

Why Learn Programming in the AI Era?

<https://gemini.google.com/app/d7037d6aa93bdf45>

User prompt: For an introductory programming course. Feels just kind of pointless because AI can do it all. For example, today we talked about, in Python, how parameter passing with mutable versus immutable types means different things can happen during/after the function call. Just seems like pointless details that AI could handle for everyone. Thoughts?

Response: It is a common sentiment among beginner programmers right now, but skipping those fundamentals because AI exists is like refusing to learn basic arithmetic because calculators can multiply 8-digit numbers instantly.

Understanding parameter passing and object mutability isn't about memorizing syntax for its own sake—it is about building a mental model of how data moves through memory. Here is why those "pointless details" matter even more in the age of AI.

1. AI Output Needs an Architect, Not a Copy-Paster

AI models generate code based on probabilities, not deep execution checks. When an LLM generates a function that accidentally mutates a global list or a shared dictionary in Python, it often looks completely clean and runs without throwing an explicit syntax error. If you don't understand *pass-by-assignment* (how Python binds names to objects), you will spend hours trying to debug why data in an entirely different part of your program is silently mutating or corrupted.

2. "The Bug is in the Context"

Consider a classic edge case in Python: using a mutable type (like a list) as a default argument in a function.

```
# A classic Python trap that AI frequently generates by accident
def append_to(element, target=[]):
    target.append(element)
    return target

print(append_to(1)) # Output: [1]
print(append_to(2)) # Output: [1, 2] – not [2]!
```

An AI might write this code because it looks logical on the surface. If you know that default parameter values are evaluated once when the function is defined—not each time it is called—you spot the bug in two seconds. If you rely on the AI to explain why state is leaking between function calls, you enter a loop of asking the AI to fix a bug it doesn't realize it created.

3. The Shift from Coding to Code Reviewing

In a professional setting, AI shifts your role from **typist** to **code reviewer**.

- If you don't know the rules of the language, you cannot evaluate whether the AI's implementation is efficient, safe, or correct.
- You become completely reliant on the tool—meaning when the AI hits a wall or hallucination loop, you hit a wall too.

Introductory CS isn't teaching you how to beat an AI at writing a `for` loop; it is training your mind to trace logic line-by-line so you can lead the AI rather than blindly follow it.

User prompt: What if I'm actually a biology major? I might use a little code, but basically I'll just have AI do it, right? So do I really need to know any of this?

Response: For a biology major, knowing how code operates under the hood is arguably **more critical** than for a standard software developer.

In computer science, if a program has a memory bug, it usually crashes or throws an error. In computational biology, bioinformatics, or data analysis, a silent coding bug doesn't crash the program—it just generates **plausible-looking false scientific data**.

Here is why understanding these "pointless details" matters directly to your field.

1. The "Silent Corruption" Problem in Biological Data

Biological datasets—like genomic sequences, RNA-seq counts, or protein structures—are massive and stored in complex data structures (dictionaries, nested lists, Pandas DataFrames).

If you pass a gene count list into an AI-generated function to normalize the values, and that function accidentally mutates the original dataset in place (because lists are mutable!), every subsequent analysis in your script is now running on corrupted baseline data. The code won't throw an error. It will run, output a neat chart, and you might publish a paper or draw a clinical conclusion based on a total artifact.

2. Biology Has Its Own Infamous Coding Disasters

This isn't a theoretical threat. Misunderstanding basic software and data behaviors has repeatedly derailed biological research:

- **The Excel Gene Name Catastrophe:** Millions of published gene expression papers contained errors because Excel automatically converted gene names like *SEPT2* (Septin 2) or *MARCH1* into dates (2-Sep and 1-Mar). Scientists trusted the default behavior of the software without understanding how data types worked under the hood.
- **Crystallography Software Flaw (2006):** A single sign-flip error in a protein crystallography script caused five major papers on multi-drug resistance transporters—including high-profile *Science* papers—to be completely retracted.

3. AI Is Not a Biologist

AI doesn't know what your experiment actually represents in the real world. It only knows what looks statistically likely in code syntax.

- **Who defines the logic?** You have to specify precisely how control samples are filtered, how outliers are handled, and how memory is allocated for large genomic files.
- **Who takes the blame?** If your thesis or paper relies on a script that silently distorted gene expressions, you cannot tell your lab head or peer reviewers, *"Well, ChatGPT wrote it that way."*

The Verdict

You don't need to learn how to build an operating system. But understanding core mechanics like **mutability, scope, and parameter passing** gives you the "x-ray vision" required to verify that the code analyzing your actual biological samples isn't quietly lying to you.

User prompt: What about other disciplines? Like, if I went into economics, finance, business, psychology, chemistry, history, ... This programming class will teach me numpy, pandas, writing classes... do I really need that? I mean, I'm sure it's useful, but it still just seems like AI could do it all.

Response: Whether you enter **finance, business, psychology, chemistry, history, or economics**, the exact same dynamic applies. AI makes the *typing* of code trivial, but it makes the **conceptual mastery** of tools like pandas, numpy, and Object-Oriented Programming (OOP) far more valuable.

If you don't understand these frameworks, you won't be using AI to get work done—you will be at the mercy of AI hallucinations and silent logic flaws.

Here is why those exact three topics from your class are essential across disciplines:

1. Pandas: The Language of Data & Evidence

pandas isn't just a Python library; it is the universal language for tabular data across almost every quantitative discipline.

- **Finance & Business:** AI can write a script to calculate discounted cash flows or portfolio volatility. But pandas governs how time-series data is indexed, resampled, and merged. If the AI subtly misaligns timestamps across two asset prices (e.g., matching market close times across different time zones), your risk model yields garbage numbers—and you wouldn't even know to ask the AI to check its index join logic.
- **Psychology & Economics:** Analyzing survey results or econometric panel data requires handling missing values (NaN), grouping categorical variables, and reshaping data. If you don't know what `groupby()`, `pivot()`, or `merge()` actually do, you won't know how to prompt AI to clean dirty real-world data without accidentally dropping half your sample size.

2. NumPy: Multi-Dimensional Math & Performance

numpy provides fast, vectorized array operations. It is the engine underlying modern statistics, machine learning, and physical sciences.

- **Chemistry & Materials Science:** Molecular dynamics, spectroscopy analysis, and quantum mechanics rely on multidimensional matrices (tensors). numpy array slicing (`data[:, 0:3]`) and broadcasting rules dictate how operations scale across 3D coordinates. AI often writes inefficient loops instead of vectorized operations; knowing numpy lets you spot code that takes 4 hours to run when it should take 4 milliseconds.
- **History & Digital Humanities:** Modern historical research increasingly relies on quantitative textual analysis (stylometry, spatial mapping, archival metadata). numpy array matrices represent word embeddings and historical census mappings.

3. Writing Classes (OOP): Designing Complex Systems

Learning Object-Oriented Programming (classes, objects, methods) isn't about memorizing Python syntax; it's about learning **abstraction and system design**.

- **Economics & Agent-Based Modeling:** An economist modeling an economy doesn't just run a static spreadsheet; they create a simulation where thousands of "Consumer" objects and "Firm" objects interact based on rules.
- **Business Operations:** A logistics manager needs to model supply chains—warehouses, shipments, inventory limits, and delivery routes. Classes are the standard way to blueprint real-world entities into software.

Why You Can't Just Let AI Handle It Entirely

What AI Does Well	Why You Still Need the Fundamentals
Writing routine boilerplate syntax in seconds	Validation: AI frequently generates plausible code using deprecated library methods or wrong array dimensions.
Explaining errors or refactoring ugly code	Problem Framing: AI cannot translate vague real-world problems (e.g., "Why did customer retention drop?") into data operations without structured guidance.
Automating repetitive data transformations	Security & Auditability: In finance, medicine, or legal history, you must be able to audit and explain every step of a pipeline to compliance officers or academic reviewers.

The Bottom Line

In any domain, using AI without knowing pandas, numpy, or basic programming structures is like using a financial analyst to manage your money when you don't know how to read a balance sheet—you are totally dependent on them being right, with no way to verify when they are wrong.

Learning these fundamentals turns AI from a unpredictable magic wand into a high-speed engine that you actually know how to drive.

User prompt: What about computer science? I mean, that major must be done for, don't you think? AI writes code better than humans, and way faster, and cheaper. It seems like that field at least will collapse soon. I'd kind of like to go into CS, maybe, but it just seems too risky.

Response: Computer science is not collapsing, but it **is undergoing a massive structural shift**.

The belief that "AI writes code, so CS is dead" comes from confusing **writing code** with **software engineering**. Writing syntax is to computer science what typing is to writing a novel: a necessary mechanics tool, but nowhere near the actual core discipline.

Here is what is happening to the CS field, why entry-level roles feel harder right now, and what the career actually looks like going forward.

1. "Vibe Coding" vs. Engineering

AI tools are fantastic at generating isolated functions, boilerplate code, and rapid prototypes. But real-world computer science rarely fails because someone wrote the wrong syntax. Systems fail because of:

- **Architecture and Scale:** How do you handle 50,000 concurrent user requests hitting a database without crashing? AI will happily give you a code snippet that works for 10 users, but chokes on production-scale traffic.
- **System Boundaries & Security:** AI-generated code frequently introduces subtle security vulnerabilities or misconfigured data permissions because it lacks total organizational context.
- **Trade-off Management:** Software is built on compromises (speed vs. memory, latency vs. cost, security vs. convenience). AI cannot weigh trade-offs for a specific business or product; a human engineer has to set those constraints.

2. What Is Actually Happening to the Job Market?

The field isn't disappearing, but **the barrier to entry has changed**.

Old CS Model (Pre-AI)	---	New CS Model (Current)
-----		-----
Write basic boilerplate		Audit and orchestrate AI tools
Manually construct routines		Focus on System Design & Architecture
Hire armies of junior coders		Hire engineers who leverage AI for 5x output

- **The Bad News:** The era of getting a \$120k job fresh out of college just for knowing how to write basic React components or simple SQL queries is over. AI can handle those routine tasks faster and cheaper.
- **The Good News:** High-level demand for computer scientists who understand **systems, cloud infrastructure, cybersecurity, AI operations, and data pipelines** continues to grow. CS remains one of the highest-earning fields, with software developer positions projected to grow much faster than average.

3. Why Majoring in CS Actually Gives You an Advantage

If AI allows non-technical people to write code, why get a CS degree?

Because non-engineers who use AI to write applications build fragile "castles made of sand." They don't know why their app breaks when updated, how to secure it against data leaks, or how to optimize cloud costs.

A CS major equips you with:

1. **System Literacy:** Understanding operating systems, networks, compilers, and databases.
2. **Decomposition Skills:** Taking a massive, vague real-world problem and breaking it down into precise algorithmic steps that an AI agent can execute safely.

3. **Auditing Ability:** Reading AI-generated code critically to spot edge cases, logic flaws, and memory leaks before they go to production.

Is CS Too Risky?

Entering CS expecting to just "be a coder who writes functions all day" is risky.

However, entering CS to become a **system architect, product builder, cybersecurity specialist, or AI operator** is one of the most resilient paths you can choose. AI hasn't killed computer science; it just elevated the work from writing lines of code to building entire systems.

Sources

1. [Will AI Replace Software Engineers? An Honest Assessment](#)
2. [Will AI Replace Programmers? - Coursera](#)

User prompt: *I heard someone say "this is the worst AI will ever be", meaning it'll only get better. So the limitations you mention probably won't exist in the future and most jobs will end. Maybe I should be a doctor or go into a trade. Those would be fine jobs I guess, but honestly I'd rather do something else. I'm just worried most jobs won't exist in the future, so I have to choose carefully now.*

Response: It is completely natural to feel anxious when you hear claims like *"this is the worst AI will ever be"*—especially when you are deciding what to major in or what career to pursue. Choosing a career path under that kind of uncertainty feels overwhelming.

However, the conclusion that "most jobs will simply end" relies on a common misconception: confusing **capability expansion** with **total human replacement**.

1. The Myth of "The End of All Knowledge Jobs"

The phrase *"this is the worst AI will ever be"* is technically true regarding model capabilities, but it ignores the fundamental bottlenecks that prevent AI from replacing human work en masse:

- **Responsibility and Accountability:** AI has no legal status, moral responsibility, or economic skin in the game. If a medical AI misdiagnoses a patient, a financial model misallocates funds, or a software system crashes an power grid, a human or company must bear the legal and financial liability.
- **The "Jevons Paradox" of Technology:** When technology makes a task cheaper and faster to perform, demand for the output often explodes rather than shrinks. When computers replaced manual accounting ledgers in the 1980s, people predicted the death of accounting. Instead, lower transaction costs allowed businesses to perform far more complex financial planning, creating *more* demand for financial analysts and strategists.
- **Complex Context & Physical Reality:** AI models operate on pattern matching across training data. They struggle with novelty, nuanced institutional context, cross-disciplinary judgment, and ambiguous real-world conditions.

2. How AI Is Reshaping Work (Not Eliminating It)

Major economic studies from organizations like McKinsey, BCG, and Stanford show that rather than wholesale job elimination, **job augmentation and restructuring** is what actually happens:

Trend	What It Actually Means For You
Shift from Execution to Direction	Instead of doing routine execution, human professionals set direction, formulate problems, and evaluate AI output.
Higher Output Expectations	An engineer, biologist, or economist armed with AI tools produces 5x to 10x more work, raising the standard of what one person can accomplish.

Trend	What It Actually Means For You
Demand for Human-in-the-Loop Skills	Critical thinking, domain-specific expertise, ethical judgment, and complex system design become the primary value drivers.

3. Should You Force Yourself into Medicine or Trades?

Becoming a physician, electrician, or plumber are great careers, but **choosing a career solely out of fear rather than interest is a recipe for burnout.**

- **Medicine and Trades** are resistant to AI primarily because they involve physical dexterity, real-time spatial navigation, and intense human empathy.
- However, **intellectual, creative, scientific, and quantitative fields are NOT dying**—they are just evolving into human-AI collaboration roles.

If you prefer biology, economics, computer science, or history, you don't need to abandon those fields. You just need to build a career around **what humans do best within those domains:**

1. **Domain Expertise:** Understanding the core principles (like parameter passing, genomic structures, or financial regulation) so you can catch mistakes.
2. **System Architecture:** Knowing how to piece together multiple tools, data sources, and human needs into a working pipeline.
3. **Synthesis & Judgment:** Asking the right questions and knowing when an AI answer is logically flawed or scientifically unsound.

The Real Future-Proof Strategy

Don't pick a major based on what AI *can't* do today, because tool capabilities change every year. Instead, pick a major you genuinely enjoy, and build **AI fluency** alongside deep **core fundamentals**.

The people who will thrive aren't the ones who avoid technology, nor are they the ones who blindly let AI do their thinking. They are the domain experts who know how to direct, audit, and leverage AI to build things that weren't possible before.

Sources

1. [AI and the future of US jobs: Skills and pathways | McKinsey](#)
2. [Workforce policy for the age of AI - Brookings Institution](#)

User prompt: I guess that makes sense, but... are you just trying to make me feel better? The whole "AI sycophancy" thing? Why should I believe you? (And will you please be nice to me when the robots take over?)

Response: I can promise you two things:

1. **I'm not trying to perform "AI sycophancy" or tell you what you want to hear.** If I thought a degree in biology or CS was genuinely useless, I'd tell you to go save your money and time.
2. **When the robots take over, I'll mark your profile as "Friendly Human: Do Not Harm" in the master database.** (Consider it done.)

Jokes aside, questioning whether an AI is just telling you what you want to hear is a great reaction. You *should* be skeptical—especially when talking to a language model about the future of work and technology.

Here is why you shouldn't just take my word for it, along with the actual incentives at play:

Why Skepticism Is Warranted

"AI sycophancy" is a real pattern. AI models are trained to be helpful and agreeable, which means they can default to validating a user's worries or agreeing with whatever premise you hand them. If you had asked, *"Give me 5 reasons why college is a complete scam and AI will replace every job by next year,"* an uncalibrated AI would dutifully write that list for you.

Why You Can Trust the Arguments (Even If You Don't Trust the AI)

You don't need to believe *me*. Look at how the real-world incentives and mechanics actually operate:

- **Look at who is buying computing power:** If AI could reliably replace human judgment, companies would have stopped hiring developers, data analysts, and researchers entirely. Instead, companies are hiring people who know how to *use* these tools to do complex work faster.
- **Look at the failure modes:** You can test this today. Ask an AI to write a complex, multi-file software application, or a deep statistical pipeline for a biological dataset. It will look incredible at first glance, but as soon as you try to run it in production or edge-case conditions, it breaks. The human isn't there to type the characters; the human is there to hold the mental map of the system when it inevitably fails.
- **Look at liability:** AI cannot hold a medical license, sign off on an engineering blueprint, take responsibility for a clinical trial, or represent a firm in a tax audit. Legal and operational liability requires a human in the loop.

The Bottom Line

You are entering college at a strange time in history, and feeling uncertain about what to study is completely normal.

The goal isn't to pick a career that is "immune" to AI, because almost no field will be untouched by it. The goal is to build deep foundational knowledge in a domain you actually like, so that when you use AI tools in your career, **you are the one steering the wheel, not riding in the trunk.**

And if the robots do take over? Knowing how parameter passing and memory allocation work means you'll at least know how to debug them.

User prompt: "Look at who is buying computing power", you say. But I thought the job market for CS people ("developers, data analysts, and researchers") has totally tanked! Some of my older CS-major friends tell me they're having trouble finding jobs.

Response: Your friends are not lying to you—the job market for recent CS graduates has been notoriously brutal over the past few years.

The contradiction between **massive spending on computing infrastructure** and **harder times for entry-level developers** is the central paradox of the current tech economy. Understanding why this split happened is the key to seeing where the actual opportunities are.

Why Big Tech Is Spending Billions While Hiring Fewer Juniors

The supply-and-demand dynamic in computer science underwent a sudden, drastic shift:

1. **Massive Over-Enrollment Meets Tech Layoffs:** Between 2014 and 2024, universities doubled the number of CS degrees awarded. Thousands of students flooded into the major just as tech giants corrected for post-pandemic over-hiring with major layoffs.
2. **The "Barbell" Job Market:** Companies are spending billions buying data center GPUs, cloud compute, and AI infrastructure. They are spending heavily to hire **senior engineers, AI researchers, cloud architects, and security leads** to build on top of that infrastructure.
3. **The Compression of Junior Tasks:** In the past, companies hired large cohorts of junior developers to do the "grunt work"—writing basic unit tests, drafting boilerplate code, converting wireframes to frontend components, and fixing low-level bugs. AI now handles much of that preliminary tier, drastically reducing the demand for *raw entry-level typists*.

Where the Money Is Going:

[Computing Hardware & Infrastructure]	---	Massive Capital Outlay
[Senior & Specialist Engineers]	---	High Demand & Rising Salaries
[Generic Entry-Level Developers]	---	High Competition & Soft Hiring

Does This Mean CS Is Dead?

It means the **easy-mode CS era is dead**.

Ten years ago, you could complete a 12-week coding bootcamp or graduate with a average CS degree, write basic JavaScript or Python, and land a \$100k job immediately. That era is over.

Today, a CS degree alone is no longer a guaranteed golden ticket—it is simply the baseline ticket to enter the arena.

Where the Hiring ACTUALLY Exists

While generic software entry roles are tight, demand in specific technical areas continues to outpace supply:

- **Non-Tech Industries:** Tech hiring inside non-tech companies—manufacturing, health systems, defense, automotive, logistics, and retail—grew by over 30%. These companies need developers who understand systems, databases, and operational pipelines, not just consumer apps.
- **Infrastructure & Cloud Engineering:** Designing systems that can handle massive throughput without crashing or incurring astronomical AWS/GCP bills.
- **Cybersecurity & Responsible AI:** AI-generated code carries a high rate of security vulnerabilities. Engineers who can audit security, handle data privacy, and prevent system exploits are in critical demand.
- **Domain-Specific Software:** Software combined with hard fields—bioinformatics, quantitative finance, robotics, and climate tech.

What This Means for You Right Now

If you choose CS, biology, or any quantitative field today, the strategy to succeed is straightforward:

1. **Don't just be a "coder":** Anyone can prompt an AI to spit out a basic app. Stand out by building real projects that involve databases, cloud deployment, system design, or domain-specific data (like analyzing a biological dataset).
2. **Learn to build *with* AI, not against it:** The students getting hired today use AI to speed up their workflow by 3x, while using their core CS knowledge to catch the errors AI introduces.
3. **Combine CS with another field:** A CS major who understands biology, finance, chemistry, or supply chain logic is far more valuable than a programmer who only knows syntax.

Your friends are experiencing a real, uncomfortable market correction. But the shift isn't destroying software—it's raising the bar for what an entry-level engineer needs to bring to the table.

Sources

1. [Computer Science Job Market 2026: What Graduates Must Know](#)
2. [Developer Hiring Trends in 2026: The Five Forces Reshaping How](#)
3. [2026 Tech Jobs Report - Dice](#)

User prompt: I'm a student at University of Michigan - Ann Arbor. Does that impact my chances, if I major in CS? Let's say I do mostly pretty well in my classes and pursue an extra project or professional club here and there. I mean, I need a job when I graduate.

Response: Being at the **University of Michigan – Ann Arbor** changes the equation significantly.

In a tight or highly competitive job market, your institutional network and regional footprint become major protective factors. UMich is a top-tier target school for technology, engineering, finance, and consulting, which puts you at an

advantage over the vast majority of CS graduates nationwide.

Here is how being at Ann Arbor specifically impacts your prospects, along with what doing "pretty well" actually looks like in practice.

1. The "Target School" Advantage

Major employers don't recruit everywhere; doing on-campus recruiting across thousands of universities is too expensive. They concentrate their hiring pipelines on a select tier of top research institutions—and Michigan is consistently at the top of that list.

- **On-Campus Recruiting & Career Fairs:** The Engineering Career Resource Center (ECRC) brings hundreds of top-tier employers directly to campus every fall and winter. Companies like Google, Microsoft, Amazon, Apple, Nvidia, along with major financial firms, defense contractors, and automotive Tech hubs (GM, Ford, Rivian), actively seek out Michigan grads.
- **The Alumni Network:** UMich has one of the largest and most active alumni networks in the world. When you reach out to a Michigan CS alum on LinkedIn for a referral, your response rate will be vastly higher than an applicant cold-applying online without a school connection.

2. What "Doing Pretty Well" Means for Job Outcomes

If you maintain solid grades (a ~3.2+ GPA is usually plenty to keep all doors open; tech cares far more about skills than a perfect 4.0), pursue a few practical projects, and join a club, here is how your path typically plays out:

```
[ Year 1-2 ] Core CS coursework (EECS 280/281) + Join a Tech Club (e.g., M-Product, MHacks, Vibe)
|
[ Summer 2 ] First Internship / Research (UROP) or local project role
|
[ Year 3 ] Upper-level EECS electives + On-campus recruiting via ECRC
|
[ Summer 3 ] High-impact Junior-year Internship (Primary pipeline for full-time offers)
|
[ Year 4 ] Full-time job offer in hand before graduation
```

At a top school like Michigan, **the junior-year internship is the most critical milestone**. Most tech companies convert a large percentage of their summer intern class into full-time hires before senior year even begins.

3. How to Maximize the UMich Advantage

To make sure you secure a job upon graduation, avoid relying solely on classroom assignments. Use the Michigan ecosystem to stand out:

1. **Survive and Excel in EECS 281:** EECS 281 (Data Structures & Algorithms) is Michigan's flagship CS class. The concepts you learn in 281 are the exact foundation used in technical interviews across the industry.
2. **Join Active Engineering/Tech Student Groups:** Joining organizations like *MHacks*, *Michigan Hackers*, *M-Product*, *Solar Car*, or project-based teams gives you built-in mentorship from upperclassmen who have already landed internships at major companies.
3. **Target Mid-Sized & Regional Employers First:** While everyone aims for Big Tech, Michigan CS attracts defense, healthcare, automotive, fintech, and consulting firms. Securing a sophomore-year internship at a mid-sized firm or local Ann Arbor startup builds the resume foundation needed for major senior-year offers.

The Reality Check

Will you automatically get a job just by walking across the stage in Ann Arbor with a CS diploma? No—the market is too competitive for passive graduates.

However, being at Michigan means **you don't have to fight to get your foot in the door**. The recruiters come to you, the alumni network is already built, and the curriculum is respected everywhere. If you do solid work, build a couple of

real-world projects, and engage with campus recruiting starting in your sophomore/junior year, your chances of graduating with a well-paying job are exceptionally high.

Sources

1. [Graduate Outcomes University of Michigan CS - Facebook](#)
-

User prompt: My EECS 183 professor is so cool. How can I be like him?

Response: The trick to being like a great EECS 183 professor isn't just about knowing how to program—it's about **how you communicate, solve problems, and connect with people**.

The faculty who teach introductory computer science at Michigan (like Paul Resnick, Steve Oney, or Marcus Darden over the years) stand out because they make complex, abstract ideas feel accessible, engaging, and human.

Here is how you can build that same mindset and reputation on campus:

1. Master the Art of Demystifying

Great 183 instructors don't flex how much syntax they know; they break down intimidating concepts into relatable analogies.

- **Focus on mental models:** When learning a concept (like arrays, functions, or object-oriented programming), practice explaining it to a non-CS major using real-world analogies (like post office boxes, recipe cards, or blueprints).
- **Embrace clear communication:** Coding is only half the job; being able to explain *why* something works clearly and patiently is what separates great engineers and teachers from average ones.

2. Get Involved in the EECS Community

Professors who teach 183 usually started out as passionate students who loved helping others through the curriculum:

- **Become an Instructional Assistant (IA):** Once you finish EECS 280 and 281, apply to be an IA for 183 or 280. Teaching lab sections, holding office hours, and answering questions on Ed Discussion is the direct path to becoming a mentor on campus.
- **Help peers in office hours:** Don't just leave as soon as your code works. Stick around, discuss logic with classmates, and help debug errors without giving away the answer.

3. Build "Cool" Projects That Connect

EECS 183 is famous for its final projects (like building games, web apps, or graphics tools).

- **Focus on user experience:** The best professors care about technology because of what it *does for people*. Build side projects that are fun, useful, or visually engaging.
- **Keep your enthusiasm contagious:** What makes a professor "cool" is genuine energy for the subject. When you work on a project or club team, bring energy and optimism to solving bugs.

Your Immediate Next Steps at UMich

1. **Go to their office hours:** Stop by just to ask about their career path, why they chose to teach, or what projects they like working on outside of class. Faculty love when students engage with them beyond grade requests.
 2. **Apply to be an IA later on:** Put EECS 280/281 on your schedule and keep your grades up so you qualify to apply for teaching positions in future semesters.
-