Debugging & Good Coding Practice

CF1

Gabe Johnson | October 31 2024 🎃

Good Coding Practices 🤣

- I’ve been doing this for 30+ years
- … and still learning!
- But I can share my experience and perspective to maybe help you all

Two main topics

Bugging and Debugging

- Are you writing code? Congrats! 🥳 Your code has bugs!
- How will you…
  - Write code?
  - Find problems?
  - Fix problems?
  - Characterize problems so as to (maybe) avoid it next time?

You Don’t Know What You’re Building
(and that’s OK)

- Average programmer: knows what they’re building
- Good programmer: that ☝ plus awareness of challenges, some unknowns
- Excellent programmer: those ☝☝ plus the moxie to start anyway

- See also:
  - Bounded rationality [Herb Simon]
  - Plans and Situated Actions [Lucy Suchman]

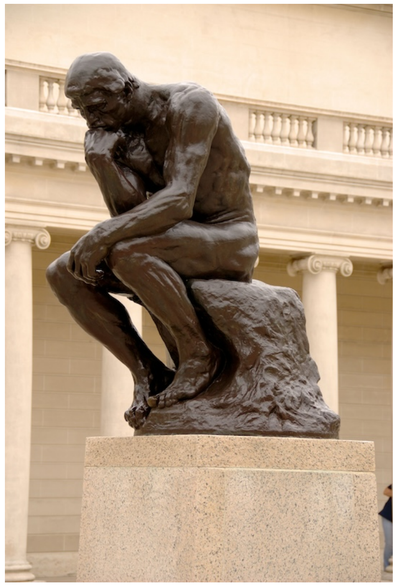
You Don’t Know What You’re Building
(and that’s OK)

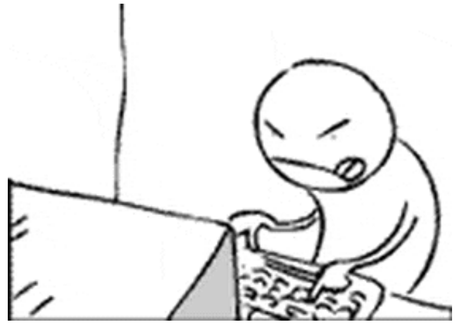
- Since you can’t know everything, it can be paralyzing
- So: develop tactics & strategies to frame problems as they come up
- … and get going!
- The more you get going, the smarter you are about what happened, the better your tactics & strategies will get.

Good Practices

Thinking & Doing

What’s apt right now depends on you and the context





Thinking

Often helpful, but usually not

“Thinking”

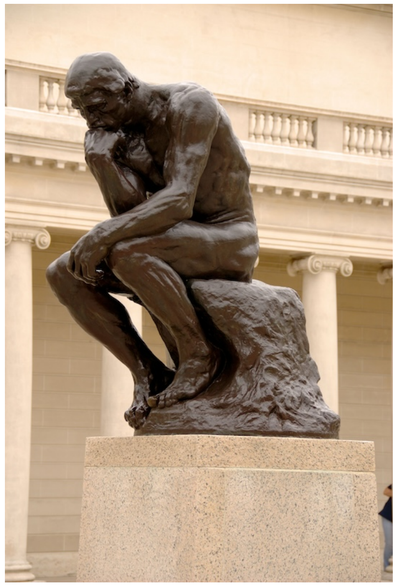
Planning, diagramming, collaborating, talking with stakeholders on their turf

Action

Don’t be afraid, Control Z is a thing

Thinking & Doing

What’s apt right now depends on you and the context



Thinking

Often helpful, but usually not

“Thinking”

Planning, diagramming, collaborating, talking with stakeholders on their turf

Action

Don’t be afraid, Control Z is a thing

Problem: Don’t know what to draw

… draw toilets!

… you will **very quickly** figure out what else to draw

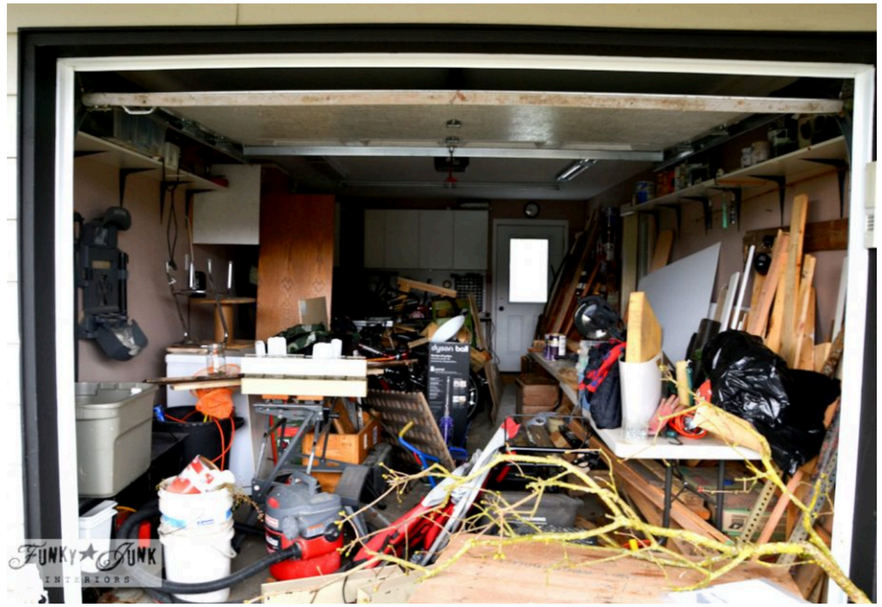
Problem: Don’t know where to get lunch

… suggest McDonalds!

… you will **very quickly** figure out where to go for real

Problem: Can’t find anything in garage

… just start cleaning and organizing



… soon it will be good enough and you can get on with life

Thinking vs “Thinking”

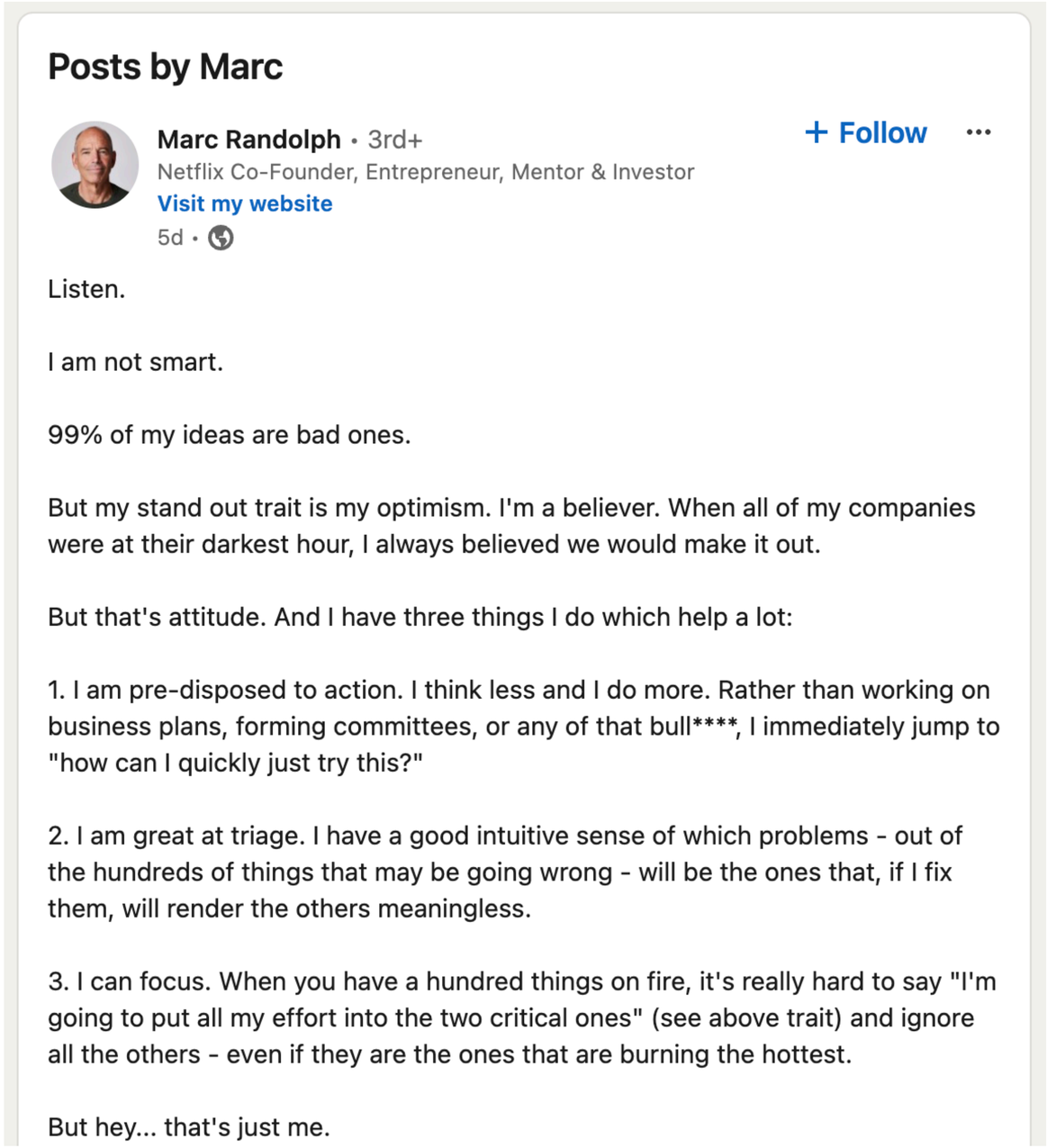
I could call it “externalizing”

- Staring at a wall and having deep thoughts can work
- Taking a walk and clearing your head can work

- But usually a more active form of thinking helps - make sketches on paper or whiteboards, talk it out, do cheap ‘what if’ scenarios

Don’t Think?

- E.g. Very Serious People™ on LinkedIn exhorting you to stop thinking and start doing.
- But is this really binary? Thinking bad, action good?
- What do you think?



Posts by Marc

Marc Randolph • 3rd+
Netflix Co-Founder, Entrepreneur, Mentor & Investor
Visit my website
5d •

+ Follow

Listen.

I am not smart.

99% of my ideas are bad ones.

But my stand out trait is my optimism. I'm a believer. When all of my companies were at their darkest hour, I always believed we would make it out.

But that's attitude. And I have three things I do which help a lot:

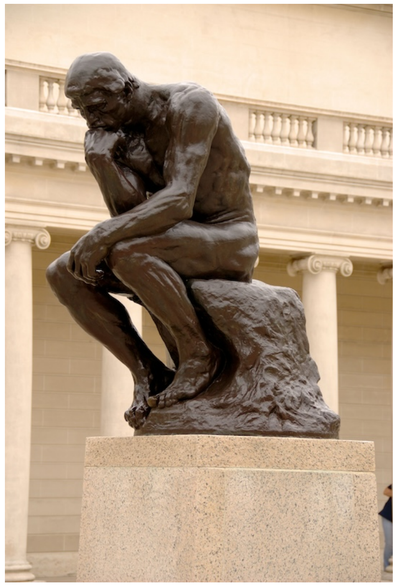
1. I am pre-disposed to action. I think less and I do more. Rather than working on business plans, forming committees, or any of that bull****, I immediately jump to "how can I quickly just try this?"

2. I am great at triage. I have a good intuitive sense of which problems - out of the hundreds of things that may be going wrong - will be the ones that, if I fix them, will render the others meaningless.

3. I can focus. When you have a hundred things on fire, it's really hard to say "I'm going to put all my effort into the two critical ones" (see above trait) and ignore all the others - even if they are the ones that are burning the hottest.

But hey... that's just me.

It is all about what will help the most right meow



Thinking

“Thinking”

Action

The Perfect is the Enemy of the Good

Which of these would you choose?

- **A**: perfectly built, highly optimized thing that solves a problem nobody has
- **B**: mostly functional, mostly usable thing that solves a problem that causes you great pain

Yes this is a strawman, but option A is so fun to build!

The Perfect is the Enemy of the Good

Understand the problem and who has it first

- Who has the problem?
- Why is it a problem?
- What are some ways to address it?
- Can we quickly build a prototype to see how off we are?
- Yes, you can polish and perfect *later* but don’t start there.

Three Virtues

Of a great programmer

- **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

Three Virtues

Of a great programmer

- **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.

Three Virtues

Of a great programmer

- **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

Three Virtues

Of a great programmer

- **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

- **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.

- **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

Debugging

Death, Taxes, and Software Bugs

Two constants of the universe

- All code has bugs. Refute *this*, Karl Popper!
- But… code is the way we solve problems? Guh, this is annoying

- You don’t know what you’re building (with 100% clarity)
- You’re human and make mistakes, overlook stuff
- You hold wrong assumptions, or assumptions that become wrong over time
- You might have to work with Gabe, and then deal with *his* bugs as well

Characterize Bugs When You Find Them

This is an important skill

Error Messages

They usually suck

- **Write-time** errors: usually syntax errors, but can also be ‘semantic’ if you have a linter (which you should)
- **Build-time** errors: usually ‘plumbing’ errors (components of the system that don’t line up), found if you have a build system (which you should)
- **Run-time** errors: data-driven? Faulty business logic? Network conditions? Could be anything! Find these when people complain, or if you do logging (which you should)
- **User reports** from the field: Using human language on social media with expletives not deleted. Find them if you care to listen (which you should)

Write-time error

```
32  for (let i = 0 i < colors.length; i++) {
33    const initialX = random(0, width);
34    const initialY = random(0, height);
35    const radius = random(10, 60);
36    boids.push(new Boid(initialX, initialY, radius, colors[i]));
37  }
```

Console

SyntaxError: Unexpected identifier 'i'

What's the actual problem?

Write-time error

```
32  for (let i = 0 i < colors.length; i++) {
33    const initialX = random(0, width);
34    const initialY = random(0, height);
35    const radius = random(10, 60);
36    boids.push(new Boid(initialX, initialY, radius, colors[i]));
37  }
```

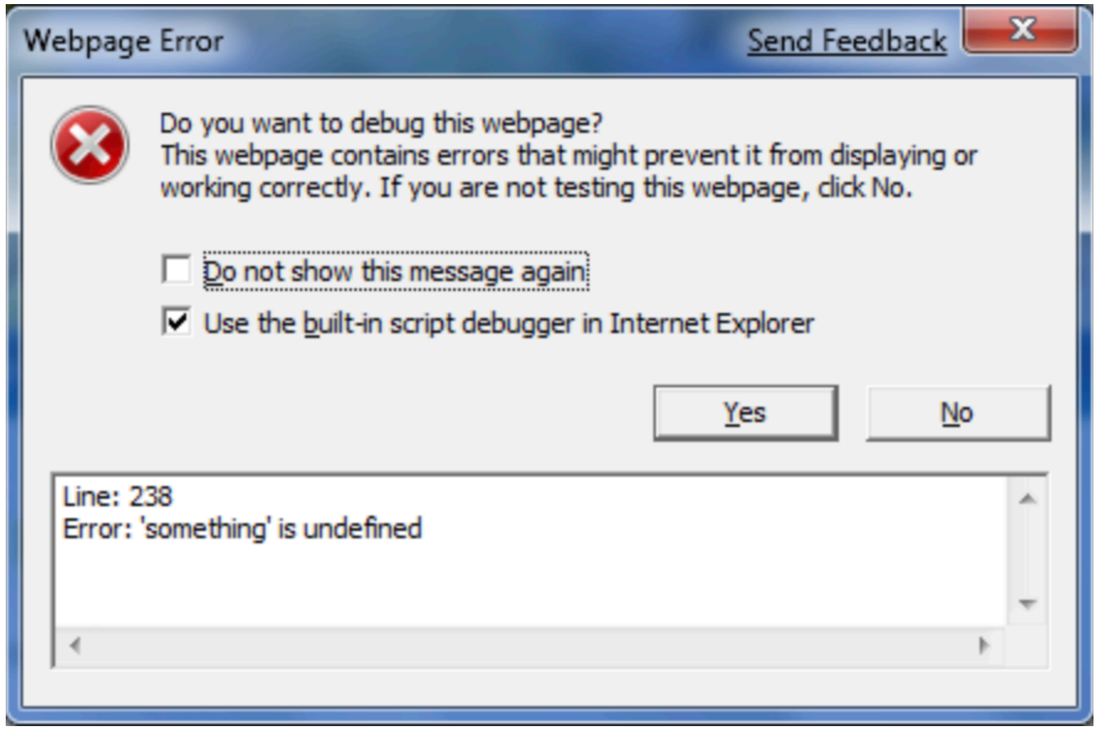
Console

SyntaxError: Unexpected identifier 'i'

Message is about the identifier, but the problem is actually a missing semicolon.

Runtime Error

You’re lucky to get a dialog box



Hey, y’all!

Something is undefined! 😱

Copy/Paste Errors into Search

- If you get an error that you don’t recognize, don’t understand quickly…
- Google it!
- Or DDG
- Or Stack Overflow
- Or Chatbot
- **This should be your first line of defense after your own brain**

Rubber Ducky

Because it is crazy to talk to yourself?

Rubber Ducky

- It forces you to slow down
- and engage the brain
- and look at every line, every statement, every function call
- makes you question your understanding of the code

- Externalizing your thinking is generally an amazingly powerful thing to do beyond programming.

Print Statements

Cheap, quick, often effective. Quickly becomes noisy though.

- In JS, `console.log(“value is:”, val)`
  - The browser console is really quite nice.
  - View -> Developer -> JavaScript Console
- Every language has some form of print statement
- Upside is that you can usually see what’s going on quickly
- Downside is that it can be very chatty
- So clean up the print statements after they’ve served their purpose

Log Statements

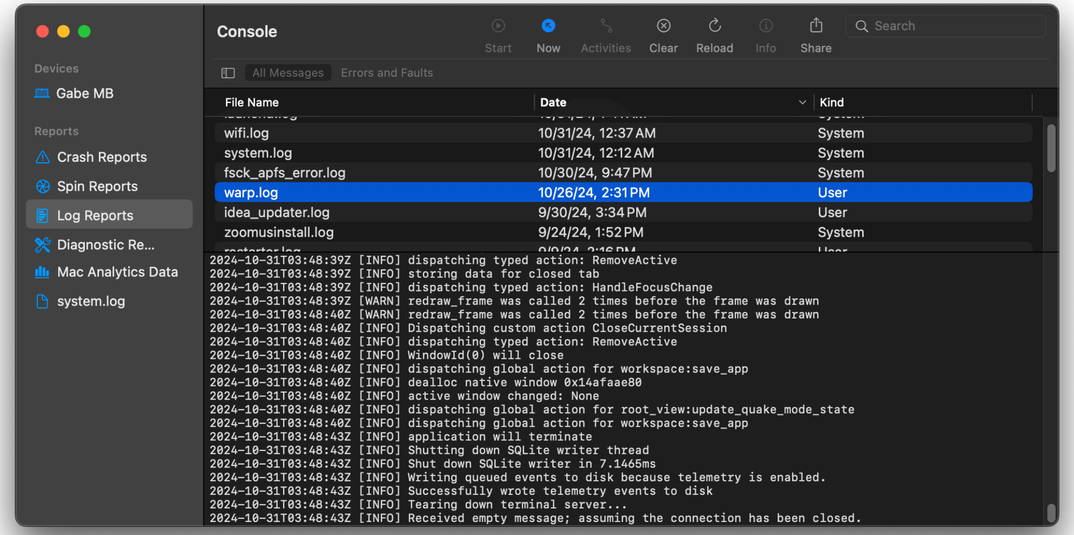
Like print statements but more official

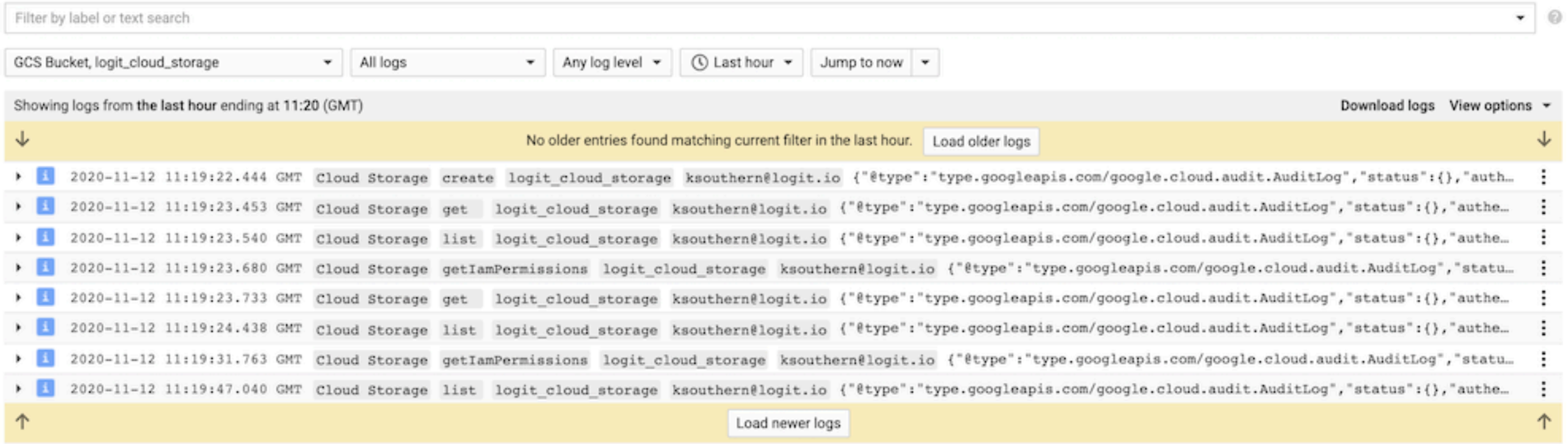
- Logs record what happened so you can:
  - Get a sense of what's going on in real time
  - Do forensics of problems after the fact
- They are often stored to a file, often on a server somewhere. Persistent!
- They don't add noise to the console, so you're still free to use print statements for quick debugging

Log Statements

Example logs on macOS and Google Cloud

Searchable, persistent, timestamped. Can be hard to use and understand but can also totally save your skin.

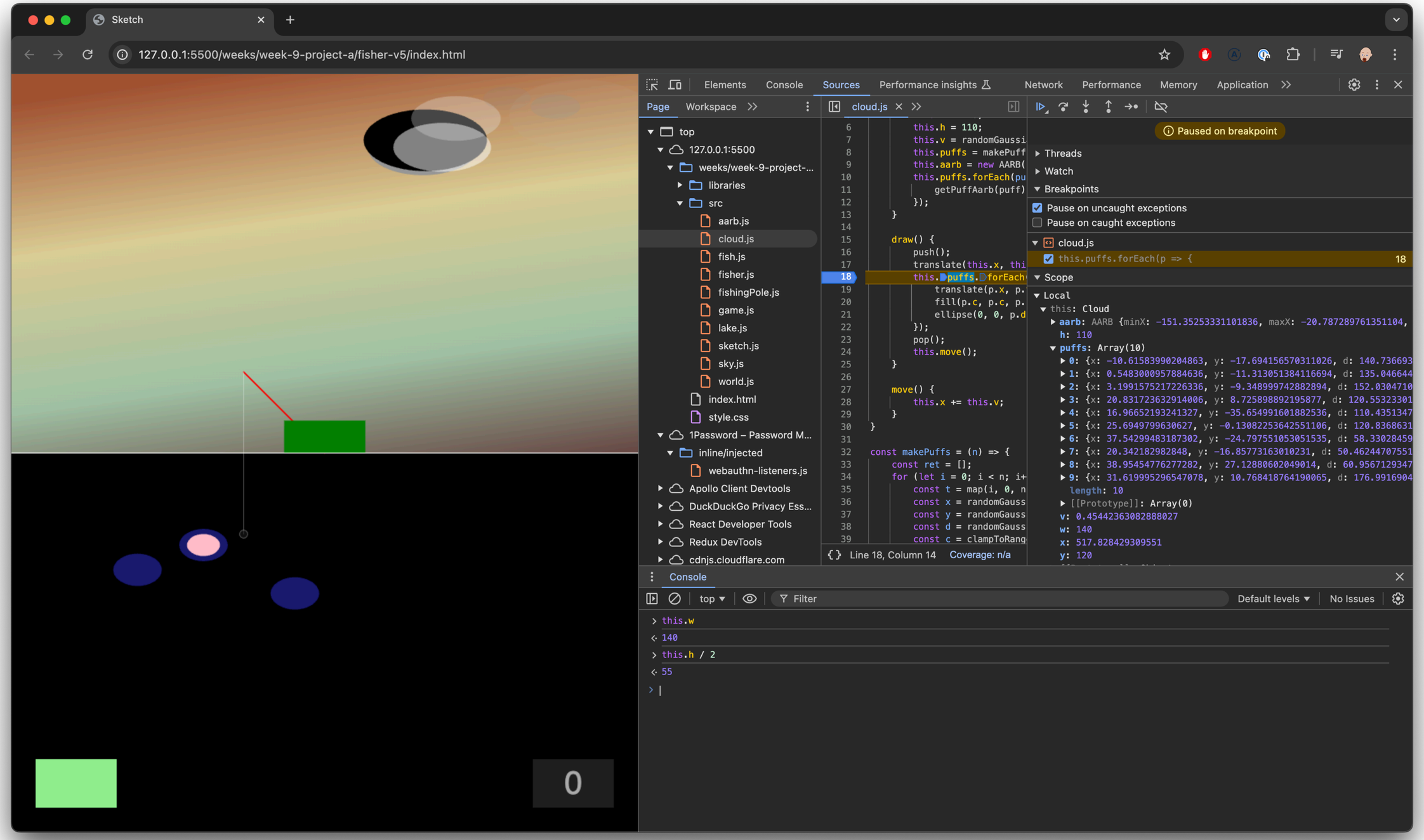




Proper Debugger

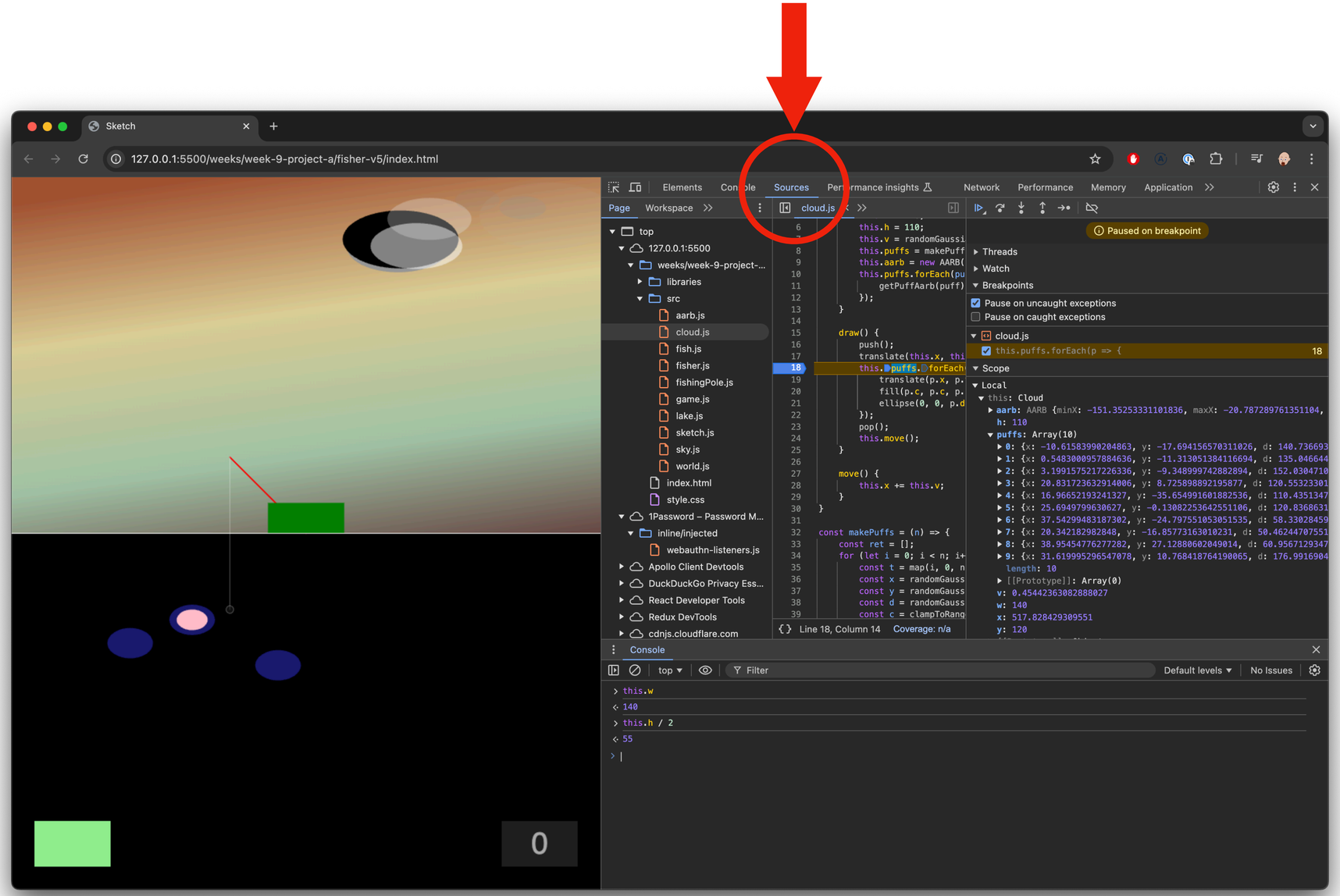
This is where it’s at

- Most languages have a ‘real’ debugger, and they’re usually similar
- It instruments your code so you can pause, step through it, and inspect program state
- Often (unfortunately) tricky to set up, but almost always worth it

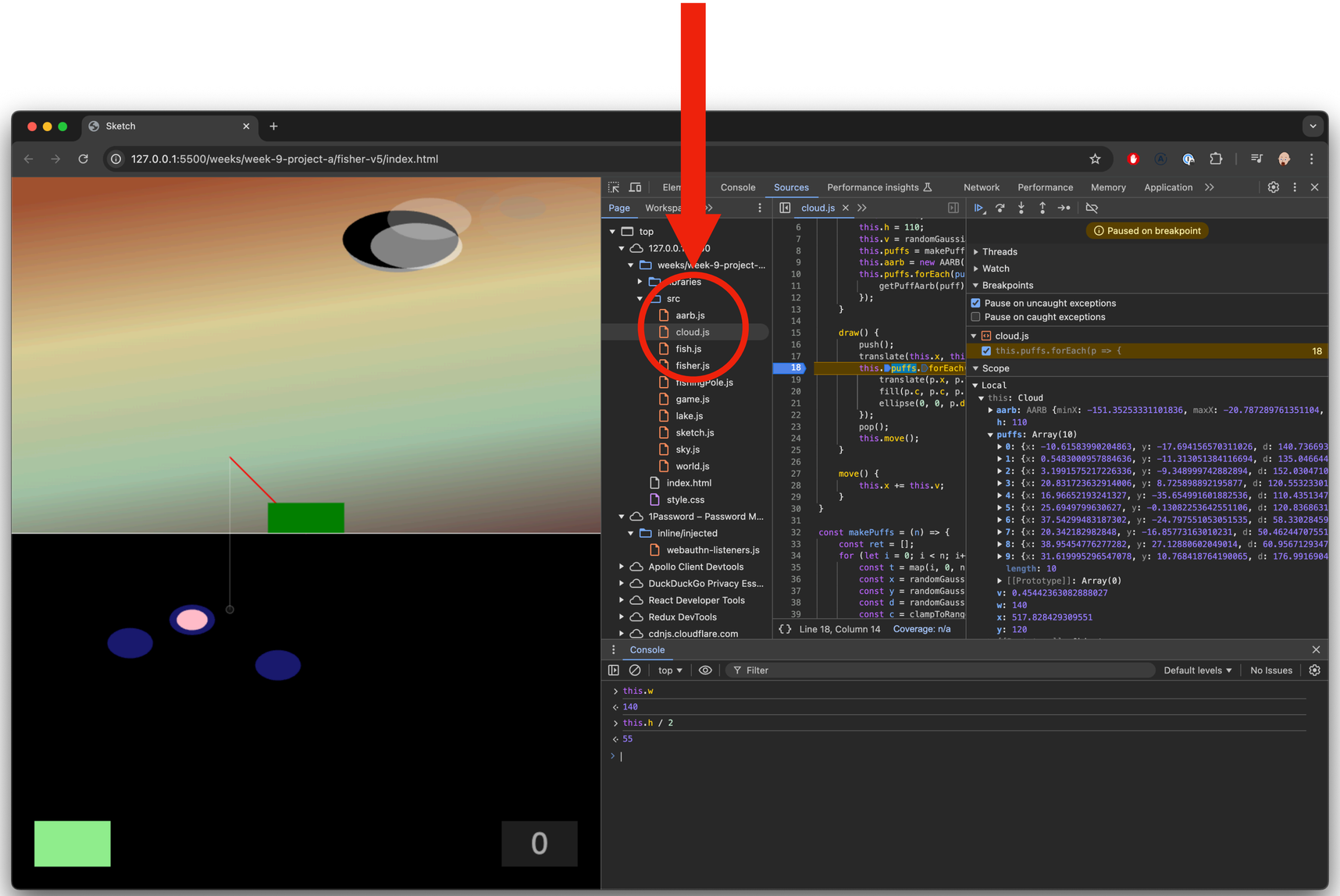

Sketch
127.0.0.1:5500/weeks/week-9-project-a/fisher-v5/index.html
0
Elements
Console
Sources
Performance insights
Network
Performance
Memory
Application
Page
Workspace
cloud.js
top
127.0.0.1:5500
weeks/week-9-project-...
libraries
src
aarb.js
cloud.js
fish.js
fisher.js
fishingPole.js
game.js
lake.js
sketch.js
sky.js
world.js
index.html
style.css
1Password – Password M...
inline/injected
webauthn-listeners.js
Apollo Client Devtools
DuckDuckGo Privacy Ess...
React Developer Tools
Redux DevTools
cdnjs.cloudflare.com
this.h = 110;
this.v = randomGaussi
this.puffs = makePuff
this.aarb = new AARB(
this.puffs.forEach(pu
getPuffAarb(puff)
});
}
draw() {
push();
translate(this.x, thi
this.puffs.forEach
translate(p.x, p.
fill(p.c, p.c, p.
ellipse(0, 0, p.d
});
pop();
this.move();
}
move() {
this.x += this.v;
}
}
const makePuffs = (n) => {
const ret = [];
for (let i = 0; i < n; i+
const t = map(i, 0, n
const x = randomGauss
const y = randomGauss
const d = randomGauss
const c = clampToRang
Line 18, Column 14
Coverage: n/a
Paused on breakpoint
Threads
Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
cloud.js
this.puffs.forEach(p => {
18
Scope
Local
this: Cloud
aarb: AARB {minX: -151.35253331101836, maxX: -20.787289761351104,
h: 110
puffs: Array(10)
0: {x: -10.61583990204863, y: -17.694156570311026, d: 140.736693
1: {x: 0.5483000957884636, y: -11.313051384116694, d: 135.046644
2: {x: 3.1991575217226336, y: -9.348999742882894, d: 152.0304710
3: {x: 20.831723632914006, y: 8.725898892195877, d: 120.55323301
4: {x: 16.96652193241327, y: -35.654991601882536, d: 110.4351347
5: {x: 25.6949799630627, y: -0.13082253642551106, d: 120.8368631
6: {x: 37.54299483187302, y: -24.797551053051535, d: 58.33028459
7: {x: 20.342182982848, y: -16.85773163010231, d: 50.46244707551
8: {x: 38.95454776277282, y: 27.12880602049014, d: 60.9567129347
9: {x: 31.619995296547078, y: 10.768418764190065, d: 176.9916904
length: 10
[[Prototype]]: Array(0)
v: 0.45442363082888027
w: 140
x: 517.828429309551
y: 120
Console
top
Filter
Default levels
No Issues
this.w
140
this.h / 2
55

This is an example of debugging a p5.js web app that is hosted locally. Process varies by language.

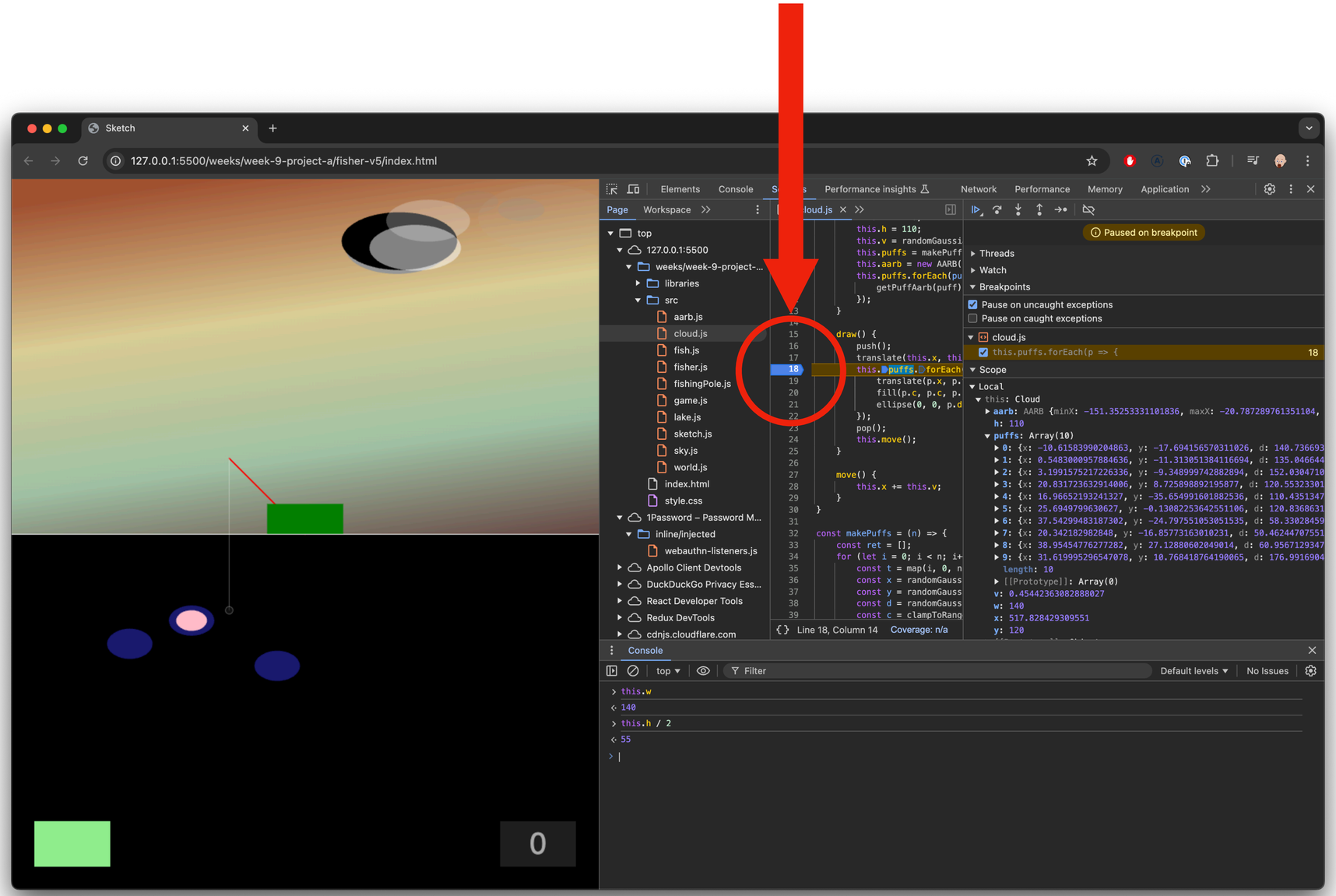
Open the developer console and go to the Sources tab.



Find the file where you want to pause execution on the left side. Click on it and the source code appears.

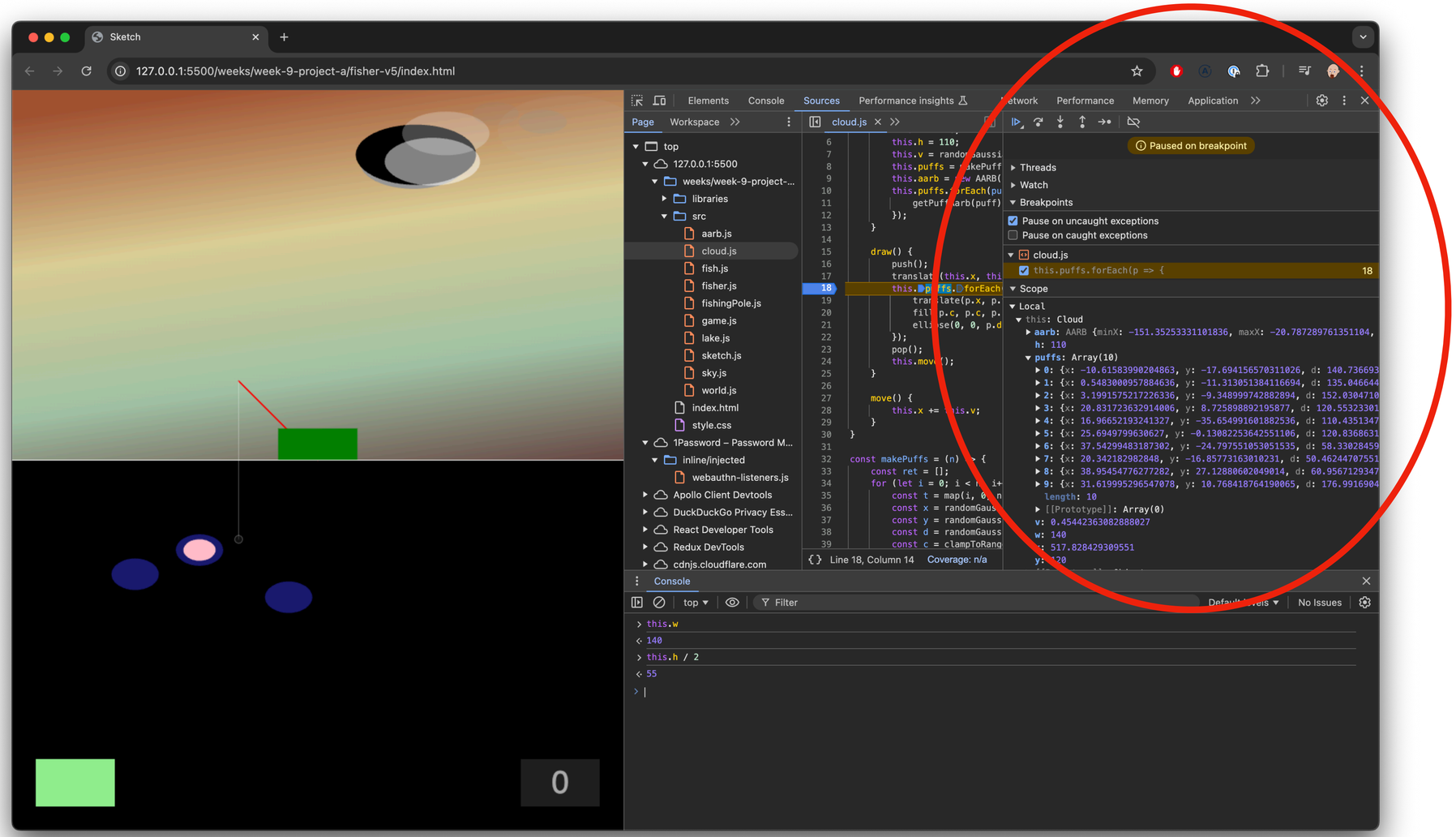


Scroll to the spot in the file where you want to pause. Click the line number on the left to insert a break point.

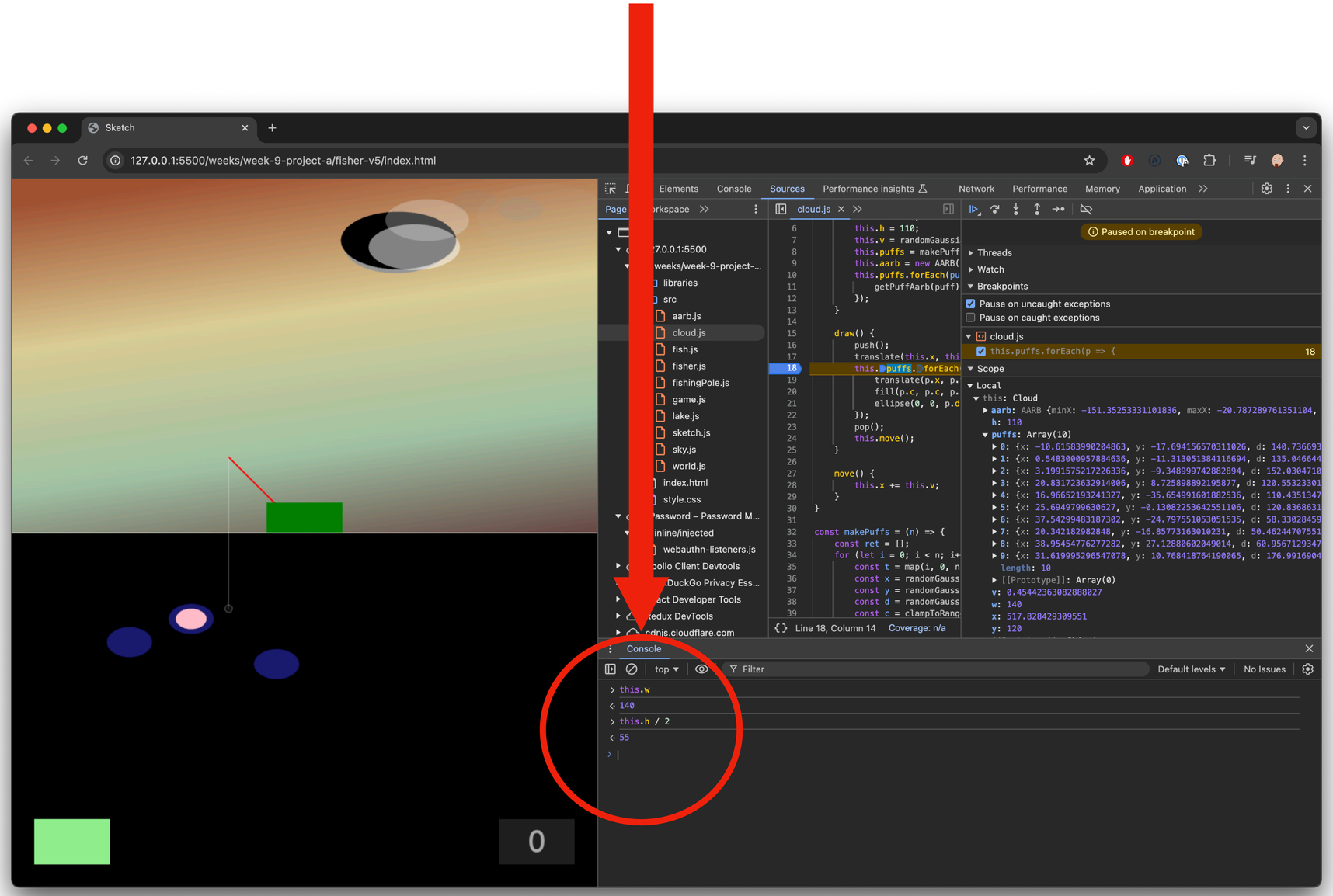


When the program runs it will stop on that line.

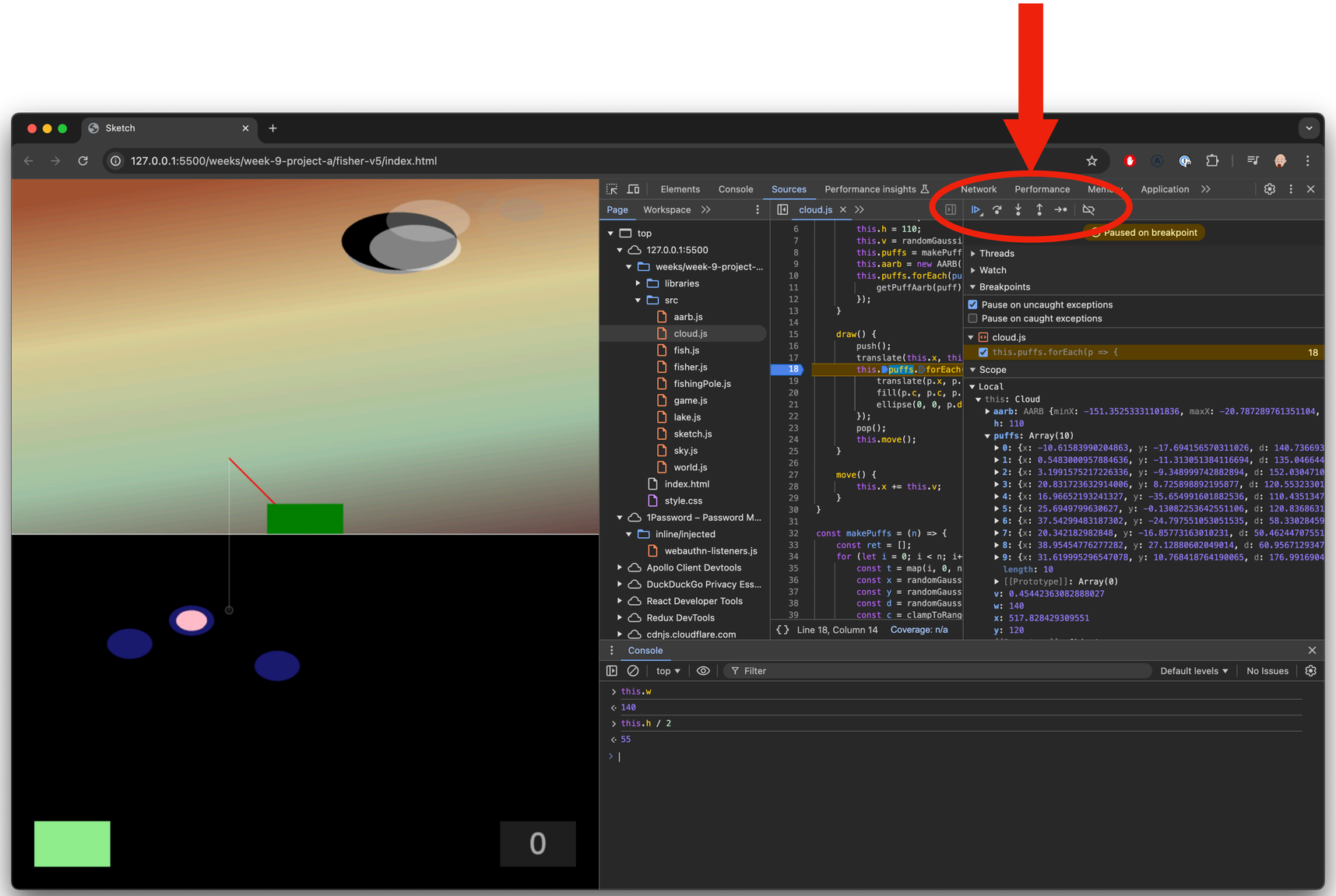
It will show you controls on the right, along with interactive data representing the program's state.



The console at the bottom also lets you type interactively to have a little conversation. You can type in symbol names to get their values, or do math with them.

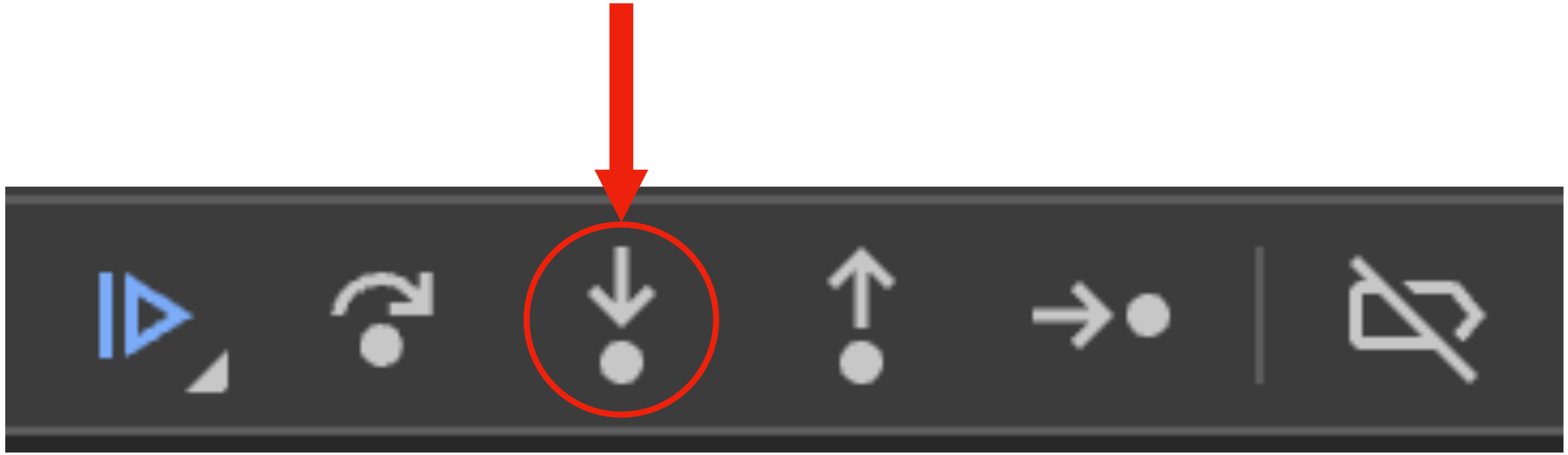


The controls are especially important for inching your way through the program.

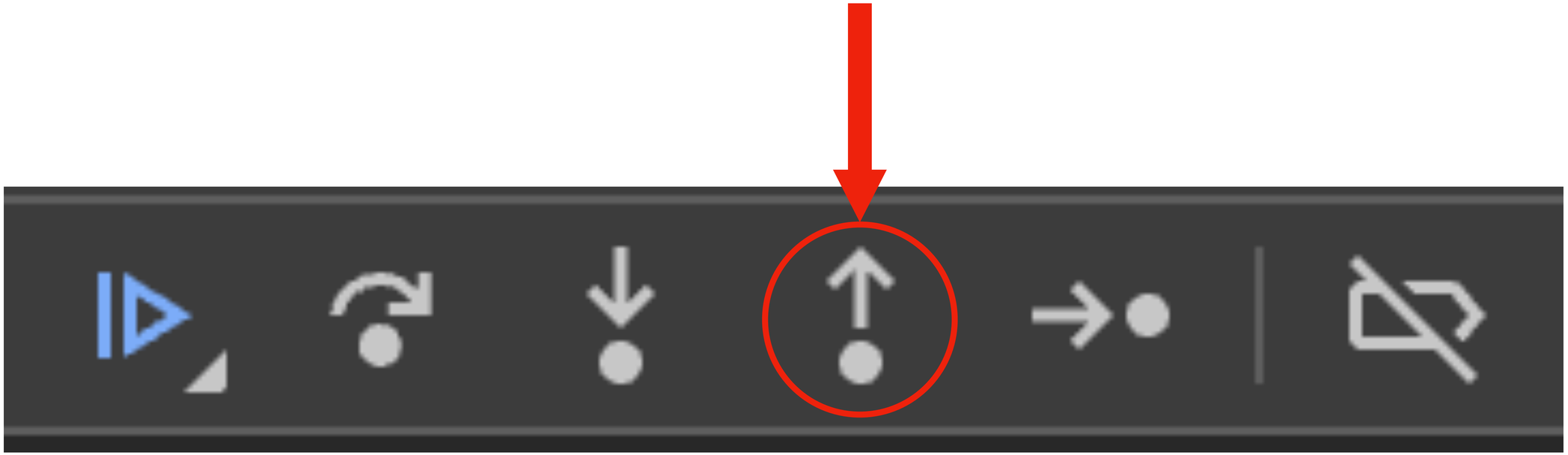


The play/pause button does what it says. I usually let a breakpoint do the pausing, and this button is for resuming playback.

The ‘step over’ button will move the program forward by a single step - a line or an expression. It will step *over* any function calls, so this essentially keeps you in the same file.



The ‘step into’ button will dive into a function call, so you might change locations within the file, or go to a different file entirely.



The ‘step out’ button will play the remainder of whatever function it is in, pop out to wherever that function was called from, and pause again.

The ‘step’ function moves forward by the smallest possible increment, including going into a function call.

This will add or remove breakpoints from the current line. I don’t think I’ve ever used this or the ‘step’ button to its left. Maybe I should!

Islands of Certainty

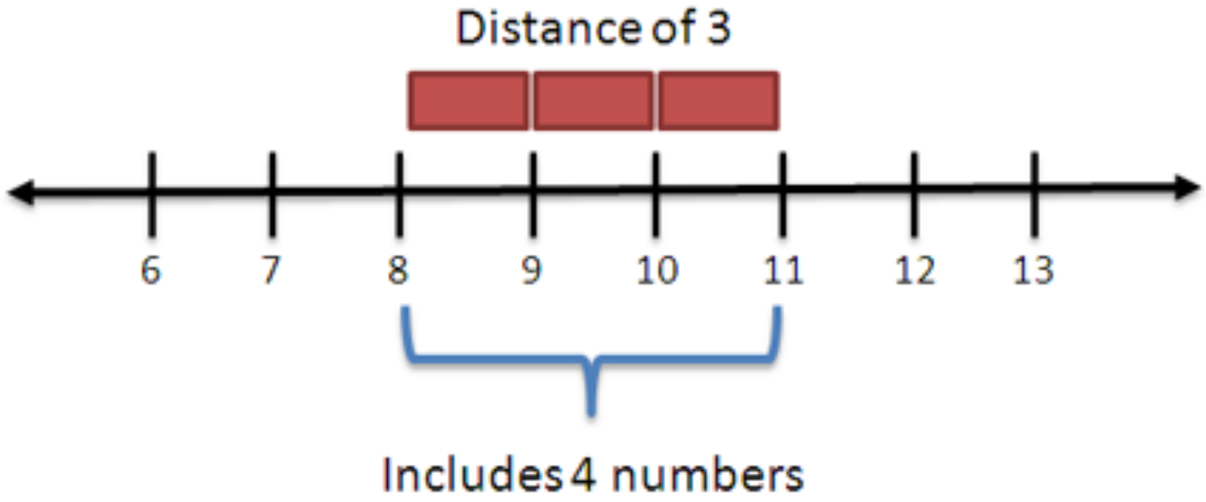
Divide and conquer!

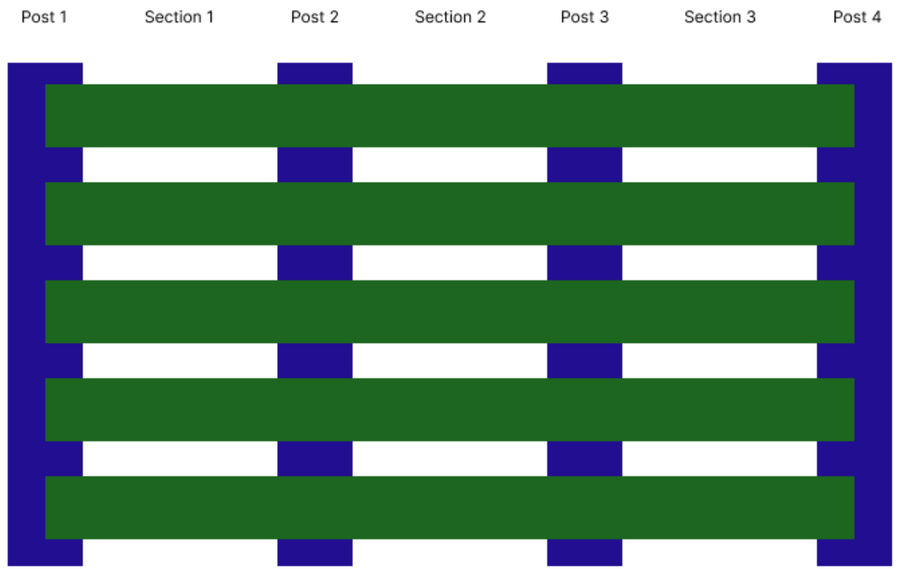
- Mentally split your code into “I know this works” and “I’m not sure about this”
- Then dive into the unclear area. Is it a behavior you’re looking for? A variable that gets an unexpected value?
- Keep expanding the “islands of certainty” until you know where the bug is.
- Eventually you’ll find it!

Off-By-One Errors

The software developer's birthright

- Happens whenever you have a collection of N things and you do something one too many or one too few times.
- Also called the 'fencepost problem':





Fencepost problem: four posts, three fence sections

Off-By-One Errors

```
for (let i=0; i <= myList.length; i++) {
  doSomething(myList[i]);
}
```

This will explode when i has the value of myList.length.

Remember why? The valid values for indexing into a list of length n are zero through through n-1.

How could we fix this?

“Forever Problems” in software are rare

- In a physical craft, “measure twice, cut once” is extremely good advice
- You can screw something up permanently
- Except drywall. Drywall is an extremely forgiving medium. 😜
- In software, we have loads of guard rails to help you recover from mistakes
- So you can be a bit cavalier if you know how to use the guard rails

“Forever Problems” in software are rare

- Undo. Easy.
- Use source control like Git.
- Write and use automated tests
- Have friends, enemies, colleagues, look at your code so they can feel smug about finding your mistakes
- In general, “think with your hands” by fearlessly writing code, knowing that problems, mistakes, bugs are likely transient.

Closing Idea

Have a bias towards making progress

- It will take time!
- Switch between active thinking and writing code
- Avoid too much philosophical wall-staring
- Develop your own tactics and strategies for making progress and identifying problems in process and in the code
- Don’t let uncertainty prevent you from proceeding. Uncertainty is the main ingredient in any interesting pursuit.