Topic 1:

Programming Language History and Basics

Why Do We Have Programming Languages?

Definition: Programming Language

Early Computers (1 / 2)

1945: ENIAC (Electronic Numerical Integrator And Computer)

First Turing–complete programmable electronic computer.

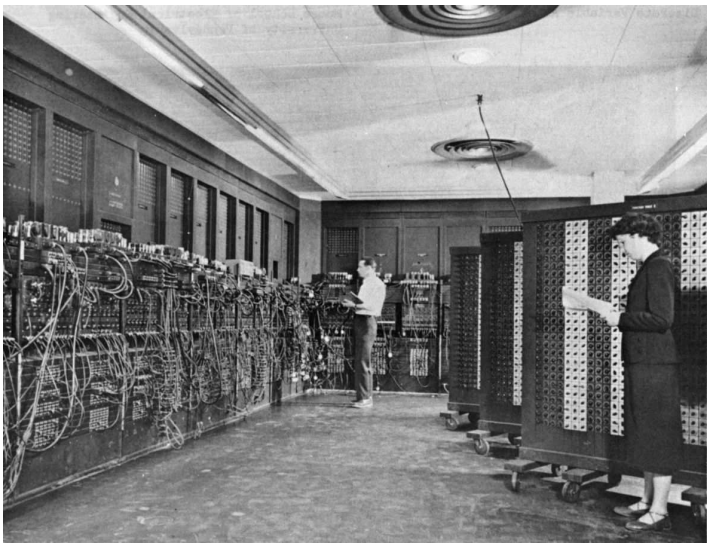


Photo: U.S. Army

Early Computers (2 / 2)

1949: EDVAC (Electronic Discrete Variable Automatic Computer)

A binary stored–program electronic computer

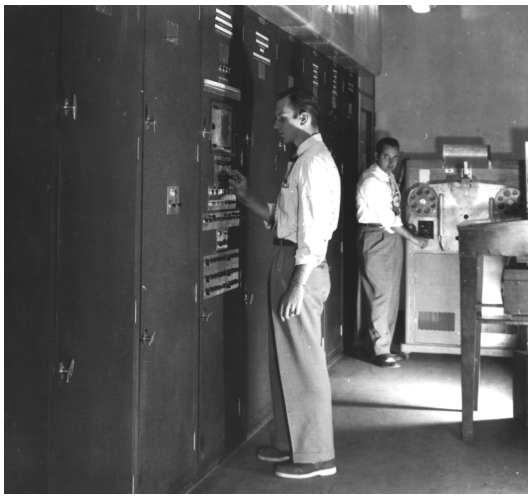


Photo: U.S. Army

The Von Neumann Architecture

. . . is a critical contribution of Von Neumann's report.

Programming Language Progression (1 / 10)

(a) Machine Language: Groups of bits represent CPU instructions ("op codes") and the data / addresses those op codes require.

Example:
```
1010 1001
0000 0010
0110 1001
0000 0101
1000 1101
1010 0000
0000 1111
0110 0000
```

Programming Language Progression (2 / 10)

(a) Machine Language (continued)

To enter programs, the earliest personal computer required programmers to set one toggle switch per bit:

Programming Language Progression (3 / 10)

(b) Assembly Language:

Instruction names (“mnemonics”) made opcodes easier to remember:

A program called an assembler converted this assembly language to machine language.

Programming Language Progression (4 / 10)

(b) Assembly Language (continued)

Here's the assembly version of our MOS 6502 subprogram,

with corresponding byte values in hexadecimal:

```
A9 02       LDA #$02
69 05       ADC #$05
8D A0 0F    STA $0FA0
60          RTS
```


Programming Language Progression (5 / 10)

(c) High–Level Languages (HLLs), a.k.a. 3rd Generation (3GLs):

Why move beyond assembly language?

Programming Language Progression (6 / 10)

(c) High–Level Languages (continued)

Example #1: Fortran (Created: mid–1950s)

Purpose: Automate scientific and engineering calculations

Example:

```
SUBROUTINE ADD(R)
INTEGER R
R = 2+5
RETURN
```


Programming Language Progression (7 / 10)

(c) High–Level Languages (continued again)

Example #1: COBOL (Created: 1959)

Purpose: Business and financial data processing (even today!)

Example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
    DISPLAY "Hello " WITH NO ADVANCING
    DISPLAY "world!"
    STOP RUN.
```

Programming Language Progression (8 / 10)

(d) Fourth–Generation Languages (4GLs):

Why another generation?

4GLs are:

Two examples:

Programming Language Progression (9 / 10)

(e) Fifth–Generation Languages (5GLs)* :

Idea:

Example:

* Not to be confused with Japan's 5th Generation Computer Systems project of the 1980s.

Programming Language Progression (10 / 10)

The greater the abstraction, the lesser the remaining connection to the Von Neumann architecture.

This is not necessarily a bad thing! 3GLs have weaknesses, such as:

Abstraction

We will organize abstractions into two categories:

(1)

(2)

With three sub–categories of each:

(a)

(b)

(c)

Data Abstraction Examples

Basic:

Structured:

Unit:

Control Abstraction Examples

Basic:

Structured:

Unit:

Overview of Computational Paradigms (1 / 5)

Definition: Computational Paradigm

Four major language paradigms:

Others include:

Overview of Computational Paradigms (2 / 5)

(a) The Imperative Paradigm Covers ML, Assembly, and 3GLs

Features Include:

Problem:

Example Languages:

Overview of Computational Paradigms (3 / 5)

(b) The Object–Oriented (OO) Paradigm

The core ideas include:

Overview of Computational Paradigms (4 / 5)

(c) The Functional Paradigm

The core ideas include:

Overview of Computational Paradigms (5 / 5)

(d) The Logic Paradigm

The core ideas include:

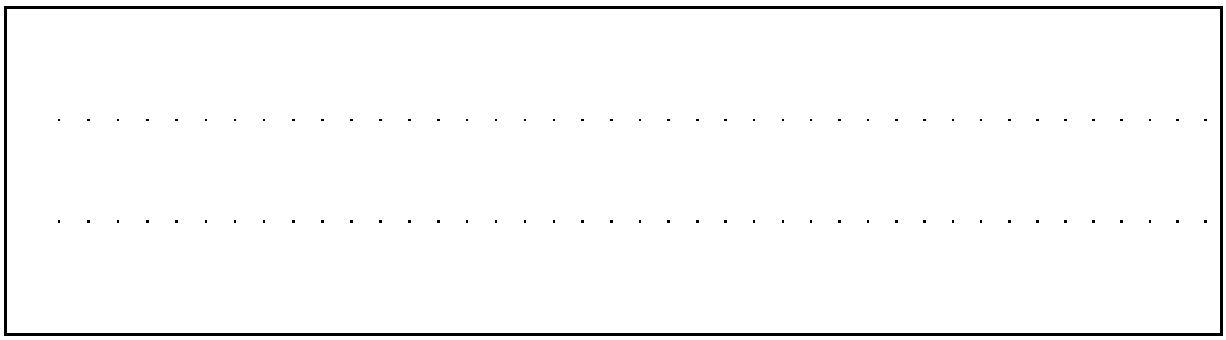
How are Languages Defined?

Historically:

Today:

Language Syntax (1 / 2)

Definition: Syntax



Example, using English syntax:

Incorrect syntax:

Correct syntax:

Language Syntax (2 / 2)

A Grammar Example:

Language Semantics

Definition: Semantics

Three formal methods used to define semantics:

-
-
-

"Why Study PLs? GenAIs Now Write Code!"

Alternate slide title: "Should we change majors?"

Happily for humanity, LLMs and Generative A.I.s still have problems, such as:

- 'Hallucinating' hypotheses
- Producing incorrect results (especially subtle ones!)
- Inability to design/manage complex systems
- Being intuitive and creative