A Brief History of Computing

Table of contents

Outline 1
What is a computer? 2
What is software? 2
How do we describe something to a computer? 3
Algorithms, computer science, software engineering 3
Where this comes back 4
Next lecture 4

Lecture 1A, the first hour of week 1. Course logistics are in csse1001[1]. Lecture 1B, the same day, is 2025-07-29-python-as-a-calculator[2].

Six questions, grouped into four sections below. Five get a tentative definition, a "perhaps…", not a formal claim. The exception is *how do we describe something to a computer?*, which is answered with examples and code listings, not a definition. This hour is one of the two blocks csse1001[3] carves out of assessment (the other is all of week 13). It sets up the definitions that the end of the course returns to. See history-of-computing[4] for the full reference, including the hardware timeline, the Turing machine figure and the language progression.

Outline

- What is a computer?
- What is software?
- How do we describe something to a computer?
- What is an algorithm? What is computer science? What is software engineering?

[1] courses/csse1001/csse1001.pdf
[2] courses/csse1001/2025-07-29-python-as-a-calculator.pdf
[3] courses/csse1001/csse1001.pdf
[4] courses/csse1001/history-of-computing.pdf

What is a computer?

> A computer is a machine that can carry out sequences of arithmetic or logical operations automatically.

The word *machine* has not always applied: the Harvard "Computers" of the 1800s were people doing calculations by hand.

Formally, a **finite state machine** changes state when given input; a **Turing Machine** is a finite state machine with (infinite) read/write memory called tape, and is a *general purpose* computer — one machine that can be programmed to do different tasks. **Alan Turing** gave this definition, showed some problems are *not computable*, and helped break Enigma. **Charles Babbage** designed the first general purpose computer (the Analytical Engine; the schematics are dated 1840), never built in his lifetime — hardware following the design was assembled in 1910 as a four-operation calculator.

A concrete example of a finite state machine is a Turing machine that adds two binary numbers, with `carry` and `no carry` as its two working states. The formal theory belongs to Theory of Computing (COMP2048), so it is background here, not examinable material. The machine is drawn and read transition by transition in history-of-computing[5].

The hardware timeline runs Z3 (1941) → ENIAC (1945) → EDVAC (1949) → UNIVAC 1 (1951) → integrated circuit (1958) → UQ's GE-225 (1962) → Intel 4004 (1971) → 8086 (1978) → Pentium (1993) → today's Threadripper. UQ's own entry is the **GE-225, installed in the Hawken Building (now the Prentice Centre) in 1962**, weighing 900kg.

What is software?

> Software is the complement of hardware. They are instructions for the machine.

Ada Lovelace described an algorithm for the Analytical Engine to compute Bernoulli numbers, making her the first computer programmer. **Grace Hopper** pioneered machine-independent, English-word-based programming and invented COBOL.

The first computer bug is a moth taped to a 1947 logbook page. The entry, made at 15:45 on 9 September, reads:

> Relay #70 Panel F (moth) in relay.
>
> First actual case of bug being found.

[5] courses/csse1001/history-of-computing.pdf

How do we describe something to a computer?

First the Jacquard loom, punch cards and the 5MB punch-card program. Then a ladder of languages, every rung *the same* integer exponentiation program:

machine code → assembly (1947) → Fortran (1957) → COBOL (1959) → C (1972) → Python (1991)

The listings get shorter overall as they get higher-level, though not at every step: 19 lines of x86 machine code, 22 lines of assembly, 11 of Fortran, 19 of COBOL, 7 of C, 5 of Python. By the Python version the loop machinery has vanished entirely:

```
def power(x: int, y: int) -> int:
    ans = 1
    for _ in range(y):
        ans = ans * x
    return ans
```

Python itself was written by **Guido van Rossum** and named after Monty Python — a general-purpose, high-level language meant to support several programming styles (see programming-paradigms[6]) and be fun to use.

Algorithms, computer science, software engineering

> An **algorithm** is an ordered sequence of unambiguous instructions that eventually terminates.
>
> The language we use to express ourselves is less important than what we are expressing: an instruction sequence that does something useful, like sorting a list.

The example for the second statement is a nine-row trace of selection sort on `[26, 54, 93, 17, 77, 31, 44, 55, 20]`, with no code beside it. The trace is drawn in sorting-algorithms[7], and selection sort comes back in week 13.

> **Computer science** is the practice/art of creating algorithms and measuring their relative effectiveness.
>
> **Software engineering** is the practice/art of creating and maintaining software (realized algorithms) in groups.

[6] courses/csse1001/programming-paradigms.pdf
[7] courses/csse1001/sorting-algorithms.pdf

Where this comes back

"Creating algorithms and measuring their relative effectiveness" is the exact brief of the final lecture, 2025-10-28-complexity-and-sorting[8] — five ways to sort a list (sorting-algorithms[9]) and a formal way to compare them (algorithmic-complexity[10]). The course opens and closes on the same definition, and on the same selection-sort trace.

Next lecture

2025-07-29-python-as-a-calculator[11], Lecture 1B: arithmetic expressions and variables.

[8] courses/csse1001/2025-10-28-complexity-and-sorting.pdf
[9] courses/csse1001/sorting-algorithms.pdf
[10] courses/csse1001/algorithmic-complexity.pdf
[11] courses/csse1001/2025-07-29-python-as-a-calculator.pdf