CS 4350: Fundamentals of Software Engineering
Lesson 4.4: Debugging

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand

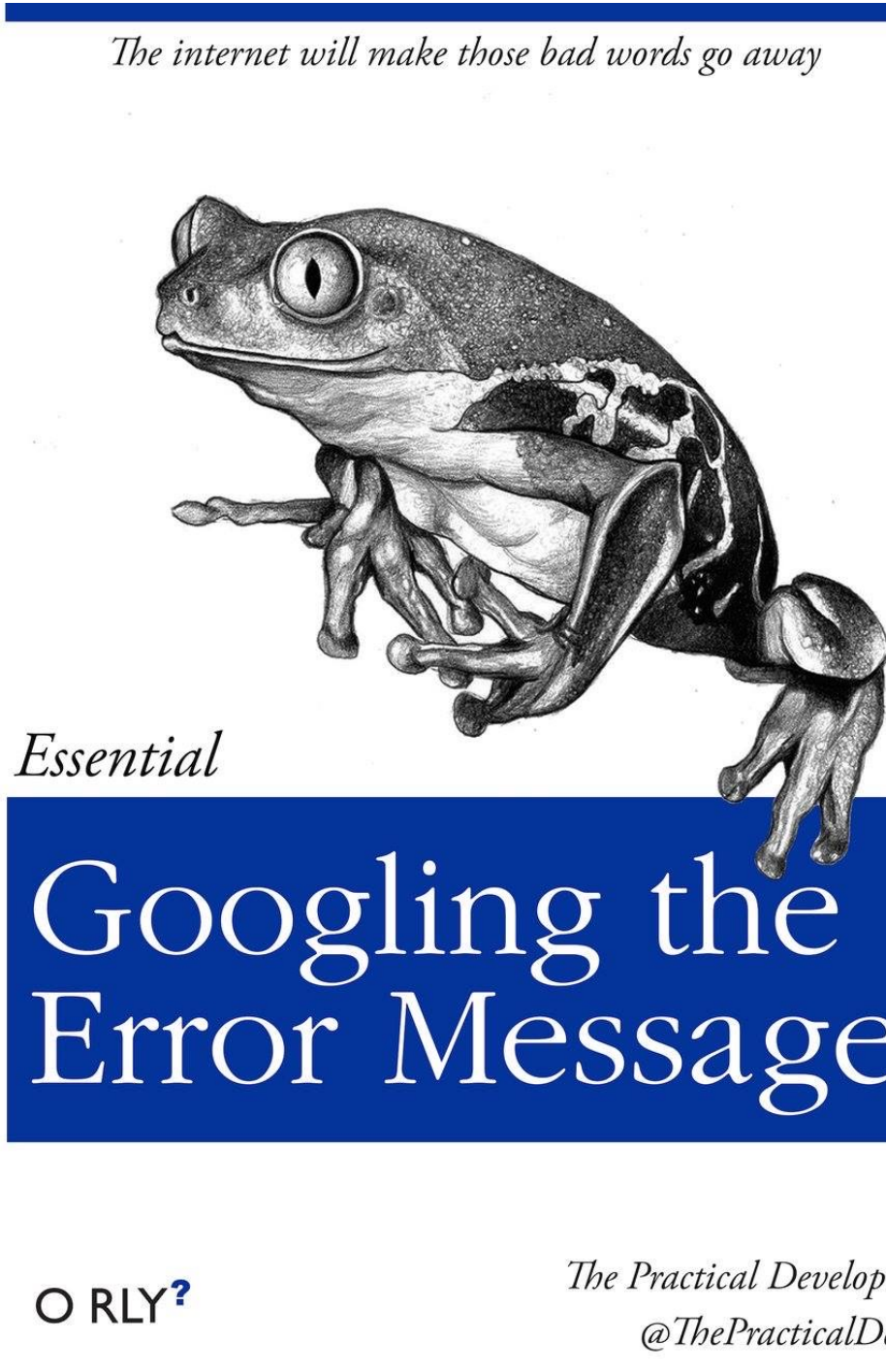
Khoury College of Computer Sciences

Learning Objectives for this Lesson

- By the end of this lesson you should be able to:
  - Understand the value of scientific debugging
  - Enumerate general debugging strategies beyond “google it” or “turn it off and on again”

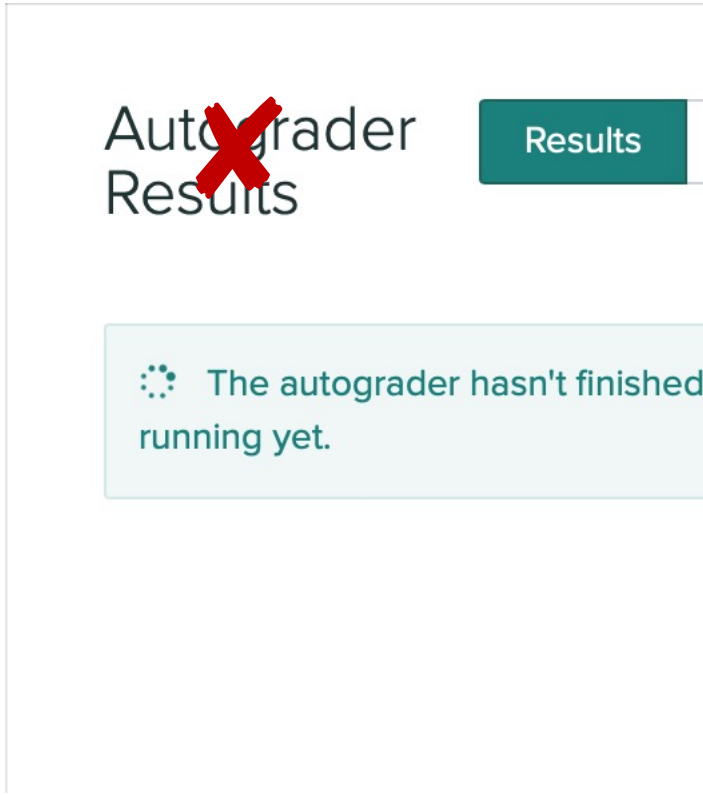
It is possible to "get better" at debugging

- Debugging is NOT just "googling the error message"
- Some developers are better at debugging than others
- The overall goal in this lesson is to teach you about different *strategies* for debugging, how to decide which strategy to apply, and how to view debugging as a science



General Strategy: Enable Efficient Reproduction of the Problem

- If you haven't fixed the bug yet, it's likely you will need to execute the program many times, tweak things, and see what happens
- Create the fastest test case possible to reproduce the bug:
  - Consider starting "top-down" from the application's entry point, or "bottom-up" by directly invoking some buggy code
  - Ensure that it is self-contained
- Minimize turnaround time from changes to result:
  - Make sure that you can quickly recompile your code and run the test



General Strategy: Scientific Debugging

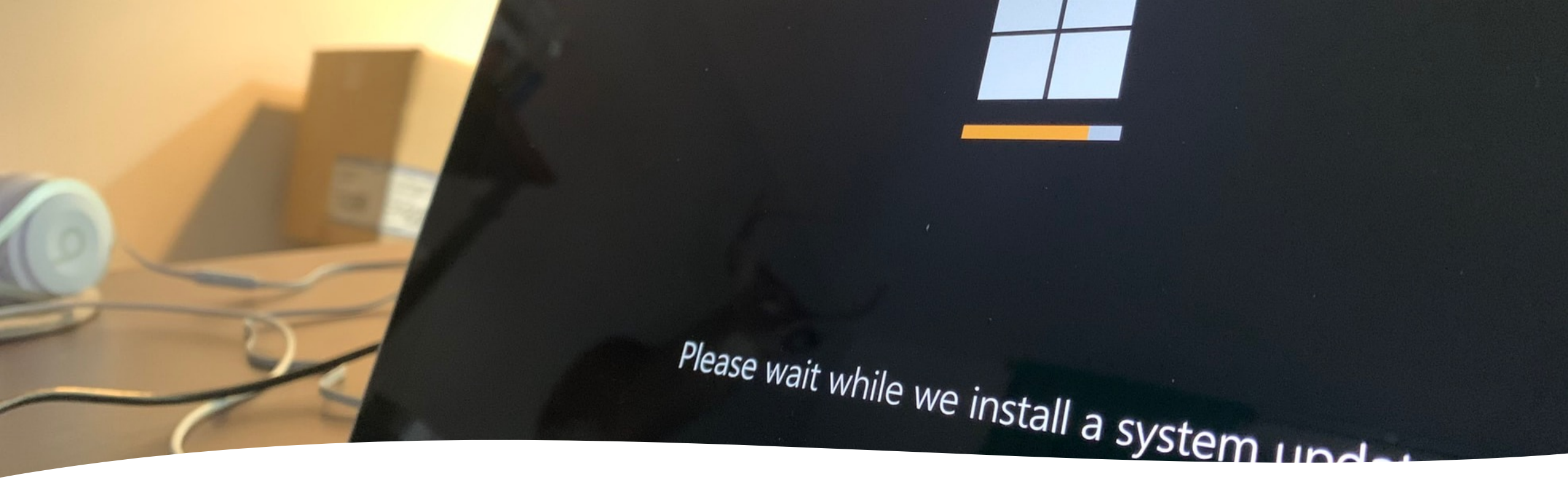
- Create a debugging log:
  - What was the input/application state that caused the bug?
  - What was the behavior that I expected?
  - What was the behavior that I observed?
  - What are possible hypotheses for that behavior?
  - How have I tested those hypotheses, and what was the result?
- Do this in an issue tracker, or even in a personal note

Generating Hypotheses: Why is this buggy?

- Good hypotheses are *testable*
- Start with a few hypotheses, and let your inquiry drive the generation of new hypotheses
- Determine if you *need* to know exactly why the bug occurred, or fixing it is good enough
- Has recent code that I've added introduced this bug?
- Have I seen a similar bug before?
- Have I made an incorrect assumption about how a library or API works?

High-Level Debugging and Hypothesis Generating Strategies

- Use focused queries on the web to search for insights
- Confirm that (assumed) preconditions actually hold
- Find the difference between a working and failing:
  - Version of your code
  - Inputs to your code
  - Environments where your code runs



“Is there an updated version of this library?”

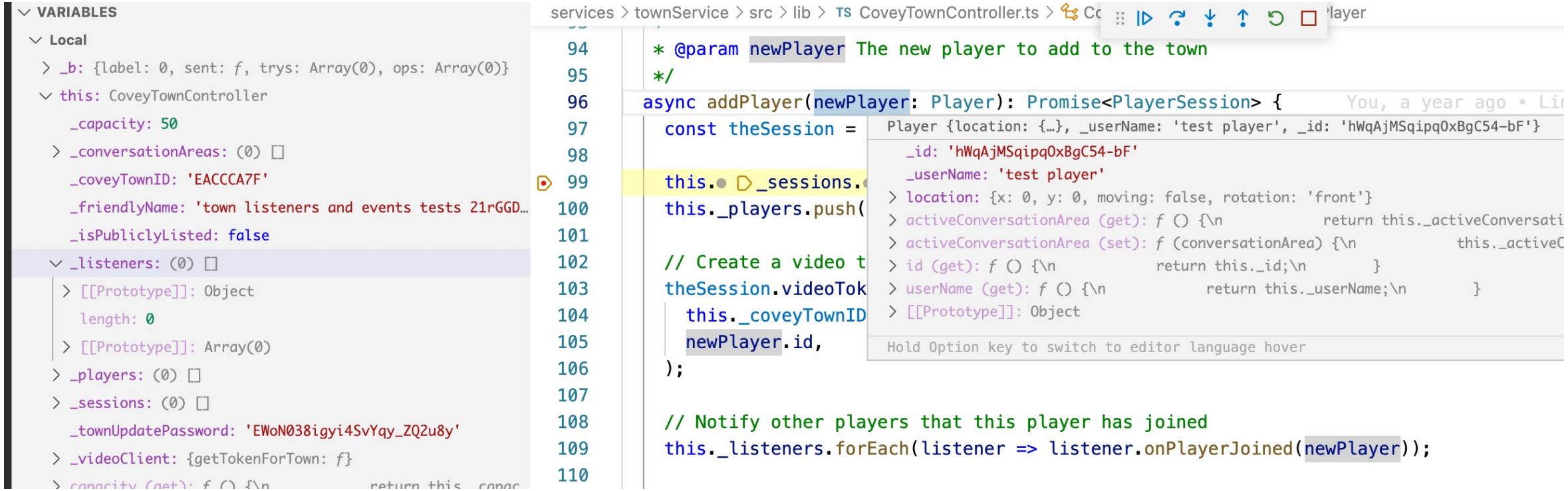
- Consider updating all dependencies, or just those related to bug
- Be prepared to revert to old versions if needed
- For NodeJS:
  - `rm -rf node_modules && rm -rf package-lock.json && npm install`

Reasoning Through Code

- Carefully examine each line of code, explaining to yourself, your colleague, or “a rubber duck” what it should do
- Represent complex data structures (and their manipulation) on paper, or in a separate text file

Use A Debugger

- Helps narrow down on the code that interests you
- Set breakpoints
- Look for errors by examining values of variables and expresisons
- Get comfortable with the keyboard commands to quickly interact with the debugger



Add Logging Statements

- Add log messages to help you understand the program's execution
- Particularly useful for non-deterministic bugs ("heisenbugs"), or bugs that involve many systems interacting
- Consider using a logging library to help you trace the source of each log message – log messages are a useful artifact for future debugging

```
Successfully reuqested job
JWT token might be expired, renewing
Renewing JWT token
Renewing JWT token: success
Received runner launch request:
{
  "gitHubURL": "https://github.com/neu-se/covey.town",
  "launcherToken": "asdfasdfasdfadsfwerasdfsdf",
  "runnerLabels": "self-hosted"
}
--unattended --replace --url https://github.com/neu-se/covey.town --token
$GHTOKEN --ephemeral --work gha_work --labels 'self-hosted'
Making request to slurm:
{"script":"#!/bin/bash\nfunction checkForRunning(){\n  sleep
120\n  RUNNING=$(grep Running /tmp/gha-stdout | wc -l)\n  if [ $RUNNING -eq 0
]; then\n      echo \"Does not look like GHA found a job to work
on!\"\n      kill $$\n  fi\n}\nset -x\nhostname\nexport HOME=/tmp/home/ci-
runner\necho \"GHA Runner is loading...\"\necho $HOME\ncd $HOME\npwd\nmkdir
gha\ncd gha\ncp /experiment/util/actions-runner-linux-x64-2.285.1.tar.gz
.\ntar xzf actions-runner-linux-x64-
2.285.1.tar.gz\nTOKEN=asdfasdfasdfadsfwerasdfsdf\nGHTOKEN=$(curl -s -X POST -
H \"Authorization: $TOKEN\" https://ci.in.ripley.cloud/gha/runner | jq -r
'.token')\necho  --unattended --replace --url https://github.com/neu-
se/covey.town --token $GHTOKEN --ephemeral --work gha_work --labels 'self-
hosted' \n./config.sh --unattended --replace --url https://github.com/neu-
se/covey.town --token $GHTOKEN --ephemeral --work gha_work --labels 'self-
hosted' \ncheckForRunning &\n./run.sh\n","job":{"comment":"GitHub Actions
Builder for https://github.com/neu-
se/covye.town","name":"GitHubActions","ntasks":1,"partition":"gha","standard_
output":"/tmp/gha-stdout","standard_error":"/tmp/gha-
stderr","current_working_directory":"/tmp","environment":{"fooo":"bar"}}}
```

Write Automated Tests

- Encode the minimal steps to reproduce the bug and the expected result in a test
- Simplifies reproduction and evaluating fixes
- Detects regressions

```
describe('Create student', () => {
  it('should return a valid ID', async () => {
    const createdStudent = await client.addStudent('Avery');
    expect(createdStudent.studentID).toBeGreaterThan(0);
  });
})
```

Example test that might be useful if this input revealed a bug in student ID generation

Add Assertions

- Capture what you think the state of your program should be in assertions
- Assertions are a form of documentation
- Common assumptions to check:
  - Pre-conditions and post-conditions of functions
  - After calls to APIs that are expected not to fail
  - After loading remote resources
  - After evaluating complex expressions to make sure that the result has some expected property or is otherwise reasonable
  - In the “default” case of a switch statement

Use a specialized debugging tool

Example: Valgrind for C/C++ Memory Errors

```
$ ./main
Segmentation fault (core dumped)
```

https://www.valgrind.org/info/tools.html

```
$ valgrind ./main
==8515== Memcheck, a memory error detector
==8515== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==8515== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==8515== Command: ./main
==8515==
==8515== Conditional jump or move depends on uninitialised value(s)
==8515==    at 0x400813: fail() (main.cpp:7)
==8515==    by 0x40083F: main (main.cpp:13)
==8515==
==8515== Invalid read of size 4
==8515==    at 0x400819: fail() (main.cpp:8)
==8515==    by 0x40083F: main (main.cpp:13)
==8515==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==8515==
==8515==
==8515== Process terminating with default action of signal 11 (SIGSEGV): dumping core
==8515==  Access not within mapped region at address 0x0
==8515==    at 0x400819: fail() (main.cpp:8)
==8515==    by 0x40083F: main (main.cpp:13)
==8515==  If you believe this happened as a result of a stack
==8515==  overflow in your program's main thread (unlikely but
==8515==  possible), you can try to increase the size of the
==8515==  main thread stack using the --main-stacksize= flag.
==8515==  The main thread stack size used in this run was 8388608.
==8515==
==8515== HEAP SUMMARY:
==8515==     in use at exit: 72,704 bytes in 1 blocks
==8515==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==8515==
==8515== LEAK SUMMARY:
==8515==    definitely lost: 0 bytes in 0 blocks
==8515==    indirectly lost: 0 bytes in 0 blocks
==8515==      possibly lost: 0 bytes in 0 blocks
==8515==    still reachable: 72,704 bytes in 1 blocks
==8515==         suppressed: 0 bytes in 0 blocks
==8515== Rerun with --leak-check=full to see details of leaked memory
==8515==
==8515== For counts of detected and suppressed errors, rerun with: -v
```

Set Yourself Up For Debugging Success

- Invest in growing your experience of applying these debugging strategies
- For complex problems:
  - Consider each of these debugging strategies
  - Use a notebook to keep track of what you've tried
  - Take a break! Do not spend more than a few hours at a time in a debugging session
  - Be persistent

Review: Learning Objectives for this Lesson

- By the end of this lesson you should be able to:
  - Understand the value of scientific debugging
  - Enumerate general debugging strategies beyond “google it” or “turn it off and on again”