Hi there! I'm happy you're here. Before we start talking about programming, let's take a minute to reflect on why we're here and what lies ahead. This chapter isn't going to be very technical at all; these are just some things I wish someone had told me when I first got into programming. Hopefully you find it a little bit enlightening.

What is computer science?

First things first. In this book we're going to learn how to write computer programs in a language called Python. But this book is not about Python. It's about computer science. What's the difference? Think of the distinction between learning to write good essays, versus learning the English language. Writing a good essay is a creative process that requires a lot of practice. It is also a useful, valuable skill. Learning the English language is important if you want to write good essays, but just because you're fluent in English doesn't mean you're a good author. Similarly, though it is wise to become fluent in a programming language like Python, it is not the case that knowing Python makes you a good computer scientist. So while I am going to teach you Python, that is not my primary goal. Rather, I hope to teach you how to write good programs. With that in mind, please don't stress out too much about the semantics of the Python language. Don't focus too much on terminology either. That's not what's important. Instead focus on the big ideas, like how to think analytically about different types of problems and solve them. That's what computer science is about.

What is a computer scientist?

Many of my students walk into class the first day, with a preconceived idea of what a computer scientist looks like and acts like. That's understandable, since there are so many baseless stereotypes about this field. The good news is that all the stereotypes are wrong. Computer scientists are just ... people. They come from every ethnicity, every gender, every background, and every walk of life.

Diversity is not simply beneficial to the learning environment. It's a necessity, and it is valued here. You are valued here. Let's talk about making yourself feel welcome studying computer science.

Stereotype threat

stereotypes about themselves or groups that they identify with.

Here's a classic example. Many standardized tests, including the SAT, ask students to answer some background questions before taking the exam. Often, one of the background questions is "What is your gender?" and there are two options for "Male" or "Female". This question subconsciously primes the students to think about their gender, and the stereotypes associated with it. If they're also told the test is going to measure mathematical ability, then the female students in the room feel pressured to perform well, for fear of confirming bad stereotypes about women in math. Now, rather than being 100% focused on the test, they're busy stressing over the stereotypes they've heard about themselves. Since they're not as focused as they could be, they don't perform their best. Repeated studies have shown that if the exam simply asks for background information at the end of the assessment instead of the beginning, then underrepresented students perform significantly better. This makes sense, since they're not primed beforehand to worry about reaffirming negative stereotypes.

This effect is extremely common, and has probably affected everyone you know — whether they're aware of it or not — because most people have at least one social identity that's negatively stereotyped. The good news is that we can fight it. In fact, just being aware of stereotype threat makes you less susceptible to it. The computer science department also fights stereotype threat by striving to create a safe and comfortable learning environment. For instance, I teach a class where I train TAs and tutors how to teach effectively; this involves a few lessons on promoting inclusive stereotype-free classrooms. I'm also involved with programs like CS Scholars and CS Kickstart which provide a sense of community to underrepresented students and female students who are new to computer science.

Imposter syndrome

Imposter syndrome is a related psychosocial effect, especially prevalent in high-achieving people who surround themselves with other high-achieving people. That makes it pretty common at top universities like UC Berkeley. It's when you feel inadequate, even though there's evidence that you're in fact a successful and capable individual. People suffering from imposter syndrome tend to dismiss their achievements as luck, or the result of portraying themselves to others as more accomplished than they really are. Imposter syndrome can be worse for underrepresented minorities, due to some destructive comments that their success is merely due to affirmative action. (Clearly this is not the case because there is

and ethnicities are hidden from admissions officers.)

I had a case of imposter syndrome my first semester here. I did the worst thing you could possibly do when you enter a new environment: I compared myself with the people around me. A surprising number of them were local to the Bay Area and spent their lives a stone's throw from Silicon Valley. Some of my peers had been programming since high school or even before then, and they all seemed to have really rich successful families. They were also 2 or 3 years older than me on average, which was intimidating to boot. I did not fit the same description at all. I was coming from Bolivia via the Mojave Desert, and though I was interested in tech I had never seen the industry up close or even thought about studying computer science. In fact, I came to UC Berkeley intending to major in biology and I only took my first programming class on a whim because I'd heard good things about it. When it seems like you're the outsider, it can be easy to wonder whether you really belong here. Well, you do.

It's important to remember that imposter syndrome is not reality. You are a smart, capable person, and speaking as an experienced computer science educator, I believe in you. But what matters even more is that you believe in yourself. If you have any reservations at all, reflect on the reasons why you feel like you don't belong and talk them through with someone who you trust. Think about your values, your goals, and how you this class will help you achieve those goals. Most of all, remember that your accomplishments aren't just luck; they're proof of how hard-working and intelligent you are. (PS: Keep in mind that by and large, folks' concerns about belonging tend to decrease over time. I've seen it in myself, and my students.)

A note on prior experience

Speaking about belonging, students commonly come to me with concerns that they're just starting computer science, whereas a lot their peers have years of experience from taking AP classes in high school. First of all, I want to make a distinction here. I never took AP classes, but from what I gather, they don't seem to teach computer science very well. It overlaps with perhaps the first six or seven chapters of this book, and that's it. The AP curriculum seems to focus more on syntax than actual problem-solving techniques. Using our earlier analogy, it's like learning English instead of learning how to write good literature.

If you have no prior experience, that means you're not far behind. You're probably a bit lost, and that's okay because you've never seen computer code before or what

mastery of the AP curriculum I don't think you can claim to know computer science quite yet.

Also keep in mind that this is an *introductory* computer science class. It was designed so that you can take it with or without past experience. Yes, it's nice to have a little bit of a head start, but it's not necessary or even expected. This class is a learning experience for everyone taking it, and although you may start at slightly different places you are all on the same path together. It doesn't matter where other folks started out, because once upon a time they were just as clueless as anyone else. It only matters where you started, and the progress you've made since then. That's the only metric that makes sense, because you're here to learn and learning is about growing beyond your own limits, not somebody else's.

Fixed and growth mindsets

While we're on the topic, what does it mean to grow beyond your limits? Isn't that impossible, by definition? It depends on your mindset. This is another well-researched area of education theory, pioneered by Stanford psychology professor Carol Dweck, Ph.D.

A fixed mindset refers to the idea that everyone has an innate level of intelligence, which cannot change. Under a fixed mindset, challenges are bad. Even worse, effort is bad. Naturally smart people ace their classes without putting in any effort, while the rest of us study hard to get good grades. If you have to put in effort, then you must not be smart. By corollary, people with a fixed mindset tend to give up pretty easily. After all, if you meet a difficult problem and you're convinced you can't overcome it, then there's not much else you can do.

A growth mindset refers to the idea that intelligence is dynamic. Like using your muscles makes you stronger, using your brain makes you smarter. Under a growth mindset, challenges are good and so is effort. Putting in effort means that you're learning and growing and becoming smarter. If you don't put in effort, then you're wasting away.

Here's the key difference. People with a fixed mindset are concerned about *looking smart*. They want to seem to others as if everything comes naturally to them, even though that's not the case. Meanwhile, people with a growth mindset are concerned about *learning*. They want to become smarter, so they study hard and actively seek new learning experiences. Adding to that, people with a fixed mindset tend to be more competitive. They feel threatened by others' success, because it's hard to look

great way to learn.

As you might guess, people with a growth mindset tend to perform significantly better than those with a fixed mindset. Sounds like you'd rather have a growth mindset, right? Lucky for you, the fact that a growth mindset exists implies that you can adopt one. Do these things, and the right mindset will follow:

- Don't be deterred by adversity.
- Seek challenges because they help you learn.
- Don't be afraid to help others, or seek help from them.
- Learn from criticism, rather than ignoring it.

So after all this discussion, what is a computer scientist? Foremost, a computer scientist is someone who values diversity. It's someone who recognizes their own potential. It's someone who was once a beginner, and who strives to help others learn. It's someone with a growth mindset. It's you.

Studying tips

Being a successful student myself, and having taught hundreds more, I've noticed some study habits that seem to work better than others. Let's talk about what you can do to maximize your learning in this class.

Do the reading

Reading an academic text is not like reading a novel. It's an active process, not passive. There will be confusion, but it only gets worse if you ignore it. If you feel like something doesn't make sense, then go back to the last place you had full confidence in your understanding and re-read from there. When I'm studying I usually make quite a few passes over a chapter or a section before I really get it. There's a difference between *reading* the words and *understanding* them.

Your role isn't to absorb information, but instead to question it. Don't take anything I say for granted. Ask yourself why it makes sense, or why it doesn't make sense. Contemplate how each concept relates to what you already know, and how to make new things intuitive. When we talk about specific examples, try predicting what happens next; don't wait for me to tell you. You should perpetually be asking "Why?" or "What if?". I'll try my best to answer these questions, but it's hard because I'm not sitting right there next to you. Luckily, you have everything you need to answer them

It may also help to read in chunks. Active reading takes a lot of brainpower, and some of these chapters are pretty long. Make sure you're focused and attentive when you sit down to study.

Consider taking notes, too. It's not really necessary because you can always refer back to the chapter, and everyone has a different learning process so do whatever works for you. Personally I find note-taking is a good way to keep myself reading actively instead of passively. Whatever ideas I write down, I make sure I can explain them thoroughly including their nuances and what-ifs.

Do the practice

When you do a practice question, try it on your own first. Don't peek at the solution or work with a friend. If you're stuck, then come back to it later, think about it some more, and try it a few different ways. Sometimes I only get a problem after scrapping my first three or four attempts. It may also help to refer back to the chapter, to refresh your understanding of the problem-solving process. While you're working, I highly encourage you to make notes in the margin if you're unsure about something, or if you're even briefly stuck. This will help you review your thought process after you finish.

When you think you have the solution, try it out on the provided test cases to see if it works. To practice your debugging skills, you should think through the test cases on your own, rather than using the computer. If you're still not so sure about your program, ask yourself where your uncertainty is coming from. Only look at the solution when you're 100% confident in your answer.

Sometimes you will get problems wrong. The first step is to understand the correct solution. Reason through it and make sure you see how it works. Refer back to the chapter if necessary. Then, more importantly, take 5 to 10 minutes to reflect on the problem:

- Where did your reasoning start deviating from the solution?
- Did you uncover any of your misconceptions while working?
- If it took you a while to solve, why? What parts did you get stuck on?
- Do you need to review anything?

Then identify how you're going to modify your problem-solving technique so that you get similar questions right in the future. Personally, I also write down on a piece of

same problem after a few days, once you've forgotten the solution, to make sure you really learned how to do it. (It might also help to keep a catalog of the problems you're going to retry, and what day you first attempted them.)

Remember, you're probably never going to see the same problem again, after you do it. That means it doesn't really matter whether you understand the solution that one specific problem. What matters is your understanding of the problem-solving process. That's why it's *so important* to reflect on your technique. The practice problems are a guide, not the end goal. Use them as an opportunity to refine your thought process, rather than an opportunity to learn the solutions to specific problems.

Take care of yourself

I've seen too many students make this mistake, so let me make it explicit right now. It is a bad idea to study all night long. In general, it is a bad idea to work all night long under any circumstances. It is a bad idea to skip meals. It is a bad idea to fuel yourself on energy drinks. It is a bad idea to neglect your health or cleanliness. Why? Because eating, sleeping, and basic hygiene are necessary to function properly. If you neglect any of them, then you're not functioning properly. And when you're not functioning properly, you can't learn. I've seen firsthand what it does to students and their grades, when they don't take care of themselves. It's not good.

Stick to a schedule. That means you go to bed and wake up at a consistent time every day. Even if you sleep for eight hours per night, it can be pretty damaging if you don't sleep for the same eight hours every night. Set a bed time, and set a consistent alarm. Stick to your schedule on the weekends too.

Set goals

To keep yourself on track, you should also set concrete actionable goals. Don't pick vague goals, like "study a lot" or "do really well on the test", because then it's a judgement call whether you met your goal or not. Goals like "get an A in the class" are also vague, because it's not immediately clear how to work towards that goal in a way that you can measure. Don't pick unrealistic goals either, like "study 12 hours per day, 7 days per week", because that makes it less likely you're actually going to stick to your plan. Goals should be measurable, like spending a certain amount of time studying, or finishing a certain number of chapters or practice problems every

Discipline is important too. Don't waste time on YouTube, Facebook, or chatting with friends when you are meant to be working. That doesn't mean you have to become a monk. Obviously, do whatever you want in your free time. (In fact, do whatever you want, period. I'm not the boss of you.) But my advice is to identify the things that distract you when you're working, and take action to prevent yourself from losing sight of your goals. I used to have a plug-in installed in my web browser, which would put a 3 minute delay on YouTube and any Google queries about dinosaurs. That way I could still access these sites if I needed to, but I wouldn't be tempted to visit them otherwise. I also recommend you find a quiet place to study, and tell your friends not to disturb you when you're at work.

Closing remarks

Computer science can be a lot of fun. It's an extremely versatile field, and it empowers you to tackle problems in just about any other field that interests you. For me, it has also been a growing experience and an invaluable source of life lessons. Has it been hard too? Oh my god yes. But I like to think the nice parts outweigh the stressful parts. Remember it's okay to make mistakes as long as you learn from them, and remember you are valued in this community.

Learn. Question. Have fun. Welcome to computer science.