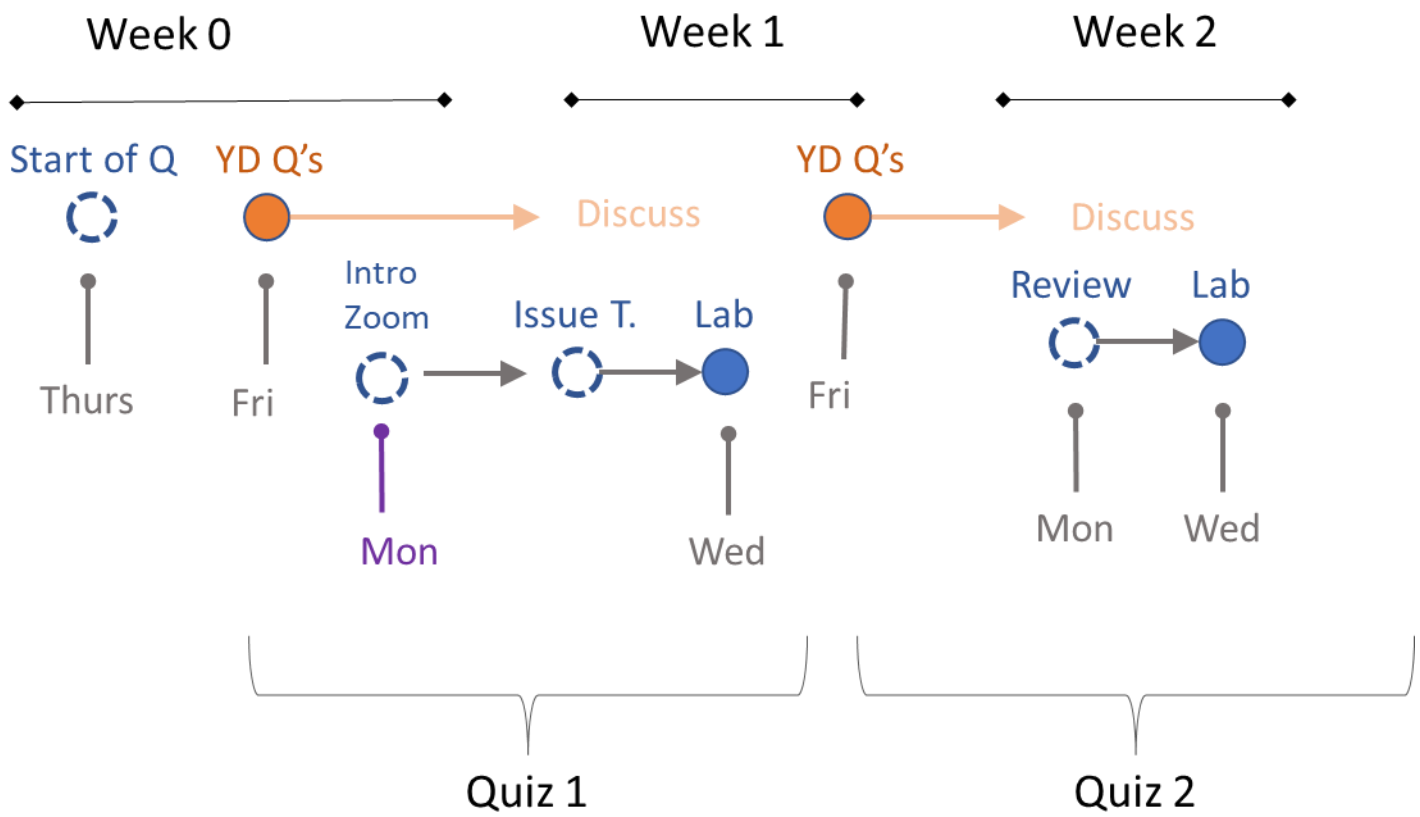
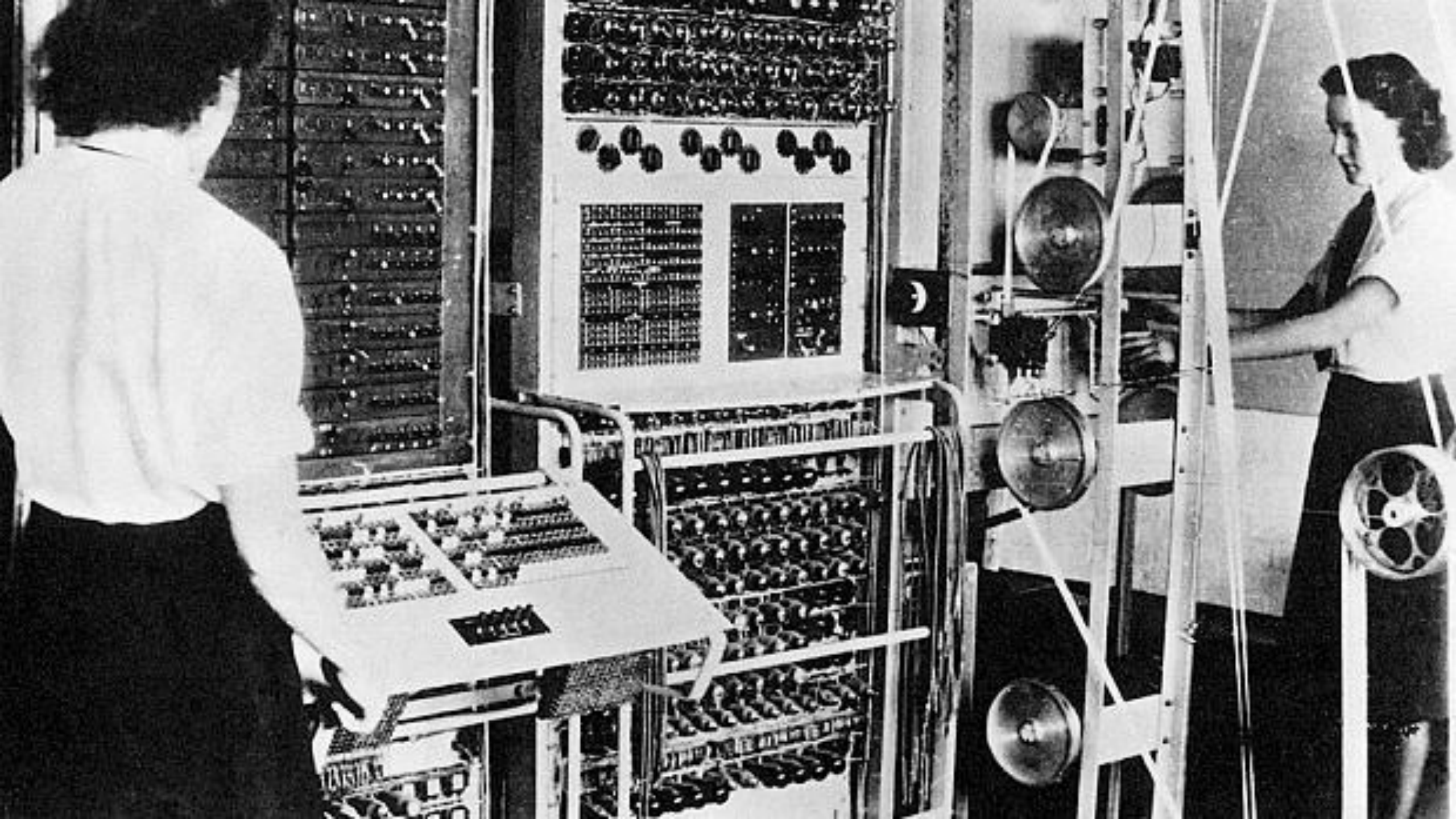

Week 0
Week 1
Week 2
Start of Q
YD Q's
Discuss
YD Q's
Discuss
Intro
Zoom
Issue T.
Lab
Review
Lab
Thurs
Fri
Mon
Wed
Fri
Mon
Wed
Quiz 1
Quiz 2

AI Policies

What is a computer?

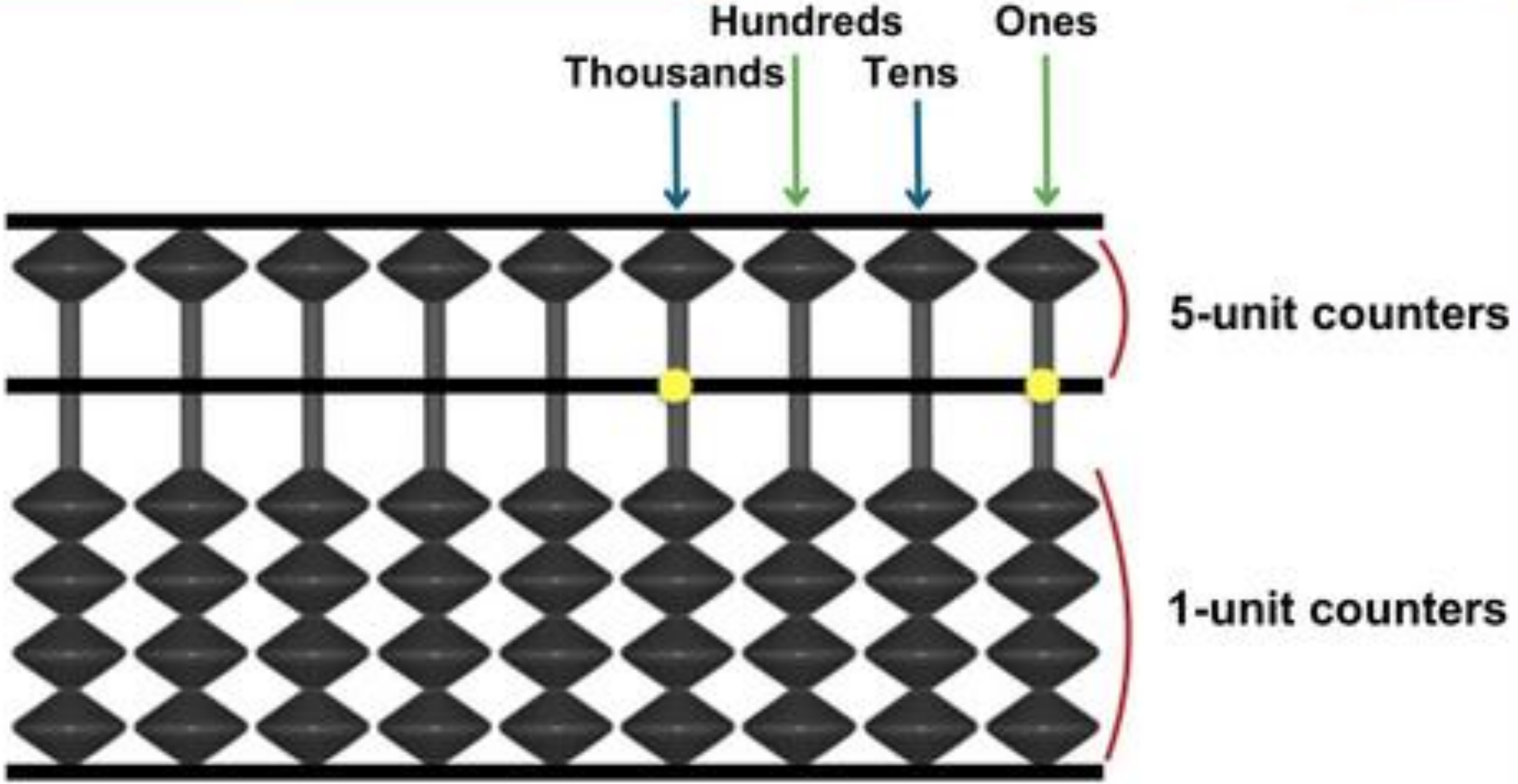
Computing Division of the US Veterans Bureau, 1924.



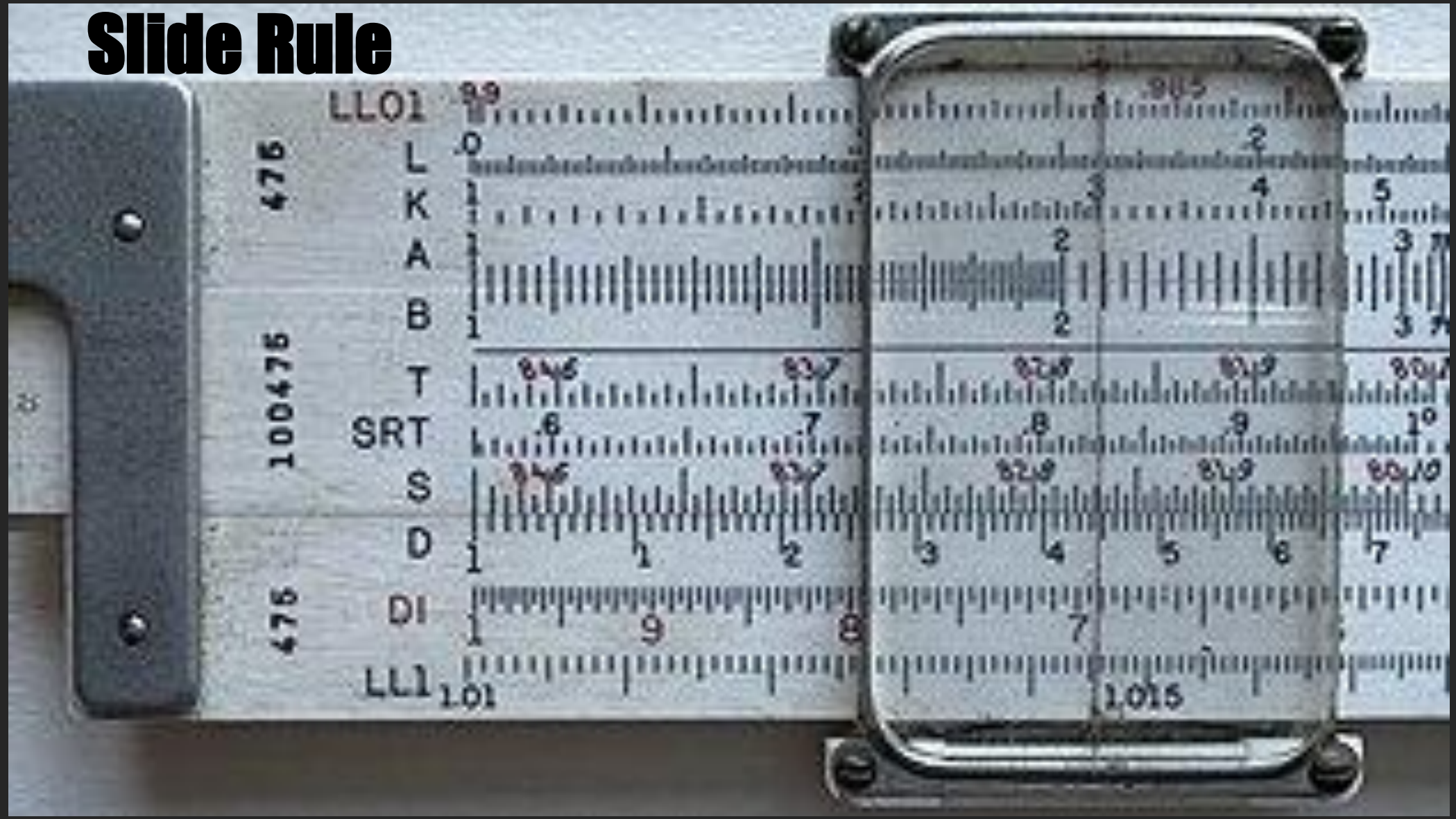
What is a Computer?

- **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

Parts of Japanese Abacus



Slide Rule



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.

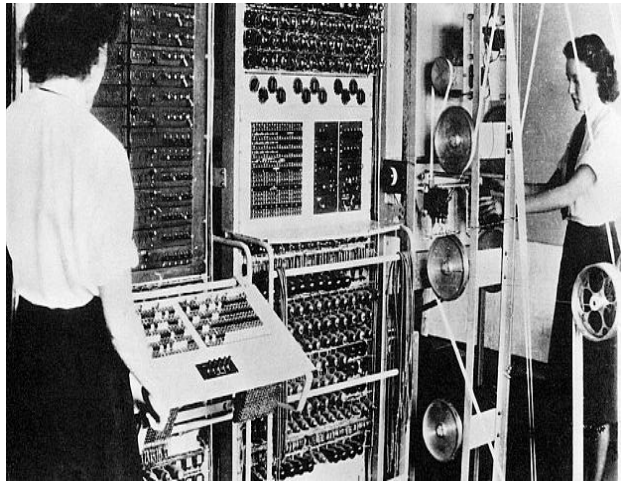
- Slide rules disappeared fast (1972–1976). After the HP-35 (1972), handhelds spread and slide-rule use collapsed during the mid-1970s.

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.

Changes to Data Science Curriculum



Changes to Data Science Curriculum

- Writing code **well** requires putting in your 10,000 hours of practice to:
    - become familiar with packages
    - reading documentation
    - memorizing functions and arguments
    - spending hours on discussion boards when code is not working
- 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.

- These 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

- **Junior Engineering Mindset**

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

- **Senior Engineering Mindset**

  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

- 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
- 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)

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

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.

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).

What is a Data Scientist?

- 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 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

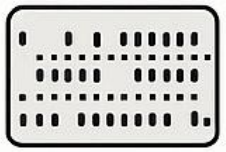
- **EVENTUAL:** more emphasis on domain expertise and statistical modeling, less expertise needed in programming to be a competent data scientist.

  - 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

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 Advent (2022): prompt-based programming

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.