required; contain
function definition

Relating To Earlier Functions

- In the Lightbot functions (Assignment 3) we used the form

F.turn_around() Right, Right.

- IF we did something like this in Processing, it would look different but have the same parts

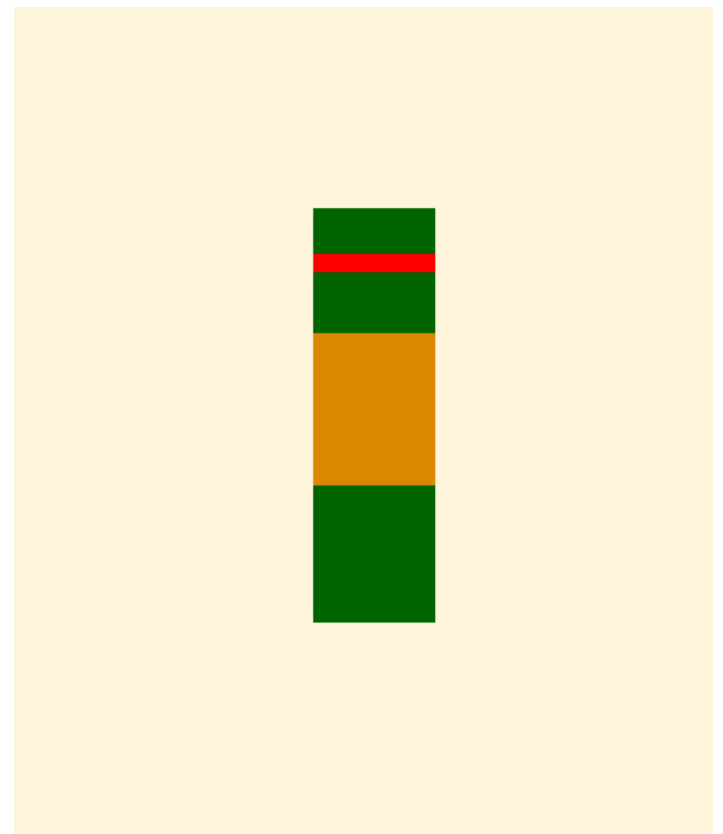
No void needed	Name	Works like {	Definition	Works like }
----------------	------	--------------	------------	--------------

void F.turn_around() Right, Right .

An Example Of The Parts

- Find the <type>, <name>, parameter parens, the braces enclosing the def, and the definition

```
void raff() {  
    fill(0,100,0);  
    rect(240 ,260, 40, 45);  
    fill(219,136,0);  
    rect(240 ,210, 40, 50);  
    fill(0,100,0);  
    rect(240 ,190, 40, 20);  
    fill(255,0,0);  
    rect(240 , 184, 40, 6);  
    fill(0,100,0);  
    rect(240 , 169, 40, 15);  
}
```

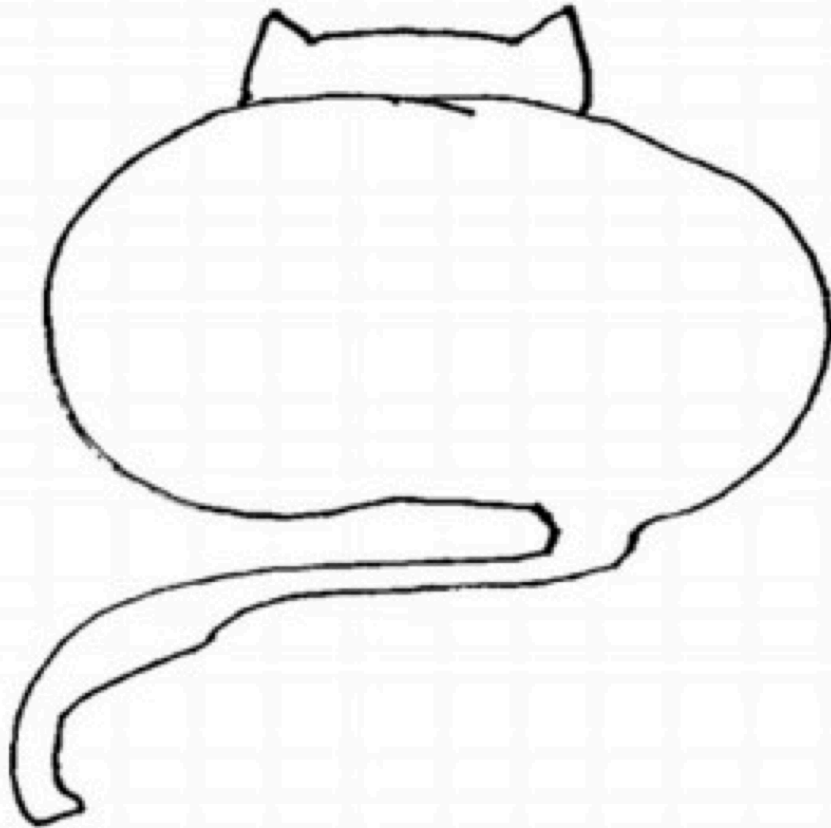


Might As Well Memorize the 5

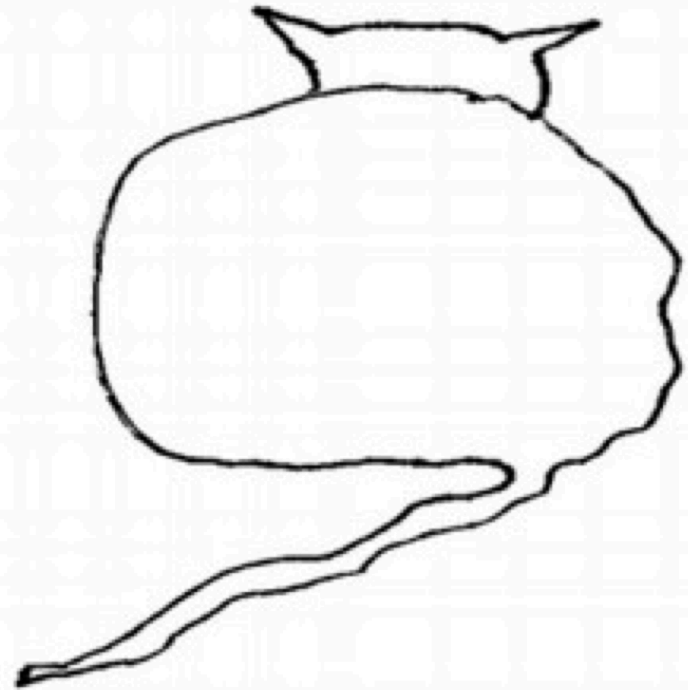
- All functions have these parts in one form or another:
 - Return Type – not always applicable
 - Name
 - Parentheses (even if there are no parameters)
 - Enclosing braces (or other symbols)
 - Definition – normal statements of the language

Return Types

What is a return type?



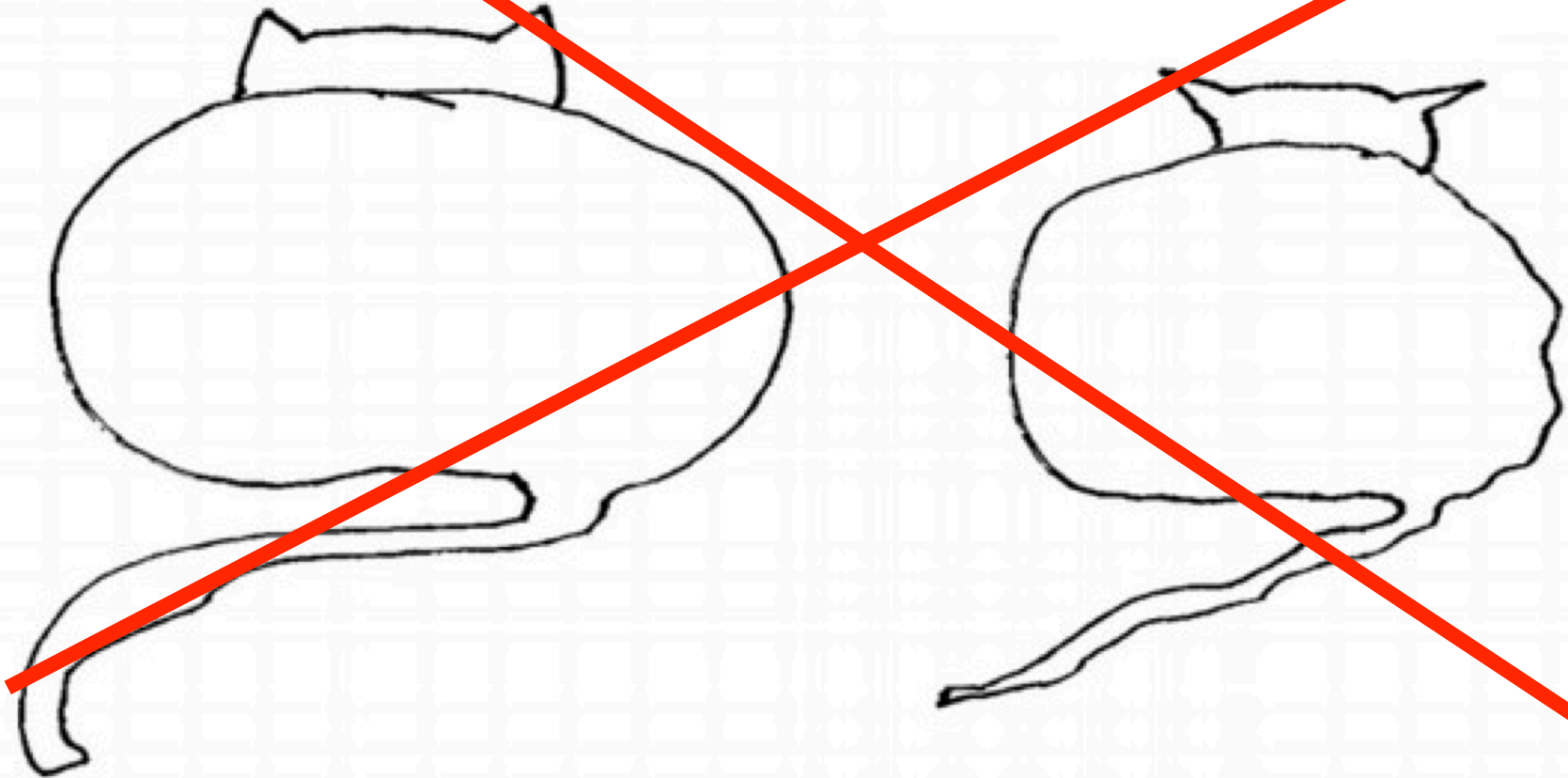
Unwanted font.



Return Types

What does return type mean?

Unwanted font.



Return Types

- A return type is the *kind of value* a function computes

- So, it will be one of the data types, e.g. **int** or **float**
- If the function doesn't return anything, use **void**

```
float areaFromCorner( ) {  
    return mouseX * mouseY;  
}
```

```
int randomEvenNumberLT20 ( ) {  
    return 2 * int(random(0, 10));  
}
```

```
void change2red ( ) {  
    fill(255,0,0);  
}
```

Parameters

- Parameters are the information that go inside of the parentheses

```
void whiteBox5x5 (int xPos, int yPos) {  
    fill(255,255,255);  
    rect(xPos, yPos, 5, 5);  
}
```




- Notice:
 - The datatype of the parameter must be given
 - Parameters are separated by commas
 - Parameter names like other names – no conflicts

Functions With Parameters

- Parameters give the data for the function to operate on ... then to do the same operation on different data just change the values they get

Input to the function: x position, y position and color of box

```
void tile(int xPos, int yPos, color tinto) {  
    fill( tinto );           //set box color  
    rect(xPos, yPos, 5, 5); //draw small box  
}
```



Notice:

- (a) We always give a name to the data, e.g. `xPos`
- (b) We always say what type the data is, e.g. `int`
- (c) The order, which we choose, will always have to be followed

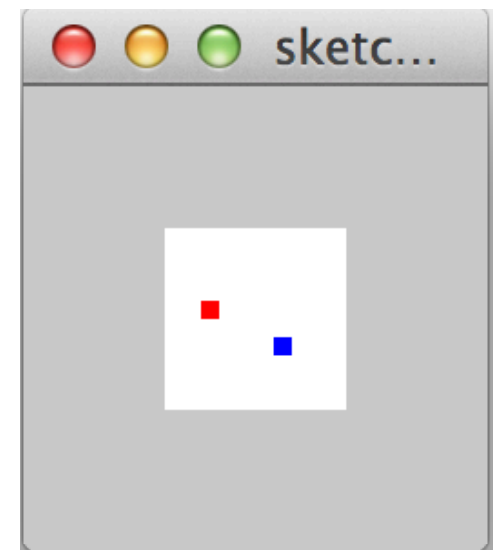
The Function Call

- Writing out how the function works is called its **definition** ... write it just once
- To use the function, we **call** it by giving the data to be used when it runs ... use it repeatedly

```
tile(10, 20, color(255,0,0)); //red tile  
tile(30, 30, color(0,0,255)); //blue tile
```

- Notice:

- (a) The data we give are called **arguments**
- (b) Only use values of the right type
- (c) Order of arguments must match order of the parameters they go with



Parameter/Argument Summary

- Define functions just once

```
void tile(int xPos, int yPos, color tinto) {  
    fill( tinto );           //set box color  
    rect(xPos, yPos, 5, 5); //draw small box  
}
```

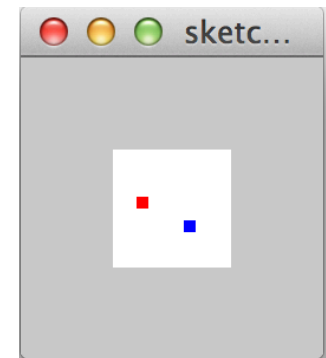
- Use **parameters** for all values that will change
- Call function when needed & give **arguments**

```
tile(10, 20, color(255,0,0)); //red tile  
tile(30, 30, color(0,0,255)); //blue tile
```

- A call is as if values used directly

```
void tile(int xPos, int yPos, color tinto) {  
    fill(color(255,0,0)); //set box color  
    rect(10, 20, 5, 5); //draw small box  
}
```

Diagram illustrating argument passing: yellow arrows point from the values 10, 20, and color(255,0,0) in the function call above to the corresponding parameter names (xPos, yPos, tinto) in the function definition below.



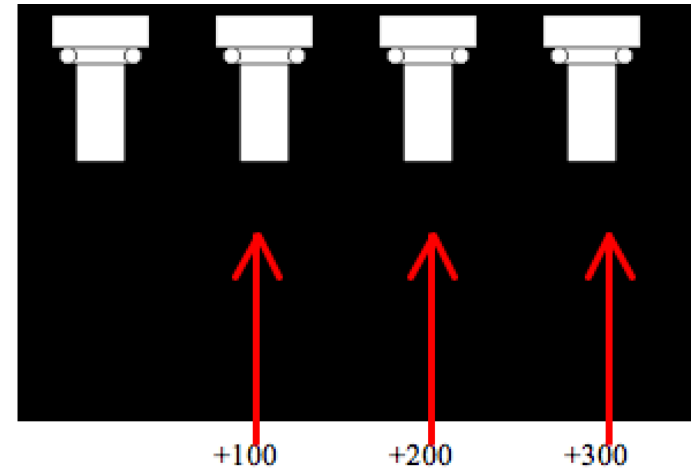
Pause To Consider Abstraction ...

- Recall yesterday's lab
- Task: Make four column

UW logo

```
rect(0,0,60,20);  
rect(10,20,40,10);  
ellipse(10,25,10,10);  
ellipse(50,25,10,10);  
rect(15,30,30,60);
```

```
rect(0+100,0,60,20);  
rect(10+100,20,40,10);  
ellipse(10+100,25,10,10);  
ellipse(50+100,25,10,10);  
rect(15+100,30,30,60);
```



You were asked to make the additional columns by copying the code, and editing it to add +100 or +200 or +300

This request should have disgusted you! You probably said, “Wait, Larry, the column is an abstraction ... shouldn't we put it in a function???” And, of course, you were right!

A Column Packaged As A Function

Layout and color are constant; position is variable, so it is parameterized.

```
void draw( ) {  
    column(10, 20);  
    column(110, 20);  
    column(210, 20);  
    column(310, 20);  
}
```

```
void column(int xPos, int yPos) {  
    rect(xPos, yPos, 60, 20);  
    rect(10+xPos, 20+yPos, 40, 10);  
    ellipse(10+xPos, 25+yPos, 10, 10);  
    ellipse(50+xPos, 25+yPos, 10, 10);  
    rect(15+xPos, 30+yPos, 30, 60);  
}
```

Illustrate What We Just Learned

- Today, we solve a problem that is much like Assignment 6
- The demo gives a chance to discuss how to translate the instructions of the exercise into a solution ...
- Use this exercise as a guide for Assignment 6

Just Do It!

The Set Up

- Problems of all sorts often begin as a rehash of what is known or given

“Here we are given a Processing function to draw a mouse of a given color at a given place.”



- What to do: Understand what is given.
- In this case look at the code and notice how it controls the color and the position.

Mouse Function

- Check it out!

```
mouse(0,0,  
color(0,200,200));
```



```
void mouse(int xpos, int ypos, color c) {  
  fill(c);  
  noStroke( );  
  ellipse(50+xpos,50+ypos,50,50);  
  ellipse(25+xpos,30+ypos,30,30);  
  ellipse(75+xpos,30+ypos,30,30);  
  fill(0);  
  ellipse(40+xpos,44+ypos, 10,10);  
  ellipse(60+xpos,44+ypos, 10,10);  
  stroke(0);  
  line(20+xpos,50+ypos, 48+xpos,60+ypos);  
  line(80+xpos,50+ypos, 52+xpos,60+ypos);  
  line(25+xpos,70+ypos, 48+xpos,60+ypos);  
  line(75+xpos,70+ypos, 52+xpos,60+ypos);  
}
```

Example Task

- Task statements usually give the goal, plus a series of additional properties or conditions that the solution should have

“Make a function to draw a row of eight mice so that their ears overlap and all but one are the same given color; the odd one is to be red”

- What to do: Break the task down into subtasks that are easy to do.

Example Task (continued)

- “Make a function to draw a row” means we have to outline the structure:

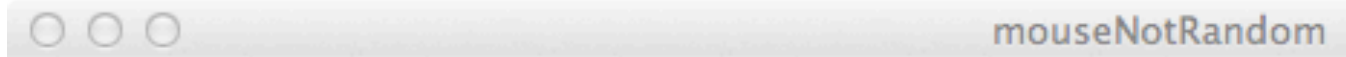
```
void row(    ) {  
    }  
}
```
- “Draw 8 mice so their ears overlap” means
 - we need x, y parameters for the position of the first
 - we draw a second one and try to adjust the x -coordinate to align their ears
 - then we draw the rest using the same alignment
- “All but one are the same given color” means we need a parameter for the given color
- “One is red” means to pick one; color it red

Mouse Row

```
row(0, 0, color(100))
```

All of those steps produce ...

```
void row (int xpos, int ypos, color c ) {  
  mouse(xpos+0,  ypos, c);  
  mouse(xpos+50,  ypos, c);  
  mouse(xpos+100, ypos, color(255,0,0));  
  mouse(xpos+150, ypos, c);  
  mouse(xpos+200, ypos, c);  
  mouse(xpos+250, ypos, c);  
  mouse(xpos+300, ypos, c);  
  mouse(xpos+350, ypos, c);  
}
```



Another Task

“Make a swarm of mice by drawing six rows each of a different color”

- The task leaves many things unspecified, and in that case we will make a sensible decision about what to do.

Another Task (continued)

- “Make a swarm of mice” means we need another function, because the concept of swarm is different from the row we have. *Remember – functions are abstractions, each represents a different concept!*

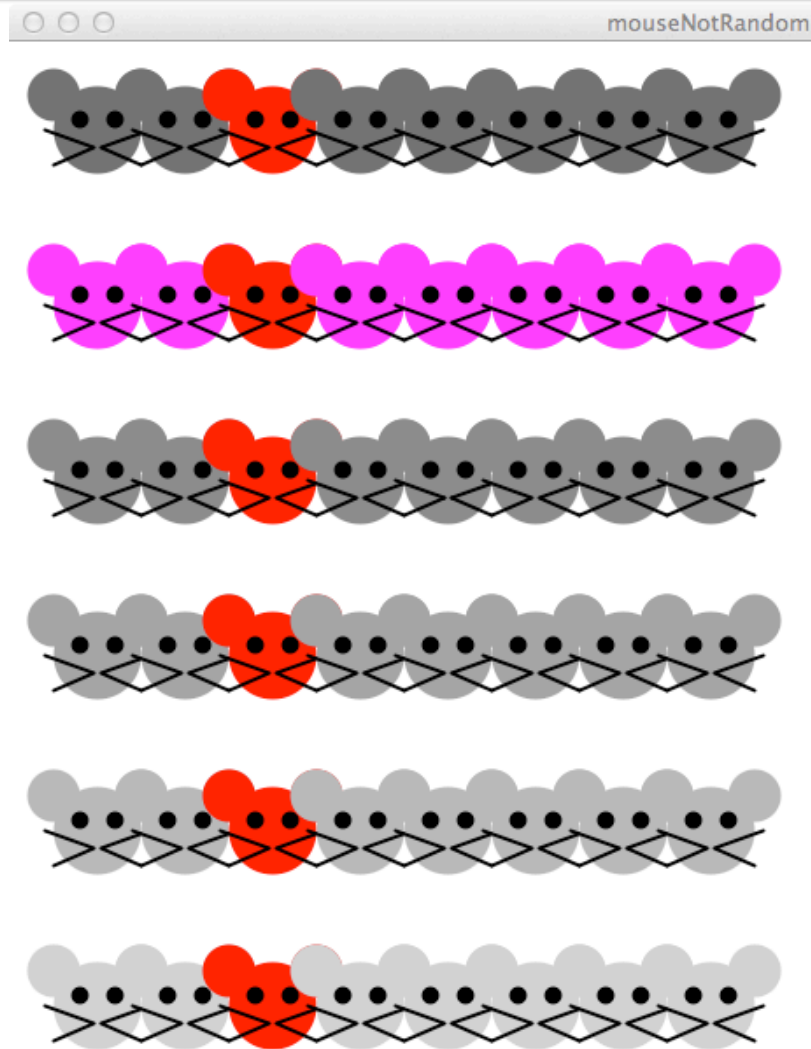
```
void swarm(    ) {  
  
}
```

- The starting position of rows is not stated, so like always, we add parameters for xpos, ypos
- Row separation not stated, so we just pick it
- Colors not stated, so we just pick some

Solution Code

```
void swarm (int xpos, int ypos) {  
    row(xpos, ypos+0,    color(100,100,100));  
    row(xpos, ypos+100, color(255,0,255));    // pink  
    row(xpos, ypos+200, color(125,125,125));  
    row(xpos, ypos+300, color(150,150,150));  
    row(xpos, ypos+400, color(175,175,175));  
    row(xpos, ypos+500, color(200,200,200));  
}
```

Solution



Summary

- We worked through an exercise similar to Assignment 6.
 - As we worked through it, we studied how to solve these kinds of problems
 - We read carefully what was required
 - When we had a “largish” task, we broke it down to several “smallish” tasks we could easily solve
 - When the problem didn’t specify what to do, we just make a sensible decision, which often meant adding a parameter so we change it if needed