Hartley Brody

10 Debugging Tips for Beginners: How Troubleshoot and Fix Your Code Witl ur Hair Out

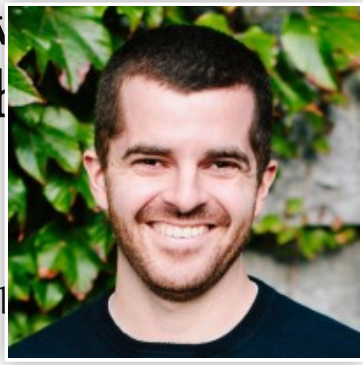
ıber 21, 2016

past summer, I launched a web scraping class targeted at total l

talking to a lot of people who were having web scraping challenges over the past ears, I realized that many of them weren't coders and didn't have any technical grounds. Many people who were looking to learn web scraping had never coded e.

hen I was making the material and lessons for the course, I had to keep this in the of mind. I couldn't expect students to know what a for-loop is or how to spot x errors in their text editors.

I really tried to hold students' hands through all of the class material, writing the simplest possible code, explaining things line by line as I went, and intentionally running into common mistakes to show them how to power through.



I'm a full-stack web developer and tech lead with 8 years of experience across many modern tech stacks.

I'm always looking to talk to developers and contribute to cool projects. Contact me or check out my side projects.

Get Email Updates

One or two emails a month about the latest technology I'm hacking on.

email address...

Join »

Web Scraping Guides

Check out my web scraping content library for articles, books and other resources.

But even with all of that assistance, the real world is a lot messier. Students would struggle to stumble when they'd go out on their own. Their code **"wouldn't work" and they'd feel hopelessly stuck pretty quickly**.

I realized that the process of troubleshooting and fixing the bugs in your code isn't intuitive yet to anyone who hasn't spent a long time learning to code [illegible] decided I'd collect some of my **most common tips for beginners** who [illegible] their code is giving errors or isn't doing what they want it to do.

e these tips are generally geared towards beginners, many of them [illegible] still follow day-to-day when debugging issues with my own code. Even as one who has been coding full time for over 7 years, I use these st[illegible] antly (especially #9).

Print things a lot

very single line of code, you should have a sense of what all of th[illegible] f you're not sure, print them out!

when you run your program, you can look at the console and se[illegible] t be changing or getting set to null values in ways you're not exp[illegible]

Sometimes it's helpful to print a fixed string right before you print a v[illegible] your print statements don't all run together and you can tell what is b[illegible] where

```
print "about to check some_var"
print some_var
```

Sometimes you may not be sure if a block of code is being run at all. A [illegible] `"got here"` is usually enough to see whether you have a mistake in yo[illegible] like if-statements or for-loops.

#2. Start with code that already works

When in doubt, start with someone else's existing code that already w[illegible] you're a beginner, you're still more of a Hacker than an Engineer, and so it's better to start with an existing structure and tweak it to meet your needs.

If you're working on your own project, try googling around for a script that does what you're trying to do. In my web scraping class, I provide working pyth[illegible] that completes the tasks in each lesson.

Popular Articles

- I Don't Need No Stinking API: Web Scraping For Fun and Profit
- Facebook Messenger Bot Tutorial: Step-by-Step Instructions for Building a Basic Facebook Chat Bot
- Web Scraping Reference: A Simple Cheat Sheet for Web Scraping with Python
- Startup Security Guide: Minimum Viable Security Checklist for a Cloud-Based Web Application
- How to Scrape Amazon.com: 19 Lessons I Learned While Crawling 1MM+ Product Listings
- Scaling Your Web App's 101: Lessons in Architecture Under Load
- How HTTPS Secures Connections: What Every Web Dev Should Know
- Peeling Back the ORM: Demystifying Relational Databases For New Web Developers
- Lightning Fast Data Serialization in Python
- Minimum Viable Git Best Practices for Small Teams
- 7 Reasons I Won't Sign Your NDA Before a Coffee Meeting
- Focus on the Product, Not the Code: How I Build Software for Clients
- How I Learned to Code in Only 6 Years: And You Can Too!

More Info

- Web Scraping Content Library
- Contact Me

Make sure you run the code you find before you make any changes to it works properly and does what it claims to do. Then make small changes to the existing code and test it often to see if your changes have introduced bugs.

#3. Run your code every time you make a small change

Do not start with a blank file, sit down and code for an hour and then run your code for the first time. You'll be endlessly confused with all of the little errors you may have created that are now stacked on top of each other. It'll take you forever to peel back all the layers and figure out what is going on.

Instead, you should be running any script changes or web page updates every few minutes – it's really not possible to test and run your code *too* often.

The more code that you change or write between times that you run your code, the more places you have to go back and search if you hit an error.

Also, every time you run your code, you're getting feedback on your work. Is it getting closer to what you want, or is it suddenly failing?

Read the error message

It's really easy to throw your hands up and say "my code has an error" and feel lost when you see a stacktrace. But in my experience, about 2/3rds of error messages you'll see are fairly accurate and descriptive.

The language runtime tried to execute your program, but ran into a problem. Maybe something was missing, or there was a typo, or perhaps you skipped a step and now it's not sure what you want it to do.

The error message does its best to tell you what went wrong. At the very least, it will tell you what line number it got to in your program before crashing, which gives you a great clue for places to start hunting for bugs.

#5. Google the error message

If you can't seem to figure out what your error message is trying to tell you, your best bet is to copy and paste the last line of the stacktrace into Google. Chances are, you'll get a few stackoverflow.com results, where people have asked similar questions and gotten explanations and answers.

can sometimes be hit-or-miss, depending on how specific or generic your error is. sure you try to read the code in the question and see if it's similar to yours – sure it's the same language! Then read through the comments and the first 2-3 ers to see if any of them work for you.

Guess and Check

're not 100% sure how to fix something, be open to trying 2 or 3 things to see happens. You should be running your code often (see #3), so you'll get feedback ly. Does this fix my error? No? Okay, let's go back and try something else.

There's a solid possibility that the fix you try may introduce some new error, and it can be hard to tell if you're getting closer or farther away. Try not to go so far down a rabbit hole that you don't know how to get back to where you started.

Trying a few things on your own is important because if you go to ask someone else for help (see #10), one of the first things they'll ask you for is to list 2-3 things you've tried already. This helps save everyone time by avoiding suggestions you've already tried, and shows that you're committed to solving your problem and not just looking for someone else to give you free, working code.

#7. Comment-out code

Every programming language has a concept of a comment, which is a way for developers to leave notes in the code, without the language runtime trying to execute the notes as programming instructions.

You can take advantage of this language feature by temporarily "commenting out" code that you don't want to lose track of, but that you just don't want running right now. This works by basically just putting the "comment character" for your language at the start (and sometimes at the end) of the lines that you're commenting out.

```
# in python, you use the "hashtag" character to turn a line of code into a
# here_is = some_code.that_is_commented_out()  # this won't run
here_is = some_code.that_is_NOT_commented_out()  # this WILL run
```

If your script is long, you can comment out parts of the code that are unrelated to the specific changes you're working on. This might make it run faster and make it easier to search the remainder of the code for the mistake.

nake sure you don't comment out some code that sets variables that your program ıg later on – the commented-out code is skipped entirely, so statements that run on won't have access to any variables that are set or updated in the commented out ons.

of course, make sure you remove the comment characters so that it turns back into ıctions when you're done testing the other sections.

If you're not sure where the problem is, do a binary search

nore code you have that's running, the more places you have to check for an error. cially as your project grows past a few dozen lines, it can get more and more ult to find out where errors are happening. Your stack trace and error message should give you a clue as to where things are going wrong, but sometimes they're not too helpful.

In that case, it's helpful to do a binary search to hone in on the section of code that's misbehaving.

At a high level, a binary search involves splitting something in half and searching each of the halves for what you're looking for. Once you decide which half it's in, you repeat the process again on that half. This is one of the quickest ways to hone in on where something is, in an otherwise large list of instructions.

For finding bugs in your script or web app, just run the first half of your code, comment out the second half, and then print the half-way done results. If those look right, then the first half of your code is running fine and the problem you're encountering must be in the second half. If there's a problem with the half-way done results, then the error is occurring somewhere in the first half.

Repeat this process over and over, and you'll be able to quickly hone in on the 2 or 3 lines that seem to be leading your program astray.

You may have noticed that this method combines a lot of the earlier steps of printing variables out, commenting code out and reading the error messages looking for line numbers. 👍

#9. Take a break and walk away from the keyboard

It's really easy to get caught up in the details of your current implementation. You get bogged down in little details and start to lose sight of the forest through the trees.

ses where I feel like I've been spinning my wheels for the last 20 or 30 minutes, I step away from the code and do some other activity for a little while before ng back. I'll go get a drink of water, meander around a bit or have a snack. It's a way to reset your mind, pull back and start to think of another approach.

ize that it's also seemingly unsatisfying if you haven't tried it. "If I walk away I'll forget all of these details! I'll have to start all over again. Plus I don't feel ied leaving code in a broken state. What if I never fix it and I'm a failure. I can't until it's working again." I used to think all of those things as well.

t has become one of my favorites tips and has helped me past dozens of bugs over ears. If you try it you might be surprised just how helpful that can be.

#10. How to ask for help

Okay okay, so you've really tried *everything* and it seems like nothing is working. Now you feel like you're really ready to ask someone else for help.

Before asking anyone about a bug in your code, it's important that you make sure you have all of the following components of an excellent question:

1. Explain what you're trying to do
2. Show the code that's giving the error
3. Show the entire stack trace including the error message
4. Explain 2-3 things that you've tried already and why they didn't work

Sometimes, in the simple process of going through these items in your mind and writing them down, a new solution becomes obvious. I call this **the "Stack Overflow" effect**.

Over the years, the times when I've really felt like I couldn't figure something out and need to ask for help, simply writing each of these 4 things down makes it suddenly

obvious what the problem is, or something new that I can try. I've abandoned many Stack Overflow question form boxes when the act of asking a good question makes the answer obvious.

It may work for you as well!

Hopefully that's helpful to anyone who is new to programming or getting started with first coding projects. Be sure to **share this with any beginner coders you know** – it might save them hours of feeling lost or frustrated.