There's a flip side to dealing with the agonizing precision of code and the grind of constant, bug-ridden failure. When a bug is finally quashed, the sense of accomplishment is electric. You are now Sherlock Holmes in his moment of cerebral triumph, patiently tracing back the evidence and uncovering the murderer, illuminating the crime scene using nothing but the arc light of your incandescent mind.

My friend Max Whitney has been a programmer for over two decades, but she still remembers the first time she fixed a truly fiendish bug. She was working as a programmer for New York University, and students were reporting some trouble logging in to the university's main web portal. Specifically, they would sometimes discover that they were logged into someone else's account. Whatever was going on?

At first, they noticed that a large number of complaints came from students who were logging into NYU's portal while using computers

at the Kinko's copy center around the corner. Maybe Kinko's was somehow to blame? But then Whitney saw reports of the same login bug from computers located on-campus. It became clear that the culprit was the university's login system itself. Unfortunately, that login code had been written years earlier by a staff programmer who no longer worked for NYU. Since Whitney couldn't ask him to help debug his code, she sat down to scrutinize it line by line with the help of another expert programmer.

Reading someone else's code can be a baffling task. That's because there's rarely a simple, obvious way to write a piece of code. Idiosyncrasies abound. Different coders have very different styles. If you asked four different programmers to write a pretty basic algorithm—say, one that figures out and prints the first 10,0000 prime numbers, for example—you'll likely get four different approaches that were structured and looked a bit different. Even something as simple as picking the names for one's variables can be a source of bitter argument between coders. Some prefer to use extremely short, one-letter variables (`x = "Hello, World!"`), arguing it keeps their code more compact and thus easier to glance at. Others prefer to use more descriptive variable names (`greetingToUser = "Hello, World!"`), pointing out that it'll be easier, a year later when the code is crashing, to look at a variable like `greetingToUser` and know what it means. When code gets particularly lengthy, or if something is particularly dense, coders are usually encouraged to leave little comments in their code that explain exactly what the heck is going on, so that some poor soul years hence will have guidance in sifting through the thicket. But often when coders are working fast, or under pressure, they don't "document" their code very much; and even with comments, frankly, figuring out the flow and logic of a piece of code can still be a brow-furrowing affair. (In well-functioning firms, no code is put into production until it's undergone "code review," with colleagues looking it over—not just to make sure it works, but that it's sufficiently readable by others.) One estimate suggests that coders spend 10 times the amount of time parsing lines of software than they do writing them. This is another reason coders can be so snippish and judgy about the style of their colleague's code. They know they may eventually need to read it.

This is the situation in which Whitney found herself. She and her colleague pored over the login code for hours, slowly figuring out how it worked, like an electrician patiently following the tangled wires that someone else had laid down in an apartment. *Hmmm, this* section triggers *that* chunk of code, which would get that *other* function to start up. . . .

Then suddenly, they saw it. When they finally had enough of the code's structure loaded into their minds, they could see the bug.

The problem began the moment someone connected to NYU's network. When students logged in, the system gave them a random temporary ID number for that session. To generate the random ID number, the program would "seed" its random-number generator using the timestamp, the exact instant that the student logged in. But what if two students just coincidentally logged in at precisely the same second? They'd be issued the same quasi-random number; oops! To prevent this, the programmer added another "seed" number, the IP address of the computer that the student was using. NYU had tons of IP addresses, so the programmer figured there was no chance any two students logging in would have precisely the same timestamp and precisely the same IP address. Right?

Nope. Years later, NYU and Kinko's switched over to a new technology that funneled lots of computers through just one or two IP addresses. They did this to handle the explosive growth of internet use on campus, but nobody realized it might interfere with the old login system, written so many years previously. But it did. Suddenly it became possible for two people to log in at the same time *and* have the

same IP address. The two users would be assigned the exact same session ID, and presto: One of them would be logged into the other person's account, able to see the other person's email and notes.

In a flurry, Whitney wrote some code to test to see if their diagnosis was correct. It was. They'd figured it out. It'd take more weeks of slogging to actually fix the bug, but at least the mystery was solved.

And she was suffused with a drug-like euphoria, a feeling of mastery and accomplishment that rendered her aglow. "It was wonderful," she recalls. "I walked the halls of Warren Weaver Hall, up and down the little H-shaped hall, just going, *I am a golden god! I am a golden god!*"

She wanted to savor the moment, because she knew it wouldn't last.

"I knew that the moment I sat down again, I was gonna find the next thing that was broken," she says, and sighs. Sure, lots of the code at NYU worked fine. Most of it did, probably some of which she had written herself. But you didn't spend much time pondering the stuff that worked. Indeed, by definition, if it's working, you're usually ignoring it. "The actual thing that a programmer spends their time on is all the shit that's broken. The entire activity of programming is an exercise in continual failure.

"The programmer personality is someone who has the ability to derive a tremendous sense of joy from an incredibly small moment of success."

Part of what's so thrilling about a programming "win" is how abruptly it can emerge. "Code can quickly change states; it goes from not working at all to working, in a flash," as Cal Henderson, the CTO and cofounder of Slack, once told me.

This provides a narcotic jolt of pleasure, one so intense that coders will endure almost any grinding frustration just to taste it again,