Typology of Programming Languages

Languages and Computers

May 2025

Exercise

What are your favorite and least favorite programming language?

The Tower of Babel

Exercise

How many programming languages are there?

The Tower of Babel

Exercise

How many programming languages are there?

Wikipedia lists around 700. Why so many? Why so different?

Hello, World!

```
def hello():
  print("Hello, World!")
```

Hello, World!

```
def hello():
  print("Hello, World!")
```

- Python is a **general purpose** and **scripting** language.
- Dynamically-typed, interpreted, and supports object-oriented programming, functional programming among other paradigms.
- Used in data science (and AI in particular), build systems, testing, web…

Hello, World!

```
with Ada.Text_IO;

procedure Hello is
begin
  Ada.Text_IO.Put_Line("Hello, World!")
end Hello;
```

Hello, World!

```ada
with Ada.Text_IO;

procedure Hello is
begin
  Ada.Text_IO.Put_Line("Hello, World!")
end Hello;
```

- Ada is a **general purpose** and **systems programming** language.
- Strongly and statically-typed, compiled, and supports object-oriented programming, functional programming, generic programming, and design by contract.
- Mainly used embedded systems and high integrity software.

Hello, World!

```
: HELLO  ( -- )  CR ." Hello, World!" ;
```

Hello, World!

```
: HELLO  ( -- )  CR ." Hello, World!" ;
```

- Forth is a **stack-oriented** programming language.
- Supports interpreting and compiling code on the fly through its shell, and is small enough to fit on 8-bit systems.
- Has been used in many context, including spaceflights such as the 2004 ESA Philae lander which performed the first landing on a comet in 2014.

Hello, World!

Hello, World!

- Piet is a stack-based **esoteric** language.
- Its programs are bitmaps looking like abstract paintings. The pixels' color encode information that the interpreter reads using a "pointer" which moves around the image.
- Turing-complete, it can be used to write programs of any kind.

Paradigms

Definition

A **programming paradigm** is a set of principles and concepts describing a given approach to programming.
These can be implemented directly by a language or through library support.

What is a programming language?

Exercise

What is a programming language?

What is a programming language?

Exercise

What is a programming language?

- Is Python a programming language?
- Is Spanish a programming language?
- Is HTML a programming language?
- Is X86 assembly a programming language?

What is a programming language?

Exercise

What is a programming language?

- Is Python a programming language?
- Is Spanish a programming language?
- Is HTML a programming language?
- Is X86 assembly a programming language?

Natural, *human* languages are a means of communication. We use them to convey ideas so that they can be understood by the listener.

Computers are not capable of *understanding*.

What is a programming language?

As per the Cambridge Dictionnary and Dictionnaire Larousse :

Definition

A programming language is a *system of* ***symbols*** *and* ***rules*** *for writing* ***instructions*** *for computers.*

What is a programming language?

As per the Cambridge Dictionnary and Dictionnaire Larousse :

Definition

A programming language is a *system of* ***symbols*** *and* ***rules*** *for writing* ***instructions*** *for computers.*

- Symbols?
- Rules?
- Instructions?

What is a programming language?

As per the Cambridge Dictionnary and Dictionnaire Larousse :

Definition

A programming language is a *system of* ***symbols*** *and* ***rules*** *for writing* ***instructions*** *for computers.*

- Symbols?
- Rules?
- Instructions?

- Computers?

Section 1

Early Computing (3500 BCE - 1954 AD)

What is a computer?

Exercise

What is a computer? What is computing?

What is a computer?

Exercise

What is a computer? What is computing?

Definition

Computing consists in performing sequences of arithmetic and logical operations.

What is a computer?

Definition

A **computer** is a *person* who performs sequences of arithmetic and logical operations.

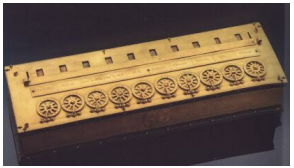
Computers for the Explorer 1 satellite trajectory (1958)

Computing, long before computers

3500 BCE - 1837

Kish Tablet (3500 BCE)

Antikythera mechanism (205-87 BCE)



Pascaline (1652)

Jacquard looms

1804

Joseph Marie Jacquard introduces punched cards to operate looms.

The Analytical Engine

1837

- Conceived by Charles Babbage in 1837.
- Digital mechanical general purpose computer.
- Programmable using punched cards *à la* Jacquard.
  - Both for inputting data and instructions.
  - Could also output punched cards for later use.

The Analytical Engine

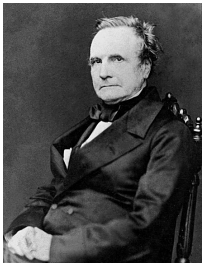
1837

- Conceived by Charles Babbage in 1837.
- Digital mechanical general purpose computer.
- Programmable using punched cards *à la* Jacquard.
  - Both for inputting data and instructions.
  - Could also output punched cards for later use.
- Incorporated an arithmetic logic unit, control flow, conditional branches memory…!

The Analytical Engine

1837

- Conceived by Charles Babbage in 1837.
- Digital mechanical general purpose computer.
- Programmable using punched cards *à la* Jacquard.
  - Both for inputting data and instructions.
  - Could also output punched cards for later use.
- Incorporated an arithmetic logic unit, control flow, conditional branches memory…!
- Turing-complete!



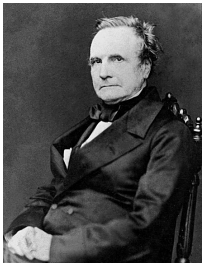
Charles Babbage

Ada Lovelace

The Analytical Engine

1837

- Conceived by Charles Babbage in 1837.
- Digital mechanical general purpose computer.
- Programmable using punched cards *à la* Jacquard.
  - Both for inputting data and instructions.
  - Could also output punched cards for later use.
- Incorporated an arithmetic logic unit, control flow, conditional branches memory…!
- Turing-complete!



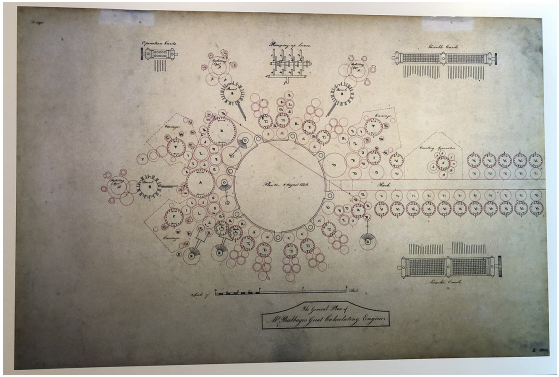
Charles Babbage

Ada Lovelace

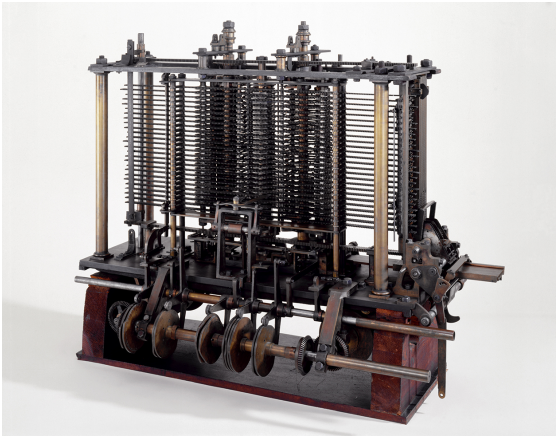
Never built due to conflicts and lack of funding.

The Analytical Engine

1837



Plan diagram of the analytical engine (1840).

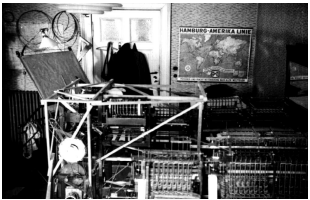


Part of the analytical engine completed by Babbage before his death in 1871.

Konrad Zuse's Z1 and Z3

1936-1941

- First computer based on boolean logic.
- Mechanical, motor-driven, very unreliable.
- Limited programmability.

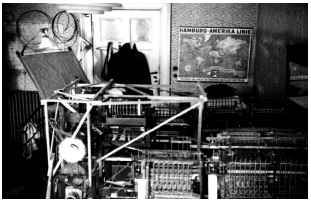


Z1 (1936)

Konrad Zuse's Z1 and Z3

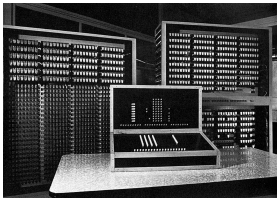
1936-1941

- First computer based on boolean logic.
- Mechanical, motor-driven, very unreliable.
- Limited programmability.



Z1 (1936)

- Reliable, freely programmable.
- First Turing-complete machine.
  - More than 100 years after the analytical engine!



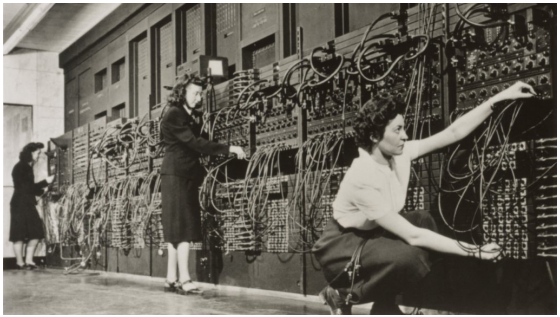
Z3 (1941)

ENIAC

1945

- Designed and built over 4 years
- Financed by the US Army
- First completely electronic, programmable, Turing-complete computer
- Programmed by former human computers!

Cost: $487,000 ($6,900,000 today)

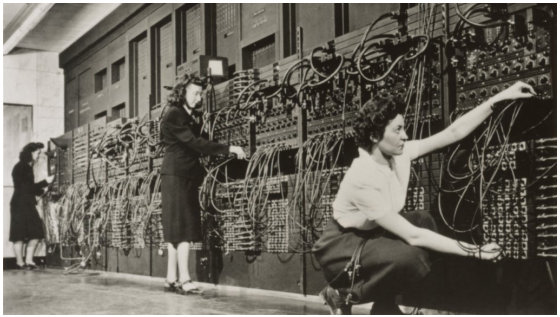


ENIAC

1945

- Designed and built over 4 years
- Financed by the US Army
- First completely electronic, programmable, Turing-complete computer
- Programmed by former human computers!

Cost: $487,000 ($6,900,000 today)

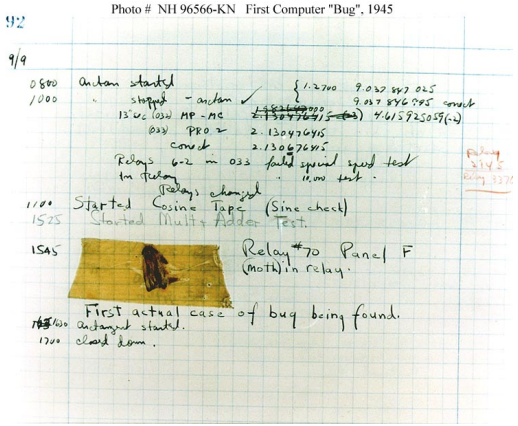


Definition

A **computer** is a *machine* that can automatically perform sequences of arithmetic and logical operations.

The first bug

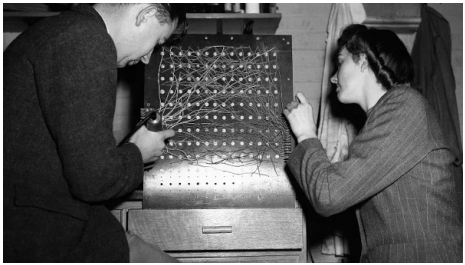
1945, Sep 9th



Grace Hopper finds the first *bug* on a Harvard MkII

ARC Assembly

1947



Andrew and Kathleen Booth (1946)

- *Automatic Relay Computer*, based on Von Neumann's work.
- Kathleen designed and implemented the first assembler and assembly language.

Low and high level languages

Definition

A **low-level** programming language is specific to a given machine or architecture.
Namely: machine code and assembly. Not the point of this class.

Low and high level languages

Definition

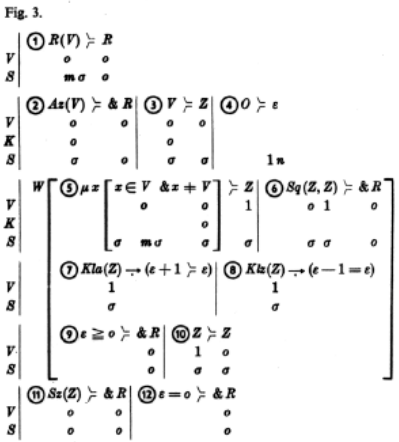
A **low-level** programming language is specific to a given machine or architecture.
Namely: machine code and assembly. Not the point of this class.

Definition

A **high-level** programming language aims to be agnostic of the machine it is to be run on.
These are the "programming languages" we focus on.

Proto-languages

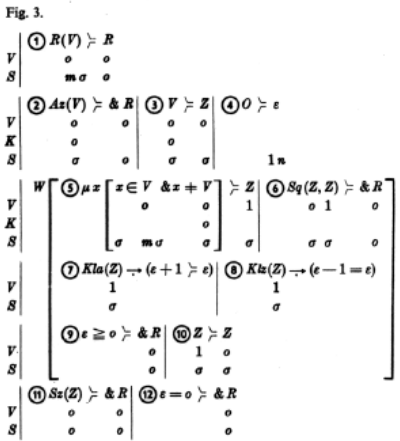
1948-1953



Zuse's Plankalkül (1948)

Proto-languages

1948-1953



Zuse's Plankalkül (1948)

- 1950: Short Code for UNIVAC I
  - Represents mathematic expressions instead of machine instructions
  - Interpreted, up to 50 times slower than machine code
- 1953: Speedcoding for IBM 701
  - Extends assembly with higher-level pseudo-instructions
  - Interpreted, up to 20 times slower than machine code

Commercial computers

UNIVAC I (1950)

IBM 704 (1954)

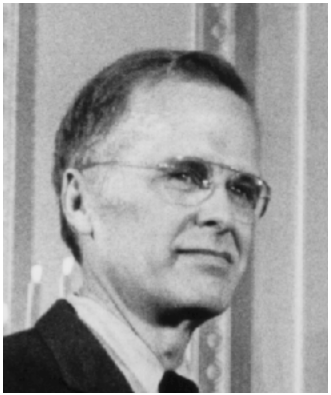
1956 UNIVAC I commercial

Section 2

Early Languages (1954 - 1960)

IBM Mathematical Formula Translating System

1954



John Backus

> *It cost millions to rent machines and yet the cost of programming was as big or bigger. [...] Assembly language was time-consuming. [...] As a consequence, the ultimate goal of the program was often lost in the suffle.*

FORTRAN

1954-1956

- Small team at IBM, lead by John Backus
- Developped from 1954 to 1956
  - First compiler in 1957.
- Goal: making writing programs for the IBM 704 easier

> *It would just mean getting programming done a lot faster.*
>
> *– John Backus*

FORTRAN

1954-1956

- Small team at IBM, lead by John Backus
- Developped from 1954 to 1956
  - First compiler in 1957.
- Goal: making writing programs for the IBM 704 easier

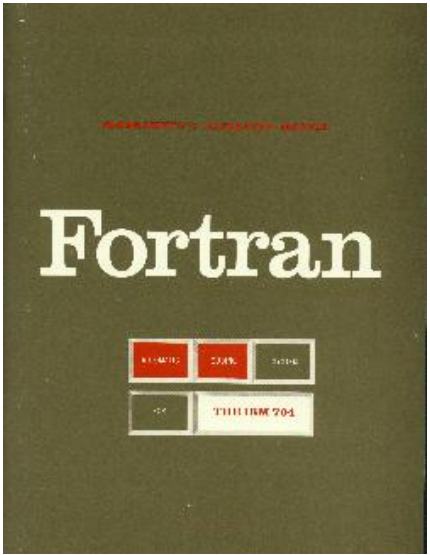
> *It would just mean getting programming done a lot faster.*
>
> *– John Backus*

Definition

A **programming language** is a **developper tool**, used to write *computer programs*.

FORTRAN

1954-1956



Dear regards Walter
John Backus

Programmer's
Reference Manual
October 15, 1956

THE FORTRAN
AUTOMATIC CODING SYSTEM
FOR THE
IBM 704 EDPM®

This manual supersedes all earlier information about the FORTRAN system. It describes the system which will be made available during late 1956, and is intended to permit planning and FORTRAN coding in advance of that time. An Introductory Programmer's Manual and an Operator's Manual will also be issued.

APPLIED SCIENCE DIVISION
AND PROGRAMMING RESEARCH DEPT.
International Business Machines Corporation
590 Madison Ave., New York 22, N. Y.

WORKING COMMITTEE

J. W. BACKUS
R. J. BEEBER
S. BEST
R. GOLDBERG
H. L. HERRICK
R. A. HUGHES
University of California
Radiation Laboratory,
Livermore, Calif.
I. B. MITCHELL
P. A. NELSON
R. NUTT
United Aircraft Corp.,
East Hartford, Conn.
D. SAYRE
P. B. SHERIDAN
H. STERN
I. ZILLER

FORTRAN

1954-1956

```
      DIMENSION A(99)
      REAL MEAN
      READ(1,5)N
 5    FORMAT(I2)
      READ(1,10)(A(I),I=1,N)
 10   FORMAT(6F10.5)
      SUM=0.0
      DO 15 I=1,N
 15      SUM=SUM+A(I)
      MEAN=SUM/FLOAT(N)
      NUMBER=0
      DO 20 I=1,N
         IF (A(I) .LE. MEAN) GOTO 20
         NUMBER=NUMBER+1
```

Exercise

What does this program do?

```
C     (continued)
 20   CONTINUE
      WRITE (2,25) MEAN,NUMBER
 25   FORMAT(11H MEAN = ,F10.5,
        5X,21H NUMBER SUP = ,I5)
      STOP
      END
```

FORTRAN

1954-1956

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)	IDENTIFICATION (73–80)
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		DIMENSION A(999)	
		FREQUENCY 30(2,1,10), 5(100)	
		READ 1, N, (A(I), I = 1,N)	
1		FORMAT (I3/(12F6.2))	
		BIGA = A(1)	
5		DO 20 I = 2,N	
30		IF (BIGA-A(I)) 10,20,20	
10		BIGA = A(I)	
20		CONTINUE	
		PRINT 2, N, BIGA	
2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
		STOP 77777	

FORTRAN

1954-1956

Main goal: user satisfaction. Industrial language, not academic.

- Easy to learn
- Easy to write
  - Arrays, `DO` loops, subprograms, arithmetic expressions.
- Easy to work with
  - Useful abstractions (`I/O`).

FORTRAN

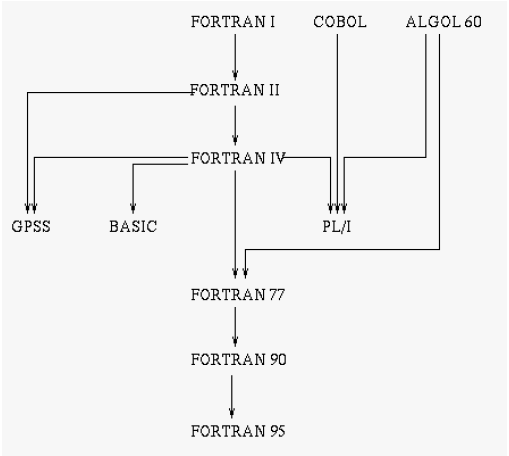
1954-1956

Main goal: user satisfaction. Industrial language, not academic.

- Easy to learn
- Easy to write
  - Arrays, `DO` loops, subprograms, arithmetic expressions.
- Easy to work with
  - Useful abstractions (`I/O`).
- Efficient
  - *The* main selling point.
  - First language with an *optimizing compiler*!
  - Writing FORTRAN programs is as efficient as hand writing assembly! (if not more efficient)

FORTRAN - Legacy

1954-today



Last standard: Fortran 2023.

COBOL

1959

- Developped to provide a COmmon Business-Oriented Language to reduce language switching costs.
- Designed by multiple committees, with a first specification made over 3 months.

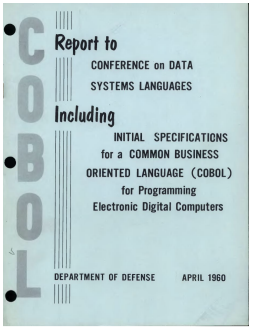
COBOL

1959

- Developped to provide a COmmon Business-Oriented Language to reduce language switching costs.
- Designed by multiple committees, with a first specification made over 3 months.

Definition

General Purpose Languages (GPL) are designed for use in most contexts.



COBOL 60 Report

COBOL

1959

```
IDENTIFICATION DIVISION.

PROGRAM-ID. INOUT.

* Read a file, add infos to records,
* and save as another file.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT INP-FIL ASSIGN TO INFILE.
SELECT OUT-FIL ASSIGN TO OUTFILE.
```

```
DATA DIVISION.
FILE SECTION.

FD    INP-FIL
      LABEL RECORDS STANDARD
      DATA RECORD IS REC-IN.
01    REC-IN.
      05 ALPHA-IN PIC A(4).
      05 SP-CH-IN PIC X(4).
      05 NUM-IN   PIC 9(4).
FD    OUT-FIL
      LABEL RECORDS STANDARD
      DATA RECORD IS REC-OUT.
01    REC-OUT.
      05 ALPHA-OUT PIC A(4).
      05 SP-CH-OUT PIC X(4).
      05 NUM-OUT   PIC 9(4).
      05 EXTRAS    PIC X(16).
```

Quotes on COBOL

> *The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense.*
>
> –
>
> *Edsger Dijkstra*

COBOL

1959

Commodore Grace Hopper with President Reagan (1983)

- Based mostly on Grace Hopper's work:
  - Her earlier FLOW-MATIC language.
  - Her philosophy of using english words and syntax in programming languages.
- Imposed by the DOD, thanks to Grace Hopper:
  - To have a contract, a COBOL compiler was required.
  - Any material bought on governmental funding had to have a COBOL compiler.

Most used language worldwide for a *long* time. Last standard:

COBOL

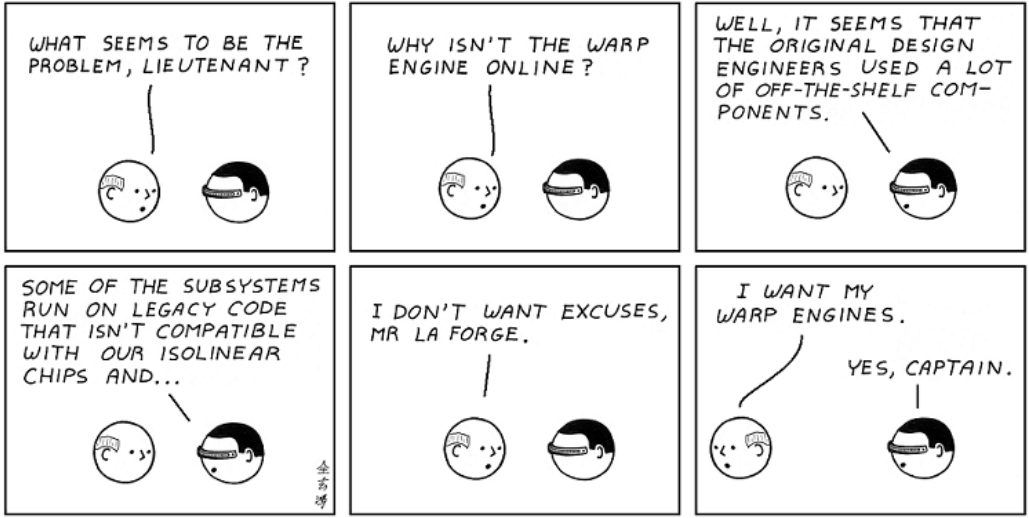
1959

Commodore Grace Hopper with President Reagan (1983)

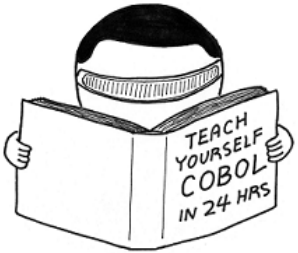
- Based mostly on Grace Hopper's work:
  - Her earlier FLOW-MATIC language.
  - Her philosophy of using english words and syntax in programming languages.
- Imposed by the DOD, thanks to Grace Hopper:
  - To have a contract, a COBOL compiler was required.
  - Any material bought on governmental funding had to have a COBOL compiler.

Most used language worldwide for a *long* time. Last standard: COBOL-2023.

COBOL - Legacy



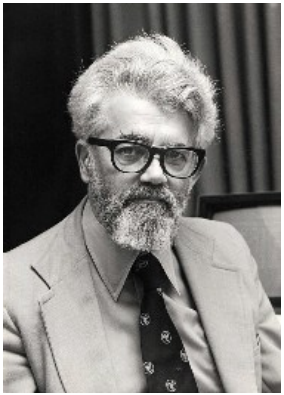
COBOL - Legacy



New technologies will come and go
but COBOL is forever.

Lisp

1960



John McCarthy

- Developped by John McCarthy in the late 50s for work in AI.
- Pioneering language, introducing concepts such as:
  - Recursion.
  - Conditionnals.
  - First class functions.
  - Expression-based programming.
  - Garbage collection.
  - Functionnal programming.
  - The `eval` function.
- First implementation by Steve Russell on an IBM 704.

> *If Fortran had allowed recursion, I would have used it.*

Lisp - Example

```
eval[e;p] = [
  atom[e] → [assoc[e;p];
  atom[car[e]] → [
    eq[car[e];QUOTE] → cadr[e];
    eq[car[e];ATOM] → atom[eval[cadr[e];p]];
    eq[car[e];EQ] → eq[eval[cadr[e];p];eval[caddr[e];p]];
    eq[car[e];COND] → evcon[cdr[e];p];
    eq[car[e];CAR] → car[eval[cadr[e];p]];
    eq[car[e];CDR] → cdr[eval[cadr[e];p]];
    eq[car[e];CONS] → cons[eval[cadr[e];p];eval[caddr[e];p]];
       T → eval[cons[assoc[car[e];p];evlis[cdr[e];p]];p]];
    eq[caar[e];LABEL] → eval[cons[caddar[e];cdr[e]];cdr[e]];
       cons[list[cadar[e];car[e]];p]];
    eq[caar[e];LAMBDA] → eval[caddar[e];append[pair[cadar[e];
       evlis[cdr[e];p];p]]]]
```

Snippet of 1960 LISP-I Manual describing the `eval` function

Lisp - Legacy

One of the most influential languages, if not *the* most influent.

A plethora of dialects:

- Lisp for high-performance functionnal programming? Scheme
- Lisp to do high-level object programming? Common Lisp Object System
- Lisp to write compilers and interpreters? Racket
- Lisp in your JVM? Clojure
- Lisp in your editor? Emacs Lisp
- Lisp but French? LeLisp

ALGOL 60

1960

- Updated version of ALGOL 58, designed as an academic language:
  1. Usable for algorithm publications in scientific reviews.
  2. As close as possible to the usual mathematical notations.
  3. Readable without assistance.
  4. Automatically translatable into machine code.
- Designed by an international team.
- Introduced fundamental notions such as compound statements, local variables, blocks…

Directly competing with FORTRAN, which prevented it from widespread adoption.

ALGOL 60

John McCarthy, Fritz Bauer, Joe Wegstein

John Backus, Peter Naur, Alan Perlis

ALGOL 60 - One syntax, three lexics

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / 6.02 x 10^23;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
a[i+1] := (a[i] + pi * r^2) / 6.02E23;
```

ALGOL 60 - One syntax, three lexics

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / 6.02 x 10^23;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
    a[i+1] := (a[i] + pi * r^2) / 6.02E23;
or  a(/i+1/) := (a(/i/) + pi * r ** 2) / 6,02e23;
```

ALGOL 60 - One syntax, three lexics

Reference language (used in the ALGOL-60 Report)

a[i+1] := (a[i] + pi x r^2) / 6.02 x 10^23;

Publication language

$a_{i+1} \leftarrow \{a_i + \pi \times r^2\}/6.02 \times 10^{23}$;

Hardware representations – implementation dependent

```
    a[i+1] := (a[i] + pi * r^2) / 6.02E23;
or  a(/i+1/) := (a(/i/) + pi * r ** 2) / 6,02e23;
or  A(.I+1.) .= (A(.I.) + PI * R 'POWER' 2) / 6.02'23.,
```

ALGOL 60 - For Loops BNF

`for` loop syntax

```
<for statement> ::= <for clause> <statement>
                  | <label>: <for statement>

<for clause> ::= for <variable> := <for list> do

<for list> ::= <for list element>
             | <for list> , <for list element>

<for list element>  ::= <arithmetic expression>
   | <arithmetic expression> step <arithmetic expression>
                            until <arithmetic expression>
   | <arithmetic expression> while <Boolean expression>
```

ALGOL 60 - For Loops

for step until

```
for i := 1 step 2 until N do
      a[i] := b[i];
```

for while

```
for newGuess := Improve (oldGuess)
      while abs (newGuess - oldGuess)
            > 0.0001 do
   oldGuess := newGuess;
```

for enumerations

```
for days := 31,
      if mod( year, 4 ) = 0
      then 29 else 28,
      31, 30, 31, 30, 31,
      31, 30, 31, 30, 31
    do
   . . .
```

for complete

```
for i := 3, 7,
      11 step 1 until 16,
      i / 2 while i >= 1,
      2 step i until 32 do
        print (i);
```

ALGOL 60 - Example

```
begin
  comment The mean of numbers and the number of greater values;
  integer n;
  read(n);
  begin
    real array a[1:n];
    integer i, number;
    real sum, mean;
    for i := 1 step 1 until n do read (a[i]);
    sum := 0;
    for i := 1 step 1 until n do sum := sum + a[i];
    mean := sum / n;
    number := 0;
    for i := 1 step 1 until n do
      if a[i] > mean then
        number := number + 1;
    write ("Mean = ", mean, "Number sups = ", number);
  end
end
```

ALGOL 58 & 60 - Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.
- Backus-Naur Form

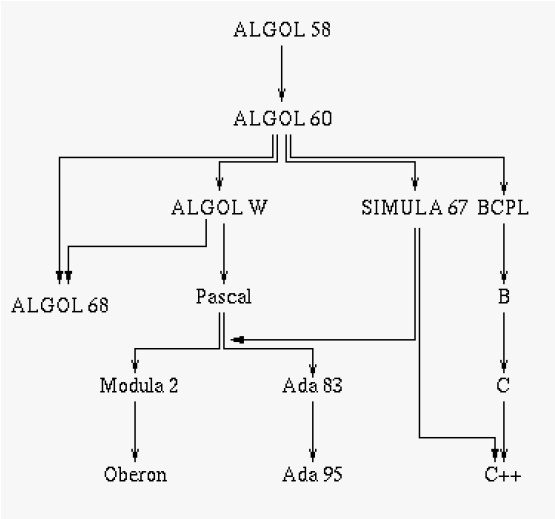
ALGOL 58 & 60 - Legacy

- block,
- call by value, call by name,
- typed procedures,
- declaration scope,
- dynamic arrays,
- `own` variables,
- side effects,
- global and local variables,
- primary, term, factor,
- `step`, `until`, `while`, `if then else`,
- bound pair,
- display stack technique,
- thunks,
- activation records,
- recursive descent parser.
- Backus-Naur Form

> *Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors.*
>
> *– C.A.R. Hoare*

“ALGOL-like” languages

1958-today



Section 3

Before Personal Computers (1960 - 1975)

Miniaturization on its way

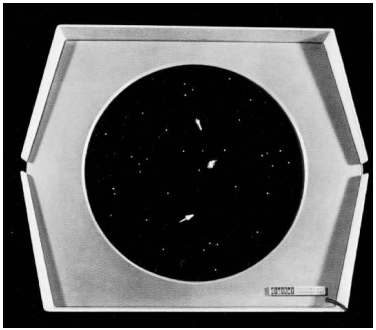
1960

First minicomputer: the PDP-1

Removable disks on IBM 1311 (6 disks, 2.6MB)

Spacewar!

1962



Steve Russell and a PDP-1, 2002

> *The game of Spacewar! blossoms spontaneously wherever there is a graphics display connected to a computer.*
>
> *– Alan Kay*

BASIC

1964

```
 20 DIM A(99)
 30 INPUT N
 40 FOR I = 1 TO N
 50 INPUT A(I)
 60 LET S = S + A(I)
 70 NEXT I
 80 LET M = S / N
 90 LET K = 0
100 FOR I = 1 TO N
110 IF A(I) < M THEN 130
120 LET K = K + 1
130 NEXT I
140 PRINT "MEAN = ", M
150 PRINT "NUMBER SUP = ", K
160 STOP
170 END
```

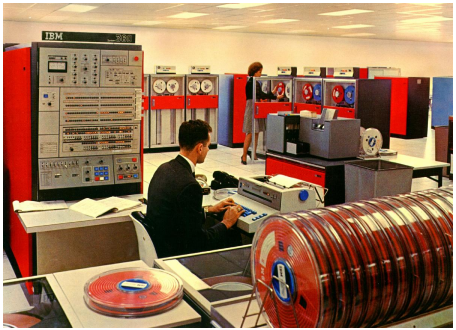
- Beginner's All-purpose Symbolic Instruction Code, by J.Kemeny and T.Kurtz.
- Easy, simple, interpreted, with *many* dialects.

Most people born 1955-2000 learned programming with a BASIC dialect.

PL/I

1965

- Designed by a committee at IBM, wanting a common language for all use cases for the IBM System/360
- Specified 1965, first compiler 1966, standardized 1976



IBM 360, first integrated-circuit based computer

PL/I

1965

- Supposedly “includes” FORTRAN IV, ALGOL 60, COBOL 60 and JOVIAL
- Aims to address all the needs:
  - scientific (floats, arrays, procedures, efficient computation)
  - business (fixed points, fast asychronous I/O, string processing functions, search and sort routines)
  - real time
  - filtering
  - bit strings
  - lists

PL/I Surprises

- No reserved keywords in PL/I.

```
IF IF = THEN THEN
  THEN = ELSE
  ELSE ELSE = IF
```

PL/I Surprises

- No reserved keywords in PL/I.

```
IF IF = THEN THEN
  THEN = ELSE
  ELSE ELSE = IF
```

- Abbreviations: DCL for DECLARE, ...

PL/I Surprises

- No reserved keywords in PL/I.

```
IF IF = THEN THEN
  THEN = ELSE
  ELSE ELSE = IF
```

- Abbreviations: DCL for DECLARE, …
- 25 + 1/3 yields 5.3333333333 while 25 + 01/3 behaves as expected…

PL/I Surprises

- No reserved keywords in PL/I.

```
IF IF = THEN THEN
  THEN = ELSE
  ELSE ELSE = IF
```

- Abbreviations: DCL for DECLARE, ...
- 25 + 1/3 yields 5.3333333333 while 25 + 01/3 behaves as expected...
- This loop is executed zero times.

```
DO I = 1 TO 32/2,
  Statements  END;
```

PL/I Surprises

- No reserved keywords in PL/I.

```
IF IF = THEN THEN
  THEN = ELSE
  ELSE ELSE = IF
```

- Abbreviations: DCL for DECLARE, …
- 25 + 1/3 yields 5.3333333333 while 25 + 01/3 behaves as expected…
- This loop is executed zero times.

```
DO I = 1 TO 32/2,
  Statements  END;
```

- “Advanced” control structures.

```
GO TO I,(1,2,3,92)
```

PL/I - Example

```
EXAMPLE : PROCEDURE OPTIONS (MAIN);
  /* Find the mean of n numbers and the number of
     values greater than it */
  GET LIST (N);
  IF N > 0 THEN
      BEGIN;
      DECLARE MEAN, A(N), DECIMAL POINT
              NUM DEC FLOAT INITIAL(0),
              NUMBER FIXED INITIAL (0)
      GET LIST (A);
      DO I = 1 TO N;
        SUM = SUM + A(I);
      END
      MEAN = SUM / N;
      DO I = 1 TO N;
        IF A(I) > MEAN THEN
          NUMBER = NUMBER + 1;
      END
      PUT LIST ('MEAM = ', MEAN,
                'NUMBER SUP = ', NUMBER);
END EXAMPLE;
```

Quotes on PL/I

> *When FORTRAN has been called an infantile disorder, full PL/1, with its growth characteristics of a dangerous tumor, could turn out to be a fatal disease.*
>
> *– Edsger Dijkstra*

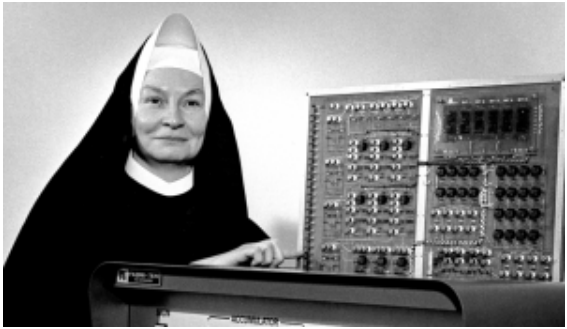
Quotes on PL/I

> *When FORTRAN has been called an infantile disorder, full PL/1, with its growth characteristics of a dangerous tumor, could turn out to be a fatal disease.*
>
> *– Edsger Dijkstra*

> *Using PL/I must be like flying a plane with 7000 buttons, switches, and handles to manipulate in the cockpit. I absolutely fail to see how we can keep our growing programs firmly within our intellectual grip when by its sheer baroqueness, the programming language–our basic tool, mind you!–already escapes our intellectual control.*
> *And if I have to describe the influence PL/I can have on its users, the closest metaphor that comes to my mind is that of a drug.*
>
> *– Edsger Dijkstra*

First computer science PHD

1965



Mary Kenneth Keller

> *We're having an information explosion… and it's certainly obvious that information is of no use unless it's available.*

APL

1966

Kenneth E. Iverson

APL

1966

Kenneth E. Iverson

"A Programming Language"

Quotes on APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> *– David Given*

Quotes on APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> *– David Given*

> *APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.*
>
> *– Edsger Dijkstra, 1968*

Quotes on APL

> *APL, in which you can write a program to simulate shuffling a deck of cards and then dealing them out to several players in four characters, none of which appear on a standard keyboard.*
>
> *– David Given*

> *APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.*
>
> *– Edsger Dijkstra, 1968*

> *By the time the practical people found out what had happened; APL was so important a part of how IBM ran its business that it could not possibly be uprooted.*
>
> *– Michael S. Montalbano, 1982*

Writing APL

Example Program: Prime Numbers up to R

Writing APL

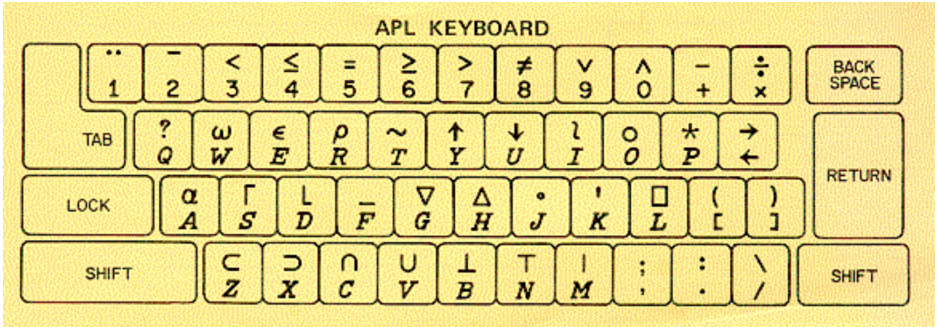
Example Program: Prime Numbers up to R

$$(\sim R \in R \circ . \times R)/R \leftarrow 1 \downarrow \iota R$$

Writing APL

Example Program: Prime Numbers up to R

$$(\sim R \in R \circ . \times R)/R \leftarrow 1 \downarrow \iota R$$



Luckily, there's a dedicated keyboard!

APL - Legacy

1966

Definition

Array programming performs operations directly on arrays, removing the need for loops. This allows for easy parallelization of computations.

Simula 67

1967

Ole-Johan Dahl & Kristen Nygaard (1963)

- Successor to Simula I (1962).
  - Originally built on top of ALGOL 60.
- Introduces:
  - The concept of object.
  - The concept of class.
  - Literal objects (constructors).
  - The concept of inheritance.
  - The concept of virtual methods.
  - Attribute hiding.

The basis of most modern object-oriented programming.

Simula 67 - Example

```
class Shape(x, y); integer x; integer y;
virtual: procedure draw is procedure draw;;
begin
    comment -- get the x & y components for the object --;
    integer procedure getX;
        getX := x;
    integer procedure getY;
        getY := y;
    comment -- set the x & y coordinates for the object --;
    integer procedure setX(newx); integer newx;
        x := newx;
    integer procedure setY(newy); integer newy;
        y := newy;
    comment -- move the x & y position of the object --;
    procedure moveTo(newx, newy); integer newx; integer newy;
        begin
            setX(newx);
            setY(newy);
        end moveTo;
    procedure rMoveTo(deltax, deltay); integer deltax; integer deltay;
        begin
            setX(deltax + getX);
            setY(deltay + getY);
```

Simula 67 - Example

```
Shape class Rectangle(width, height);
    integer width; integer height;
begin
    comment -- get the width & height of the object --;
    integer procedure getWidth;
        getWidth := width;
    integer procedure getHeight;
        getHeight := height;
    comment -- set the width & height of the object --;
    integer procedure setWidth(newwidth); integer newwidth;
        width := newwidth;
    integer procedure setHeight(newheight); integer newheight;
        height := newheight;
    comment -- draw the rectangle --;
    procedure draw;
        begin
            Outtext("Drawing a Rectangle at:(");
            Outint(getX, 0);  Outtext(",");  Outint(getY, 0);
            Outtext("), width ");  Outint(getWidth, 0);
            Outtext(", height ");  Outint(getHeight, 0);
            Outimage;
        end draw;
end Rectangle;
```

Simula 67 - Example

```
Shape class Circle(radius); integer radius;
begin
    comment -- get the radius of the object --;
    integer procedure getRadius;
        getRadius := radius;

    comment -- set the radius of the object --;
    integer procedure setRadius(newradius); integer newradius;
        radius := newradius;

    comment -- draw the circle --;
    procedure draw;
        begin
            Outtext("Drawing a Circle at:(");
            Outint(getX, 0);
            Outtext(",");
            Outint(getY, 0);
            Outtext("), radius ");
            Outint(getRadius, 0);
            Outimage;
        end draw;
end Circle;
```

Simula 67 - Example

```
comment -- declare the variables used --;
ref(Shape) array scribble(1:2);
ref(Rectangle) arectangle;
integer i;

comment -- populate the array with various shape instances --;
scribble(1) :- new Rectangle(10, 20, 5, 6);
scribble(2) :- new Circle(15, 25, 8);

comment -- iterate on the list, handle shapes polymorphically --;
for i := 1 step 1 until 2 do
    begin
        scribble(i).draw;
        scribble(i).rMoveTo(100, 100);
        scribble(i).draw;
    end;

comment -- call a rectangle specific instance --;
arectangle :- new Rectangle(0, 0, 15, 15);
arectangle.draw;
arectangle.setWidth(30);
arectangle.draw;
```

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - ★ Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.
  - Declaration modes to supplement the type of the variable.
    - `FLEX` to have a dynamically sized array.
    - `HEAP` and `LOC` to specify memory space.
    - `LONG` and `SHORT` to impact the memory size of the variable.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.
  - Declaration modes to supplement the type of the variable.
    - `FLEX` to have a dynamically sized array.
    - `HEAP` and `LOC` to specify memory space.
    - `LONG` and `SHORT` to impact the memory size of the variable.
  - Coercions and a coercion hierarchy.
    - 5 levels from *soft* to *weak* to *meek* to *firm* to *strong*.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.
  - Declaration modes to supplement the type of the variable.
    - `FLEX` to have a dynamically sized array.
    - `HEAP` and `LOC` to specify memory space.
    - `LONG` and `SHORT` to impact the memory size of the variable.
  - Coercions and a coercion hierarchy.
    - 5 levels from *soft* to *weak* to *meek* to *firm* to *strong*.
  - `PRAGMAT` directives.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.
  - Declaration modes to supplement the type of the variable.
    - `FLEX` to have a dynamically sized array.
    - `HEAP` and `LOC` to specify memory space.
    - `LONG` and `SHORT` to impact the memory size of the variable.
  - Coercions and a coercion hierarchy.
    - 5 levels from *soft* to *weak* to *meek* to *firm* to *strong*.
  - `PRAGMAT` directives.
  - Parallel processing.

ALGOL 68

1968

Successor to ALGOL 60, designed to be *much* wider.

- Focused on *orthogonality* and security.
- Adds a myriad of concepts to ALGOL 60.
  - 60 reserved keywords.
    - Including `FI`, `OD`, `ESAC`… to end complex statements.
  - Many (optional) special APL-like characters.
  - Declaration modes to supplement the type of the variable.
    - `FLEX` to have a dynamically sized array.
    - `HEAP` and `LOC` to specify memory space.
    - `LONG` and `SHORT` to impact the memory size of the variable.
  - Coercions and a coercion hierarchy.
    - 5 levels from *soft* to *weak* to *meek* to *firm* to *strong*.
  - `PRAGMAT` directives.
  - Parallel processing.
  - …

ALGOL 68

1968

- Very criticized, including by members of its own design committee such as C.A.R. Hoare or Edsger Dijkstra.
- Main reproach: abandonning the simplicity of ALGOL 60 and being overly complex.

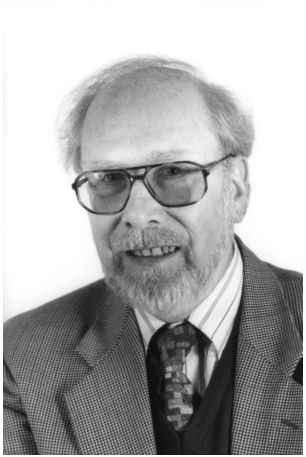
ALGOL 68

1968

- Very criticized, including by members of its own design committee such as C.A.R. Hoare or Edsger Dijkstra.
- Main reproach: abandonning the simplicity of ALGOL 60 and being overly complex.
- Ultimately revised (and simplified) in 1973, but it was already too late.
- Not widely used, but influential nonetheless.

ALGOL W & Pascal

1966-1970

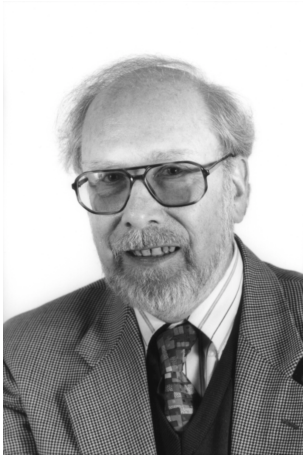


Niklaus Wirth

- 1966: ALGOL W, improvement over ALGOL 60 adding strings, references, `while` and `case` statements…
  - Overall improvement of ALGOL 60, competing with ALGOL 68 as the successor
  - Ultimately considered “too little of an improvement”, ALGOL 68 was chosen instead

ALGOL W & Pascal

1966-1970



Niklaus Wirth

- 1966: ALGOL W, improvement over ALGOL 60 adding strings, references, `while` and `case` statements…
  - Overall improvement of ALGOL 60, competing with ALGOL 68 as the successor
  - Ultimately considered “too little of an improvement”, ALGOL 68 was chosen instead
- 1970: Pascal, developped from ALGOL W
  - Keep the ALGOL 60 structure and mindset, but obtain FORTRAN’s performances.
  - Features rich strong typing:
    - ★ Enumerated types.
    - ★ Interval types.
    - ★ Set types.
    - ★ Record types.
  - Very successful in the 70s.
  - Many influential descendants such as Ada, Eiffel, Modula-2, Oberon.

ALGOL 68 Samples

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

ALGOL 68 Samples

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

ALGOL 68 Samples

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

ALGOL 68 Samples

Assignments

```
real twice pi = 2 * real pi = 3.1415926;
```

Complex Expressions

```
(int sum := 0; for i to N do sum +:= f(i) od; sum)
```

Procedures

```
proc max of real (real a, b) real:
  if a > b then a else b fi;
```

Ternary Operator

```
proc max of real (real a, b) real: (a > b | a | b);
```

ALGOL 68 Samples

Arrays, Functional Arguments

```
proc apply (ref [] real a, proc (real) real f):
  for i from lwb a to upb a do a[i] := f(a[i]) od;
```

ALGOL 68 Samples

Arrays, Functional Arguments

```
proc apply (ref [] real a, proc (real) real f):
  for i from lwb a to upb a do a[i] := f(a[i]) od;
```

User Defined Operators

```
prio max = 9;

op max = (int a,b) int: (a>b | a | b);
op max = (real a,b) real: (a>b | a | b);
op max = (compl a,b) compl: (abs a > abs b | a | b);

op max = ([]real a) real:
   (real x := - max real;
    for i from lwb a to upb a
       do (a[i]>x | x:=a[i]) od;
    x);
```

Quotes on ALGOL 68

> *ALGOL 68 is a language with a lot of "history". The reader will hear of discord, resignations, unreadable documents, a minority report, and all manner of politicking.*
>
> –
>
> *A History of ALGOL 68, C.H. Lindsey*

Quotes on ALGOL 68

> *ALGOL 68 is a language with a lot of "history". The reader will hear of discord, resignations, unreadable documents, a minority report, and all manner of politicking.*
>
> –
>
> *A History of ALGOL 68, C.H. Lindsey*

> *[...] The best we could do was to send with it a minority report, stating our considered view that, "... as a tool for the reliable creation of sophisticated programs, the language was a failure." [...]*
>
> –
>
> *C. A. R. Hoare in his Oct 1980 Turing Award Lecture*

Quotes on ALGOL 68

> *The more I see it, the more unhappy I become.*
> *[...] Size and complexity of the defining apparatus you needed terrify me. Being well-acquainted with your ingenuity I think it a safe assumption that ALGOL 68 as conceived can hardly be defined by significantly more concise and transparent means.*
>
> –
>
> *Edsger Dijkstra*

Quotes on ALGOL 68

> *The more I see it, the more unhappy I become.*
> *[...] Size and complexity of the defining apparatus you needed terrify me. Being well-acquainted with your ingenuity I think it a safe assumption that ALGOL 68 as conceived can hardly be defined by significantly more concise and transparent means.*
>
> –
>
> *Edsger Dijkstra*

> *[...] it was said that A68's popularity was inversely proportional to [...] the distance from Amsterdam*
>
> –
>
> *Guido van Rossum*

SRI International's Augmentation Research Center

1964

Douglas Engelbart

The first mouse!

1968 SRI demonstration

Mother of all demos

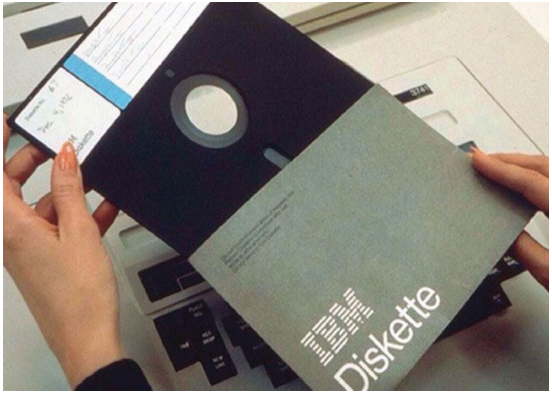
1968, Dec 9th

According to Herb Sutter, this demonstrated prototypes for:

- The personal computer for dedicated individual use all day long.
- The mouse.
- Internetworks.
- Network service discovery.
- Live collaboration and desktop/app sharing.
- Hierarchical structure within a file system and within a document.
- Cut/copy/paste, with drag-and-drop.
- Paper metaphor for word processing.
- Advanced pattern search and macro search.
- Keyword search and multiple weighted keyword search.
- Catalog-based information retrieval.
- Flexible interactive formatting and line drawing.
- Hyperlinks within a document and across documents.
- Tagging graphics, and parts of graphics, as hyperlinks.
- Shared workgroup document collaboration with annotations etc.
- Live shared workgroup collaboration with live audio/video teleconference in a window.

Floppy disks

1971



IBM releases the first floppy disks.

The original 8 inch floppy disk had “the capacity of 3000 punched cards”.

INTERCAL

1972

```
DO ,1 <- #13
PLEASE DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #108
DO ,1 SUB #3 <- #112
DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #194
DO ,1 SUB #7 <- #48
PLEASE DO ,1 SUB #8 <- #22
DO ,1 SUB #9 <- #248
DO ,1 SUB #10 <- #168
DO ,1 SUB #11 <- #24
DO ,1 SUB #12 <- #16
DO ,1 SUB #13 <- #162
PLEASE READ OUT ,1
PLEASE GIVE UP
```

- Developped by Don Woods and Jim Lyon in 1972.
- Famous for its PLEASE modifier and COMEFROM statement.
- Also features a DONT loop in complement of DO.

INTERCAL

1972

```
DO ,1 <- #13
PLEASE DO ,1 SUB #1 <- #238
DO ,1 SUB #2 <- #108
DO ,1 SUB #3 <- #112
DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #194
DO ,1 SUB #7 <- #48
PLEASE DO ,1 SUB #8 <- #22
DO ,1 SUB #9 <- #248
DO ,1 SUB #10 <- #168
DO ,1 SUB #11 <- #24
DO ,1 SUB #12 <- #16
DO ,1 SUB #13 <- #162
PLEASE READ OUT ,1
PLEASE GIVE UP
```

- Developped by Don Woods and Jim Lyon in 1972.
- Famous for its PLEASE modifier and COMEFROM statement.
- Also features a DONT loop in complement of DO.

Exercise

What does this program do?

INTERCAL

1972

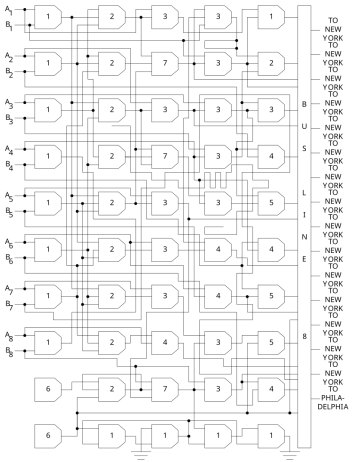


Diagram from the manual explaining the “select” operator

INTERCAL

1972

> *The full name of the compiler is "Compiler Language With No Pronounceable Acronym", which is, for obvious reasons, abbreviated "INTERCAL".*
>
> *– INTERCAL-72 Manual*

> *If PLEASE was not encountered often enough, the program would be rejected; that is, ignored without explanation by the compiler. Too often and it would still be rejected, this time for sniveling.*

INTERCAL

1972

> *The full name of the compiler is "Compiler Language With No Pronounceable Acronym", which is, for obvious reasons, abbreviated "INTERCAL".*
>
> – *INTERCAL-72 Manual*

> *If PLEASE was not encountered often enough, the program would be rejected; that is, ignored without explanation by the compiler. Too often and it would still be rejected, this time for sniveling.*

Definition

Esoteric languages are designed not for a use case but to test boundaries of programming, be it as artistic expression or simply a joke.

Unix and the C language

1972

- 1969: Bell Labs drop out of the MULTICS project due to its size and complexity.

Unix and the C language

1972

- 1969: Bell Labs drop out of the MULTICS project due to its size and complexity.
  - They were writing it in PL/I!

Unix and the C language

1972

- 1969: Bell Labs drop out of the MULTICS project due to its size and complexity.
  - They were writing it in PL/I!
- 1969: Ken Thompson implements UNICS on a PDP/7 (4k of 18 bit words) in one month.
- 1972: Dennis Ritchie develops the C language to write utilities for Unix
  - Ultimately used to write Unix itself, making it the first *portable* OS

Unix and the C language

1972

- 1969: Bell Labs drop out of the MULTICS project due to its size and complexity.
  - They were writing it in PL/I!
- 1969: Ken Thompson implements UNICS on a PDP/7 (4k of 18 bit words) in one month.
- 1972: Dennis Ritchie develops the C language to write utilities for Unix
  - Ultimately used to write Unix itself, making it the first *portable* OS

Definition

System languages are designed for low-level *systems programming*.

PDP-11

1972

Ken Thompson and Dennis Ritchie with a PDP-11

Pong

1972

Pong cabinet

SEQUEL

1974

- Developped at IBM by Donald D. Chamberlin and Raymond F. Boyce
- Created to retrieve and process data stored in relationnal databases
- Later renamed to SQL

```
SELECT   AVG (SAL)
FROM     EMP
WHERE    DEPT = 'SHOE'
```

SEQUEL

1974

- Developped at IBM by Donald D. Chamberlin and Raymond F. Boyce
- Created to retrieve and process data stored in relationnal databases
- Later renamed to SQL

```
SELECT   AVG (SAL)
FROM     EMP
WHERE    DEPT = 'SHOE'
```

Definition

Domain-Specific Languages (DSL) are designed for use in a specific context. As such, they have no need to handle features outside of their specific domain.

Prolog

1975

Alain Colmerauer

Philippe Roussel

- Developped by Alain Colmerauer and Philippe Roussel
  - AI Group of the *Université d'Aix-Marseille*, for use in NLP
  - Short for *PROgrammation en LOGique*
  - Based on Horn clauses and first-order logic
- Competitor to Lisp for 80s AI research
- Turing-complete GPL

Prolog basics

- **Term**: objects/data of the program
  - variables: unknown object
  - elementary: int, string, identifiers
  - compound: structured objects
- **Atom**: relation between terms
- **Clause**:
  - facts: relations that are known to be true by the programmer
  - rules: Used to infer other facts
- **Goals**: Part of program where queries are made

Prolog examples

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
```

```
?- likes(john,food).
false.

?- likes(john,wine).
true.
```

Prolog examples

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
```

```
?- likes(john,food).
false.

?- likes(john,wine).
true.
```

```
man(alan).
man(gary).
woman(margaret).
parent(alan, gary).
parent(alan, margaret).
mother(X,Y) :- parent(X,Y), woman(Y).
father(X,Y) :- parent(X,Y), man(Y).
```

```
?- mother(alan,Mom).
Mom = margaret.
?- father(alan,Dad).
Dad = gary
```

Logic programming

Definition

In **logic programming**, programs are constructed by specifying a set of facts and rules and asking whether something is true.

Logic programming

Definition

In **logic programming**, programs are constructed by specifying a set of facts and rules and asking whether something is true.

Definition

In **declarative programming**, programs are constructed as a list of declarations without describing a control flow.

Logic programming

Definition

In **logic programming**, programs are constructed by specifying a set of facts and rules and asking whether something is true.

Definition

In **declarative programming**, programs are constructed as a list of declarations without describing a control flow.

Exercise

What are some other cases of declarative programming?

Section 4

Modern Era (1975 - 2025)

IBM PC 5100

1975

The first personnal computer

- 16K to 64K memory
- Support for BASIC and APL
- Tape drive for program storage

IBM PC 5100

1975

The first personnal computer

- 16K to 64K memory
- Support for BASIC and APL
- Tape drive for program storage

> *[...] a very professional package at a premium price. [...] a 50-lb package of interactive personal computing [...]*
>
> –
>
> *Byte Magazine*

IBM PC 5100

1975

The first personnal computer

- 16K to 64K memory
- Support for BASIC and APL
- Tape drive for program storage

> *[...] a very professional package at a premium price. [...] a 50-lb package of interactive personal computing [...]*
>
> –
>
> *Byte Magazine*

Introductory price: $8,975 to $19,975

IBM PC 5100

1975

The first personnal computer

- 16K to 64K memory
- Support for BASIC and APL
- Tape drive for program storage

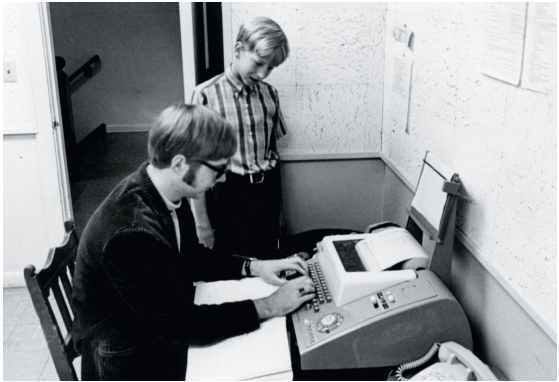
> *[...] a very professional package at a premium price. [...] a 50-lb package of interactive personal computing [...]*
>
> –
>
> *Byte Magazine*

Introductory price: $8,975 to $19,975
Adjusted to inflation: $49,000 to $109,000

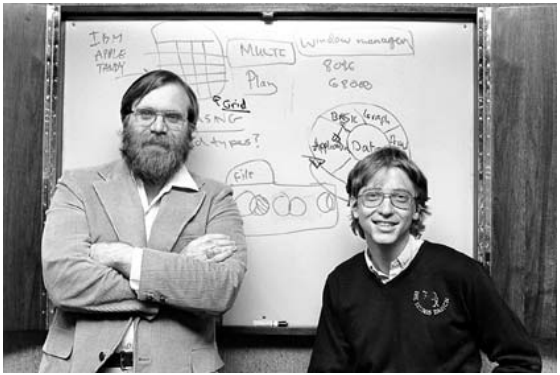
Microsoft

1975

Founded in 1975 by Bill Gates and Paul Allen.



At school in 1970



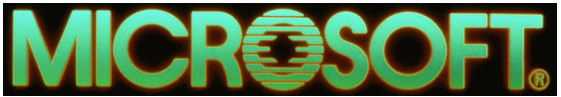
At work in 1983

Microsoft

Bill Gates in 1977

70s logo

Microsoft in 1978



80s logo

Apple II

1977

Original Apple II

- 4KiB to 48KiB RAM.
- $1,298 to $2,638 ($6,270 to $12,740 in 2022)

- First of the Apple II series.
  - Series discontinued only in 1993!
  - More than 6 million units sold.

AWK

1977

- Designed by Alfred Aho, Peter Weinberger and Brian Kernighan at Bell Labs
- Data and text processing DSL
- Maps actions to perform to patterns in an input stream

Definition

Scripting languages are used to write *scripts*, *i.e.* extend or automate the facilities of an existing system (OS, text editor, video game…).

AWK Example

```
BEGIN {
    FS="[^a-zA-Z]+"
}
{
    for (i=1; i<=NF; i++)
        words[tolower($i)]++
}
END {
    for (i in words)
        print i, words[i]
}
```

Exercise

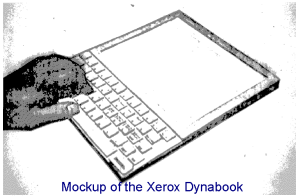
What does this do?

Smalltalk-80

1980

Alan Kay

> *We called Smalltalk Smalltalk so that nobody would expect anything from it.*

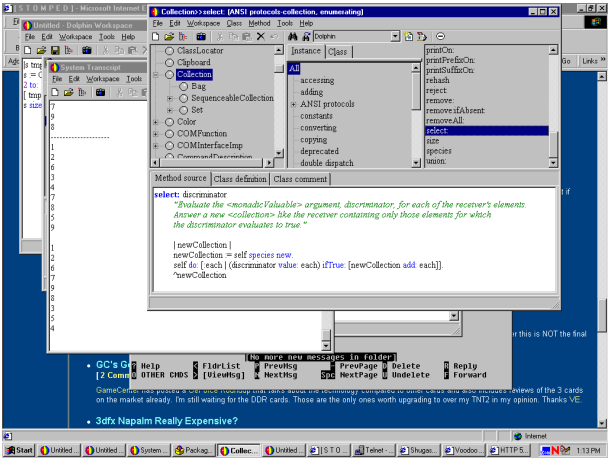


Dynabook concept (1972)

> *A computer for children of all ages.*

Smalltalk-80

1980



Aimed to provide a fully integrated language and environment for the Dynabook, including an IDE.

Smalltalk-80

1980

- Based on earlier Smalltalk 72 and Smalltalk 76, inspired from Simula 67.
- Principles:
  - Everything is object.
  - Every object is described by its class (structure, behavior).
  - Message passing is the only interface to objects.

Smalltalk-80

1980

- Based on earlier Smalltalk 72 and Smalltalk 76, inspired from Simula 67.
- Principles:
  - Everything is object.
  - Every object is described by its class (structure, behavior).
  - Message passing is the only interface to objects.

> *I invented the term Object-Oriented and I can tell you I did not have C++ in mind.*
>
> – *Alan Kay*

Smalltalk Example

```
| aString vowels |
aString := 'This is a string'.
vowels := aString select: [:aCharacter | aCharacter isVowel].
```

Smalltalk Example

```
| aString vowels |
aString := 'This is a string'.
vowels := aString select: [:aCharacter | aCharacter isVowel].


Collection>>select: aBlock
| newCollection |
newCollection := self species new.
self do: [:each |
    (aBlock value: each)
        ifTrue: [newCollection add: each]].
^newCollection
```

Smalltalk Example

```
| aString vowels |
aString := 'This is a string'.
vowels := aString select: [:aCharacter | aCharacter isVowel].


Collection>>select: aBlock
| newCollection |
newCollection := self species new.
self do: [:each |
    (aBlock value: each)
        ifTrue: [newCollection add: each]].
^newCollection
```

```
| rectangles aPoint|
rectangles := OrderedCollection
  with: (Rectangle left: 0 right: 10 top: 100 bottom: 200)
  with: (Rectangle left: 10 right: 10 top: 110 bottom: 210).
aPoint := Point x: 20 y: 20.
collisions := rectangles select: [:aRect | aRect containsPoint: aPoint].
```

Ada

1980

- Command from the US Department of Defense in the 70s for use on embedded systems.
- Descendant of Pascal, itself a descendant of ALGOL W.
- Standardized in 1983.
- Features:
  - Emphasis on software reliability.
  - *Very* strong typing.
  - Modular programming using packages.
  - Generic programming (one of the first languages with generics).
  - Rich control structures.
  - Various argument passing modes.
  - Interruptions, exceptions…

Jean Ichbiah

Ada Example

```ada
-- check.ads
generic
   Description : String;
   type T is private;
   with function
        Comparison (X, Y : T)
                   return Boolean;
procedure Check (X, Y : T);
```

```ada
-- check.adb
with Ada.Text_IO; use Ada.Text_IO;

procedure Check (X, Y : T) is
   Result : Boolean;
begin
   Result := Comparison (X, Y);
   if Result then
      Put_Line
        ("Comparison ("
         & Description
         & ") OK!");
   else
      Put_Line
        ("Comparison ("
         & Description
         & ") OK!");
   end if;
end Check;
```

Ada Example

```
with Check;

procedure Show_Formal_Subprogram is
   A, B : Integer;

   procedure Check_Is_Equal is new
     Check (Description => "equality",
            T           => Integer,
            Comparison  => Standard."=");
begin
   A := 0;
   B := 1;
   Check_Is_Equal (A, B);
end Show_Formal_Subprogram;
```

Ada Example

```
with Check;

procedure Show_Formal_Subprogram is
   A, B : Integer;

   procedure Check_Is_Equal is new
     Check (Description => "equality",
            T           => Integer,
            Comparison  => Standard."=");
begin
   A := 0;
   B := 1;
   Check_Is_Equal (A, B);
end Show_Formal_Subprogram;
```

Definition

Generic programming consists in writing algorithms and data structures that can be used with multiple types interchangeably. These generic functions and types are then *specialized* or *instantiated* for a given concrete type.

1981

Commodore VIC-20

First computer to sell over one million units.
4Kb RAM.
$299.95 ($970 in 2022).

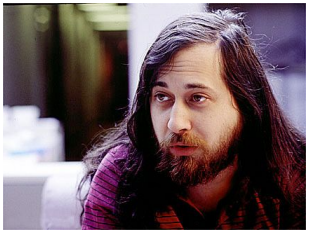
Osborne-1

First portable computer (only 11.1kg).
$1795 ($5780 in 2022).

The GNU project

1983

> *Free Unix!*
> *[...] I have decided to put together a sufficient body of free software so that I will be able to get along with any software that is not free. [...]*
>
> –
>
> *Richard Stallman's announcement of the GNU project in a Usenet message.*

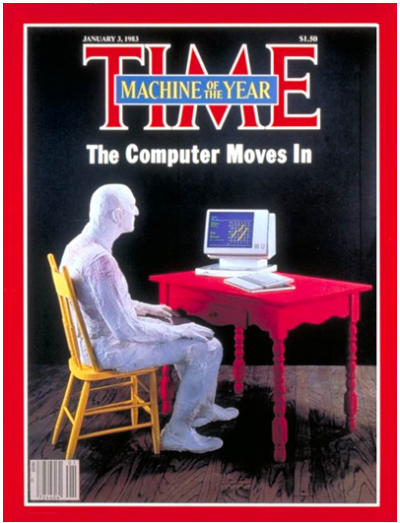


Richard Stallman

GNU's not Unix

Machine of the Year

1983



Macintosh

1984

Macintosh 128K

$2,945 (equivalent to $7,000 in 2022).

Macintosh

1984

Macintosh 128K

$2,945 (equivalent to $7,000 in 2022).

Ridley Scott's 1984 commercial.

Steve Jobs introduces Macintosh.

1984

Exercise

Let's summarize programming innovations up to 1984.

1984

Exercise

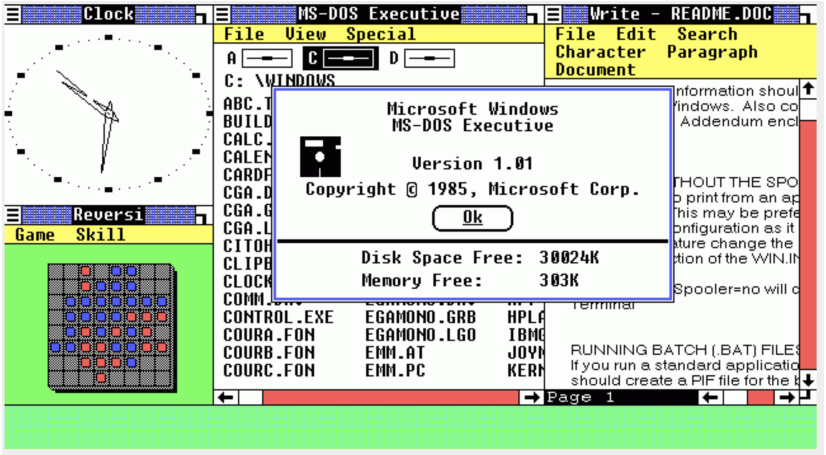
Let's summarize programming innovations up to 1984.

Exercise

What's changed since then?

Windows 1.0

1985



1990 and beyond

-  Haskell (1990)
-  Python (1990)
-  Java (1995)
-  JavaScript (1995)
-  OCaml (1996)

1990 and beyond

-  Haskell (1990)
-  Python (1990)
-  Java (1995)
-  JavaScript (1995)
-  OCaml (1996)
-  Tiger! (1998)

1990 and beyond

- Haskell (1990)
- Python (1990)
- Java (1995)
- JavaScript (1995)
- OCaml (1996)
- Tiger! (1998)
-  First GPU (1999)

1990 and beyond

- Haskell (1990)
- Python (1990)
- Java (1995)
- JavaScript (1995)
- OCaml (1996)
- Tiger! (1998)
- 
  First GPU (1999)
- C# (2001)
- Git (2005)
- Go (2009)

1990 and beyond

-  Haskell (1990)
-  Python (1990)
-  Java (1995)
-  JavaScript (1995)
-  OCaml (1996)
-  Tiger! (1998)
-  First GPU (1999)
-  C# (2001)
-  Git (2005)
-  Go (2009)
-  TypeScript (2012)
-  Swift (2014)
-  Rust (2015)

Section 5

Classifying Languages

Why?

Exercise

Why classify languages?

How?

Exercise

How to classify languages?

How?

Exercise

How to classify languages?

- By supported paradigms
- By properties
  - Type system
  - Memory management
  - Compilation model
  - Interoperability with other languages
- By purpose and use cases
  - GPL vs DSL
  - Scripting vs system vs esoteric vs ...
- By metrics
  - Speed of compilation, speed of execution
  - Efficiency
  - Popularity!

Two kinds of languages

> *There are only two kinds of languages: the ones people complain about and the ones nobody uses.*
>
> *– Bjarne Stroustrup*

Section 6

Epilogue

Half a century ago…

- 1972: Watergate scandal
- 1972: *The Godfather* releases in theaters
- 1972: Apollo 17 is the last moonlanding
- 1972: First *McDonald's* to open in France

Half a century ago…

- 1972: Watergate scandal
- 1972: *The Godfather* releases in theaters
- 1972: Apollo 17 is the last moonlanding
- 1972: First *McDonald's* to open in France
- 1972 also: Dennis Ritchie creates the C language.

Half a century ago…

- 1972: Watergate scandal
- 1972: *The Godfather* releases in theaters
- 1972: Apollo 17 is the last moonlanding
- 1972: First *McDonald's* to open in France
- 1972 also: Dennis Ritchie creates the C language.

> *The power of assembly language and the convenience of … assembly language.*
>
> *– Dennis Ritchie*

Far from perfect, yet *still* one the most used languages today.

53 years of progress!

```
main()
{
    printf("hello, world\n");
}
```

```
int main(void)
{
    printf("hello, world\n");
}
```

We can do better

	1972	2025
OS	UNIX v2 BETA	Ubuntu 24.04
Computer	PDP-11	Thinkpad L14 Gen 4
Memory	4MB RAM	64GB RAM
Storage	Punched cards	2TB SSD
System language	C	C

The year is 2025. Let's program like it's 2025.