Functions in Processing

CSE 120 Winter 2018

Instructor:
Justin Hsia

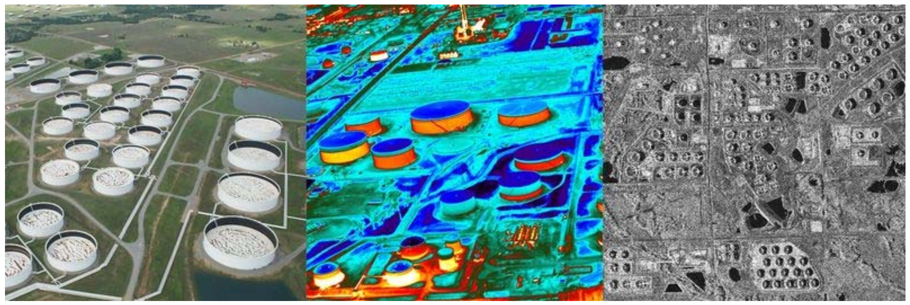
Teaching Assistants:
Anupam Gupta, Cheng Ni, Eugene Oh,
Sam Wolfson, Sophie Tian, Teagan Horkan

Satellites Are Reshaping How Traders Track Earthly Commodities

"Companies… are using satellites to try to shed light on tightly held secrets in the commodity trading world, from coal mine productivity to crude oil storage. While doubts remain around the accuracy and consistency of the data, there could come a day when traders can track supply and demand of raw materials, the operations of producers and consumers and even the output of entire economies in near-real time.

"Firms… then use artificial intelligence to scan millions of those images and translate them into useful data."

- https://www.bloomberg.com/news/articles/2017-12-16/satellites-are-reshaping-how-traders-track-earthly-commodities



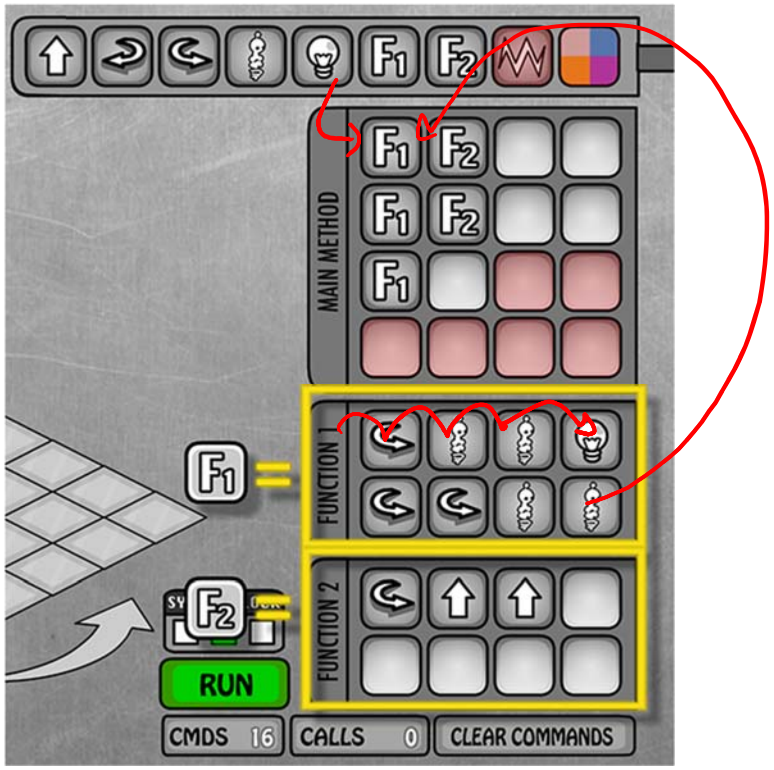
Administrivia

- Assignments:
  - Lego Family due tonight (1/17)
  - Website Setup due before lab tomorrow (1/18)
  - Reading Check 2 due tomorrow (1/18)
- Editing your portfolio from home
  - Download and install Cyberduck & VS Code
  - Re-do Steps 3 & 4 from the website setup
- Make sure to take advantage of office hours and Piazza!

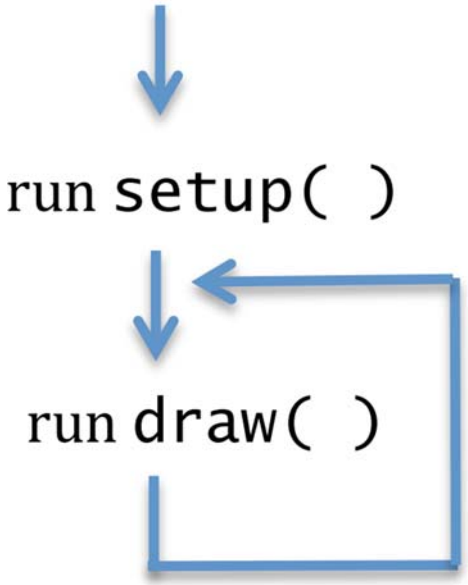
Functions (So Far)

- Used for **abstraction**
  - *Detail Removal:* subtasks with intuitive names
  - *Generalization:* don't repeat code

Lightbot:



Processing:



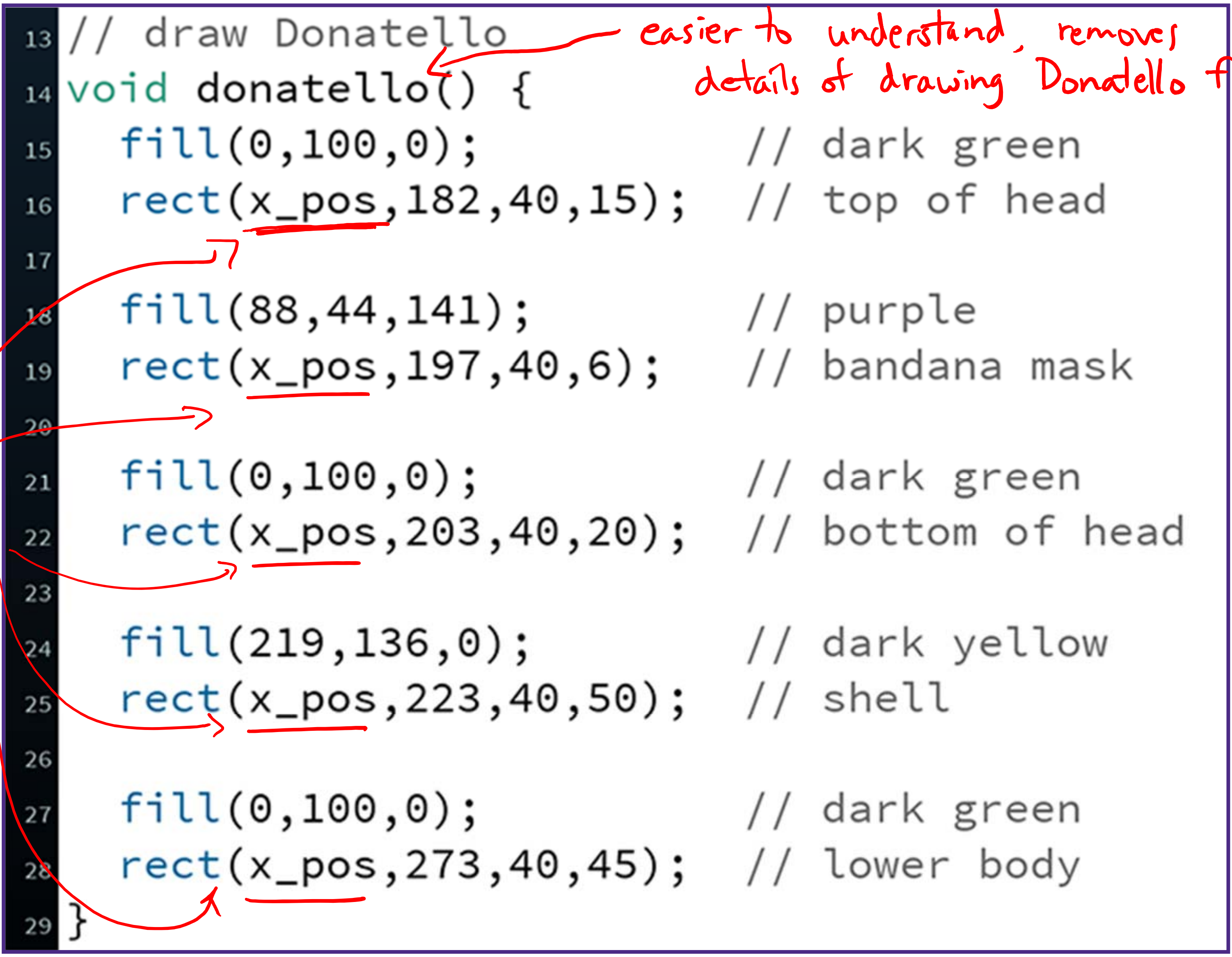
- `line()`, `rect()`, …
- `min()`, `max()`

Program Execution with Functions

- Functions "break" the normal sequential execution model
  - When function is called, begin execution of function code
  - When end of function is reached, jump back to where function was called from

- **Analogy:** Song lyrics with a repeated chorus
  - Example: Survivor – Eye of the Tiger
    - Verse 1, Verse 2, Chorus, Verse 3, CHORUS, Verse 4, CHORUS, Outro
  - *Parameterized* Example: Old MacDonald
    - Chorus(cow,moo), Chorus(pig,oink), Chorus(duck,quack), Chorus (sheep,baa)

Donatello as a Function [DEMO]



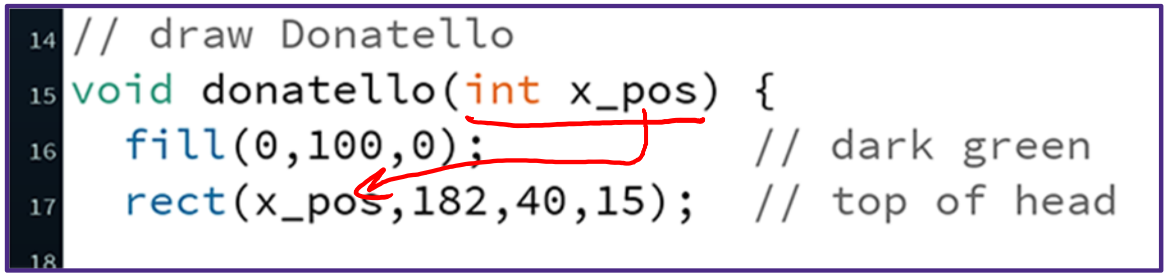
```
// draw Donatello
void donatello() {
  fill(0,100,0);             // dark green
  rect(x_pos,182,40,15);     // top of head

  fill(88,44,141);           // purple
  rect(x_pos,197,40,6);      // bandana mask

  fill(0,100,0);             // dark green
  rect(x_pos,203,40,20);     // bottom of head

  fill(219,136,0);           // dark yellow
  rect(x_pos,223,40,50);     // shell

  fill(0,100,0);             // dark green
  rect(x_pos,273,40,45);     // lower body
}
```

easier to understand, removes details of drawing Donatello from draw()

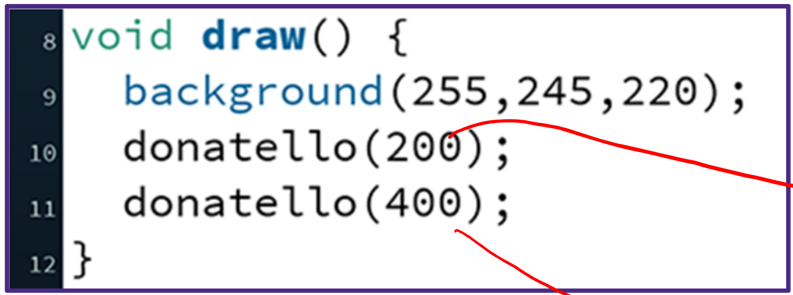
calling donatello() always reads from the same x-pos variable

Donatello Function *Parameterized*

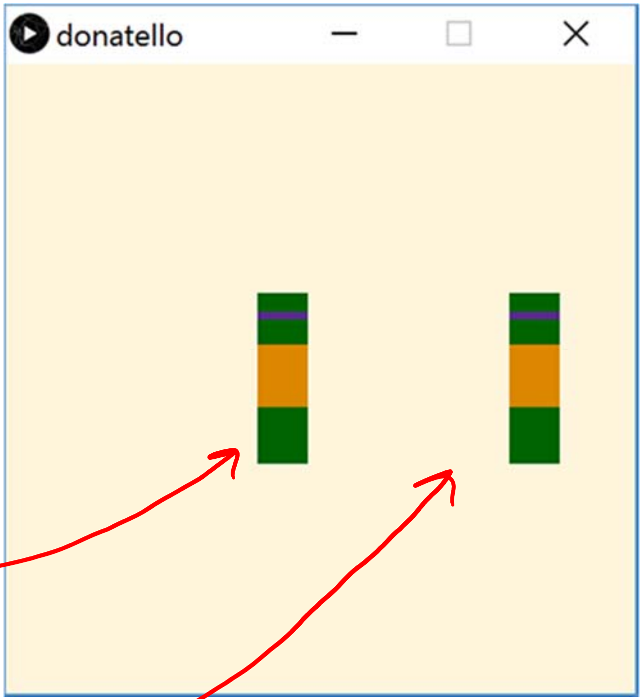
- Can now call `donatello()` function with different `x_pos`



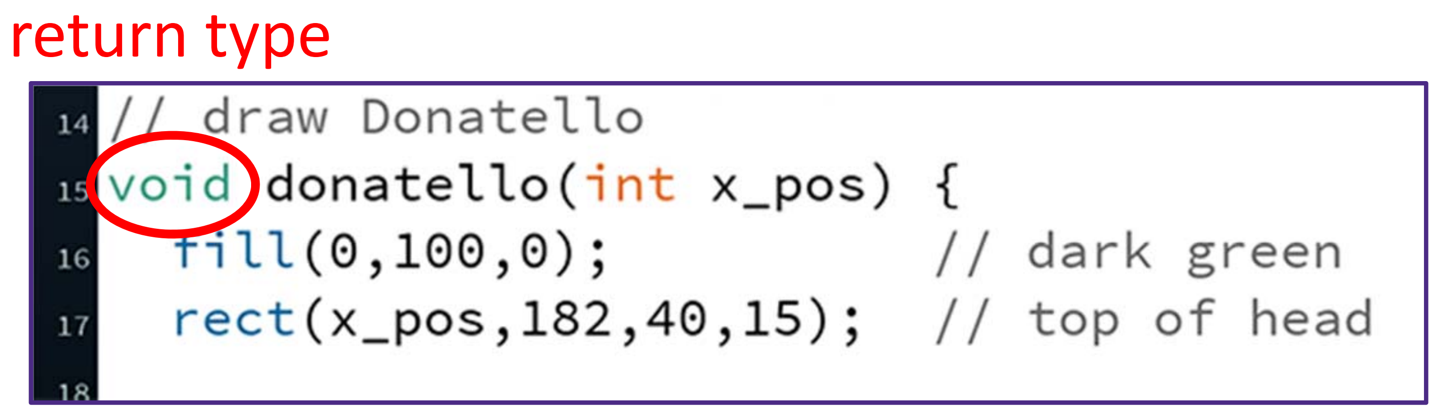
```
// draw Donatello
void donatello(int x_pos) {
  fill(0,100,0);            // dark green
  rect(x_pos,182,40,15);    // top of head
```



```
void draw() {
  background(255,245,220);
  donatello(200);
  donatello(400);
}
```



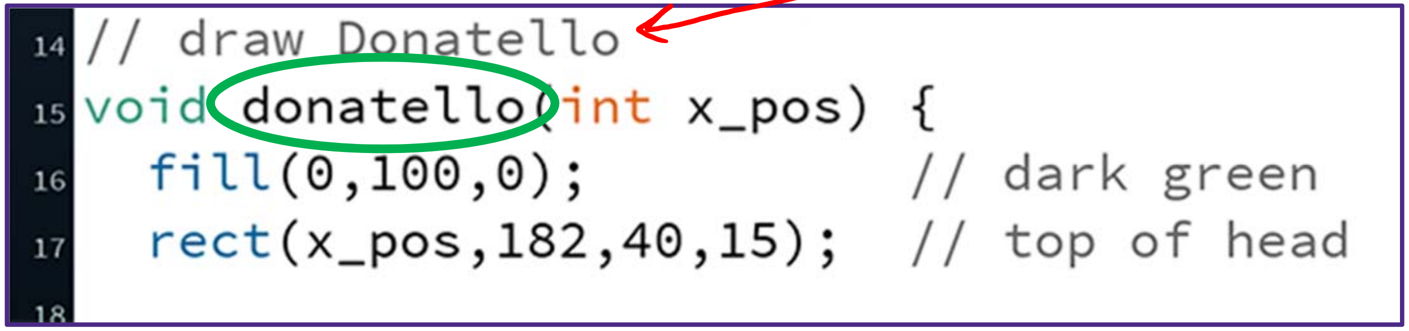
Return Type



```
// draw Donatello
void donatello(int x_pos) {
  fill(0,100,0);              // dark green
  rect(x_pos,182,40,15);      // top of head
```

- What the function sends back to whoever called it
  - Can be any of the datatypes: `int`, `float`, `color`, etc.
  - If not returning anything, then we use `void`

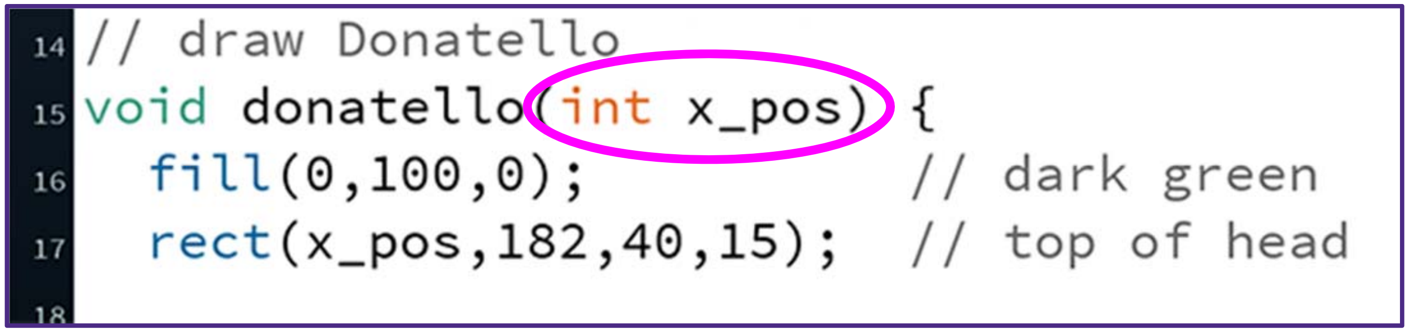
Function Name



```
// draw Donatello
void donatello(int x_pos) {
  fill(0,100,0);            // dark green
  rect(x_pos,182,40,15);    // top of head
```

- Does not matter to computer, but does to humans
  - Should describe what the function does (good style to include a comment as well!)
- Subject to same naming constraints as variables
- No two functions (or variables) can have the same name ← confuses the computer

Parameters



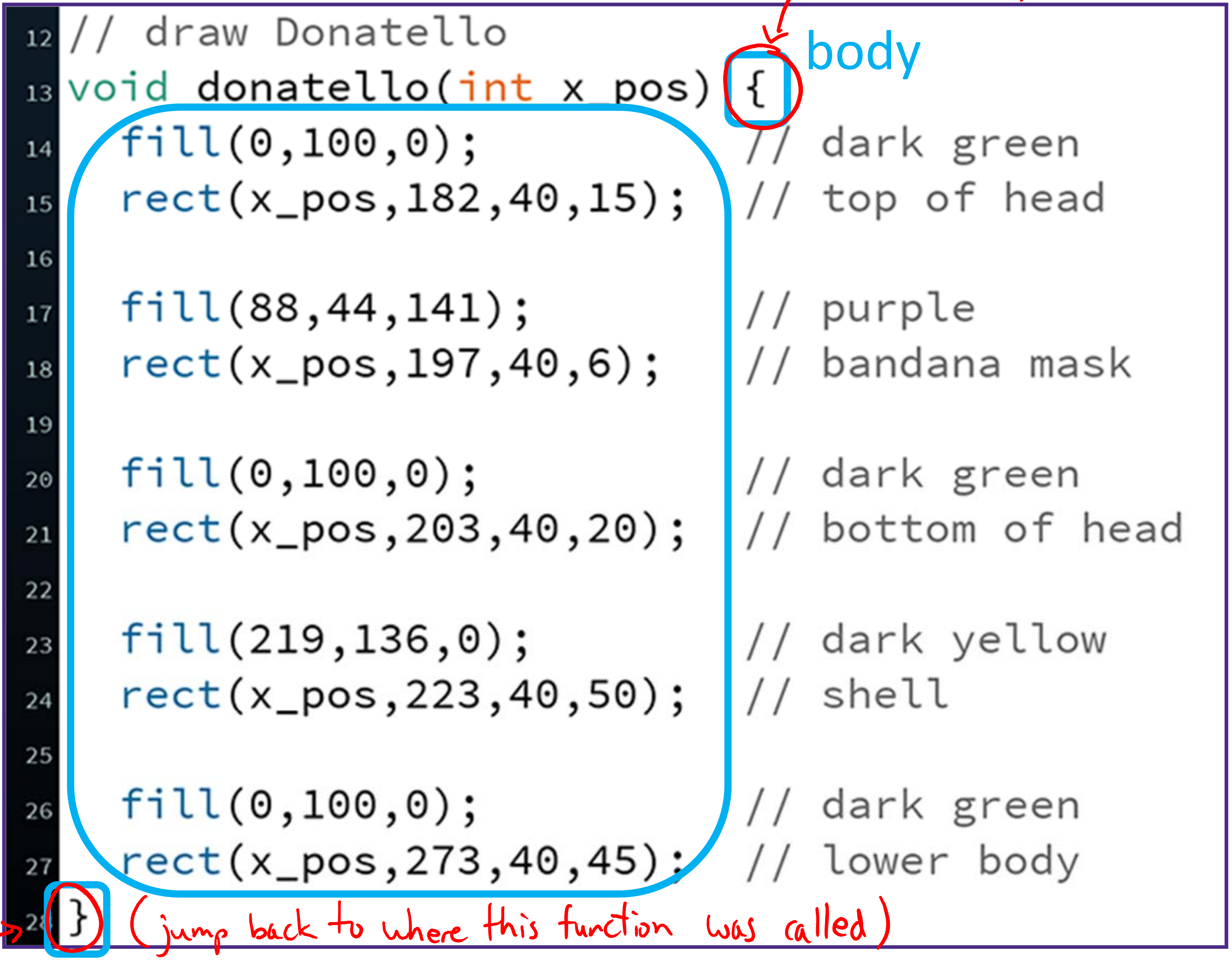
```
// draw Donatello
void donatello(int x_pos) {
  fill(0,100,0);             // dark green
  rect(x_pos,182,40,15);     // top of head
```

- Required part of every function definition
  - Must be surrounded by parentheses
  - If no parameters, parentheses are left empty

- Datatype and name for every parameter must be specified just like declaring a variable
  - Separate parameters with commas

Function Body

start of body



```
// draw Donatello
void donatello(int x_pos) {
  fill(0,100,0);              // dark green
  rect(x_pos,182,40,15);      // top of head

  fill(88,44,141);            // purple
  rect(x_pos,197,40,6);       // bandana mask

  fill(0,100,0);              // dark green
  rect(x_pos,203,40,20);      // bottom of head

  fill(219,136,0);            // dark yellow
  rect(x_pos,223,40,50);      // shell

  fill(0,100,0);              // dark green
  rect(x_pos,273,40,45);      // lower body
} (jump back to where this function was called)
```

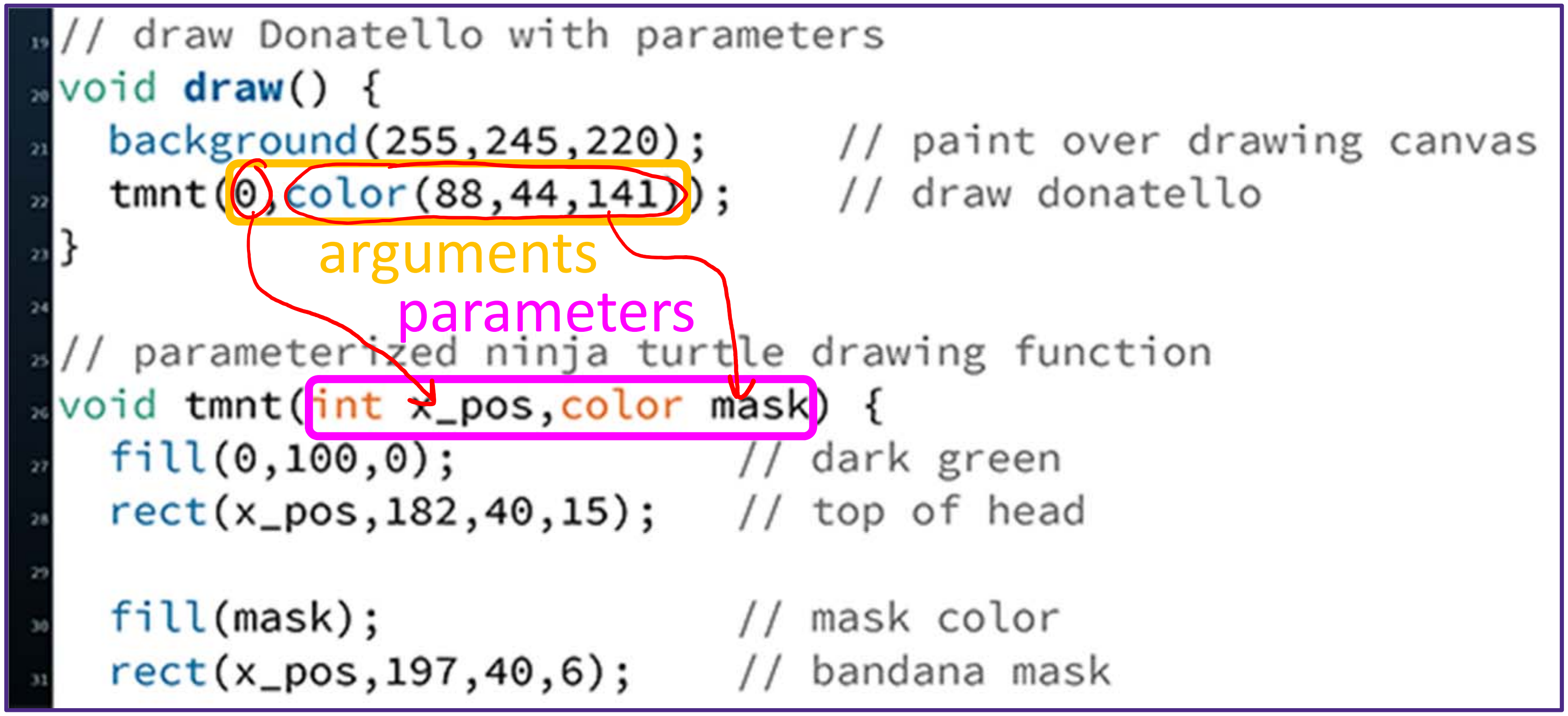
Lightbot Functions

- Lightbot functions had a different syntax, but similar parts:

function name parameters body

```
F.turn_around() Right, Right.
```

Parameters vs. Arguments [DEMO]



```
// draw Donatello with parameters
void draw() {
  background(255,245,220);      // paint over drawing canvas
  tmnt(0,color(88,44,141));     // draw donatello
}

// parameterized ninja turtle drawing function
void tmnt(int x_pos,color mask) {
  fill(0,100,0);              // dark green
  rect(x_pos,182,40,15);     // top of head

  fill(mask);                 // mask color
  rect(x_pos,197,40,6);      // bandana mask
```

- Implicit parameter/variable initialization with argument values

similar to:

```
    ...
    tmnt();
}
void tmnt() {
    int x-pos = 0;
    color mask = color(88,44,141);     } except that these values
                                         can change every time!
    fill(0,100,0);
    ...
```

Parameters vs. Arguments

- When you define a function, you specify the parameters
  - Parameters are *internal* variables/boxes for functions
  - Use parameters for values that you want to be different on different calls to this function
- When you call a function, you pass arguments
  - The order of the arguments must match the order of the parameters
- We define a function once, but can call it as many times as we want (and in $\boxed{\text{different ways}}$)!

arguments

Parameters Analogy

- Executing a program is like walking down a hallway
- Calling a function is like stepping into a room
  - Step back into the hallway after you are done with the task in that room
  - Parameters are boxes (our variable analogy) bolted to floor of room – can use while you're there, but can't leave room with them
  - Arguments are the values you place in boxes when you enter the room

program execution

local declarations

initialization

Solving Problems

- Understand the problem
  - What is the problem description?
  - What is specified and what is *un*specified?
  - What has been given to you (*e.g.* starter code)?
- Break the task down into less complex subtasks

different x, same y

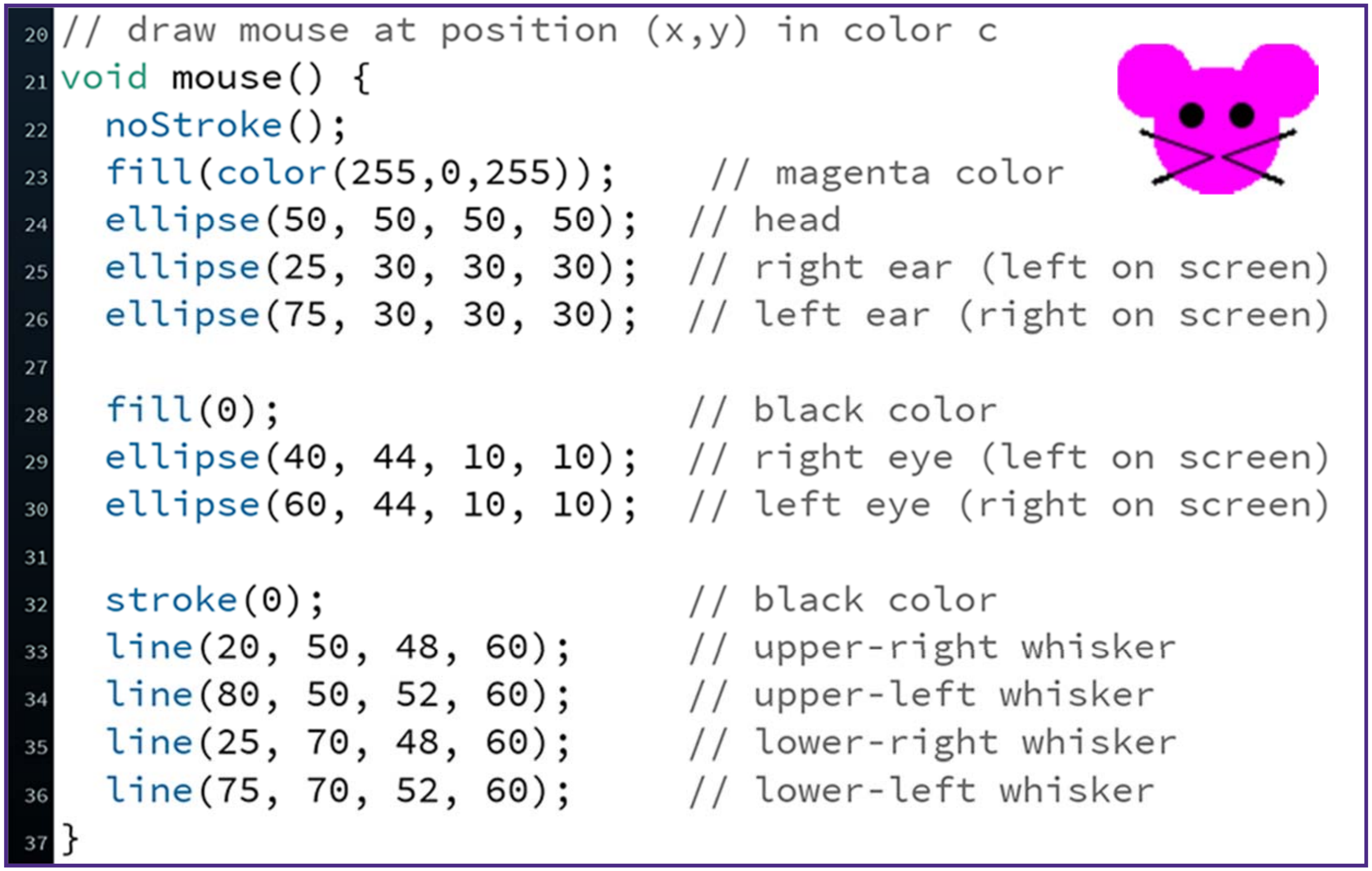
- Example: Make a function that draws a row of five mice with their ears touching/overlapping. The mice should all be the same color except for the middle one, which should be red.

different colors

main subtask: draw a mouse

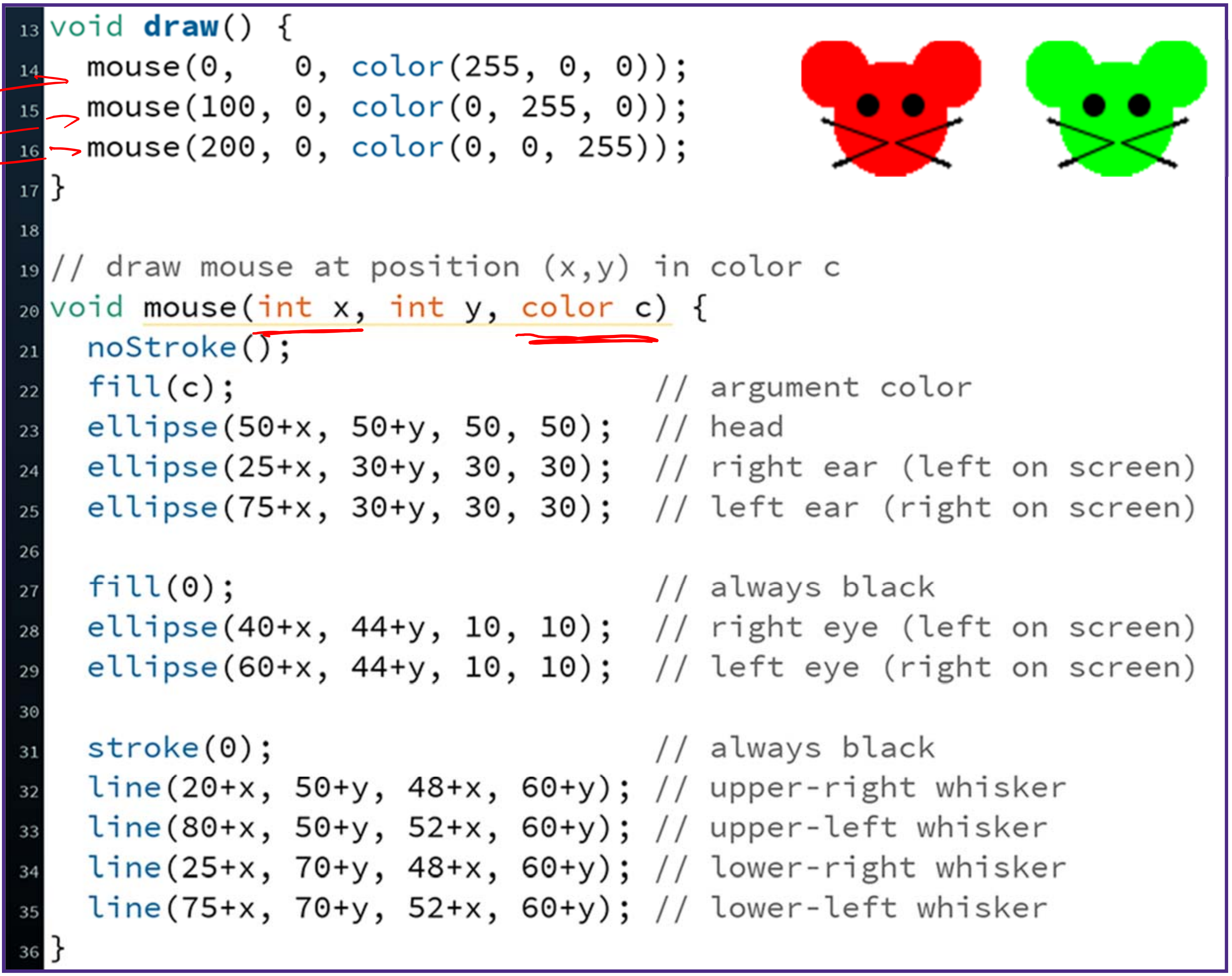
something like: void mouse(int x_pos, color c)

Parameter Example

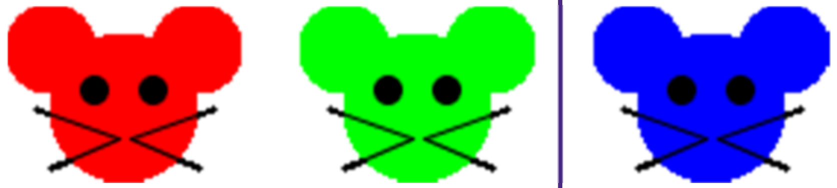


```
// draw mouse at position (x,y) in color c
void mouse() {
  noStroke();
  fill(color(255,0,255));     // magenta color
  ellipse(50, 50, 50, 50);   // head
  ellipse(25, 30, 30, 30);   // right ear (left on screen)
  ellipse(75, 30, 30, 30);   // left ear (right on screen)

  fill(0);                   // black color
  ellipse(40, 44, 10, 10);   // right eye (left on screen)
  ellipse(60, 44, 10, 10);   // left eye (right on screen)

  stroke(0);                 // black color
  line(20, 50, 48, 60);      // upper-right whisker
  line(80, 50, 52, 60);      // upper-left whisker
  line(25, 70, 48, 60);      // lower-right whisker
  line(75, 70, 52, 60);      // lower-left whisker
}
```

Parameter Example



```
void draw() {
  mouse(0,   0, color(255, 0, 0));
  mouse(100, 0, color(0, 255, 0));
  mouse(200, 0, color(0, 0, 255));
}

// draw mouse at position (x,y) in color c
void mouse(int x, int y, color c) {
  noStroke();
  fill(c);                          // argument color
  ellipse(50+x, 50+y, 50, 50);      // head
  ellipse(25+x, 30+y, 30, 30);      // right ear (left on screen)
  ellipse(75+x, 30+y, 30, 30);      // left ear (right on screen)

  fill(0);                          // always black
  ellipse(40+x, 44+y, 10, 10);      // right eye (left on screen)
  ellipse(60+x, 44+y, 10, 10);      // left eye (right on screen)

  stroke(0);                        // always black
  line(20+x, 50+y, 48+x, 60+y);    // upper-right whisker
  line(80+x, 50+y, 52+x, 60+y);    // upper-left whisker
  line(25+x, 70+y, 48+x, 60+y);    // lower-right whisker
  line(75+x, 70+y, 52+x, 60+y);    // lower-left whisker
}
```



Looking Forward

- Portfolio
  - Don’t forget to add Taijitu, Logo Design, and Lego Family!
- Animal Functions
  - Start in lab on Thursday, due Monday (1/22)
  - Design your own animal (like the mouse shown here)

Example from CSE120 Sp17 student