Hi there. Welcome to *Planning for Computer Science*. This paper encompasses all of the most important lessons I've learned while completing the CS major. First, I'll start by answering some common questions:

I like computers. Should I become a computer scientist?

Be careful. Just because you like computers does not make you a person well-suited to CS. This is a very common preconception freshmen come in with. Once they realize what CS is really about, their interest can quickly fade. I can't tell you how many kids walked in who thought computers were all Linus Tech Tips and fun times. Well, it's not like that at all. It's a lot of work; this is a degree you have to earn. There are three big fields of computing: CS, CIS, and IT.

Well, what is CS then?

Computer Science is the study of theory behind how computers work. It involves lots of theory and mid-level math. To learn about theory, you will be programming a lot. You will also have to write proofs and programs to prove algorithms and theories. People frequently associate it with programming and coders, but the reality is that computer scientists are kind of on a different level than that. CS majors gain an understanding behind the theory, and as a result are much more disciplined programmers. CS majors learn not only what to do, but why to do it. They are taught to make more optimized programs that are easier to extend and adapt.

That sounds so academic. What if I don't care about all the theory and how/why stuff works and just want to program?

CS is definitely not for you. Choose CIS or skip out on college entirely, because even CIS has some theory in it, albeit more business focused. If this is your attitude, you should abandon all hope of systems-programming, too because the theories will haunt you forever there. However, if you are coding for a framework you should be fine.

If most CS majors become programmers, why do they have to learn so much stuff they'll never use?

Because that's just how it is. Getting the degree doesn't necessarily mean you know how to do all the stuff, it just means you ***can*** do all the stuff. It also means you are of higher intelligence and work ethic than the average Joe. It means you are willing to take a risk for it with how much time and money you spent to get it. It's a shiny badge that is in high demand. If you don't like it, then oh well, too bad.

Should I do Computer Science, Computer Information Systems, or Information Technology?

It depends. Assuming you can do all that's required, then Computer Science. A Computer Science degree is generally considered the best computing-related degree of these three.

CIS is considered "CS-lite". It's a CS degree that's watered down with business classes and doesn't cover some of the higher-level math. With it, you can do IT or CS, but it depends what classes you take to emphasize your field of interest. For example, CIS graduates need to take a minimum of a couple coding classes, however there is ample opportunity to take more. They can also decide to take more business-management focused classes, instead. Long story short, CIS is the easier degree and is recommended for people who aren't good at math or programming and still want to work in computers. Employers know this and generally seek out CS majors more or pay CIS majors less. It's generally agreed that a Masters in CIS is about equivalent to a Bachelor's in CS, so there is opportunity down the line to close the gap if desired, but this will cost a lot more money.

IT is considered "CIS-lite". It waters down CS even more with more business classes and even less math. IT are going to probably be doing IT, however they can learn some coding if they wish. Otherwise, coding is almost nonexistent in this major. It's a business-management degree with an emphasis on technology in business. It's definitely the easiest of the three, but employers know this and pay accordingly. It's still far better than a general-business degree, though.

There is also an IS major, but the emphasis is somewhat different to IT. This emphasis varies by university so it's hard to pin down.

With CS, you can do almost anything in computers, perhaps aside from computer engineering unless you take some engineering courses not required for a CS major to specialize. Unfortunately, CS is a much more demanding major. It requires higher-level math skills, critical thinking and analysis skills, and troubleshooting abilities. Depending on the school, Calculus I and some discrete-math courses are usually the minimum. A lot of people can't do this level of math. You'll need to be able to look at coding solutions that are flawed and find ways to improve them. Your code also likely won't work the first time you write it, so you'll need to research and come back to it. This all takes a lot of patience, and frankly most people don't have that. You really have to be in CS because you like it. Another issue is how "front-loaded" the major is. The initial major coursework will seem to be the most difficult in the long run. This is because programming takes a totally different line of thinking than most people are accustomed to, which takes some getting used to. The major actually gets EASIER as you advance. You will often have to learn additional material outside of class because there isn't enough class time to cover all the course material. Even in the electives, it's usually expected that class time is used for introducing concepts and you should go home and learn the rest of it. Get ready to be buried in programming documentation and Stack Overflow! Overall, CS is a grind to get through, but once you get through, you're on top of the world.

How is IS different from CIS and IT?

I actually had to do some research to answer this question. I've copied the descriptions from my college. (IS majors) *"study the use of computers in organizing and processing information"*. (CIS) *"is a hybrid of computer science and information systems, and incorporates topics from both disciplines, providing flexibility in tailoring a graduate degree"*. (IT majors) *"emphasize the technology needed to convey information needed in organizations"*.

Long story short, IS and IT are more business management, planning, and development focused, while CIS is more computer-programming and interaction focused.

Why are there so many major disciplines in computers?

To an extent, there is a need for more disciplines, but it seems like it's getting much more complicated in the past few years. Just recently, Cybersecurity has been the hot new major. (Cybersecurity) *"majors study the techniques used to protect the confidentiality, integrity, and availability of data, information, and computer systems against attacks"*.

One reason is because there is simply too much computer stuff to shove into a few majors.

Another reason is because not everyone can do the difficult computer majors, so they need a major that more people are capable of completing.

Another reason is that computers are the future and the field is evolving to encompass new demands, such as cybersecurity.

The last reason I can think of is that people want to study different things and the university will offer them if there's a decent demand to study them. Not everyone is out to make the most money, some people just want to do what they want to do.

A school offers a Software Engineering degree, is this the same as Computer Science?

It could be, it's hard to say. It depends on the school. Software Engineering is a degree typically offered by business schools, however the coursework can vary from being identical to being more software focused and less theory-heavy. One thing is for certain, though: CS is more widely recognized and is generally preferred over SE, although not necessarily. It sounds silly, but yes having your piece of paper say SE instead of CS could affect your chances. This is something worth thinking about when choosing your school.

Do I have to go to college to be a computer scientist or software developer?

To be a computer scientist? In my opinion, yes. People will not take you seriously if you don't have CS credentials. To be a software developer? No, you don't, actually. However, you will have to compensate by doing a lot more footwork and making a lot more phone calls and emails. Not having a degree and experience definitely hurts your chances, however it can still work. Programming can all be learned through Google. However, having the appropriate certifications and portfolio to back up your self-claimed qualifications is critical. Claiming to know a bunch of stuff with nothing to show for it won't get you anywhere. If it's worth anything, the vast majority of top-tier programmers at companies I've visited did not have CS degrees; they usually had either no degree or totally unrelated ones. This trend actually continues across many sectors other than computing! They all worked their way into those positions, picking up skills as they went, however I do realize that the ability to work into these positions may no longer hold true. It might seem unfair that the people interviewing you have no idea about the technical questions they are asking you about, and yet they expect you to know the answer and have the degree! Unfortunately, it's just how it is. The truth is, once you have 10+ years of experience, most people don't care if you have a degree or not! While extremely helpful, a degree is merely a foundation for a career, it does not define it.

Computer Science sounds intensive. I don't want to go to school on Fridays and I want time to hang out with friends and make the "best" of the college experience.

This is a woefully short-sighted desire. The truth is that you will not be able to get Fridays off totally, but you will almost always have Friday nights off. I urge you to reconsider your life-potential after college. Indeed, I too know people who made the choice to go with a lazy major (ex: General Business) in exchange for the party life. I will not lie, they are enjoying themselves and think they've made the right choice. However, I also will not lie that people who I know that graduated in these majors are either back in college for their Master's because they aren't competitive with a mere Bachelor's in General Business or they are working the same dead-end jobs they would have worked if they didn't even go to college in the first place. Even better, they have tens of thousands of dollars of debt they hope to avoid forever. There is a sum game going on here; I can't say the value of that sum for everyone because everyone is different. Yes, in CS life will suck for 4-5 years, but after you come out the other side you will be eternally grateful you did it. High pay, flexible hours, and little physical exertion means you will have plenty of stamina, money, and time for everything you want to do in life. Have the discipline to buckle down now and you can celebrate later.

Do I need to be good at math to be a programmer or computer scientist?

To be a programmer? Not really. To be a computer scientist? Yes.

Ideally, programmers should be good at math, too but it's not as important because programmers are going to be concerned about the practical nature of math to solve problems. You will rarely be using anything passing Algebra II levels of complexity. Computer scientists on the other hand will be using math in the studies of computing theory. Math forms the basis for their understanding of how to write efficient programs and using a computer to its fullest. CS majors will need to complete Calculus I at the very least and also take Discrete Structures I and II; you don't have to be great at math but you do need to be decent at it.

Computer Science sounds cool. How do I get there?

This is my recommended path to graduation in CS. Due to the front loaded nature of the CS major, I suggest mixing electives and cores with the major coursework to distribute a more consistent workload. Some assumptions are made, of course. You probably won't get a schedule exactly like this, but there is some flexibility gained by using this distributed-type scheduling where you can swap cores as needed. I do not recommend getting more than one course w/Lab a semester; it makes things much more difficult. I've also given my perception of difficulty for each semester on a scale of least (1) to most (5) difficult, although take this with a grain of salt because difficulty is subjective. I've determined the difficulty based on my perceptions AT THE TIME instead of in hindsight. You may notice I don't have any rated less than 3, and this is because I would say Computer Science is one of the toughest majors other than Engineering majors. I've managed to distribute the workload to have only one truly hard semester. It won't be easy, but rarely should you find yourself overwhelmed.

1 – Easy
2 – Busy-work
3 – Average
4 – Challenging
5 – Hard

Semester 1 (Difficulty: 4)

- Calculus I/STATS I
- Computer Science I (w/Lab)
- Strategies in Writing
- Core course

Semester 2 (Difficulty: 4)

- Calculus I/STATS I
- Computer Science II (w/Lab)
- Public-Speaking
- Core course

Semester 3 (Difficulty: 3)

- Discrete-Structures I
- Intro to Linux
- Core Course
- Core course

Semester 4 (Difficulty: 3)

- Discrete-Structures II
- Business Writing
- Core course
- Core course

Apply to secondary admission.

Semester 5 (Difficulty: 4)

- Structure of Programming Languages
- Internship Prep
- Data Communications (w/Lab)
- Core/Elective

Apply to Internships

Semester 6 (Difficulty: 4)

- Computer Architecture (w/Lab)
- Databases
- Core/Elective
- Core/Elective

Spring/Summer Internship

Semester 7 (Difficulty: 5)

- Intro to Software Engineering
- Data Structures and Algorithms
- Core/Elective
- Core/Elective

Apply to jobs

Semester 8 (Difficulty: 4)

- Operating System Concepts (w/Lab)
- Capstone
- Core/Elective
- Core/Elective

Migrate to workforce

Recommended CS Required Cores:

- Calculus II or Statistical Computing
- General Biology I/II or General Physics I/II

Recommended CS Electives:

- Web App Development
- Mobile App Development
- Artificial Intelligence
- Information Management Science
- Data Mining
- System Security
- Information Assurance
- Cyber Forensics

Other stats you should consider:

- Most jobs filter out all CS grads with a sub-3.0 GPA, so try to have at least a 3.0. I recommend a 3.5 or higher if you want to stand-out, because it certainly got recruiters talking to me about it.
- CS has a high dropout rate compared to most majors, although nowhere near as high as engineering. From my experience, the dropout rate was around 15% with the vast majority of people dropping out in the first year.
- CS majors occupy a very disproportionately high amount of high-paying jobs compared to most majors.
- CS is almost entirely male-dominated and the field can make women feel uncomfortable or unwelcome as a result. I've also heard of some sexism happening although I've never seen it. Just something to be aware of. The vast majority of women in computing are IT, IS, or CIS majors.
- CS is very theory-heavy, so if practicality matters more to you than theory maybe you should choose CIS, instead.

What if I don't make it?

I would suggest switching to CIS or worst-case IT if things get dicey. They are much easier, yet still open many doors in computing careers.

How do I afford going to college to do this?

Read my paper on ***Planning Your Career***. Long story short: Scholarships, grants, and federal loans. Most of the people in my major have spent 5 or more years going to college and are tens of thousands of dollars in debt. I knew one person who was in their ninth year with over 100k of debt and another who was in their twelfth with over 200k of debt. Don't dawdle in college, it gets expensive. Be sure you know what you want to do BEFORE you go. I was VERY lucky and got enough scholarships to cover my whole degree, but from my observations this is not the standard.

How do I get an internship or do I really need one?

Many schools require internships for their computer science majors to graduate, but if it isn't required do you really need one? I would say...no, not really. Of the people I knew that didn't get internships, they still all got jobs after graduation. It was seemingly easier for those who had internships, but it certainly isn't required.

An internship makes you look much more employable right after graduation. Without it, you're asking businesses to take a risk on you. If there's anything I've learned about doing business, it's that it's really hard to convince people to take unnecessary risks, especially when money and time are involved. A computer science degree holder is in quite high demand so I wouldn't worry about it too much, but just know that it definitely helps.

How do you get one though? I would strongly advise asking around in your family and friends about any potential opportunities. See if anyone knows anybody and if they are willing to vouch for you. These sort of jobs are much easier to get because your competition factor will be much lower. Competition is what makes getting employed difficult, and if you can eliminate that it's much easier. Otherwise, online job applications are how things are going nowadays. Just be aware that the competition for these posted positions are much higher, meaning you likely will hit a lot of brick walls and become frustrated. However, it can definitely work; it's how I got my first internship. Then again, my stats were in the top 10% of candidates so that's worth considering.

Why is it frustrating? Well, even for me it took about 10 interviews before I got an offer. I sent out at least 40 applications over the course of 5 months. At the end of the day, you might have to get an internship, but I strongly advise against being desperate/unpaid. It will look bad on your resume, CS interns deserve to be paid. A couple months later, I interviewed at a small local company. It went okay, the interview wasn't the greatest one I've ever had so I assumed I dropped the ball. Turns out, they offered me the internship with many additional benefits including: decent initial pay, increased pay after a few weeks, paying for my last semester of college (redundant for me, as mine was already paid for), and a job after graduation (if I worked out okay)! I can't promise it will work out for everyone, but if you try early on and put out a good quality resume/cover letter to employers, someone is bound to pick you up eventually.

How many languages should I know?

This is one of those "yes" questions. In all seriousness, it doesn't really matter how MANY you know well. From my experience, you should demonstrate flexibility and adaptability. This means you should demonstrate proficiency in one of each of the following:

- Object-Oriented, including framework/non-framework languages (Java, C#, C++, Python, etc)
- Non-Object-Oriented (I recommend C, Assembly is another but it's rarely used in CS)
- Internet/Web-framework (ECMAScript, PUG, SASS, CSS, React, Angular, etc)

It's not fair to expect people to know every language, however knowing more is always better. At the very least, this range of flexibility demonstrates you can learn any other language needed to complete a job. I knew all of these languages when I graduated, so it's very likely as you progress through your major this requirement will be satisfied anyway.

Tips to Graduate:

- Get started on programming assignments as soon as they are assigned. Work on them for an hour or two everyday as needed to get them done a couple days early so you have ample time to ask questions and compare with other students.
- Do not bring preconceptions about programming when it comes to learning new languages. Ex: Don't try to learn C with Java preconceptions, you'll get very frustrated. Different languages exist for a reason; they have different purposes and strengths.
- Go to class as much as you can. Take notes as needed, sometimes you do and sometimes you don't.
- Schedule as soon as availability opens.

Common Pitfalls/Misconceptions:

- Procrastinating homework/projects
- Assuming the internship will "just happen"
- Code that compiles works perfectly
- Code that is tested works perfectly
- All the languages you will need to complete an assignment will be covered in class.
- Why is C not like Java/Python?? (Answer: Because C isn't Java/Python!!)
- If I know C++, I know C. (No, you don't! C requires totally different techniques.)
- C/C++ is an acceptable way to abbreviate combined experience. (Nope! C++ and C techniques are vastly different and combining them shows a lack of understanding this.)
- JavaScript (ECMAScript) is based on Java and is very similar. (Almost totally unrelated, many fundamental differences, although there are admittedly some similarities as there are when comparing many different languages to each other)

- A CS degree guarantees you a job. (It doesn't, and in fact most of the top-tier programmers at companies I visited either had no degrees or totally unrelated ones. Once you have 10+ years of experience, most people don't care if you have a degree or not!)

Plagiarism: Read This!!

Plagiarism is the prohibited sharing or copying of work from a disallowed resource. In certain fields such as writing, it's only plagiarism if the work referenced is not cited.

Professors recognize plagiarism either by having a deja-vu moment (*"Hmm, where have I seen this code before?"*), or seeing sheer implausibility that the student in question actually made the submission (*"There's no way this kid wrote this!"*). Sometimes, they may also have additional tools to detect plagiarism such as TurnItIn, which check papers and essays for similarity to other sources online and fellow classmates.

Due to the advancements of the Information Age, it's become extremely easy to plagiarize.

- **DO NOT SHARE YOUR CODE, PAPERS, OR ASSIGNMENTS WITH ANYONE!! NOT EVEN YOUR BEST FRIENDS.** Most universities have a zero-tolerance policy regarding plagiarism, and both the cheater and the supplier are held equally responsible. This is considered an academic integrity violation, and can result in a zero for the course or possibly expulsion. Keep your documents to yourself. People will do anything to pass a course including sacking your whole degree. Trust no one! Document theft usually happens in the following ways:

- You give the code/paper to someone under the promise/assumption they will use it as a "guide" or "inspiration". They then take this code/paper and copy-paste it to some extent.
- You host the code/paper on a repository that they can access, which they then take and copy-paste to some extent.
- You leave your computer unsupervised and someone downloads the code/paper to a thumb drive.
- You are unprotective of your code/paper and someone peaks over your shoulder to copy it via transcription.
- You leave your computer unsupervised and the computer gets stolen.

Usually, violations are not escalated to disciplinary action. Universities leave it to a professor's discretion to report violations. The professor will usually perform their own investigation first to determine their preferred course of action. They will solve the issue to their standards if they want. If the violation is disturbing enough, it may be escalated to the disciplinary board.

The disciplinary board is quite unforgiving. Typically, some leniency is issued to the supplier if the code was straight up stolen from a restricted resource (i.e. stolen computer, hacked repository, unauthorized download, etc). However, this is not certain. Because most universities don't want to let plagiarism off easy, they usually punish everyone involved, but the punishments vary from case to

case. If you give them the code, you will be punished equally. If it's stolen from you, you probably won't be punished. If it was taken by you being careless and leaving your code unguarded, there may be a slap on the wrist such as a zero for the assignment in question.

What if I explain I gave them the code not to copy but to use as a guide?

The intention of your violation is irrelevant. The fact is that you admittedly supplied code when you were not allowed to. Even if you explicitly denied them permission to copy any of your code, it doesn't matter. The intention is not the basis of your punishment, the violation is. Do not put your degree in someone else's hands, just don't do it! People are out there to exploit your life to further their own. It's a risk that's not worth taking.

What if I publicize my work after the assignment is due?

I wouldn't recommend this. Not every section has the same due date for an assignment. Also, professors regularly give extensions on assignments. Students may still be able to copy your code for their own benefit. You could be held responsible.

What if I want to publicize my assignments after the course is over?

It's usually acceptable to post your projects publically after the course is over, but be sure to consult your school's policy on this. You can still be held responsible, even after the course is over! Students from the next semester may find your repository and copy your code.

Why is the supplier blamed?

The supplier facilitates the cheating. If the supplier hadn't given their code, the cheating would have never occurred in the first place. At the end of the day, there are always going to be people struggling and desperate to pass. When push comes to shove, people will be willing to cheat. It's an issue students shouldn't facilitate.

Think of drug dealers. There are always going to be addicts who simply can't control themselves and are desperate to get by. We consider drug dealers even more guilty than the substance abusers because they facilitate the issues to continue. If we didn't have drug dealers, it would be nearly impossible to get drugs.

It's impossible to eliminate cheaters, because anyone can be one if desperate. If we eliminate the opportunities to be a cheater, then cheating can't happen. This is the logic of disciplining suppliers.

Interview/Internship Tips:

- Have personal projects to talk about. They won't ask about them necessarily, but sometimes they do and if you say you don't have any, you aren't getting the position.
- Show passion about the field.

- NETWORK! Make nepotism work for you instead of against you. People frequently beat me in interviews because they were friends with someone on the team at the company. Sometimes, these candidates didn't even have a degree!!
- Have a kick-ass resume. Make your own, buy a template, or pay someone to do it, whatever it takes. A well-laid out resume is immediately eye catching regardless of what's on it. Avoid using colors on CS resumes, it just looks goofy.
- Long-term volunteer roles look really good on the resume!
- Dress business professional as needed. Some businesses may ask you not to, but otherwise you have to. If you come looking casual inappropriately, it'll seem like you don't care at all.
- Sometimes interview questions, technical questions, or whiteboard questions do not have right answers. Sometimes, they just want to see your problem solving abilities.
- Show enthusiasm, a sense of humbleness, and humor.
- Your degree will usually get your foot in the door for an interview, but unless you have experiences to discuss with the recruiters they will quickly lose interest.
- Put your projects on GitHub. You should have readmes for every project describing what the program does, and the code should be decently commented. Put your GitHub link on your resume so recruiters can see your skills for themselves.
- You absolutely NEED a LinkedIn. At least 30 connections, fully filled out, no exceptions. Put the link on your resume. This is industry standard practice and is essentially REQUIRED.
- Cater your resume to the type of position you are applying for. Ex: Do not send a software focused resume to an IT opening, IT almost NEVER cares about programming abilities.
- The number of languages you know doesn't really matter. However, you should demonstrate proficiency in at least one of each of the following: Object-Oriented (Java, C#, Python, C++, etc), Non-Object Oriented (such as C), and one Web-Framework. (ECMAScript/JavaScript/Node.js). This shows how well you can adapt to any language.
- For the languages you do know, know them WELL. Just glancing and understanding the syntax of a language does not mean you KNOW it.
- Practice algorithms! Technical/whiteboard interviews will test you.
- It's usually at least two interviews before you get a position. The first interview serves as a character/behavioral assessment, and the second serves as a technical assessment. Sometimes, there will be a third interview to serve as a tiebreaker, usually filled with whiteboard questions.
- If you are rejected, don't ponder it too much. Businesses have many factors to consider when choosing between candidates, and they usually give generic rejection reasons (ex: not a good fit) instead of being forthcoming with their reasons. If they give you something specific to improve, then work on it.
- Bring follow-up questions to ask after the interview. If they ask if you have any questions and you say you don't have any, you aren't getting the position.
- Just because you "meet all the requirements listed" means nothing in terms of your chances. Your chances are determined by your competition. This is why standing out is important, 80% of the other people applying to the position likely have the same basic requirements you do. The company is going to choose the most competitive candidate, regardless of the requirements listing.
- I strongly advise against taking unpaid internships (aka "learning opportunities"). They will look bad on your resume because they make you look desperate.
- When they are describing the job for you, look out for fancy vocabulary that may try to conceal "crunch time". For example, I've heard crunch time try to be disguised as "sprints" and "goals". Be aware of the work you are taking on and the expectations you are expected to meet.

Make sure to ask about the expected work hours per week. I've noticed that the amount of hours they expect usually jumps 5 to 10 from the listing to the interview. If you can't put in the amount of work they want, be honest about your limitations.

- As an intern, they won't expect you to get everything right in the first couple weeks, but they will want to see you are taking the position seriously early on. It's important to take a commanding lead/effort with a sense of conviction. Show them you are trying.
- There is frequently a disconnect between HR and the department you are applying to. HR typically doesn't understand all the stuff a potential candidate needs to know. It's very common to show up to interviews and for you to realize the department in question is looking for a candidate with totally different qualities.
- Businesses want to hire candidates with the lowest investment costs. For example, if a company is a C# house, they will prefer candidates with prior C# experience because they won't have to spend as much time/money training others to do the same thing. At the end of the day, every candidate is a business decision on a spreadsheet.
- Businesses want to hire candidates that will stick around for a while. This is why IT internships can be hard for CS majors to get. They assume that after the internship is over you will begin looking for another job. This is because entry-level CS jobs pay much better than entry-level IT jobs. As a result, IT interviews will really try to put you into one of two camps: software development or IT. This was a question that came up in EVERY IT interview, for me: "Do you see yourself working as a software developer or IT guy after graduation?" If you answer indifferently or towards the former, they will likely not employ you.
- Another two-camp question is the systems or framework question: "Do you want to work in systems programming or framework programming?" Choose and answer carefully, as once again indifference will not usually help.