

## More debugging tips

---

 [Share](#)  [Tweet](#)  [Email](#)

There are many ways to debug your programs! Here's a list to get you started:

### Print debugging

As we just showed, you can insert `print()`s or `println()`s into your code to help you figure out which code is being called and with what values. Both of these functions output values in a console that pops up over the canvas. You can also use `debug()` to send the output to your browser's JavaScript console, if you know how to use that.

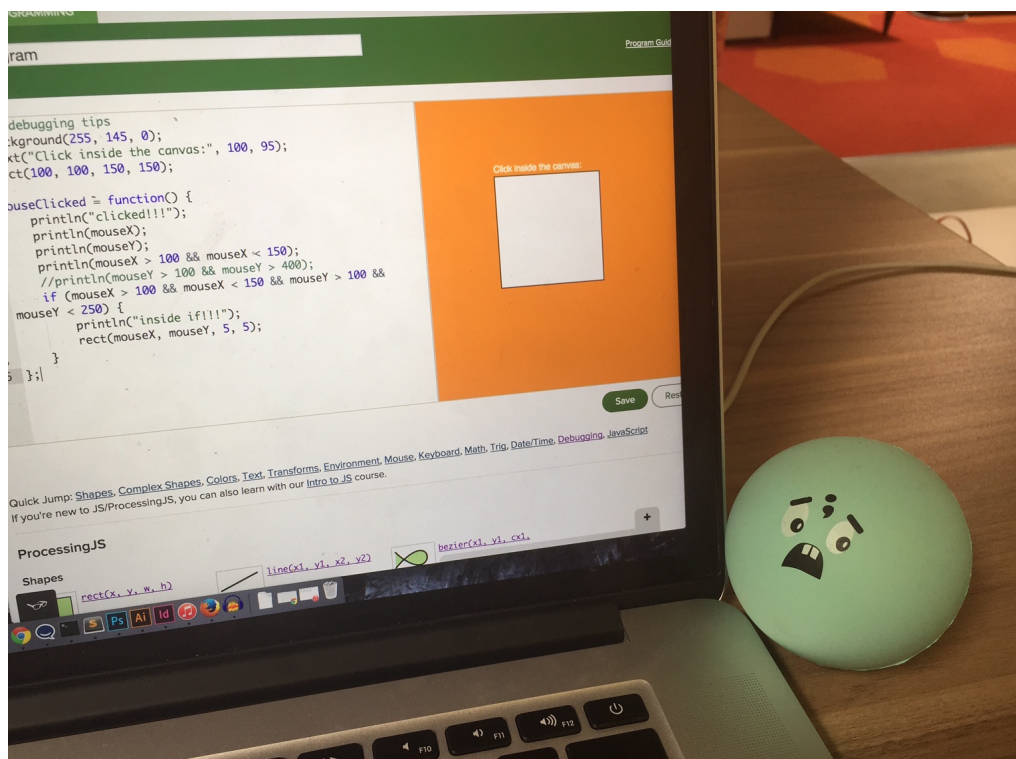
A screenshot of a browser console window. The console is dark-themed and shows several lines of text output. The text is: "inside if!!!", "clicked!!!", "188", "135", "false", and "false". There is a small grey button with a close symbol (an 'x') in the top right corner of the console window.

```
inside if!!!
clicked!!!
188
135
false
false
```

### Rubber-duck debugging

Sit a rubber duck next to your computer—or whatever duck-like object you have handy—and explain your program and problem to it, line by line. Many programmers find that just the process of putting the problem in words helps their brains realize what's wrong. You can also ask a friend or teacher to be your rubber duck, sitting and listening to you explain it. Sometimes they might even think of a solution for you, but regardless, they're doing you a service by just listening to the explanation.

Here's the closest thing we have to a rubber duck in the Khan Academy office:




## Exaggerate your output

Since you're making programs in ProcessingJS, you're dealing with lots of fill colors and strokes. When I'm not

seeing the visual output I expect, sometimes it helps to use really big or extreme values for the fill and strokes—like `strokeWeight(30)`. Since our environment is realtime and includes the number scrubbers, it's really easy to change up the numbers in your program to see what effect the change has on the output. For example, it might help you figure out where a missing shape went.

```
1 // hard to see, can you see it?  
2 point(30, 30);  
3  
4 // make it obvious with a big stroke!  
5 strokeWeight(30);  
6 point(10, 10);  
7
```



It's a good idea to get comfortable with all your options for debugging programs so that you can use whichever one works the best in a particular situation.