COS 301

Programming Languages

Evolution of the Major Programming Languages

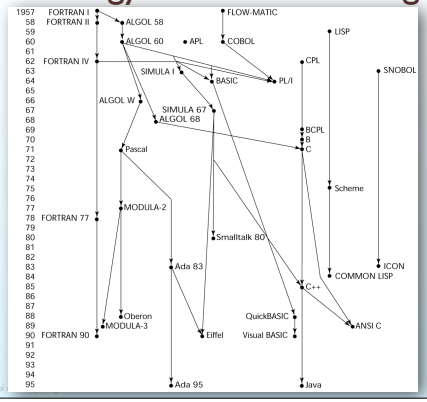
Topics

- Zuse's Plankalkül
- Minimal Hardware Programming: Pseudocodes
- The IBM 704 and Fortran
- Functional Programming: LISP
- ALGOL 60
- COBOL
- BASIC
- PL/I
- APL and SNOBOL
- SIMULA 67
- Orthogonal Design: ALGOL 68

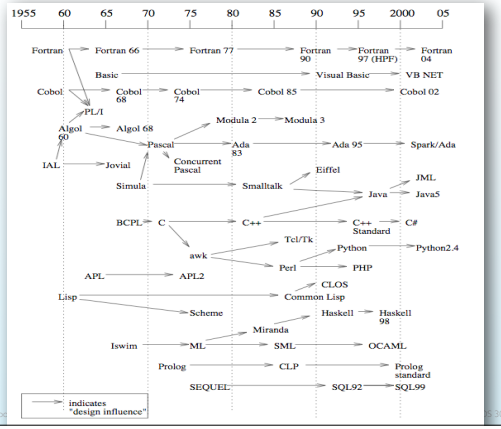
Topics (continued)

- Some Early Descendants of the ALGOLs
- Prolog
- Ada
- Object-Oriented Programming: Smalltalk
- Combining Imperative and Object-Oriented Features: C++
- Imperative-Based Object-Oriented Language: Java
- Scripting Languages
- A C-Based Language for the New Millennium: C#
- Markup/Programming Hybrid Languages

Genealogy of Common Languages



Alternate View



Zuse's Plankalkül

- Designed in 1945
- For computers based on electromechanical relays
- Not published until 1972, implemented in 2000 [Rojas *et al.*]
- Advanced data structures:
  - Two's complement integers, floating point with hidden bit, arrays, records
  - Basic data type: arrays, tuples of arrays
- Included algorithms for playing chess
- Odd: 2D language
- Functions, but no recursion
- Loops ("while") and guarded conditionals [Dijkstra, 1975]

Plankalkül Syntax

- 3 lines for a statement:
  - Operation
  - Subscripts
  - Types
- An assignment statement to assign the expression `A[4] + 1` to `A[5]`

```
     |   A + 1 => A
V    |   4       5        (subscripts)
S    |   1.n     1.n      (data types)
```


Minimal Hardware Programming: Pseudocodes

- Late 1940's – early 1950's all programming was done in machine code (*not* assembler)
- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies – no indexing or floating point
  - Absolute addressing

Pseudocodes: Short Code

- Short Code (orig.: Brief Code) developed by John Mauchly in 1949 for BINAC computers, then UNIVAC
  - Expressions coded left to right in 12 6-bit bytes
  - Example of operations:

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

- First "HLL"
- Short codes were interpreted, not translated to machine code
- So X0 = sqrt(abs(Y0)) would be

```
X0 03 20 06 Y0
```

- Interpreted, slow

Pseudocodes: Speedcoding

- Speedcoding developed by John Backus in 1954 for the IBM 701
- Turned it into a virtual 3-address calculator
  - Pseudo ops for arithmetic and math functions
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - OP1 A B C OP2
  - OP1: arithmetic or I/O; OP2: logical ops on instruction counter; A, B, C: addresses
- Example:

  523 SUBAB **100 200 300** TRPL **500**
  - 523 - address of instruction
  - SUBAB: subtract [200] from [100] → 300
  - Test [300]: if positive → 500

Pseudocodes: Speedcoding

- Format: **`OP1 A B C OP2 D`**
  - OP1: arithmetic or I/O; OP2: logical ops on instruction counter; A, B, C, D: addresses
- Example:

  `523 SUBAB 100 200 300 TRPL 500`
  - 523 - address of instruction
  - SUBAB: subtract [200] from [100] → 300
  - Test [300]: if positive → 500
- Slower than 701 machine language — but faster for programmer: weeks → hours
- But: Only 700 words left for user program!

Pseudocodes: Related Systems

- The UNIVAC Compiling System
  - Developed by a team led by Grace Hopper
  - Pseudocode expanded into machine code
- David J. Wheeler (Cambridge University)
  - developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing

Pros/Cons

- As a group: Take three minutes and list pros and cons of the things we've talked about with respect to:
  - Machine or assembly language
  - HLLs
- Take into account
  - purposes of programs at the time
  - limitations of the machines at the time

IBM 704 and Fortran

- FORTRAN: IBM Mathematical FORmula TRANslating System
- Computing environment at that time:
  - Machines: small memories, slow and unreliable
  - Mostly for scientific computation (number-crunching)
  - No programming tools
  - Overhead of interpretive systems was small compared to simulating floating point ops in software
- Fortran 0: 1954 - not implemented
- Fortran I:1957
  - Designed for the new IBM 704 — index registers, floating point hardware
  - No longer need to do FP in software ⇒ nowhere to "hide" cost of interpretation

Design Issues

- Primarily to do math
- Need good array handling, counting loops
- No need for string handling, decimal arithmetic, powerful I/O
- Maximize speed
- No need for dynamic storage was seen (if even thought about)

Fortran I Overview

- First implemented version
- Names: up to six characters
- Post-test counting loop (`DO`)
- Formatted I/O (simple)
- User-defined subprograms
- Three-way selection statement (arithmetic `IF`):

```
IF (A - B) 60,70,80
```

- Control structures based on 704 machine codes
- Implicit typing:
  - Names beginning with "I" to "N": integers
  - Others: floating point

Fortran I Overview (cont'd)

- No separate compilation
- Compiler released April 1957 – 18 worker-years
- Reliability:
  - main problem: 704 was unreliable
  - ⇒ programs > 400 lines rarely compiled
- Code very fast
- Quickly became widely used

FORTRAN II

- Distributed in 1958
- Independent compilation
- Fixed the bugs in FORTRAN I

FORTRAN II

```
C
C FORTRAN-II VERSION OF 99 BOTTLES OF BEER
C DAVE PITTS, DPITTS AT COZX.COM
C
      DO 30 J = 1, 98
      I = 100 - J
      WRITE OUTPUT TAPE 6, 100, I, I
      WRITE OUTPUT TAPE 6, 110
      I = I - 1
      IF (I - 1) 10, 10, 20
10    WRITE OUTPUT TAPE 6, 125, I
      GO TO 30
20    WRITE OUTPUT TAPE 6, 120, I
30    CONTINUE
      I = 1
      WRITE OUTPUT TAPE 6, 105, I, I
      WRITE OUTPUT TAPE 6, 110
      WRITE OUTPUT TAPE 6, 130
      CALL EXIT
C
100   FORMAT (1H0,I2,30H BOTTLES OF BEER ON THE WALL,
     1,I2,16H BOTTLES OF BEER)
105   FORMAT (1H0,I2,29H BOTTLE OF BEER ON THE WALL,
     1,I2,15H BOTTLE OF BEER)
110   FORMAT (33H TAKE ONE DOWN AND PASS IT AROUND)
120   FORMAT (1H ,I2,17H BOTTLES OF BEER.)
125   FORMAT (1H ,I2,16H BOTTLE OF BEER.)
130   FORMAT (20H NO BOTTLES OF BEER.)
      END
```

From www.99-bottles-of-beer.net

Try to understand it

```
DO 30 J = 1, 98
      I = 100 - J
      WRITE OUTPUT TAPE 6, 100, I, I
      WRITE OUTPUT TAPE 6, 110
      I = I - 1
      IF (I - 1) 10, 10, 20
10    WRITE OUTPUT TAPE 6, 125, I
      GO TO 30
20    WRITE OUTPUT TAPE 6, 120, I
30    CONTINUE
      I = 1
      WRITE OUTPUT TAPE 6, 105, I, I
      WRITE OUTPUT TAPE 6, 110
      WRITE OUTPUT TAPE 6, 130
      CALL EXIT
C
100   FORMAT (1H0,I2,30H BOTTLES OF BEER ON THE WALL,
     1,I2,16H BOTTLES OF BEER)
105   FORMAT (1H0,I2,29H BOTTLE OF BEER ON THE WALL,
     1,I2,15H BOTTLE OF BEER)
110   FORMAT (33H TAKE ONE DOWN AND PASS IT AROUND)
120   FORMAT (1H ,I2,17H BOTTLES OF BEER.)
125   FORMAT (1H ,I2,16H BOTTLE OF BEER.)
130   FORMAT (20H NO BOTTLES OF BEER.)
      END
```


Fortran IV and Fortran 77

- FORTRAN IV evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters (consider a generic sort routine)
  - ANSI standard in 1966
- Fortran 77:
  - Character string handling
  - Logical loop control statement
  - **`IF-THEN-ELSE`** statement
    - Became the new standard in 1978

FORTRAN IV

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C
C     THIS PROGRAM PRINTS THE ENTIRE LYRICS OF THE SONG - 99 BOTTLES OF
C     BEER - TO STANDARD OUTPUT.
C
C     WARNING!:
C     IT IS ASSUMED THAT INTEGERS USE FOUR BYTES OF STORAGE EACH.
C
C     COPYRIGHT 2010 JOHANNES SCHOEOEN
C
C-----------------------------------------------------------------------

      DIMENSION   IBOBTW(7), IBOB(4), IT1DPI(8)

      DATA IBOBTW / 4H BOT, 4HTLES, 4H OF , 4HBEER,
     &              4H ON , 4HTHE , 4HWALL /
      DATA IBOB   / 4H BOT, 4HTLES, 4H OF , 4HBEER /
      DATA IT1DPI / 4HTAKE, 4H ONE, 4H DOW, 4HN AN,
     &              4HD PA, 4HSS I, 4HT AR, 4HOUND /

C-----------------------------------------------------------------------

C     MOST OF THE OUTPUT IS GENERATED BY THIS LOOP, BUT THE LAST COUPLE
C     OF LINES ARE TREATED SEPARATELY. (YES, I CHEAT.)
      DO 1000 I = 99, 3, -1

C        ARITHMETIC IF STATEMENTS ARE USED TO DIFFERENTIATE BETWEEN
C        DOUBLE- AND SINGLE-DIGIT OUTPUT.

C        FIRST LINE OF VERSE.
         IF (10 - I) 10, 10, 20
 10         WRITE (6, 9100) I, IBOBTW, 2H, , I, IBOB, 1H.
         GOTO 500
 20         WRITE (6, 9110) I, IBOBTW, 2H, , I, IBOB, 1H.
```


FORTRAN IV

```
C        SECOND LINE.
 500     IF (11 - I) 30, 30, 40
 30         WRITE (6, 9200) IT1DPI, 2H, , I-1, IBOBTW, 1H.
         GOTO 1000
 40         WRITE (6, 9210) IT1DPI, 2H, , I-1, IBOBTW, 1H.

 1000 CONTINUE

C     PRINT OUT THE LAST COUPLE OF ROWS.
      WRITE (6, 9999)

      STOP

C     FIRST ROW OF VERSE, FIRST DOUBLE DIGIT, THEN SINGLE DIGIT.
 9100 FORMAT(1X, I2, 7A4, A2, I2, 4A4, A1)
 9110 FORMAT(1X, I1, 7A4, A2, I1, 4A4, A1)

C     SECOND LINE OF VERSE -- DOUBLE DIGIT AND SINGLE DIGIT
 9200 FORMAT(1X, 8A4, A2, I2, 7A4, A1, /)
 9210 FORMAT(1X, 8A4, A2, I1, 7A4, A1, /)

C     THE FINAL COUPLE OF ROWS, HARD-CODED FOR SIMPLICITY.
 9999 FORMAT(1X, 49H2 BOTTLES OF BEER ON THE WALL, 2 BOTTLES OF BEER.,/,
     &       1X, 50HTAKE ONE DOWN AND PASS IT AROUND, 1 BOTTLE OF BEER,
     &       13H ON THE WALL., /,
     &       1X, 47H1 BOTTLE OF BEER ON THE WALL, 1 BOTTLE OF BEER.,
     &       13H ON THE WALL., /,
     &       1X, 50HTAKE ONE DOWN AND PASS IT AROUND, NO MORE BOTTLES ,
     &       20HOF BEER ON THE WALL., //,
     &       1X, 37HNO MORE BOTTLES OF BEER ON THE WALL, ,
     &       24HNO MORE BOTTLES OF BEER., /,
     &       1X, 35HGO TO THE STORE AND BUY SOME MORE, ,
     &       31H99 BOTTLES OF BEER ON THE WALL.)
      END
9
```


FORTRAN 77

```
      program ninetyninebottles
      integer bottles

*     99 Bottles of Beer, as implemented in FORTRAN 77
*     Written by Alex Ford - gustavderdrache@bellsouth.net
*     Notable feature: Arithmetic IF statement

      bottles = 99

*     Format statements
 1    format (I2, A)
 2    format (A)
 3    format (I2, A, /)
 4    format (A, /)

*     First 98 or so verses
 10   write (*,1) bottles, ' bottles of beer on the wall,'
      write (*,1) bottles, ' bottles of beer.'
      write (*,2) 'Take one down, pass it around...'
      if (bottles - 1 .gt. 1) then
         write (*,3) bottles - 1, ' bottles of beer on the wall.'
      else
         write (*,3) bottles - 1, ' bottle of beer on the wall.'
      end if

      bottles = bottles - 1

      if (bottles - 1) 30, 20, 10

*     Last verse
 20   write (*,1) bottles, ' bottle of beer on the wall,'
      write (*,1) bottles, ' bottle of beer.'
      write (*,2) 'Take one down, pass it around...'
      write (*,4) 'No bottles of beer on the wall.'

 30   stop
      end
```

Compare with FORTRAN II

```
      program ninetyninebottles
      integer bottles


      bottles = 99

*     Format statements
 1    format (I2, A)
 2    format (A)
 3    format (I2, A, /)
 4    format (A, /)

*     First 98 or so verses
 10   write (*,1) bottles, ' bottles of beer on the wall,'
      write (*,1) bottles, ' bottles of beer.'
      write (*,2) 'Take one down, pass it around...'
      if (bottles - 1 .gt. 1) then
         write (*,3) bottles - 1, ' bottles of beer on the wall.'
      else
         write (*,3) bottles - 1, ' bottle of beer on the wall.'
      end if

      bottles = bottles - 1

      if (bottles - 1) 30, 20, 10

*     Last verse
 20   write (*,1) bottles, ' bottle of beer on the wall,'
      write (*,1) bottles, ' bottle of beer.'
      write (*,2) 'Take one down, pass it around...'
      write (*,4) 'No bottles of beer on the wall.'

 30   stop
      end
```


Fortran 90

- Most significant changes from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - **CASE** statement
  - Parameter type checking
- Finally dropped the fixed formatting requirements used with *punch cards*
- Started using mixed case!

Fortran 90

```
!  F90 (Fortran 90) version of 99 bottles of beer.
!    written by Akira KIDA,  SDI00379@niftyserver.or.jp
!    Note that this source is in FIXED format.

     program ninetynine
     implicit none
     integer, parameter :: BOTTLES = 99
     integer :: i
     integer :: k
     character*7 :: btl = 'bottles'

     do i = BOTTLES, 1, -1
         k = len(btl)
         if (i == 1) k = k - 1
         print *, i, btl(1:k), ' of beer on the wall, ',
    c             i, btl(1:k), ' of beer.'
         print *, 'Take one down, pass it around.'
         if (i == 0) exit
         print *, i, btl(1:k), ' of beer on the wall.'
     end do
     print *, 'No more bottles of beer on the wall.'
     end
```

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003
  - Added support for OOP (like everybody else...)
  - Parameterized derived types
  - Procedure pointers
  - C language interoperability (changes in object file format)

OOP in Fortran 2003

```
type shape
        integer :: color
        logical :: filled
        integer :: x
        integer :: y
end type shape
type, EXTENDS ( shape ) :: rectangle
        integer :: length
        integer :: width
end type rectangle
type, EXTENDS ( rectangle ) :: square
end type square
```

from www.pgroup.com/lit/articles/insider/v3n1a3.htm

Fortran is different…

- Language before Fortran 90
  - Types and storage of all variables are fixed before run time
    - Speed wins the tradeoff between speed and flexibility
  - No dynamic data structures
  - No recursion – why?
- Dramatically changed forever the way computers are used
- Characterized by Alan Perlis as the *lingua franca* of the computing world

Problem: Spaghetti code

```
      SUBROUTINE OBACT(TODO)
          INTEGER TODO,DONE,IP,BASE
          COMMON /EG1/N,L,DONE
          PARAMETER (BASE=10)
     13   IF(TODO.EQ.0) GO TO 12
          I=MOD(TODO,BASE)
          TODO=TODO/BASE
          GO
     TO(62,42,43,62,404,45,62,62,62),I
          GO TO 13
     42   CALL COPY
          GO TO 127
     43   CALL MOVE
          GO TO 144
    404   N=-N
     44   CALL DELETE
          GO TO 127
     45   CALL PRINT
          GO TO 144
     62   CALL BADACT(I)
          GO TO 12
    127   L=L+N
    144   DONE=DONE+1
          CALL RESYNC
          GO TO 13
     12   RETURN
          END
```


Functional Programming: LISP

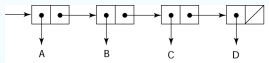
- LISt Processing language
  - Delimiters are parentheses
  - Joke: LISP = Lots of Irritating Stupid Parentheses
  - Designed by John McCarthy (MIT)
  - Replaced IPL (Information Processing Language)
- Artificial intelligence (AI) research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two primary data types: atoms and lists
- Syntax is based on Church's *lambda calculus*
  - One of several models of computation developed before computers came into existence

Lisp

- Lists – dynamic linked lists whose elements can be anything
- Lists composed of *cons cells*
- Two parts based on two registers machine had at the time: address register and data registers:
  - Two pointers of cons fit into AR and DR
  - First pointer points to element, second points to rest of the list
  - First pointer = car (contents of the AR)
  - Second pointer = cdr (contents of the DR)

Box-and-pointer example

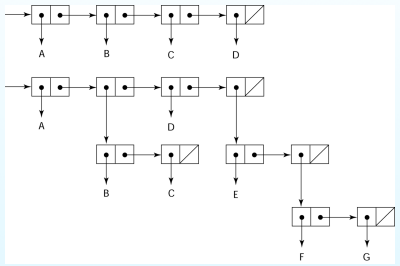
(A B C D)



Box-and-pointer example

(A B C D)

(A (B C) D (E (F G)))



Atoms

- Lisp atoms:
  - Anything that isn't a list
  - Scalar data types: integer, float, character, and, of course, and, of course, symbols
  - Some structured data types: string, complex numbers, arrays/vectors, bitstrings…
- Symbols:
  - Lisp has runtime access to its symbol tables
  - Scope and symbol tables
- Functions: also lists

Lisp

- Pioneered functional programming
  - Target domain: theorem proving
    - Required recursion and conditional expressions – features not available in FORTRAN
  - No need for variables or assignment
- Powerful macro facility
- Still the dominant language for AI (arguably)
- Common Lisp and Scheme are contemporary dialects of Lisp
- Modern Lisps: variables & assignment, loop structures, etc.
- ML, Miranda, and Haskell are related languages

Recursion and Iteration

- Fundamental control structures in any language:
  - sequential execution
  - selection/conditional execution
  - repetition
- Most languages: repetition = *iteration*
- Functional languages: repetition mostly by *recursion*
- Modeled on recursive function theory
- Developed in the 1930's: Alan Turing, Alonzo Church, Kurt Gödel, others

Lisp example

```
;; Ninety-nine bottles of beer on the wall, recursive version

(defun beer-song (n)
  (cond
   ((= n 1)
    (princ "One bottle of beer on the wall, one more bottle of beer; take one down,")
    (princ "pass it around, no   more bottles of beer on the wall."))
   (t
    (format t "~@(~R~) bottles of beer on the wall, ~:*~R bottles of beer;~%" n)
    (format t "take one down, pass it around, ~R bottle~:p of beer on the wall.~%~%"
            (1- n))
    (beer-song (1- n)))))
```

Try to understand:

```
;; Ninety-nine bottles of beer on the wall, recursive version

(defun beer-song (n)
  (cond
   ((= n 1)
    (princ "One bottle of beer on the wall, one more bottle of beer; take one down,")
    (princ "pass it around, no   more bottles of beer on the wall."))
   (t
    (format t "~@(~R~) bottles of beer on the wall, ~:*~R bottles of beer;~%" n)
    (format t "take one down, pass it around, ~R bottle~:p of beer on the wall.~%~%"
            (1- n))
    (beer-song (1- n)))))

;; Mystery function 1

(defun foo (thing1 thing2)
   (cond
     ((null thing2) nil)
     ((equal thing1 thing2) t)
     (t (foo thing1 (cdr thing2)))))

;; Mystery function2

(defun bar (thing1 thing2)
  (cond
    ((null thing2) nil)
    ((equal thing1 (second thing2)) t)
    ((bar thing1 (first thing2)))
    ((bar thing1 (third thing2)))))
```


Lisp example

```
;; Ninety-nine bottles of beer on the wall, iterative version:

(defun beer-song2 (n)
  (loop for i from n downto 1
      do
        (format t "~@(~R~) bottles of beer on the wall, ~:*~R bottles of beer;~%"
                i)
        (format t "take one down, pass it around, ~R bottle~:p of beer on the wall.~%~%"
                (1- i))
      finally
        (format t
                (concatenate 'string
                        "One bottle of beer on the wall, one more bottle of beer;"
                        "take one down,~%"
                        "pass it around, no more bottles of beer on the wall."))))
```


Slightly obfuscated Lisp example

```
(defun ninety-nine (n)
  (cond
   ((= n 0)
    (princ "No more bottles of beer on the wall."))
   (t (dotimes (i 4)
        (if (not (= i 2))
            (format t "~:(~R~) bottle~:P of beer~a~%"
                (if (= i 3) (1- n) n)
                (if (oddp i) "." " on the wall,"))
            (format t "Take one down, pass it around,~%")))
      (terpri)
      (ninety-nine (1- n)))))
```

Output

CL-USER> (ninety-nine 99)
Ninety-Nine bottles of beer on the wall,
Ninety-Nine bottles of beer.
Take one down, pass it around,
Ninety-Eight bottles of beer.

Ninety-Eight bottles of beer on the wall,
  […]
Take one down, pass it around,
One bottle of beer.

One bottle of beer on the wall,
One bottle of beer.
Take one down, pass it around,
Zero bottles of beer.

No more bottles of beer on the wall.

Obfuscated Lisp example

```
(labels ((foo (x)
           (and (<= 0 x) (cons x (foo (1- x))))))
  (format t (format nil
                    "~~{~~&~~@(~~%~~R -A -A!~~)~~:*~~&~~@(~~R
~O@*-A!~~)~~&~~@(~2@*-A!~~)~~&~~@(~~[~A~~:;~~:*~~R~~:*~~] ~O@*-A!~~)~~}"
                    "bottles of beer"
                    "on the wall"
                    "take one down, pass it around"
                    "no more"
                    )
          (foo 99)))
```


Scheme

- Descendant/dialect of LISP
- Developed at MIT — mid-1970s
- Small language
- Extensive use of static (lexical) scoping
- Functions are first-class entities
- Simple syntax (and small size) ⇒ well suited for educational applications

Common Lisp

- Goal: combine features of several dialects of Lisp (including Scheme) ⇒ single language
- Large, complex language
  - Static and dynamic scoping
  - Data types include: records, arrays, complex numbers, character strings
  - Packages facilitate abstract data type
  - OOP:
    - Flavors: Smalltalk-like
    - Later: Common Lisp Object System (CLOS)
    - CLOS: first ANSI standard for OOP

ML and other functional languages

- ML – functional language, support for imperative programming
  - Robin Milner, Edinburgh, 1970's
  - Does not use parenthesized syntax of LISP
  - Static typing
- Descendants: Miranda, Haskell, etc.
- Haskell uses lazy evaluation
  - delay expression evaluation until needed
  - some interesting capabilities – e.g., computation with infinite data structures

Pros and cons

- Comparison of FORTRAN and Lisp — pros and cons?

Toward expressiveness: ALGOL

- ALGOL development environment
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages being developed, all for specific machines
  - No portable languages; all machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 — goal was to design universal language for:
  - scientific applications
  - algorithm specification

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)
- Goals:
  - Syntax should be close to standard mathematical notation
  - Should be possible to use the language to describe algorithms in publications
  - Must be translatable to machine code

ALGOL 58

- Borrowed a lot from FORTRAN
- Concept of *type* was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**`begin ... end`**)
- Semicolon as a statement separator
- Assignment operator was :=
- **`if`** had an **`else-if`** clause
- No I/O - "would make it machine dependent"

ALGOL 58

- Not meant to be implemented
  - Variations (MAD, JOVIAL) were implemented
  - Jule's Own Version of the International Algorithmic Language (JOVIAL): official scientific language of the US Air Force until 1984
- IBM was initially enthusiastic
  - …but all support was dropped by mid-1959
  - why?

ALGOL 60

- Modified ALGOL 58 at 6-day meeting in Paris
- One of most significant developments: Backus-Naur Form (BNF) to describe syntax
- New features
  - Block structure (local/lexical scope)
  - Two parameter passing methods (by value and by name)
  - Subprogram recursion
  - Stack-dynamic arrays (variables hold index limits)
  - Still no I/O and no string handling

ALGOL 60 Successes

- Standard way to publish algorithms for over 20 years
- All subsequent imperative languages owe something to Algol 60
- Direct and indirect descendants: PL/I, Simula 97, Algol 68, C, Pascal, Ada, C++, Java, others
- First language designed to be machine-independent
- First language whose syntax was formally defined (BNF)
- Block structure and recursive subprogram calls ⇒ adoption of hardware-stack machines

ALGOL 60 Failures

- Never widely used, especially in U.S.
- Reasons:
  - Lack of I/O, the character set ⇒ programs non-portable
  - Too flexible – some features hard to implement, understand
  - Entrenchment of Fortran
  - BNF: considered strange, difficult to understand
  - Lack of support from IBM

Algol 60 Example

```
// the main program, calculate the mean of
// some numbers
begin
  integer N;
  Read Int(N);

  begin
    real array Data[1:N];
    real sum, avg;
    integer i;
    sum:=0;

    for i:=1 step 1 until N do
      begin real val;
        Read Real(val);
        Data[i]:=if val<0 then -val else val
      end;

    for i:=1 step 1 until N do
      sum:=sum   Data[i];
    avg:=sum/N;
    Print Real(avg)
  end
end
```


Easy or hard to understand?

```
// the main program, calculate the mean of
// some numbers
begin
  integer N;
  Read Int(N);

  begin
    real array Data[1:N];
    real sum, avg;
    integer i;
    sum:=0;

    for i:=1 step 1 until N do
      begin real val;
        Read Real(val);
        Data[i]:=if val<0 then -val else val
      end;

    for i:=1 step 1 until N do
      sum:=sum   Data[i];
    avg:=sum/N;
    Print Real(avg)
  end
end
```

Compared to FORTRAN?
Compared to Lisp?
Any downsides you can think of?

COBOL (Common Business Oriented Language)

- COBOL – one of the most widely used languages in the world
- Compare to ALGOL:
  - ALGOL never used, huge impact on subsequent language development
  - COBOL widely used, virtually no impact on subsequent language development (save PL/I)

COBOL

- Late 1950's
- UNIVAC used FLOW-MATIC (proprietary)
- The USAF was beginning to use AIMACO (a FLOW-MATIC variant)
- IBM was developing COMTRAN
- Grace Hopper 1953:

  "Mathematical programs should be written in mathematical notation; data processing programs should be written in English statements."

FLOW-MATIC

- Names up to 12 characters, with embedded hyphens
- English names for arithmetic operators (no arithmetic expressions)
- Data and code were completely separate
- The first word in every statement was a verb

COBOL Design Process

- First Design Meeting (Pentagon) - May 1959
- Design goals:
  - Must look like simple English
  - Must be easy to use, even if ⇒ less powerful
  - Must broaden the base of computer users
  - Must not be biased by current compiler problems
- Design committee members were all from computer manufacturers and DoD branches
- Design Problems:
  - arithmetic expressions?
  - subscripts?
  - Fights among manufacturers

COBOL Evaluation

- Contributions
  - First macro facility (`DEFINE`) in a high-level language (other than Lisp)
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division
  - Strong I/O, file operation set
- Weaknesses
  - Lack of functions
  - Prior to 1974, no parameters for subprogram calls

COBOL: DoD Influence

- First language required by DoD
- Would have failed without DoD: poor compilers
- Still most widely used business applications language
- E. Dijkstra on COBOL

  "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense."

COBOL Example 1: Multiplication

```
$ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  Multiplier.
AUTHOR.  Michael Coughlan.
* Example program using ACCEPT, DISPLAY and MULTIPLY to
* get two single digit numbers from the user and multiply them together

DATA DIVISION.

WORKING-STORAGE SECTION.
01  Num1                               PIC 9  VALUE ZEROS.
01  Num2                               PIC 9  VALUE ZEROS.
01  Result                             PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
    DISPLAY "Enter first number  (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num1.
    DISPLAY "Enter second number (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num2.
    MULTIPLY Num1 BY Num2 GIVING Result.
    DISPLAY "Result is = ", Result.
    STOP RUN.
```


Example 2: Count student records from file

```
$ SET SOURCEFORMAT "FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  StudentNumbersReport .
AUTHOR. Michael Coughlan.

*INPUT      The student record file Students.Dat  Records in this file
*           are sequenced on ascending Student Number.
*OUTPUT     Shows the number of student records in the file and the
*           number of records for males and females.
*PROCESSING For each record read;
*              Adds one to the TotalStudents count
*              IF the Gender is Male   adds one to TotalMales
*              IF the Gender is Female adds one to TotalFemales
*           At end of file writes the results to the report file.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT StudentFile ASSIGN TO "STUDENTS.DAT"
                ORGANIZATION IS LINE SEQUENTIAL.
    SELECT ReportFile ASSIGN TO "STUDENTS.RPT"
                ORGANIZATION IS LINE SEQUENTIAL.
```


Example 2: Count student records from file

```
DATA DIVISION.
FILE SECTION.
FD  StudentFile.
01  StudentDetails.
    88  EndOfStudentFile VALUE HIGH-VALUES.
    02  StudentId        PIC 9(7).
    02  StudentName.
        03 Surname       PIC X(8).
        03 Initials      PIC XX.
    02  DateOfBirth.
        03 YOBirth       PIC 9(4).
        03 MOBirth       PIC 9(2).
        03 DOBirth       PIC 9(2).
   02  CourseCode        PIC X(4).
   02  Gender            PIC X.
       88 Male           VALUE "M", "m".

FD  ReportFile.
01  PrintLine            PIC X(40).
```

Example 2: Count student records from file

```
WORKING-STORAGE SECTION.
01  HeadingLine          PIC X(21) VALUE "  Record Count Report".

01  StudentTotalLine.
    02  FILLER           PIC X(17) VALUE "Total Students = ".
    02  PrnStudentCount  PIC Z,ZZ9.

01  MaleTotalLine.
    02  FILLER           PIC X(17) VALUE "Total Males    = ".
    02  PrnMaleCount     PIC Z,ZZ9.

01  FemaleTotalLine.
    02  FILLER           PIC X(17) VALUE "Total Females  = ".
    02  PrnFemaleCount   PIC Z,ZZ9.

01  WorkTotals.
    02  StudentCount     PIC 9(4) VALUE ZERO.
    02  MaleCount        PIC 9(4) VALUE ZERO.
    02  FemaleCount      PIC 9(4) VALUE ZERO.
```


Example 2: Count student records from file

```
PROCEDURE DIVISION.
Begin.
    OPEN INPUT StudentFile
    OPEN OUTPUT ReportFile
    READ StudentFile
      AT END SET EndOfStudentFile TO TRUE
    END-READ
    PERFORM UNTIL EndOfStudentFile
       ADD 1 TO StudentCount
       IF Male  ADD 1 TO MaleCount
         ELSE   ADD 1 TO FemaleCount
       END-IF
       READ StudentFile
         AT END SET EndOfStudentFile TO TRUE
       END-READ
    END-PERFORM
    PERFORM PrintReportLines
    CLOSE StudentFile, ReportFile
    STOP RUN.
```


Example 2: Count student records from file

```
PrintReportLines.
   MOVE StudentCount TO PrnStudentCount
   MOVE MaleCount    TO PrnMaleCount
   MOVE FemaleCount  TO PrnFemaleCount

   WRITE PrintLine FROM HeadingLine
           AFTER ADVANCING PAGE
   WRITE PrintLine FROM StudentTotalLine
           AFTER ADVANCING 2 LINES
   WRITE PrintLine FROM MaleTotalLine
           AFTER ADVANCING 2 LINES
   WRITE PrintLine FROM FemaleTotalLine
           AFTER ADVANCING 2 LINES.
```

99 Bottles of beer

```
IDENTIFICATION DIVISION.
       PROGRAM-ID. 99-Bottles-of-Beer-On-The-Wall.
       AUTHOR. Joseph James Frantz.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 Keeping-Track-Variables.
          05 Bottles                    PIC S99   VALUE 0.
          05 Remaining-Bottles          PIC S99   VALUE 0.
          05 Counting                   PIC 99    VALUE 0.
          05 Start-Position             PIC 99    VALUE 0.
          05 Positions                  PIC 99    VALUE 0.
       PROCEDURE DIVISION.
       PASS-AROUND-THOSE-BEERS.
        PERFORM VARYING Bottles FROM 99 BY -1 UNTIL Bottles = -1
          DISPLAY SPACES
          SUBTRACT 1 FROM Bottles GIVING Remaining-Bottles
          EVALUATE Bottles
            WHEN 0
              DISPLAY "No more bottles of beer on the wall, "
                      "no more bottles of beer."
              DISPLAY "Go to the store and buy some more, "
                      "99 bottles of beer on the wall."
            WHEN 1
              DISPLAY "1 bottle of beer on the wall, "
                      "1 bottle of beer."
              DISPLAY "Take one down and pass it around, "
                      "no more bottles of beer on the wall."
```


99 Bottles… (cont'd)

```
            WHEN 2 Thru 99
              MOVE ZEROES TO Counting
              INSPECT Bottles,
                 TALLYING Counting FOR LEADING ZEROES
              ADD 1 TO Counting GIVING Start-Position
              SUBTRACT Counting FROM 2 GIVING Positions
              DISPLAY Bottles(Start-Position:Positions)
                      " bottles of beer on the wall, "
                      Bottles(Start-Position:Positions)
                      " bottles of beer."
              MOVE ZEROES TO Counting
              INSPECT Remaining-Bottles TALLYING
                 Counting FOR LEADING ZEROES
              ADD 1 TO Counting GIVING Start-Position
              SUBTRACT Counting FROM 2 GIVING Positions
              DISPLAY "Take one down and pass it around, "
                      Remaining-Bottles(Start-Position:Positions)
                      " bottles of beer on the wall."
          END-EVALUATE
        END-PERFORM
       STOP RUN.
```


Compare & contrast

- FORTRAN
- Algol
- Lisp
- COBOL

BASIC

- Like COBOL, widely used but gets little respect
  - "The Rodney Dangerfield of computer languages"
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be "pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- BASIC was designed for interactive terminals on a time-sharing system

BASIC

- Based on FORTRAN
- Many different versions came into existence; 1978 ANSI standard was minimal
- Digital used a version of BASIC to write part of the operating system for the PDP-11

E. Dijkstra on BASIC

It is practically impossible to teach good programming to students that have had a prior exposure to BASIC; as potential programmers they are mentally mutilated beyond hope of regeneration.

Unstructured Programming

- Djikstra's comment referred to code like this:

```
10 IF X = 42 GOTO 40
20 X = X + 1
30 GOTO 10
40 PRINT "X is finally 42!"
```


Modern BASIC

- Most hobby computers in1970s had tiny BASIC interpreters
- MS-DOS include BASICA and later QBASIC
- With Windows, Microsoft started developing Visual Basic
  - Even the oldest VB versions: object-oriented languages with classes, inheritance, etc.
  - Visual Studio 6 (1998) was the most popular version
  - VBScript was (and still is) used for web development (Classic ASP)
  - VBA was (and still is) used to automate Office applications

VB.NET

- VB 7
  - released 2002 with .NET
  - broke compatibility with earlier versions
- Can be used for anything from console applications to web development
- Virtually same capabilities as C#
- Visual Studio 2008: VB acquired capabilities such as
  - lambda expressions
  - anonymous types
  - type inferencing, etc.

Everything for everybody: PL/I

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
  - Scientific computing
    - IBM 1620 and 7090 computers
    - FORTRAN
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

PL/I: Background

- By 1963
  - Scientific users began to need more elaborate I/O
  - Business users began to need floating point type, arrays for MIS
  - Too costly to have two kinds of computers, languages
- Obvious solution
  - Build new computer to do both kinds of applications
  - Design new language to do both kinds of applications
  - Goal: PL/I could replace COBOL, FORTRAN, LISP and assembler

PL/I: Design Process

- Designed in five months by the 3 X 3 Committee
  - Three members from IBM, three members from SHARE
- Initial concept was an extension of Fortran IV
- Initially: NPL (New Programming Language)
- Name changed (1965): PL/I (Programming Language/I)

PL/I Overview

- Famous for "kitchen sink" approach
- PL/I contributions:
  - Programs could create concurrently executing subprograms
  - First exception handling in a programming language
  - Recursion allowed, but could disabled for efficient function calls
  - Pointer data type
  - Array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

PL/I …

- Partial success, but…
  - slow compilers
  - difficult-to-use features,
  - partial implementations
  - buggy compilers
- Many subsets: PL/C for teaching, PL/S for systems programming,…
- Widely used in 1970's on mainframes
- Used for IBM OS development
- Usage continued until the 1990's with some PC implementations
- Virtually dead now

PL/I example

```
BOTTLES: PROC OPTIONS(MAIN);

   DCL NUM_BOT FIXED DEC(3);
   DCL PHRASE1 CHAR(100) VAR;
   DCL PHRASE2 CHAR(100) VAR;
   DCL PHRASE3 CHAR(100) VAR;

   DO NUM_BOT = 100 TO 1 BY -1;

      PHRASE1 = NUM_BOT||' Bottles of Beer on the wall,';
      PHRASE2 = NUM_BOT||' Bottles of Beer';
      PHRASE3 = 'Take one down and pass it around';
      DISPLAY(PHRASE1||PHRASE2);
      DISPLAY(PHRASE3);
   END;
   PHRASE1 = 'No more Bottles of Beer on the wall, ';
   PHRASE2 = 'No more Bottles of Beer';
   PHRASE3 = 'Go to the store and buy some more';
   DISPLAY(PHRASE1||PHRASE2);
   DISPLAY(PHRASE3);
END BOTTLES;
```

Ada

- Named for Augusta Ada Byron, Countess of Lovelace
  - Lord Byron's daughter
  - Worked with Charles Babbage
  - First programmer in history
- Department of Defense (DoD) drove development:
  - Explosion of PLs in use: ~450 by 1974
  - Embedded systems:
    - > half of applications
    - Many: assembly language for special-purpose processors
    - Usually: no HLL suitable
  - Little code reuse
  - No general SW development tools

Ada

- High Order Language Working Group (HOLWG) produced requirement documents
- Huge design effort:
  - hundreds of people, much $$, and ~8 years
  - Phases:
    - Strawman requirements (April 1975)
    - Woodenman requirements (August 1975)
    - Tinman requirements (1976)
    - Ironman requirments (1977)
    - Steelman requirements (1978)
- By 1979: 100s of proposals → 4 — all based on Pascal

Ada

- Contributions
  - Packages - data abstraction by encapsulating data types, objects and procedures
  - Elaborate exception handling model
  - Generic program units: allowed algorithms to be implemented without specifying data types
  - Concurrency - through rendezvous mechanism
- Good:
  - Competitive design
  - Included all then known about SW engineering, PL design
- Not so good:
  - Building first compiler: very difficult
  - First really usable compiler: ~5 years after PL design complete

Ada 95

- Ada 95 (began in 1988)
- Packages: very similar to classes
- …but no components could be added to base "class"
- Added support for OOP:
  - type derivation
  - runtime subprogram dispatching
- Better control mechanisms for shared data
- New concurrency features
- More flexible libraries
- Popularity decreased over time: DoD no longer requires it

Ada example

```
with Text_IO;
procedure Bar is

  Out_Of_Beer : Exception;

  protected Bartender is
    function Count return Integer;
    procedure Take_One_Down;
  private
    Remaining : Integer range 0 .. 99 := 99;
  end Bartender;

  protected body Bartender is
    function Count return Integer is
    begin return Remaining; end Count;

    procedure Take_One_Down is
    begin
      if Remaining = 0 then raise Out_Of_Beer;
      else Remaining := Remaining - 1;
      end if;
    end Take_One_Down;
  end Bartender;
```


Ada example

```
type Names is (Charles, Ada, John, Grace, Donald,
               Edsger, Niklaus, Seymour, Fred, Harlan);

task type Customers is
  entry Enter_Bar(Who : in Names);
end Customers;

Customer_List : array(Names) of Customers;

task body Customers is
  Me : Names;
  procedure Sing_And_Drink(Singer_ID : in String) is
    procedure Sing(S : in String) renames Text_IO.Put_Line;
  begin
    loop
      declare
        Bottle_Part : constant String
          := Integer'image(Bartender.Count) & " bottles of beer";
      begin
        Sing(Bottle_Part & " on the wall" & Singer_ID);
        Sing(Bottle_Part & Singer_ID);
      end;
      Sing(" Take one down and pass it around" & Singer_ID);
      Bartender.Take_One_Down;
      delay 10.0;  -- allow ten seconds to gulp one down
    end loop;
  exception
    when Out_Of_Beer => Sing("no more beer!" & Singer_ID);
  end Sing_And_Drink;
```

Ada example (cont'd)

```
  begin -- customer task
    accept Enter_Bar(Who : in Names) do
      Me := Who;
    end Enter_Bar;
    Sing_And_Drink(" - " & Names'image(Me));
  end Customers;

begin -- operating bar

  for Person in Customer_List'range loop
    Customer_List(Person).Enter_Bar(Person);
    delay 2.0; -- allow two seconds between customers
entering bar
  end loop;

end Bar;
```


Parallels with RISC/CISC

- Take a couple of minutes: any parallels between:
  - FORTRAN & COBOL vs PL/I & Ada

    and
  - RISC vs CISC?

RISC = Reduced Instruction Set Computers

CISC = Complex Instruction Set Computers

Early dynamic languages

- Dynamic typing and dynamic storage allocation
- "Variables are untyped" ⇒ "no types declared"
- Variable acquires type when assigned a value
- Storage allocated when variable assigned value
- First: Lisp
- Other early ones: APL, SNOBOL
- Now: Ruby, Python, …

APL: A Programming Language

- Designed as hardware description language at IBM by Ken Iverson around 1960
- Highly expressive - many operators, for both scalars and arrays of various dimensions
- Programs very difficult to read:
  - use of single special characters for complex operations
  - called a "write-only" language
- Still in use after 45 years; minimal changes

Example

- Using the "Sieve of Eratosthenes" method, find all prime numbers less than or equal to X
- C version:

```
/* Sieve of Eratosthenes in C */
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
    unsigned long n, x, y, *primes;

    /* Get the upper limit value, n */
    if (argc != 2) {
        fprintf (stderr, "Usage is e.g:\n    %s 10\n", argv[0]);
        return -1;
    }

    n = strtoul (argv[1], NULL, 0);
    if (n == 0) {
        fprintf (stderr, "Argument must be greater than 0\n");
        return -1;
    }

    /* Run the sieve algorithm */
    primes = (unsigned long *) calloc (n+1, sizeof (unsigned long));
    if (primes == NULL) {
        fprintf (stderr, "Out of memory\n");
        return -1;
    }

    for (x = 2; x <= n; x++) {
        for (y = 2; y <= x; y++) {
            if (x * y > n)
                break;
            primes [x * y] = 1;
        }
    }

    /* Print the results */
    for (x = 2; x <= n; x++) {
        if (primes [x] == 0)
            printf ("%d ", x);
    }
    printf ("\n");
    return 0;
}
```


Example

- Using the "Sieve of Eratosthenes" method, find all prime numbers less than or equal to X
- APL version:

```
(2=+⌿0=(⍳X)∘.|⍳X)/⍳X
```

- And, no, I can't explain it to you!

SNOBOL (String Oriented Symbolic Language)

- String manipulation language (Bell Labs; Farber, Griswold, Polensky, 1964)
- Patterns: first-class objects in the language
  - Could be very simple (strings)
  - Could be entire programming language grammars
- Powerful string pattern matching operators
  - Can create strings, treat as programs, execute them
  - SNOBOL patterns equivalent to context-free grammars
  - More powerful than regular expressions (e.g., Perl, JavaScript, awk, etc.)
- Pattern matching: backtracking algorithm similar to Prolog execution

SNOBOL

- Still used for some text-processing tasks
  - See http://www.snobol4.org
- SPITBOL (A Speedy Implementation of SNOBOL) — released under GNU license in 2009
  - See http://code.google.com/p/spitbol/

SNOBOL example

```
        B = 99
LOOP    SENTENCE1 = "?? BOTTLES OF BEER ON THE WALL, ?? BOTTLES OF BEER"
        SENTENCE2 = "TAKE ON AND DOWN PASS IT AROUND, ?? BOTTLES OF BEER ON THE WALL."
S1      SENTENCE1 "??" = B                              :S(S1)
S2      SENTENCE1 "BOTTLES" = EQ(B,1) "BOTTLE"          :S(S2)
        OUTPUT = SENTENCE1
        B = B - 1
        EQ(B,0)                                         :S(FINISH)
        SENTENCE2 "??" = B
        SENTENCE2 "BOTTLES" = EQ(B,1) "BOTTLE"
        OUTPUT = SENTENCE2
        OUTPUT = " "
        GT(B,0)                                         :S(LOOP)
FINISH  OUTPUT = "TAKE ONE DOWN AND PASS IT AROUND, NO MORE BOTTLES OF BEER ON THE WALL."
        OUTPUT = " "
        OUTPUT = "NO MORE BOTTLES OF BEER ON THE WALL, NO MORE BOTTLES OF BEER"
        OUTPUT = "GO TO THE STORE AND BUY SOME MORE, 99 BOTTLES OF BEER ON THE WALL."
END
```

Python

- Interpreted, dynamic ("scripting") language
- Guido van Rossum; named after Monty Python
- Type checked but dynamically typed
- Basic data types: numbers, etc., and lists, tuples, and hashes (associative arrays)
- Designed for readability – spaces as delimiters
- Designed as an extensible language
- Large set of libraries available
- Very widely used
- A major language for Deep Learning — with libraries

Example

```
"""
99 Bottles of Beer (by Gerold Penz)
Python can be simple, too :-)
"""

for quant in range(99, 0, -1):
   if quant > 1:
      print quant, "bottles of beer on the wall,", quant, "bottles of beer."
      if quant > 2:
         suffix = str(quant - 1) + " bottles of beer on the wall."
      else:
         suffix = "1 bottle of beer on the wall."
   elif quant == 1:
      print "1 bottle of beer on the wall, 1 bottle of beer."
      suffix = "no more beer on the wall!"
   print "Take one down, pass it around,", suffix
   print "--"
```


Example

```
# Readable Python Version of "99 Bottles of Beer" Program
# Well, its readable if you know Python reasonably well.
# Public Domain by J Adrian Zimmer [[ jazimmer.net ]]

verse1 = lambda x: \
"""%s of beer on the wall, %s of beer.
Take one down, pass it around, %s of beer on the wall.
""" % (bottle(x),bottle(x),bottle(x-1))

verse2 = \
"""No more bottles of beer on the wall, no more bottles of beer.
Go to the store, buy some more, 99 bottles of beer on the wall.
"""

def verse(x):
    if x==0:     return verse2
    else:        return verse1(x)

def bottle(x):
    if x==0:   return "no more bottles"
    elif x==1: return str(x) + " bottle"
    else:      return str(x) + " bottles"

print "\n".join( [ verse(x) for x in range(99,-1,-1) ] )
```

Example

```
#! /usr/bin/env python

class BottleException(Exception):
    def __init__(self, i, c):
        self.cause = c
        self.cnt = i
        try:
            a = 1/(99-i)
            raise BottleException(i+1, self)
        except ZeroDivisionError:
            pass

    def getCause(self):
        return self.cause

    def printStackTrace(self):
        print("%d Bottle(s) of beer on the wall, %d Bottle(s) of beer" % (self.cnt, self.cnt))
        print("Take one down and pass it around,")
        print("%d Bottle(s) of beer on the wall" % (self.cnt - 1))
        try:
            self.getCause().printStackTrace()
        except AttributeError:
            pass

try:
    raise BottleException(1, None)
except Exception, e:
    e.printStackTrace()
```


Example

```
a,t="\n%s bottles of beer on the wall","\nTake one down, pass it around"
for d in range(99,0,-1):print(a%d*2)[:-12]+t+a%(d-1 or'No')
```


The Zen of Python

http://www.python.org/dev/peps/pep-0020/

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-and preferably only one-obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Lua

- Functional and imperative
- Limited support for OO as with Javascript
- Functions are first-class values
- Designed to be extensible
- Only 21 reserved words
- Small number of atomic data types (Booleans, Numbers, Strings)
- One complex data structure called a "table" similar to associative arrays or hashes
- Lua very popular for video game scripting: used in both Angry Birds and in the Flame cyber weapon

Lua example

```
do
  local oldprint = print    -- Store current print function as oldprint
  function print(s)         -- Redefine print function, the usual print function
can still be used
    if s == "foo" then
      oldprint("bar")
    else
      oldprint(s)
    end
  end
end
```

from <sigh> Wikipedia

Lua Example

```
-- Lua 99 Bottles of Beer
-- by Philippe Lhoste <PhiLho@GMX.net> http://jove.prohosting.com/~philho/

function PrintBottleNumber(n)
  local bs
  if n == 0 then
    bs = "No more bottles"
  elseif n == 1 then
    bs = "One bottle"
  else
    bs = n .. " bottles"
  end
  return bs .. " of beer"
end

for bn = 99, 1, -1 do
  write(PrintBottleNumber(bn), " on the wall, \n")
  write(PrintBottleNumber(bn), "\n")
  write("Take one down and pass it around,\n")
  write(PrintBottleNumber(bn-1), " on the wall, \n\n")
end
write("No more bottles of beer on the wall,\nNo more bottles of beer\n")
write("Go to the store, buy some more!\n")
```

Ruby

- Author: Yukihiro Matsumoto (a.k.a, "Matz")
- Began as a replacement for Perl and Python
- Pure object-oriented language – everything is an object
- Operators are methods, can be redefined
- A number features from Perl (e.g., implicit variables)
- Extensible like Python, Perl
- First Japanese language widely adopted in US
- Ruby on Rails: popular web application framework using Ruby
- Example: www.99-bottles-of-beer.net/language-ruby-1272.html

Example

```
class Wall

  def initialize(num_of_bottles)
    @bottles = num_of_bottles
  end

  def sing_1_verse
    @output = sing_num(@bottles) + " on the wall, " + sing_num(@bottles) + "\n"
    @output += "take one down, pass it around, " + sing_num(@bottles-1) + "\n\n"
    return @output
  end

  def sing_all
    @output = ''
    while @bottles > 0 do
      @output += sing_1_verse()
      @bottles -= 1
    end
    return @output
  end

  def sing_num(num)
    @counter = (num > 1) ? 'bottles' : 'bottle'
    "#{num} #{@counter} of beer"
  end

end # class Wall

wall = Wall.new(99)
puts wall.sing_all()
```


Dynamic languages

- Pros and cons vs "static" languages (with some/most static storage & with types)

Orthogonal design philosophy

- Provide a few basic, non-overlapping concepts
  - control structures
  - variables/types
  - other features
- Provide a few combining mechanisms
- Pro: Clean language, small grammars, smaller/faster compilers/interpreters
- Con: Puts effort → programmer, longer programs
- (RISC vs CISC again?)

ALGOL 68

- Continuation of ALGOL 60 – but not a superset
- Didn't achieve widespread use – but introduced:
  - User-defined data types
  - Dynamic arrays
  - Reference types
- Strongly influenced subsequent languages, especially Pascal, C, and Ada
- Language used to describe it was a problem

> *"The coercion is called deproceduring. This can be employed in any soft, and therefore any weak, meek, firm or strong syntactic position."*
>
> *From ALGOL 68: A First and Second course (Andrew D. McGettrick)*

ALGOL 68 example

```
# 99 Bottles of Beer                          #
# by Otto Stolz <Otto.Stolz@Uni-Konstanz.de> #
( PROC width = (INT x) INT: (x>9 | 2 | 1)
; FOR i FROM 99 BY -1 TO 1
  DO  printf ( ( $ 2l n(width(i))d
                 , x "bottle" b("","s") x "of beer on the wall,"
                 , x n(width(i))d
                 , x "bottle" b("","s") x "of beer."
                 , l "Take one down, pass it around,"
                 , x n(width(i-1))d
                 , x "bottle" b("","s") x "of beer."
                 $
               , i  , i=1
               , i  , i=1
               , i-1, i=2
             ) )
  OD
)
```

Pascal

- ALGOL strongly influenced development of Pascal (Wirth, 1971)
- Niklaus Wirth was member of ALGOL 68 committee

> *"If you call me by name, it is Neeklaws Veert, but if you call me by value, it is Nickle's Worth."*

- Designed for teaching structured programming
- Small, simple, nothing really new
- 70s–90s: most widely-used teaching language
- Emphasis on reliable programming: type-safety, index bounds check, etc.

Pascal

- Lacked features necessary for real-world programming, e.g.:
  - separate compilation
  - decent I/O
- Non-standard dialects were developed
  - E.g., Turbo Pascal (Borland) for IBM PC
  - 35 KB of code written in assembler
  - Included complete editor, compiler and debugger
- ⇒ Modula-2
- Pascal not used much anymore
- Delphi is OO descendent, in use

Pascal example

```
program BottlesOfBeer (output);
{this program plays the 99 bottles of beer song}

const
  BOTTLESSTART = 99;
  BOTTLESEND = 1;

type
  tBottles = BOTTLESEND..BOTTLESSTART;

var
  bottles : tBottles;

begin
  for bottles := BOTTLESSTART downto BOTTLESEND do
  begin
    if bottles > 1 then
    begin
      writeln (bottles,' bottles of beer on the wall, ',bottles, ' bottles of beer.');
      write ('Take one down, pass it around, ');
      writeln (bottles - 1, ' bottles of beer on the wall.');
      writeln
    end
    else
    begin
      writeln ('1 bottle of beer on the wall, one bottle of beer.');
      writeln ('Take one down, pass it around, no more bottles of beer on the wall');
      writeln;
      writeln ('No more bottles of beer on the wall, no more bottles of beer.');
      writeln ('Go to the store and buy some more, 99 bottles of beer on the wall.')
    end
  end
end.
```

C

- C language designed 1972 (Dennis Richie, Bell Labs)
- For systems programming
- Evolved primarily from BCLP, B, but also ALGOL 68
  - BCLP and B are not typed languages
  - All data: considered to be machine words
- Very low-level HLL
- Powerful set of operators – poor type checking
- Used to develop Unix
- Very widely used, esp. for systems programming

C

- No standard for the language initially
  - Kernigan and Ritchie's *C Programming Language*
  - ANSI standard created in 1989
  - 2nd edition of K&R came out after 1989 ANSI C
- Weak type support/checking: e.g.:
  - No boolean types: ints are used
  - No built-in character or string support
    - Characters: 8-bit numbers (`char`)
    - Strings: arrays of `char`
  - Pointers
  - Little or no runtime type checking

Prolog

- Logic-based programming language
- Developed by Comerauer, Roussel, & Kowalski (U. Aix-Marseille and U. Edinburg)
- Based on subset of predicate logic — Horn clauses
  - Disjunction with at most one negated literal
  - Equiv: $X_1 \vee X_2 \vee \ldots \vee X_{n-1} \Rightarrow X_n$
- Resolution theorem proving
  - Inference mechanism: $(A \vee B) \wedge (\neg A \vee B) \rightarrow B$
  - Backtracking search built in
- Non-procedural – declarative
- Can view: intelligent DB system w/ inferencing ⇒ truth of queries
- Inefficient…
- … but some Prolog chips were developed ⇒ high-speed inferencing

Prolog Programs

- Consist of two components: facts and rules. Ex:

```
speaks(allen, russian).
speaks(bob, english).
speaks(mary, russian).
speaks(mary, english).
talkswith(P1,P2) :- speaks(P1,L),speaks(P2,L), P1\= P2.
```


Prolog programs

- Consist of two components: facts and rules. Ex:

```
speaks(allen, russian).
speaks(bob, english).
speaks(mary, russian).
speaks(mary, english).
talkswith(P1,P2) :- speaks(P1,L),speaks(P2,L), P1\= P2.
```

- Queries:

```
?- speaks(Who, russian).
```

  - Asks for: instantiation of variable `Who` that makes the query true
  - asks for an instantiation of the variable Who for which the query `speaks(Who, russian)` succeeds.
  - Prolog considers every fact and rule whose head is `speaks`. (If more than one, consider them in order.)

```
Who = allen ;
Who = mary ;
No
```


Prolog example

```
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of beer,'), nl,
    write('Take one down, and pass it around,'), nl,
    write('Now they are all gone.'), nl,!.
bottles(X) :-
    write(X), write(' bottles of beer on the wall,'), nl,
    write(X), write(' bottles of beer,'), nl,
    write('Take one down and pass it around,'), nl,
    NX is X - 1,
    write(NX), write(' bottles of beer on the wall.'), nl, nl,
    bottles(NX).
```

Try to understand it:

```
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of beer,'), nl,
    write('Take one down, and pass it around,'), nl,
    write('Now they are all gone.'), nl,!.
bottles(X) :-
    write(X), write(' bottles of beer on the wall,'), nl,
    write(X), write(' bottles of beer,'), nl,
    write('Take one down and pass it around,'), nl,
    NX is X - 1,
    write(NX), write(' bottles of beer on the wall.'), nl, nl,
    bottles(NX).
```

Call with: bottles(99)

OO languages

- Early: Simula, CLU, Smalltalk
- Later: Objective C, Swift, Ruby
- Mixed: Java, C++, C#
- Add-ons to other languages:
  - Flavors (Lisp)
  - CLOS (Lisp)
  - C++ (originally)
  - Python
  - Perl
  - Fortran, COBOL, etc., etc.

SIMULA 67

- Simulation language (Nygaard & Dahl; Norway)
- Based on ALGOL 60 – superset of it
- First OO language (though cf. CLU)
- Primary contributions:
  - Classes, objects, and inheritance
  - Coroutines - a kind of subprogram
- The main ancestor of Smalltalk

Simula example

```
BEGIN
   COMMENT
      Simula version of 99 beers
      Maciej Macowicz (mm@cpe.ipl.fr)
      Status: UNTESTED :)

      Amended 2007-03-10 by Jack Leunissen (jack.leunissen@wur.nl)
      Status: WORKING (at least it prints and counts correctly)
   ;
   INTEGER bottles;
   INTEGER num;

   num := 2;
   FOR bottles:= 99 STEP -1 UNTIL 1 DO
   BEGIN
     IF (bottles < 10) THEN num := 1;
     OutInt(bottles,num);
     OutText(" bottle(s) of beer on the wall, ");
     OutInt(bottles,num);
     OutText(" bottle(s) of beer");
     OutImage;
     Outtext("Take one down, pass it around, ");
     OutInt(bottles - 1,num);
     OutText(" bottle(s) of beer on the wall. ");
     OutImage;
     OutImage;
   END;
   OutText("1 bottle of beer on the wall, one bottle of beer.");
   OutImage;
   OutText("Take one down, pass it around, ");
   OutText("no more bottles of beer on the wall");
   OutImage
END
```


Smalltalk

- One of the first object-oriented languages
- Xerox PARC — Alan Kay, Adele Goldberg
- First full implementation of an OO language
  - data abstraction
  - inheritance
  - dynamic binding
- Kay foresaw development of desktop PC, use of computers by non-programmers
- Pioneered the graphical user interface design based on a desktop model
- Model adopted with permission by Macintosh after Steve Jobs visited PARC…
- …then "borrowed" by Microsoft (and Linux, and…)

Smalltalk

- Very small, simple language
- No conventional control structures:

  uses objects + messages instead
- Much of Smalltalk is defined in Smalltalk
- Smalltalk world: populated by objects
  - booleans, numbers, strings
  - also large complex things — e.g., Class BitBlt used for drawing bitmaps
- Objects pass messages to other objects

Smalltalk example

Count the number of characters are 'a' or 'A' in collection letters:

```
count ← 0.
letters do: [:each | each asLowercase == $a
                        ifTrue: [count ← count + 1]]
```


Smalltalk: longer example

```
' Smalltalk class to constrain a 2D point to a fixed grid
Point subclass: #GriddedPoint
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
!GriddedPoint methodsFor: 'accessing'!

x: xInteger
  "Set the x coordinate gridded to 10 (using rounding, alternatively I
   could use truncating)."
        ^ super x: (xInteger roundTo: 10)

y: yInteger
  "Set the y coordinate gridded to 10 (using rounding, alternatively I
   could use truncating)."
        ^ super y: (yInteger roundTo: 10)

!GriddedPoint methodsFor: 'private'!
setX: xPoint setY: yPoint
  "Initialize the instance variables rounding to 10."
   ^ super setX: (xPoint roundTo: 10) setY: (yPoint roundTo: 10)
```

See if you can figure this out:

```
"Copy into a workspace, highlight the code and choose do it."
"Tested under Squeak 3.7 and VisualWorks 7.3"
| verseBlock |
verseBlock := [ :bottles  | | verse |
  verse := WriteStream with: (String new).
    bottles = 0 ifTrue:
      [verse
        nextPutAll: 'No more bottles of beer on the wall. No more bottles of beer...'; cr;
        nextPutAll: 'Go to the store and buy some more... 99 bottles of beer.'; cr].
    bottles = 1 ifTrue:
      [verse
        nextPutAll: '1 bottle of beer on the wall. 1 bottle of beer...'; cr;
        nextPutAll: 'Take one down and pass it around, no more bottles of beer on the wall';
cr].
    bottles > 1 ifTrue:
      [verse
        nextPutAll: bottles printString; nextPutAll: ' bottles of beer on the wall. ';
        nextPutAll: bottles printString; nextPutAll: ' bottles of beer...'; cr;
        nextPutAll: 'Take one down and pass it around, ';
        nextPutAll: (bottles - 1) printString, ' bottle';
        nextPutAll: (((bottles - 1) > 1) ifTrue: ['s '] ifFalse: [' ']);
        nextPutAll: 'of beer on the wall'; cr].
  verse contents].

99 to: 0 by: -1 do: [: i | Transcript show: (verseBlock value: i); cr].
```

Objective C

- Early: Pre-processor to add OO to C
- Brad Cox (Stepstone)
- NeXT Computer (Steve Jobs) licensed it for NeXTSTEP, later bought rights
- Apple acquired NeXT & Objective C in 1996
- Objective C became Apple's main language until recently:
  - Mac OS/macOS (now OS X)
  - iOS

Hansen Hsu, https://medium.com/chmcore/a-short-history-of-objective-c-aff9d2bdeBdd

Objective-C

- Merged ideas from C & Smalltalk
- Pure OO language (pretty much)
- Communication between objects: messages
- Separate interface and implementation files
- Compilers: Xcode, gcc, others
- Garbage collection facilities
- No type checking for messages

Example

- Long example here

Swift

- Created by Apple to replace Objective-C
- Modern, powerful, easy-to-learn OOP language
- Xcode support
- Interoperable with Objective-C
- Open source (swift.org)
- Objects/classes, closures, enumerated types, generic functions & types, tuples, optional types, type-safety, type inference, exception handling, assertions and preconditions,
- The Playground in Xcode

Example

```
var i = 99
while i > 0
    println(i + " bottles of beer on the wall, " + i + "bottles of beer.")
    var num = i - 1
    if i == 1 {
      var num = "no more"
    }
    println("Take one down and pass it around, " + num + "bottles of beer on the wall.")
println("No more bottles of beer on the wall, no more bottles of beer.")
println("Go to the store and buy some more, 99 bottles of beer on the wall.")
```


Example

```
import Foundation

func primes(n: Int) -> AnyGenerator<Int> {

  var (seive, i) = ([Int](0..<n), 1)
  let lim = Int(sqrt(Double(n)))

  return anyGenerator {
    while ++i < n {
      if seive[i] != 0 {
        if i <= lim {
          for notPrime in stride(from: i*i, to: n, by: i) {
            seive[notPrime] = 0
          }
        }
        return i
      }
    }
    return nil
  }
}
```

C++: Imperative programming & OOP

- Bell Labs, Bjarne Stroustrup in 1980
- Evolved from C and SIMULA 67 (⇒ OO)
- Design: provide classes, inheritance w/ no performance hit
- Exception handling
- Large and complex language – in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft's version (released with .NET in 2002)
  - Managed C++
  - delegates, interfaces, no multiple inheritance

OO and C++

- Alan Kay coined the term "object oriented"

  "…and I can tell you I did not have C++ in mind."
- Combining object oriented constructions with a low-level language like C can produce some strange constructs:

  "If you think C++ is not overly complicated, just what is a protected abstract virtual base pure virtual private destructor, and when was the last time you needed one?"

  – Tom Cargil, *C++ Journal*

Related OOP languages

- Eiffel (designed by Bertrand Meyer - 1992)
  - Not directly derived from any other language
  - Smaller and simpler than C++, but still has most of the power
  - Lacked popularity of C++ – many potential C++ programmers already used C
- Delphi (Borland)
  - Pascal plus features to support OOP
  - Smaller, more elegant and safer than C++
- Example:
  www.99-bottles-of-beer.net/language-c++-108.html

Java

- Sun Microsystems in the early 1990s
  - Needed language for embedded electronics
  - C and C++ were deemed unsatisfactory
  - They are unsafe, unreliable and not (truly?) object-oriented
- Based on C++
- Significantly simplified:
  - Does not allow pointer arithmetic
  - Only allows safe "widening" type coercions, e.g., int → float is OK, float → int is not
  - Does not include struct, union, enum (Why not?)
  - Completely OO
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency
  - Automated memory management
  - Does not support multiple inheritance

Java

- Eliminated many unsafe features of C++ – at the expense of verbosity, some convenience (e.g., pointer arithmetic)
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Use increased faster than almost any previous language
- Java 6 was released in 2006 with significant runtime performance enhancement
- Current (2018) version: 8
- Example?

Well, *you* should be able to write the example!

C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- A few improvements over C++
- Provides a language for component-based software development
- All .NET languages use Common Type System (CTS) – common class library
- Compiled to byte code for the Common Language Runtime (CLR)
- Used as scripting language in Unity, e.g.
- Example: at bottles of beer website

Scripting Languages

- Scripting languages
  - Designed for particular environment
  - Automate things that could be done by commands
  - E.g., file/computer management tools for superusers
- Shell: command processor
  - sh (Bourne shell) – first one, on Unix
  - Many others: e.g., ksh (Korn shell), csh (C shell), bash (Bourne-again shell) – even DOS
- Script: list of shell commands in a file
- awk: a scripting/programming language for text manipulation
- Perl: scripting language for systems work, reports
- Scripting languages and web (server-side)

sh example

```
#!/bin/sh
#The real sh not with bash extensions
#for testing use dash as interpreter because sh is often simlinked to bash
bottles(){
    if test $1 -eq 1
        then echo 1 bottle
    elif test $1 -eq 0
        then echo no more bottles
    else echo $1 bottles
    fi
}
i=99
until test $i -eq 0
    do echo `bottles $i` of beer standing on the wall, `bottles $i` of beer.
    i=$(($i-1))
    echo Take one down and pass it around, `bottles $i` of beer on the wall.
    echo
done
echo No more bottles of beer standing on the wall, `bottles 0` of beer.
echo Go to the store and buy some more, `bottles 99` of beer on the wall.
```


Perl

- Larry Wall, 1987
- Kind of combination of sh and awk
- Variables are statically typed but implicitly declared
- Three distinctive namespaces, denoted by the first character of a variable's name:
  - $xxx – scalar
  - @xxx – array
  - %xxx – associative array
- Large number of implicit variables, e.g., $_, @_, $$
- Very expressive: "Swiss Army chainsaw"
- Difficult to read

Perl

- Somewhat dangerous: type coercions
- Gained widespread use for UNIX administration, then CGI programming on the Web
- Now extensively used in computational biology and bioinformatics, still for systems work

Perl example

```
#!/usr/bin/perl

$num = 99;

while ($num > 1) {
    print("$num bottles of bear on the wall, $num bottles of beer.\n");
    $num--;
    print("Take one down, pass it around, $num bottle");
    print("s") if $num > 1;
    print(" of bear on the wall\n");
}
print("One bottle of beer on the wall, one bottle of beer.\n");
print("Take it down, pass it around, no more bottles of beer on the
wall.\n");

1;
```


Perl example

```
        ''=~(       '(?{'       .(``        |'&')       .('[        ^'-')
        .(``        |'!')       .(``        |',')       .'"'.       '\\$'
        .'=='       .('[        ^'+')       .(``        |'/')       .('[
        ^'+')       .'||'       .(';'       &'=')       .(';'       &'=')
        .';-'       .'-'.       '\\$'       .'=;'       .('[        ^'(')
        .('[        ^'.')       .(``        |'"')       .('!'       ^'+')
       .'_\\{'      .'(\\$'     .';=('.     '\\$=|'     ."\|".(     ``'^'.'
     ).((``')|    '/').')'.    .'\\"'.+(    '{'^'[').    (``'|'"')    .(``'|'/'
   ).('[^^'/')  .('[^^'/').  (``'|',').(  ``'|('&')).  '\\".\\"'.(  '[^^('(')).
   '\\"'.('[^   '#').'!!--'  .'\\$=.\\"'  .('{'^'[').  (``'|'/').(  ``'|"\&").(
   '{'^"\[").(  ``'|"\"").(  ``'|"\&").(  ``'|"\$").(  '[^^('(').  '\\").\\"'.
   ('{'^'[').(  ``'|"\/").(  ``'|"\.").(  '{'^"\[").(  '[^^\/").(  ``'|"\(").(
   ``'|"\&").(  '('^"\[").(  '[^"\,").(  ``'|"\!").(  ``'|"\,").(  ``'|(',')).
   '\\"\\}'.+(  '[^"\+").(  '[^"\}").(  ``'|"\}").(  ``'|"\.").(  '[^('/')).
   '+_,\\",'.(  '('^('[')).  ('\\$;!').(  '!'^"\+").(  '{'^"\/").(  ``'|"\!").(
   ``'|"\+").(  ``'|"\&").(  '('^"\[").(  ``'|"\/").(  ``'|"\.").(  ``'|"\&").(
   '{'^"\[").(  ``'|"\$").(  ``'|"\/").(  '[^"\,").(  ``'|('.')).  ',',((('}'^
   '[').("\[^   '+').("\`"|  '!').("\[^   '(').("\[^   '(').("\{"^  '[').("\`"|
   ')').("\[^   '/').("\{"^  '[').("\`"|  '!').("\[^   ')').("\`"|  '/').("\[^
   '.').("\`"|  '.').("\`"|  '$').".\",.(  '!'^('+')).  '\\",_;\\"'  .'!'.("\!"^
   '+').("\!"^  '+').'\\"'.  ('['^',').(  ``'|"\(").(  ``'|"\)").(  ``'|"\,").(
   ``'|('&')).  '++\\$="})'  );$:=('.')^  '~';$~=~'@'|  '(';$^='}'^  '[';$/=``';
```

JavaScript

- Client-side HTML-embedded scripting language
  - Used to create dynamic HTML documents
  - Processing on the client side, rather than server
- Related to Java only through similar syntax
- Not a true object-oriented language: object-centered or object-based language
- Began at Netscape, later Netscape and Sun
- Purely interpreted by the browser
- Ancestor of ActionScript (Flash programming language)
- Real name: ECMAScript (standard)

JavaScript

- JS is relatively low-level
- Subject to browser incompatibilities
- However, now supplemented with very high level standard libraries such as jQuery and Prototype
- AJAX (Asynchronous Javascript and XML) technology has become very popular over the last few years
- Complexity of Javascript apps has grown significantly
- Chrome browser (Google) has had significant impact on the maturity of Javascript

JavaScript example

```
/**
 * 99 Bottles of Beer on the Wall in JavaScript
 * This program prints out the lyrics of an old pub song.
 * Copyright (C) 1996, Brian Patrick Lee (blee@media-lab.mit.edu)
 */
if (confirm("Are you old enough to read about beer\n" +
        "according to your local community standards?")) {
  for (i = 99 ; i > 0 ; i--) {
    j = i - 1;
    if (i != 1) {
      icase = "bottles";
    } else {
      icase = "bottle";
    }
    if (j != 1) {
      jcase = "bottles";
    } else {
      jcase = "bottle";
    }
    document.writeln(i + " " + icase + " of beer on the wall,");
    document.writeln(i + " " + icase + " of beer,");
    document.writeln("Take 1 down, pass it around,");
    if (j != 0) {
      document.writeln(j + " " + jcase + " of beer on the wall.");
    } else {
      document.writeln("No more bottles of beer on the wall!");
    }
    document.writeln()
  }
} else {
  document.write("You might want think about moving to another community.")
}
```

PHP

- Rasmus Lerdorf: Personal Home Page
- Now just called PHP, or Hypertext Preprocessor
- Interpreted, sever-side, HTML-embedded scripting language
- Requires web server support (as do other server-side languages)
- Often used for form processing, DB access
- Features: dynamic strings, associative dynamic arrays, free use of type coercions
- Support for OOP added with second release
- Extensive support for form processing, back-end databases
- Open source
- Huge number of libraries available

PHP example

```
<table>
  <tr>
  <?php
  $menu["Home"] = "$root";
  $menu["Announcements"] = "$root/announcements";
  $menu["People"] = "$root/personnel";
  $menu["Projects"] = "$root/projects";
  $menu["Publications"] = "$root/pubs";
  $menu["AI"] = "$root/AI";
  $menu["Software"] = "$root/software";
  $menu["Contact"] = "$root/contact";
  $menu["Private"] = "$root/internal";
  $menu["CS"] = "http://www.umcs.maine.edu";
  $menu["CIS"] = "http://www.umaine.edu/cis";
  $menu["UMaine"] = "http://www.umaine.edu";
  foreach ($menu as $name => $link) {
    print("    <td><a href=\"$link\">$name</a></td>\n");
    }
    ?>
  </tr>
</table>
```


Markup/Programming Hybrid Languages

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSLT) transforms XML documents for display
  - Programming constructs (e.g., looping) — Turing complete
- JSP
  - Java Server Pages: a collection of technologies to support dynamic Web documents
  - servlet: a Java program that resides on a Web server and is enacted when called by a requested HTML document; a servlet's output is displayed by the browser
  - JSTL includes programming constructs in the form of HTML elements
- ASP and ASP.NET
  - Active Server Pages
  - Similar to JSP. .NET elements look like HTML but are interpreted server side and rendered in HTML
  - Any .NET language can be used for programming