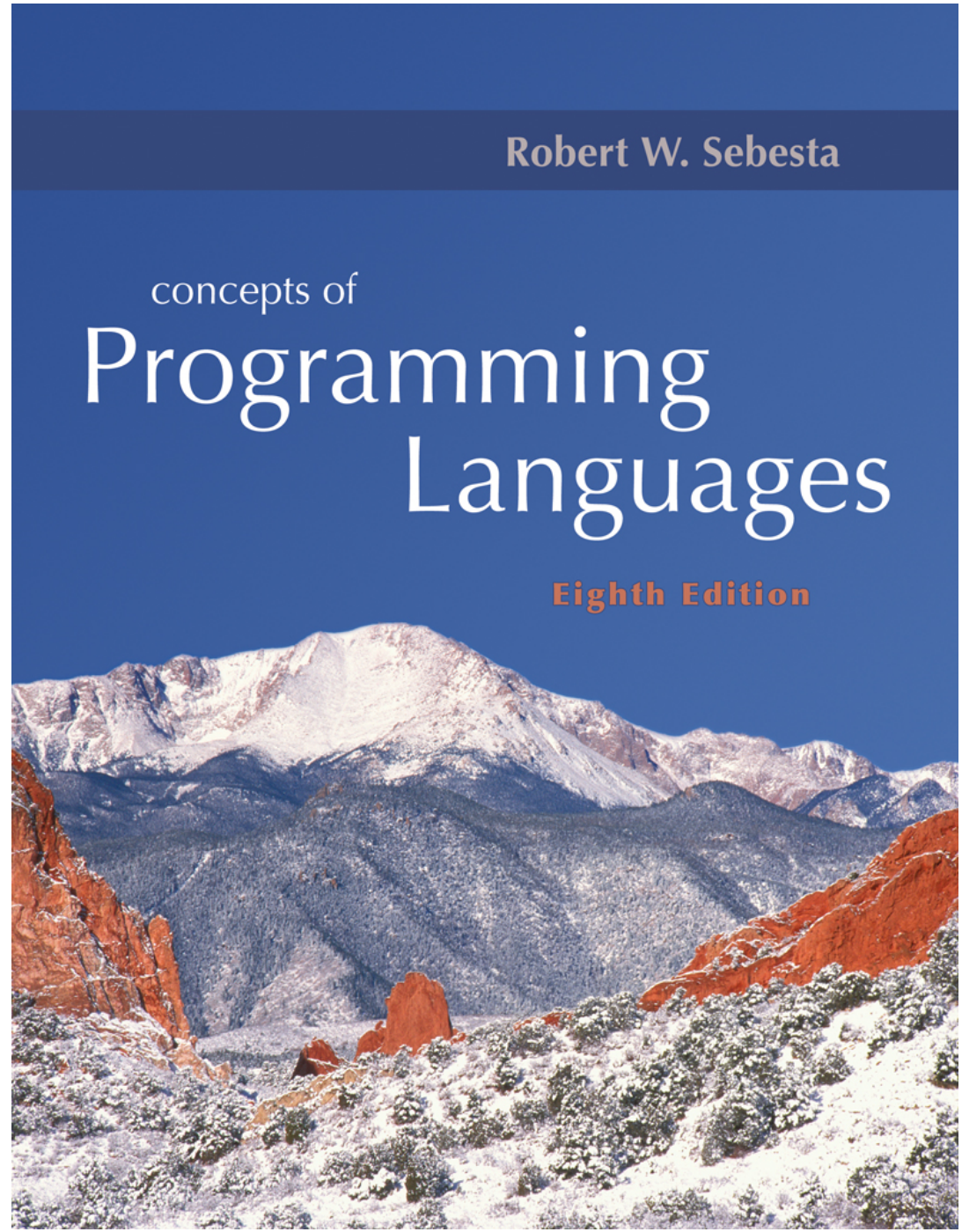
Chapter 2

Evolution of the Major Programming Languages



ISBN 0-321-49362-1

Chapter 2 Topics

- Zuse's Plankalkül
- Minimal Hardware Programming: Pseudocodes
- The IBM 704 and Fortran
- Functional Programming: LISP
- The First Step Toward Sophistication: ALGOL 60
- Computerizing Business Records: COBOL
- The Beginnings of Timesharing: BASIC

Chapter 2 Topics (continued)

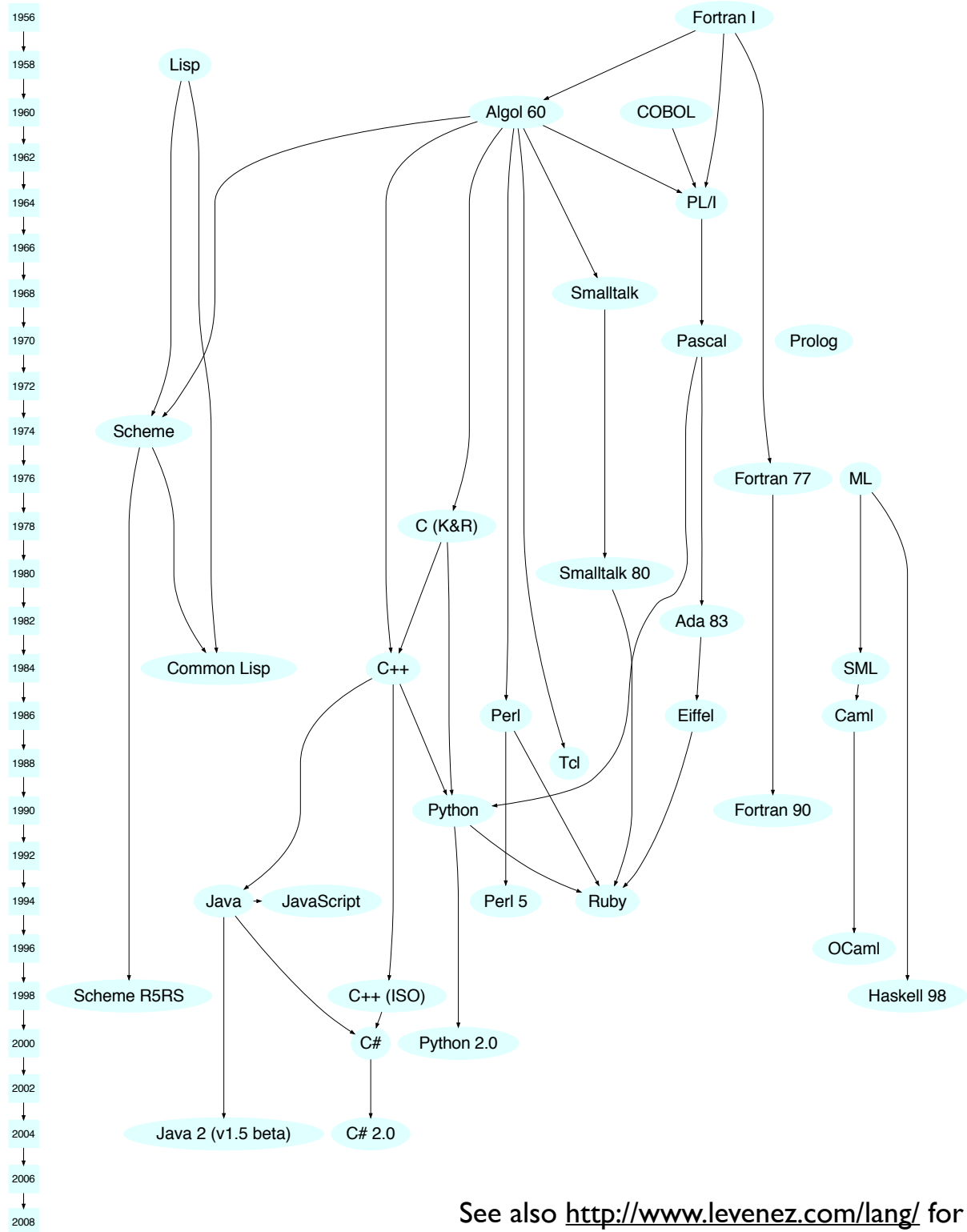
- Everything for Everybody: PL/I
- Two Early Dynamic Languages: APL and SNOBOL
- The Beginnings of Data Abstraction: SIMULA 67
- Orthogonal Design: ALGOL 68
- Some Early Descendants of the ALGOLs
- Programming Based on Logic: Prolog
- History's Largest Design Effort: Ada

Chapter 2 Topics (continued)

- Object-Oriented Programming: Smalltalk
- Combining Imperative and Object-Oriented Features: C++
- An Imperative-Based Object-Oriented Language: Java
- Scripting Languages: JavaScript, PHP, Python, and Ruby
- A C-Based Language for the New Millennium: C#
- Markup/Programming Hybrid Languages

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
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants

Plankalkül Syntax

- 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

- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point

Pseudocodes: Short Code

- Short Code developed by Mauchly in 1949 for BINAC computers
  - Expressions were coded, left to right
  - Example of operations:

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: Speedcoding

- Speedcoding developed by Backus in 1954 for IBM 701
  - Pseudo operations for arithmetic and math functions
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Slow!
  - 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

IBM 704 and Fortran

- Fortran 0: 1954 – not implemented
- 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, because there was no place to hide the cost of interpretation (no floating-point software)
  - Environment of development
    - Computers were small and unreliable
    - Applications were scientific
    - No programming methodology or tools
    - Machine efficiency was the most important concern

Source: Columbia University

Design Process of Fortran

- Impact of environment on design of Fortran I
  - No need for dynamic storage
  - Need good array handling and counting loops
  - No string handling, decimal arithmetic, or powerful input/output (for business software)

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

Fortran I Overview (continued)

- First implemented version of FORTRAN
  - No separate compilation
  - 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 II

- Distributed in 1958
  - Independent compilation
  - Fixed the bugs

Fortran IV

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

Fortran 77

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

Fortran 90

- Most significant changes 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 – ditto

Fortran Evaluation

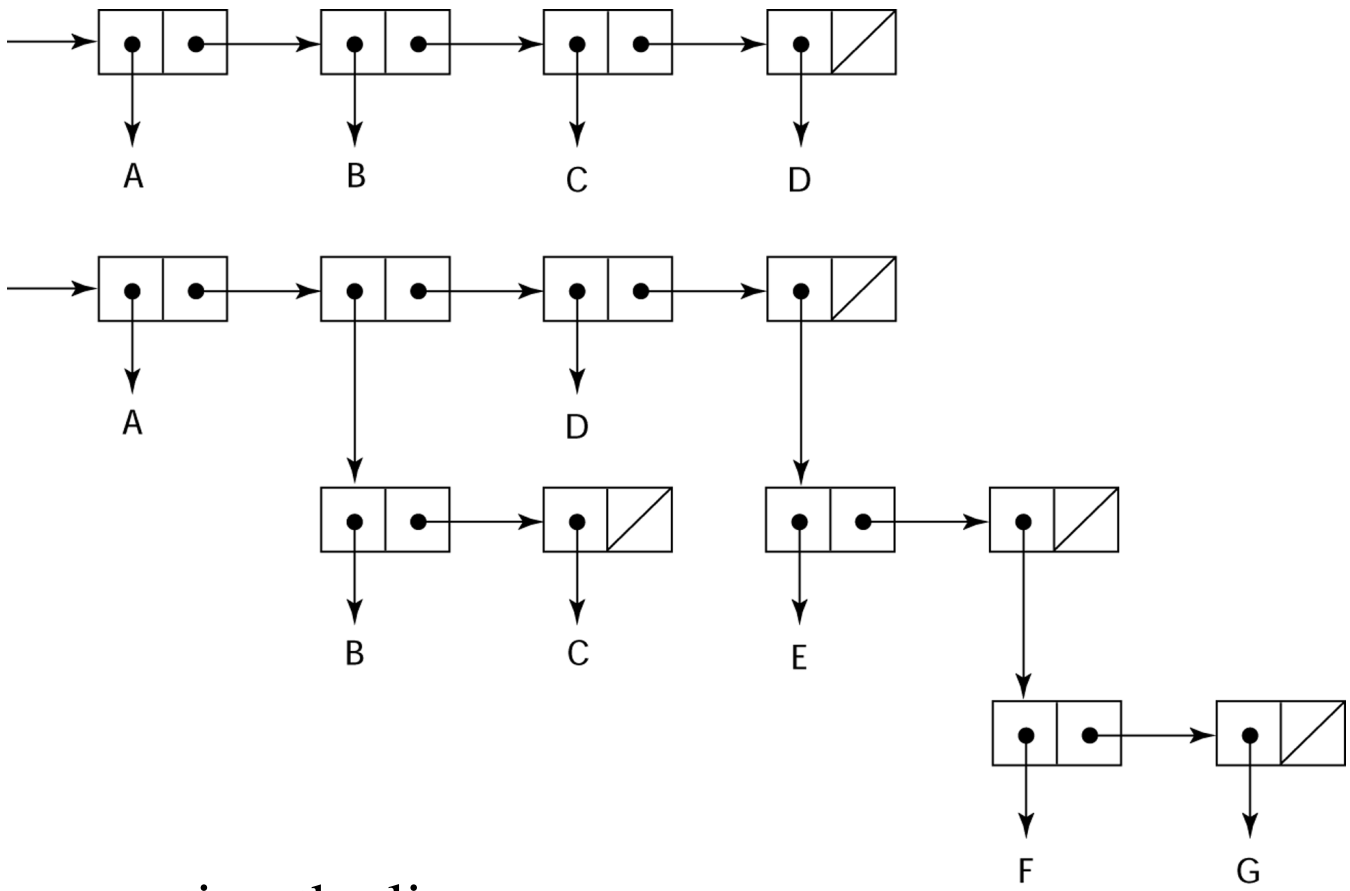
- Highly optimizing compilers (all versions before 90)
  - Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used
- Characterized as the lingua franca of the computing world

```
program main

!*****************************************************************************80
!
!! MAIN is the main program for ARGS.
!
!  Discussion:
!    ARGS demonstrates the use of the (semi-standard) GETARGS utility.
!
!  Modified:
!    04 September 2002
!  Author:
!    John Burkardt
!  Usage:
!    args arg1 arg2 arg3 ...
!
  implicit none

  character ( len = 80 ) arg
  integer i
  integer iargc
  integer numarg

  call timestamp ( )
  write ( *, '(a)' ) ' '
  write ( *, '(a)' ) 'ARGS'
  write ( *, '(a)' ) '  FORTRAN90 version'
  write ( *, '(a)' ) ' '
  write ( *, '(a)' ) '  Demonstrate the use of command line argument'
  write ( *, '(a)' ) '  routines in a FORTRAN program.'
  write ( *, '(a)' ) '  These include GETARG and IARGC.'

  numarg = iargc ( )

  write ( *, '(a)' ) ' '
  write ( *, '(a,i6,a)' ) &
    '  ARGS was called with IARGC() = ', numarg, ' arguments.'

  write ( *, '(a)' ) ' '
  write ( *, '(a)' ) '  CALL GETARG(I,ARG) returns the arguments:'
  write ( *, '(a)' ) ' '
  write ( *, '(a)' ) '  I     ARG '
  write ( *, '(a)' ) ' '

  do i = 0, numarg
    call getarg ( i, arg )
    write ( *, '(2x,i3,2x,a20)' ) i, arg
  end do

  write ( *, '(a)' ) ' '
  write ( *, '(a)' ) 'ARGS:'
  write ( *, '(a)' ) '  Normal end of execution.'
  write ( *, '(a)' ) ' '
  call timestamp ( )

  stop
end
```

Functional Programming: LISP

- LISt Processing language
  - Designed at MIT by McCarthy
- 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
- Syntax is based on lambda calculus

Representation of Two LISP Lists



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

LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- COMMON LISP and Scheme are contemporary dialects of LISP
- ML, Miranda, and Haskell are related languages

Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax (and small size) make it ideal for educational applications

COMMON LISP

- An effort to combine features of several dialects of LISP into a single language
- Large, complex

The First Step Toward Sophistication: ALGOL 60

- Environment of development
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable language; all were machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)

- Goals of the language
  - Syntax close to mathematical notation
  - Good for describing algorithms in publications
  - Must be mechanically translatable into machine code

ALGOL 58

- 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 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods: pass by value and pass by name
  - Subprogram recursion
  - Stack-dynamic arrays

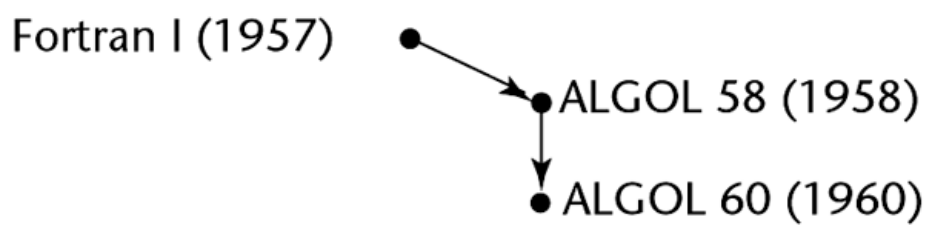
  - Still no I/O and no string handling

ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
  - First machine-independent language
  - First language whose syntax was formally defined (BNF)

Figure 2.3

Genealogy of ALGOL 60



ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible--hard to implement
    - Entrenchment of Fortran
    - Formal syntax description
    - Lack of support from IBM

```
// the main program (this is a comment)
```

```
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
```

Source: http://www.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html, October 2008

Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC was beginning to use FLOW-MATIC
  - USAF was beginning to use AIMACO
  - IBM was developing COMTRAN

COBOL Historical Background

- Based on FLOW-MATIC
- FLOW-MATIC features
  - 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

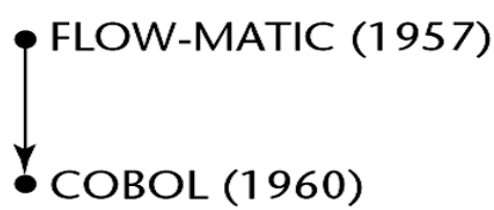
- First Design Meeting (Pentagon) – May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must broaden the base of computer users
  - Must not be biased by current compiler problems
- Design committee members were all from computer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

COBOL Evaluation

- Contributions
  - First macro facility in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

Figure 2.4

Genealogy of COBOL



COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

The Beginning of Timesharing: BASIC

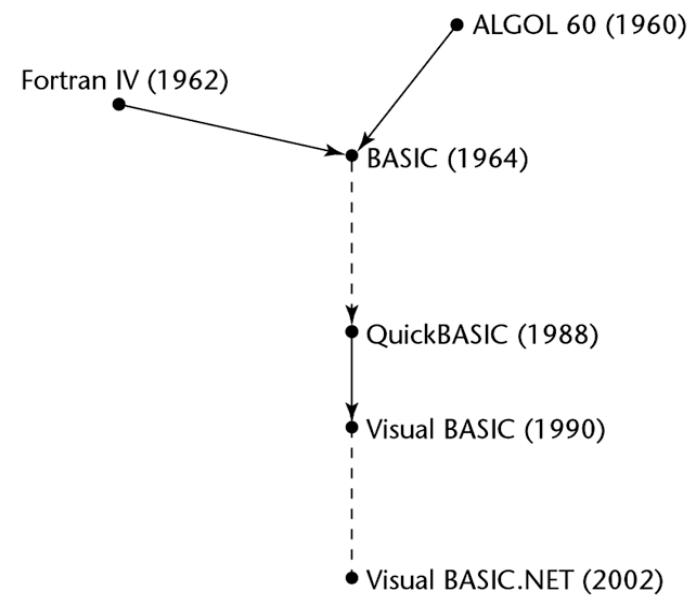
- Designed by Kemeny & Kurtz at Dartmouth
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be “pleasant and friendly”
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Current popular dialect: Visual BASIC
- First widely used language with time sharing

BASIC Evaluation

- First widely used language that used terminals
- Design largely from FORTRAN, some from ALGOL
- BASIC ANSI standard in 1978, minimal
- Criticized for poor structure of programs
- Readability and reliability
- Resurgence by Visual Basic
  - GUI
  - VB .NET

Figure 2.5

Genealogy of BASIC



```
CLS
x = INT(RND * 10) + 1
PRINT "I am thinking of a number between 1 and 10, can you guess it?"
PRINT "You have 3 chances"
INPUT "What is it, Chance 1"; i%
IF i% = x GOTO win
IF i% x THEN PRINT "Guess lower!"
INPUT "Chance 2"; i%
IF i% = x GOTO win
IF i% x THEN PRINT "Guess lower!"
INPUT "Chance 3, last chance"; i%
IF i% = x GOTO win
IF i% x THEN GOTO loose
win: CLS
COLOR 9
PRINT "Congradulations!!! You guessed right"
END

loose: CLS
PRINT "You lost!"
PRINT
PRINT "The correct number was "; x
```

2.8 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, like COBOL had; business users began to need floating point and arrays for MIS
  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly
- The obvious solution
  - Build a new computer to do both kinds of applications
  - Design a new language to do both kinds of applications

PL/I: Design Process

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

PL/I: Language overview

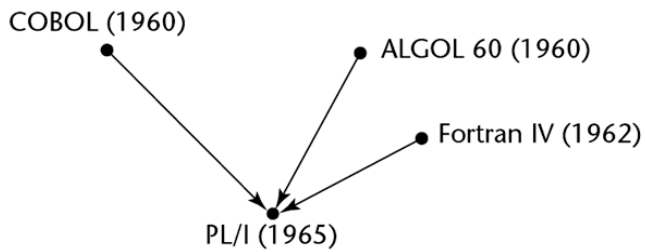
- Included best parts of
  - ALGOL 60: recursion and block structure
  - Fortran IV: separate compilation, communication through global data
  - COBOL 60: data structures, i/o, report generation

PL/I: Evaluation

- PL/I contributions
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

Figure 2.6

Genealogy of PL/I



```
SHELL:   PROCEDURE OPTIONS (MAIN);
         DECLARE
           ARRAY(50) FIXED BIN(15),
           (K,N) FIXED BIN(15);

         GET LIST(N);
         GET EDIT((ARRAY(K) DO K = 1 TO N));
         PUT EDIT((ARRAY(K) DO K = 1 TO N));
         CALL BUBBLE(ARRAY,N);

END BUBBLE;

BUBBLE:   PROCEDURE(ARRAY,N); /* BUBBLE SORT*/
          DECLARE (I,J) FIXED BIN(15);
          DECLARE S BIT(1);        /* SWITCH */
          DECLARE Y FIXED BIN(15); /* TEMPO */
          DO I = N-1 BY -1 TO 1;
            S = '1'B;
            DO J = 1 TO I;
              IF X(J)>X(J+1) THEN DO;
                S = '0'B;
                Y = X(J);
                X(J) = X(J+1);
                X(J+1) = Y;
                END;
              END;
            IF S THEN RETURN;
            END;
          RETURN;
          END SRT;
```

Source: http://www.engin.umd.umich.edu/CIS/course.des/cis400/pl1/pl1bubble.html verified on Sep 9, 2008

Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation
- Variables are untyped
  - A variable acquires a type when it is assigned a value
- Storage is allocated to a variable when it is assigned a value

APL: A Programming Language

- Designed as a hardware description language at IBM by Ken Iverson around 1960
  - Highly expressive (many operators, for both scalars and arrays of various dimensions)
  - Programs are very difficult to read
- Still in use; minimal changes

SNOBOL

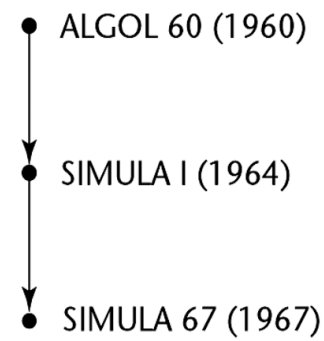
- Designed as a string manipulation language at Bell Labs by Farber, Griswold, and Polensky in 1964
- Powerful operators for string pattern matching
- Slower than alternative languages (and thus no longer used for writing editors)
- Still used for certain text processing tasks

The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for system simulation in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
  - Coroutines – a kind of subprogram
  - Classes, objects, and inheritance

Figure 2.7

Genealogy of SIMULA 67



Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms

ALGOL 68 Evaluation

- Contributions
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)

- Comments
  - Less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

```
// the main program (this is a comment)
```

```
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
```

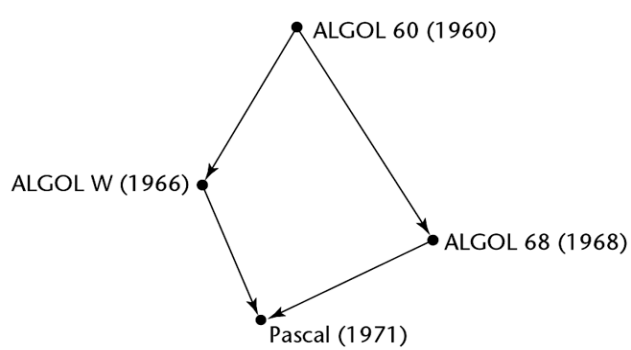
Source: http://www.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html

Pascal – 1971

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact was on teaching programming
  - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming
  - ,

Figure 2.9

Genealogy of Pascal



```
program temperature(output) ;

{ Program to convert temperatures from
 Fahrenheit to Celsius. }

const
    MIN = 32 ;
    MAX = 50 ;
    CONVERT = 5 / 9 ;

var
    fahren: integer ;
    celsius: real ;

begin
    writeln('Fahrenheit     Celsius') ;
    writeln('----------     -------') ;
    for fahren := MIN to MAX do begin
        celsius := CONVERT * (fahren - 32) ;
        writeln(fahren: 5, celsius: 18: 2) ;
 end ;
end.
```

Source: http://www.informatik.uni-hamburg.de/RZ/software/SUNWspro/pascal/user_guide/using_pascalug.doc.html, verified on Sep 11, 2008l

C – 1972

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP, B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Many areas of application

```
#include <stdio.h>

inline float convert(float f)  {
    return ((5.0/9.0) * (f - 32));
}

int main()  {
    float f;
    for(f = -40; f <= 220; f += 10) {
      printf("%f degrees fahrenheit = %f degrees celsius.\n", f, convert(f));
    }
}
```

Perl

- Related to ALGOL only through C
- A scripting language
  - A script (file) contains instructions to be executed
  - Other examples: sh, awk, tcl/tk
- Developed by Larry Wall
- Perl variables are statically typed and implicitly declared
  - Three distinctive namespaces, denoted by the first character of a variable's name
- Powerful but somewhat dangerous
- Widely used as a general purpose language and for CGI programming on the Web

```
$top_number = 100;
$x = 1;
$total = 0;
while ( $x <= $top_number ) {
   $total = $total + $x; # short form: $total += $x;
   $x += 1;     # do you follow this short form?
}

print "The total from 1 to $top_number is $total\n";
```

```
@flavors = ( "vanilla", "chocolate", "strawberry" );

for $flavor ( @flavors )  {
   print "We have $flavor milkshakes\n";
}

print "They are 2.95 each\n";
print "Please email your order for home delivery\n";
```

Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries
- Highly inefficient, small application areas

```
mother(joanne,jake).
father(vern,joanne).


grandparent(X,Z) :=
  parent(X,Y),
  parent(Y,Z).
```

```
Query: father(bob,darcie).
```

History's Largest Design Effort: Ada

- Huge design effort, involving hundreds of people, much money, and about eight years
  - Strawman requirements (April 1975)
  - Woodman requirements (August 1975)
  - Tinman requirements (1976)
  - Ironman equipments (1977)
  - Steelman requirements (1978)
- Named Ada after Augusta Ada Byron, the first programmer

Ada Evaluation

- Contributions
  - Packages – support for data abstraction
  - Exception handling – elaborate
  - Generic program units
  - Concurrency – through the tasking model
- Comments
  - Competitive design, no limit on participation
  - Included all that was then known about software engineering and language design
  - First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

```
1  -- main.adb:  main program for approximate string matching
2
3  with
4    Ada.Command_Line,      -- Access to external execution env (Ada95 A.15)
5    Ada.Text_IO,           -- Usual string oriented IO package
6    Approx;                -- User defined function
7  use Ada;
8
9  procedure main is
10    K : constant Natural := Integer'Value (Command_Line.Argument(1));
11    M : constant Boolean := Approx (K,
12       Command_Line.Argument (2), Command_Line.Argument (3));
13 begin
14    if M then
15       Text_IO.Put_Line ("Match.");
16    else
17       Text_IO.Put_Line ("No match.");
18    end if;
19 end main;
```

Source: http://www.cs.fit.edu/~ryan/ada/programs/access/main-adb.html, verified on Sep 11, 2008

Ada 95

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

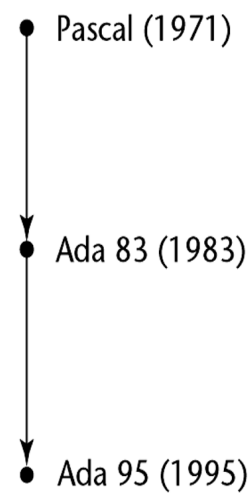


Figure 2.11

Genealogy of Ada

Object-Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)
- Pioneered the graphical user interface design
- Promoted OOP

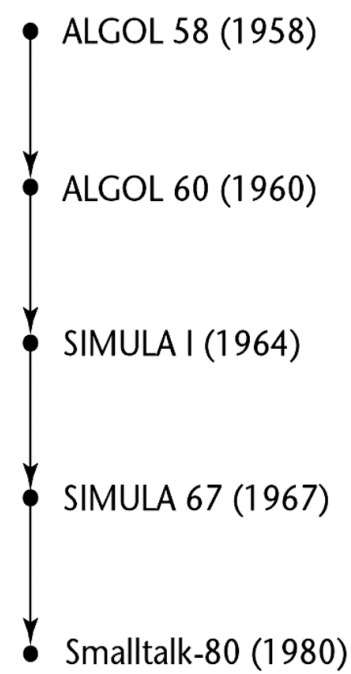
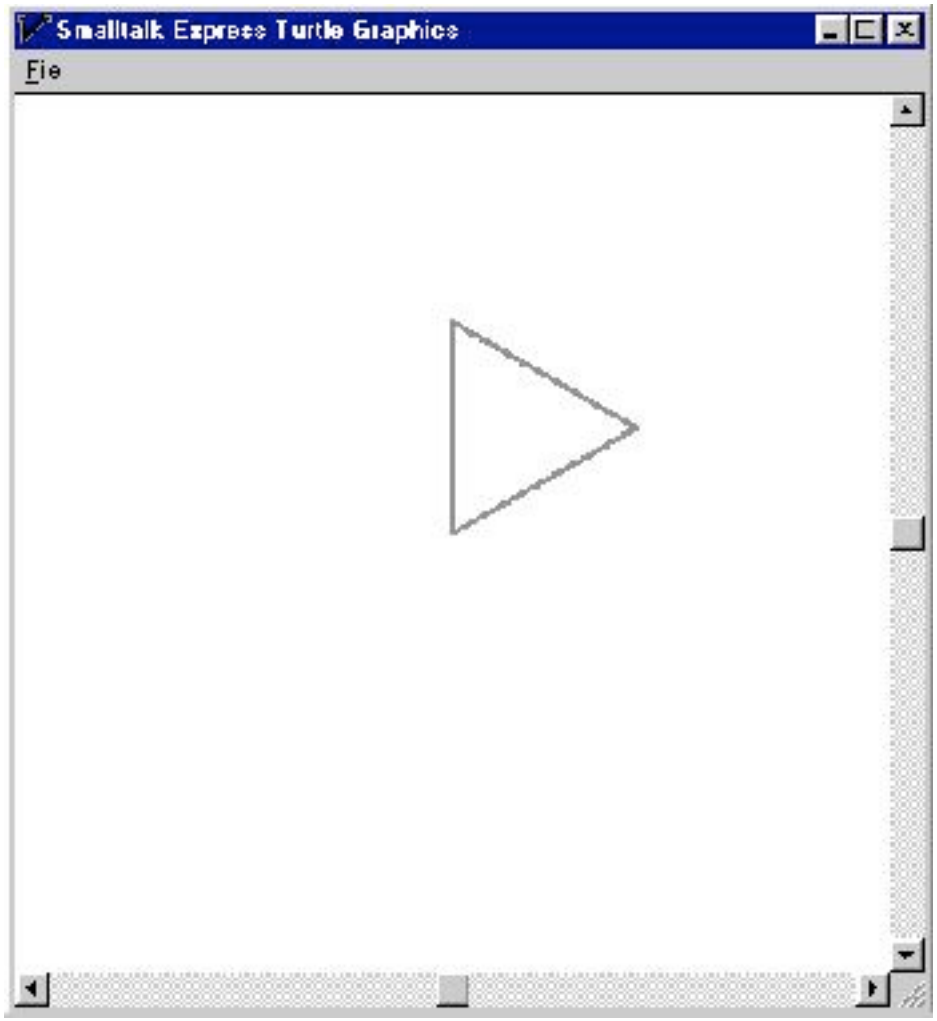


Figure 2.12

Genealogy of Smalltalk

```
Window turtleWindow: 'Turtle Graphics'.
     Turtle
         defaultNib: 2;
         foreColor: ClrDarkgray;
         home;
         go: 100;
         turn: 120;
         go: 100;
         turn: 120;
         go: 100;
         turn: 120
```

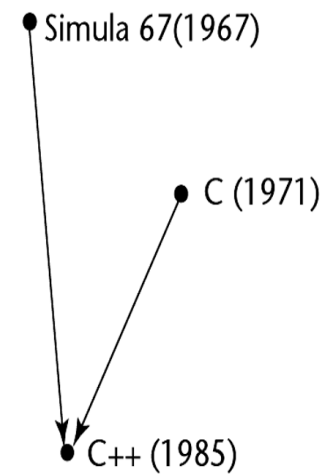


Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- Provides exception handling
- A 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 97
- Microsoft's version (released with .NET in 2002): Managed C++
  - delegates, interfaces, no multiple inheritance

Figure 2.13

The ancestry of C++



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++ because many C++ enthusiasts were already C programmers
- Delphi (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++

An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
- Based on C++
  - Significantly simplified (does not include **`struct`**, **`union`**, **`enum`**, pointer arithmetic, and half of the assignment coercions of C++)
  - Supports only OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency

Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 5.0, released in 2004

Figure 2.14

The ancestry of Java

Scripting Languages for the Web

- JavaScript
  - Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
  - A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
  - Purely interpreted
  - Related to Java only through similar syntax
- PHP
  - PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
  - A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
  - Purely interpreted
- Python
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Dynamically typed, but type checked
  - Supports lists, tuples, and hashes

```
/*   $Id: infolayer.js,v 1.4 2008/05/30 12:25:00 ilameter Exp $   */


var infodata=new Array();


function show_infolayer(e,newx,newy,player_id,y) {


   x = document.all ? window.event.x : e.pageX;
   y = document.all ? window.event.y : e.pageY;

   if(newy==0) {
      newy=y-45;
      if(newy<5) {
         newy=5;
      } else if(newy>350) {
         newy=350;
      }
   }

   dd.elements.infolayer.moveTo(newx,newy);
   dd.elements.infolayer.show(true);

document.getElementById("infolayer_content").innerHTML="<b>Daten...<b>";
   sendInfoRequest(player_id);
}

function hide_infolayer() {
   dd.elements.infolayer.hide(true);
}

function sendInfoRequest(player_id) {
   if (!infodata[player_id]) {

   var req = Ajax.getTransport();
   req.onreadystatechange=function () {
      if(req.readyState == 4){
            infodata[player_id] = req.responseText;

document.getElementById("infolayer_content").innerHTML=infodata[player_id]
;
     }
   };
   req.open("GET", "infolayer.php?player_id="+player_id, true);
   req.setRequestHeader("Pragma", "no-cache");
   req.setRequestHeader("Cache-Control"," must-revalidate");
   req.setRequestHeader("If-Modified-Since",document.lastModified);
   req.send(null);
   }
   else {

document.getElementById("infolayer_content").innerHTML=infodata[player_id]
;
  }
}
```

Scripting Languages for the Web

- Ruby
  - Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
  - Began as a replacement for Perl and Python
  - A pure object-oriented scripting language
    - All data are objects
  - Most operators are implemented as methods, which can be redefined by user code
  - Purely interpreted

A C-Based Language for the New Millennium: C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- Provides a language for component-based software development
- All .NET languages (C#, Visual BASIC.NET, Managed C++, J#.NET, and Jscript.NET) use Common Type System (CTS), which provides a common class library

Markup/Programming Hybrid Languages

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
  - Programming constructs (e.g., looping)

- 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

Summary

- Development, development environment, and evaluation of a number of important programming languages
- Perspective into current issues in language design