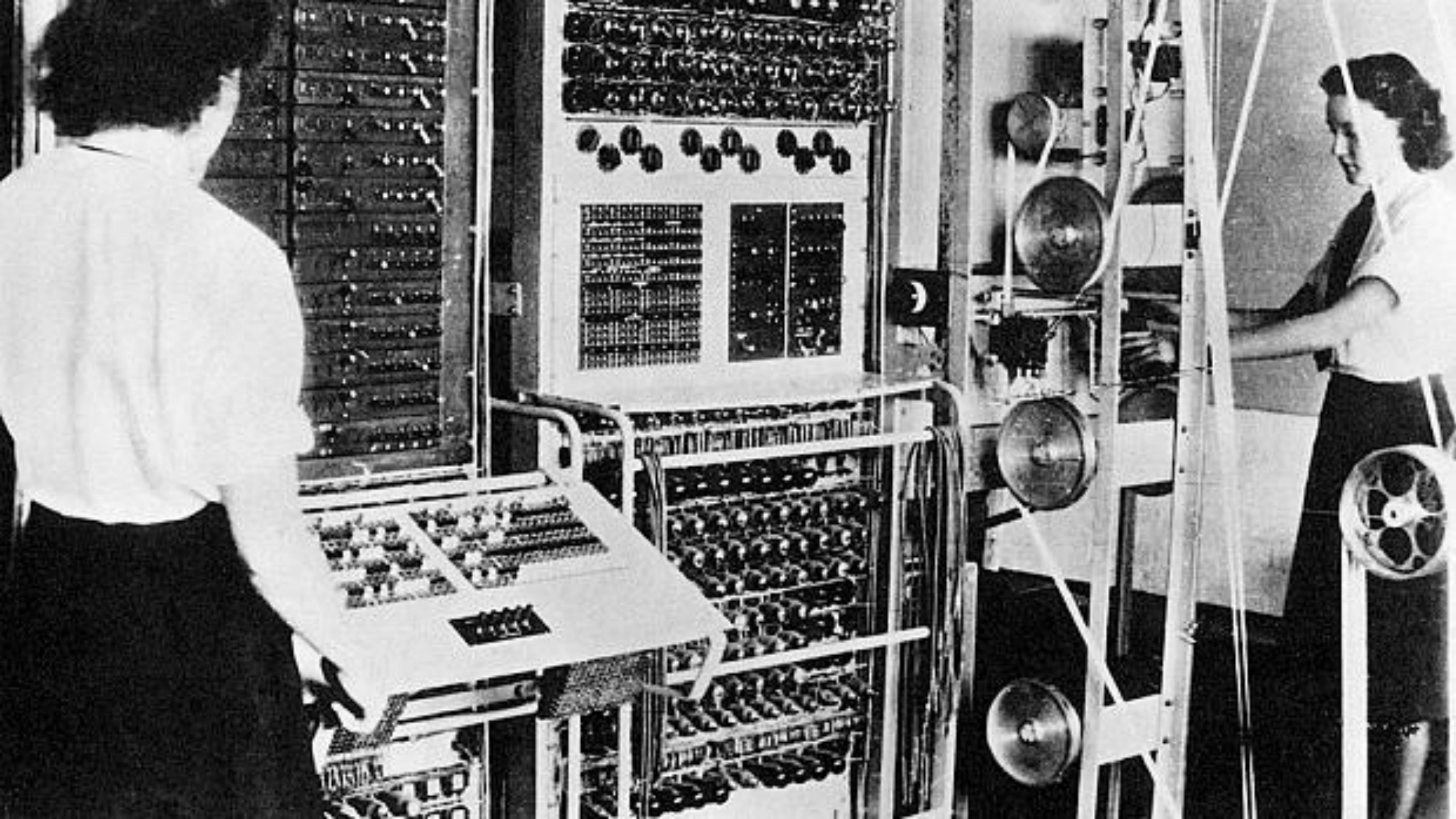
Managing Your Career in the Time of GenAI

Jesse Lecy, PhD

What is a computer?

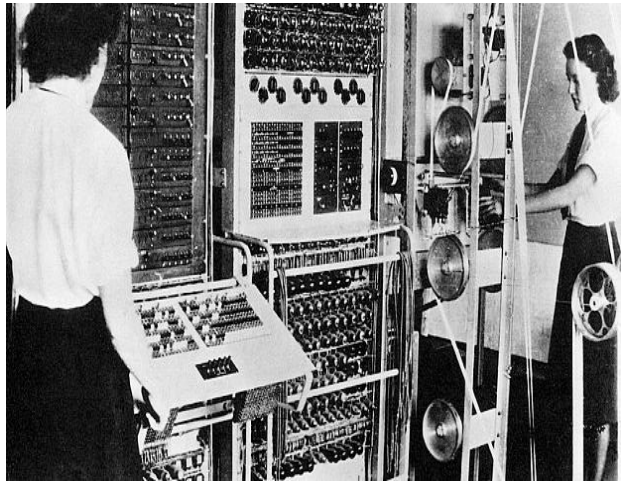
Computing Division of the US Veterans Bureau, 1924.



What is a Computer?

- **ORIGINAL:** one who computes, or a mathematically skilled individual that can solve complex numeric problems
- **EVENTUAL:** one who runs a computing machine that solves numeric problems
- **CONTEMPORARY:** a computing machine that solves numeric problems

Technological Change



The computer not only stole their jobs, but it took their title as well!

Paradigm Shifts in Engineering Curriculum as a Result of New Technology (the Calculator)

At West Point the time devoted to slide-rule instruction in first-year math “varied from five to nine classes”—time that simply vanished from the syllabus once calculators took over.

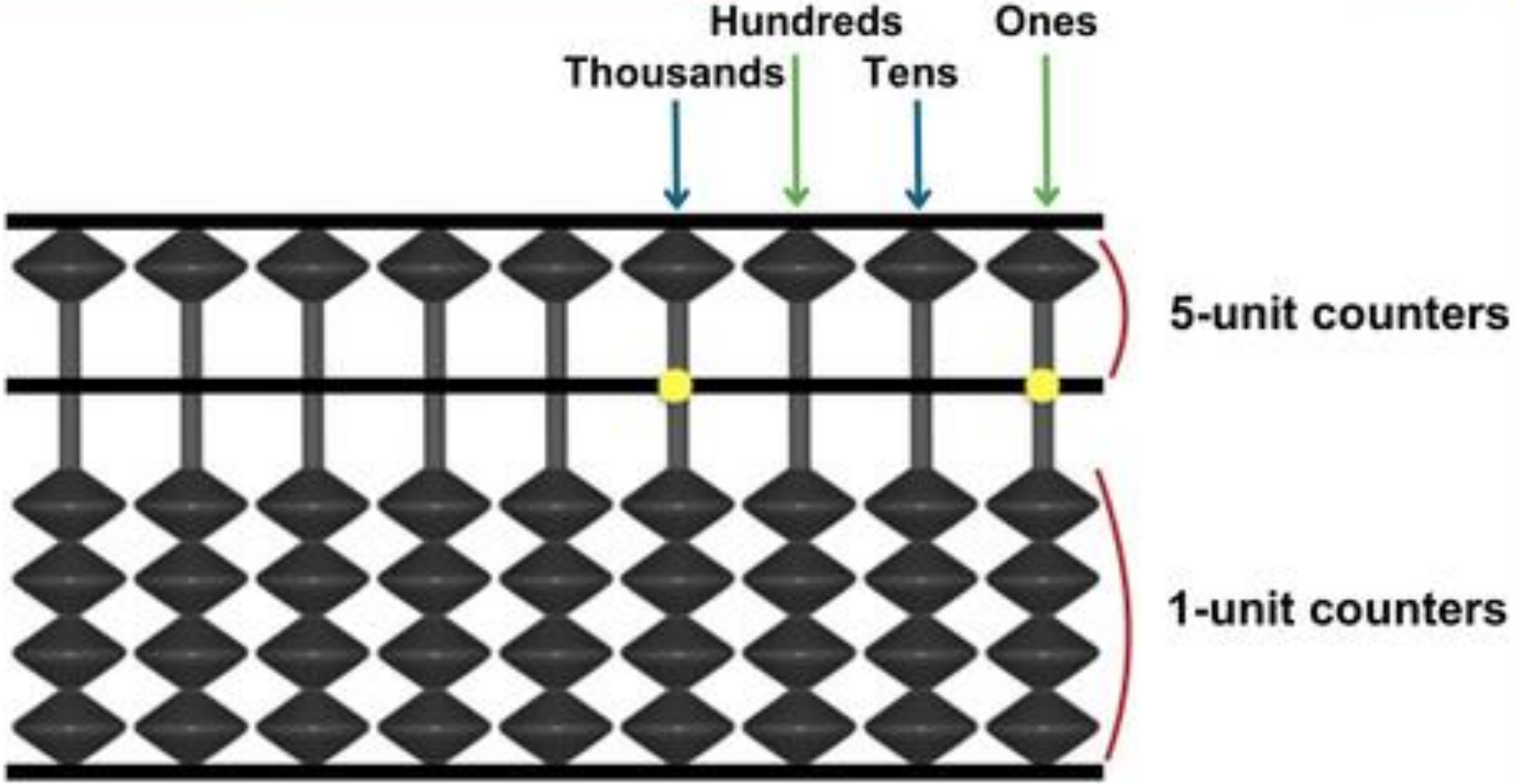
- Slide rules disappeared fast (1972–1976). After the HP-35 (1972), handhelds spread and slide-rule use collapsed during the mid-1970s.

The curriculum shifted from emphasizing calculations that humans are good at (well studied numerical approximation techniques that rely on tables and tricks) to calculations that computers are good at (matrix algebra, numeric integration).

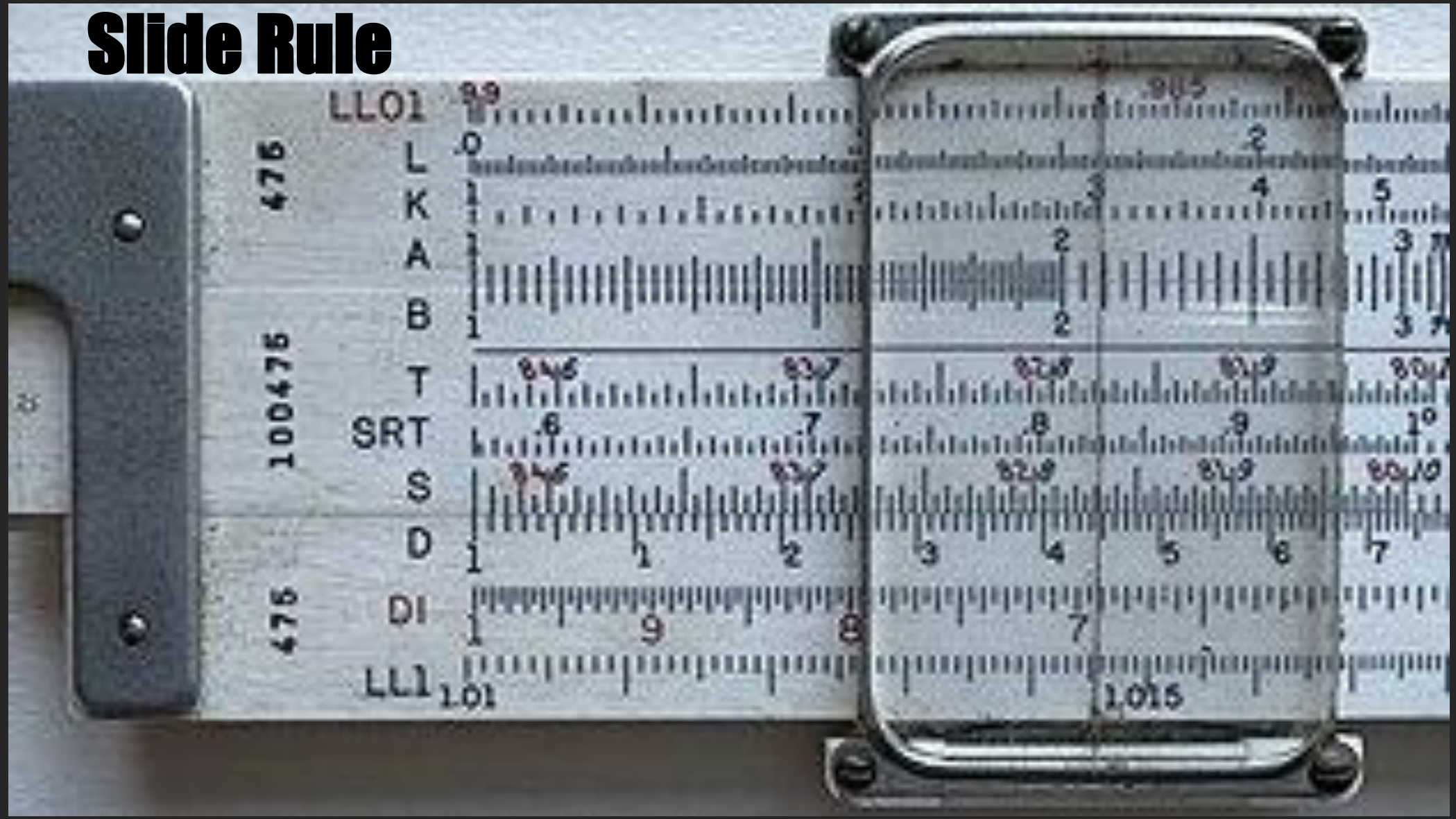
- **Structural analysis:** Hardy Cross’s moment distribution dominated because it was doable by hand; it was supplanted in teaching by the stiffness/matrix methods and then FEM as computing spread.

- **Fluids:** Students long used the Moody chart to avoid nasty algebra; with calculators it became normal to solve the Colebrook/Haaland relations directly and to use the chart mainly for intuition.

Parts of Japanese Abacus



Slide Rule



Human Computers (esp. women at NASA and observatories)

Before electronic computers, “computers” were people — often highly-trained women — who could perform complex calculations for astronomy, ballistics, or engineering.

- Katherine Johnson and her colleagues at NASA were indispensable for calculating orbital trajectories in the 1950s–60s. Their skills were critical, but by the late 1960s mainframes started replacing much of this manual work.

- Harvard’s “computers” in astronomy (like Henrietta Leavitt) once manually processed huge tables of stellar data. By the mid-20th century, their computational roles were largely absorbed by machines.

Their mathematical insight was still valuable, but the *rote calculation speed* that had once set them apart was no longer rare or marketable.

Actuarial and Financial “Lightning Calculators”

In insurance and banking, firms once hired people for raw mental arithmetic speed — “lightning calculators” who could multiply six-digit numbers in their heads or maintain vast interest tables. Their abilities were legendary… until mainframes in the 1950s and handheld calculators later made those talents unnecessary.

Slide Rule Masters

Engineers before the 1970s often prided themselves on speed and accuracy with slide rules. Being a true slide rule wizard could win you prestige in engineering firms. But with the introduction of handheld calculators (like the HP-35 in 1972), those decades of cultivated skill became obsolete in just a few years. A younger engineer with a calculator could outpace even the most seasoned slide-rule savant.

Ballistics Tables during WWII

The U.S. Army employed hundreds of mathematicians to compute firing tables for artillery. Being lightning-fast and accurate with differential equations and approximations was considered elite work. With the arrival of early electronic computers like ENIAC (built in 1945 precisely for this task), entire departments of human “computers” were displaced.

Paradigm Shifts in Engineering Curriculum as a Result of New Technology (the Calculator)

The curriculum shifted from emphasizing calculations that humans are good at (well studied numerical approximation techniques that rely on tables and tricks) to calculations that computers are good at (matrix algebra, numeric integration).

- **Structural analysis:** Hardy Cross's moment distribution dominated because it was doable by hand; it was supplanted in teaching by the stiffness/matrix methods and then FEM as computing spread.

- **Fluids:** Students long used the Moody chart to avoid nasty algebra; with calculators it became normal to solve the Colebrook/Haaland relations directly and to use the chart mainly for intuition.

Programs shifted focus to teaching tasks that traditional computer are NOT good at (theory, modeling, design). Engineering jobs became more lucrative because computers made engineers more productive.

Democratization of Programming

What is a Data Scientist?

- **ORIGINAL:** one who has deep expertise in some domain (business, policy, medicine, etc.) and solves hard problems that require data, programming, and statistical modeling

- **EVENTUAL:** more emphasis on domain expertise and statistical modeling, less expertise needed in programming to be a competent data scientist.

    - Democratization of data science approaches to problems

    - Programming skills become commoditized, thus more competitive, but at the same time the ability to apply the skills will become more valuable

What has changed since GenAI?

- The book *Prediction Machines: The Economics of AI* explains how economists developed an analytical framework for understanding economic changes catalyzed by the internet: what economic input changes? And what happens if that goes to zero?

    - Searching for products used to require access to catalogs, many phone calls, or extensive networks of knowledgeable people. Google made search basically free, and Yelp eliminated the risk of transacting with unfamiliar businesses. As a result, niche industries began to thrive, and the economy broadened.

    - Analogously, machine learning and AI are driving the cost of making predictions or decisions to zero. Industries that required significant labor focused on tedious information processing tasks will be most impacted.

Analogous Changes in Tech

- Before GUI-based operating systems (Mac in 1984, Windows in 1990) users had to learn Unix commands (basic programming) in order to turn a computer on, install programs, and use apps.

- The visual user interface democratized personal computers by making them accessible to teachers, students, and non-technical experts. This led to a wider adoption and a more productive workforce.

- AI similarly removes key barriers to programming by making it more accessible to the average user.

What has changed since GenAI?

- Analogously, we can try to pinpoint which expensive activities will change in the field of data science (analytics, big data, evidence-based professions). Which tasks are most time-consuming, require skilled labor, and thus are most expensive?

    - Programming
    - Cleaning data
    - Synthesizing large quantities of information
    - Writing first drafts of analysis
    - Creating templates for websites / reports

What has changed since GenAI?

- For programming specifically, we can see a long trend where computer programming languages have been redesigned and reengineered to move programming syntax closer and closer to human languages while making them less rigid (more forgiving of small errors) and more expressive (easier to do a lot of things with a little code)

- GenAI is just the final stage of this trend, making it possible to write programs with prompts (human language syntax) instead of **mastering** new technical computer languages.

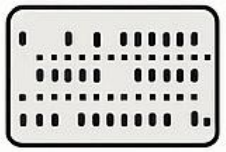
- Note the emphasis on “mastering” instead of learning – the user still needs to know how to evaluate and deploy the code that results from the human language prompts, so the user still needs to learn the programming language, even if the process is much faster and more superficial.

- This is not a huge departure from where things were headed.

Innovation (Year)	Significance / Advance
ENIAC & Plugboards (1945)	Programs set mechanically with switches and cables; required knowledge of logic and electrical engineering.
Punch Cards (1950s)	Programming with physical cards standardized input, but still abstract and mechanical, demanding careful sequencing.
FORTRAN (1957)	First high-level language; allowed scientists to write in algebraic notation instead of machine code.
COBOL (1960)	Business-oriented language with English-like syntax; opened programming to non-scientists.
UNIX (1969)	Early portable operating system; abstracted away hardware differences and supported multi-user environments.
C (1972)	Structured, portable language; closer to hardware than higher-level languages but still more abstract than assembly.
C++ (1985)	Brought Object-Oriented Programming to the mainstream; enabled abstraction through classes, inheritance, and automatic handling of many low-level details.
Java (1995)	“Write once, run anywhere” with the Java Virtual Machine; simplified cross-platform programming.
Python (1991, mainstream in 2000s)	Interpreted, highly readable syntax close to human language; removed need for manual memory management.
Jupyter / Literate Programming (2010s)	Integrated code with documentation and narrative; emphasized readability, collaboration, and human-centric programming.

EVOLUTION OF PROGRAMMING LANGUAGES

ENIAC & Plugboards (1945)
Programs set mechanically with switches and cables; required knowledge of logic and electrical engineer)



Punch Cards (1957)
First high-level language, allowe- scientists to write in algebraic notation instead of machine code,

COBOL (1960)
Business-oriented language with English like syntax; opened programming to non-scientists

UNIX (1969)
Early portable operating system abstracted away hardware dirfere- ence mui–usper- environments

C 1972)
Structured, portable language; closer to hardware than higher-level languages

C++ (1985)
Biought Object- Orient- ted Progra- mming

Java (1995)
Write once, run anywhe're' with the Jaya Virtual Machine; simplified cross platform programming

Punch Cards (1950s)
Programming with physical cards standardized input, but still abstract and mechanical demariding careelng

First high-level
Business-oriented language with Eng- lish like syntax; opened programn- ing to non-scien-

Jippeter / Literate Programming (2010s)
Integreted smta reabiale yntay. close to human language but

Interpreted, readitar to monual memory

Python (1991, mainstream in 2000s)
interpreted, highly readable syntax close to human language: removed need for manual memory management

Jupyter / Literate Programming (2010s)
Integrated code with documentation and narrative; emphasized readability, collaboration and human-centric prog

GenAI (2022): prompt-based programming

What has changed since GenAI?

- What happens when more people have programming skills?
- Many companies are downsizing their software engineering teams because they can maintain similar levels of productivity with smaller teams. This always happens in large companies beholden to shareholders. Older firms cut costs to maintain profit margins as market share decreases when rapid innovation occurs.
- New companies can now leverage large armies of cheap programmers or small armies of highly-efficient programmers to bring new products to market. New opportunities will likely proliferate.
- Once everyone can learn to program **quickly** there is less incentive to learn to program **well**. The labor market will become flooded with “cheap commodity goods” as there are more potential employees with skills that used to be rare, and more analytics firms that make it inexpensive to outsource capacity.
- These conditions result in a lot of noise and makes it difficult to assess quality – of potential employees and of analytics products that now resemble fast fashion – trendy and cheap.
- How can you add value in this market? And how can you signal your value?

Democratization of Technology and the Evolution of Industry

How does new tech impact your job market?

- To forecast how the field of data science / analytics might change as a result of GenAI, we need to look at analogous technological change.

- Examine examples where specialized knowledge that is hard to acquire becomes more accessible, or industries that required specialized equipment as the equipment becomes cheap & ubiquitous.

How does new tech impact your job market?

- Notably, these cases are different than technologies that replace unskilled labor (manufacturing, farming, etc.) that have produced mass layoffs
- Barriers to entry prevented talent from participating, removing barriers results in the growth of a field, not contraction
- Knowledge-intensive industries benefit from ***agglomeration economies*** where industry density produces knowledge spillovers and specialization that increase productivity
- Competition intensifies, but the industry is growing: there are many opportunities to build a niche as the industry evolves and matures

Steel and the Mini Mill Revolution

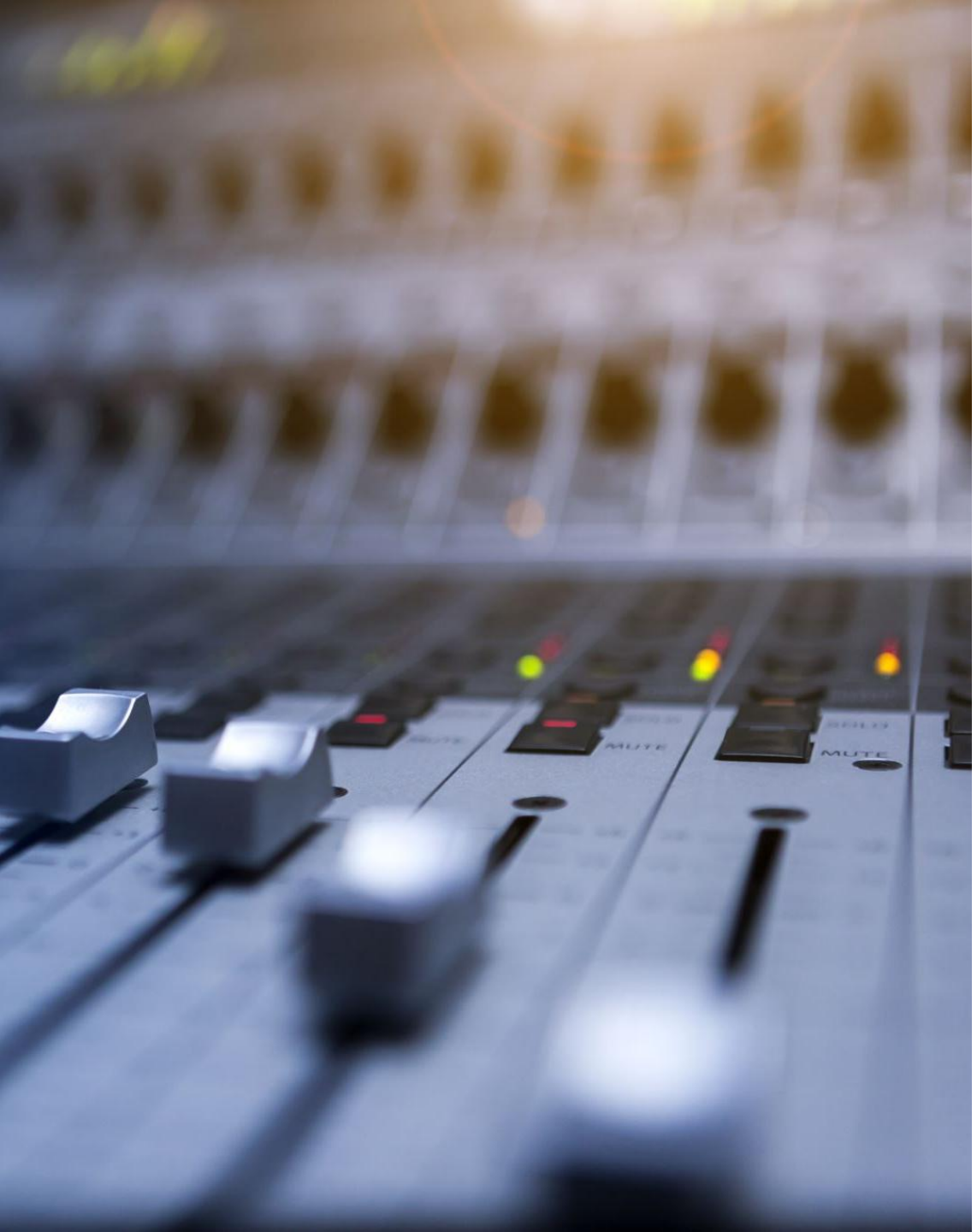
- **Before**: Traditional integrated steel mills required billions in capital, huge workforces, and decades of know-how. Firms like U.S. Steel had massive barriers protecting them.

- **Mini mills**: Nucor and others pioneered small electric arc furnaces in the 1960s–70s, which could melt scrap cheaply with far less capital investment. They started at the “low end” (rebar, rods) but quality improved, and eventually mini mills moved into sheet steel and higher grades.

- **Impact on firms**: Incumbent giants lost market share and closed plants. Nucor grew from into one of the biggest steelmakers in America. Knowledge barriers (metallurgy, furnace ops) shrank, because new processes were simpler to run. Workforce shifted from unskilled labor to skilled metal fabrication processes.

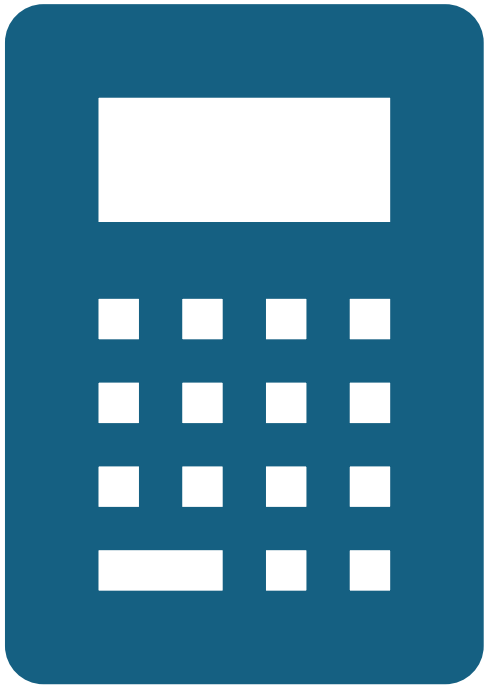
Printing and Publishing (Desktop Publishing in the 1980s–90s)

- **Before**: Typesetting was a highly specialized craft, with whole unions of skilled compositors. Running presses required enormous capital and expertise.
- **After desktop publishing**: Apple's Macintosh, laser printers, and PageMaker software suddenly let small shops — even individuals — design and print professional-looking materials.
- **Impact on jobs**: Typesetters lost most of their work, print shops consolidated, and new creative industries (graphic design, marketing, indie publishing) flourished. The bottleneck shifted from technical mastery to design talent.



Music Recording (Studio Tech to Home Studios)

- **Before**: Professional studios cost millions, and audio engineers were gatekeepers of music production. Labels controlled access.
- **After**: Digital audio workstations (like Pro Tools, Ableton, Logic) and affordable high-quality mics / interfaces let anyone with a laptop make near-studio-quality music.
- **Impact on jobs**: Traditional studio engineers and large recording complexes declined, while a long tail of producers and indie artists exploded. Expertise shifted toward mixing / mastering finesse, not access to million-dollar gear.



Finance (Quantitative Analysis & Cheap Computing)

- **Before**: High-level financial modeling required access to mainframes and staff trained in numerical methods.
- **After**: Personal computers + Excel + statistical packages democratized modeling. MBAs learned what once took PhD-level quants.
- **Impact on jobs**: Specialist numerical analysts declined in proportion, while finance jobs exploded overall — but with new emphasis on interpretation, strategy, and innovation, not raw computation.

The Pattern

1. **Barrier falls** → Capital, knowledge, or equipment that was scarce becomes widely available.

2. **Incumbents weaken** → Firms and unions built on guarding those barriers lose leverage.

3. **Market expands but redistributes** → Total output may grow (more books, more maps, more music, more steel), but the jobs reorganize around creativity, design, management, and integration rather than the old craft bottleneck.

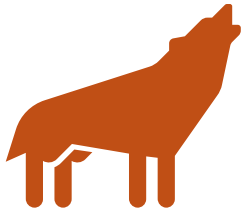
4. **Winner's curse** → New entrants often win because incumbents are too locked into their high-cost models (classic Clayton Christensen "disruption" story).

5. **Insurgent advantage** → leveraging new technologies to first compete on price, then use those opportunities to refine processes and products so you can start competing on value

How does this impact a program like PEDA?

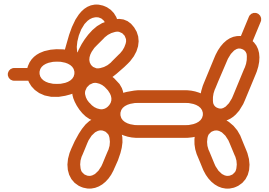
Changes to Data Science Curriculum

Senior Analyst

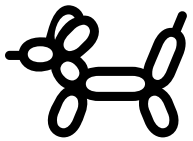


now

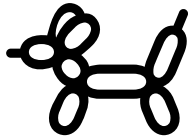
Project Manager Mindset



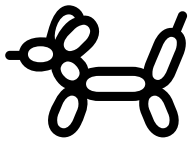
Junior Analyst



Junior Analyst



Junior Analyst



GenAI

GenAI

GenAI

Changes to Data Science Curriculum

The focus of an applied master's program is training for entry level positions in an industry where knowledge-acquisition is ongoing. We used to focus on junior engineering skills:

- **Junior Engineering Mindset (first draft productivity)**

  A junior engineer requires detailed instructions, focuses on completing specific tasks, and relies on seniors for checking the accuracy of work.

- **Senior Engineering Mindset (project management)**

  A senior engineer, in contrast, possesses significant experience and in-depth knowledge, can identify and solve problems with less direction, can translate a poorly-specified problem into a concrete plan by designing and prototyping a workflow, outline technical requirements, and proving the documentation needed for others to execute. Senior engineers tend to think in terms of long-term business value and synergies across projects.

Changes to Data Science Curriculum

- Becoming an expert coder requires putting in your 10,000 hours of practice to:
    - become familiar with packages
    - read documentation
    - memorize common functions and arguments
    - spend days on discussion board research bugs
- To put skill acquisition in perspective:
    - **Part Time:** 10 hours / week ~ requires **20 years of practice**
    - **Full Time:** 40 hours / week ~ requires **5 years of practice**

Changes to Data Science Curriculum

- Syntactic Skills in Computer Science:

  A programmer's ability to correctly use the grammar and structural rules of a specific programming language. This mastery allows a programmer to write code that can be properly interpreted or compiled by a computer, i.e. executing without producing errors.

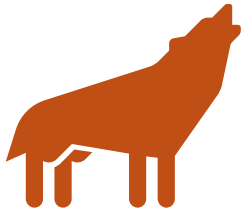
- Are analogous to “computing” skills in the engineering profession prior to the advent of calculators:

  Math skills took time to develop, and thus were valuable

  Once calculators were introduced, simply doing the math no longer added value. It was knowing which math equations to solve (theory and modeling skills), and which types of problems align with priorities of an organization.

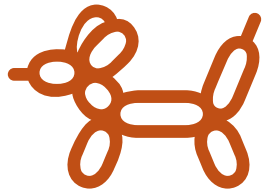
Changes to Data Science Curriculum

Senior Analyst

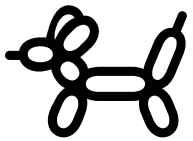


now

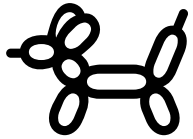
Project Manager Mindset



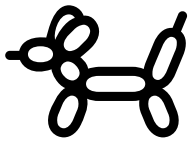
Junior Analyst



Junior Analyst



Junior Analyst



GenAI

GenAI

GenAI

Changes to Data Science Curriculum

Junior analyst skills might not be the best way to add value to a team.

- Shift from an emphasis on writing code to using AI to write a first draft for you
  - Requires a clear understanding of the problem you are trying to solve
  - Break the problem into discrete tasks
  - Use appropriate prompts to produce code for each task

- Shift from writing code to designing workflows
  - Integrate all of the discrete parts into a single coherent workflow
  - Become skillful at debugging workflows

Changes to Data Science Curriculum

- Learn to check the accuracy of your own code
    - Emphasis on documentation and unit testing
    - It’s not enough to show your boss the code runs, must also demonstrate that it works as expected
- Shift from writing a single working script to scaling your work
    - Turn a script into a package so code is properly documented, and it can be reused
    - Eventually be able to architect solutions that others can execute (members of your team or consultants)
- Focus on designing processes and quality control, not specialization. Highly specialized knowledge is easily replaced. The ability to prototype, iterate, and institutionalize the knowledge generated from that process will never be replaceable.

Changes to Data Science Curriculum

- The goal is to use the time you gain in mastering syntax and writing first drafts of code to practice higher-level skills

- You still need to learn what the code means and how it works! You just don't have to memorize all of the functions and arguments yourself.

- Instead of thinking at the syntax level (which functions do I need) you should be thinking at the workflow level (what problem am I trying to solve, what does the final deliverable look like, and can I work backwards to identify all discrete steps in the solution?).

- Learning to write pseudocode is analogous to good prompt engineering – they both operate at a similar level of abstraction.

Changes to Data Science Curriculum

AI makes A LOT of mistakes. Never take a solution at face value.

Any code written by humans or by AI should be stress-tested to ensure it is working as expected. You need to learn how to write unit tests for your code.

The way a unit test is created, independent of running any code, demonstrates whether you understand the problem.

Executing unit tests ensure the code is working as expected AND is also robust to real world cases (for example, data contains special values or inputs are the wrong data type).

Changes to Data Science Curriculum

You still need to learn what the code means and how it works! You just don't have to memorize all of the functions and arguments yourself.

Rule of thumb: if you don't understand what your code is doing, then don't give it to your boss or sell it to your client (also don't turn in assignments with code you don't understand).

AI is good at creating first drafts quickly, but it rarely gets it right the first time. You need to review the code, which requires you to understand the code.

Thriving in Chaotic Job Markets

When written in Chinese, the word 'crisis' is composed of two characters—one represents danger and the other represents opportunity. ~ JFK

Cultivate an Adaptive Identity

Instead of thinking: **“I am a data scientist”**

Think: **“I help people make better decisions with evidence”**

That framing makes it easier to evolve as tools and labels change.

Focus on Durable Skills, Not Tools

- Tools (R, Python, SQL, TensorFlow) will come and go. What sticks is *how you think about problems*:
    - Framing good questions.
    - Understanding data-generating processes.
    - Statistical reasoning.
    - Communicating uncertainty.
- Example: people who mastered **data modeling and experimental design** decades ago are still relevant, even if their FORTRAN code is ancient history.

Climb Up the Value Stack

Low-level tasks (cleaning, basic analysis, even coding) will get automated. What's harder to automate is judgment:

- Choosing which problems matter.
- Translating results into business, policy, or scientific decisions.
- Weighing tradeoffs and ethics.

In short:

- don't compete with the machine on speed — compete on interpretation, creativity, and wisdom.
- don't compete with other people on expertise, compete on process

Develop “Glue” Abilities

Industries in upheaval create *interfaces* where opportunities bloom:

- Bridging technical people and decision-makers.
- Connecting messy real-world domains (healthcare, energy, nonprofits) to abstract modeling.
- Being the person who can explain complex models to a skeptical regulator or board.
- These “translator” roles become linchpins when tech is moving too fast for most people to follow.

Watch for Structural Shifts in Opportunity

- As tools democratize, *supply of analysts grows*, so wages for routine work may compress.
- But *demand for domain-specific insight grows*. Example: climate modelers, genomics analysts, or nonprofit data scientists who understand both data and context.
- Keep an eye on where scarcity *moves next*: it won't be raw modeling skill; it may be **interpretive skill, governance, or sector-specific expertise**.

Think Portfolio, Not Ladder

Your mentors may have climbed a straight ladder: junior analyst → senior analyst → director. In turbulent fields, ladders wobble. A safer bet is a **portfolio of projects**:

- Build things, publish analyses, contribute to open-source, collaborate widely.
- This builds resilience, reputation, and optionality.
- It’s closer to how designers or entrepreneurs build careers than to traditional corporate paths.