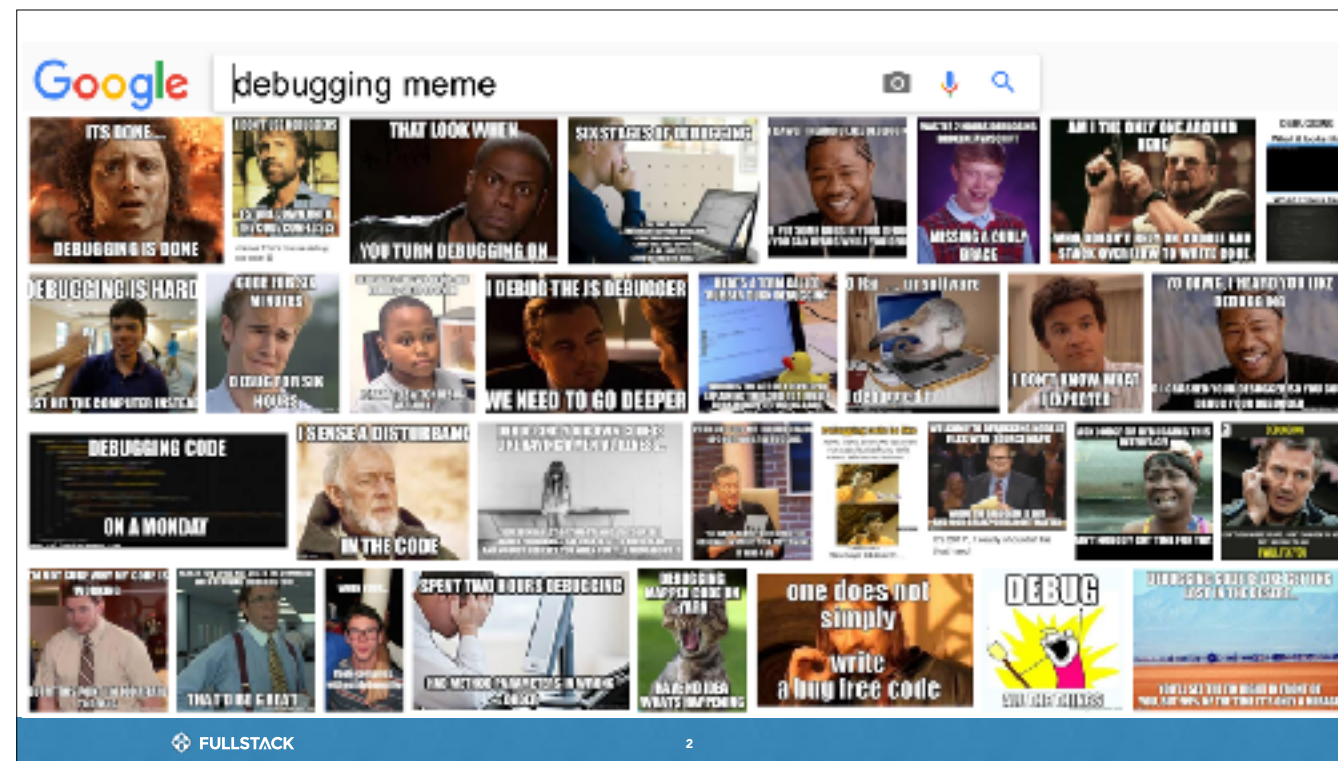


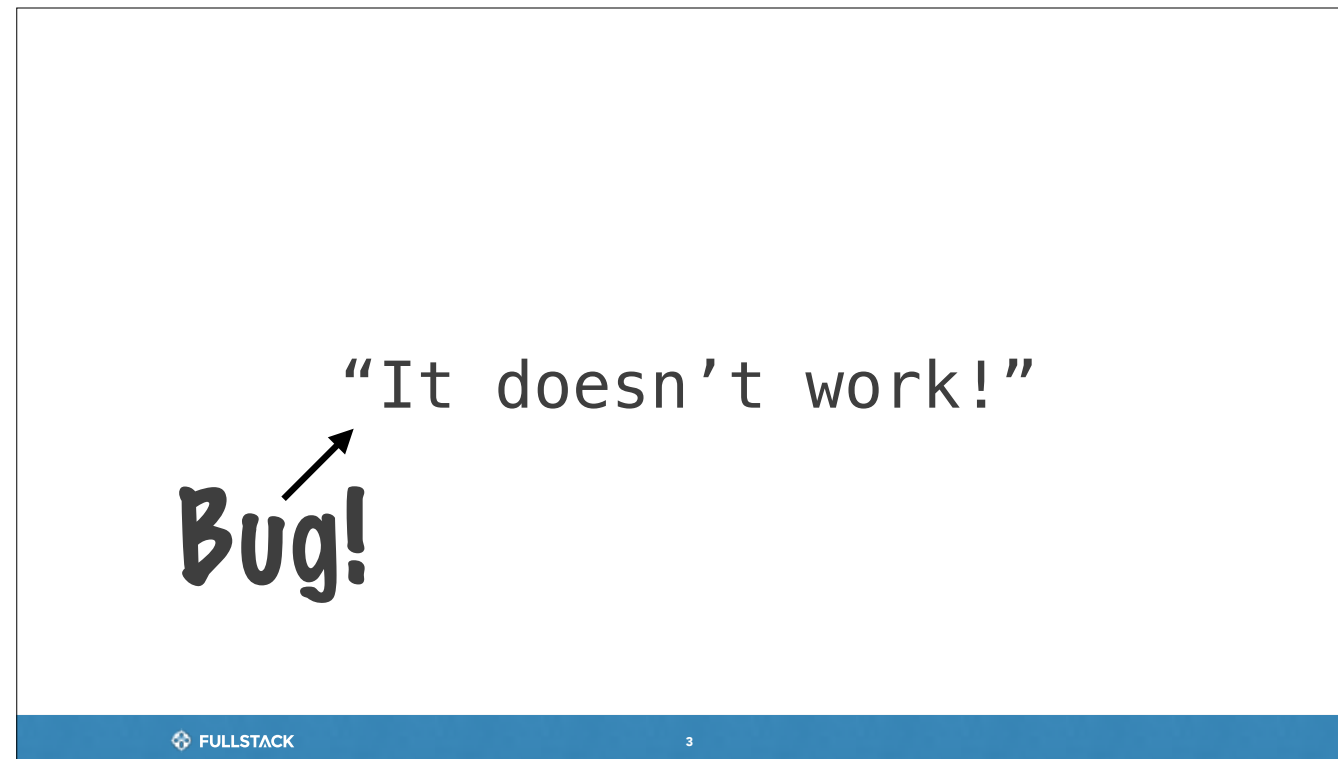
# DEBUGGING





Debugging is a HOT TOPIC


Pro tip: If you're at a networking event and you don't know what to talk about to a developer, ask them about their worst debugging session.




How to know you have a bug.  
If somebody says...

“Why is it doing that?”

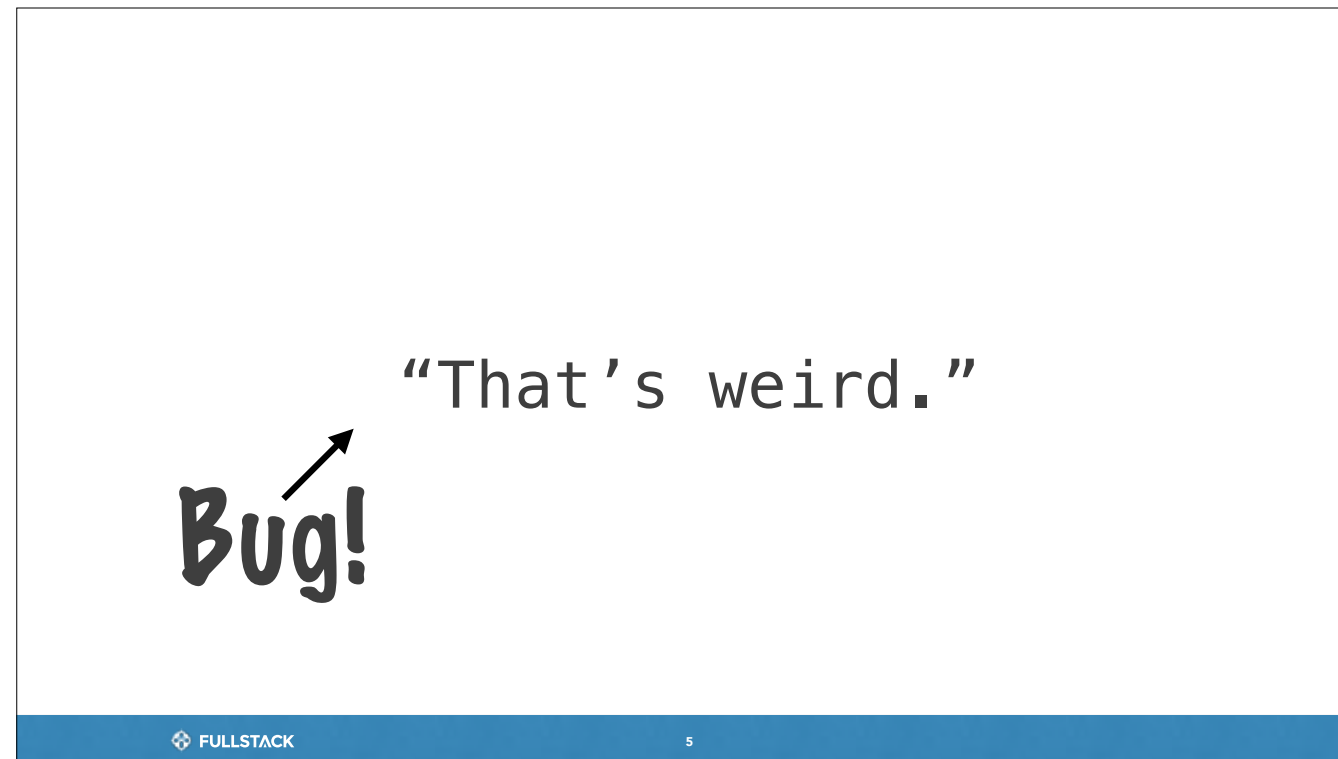
**Bug!**



 FULLSTACK

4

Or when they ask...



The computer exhibits weird behavior....  
It's a bug!:

```
const foo = `bar`;
```

Some bugs are very simple.

```
funciton bar () {  
  
}
```

Some are cruel

"We can't send [email]  
more than 500 miles"

<https://www.ibiblio.org/harris/500milemail.html>

Some are **very** cruel.

Yes, this is real story: <https://www.ibiblio.org/harris/500milemail.html>

There was a campus email system that was unable to deliver emails to locations more than 500 miles away.

The system was set to timeout on a very short interval, it turned out to be about 3 milliseconds.

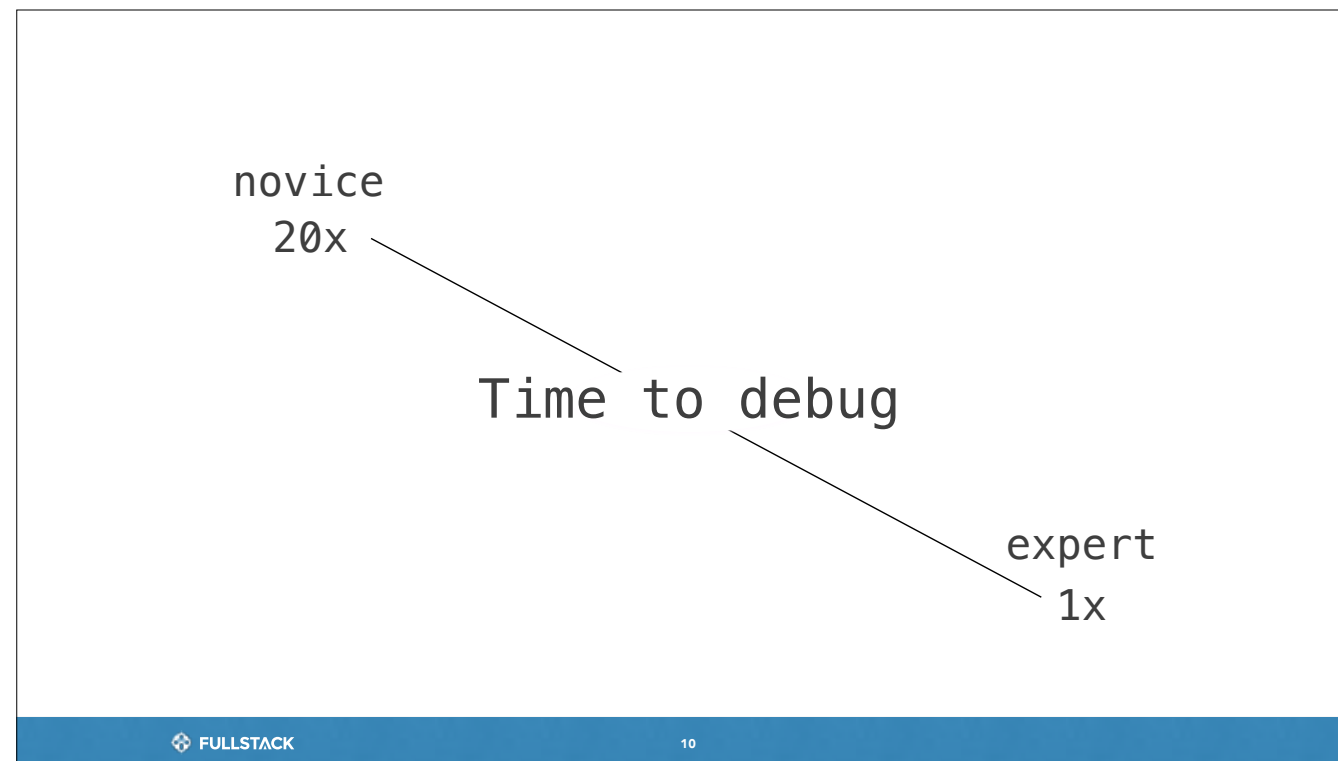
Light travels ~558 miles in 3 ms



It is not so bad.

But others have come here before us.

There is a bounty of debugging tactics and tools.



And you will be greatly rewarded by improving your debugging skills

Studies show a **20:1** difference in time to debug between experienced and inexperienced developers, also leaving (and creating!) far fewer bugs in the process (McConnell, Code Complete).

What You'll Learn Today

BUG Prevention  
Detection  
Diagnosis  
Fixing

Hey look. bug fixing is last

# PREVENTION

An ounce of prevention is worth a pound of cure.

A stitch in time saves nine.

## PREVENTION

“Debugging is twice as hard as writing the code in the first place.

Therefore, if you write the code as cleverly as possible, you are,

by definition, not smart enough to debug it.”

– Brian W. Kernighan

Brian W. Kernighan, Co-Author of “the C programming language” (regarded by many to be the authoritative reference on C) among many other books.

# PREVENTION

- 🐛 Readability > smallness
- 🐛 Simple > clever
- 🐛 git hygiene
- 🐛 Formatting matters!

Thousand-page books have been written on this subject, but here are some tips that will keep many bugs away – Keep them in mind

Formatting matters! It's not just about opinionated aesthetics. It is about reducing complexity, improving maintainability, and consequently eliminating the potential for bugs.

# PREVENTION

```
1 async function main
2   {}() {
3   try { throws();
4   }catch (err) {console.log('caught');}}
```

indentation matters

:screamface:

# PREVENTION

```
1 async function main () {  
2   try {  
3     throws();  
4   }  
5   catch (err) {  
6     console.log('caught');  
7   }  
8 }
```

indentation matters



# PREVENTION

- 🐛 Readability > smallness
- 🐛 Simple > clever
- 🐛 git hygiene
- 🐛 Formatting matters!
- 🐛 Use functions in your favor
  - Prefer small, single-purposed and pure functions

Functions are an integral part of coding in languages such as Js. Here are a few tips:

- Use small functions. Try imposing an arbitrary limit (10 lines is plenty).
- Prefer single-purpose functions. If your function does two things, maybe it should be two functions.
- Prefer pure functions. Pure functions neither rely on nor influence the surrounding scope. They take in inputs and return outputs — nothing more. That makes them easy to reason about (stateless)

# PREVENTION

```
1 let result
2 const add (a, b) => {
3   result = a + b
4 }
5 add(10, 20)
6 console.log(result)
```

avoid side-effects

This function `add` is changing a variable outside of itself.

This is difficult to read, and easy for another developer to misunderstand.

# PREVENTION

```
1 const add = (a, b) => a + b  
2 console.log(add(10, 20))
```

avoid side-effects

We didn't even need that outside variable.

# PREVENTION

```
1 const addOrSubtract = (a, b, operation) =>  
2   operation === 'add' ? a + b : a - b
```

single-purpose functions

# PREVENTION

```
1 const add = (a, b) => a + b  
2 const subtract = (a, b) => a - b
```


single-purpose functions

Split out as two functions this is much simpler to reason about



# PREVENTION

**Tools can get help.**

Actually, you can set assistive tooling in your computer, and it helps quite a bit with this.



# PREVENTION

-  linter
-  auto-formatter

FULLSTACK 23

Assistive tooling can help reduce entire classes of bugs.

After this lecture, you'll be installing some helpful tools to help with this.

# PREVENTION

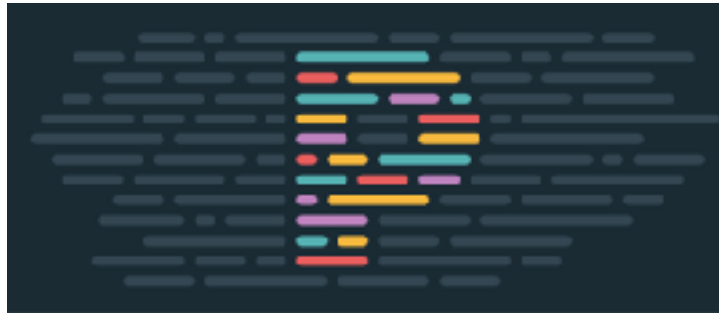


ESLint is great too.

[ demo eslint: <https://eslint.org/demo/> ]



# PREVENTION



Prettier is great.

[ demo prettier: <https://prettier.io/> ]