UNC COMP 590-145
Debugging
Monday, March 23, 2020

(Activity: post in Zoom's chat: where are you right now? And, if you feel comfortable sharing, how has COVID-19 affected you?)

(Song: Sweet Caroline by Niel Diamond, because even though we're not together on campus, we're still Carolina.)
Announcements

Videos

- Moving to online classes (a.k.a. using Zoom for class)
- Git branches, refs, and the HEAD ref (a.k.a. concepts for A4)
  - (You've watched these already, right?)
- Using virtual office hours (optional)
- Pair programming with zoom tutorial (optional)
Announcements

New options for final grades

- You can decide to take this class pass/fail
  - Can decide as late as Aug. 7, 2020
  - A pass won't affect your GPA (but a fail will, apparently)

- You can get a "CV" incomplete
  - Like an incomplete but with 3 months longer to finish
  - (But what does an incomplete mean in this course?)
Announcements

Regarding A3

- As of this morning, 23 of you are at or near full credit
- Many of you are struggling, mainly with write-wtree
- COVID-19 changes have made it harder to get help
- So, new late policy:
  - Max 10% late penalty—so get a CV grade and submit in July, only 10% penalty
  - New late window is 1 week
  - Still linear scaling of the penalty
Logistical questions?

(Raise your hand in Zoom, and I'll call on you. Remember to unmute yourself (e.g. by holding down the space bar).
Poll: git branching

Go to pollev.com/jeffterrell and answer the questions.
Debugging
Debugging

Why now?

- I originally planned to talk about polymorphism and interfaces at this point
- But I realized that this might be more helpful to y'all, including for A3
- Also, I read a really good book on debugging over the break
Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems by David J. Agans
Debugging

General comments

we fumble around when it comes to debugging

but there is a methodical approach

not all bugs need a methodical approach

for those that do, it's nice to have one
Debugging

The 9 rules

Understand the system
Make it fail
Quit thinking and look
Divide and conquer
Change one thing at a time
Keep an audit trail
Check the plug
Get a fresh view
If you didn’t fix it, it ain’t fixed
Debugging rule 1: understand the system
Debugging: 1. Understand the system

Foundations

The foundational assumption in programming is: the computer is only doing what it's told.

When this (very rarely!) isn't true, e.g. because of a hardware failure, there's an even more foundational rule that's always true:

- _There's a good reason why that happened._

Story: ice cream flavor affects car's ability to start (from the book)
Debugging: 1. Understand the system

Human reactions

How might we react to a bug?

I knew a guy once who would loudly curse his computer and beat on it (seriously)

More commonly, we get frustrated, right?

But examine this reaction in light of the foundational assumption. The computer is only doing what it's told! No sense getting frustrated, at least not at it.

A better reaction: "This is happening for a reason. Let's understand what that reason is."
Debugging: 1. Understand the system

Perspectives

Typical perspective: "Stupid computer! Do what I mean!"

Typical perspective: "Ugh. I must be an idiot."

Better perspective:

There's a reason why this isn't working, and I don't understand that reason yet. Once I do, I'll understand the system better and be better equipped to deal with future bugs. Let's get started!

- In other words, bugs aren't a waste of time, they're a growth opportunity.
Debugging: 1. Understand the system

The tortoise vs. the hare

Typical response: thrashing around

This often makes the problem worse.

Even if you fix it, you probably didn't learn much.

Better response: be methodical and mindful of what you're doing

Takeaway: sometimes, going fast is the slowest way to go, and going slow is the fastest

(source: Wonderopolis.org)
Different kinds of programmers

There's a kind of programmer I like to call a "framework jockey"

- They know how to use the framework, and that's about it.
- They can crank out some programs—so long as those programs stay on the happy path of what the framework intends to support.
- But any significant changes are a no-go for the framework jockey—there be dragons!

Don't be a framework jockey.
Debugging: 1. Understand the system

Pursue mastery

Rather than ride a framework, **pursue mastery**.

It's a rewarding, satisfying pursuit.

*It is possible to master your craft.*

It's not an easy road, but it shouldn't be a mysterious one.

One starting point: a bug, which is an invitation to pursue mastery.
Debugging: 1. Understand the system

Tips for self-learning

- (From 7 tips for successful self-learning, by Bradford Cross and Hamilton Ulmer)

  Go slow (see tortoise vs. hare above)

  Avoid multitasking: turn off notifications; close chat windows; focus

  Don't read through books, *work* through them. (Maybe with a REPL.)
plan your day around your classes. There are distractions, of course, but if you’re concerned with learning at school, you prioritize your classes over other things. You don’t have to be
Debugging: 1. Understand the system

Manuals

Manuals are often overlooked but are one of the richest sources of understanding.

They're *systematic*: they're thorough, covering everything you need to know about how the system works.

Some of you will prefer to avoid reading where possible. If you can find a *systematic* treatment of a topic, e.g. in a screencast, great. But often such things don't exist. It pays to get used to reading.

Example: I read through the Vim manual in college, and have always been glad I did. (Hint: pursue mastery of your development tools, too.)
Debugging: 1. Understand the system

Story time

- Junior and Kneejerk (from the *Debugging* book)
Reflection exercise

See pollev.com/jeffterrell.

You have 5 minutes.

(countdown timer)
Bonus: A3 foundational concepts

Here are some of the underlying concepts in A3. Are there any you feel shaky on?

1. the file system: directories (and nesting thereof), files, and paths
2. git's content-addressable object database: object types, addresses, storage locations
3. hexadecimal notation
4. **unicode** and difficulties with strings and binary data
5. the format of blobs, trees, and commits in git
6. how Clojure can deal with binary data, including the functions I provided