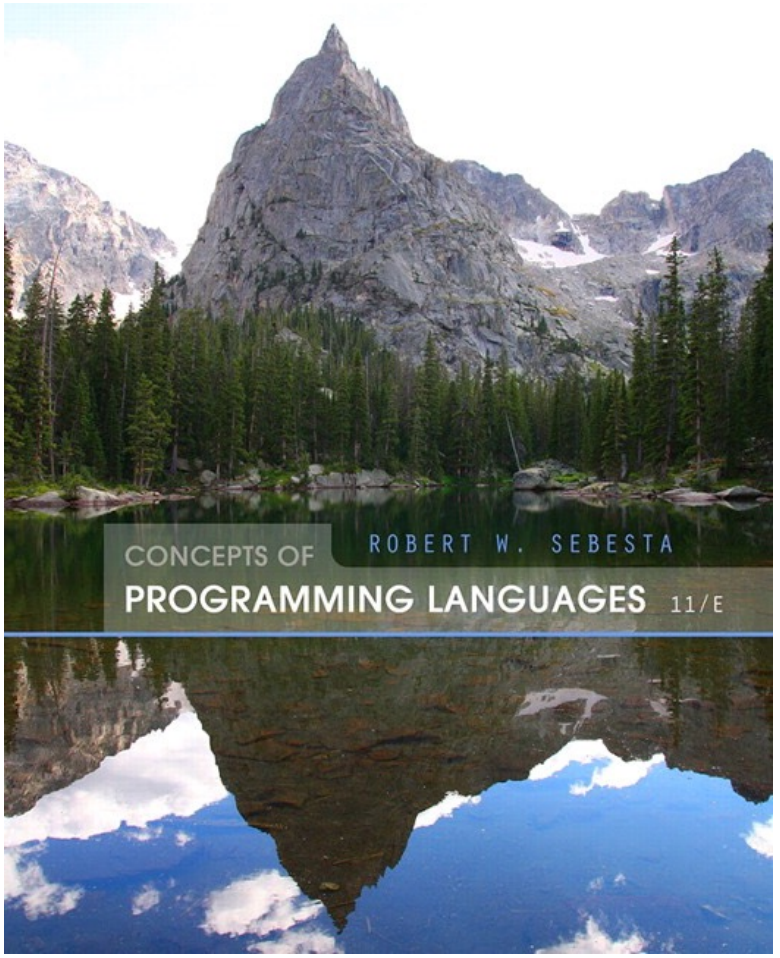
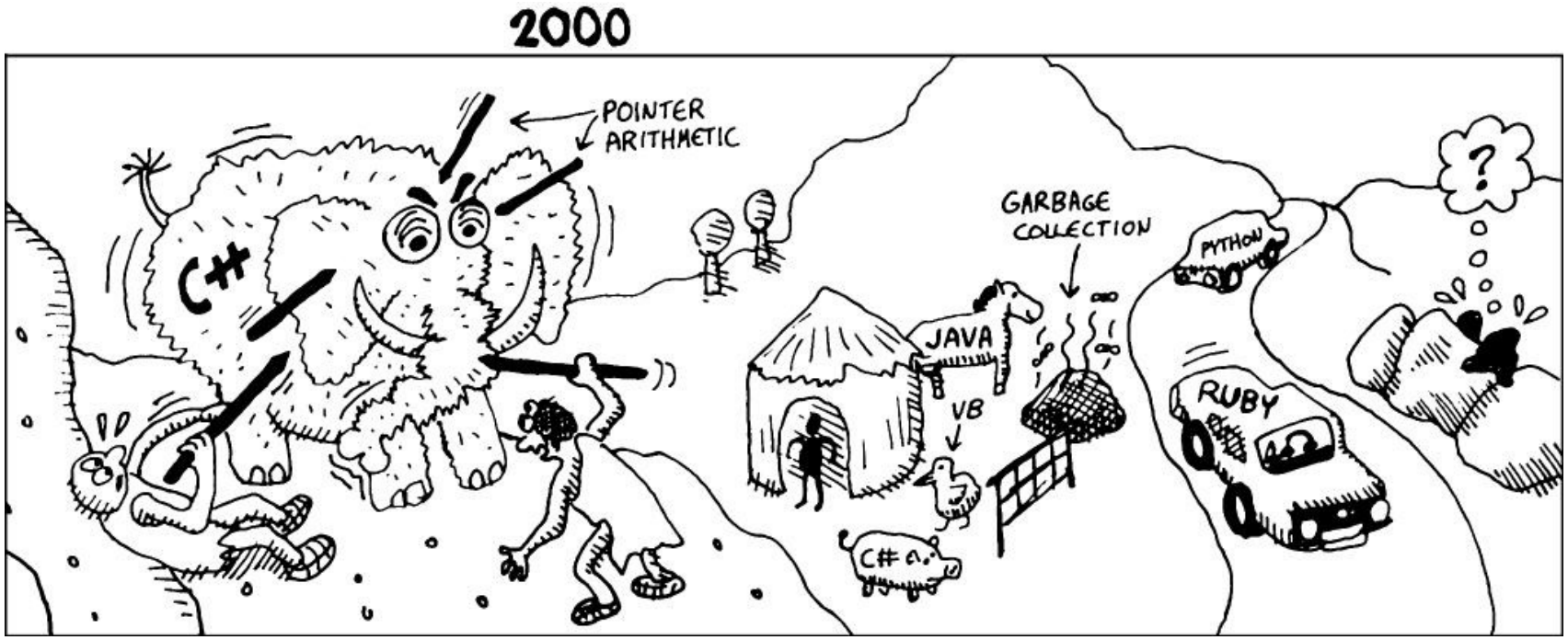
Chapter 2

Evolution of the Major Programming Languages (part 1)



ISBN 0-321-49362-1

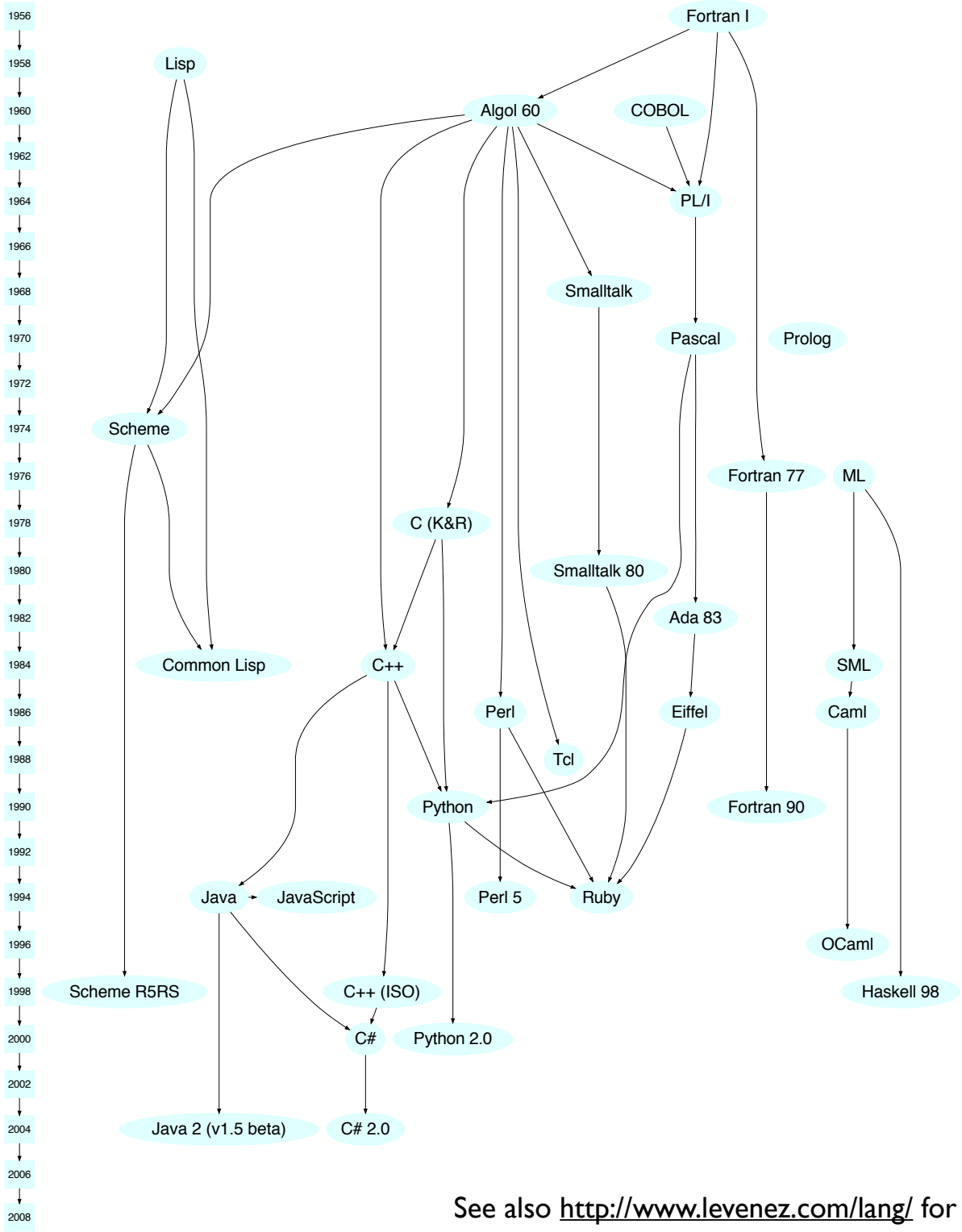


Source: https://onionesquereality.wordpress.com/tag/lisp/
referencing “Land of Lisp: Learn to Program in List, One Game at a time” by M. D. Conrad Barski.

Chapter 2 Topics

- Go through some history and major evolution of some Programming Languages
- We will show example code to get a sense
- But we will not yet delve into any one language

Genealogy of Common Languages
(light version)



See also http://www.levenez.com/lang/ for a complete list.

Zuse's Plankalkül

- Designed in 1945, but not published until 1972! Some features got reinvented around 15 years later…

Zuse's Plankalkül

- Designed in 1945, but not published until 1972! Some features got reinvented around 15 years later…
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Interesting feature: Mathematical expressions similar to assertions
- 49 pages of algorithms for playing chess!
- Wikipedia: kalkül = formal system; Plankalkül = formal system for planning

Plankalkül Syntax

- But funky/intimidating notation…

- An assignment statement to assign the expression A[4] + 1 to A[5]

	I	A + 1 => A		(statement)
V	I	4	5	(subscripts of array)
S	I	1.n	1.n	(data types)

integer n bits

Minimal Hardware Programming: Pseudocodes

- 1940's and early 50s
- Not same as contemporary meaning,
- There was no high level machine language, or even assembly, so programming done in machine code
- What was wrong with using machine code?

Minimal Hardware Programming: Pseudocodes

- 1940's and early 50s
- Not same as contemporary meaning,
- There was no high level machine language, or even assembly, so programming done in machine code
- What was wrong with using machine code?
  - tedious to modify and error prone; absolute addressing
  - hard to program
  - readability
  - Machine deficiencies--no indexing or floating point

Pseudocodes: Short Code

Coded versions of mathematical expressions to be evaluated:

01	-		06	abs		1n	(n+2)nd power
02	)		07	+		2n	(n+2)nd root
03	=		08	pause		4n	if >= n
04	/		09	(		58	print and tab

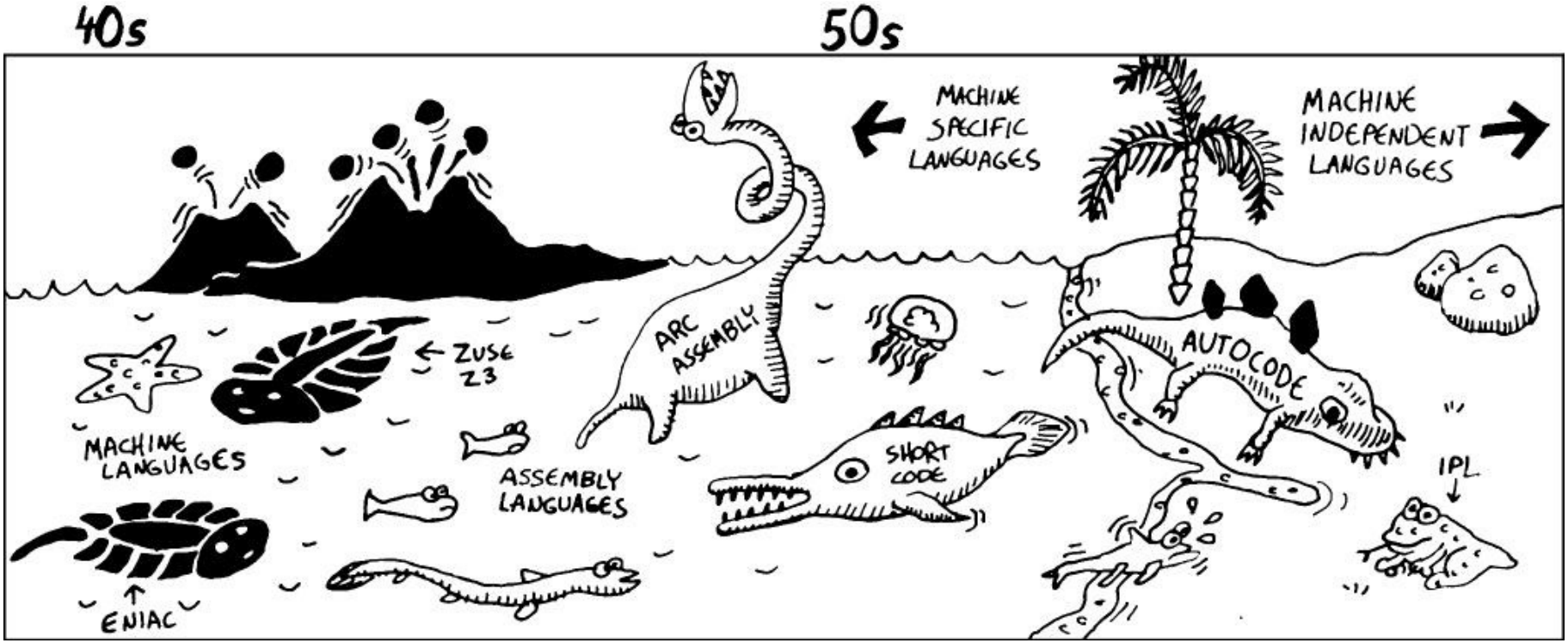
Pseudocodes: Short Code

Example: X0=SQRT(ABS(Y0))

coded as: 00 X0 03 20 09 06 09 Y0 02 02
(with 00 padding)

01	-		06	abs		1n	(n+2)nd power
02	)		07	+		2n	(n+2)nd root
03	=		08	pause		4n	if >= n
04	/		09	(		58	print and tab

Pseudocodes: Short Code



Source: https://onionesquereality.wordpress.com/tag/lisp/
referencing “Land of Lisp: Learn to Program in List, One Game at a time” by M. D. Conrad Barski.

Pseudocodes: Short Code

- UNIVAC I Computer: first commercial computer sold in US, used short code

http://www.computerhistory.org/timeline/1951/

Pseudocodes: Short Code

- UNIVAC I Computer: first commercial computer sold in US, used short code

Univac (Universal Automatic Computer)
Used for??

http://www.computerhistory.org/timeline/1951/

Pseudocodes: Short Code

- UNIVAC I Computer: first commercial computer sold in US, used short code

https://www.thocp.net/hardware/univac.htm#1

- “Univac (Universal Automatic Computer) computers were used in many different applications but utilities, insurance companies and the US military were major customers…”

- Apparently used in 1952 presidential elections to predict Eisenhower would win over Stevens on the evening of the election: “CBS withheld its predictions from the air, but as the night went on, Walter Cronkite announced UNIVAC was right and Eisenhower had won.”

- “One biblical scholar even used a Univac 1 to compile a concordance to the King James version of the Bible"

http://www.computerhistory.org/timeline/1951/

Pseudocodes: Speedcoding

- Speedcoding developed by Backus in 1954 for IBM 701
  - Included floating point operations
  - Slow! (add instruction: 4.2 millisec to execute)
  - Only 700 words left for user program after loading interpreter!

IBM 704 and Fortran

One of the greatest single advances in computing …

Source: Columbia University

IBM 704 and Fortran

- Fortran 0: 1954 - not implemented
  Report: "The IBM Mathematical FORmula TRANslating System: FORTRAN"

- Fortran I: 1957
  - Designed for the new IBM 704
  - Considered one of the greatest single advances in computing
  - Prompted development of Fortran high-level language

Source: Columbia University

IBM 704 and Fortran

- Fortran I:1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
  - This led to the idea of compiled programming languages, (primary reason for interpreters being tolerated before was the lack of floating point ops in hardware. All had to be done in software)

Source: Columbia University

IBM 704 and Fortran

- Fortran I:1957
  - Environment of development
    - Computers were unreliable
    - Applications were scientific
    - No programming methodology or tools
    - Machine efficiency was the most important concern

Source: Columbia University

Fortran I Overview

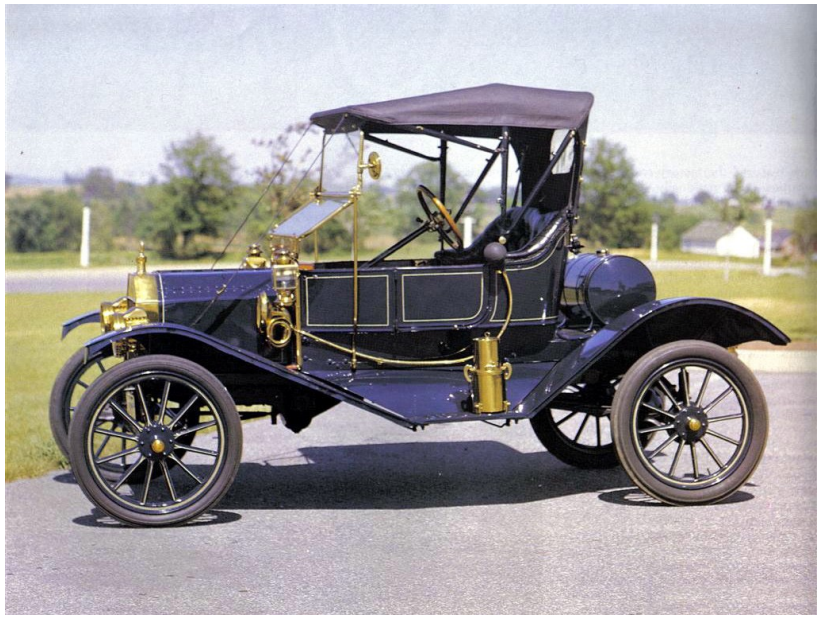
- First implemented version of Fortran
  - Names could have up to six characters (2 in version 0)
  - Post-test counting loop (`DO`)
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic `IF`)
  - No data typing statements (names beginning I,J,K,L,M,N implicitly integer; others float…)

Fortran I Overview (continued)

- First implemented version of FORTRAN
  - Compiler released in April 1957, after 18 worker-years of effort!
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  - Code was very fast
  - Quickly became widely used

Fortran I Overview (continued)

- "…early versions of Fortran suffer in a variety of ways, as would be expected … afterall, it would not be fair to compare the performance or comfort of a 1910 model T Ford to a 2015 Ford Mustang"; Sebesta book

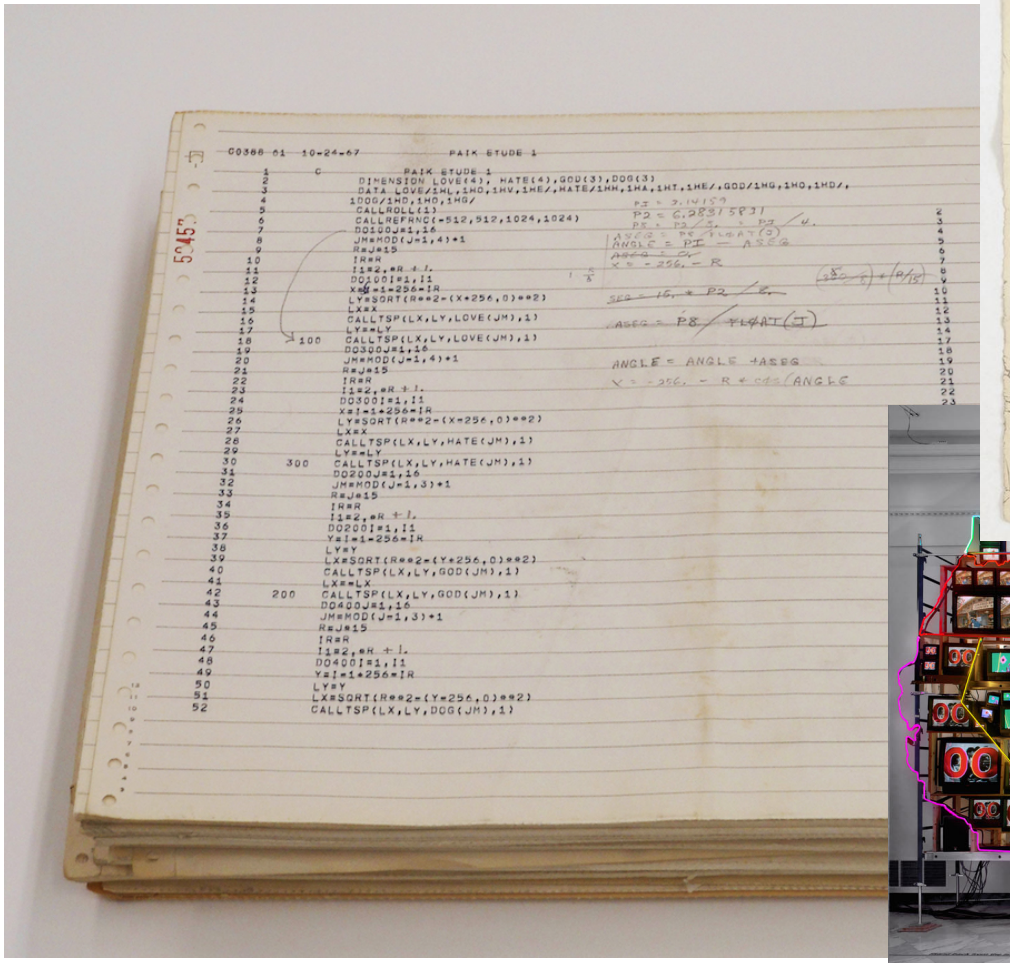


Fortran IV (1960-62)

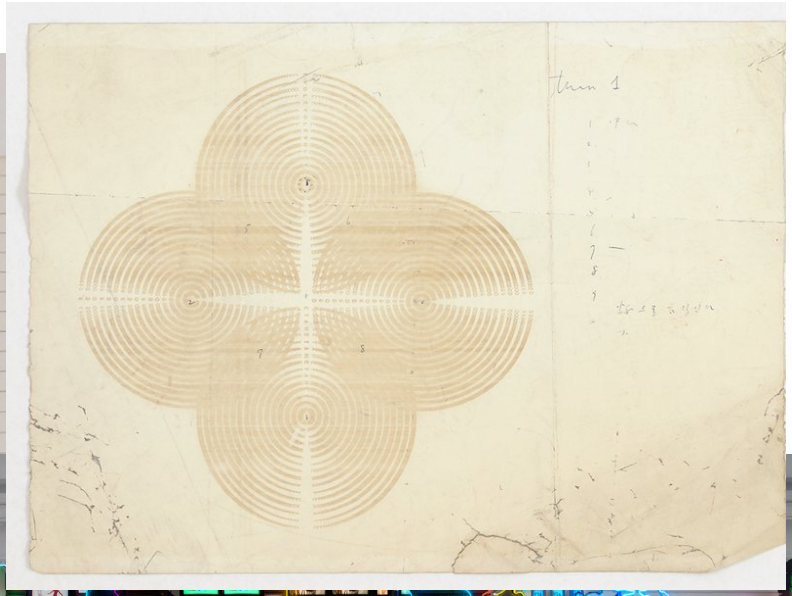
- Evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters
  - ANSI standard in 1966

Fortran and art (1967)

- Nam Jun Paik (video artist) uses Fortran (1967); Smithsonian Watch This! Revelations in media art 2015



```
C0388-01  10-24-67              PAIK ETUDE 1
      1         C               PAIK ETUDE 1
      2               DIMENSION LOVE(4), HATE(4),GOD(3),DOG(3)
      3               DATA LOVE/1HL,1HO,1HV,1HE/,HATE/1HH,1HA,1HT,1HE/,GOD/1HG,1HO,1HD/,
      4              1DOG/1HD,1HO,1HG/
      5               CALL PLOT(1)
      6               CALL REFRNC(-512,912,1024,1024)
      7               DO 100 J=1,16
      8               JM=MOD(J-1,4)+1
      9               R=J*15
     10               I=R
     11               I1=2.*R+1.
     12               DO 100 I=1,I1
     13               X=I*1-256-IR
     14               LY=SQRT(R**2-(X+256.0)**2)
     15               LX=X
     16               CALL TSP(LX,LY,LOVE(JM),1)
     17               LY=-LY
     18   100         CALL TSP(LX,LY,LOVE(JM),1)
     19               DO 300 J=1,16
     20               JM=MOD(J-1,4)+1
     21               R=J*15
     22               I=R
     23               I1=2.*R+1.
     24               DO 300 I=1,I1
     25               X=I*1-256-IR
     26               LY=SQRT(R**2-(X-256.0)**2)
     27               LX=X
     28               CALL TSP(LX,LY,HATE(JM),1)
     29               LY=-LY
     30   300         CALL TSP(LX,LY,HATE(JM),1)
     31               DO 200 J=1,16
     32               JM=MOD(J-1,3)+1
     33               R=J*15
     34               I=R
     35               I1=2.*R+1.
     36               DO 200 I=1,I1
     37               Y=I*1-256-IR
     38               LY=Y
     39               LX=SQRT(R**2-(Y+256.0)**2)
     40               CALL TSP(LX,LY,GOD(JM),1)
     41               LX=-LX
     42   200         CALL TSP(LX,LY,GOD(JM),1)
     43               DO 400 J=1,16
     44               JM=MOD(J-1,3)+1
     45               R=J*15
     46               I=R
     47               I1=2.*R+1.
     48               DO 400 I=1,I1
     49               Y=I*1-256-IR
     50               LY=Y
     51               LX=SQRT(R**2-(Y-256.0)**2)
     52               CALL TSP(LX,LY,DOG(JM),1)
```



Fortran 77

- Became the new standard in 1978
  - Character string handling
  - Logical loop control statement
  - **`IF-THEN-ELSE`** statement

Fortran 90

- Changed significantly from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - **`CASE`** statement
  - Parameter type checking

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003 - supports OOP
- Summary: Fortran accredited as first high-level language; has evolved with modern features and still used today

Fortran Evaluation

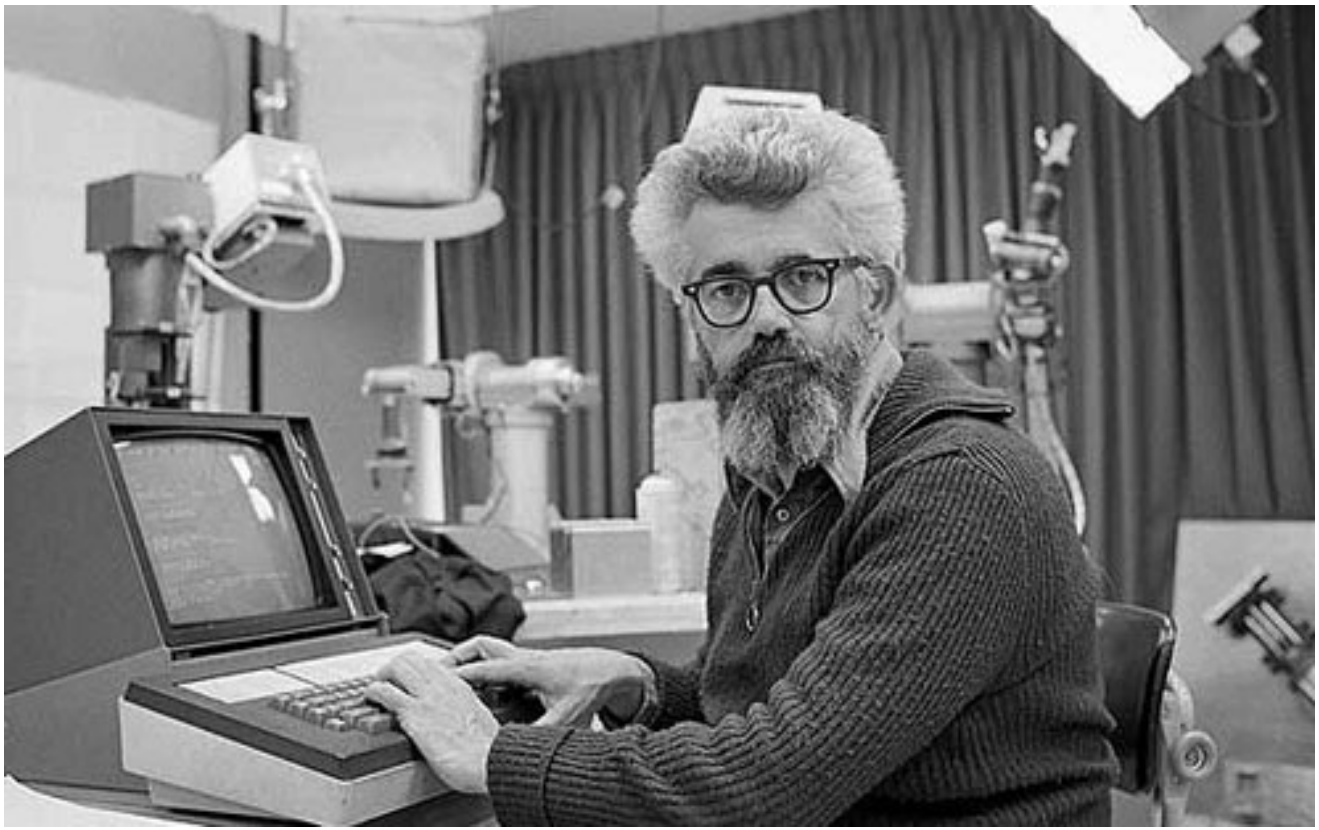
- Highly optimizing compilers
  - Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used
- Characterized as the lingua franca of the computing world

Fortran example code

```
! Fortran 95 Example program
!  Input:  An integer, List_Len, where List_Len is less
!          than 100, followed by List_Len-Integer values
!  Output: The number of input values that are greater
!          than the average of all input values
Implicit none
Integer Dimension(99) :: Int_List
Integer :: List_Len, Counter, Sum, Average, Result
Result= 0
Sum = 0
Read *, List_Len
If ((List_Len > 0) .AND. (List_Len < 100)) Then
! Read input data into an array and compute its sum
   Do Counter = 1, List_Len
      Read *, Int_List(Counter)
      Sum = Sum + Int_List(Counter)
                End Do
             ! Compute the average
                Average = Sum / List_Len
             ! Count the values that are greater than the average
                Do Counter = 1, List_Len
                   If (Int_List(Counter) > Average) Then
                      Result = Result + 1
                   End If
                End Do
             ! Print the result
                Print *, 'Number of values > Average is:', Result
             Else
                Print *, 'Error - list length value is not legal'
             End If
             End Program Example
```

Functional Programming: LISP

- LISP Processing language
  - Designed at MIT by McCarthy



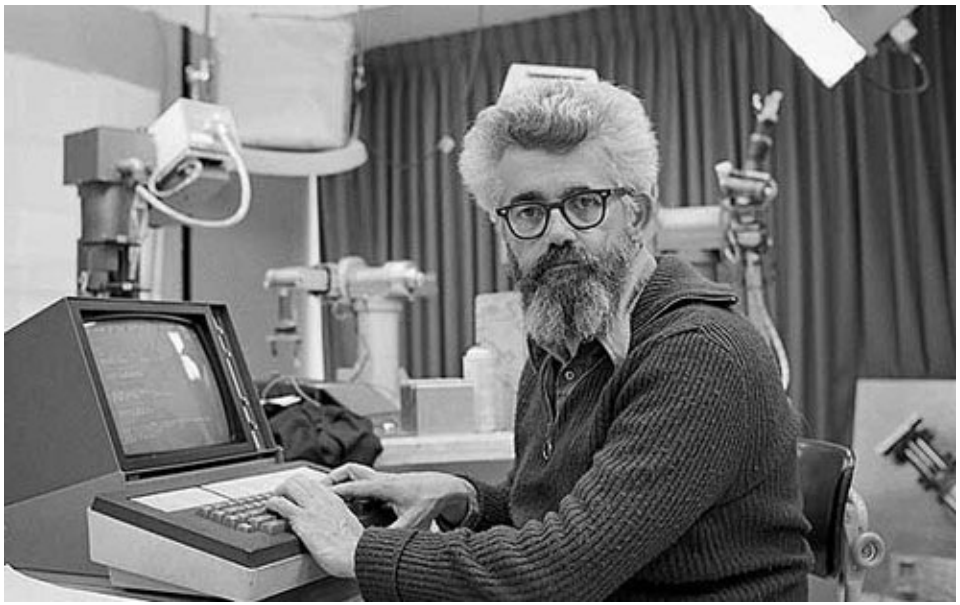
http://www-formal.stanford.edu/jmc/history/lisp/lisp.html

Functional Programming: LISP

- AI research needed a language
  - Interest in AI in mid 1950s
  - Interest from linguists and natural language processing, psychologists and the brain, mathematicians and mechanizing intelligent processes

Functional Programming: LISP

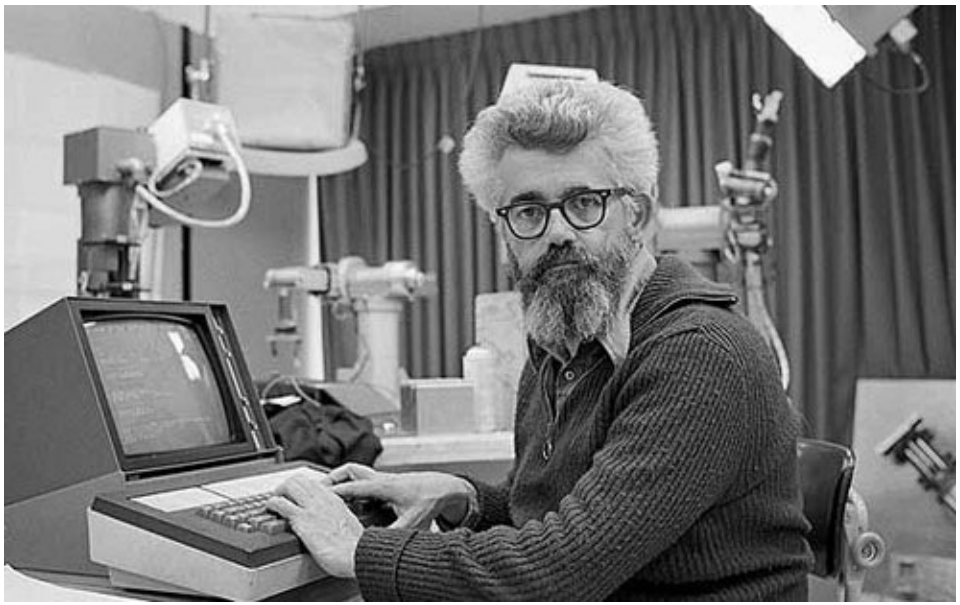
"My desire for an algebraic list processing language for artificial intelligence work on the IBM 704 computer arose in the summer of 1956 during the Dartmouth Summer Research Project on Artificial Intelligence which was the first organized study of AI …
There were two motivations for developing a language for the IBM 704. First, IBM was generously establishing a New England Computation Center at M.I.T. which Dartmouth would use. Second, IBM was undertaking to develop a program for proving theorems in plane geometry (based on an idea of Marvin Minsky's), and I was to serve as a consultant to that project…"



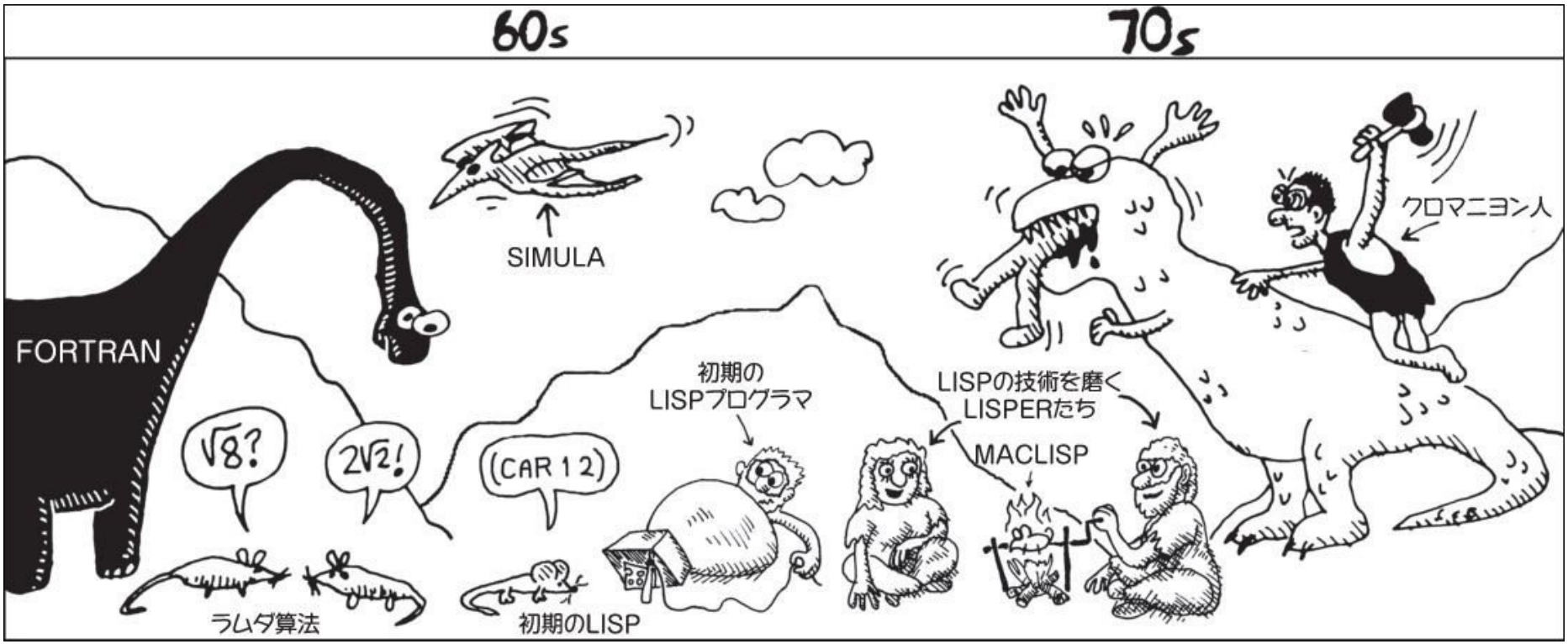
http://www-formal.stanford.edu/jmc/history/lisp/lisp.html

Functional Programming: LISP

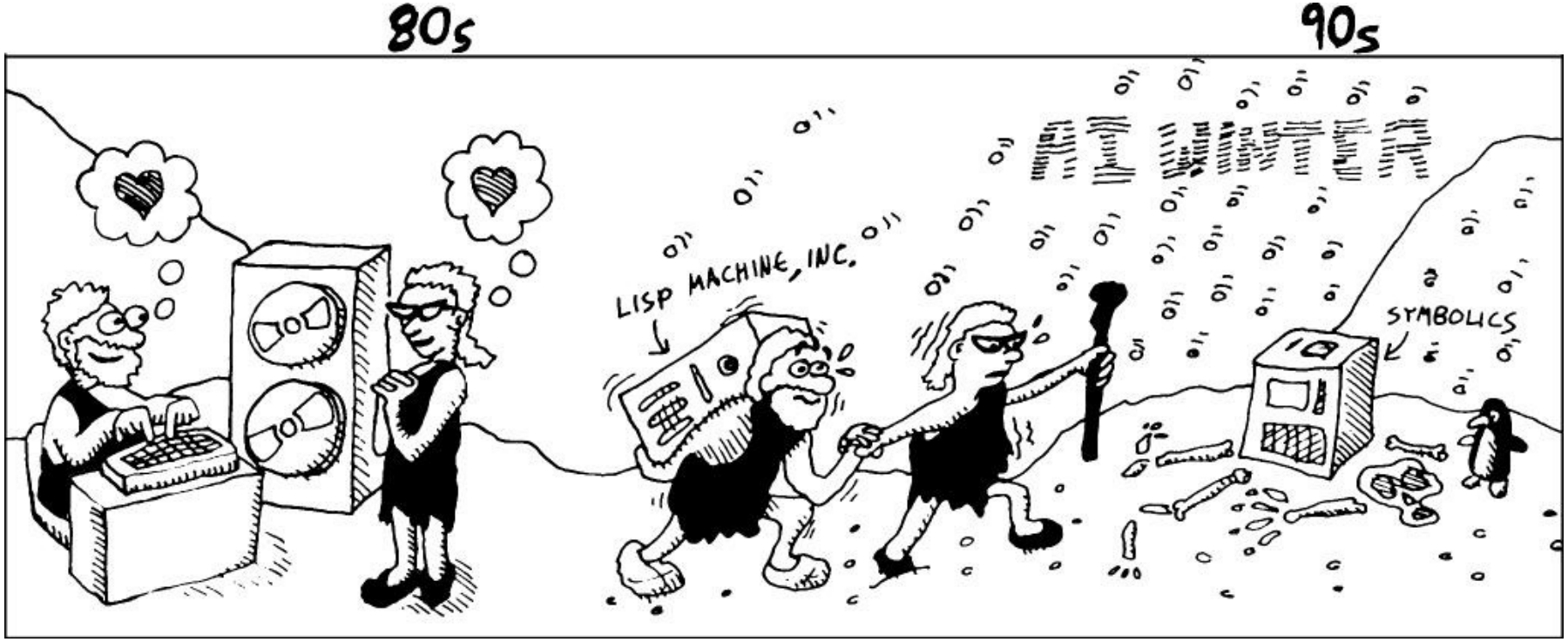
"my own research in artificial intelligence was proceeding along the lines that led to the Advice Taker proposal in 1958 (McCarthy 1959). This involved representing information about the world by sentences in a suitable formal language and a reasoning program that would decide what to do by making logical inferences. Representing sentences by list structure seemed appropriate - it still is - and a list processing language also seemed appropriate for programming the operations involved in deduction - and still is."



http://www-formal.stanford.edu/jmc/history/lisp/lisp.html



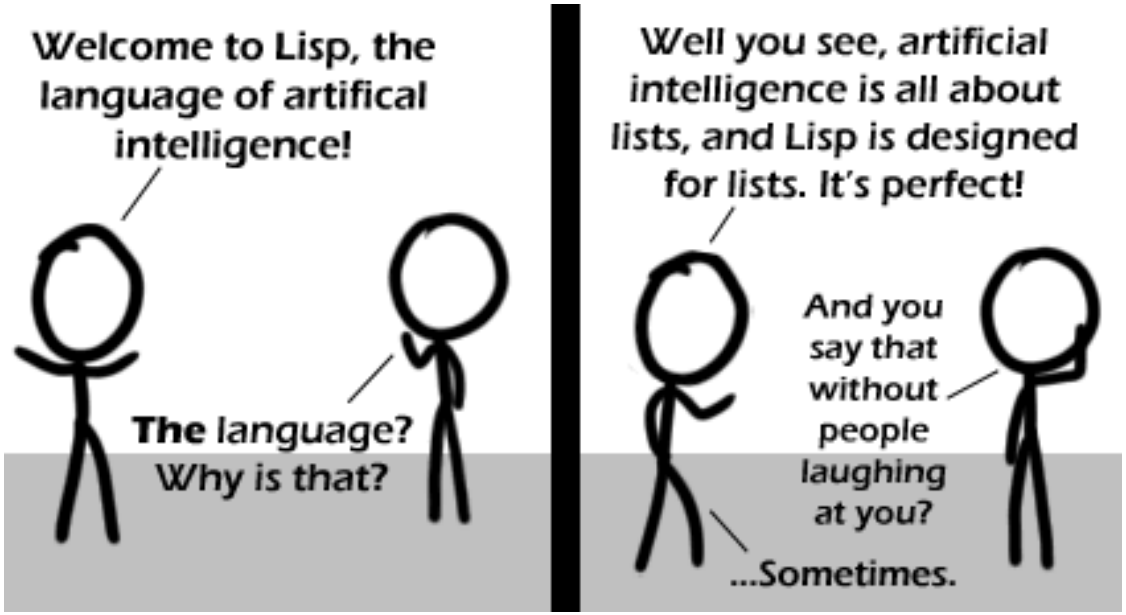
Source: https://onionesquereality.wordpress.com/tag/lisp/
referencing “Land of Lisp: Learn to Program in List, One Game at a time” by M. D. Conrad Barski.



Source: https://onionesquereality.wordpress.com/tag/lisp/
referencing “Land of Lisp: Learn to Program in List, One Game at a time” by M. D. Conrad Barski.

Functional Programming: LISP

- AI research needed a language to
  - Process data in lists (rather than arrays). Why lists?



Functional Programming: LISP

- AI research needed a language to
  - Process data in lists (rather than arrays). Why lists?

    More flexible than arrays, allows more heterogeneous data, hierarchies, decision making on trees, recursion

Functional Programming: LISP

- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)

  What is symbolic? Example lists?

Functional Programming: LISP

- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)

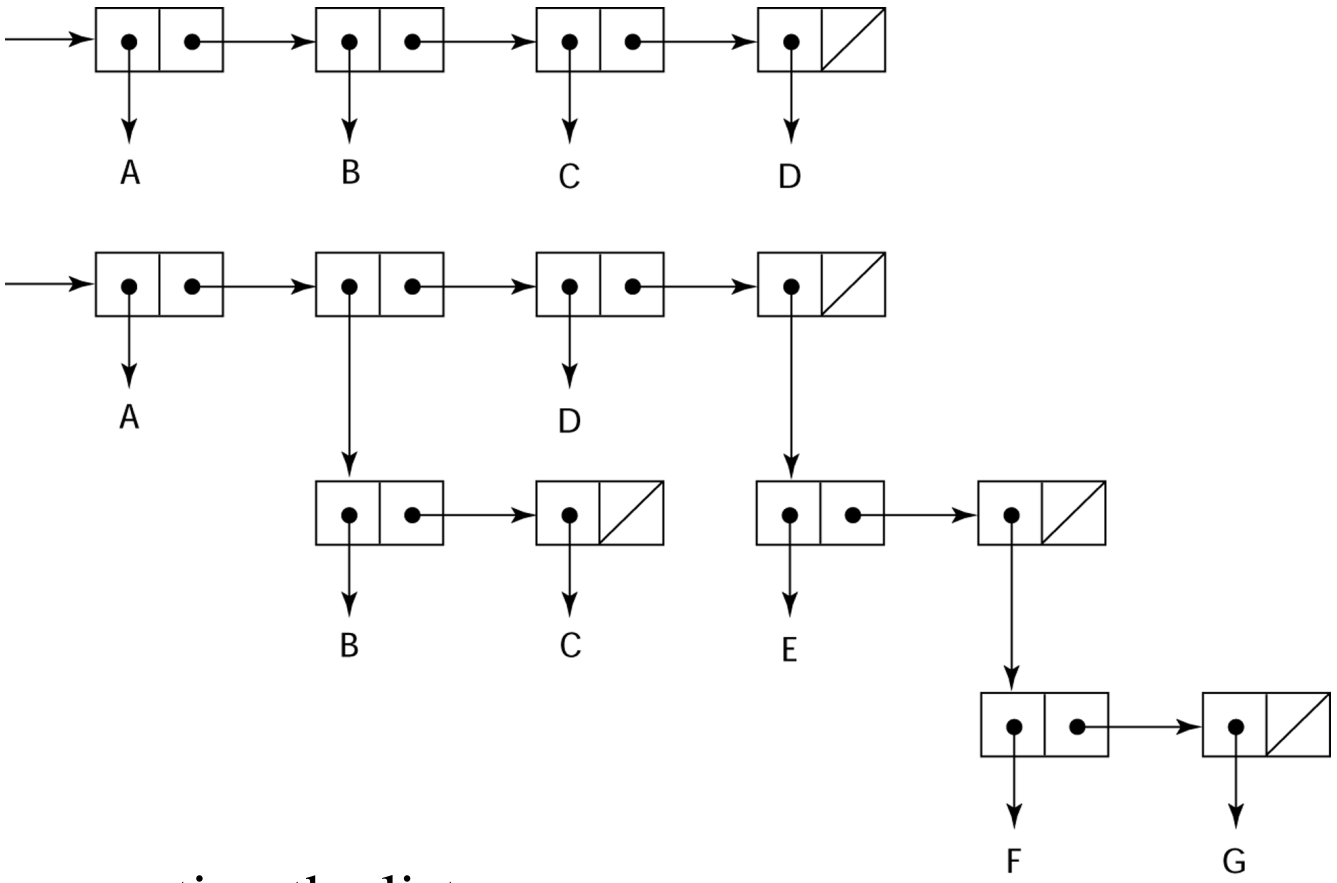
  Symbolic as in symbols, e.g., relationship between items in a list - more abstract (colors, places, people, hierarchical parts of objects, things you remember, etc.); can have semantic value

Functional Programming: LISP

- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)

- Only two data types: atoms and lists

Representation of Two LISP Lists



Representing the lists `(A B C D)` and `(A (B C) D (E (F G)))`

Functional Programming: LISP

Very different from Imperative Languages

- All computations by applying functions to arguments

Functional Programming: LISP

Very different from Imperative Languages

- All computations by applying functions to arguments
- assignment statements and variables not necessary

Functional Programming: LISP

Very different from Imperative Languages

- All computations by applying functions to arguments
- assignment statements and variables not necessary
- No loops and all done with recursion

Functional Programming: LISP

Very different from Imperative Languages

- All computations by applying functions to arguments
- assignment statements and variables not necessary
- No loops and all done with recursion
- Syntax very different (compare C to LISP and its simplicity)

Functional Programming: LISP

Very different from Imperative Languages

- All computations by applying functions to arguments
- assignment statements and variables not necessary
- No loops and all done with recursion
- Syntax very different (compare C to LISP and its simplicity)
- Formally, based on lambda calculus, introduced to investigate function definition, application, and recursion in 1930s

LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Original LISP is “Pure Lisp” because purely functional
- Dominated AI for 25 years and still dominant [though note today: renewed interest in machine learning/AI, and other imperative languages such as Python]

LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Original LISP is “Pure Lisp” because purely functional
- Dominated AI for 25 years and still dominant [though note today: renewed interest in machine learning/AI, and other imperative languages such as Python]
- COMMON LISP (which has added imperative language capabilities) and Scheme are contemporary dialects of LISP (more later)
- Related languages today?

LISP Evaluation

- Dominated AI for 25 years and still dominant [though note today: renewed interest in machine learning/AI, and other imperative languages such as Python]
- COMMON LISP (which has added imperative language capabilities) and Scheme are contemporary dialects of LISP (more later)
- ML, Miranda, and Haskell are related languages
- ML some imperative and doesn't use parenthesis; Haskell, lazy evaluation.

LISP Example Program

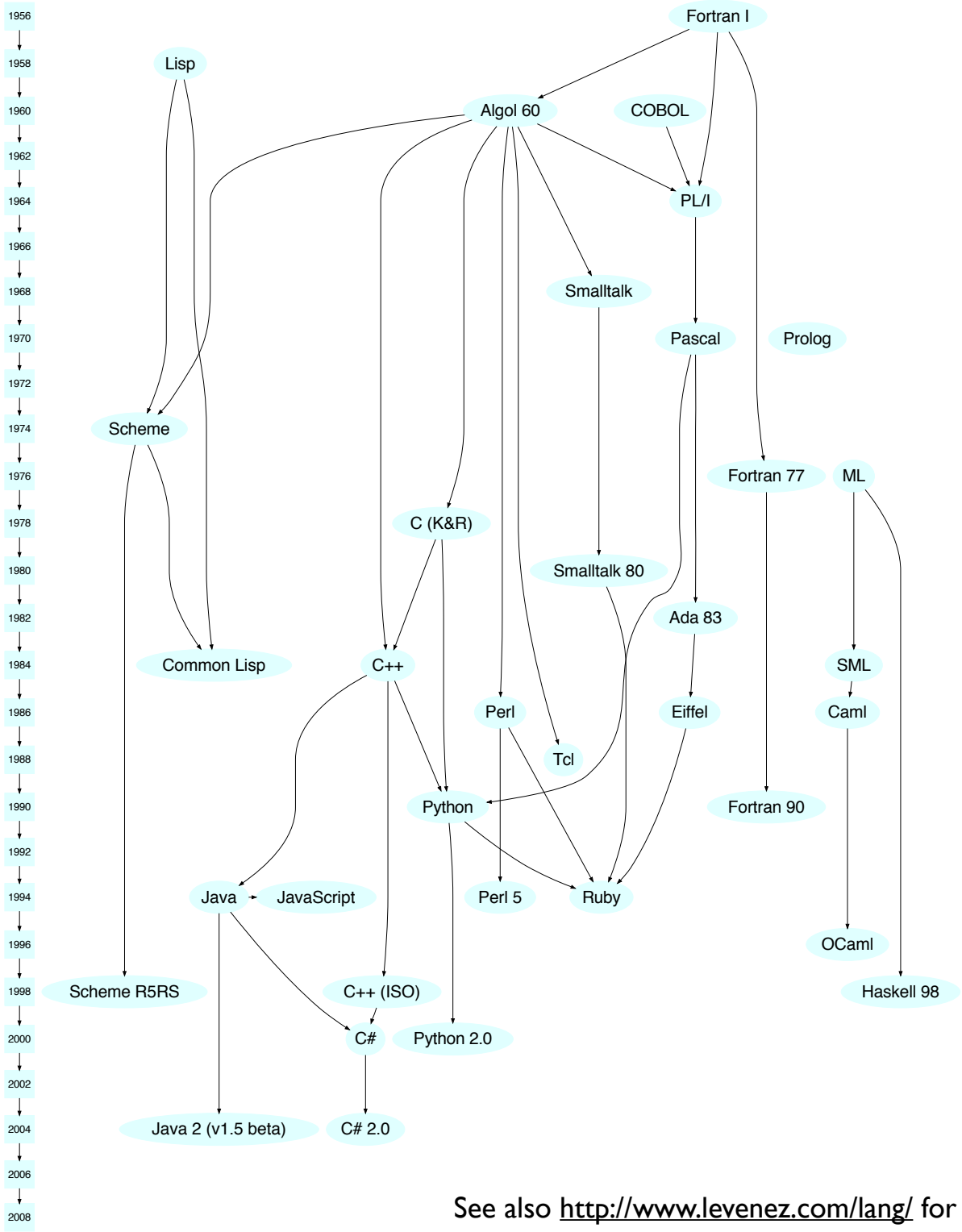
```
;  LISP Example function
;  The following code defines a LISP predicate function
;  that  takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (false) otherwise
   (DEFUN equal_lists (lis1 lis2)
    (COND
      ((ATOM lis1) (EQ lis1 lis2))
      ((ATOM lis2) NIL)
      ((equal_lists (CAR lis1) (CAR lis2))
            (equal_lists (CDR lis1) (CDR lis2)))
(T NIL) )

)
```

CAR, CDR?

Where is the recursion?

Genealogy of Common Languages
(light version)



See also http://www.levenez.com/lang/ for a complete list.