Computing Is Just Learning, Debugging, or Burning!

After working for nearly six months on my project for the National Scientific Week competition, I found myself sitting the night before the event, fixing the final details and trying to make everything perfect—when suddenly, it exploded. Literally. It was one of the craziest moments of my life, but in a strange way, it reminded me of the long, unpredictable journey I've had with computers and technology.

It all began with my very first phone, a Lava Iris 600. As a 9-year-old, I saw it purely as a toy for games. But my dad had a different plan. Instead of letting me spend hours playing, he slowly introduced me to his own projects and encouraged me to explore something more meaningful. He knew I enjoyed math and computers, and he often reminded me that no matter what career I chose in the future, understanding both would always be valuable. So he became my first literacy sponsor—guiding me through small lessons and signing me up for courses.

That's how I discovered Scratch. At first, it was just dragging colorful blocks around the screen to make a little cat move. But to me, it felt like magic. For the first time, I could tell a computer what to do, and it actually listened. I created tiny projects, including a ridiculous game I named **Flappy Potato**. Even though the idea was silly, spending a week copying it and making it work made me feel like I had built something huge. Those early projects sparked my curiosity and pushed me to want more.

When I entered preparatory school, I decided to try "real" programming. I started with HTML, thinking it would be easy—just a few tags and suddenly you had a webpage. But once I attempted to recreate Twitter's old interface, I realized how challenging it actually was. HTML alone wasn't enough; I needed CSS to make it look good and JavaScript to make it function. That's when the struggle began. A single missing bracket or semicolon could destroy the entire layout. I remember refreshing my browser a hundred times only to find text floating everywhere and colors completely messed up.

I once yelled at the screen,
"Why won't you just work?!"
And in my head, the computer replied,
"Because you forgot a semicolon, genius."

That's when I realized programming was like playing chess against myself. Every move mattered, every mistake taught me something new, and each bug forced me to think more clearly. Instead of discouraging me, the errors made me want to understand what went wrong and how to fix it. That was the moment I started to actually feel like a programmer.

High school took everything to another level. I wasn't coding just for fun anymore—I was building real projects with purpose. I discovered microcontrollers like ESP boards and entered the world of robotics, where programming met math, physics, and engineering. The biggest project I ever worked on was **Roboworm**, a device created to help beginners in gardening and sustainability. Our goal was to design something that could literally "guide the soil."

I built a working prototype equipped with moisture, temperature, and light sensors. These sent data to an app and a website connected to a database. The system used AI to analyze the information and give simple recommendations—like which plant suited the soil or what fertilizer to use. It wasn't just hardware or software; it was both, plus AI, working together. Roboworm made me see how code could extend far beyond a screen and actually improve people's lives.

But with every project came the same enemy: **the Bug**. It always appeared at the worst possible moment. In preparatory school, a single bug could ruin my whole day. I remember preparing to show a small project to my principal so he could approve it for a competition. I had spent a week polishing it, and I was finally confident—until it crashed the moment I hit run. My heart dropped. But instead of panicking, I retraced my steps, found the missing line of code, fixed it, and managed to run it successfully in front of him. That moment taught me that bugs aren't roadblocks—they're tests.

By high school, I accepted that fixing errors was simply part of the job. You fix one, another appears, and you fix that too. Even when the Roboworm exploded the night before the competition, I understood that failure was not the opposite of success—it was part of the journey. The explosion happened because I had left the prototype running nonstop for a week. It used solar power during the day and a battery at night, and the system eventually overheated. When it blew up, my sibling and I were frozen in shock, too exhausted to react. But my parents woke up, calmed us down, and reminded us that we still had time to rebuild.

So we did. We stayed up again, repaired everything we could, rewired the circuit, and used the saved code to restore the system from scratch. And all those sleepless nights ended up paying off—we won **first place**.

High school changed how I viewed programming. It stopped being a hobby and became something I truly loved—something I want to use to build real solutions and solve real problems. That's why I want to continue this journey not just as a student working on passion projects, but as someone ready to pursue computer engineering and explore how technology can continue shaping ideas into impact.