Functions

CF1

Gabe Johnson | September 25 & 26, 2024

attendance code:

reused

Chatbots and You

as in, Why Are You Here?

- Chatbot can help
- Chatbot is frequently full of shit
- If you use it (& that's OK) *please type everything yourself and don't copy paste*
- … this is just one of many approaches for retaining a little of Chatbot's possibly full-of-shit responses and maybe learning something arguably correct

What’s a function?

- A named block of code that has inputs (maybe) and outputs (maybe)
- We’ve used built-in functions before: draw, rect, ellipse, noStroke, etc.
- Make functions when you have an action that you’d like to name and re-use
- If the function takes parameters, the behavior can depend on them
- … e.g. a single robot function can take different geometry and color params

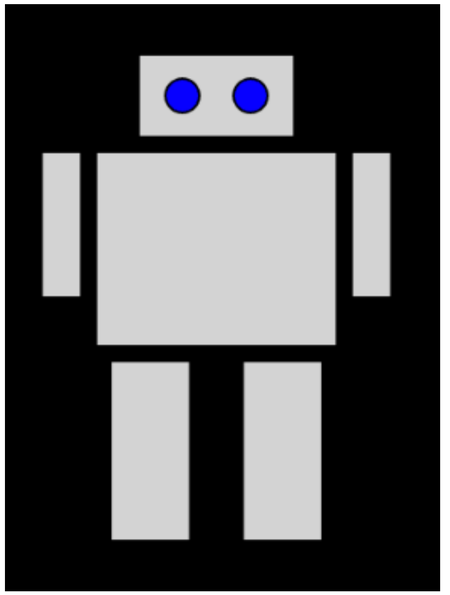
Modularity

What is the minimum information we need to do something?

- In the videos, he uses the term “modularity”
- I tend to think of this as “behavior”. E.g. bounce, move, display.
- You can think of a function as a verb. It performs some action.
  - It might modify the world, e.g. putting pixels on the screen
  - It might compute something, e.g. add(2, 4) could *return* 6

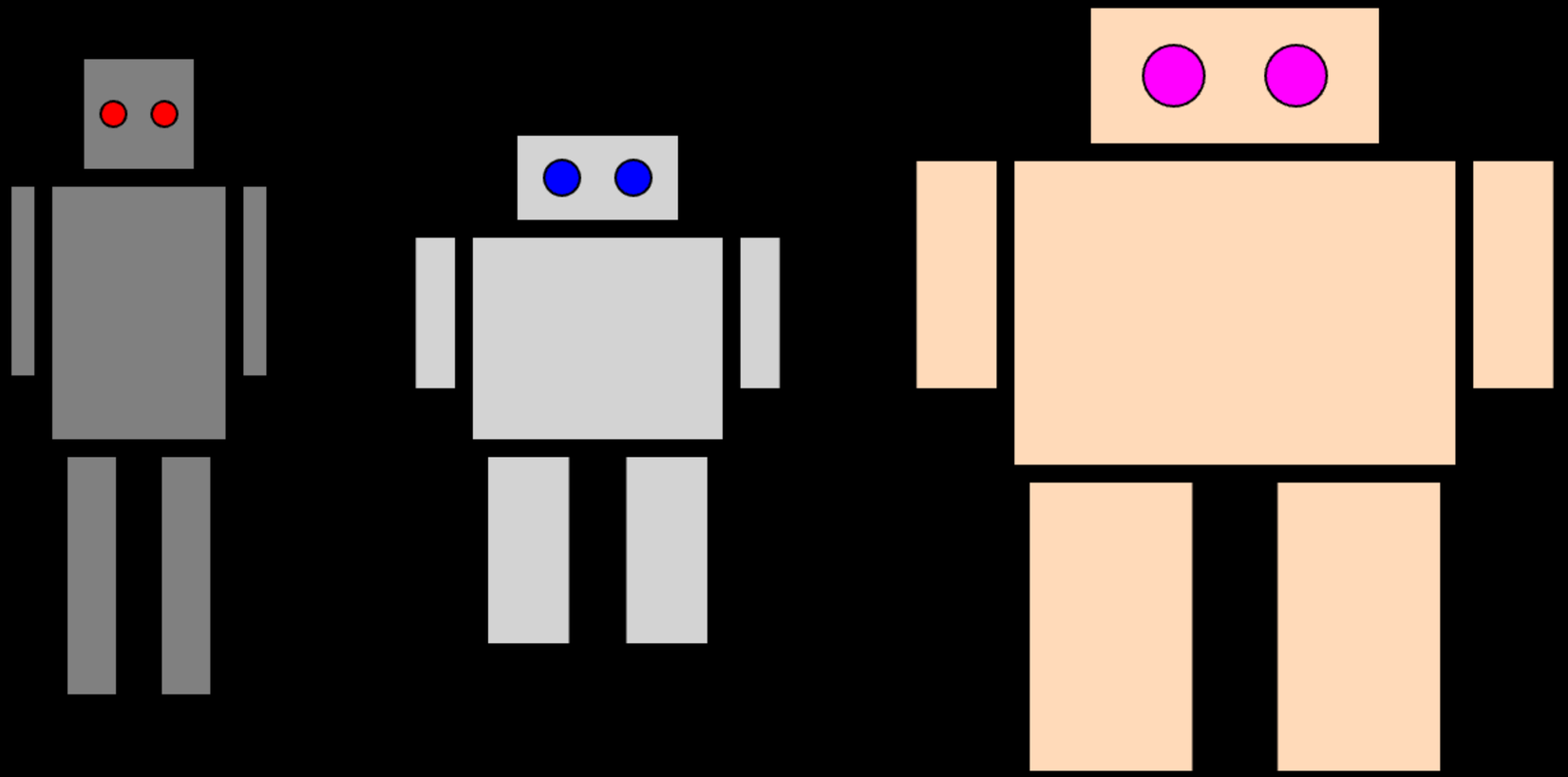
Calling functions

- Simply provide the name of the function, and then arguments in parentheses
- You've done this a zillion times: rect(10, 30, 200, 70)
- robot(200, 80, 140, 200, "blue", "lightgray");
- Use different arguments to get different results without having to repeat your code a zillion times



```
function setup() {
  createCanvas(700, 400);
}

function draw() {
  background(0);
  robot(40, 50, 100, 250, "red", "gray");
  robot(200, 80, 140, 200, "blue", "lightgray");
  robot(400, 30, 240, 300, "magenta", "peachpuff");
}

function robot(x, y, w, h, eyeColor, bodyColor) {
  // intentionally long passage of code to demonstrate why
  // it is helpful to pack stuff off into functions.
  // this one takes parameters - the x/y position, robot
  // width and height, and colors for eye and body
  const centerX = w / 2;
  const eyeRadius = w * 0.1;
  const eyeOffset = w * 0.1;
  const leftEyeX = centerX - eyeOffset;
  const rightEyeX = centerX + eyeOffset;
  const armGap = 6;
  const headWidth = 0.5 * w - 6;
  const headHeight = 0.2 * h - 6;
  const headLeft = centerX - headWidth / 2;
  const eyeCenterY = headHeight / 2;
  const armWidth = 0.16 * w - 6;
  const armHeight = 0.3 * h;
  const bodyWidth = 0.75 * w - 6;
  const bodyHeight = 0.4 * h;
  const bodyLeft = centerX - bodyWidth / 2;
  const bodyRight = bodyLeft + bodyWidth;
  const bodyTop = headHeight + armGap;
  const bodyBottom = bodyTop + bodyHeight;
  const leftArmLeft = bodyLeft - armGap - armWidth;
  const rightArmLeft = bodyRight + armGap;
  const legHeight = h - (bodyBottom + armGap);
  const legWidth = armWidth * 2;
  const legTop = bodyBottom + armGap;
  const leftLegLeft = centerX - bodyWidth / 2 + armGap;
  const rightLegRight = centerX + bodyWidth / 2 - armGap;
  const rightLegLeft = rightLegRight - legWidth;

  push();
  translate(x, y);
  // head using body color
  fill(bodyColor);
  rect(headLeft, 0, headWidth, headHeight);
  // eyes, filled with eye color
  fill(eyeColor);
  circle(leftEyeX, eyeCenterY, eyeRadius);
  circle(rightEyeX, eyeCenterY, eyeRadius);
  // fill using body color again
  fill(bodyColor);
  // left arm
  rect(leftArmLeft, 0.2 * h, armWidth, armHeight);
  // right arm
  rect(rightArmLeft, 0.2 * h, armWidth, armHeight);
  // body
  rect(bodyLeft, bodyTop, bodyWidth, bodyHeight);
  // left leg
  rect(leftLegLeft, legTop, legWidth, legHeight);
  // right leg
  rect(rightLegLeft, legTop, legWidth, legHeight);
  pop();
}

```

One function made all these robots by using different arguments for position, size, and colors.



https://editor.p5js.org/johnsogg/sketches/NL0J0VIv_

Function arguments

Arguments & Parameters

- When you *call* a function, the values you send in are called *arguments*
- When you *define* a function, the values you receive are called *parameters*

- If a function has no arguments, it still needs the parentheses, like:
  - (when invoking) doSomething()
  - (when defining) function doSomething() { }
-

Defining functions

```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

Defining functions

Start with ‘function’ keyword

```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

Defining functions

Then name give it a descriptive name like ‘robot’

```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

Defining functions

After the function name, always include opening and closing parentheses

```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

Defining functions


```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

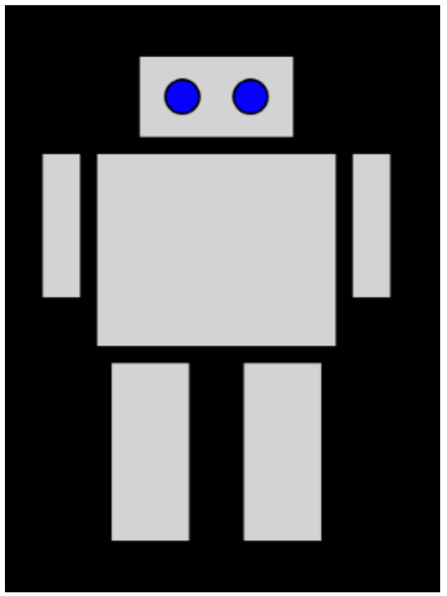
Defining functions

Then use curly braces to define the body of the function.

```
function robot(x, y, w, h, eyeColor, bodyColor) {
  // do what you need using the parameters
  // listed above
}
```

Calling functions

- Your function can now be invoked, called, executed
- (all those words mean the same thing)
- `robot(200, 80, 140, 200, "blue", “lightgray”);`



Return values

This is super useful

- In p5.js you often see functions that simply do work, like drawing robots
- But you can always define functions that compute some data:

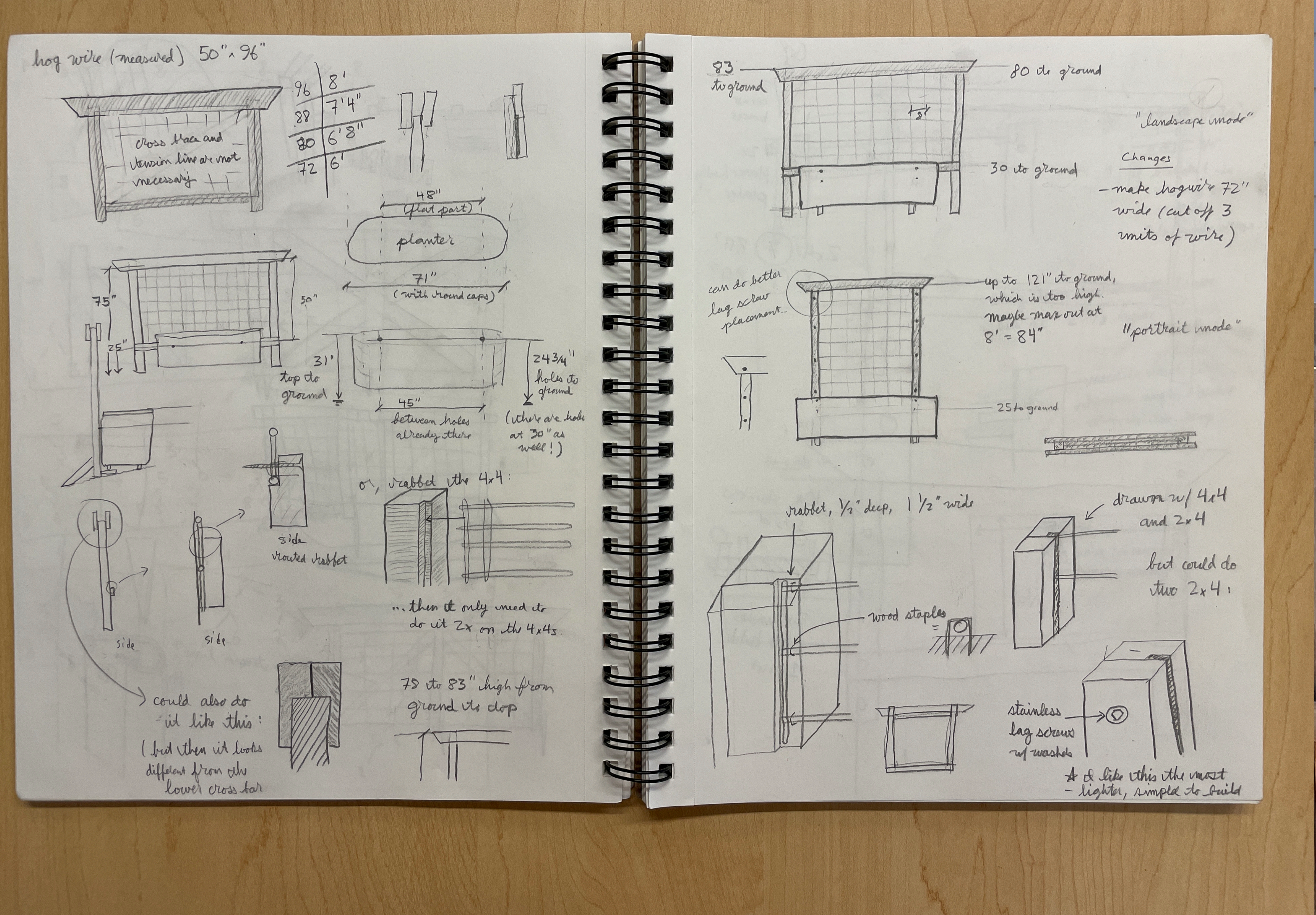
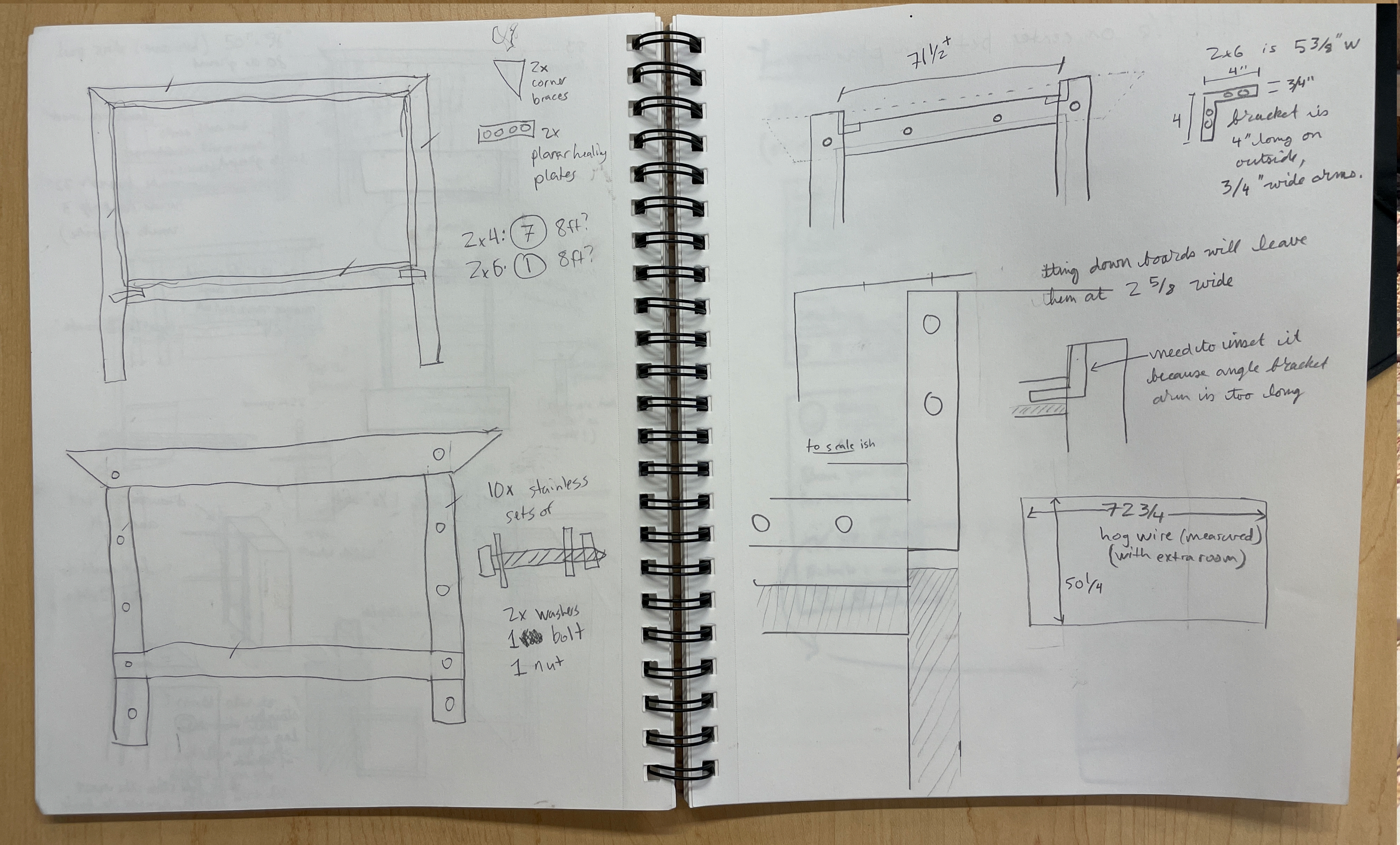
```
function numberCubed(x) {
    return x * x * x;
}
```

- and then use it:

```
const v = numberCubed(5); // v will be 125
```

Use Functions Smartly

- I tend to sketch out what I want to help me spot patterns, points of interest, and things that I wouldn't "see" when braining my way through it
- You can decompose problems into subproblems, and those are good candidates for turning into functions
- This works beyond software - if a function is just a process and the parameters are just knobs, you can think of building physical things as just executing a "program" built out of "functions"


hog wire (measured) 50" x 96"
cross brace and vertical lines are not necessary
planter
71" (with round caps)
top to ground
between holes
could also do it like this
79 to 83" high from ground to drop
83 to ground
80 to ground
30 to ground
"landscape mode"
Changes
make hog wire 72" wide (cut off 3 units of wire)
"portrait mode"
25 to ground
rabbet, 1/2" deep, 1 1/2" wide
wood staples
drawn w/ 4x4 and 2x4
but could do two 2x4
stainless lag screw w/ washer
I like this the most - lighter, simpler to build

2x corner braces
2x planter holding plates
10x stainless sets of
2x washers
1 nut
71 1/2"
2x6 is 5 3/8" W
3/4"
bracket is 4" long on outside, 3/4" wide arms.
to scale ish
hog wire (measured) (with extra room)
72 3/4
50 1/4


Homework

If we have time, live coding to give a head start

- Re-purpose homework from last week (or before)
- Identify spots where you could modularize your code with functions
- Anyone have a good candidate sketch to work out in class?
- If so, email me - gabriel.johnson@colorado.edu