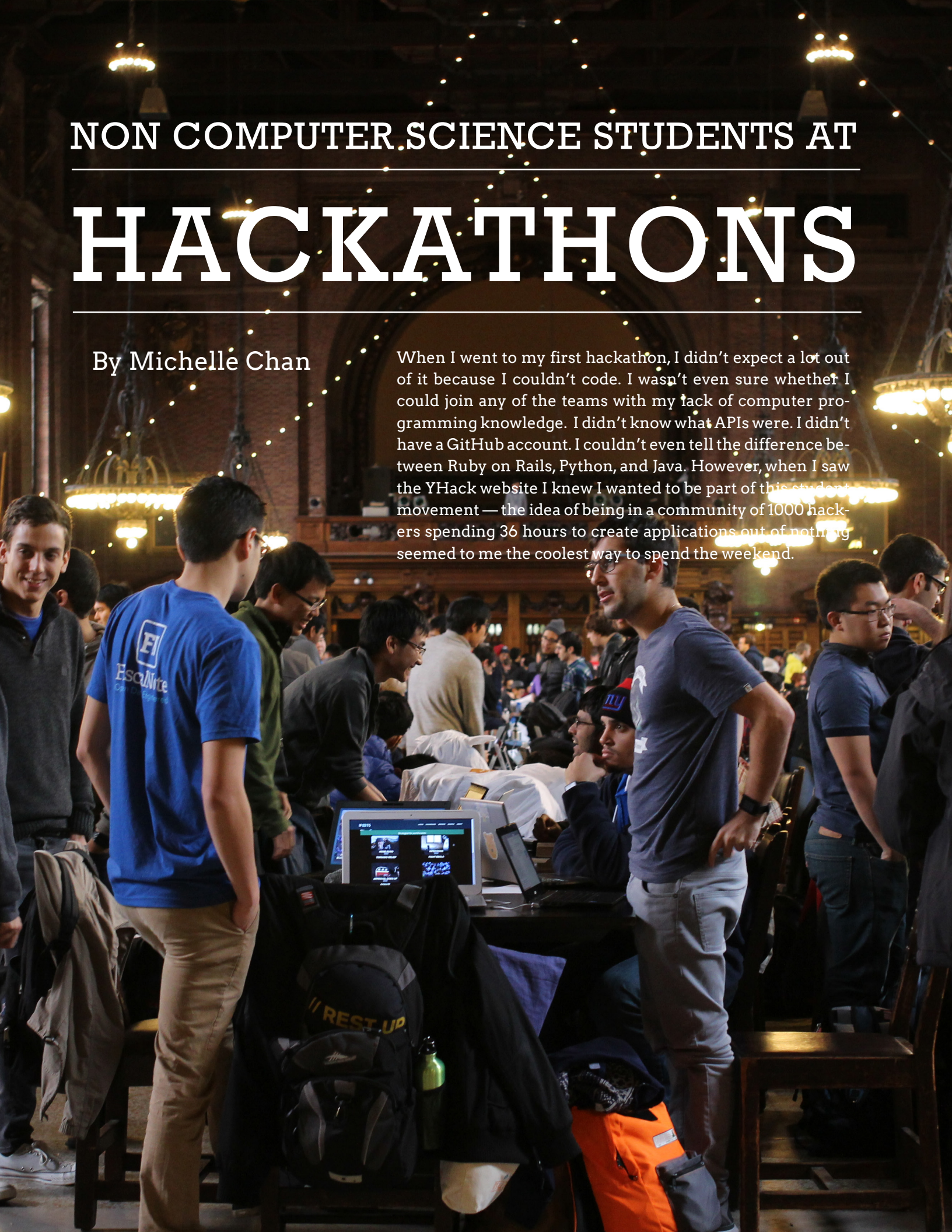
NON COMPUTER SCIENCE STUDENTS AT HACKATHONS

By Michelle Chan

When I went to my first hackathon, I didn't expect a lot out of it because I couldn't code. I wasn't even sure whether I could join any of the teams with my lack of computer programming knowledge. I didn't know what APIs were. I didn't have a GitHub account. I couldn't even tell the difference between Ruby on Rails, Python, and Java. However, when I saw the YHack website I knew I wanted to be part of this student movement — the idea of being in a community of 1000 hackers spending 36 hours to create applications out of nothing seemed to me the coolest way to spend the weekend.



The most common misconception about hackathons is that they are about breaking down systems. On the contrary, hackathons are about building things—robots, games, applications—with the help of mentors and company staff that teach you how to code and make use of their APIs and database. There is no such thing as losing a hackathon, because its true value is the experience. Whether you are an experienced programmer, a person who just took CS 101, or a complete beginner, you will still be able to gain a great sense of self worth after the 36 hours for being able to accomplish something you haven't done before.

I had the same concern that most non-CS people had about hackathons—if I don't code, how am I going to contribute, and what can I learn? My advice just to come and join a group. Although I couldn't contribute as significantly as other CS students, I could still offer help and learned many skills from them. For example, during the first hackathon, I joined a group of CS students from Purdue and the University of Waterloo who were making a web application that track Twitter feeds of specific topics and locations. I designed their logo, created a wireframe for them, and brainstormed ways they could pitch their product.

No matter how well the backend works, when you present the product to the judges, a significant part of the decision depends on the front-end design. Judges won't spend too much time testing whether everything in the backend works. Over 80% of the teams had bugs in the backend of their app. This really doesn't matter—teams still created amazing given the limited time. Even if you're not a coder, you can still contribute front-end design skills. If you're a business person, you can find a way to pitch effectively to the judges. If you don't have experience in either of these areas you can help with testing; an important process of product development.
Hackathons didn't make me proficient at coding overnight. Mentors teach programmers how to use databases like MongoDB and specific APIs. What hackathons did teach me, however, was how to talk about code, work with software engineers, and speak their language. Even though I didn't do the actual coding, I can explain steps of what each programmer did because I watched how they did it and asked them questions along the way. Most of them knew only half of what they needed to know to fulfill their tasks. They would look for answers on Stackoverflow when they had any questions, read API references to figure out how to use them, and hack it along the way to build something workable. I also learned about version control and Git and how programmers divide their work, communicate, and patch them up together in the end. Although I couldn't code the app myself, I could probably tell you how different components work in theory after reading their code and asking them questions.

While colleges are struggling to motivate computer science students, hackers voluntarily sacrifice their weekends to take on the more difficult challenges of programming. It's interesting how CS students skip classes all the time at school, but are willing to take 10 hours to travel to another state by bus over the weekend in order to forgo sleep to build web applications. It seems to many students that joining hackathons is the better and more modern way to computer science education, because the CS curriculum in most schools is either out-of-date or irrelevant to the real world. There are not that many schools that have classes like Harvard CS50. Lectures often lack a good feedback loop and an updated syllabus about the programming language that is used today.

Seeing this opportunity, the commercial world has created tech bootcamps that emphasize providing real-world. Some bootcamps even offered loans to students that are paid in the future by a portion of their future internship or full-time job salaries. Are hackathons and bootcamps the new path to the modern CS education?

I think the answer is no. There is a difference between computer science and technology. Hackathons and bootcamps teach students technology that is relevant to the present world, but college is still the best environment to learn algorithms, data structures, paradigms, and design structures that constitute good code. This is probably more difficult to learn than Ruby on Rails or Node.js, but it is equally important because it teaches you how to write good code even when technology changes in the future. There are many situations in which startups hired beginner developers to design their apps and ended up creating workable products that are a mess in the backend, making it a pain if anything has to be modified by another developer. There are still a lot of things to learn about choosing the right data structures, writivng readable and testable code, avoiding duplicate code, and making optimizations that reduce runtime—all of which could hardly be touched by hackathons and bootcamps given the limited amount of time. They should act as complements to the CS education, and could not really replace a CS degree.

For students, like me, who can't code, hackathons are really a great way to understand how tech products work and how they are built. Who knows, you may even learn you love CS, and choose to study it in college.