Ammar Mohamed Taha

Section 6

13-Sep-25

Computing Is Just Learning, Debugging, or Burning!

After working for almost six months on my project for the national scientific week competition and just when I was sitting there the night before, fixing the last details and trying to make it perfect, it exploded. Literally. That was one of the craziest moments of my life, but honestly, it just reminded me how long this journey has been for me with computers and tech.

It all started with my very first phone, a Lava Iris 600. To me as a 9-year-old, it was only a toy for games. But my dad wasn't having that. Instead of letting me waste my time, he slowly showed me his own projects and tried to get me interested in something bigger. He knew I liked math and computers, and he told me that no matter what career I'd choose later, knowing math and computers would always be useful. So he started guiding me step by step, sometimes with small lessons, sometimes by enrolling me in courses.

That's how I got into Scratch. At the beginning, it was just dragging blocks to make a cat move or create some silly animations, but for me it felt like magic. It was the first time I could actually tell a computer what to do and it listened. I used to make small projects—for instance, I had a game named *Flappy Potato*. You can see from the name how silly it was, nothing serious but it was fun. Also, for a beginner like me, I felt that I had done something incredible by copying this game and working on it for nearly a week. These small projects made me curious to learn more.

When I reached preparatory school, I decided it was time to try "real" programming. That's when I picked up HTML. At first, it looked super easy—just writing a few tags and boom, you have a webpage. But when I tried to make something proper, like a copy of Twitter's old website, I realized it wasn't easy at all. I had to use CSS to make it look nice and JavaScript to make it actually work. And wow, that was the hard part. One tiny mistake, a missing semicolon or bracket, could break the entire page. I remember refreshing the browser like a hundred times only to see the layout all broken, with text floating in random places and the colors looking like a rainbow gone wrong.

I even remember yelling at the screen once:

“Why won’t you just work?!”

And in my head the computer replied,

“Because you forgot a semicolon, genius.”

That’s when it hit me, coding was like playing chess against myself. Every move mattered, and every mistake taught me how to think sharper.

But honestly, those mistakes didn’t stop me. They made me want to know why it broke, and how to fix it. That’s when I really started to feel like a programmer.

High school was when things got much more serious. I wasn’t just making websites for fun anymore; I was building projects that actually meant something, and I encountered devices—or let’s say microcontrollers—that I had never heard about, like ESPs. I got into robotics and started working on things where coding connected with math and even physics. And it wasn’t only about building a physical model and its software, it was about the whole package.

The biggest project I have ever worked on was the *Roboworm*. It wasn’t just a random idea; it was something that had a real purpose. The project was designed to help people with gardening and sustainability, especially beginners who don’t have any background in agriculture. We wanted to create something that could literally “guide the soil.” I built a physical and functional prototype that had sensors for moisture, temperature, and light. These sensors sent data to an app and a website we created, both connected to a database. The system was powered by AI, which analyzed the data and gave simple advice—like what kind of plant would work best in the soil, or which fertilizer to use if something was lacking. It wasn’t just a piece of hardware, it was hardware, software, and AI all working together. For me, that project was proof that coding wasn’t just lines of text—it could actually change how people live and interact with nature.

And of course, the nightmare of every programmer showed up: the Bug. It always came at the worst times, like right before a presentation. Back in preparatory school, one error used to ruin my whole day. I still remember preparing to show one of my small projects to the principal, just so he could approve it for a school competition.

I had spent the whole week fixing details, polishing the design, and I was finally confident enough to show it. Then, just as I clicked run, the whole thing crashed—black screen, nothing. I froze. My heart was racing because I knew this was my only shot to convince him. Instead of panicking, I quickly retraced my steps, fixed the missing line of code, and somehow got it to run again in front of him. That moment taught me that bugs aren't the end of the road—they're just hurdles that test how much you want it.

By high school, I had learned that errors were just part of the process. You fix one, another shows up, you fix that too. Even when the whole *Roboworm* project exploded the night before the competition—and that happened because I didn't realize that even machines get tired. I had left the prototype running for a full week. It got its power from the sun during the day through a small solar panel, and at night it used the stored energy in the battery. Since it was only a prototype, the system overheated and, well, boom—exploded. My sibling and I were terrified; we couldn't even speak because we were exhausted from staying up late. We just looked at each other in shock, and I was honestly crying inside. But my parents woke up and calmed us down, reminding us that even if it was late, we could still rebuild and try again. And that's exactly what we did. We pushed ourselves to repair it as quickly as possible. Luckily, we still had all the code files, but the circuit was completely ruined, so we rebuilt it from scratch. Forntunatly all the late nights and stress paid off, because at the end of the competition, we walked away with first place. That's when I realized that failure isn't the opposite of success, it's part of the journey.

High school really changed the way I looked at programming. It wasn't just a hobby anymore. It became something I loved, something I wanted to use to actually build and solve problems in the real world. And that's why I want to continue this journey not just as a student tinkering with projects, but as someone ready to take it further, to study computer engineering and explore how technology can keep connecting ideas with impact.