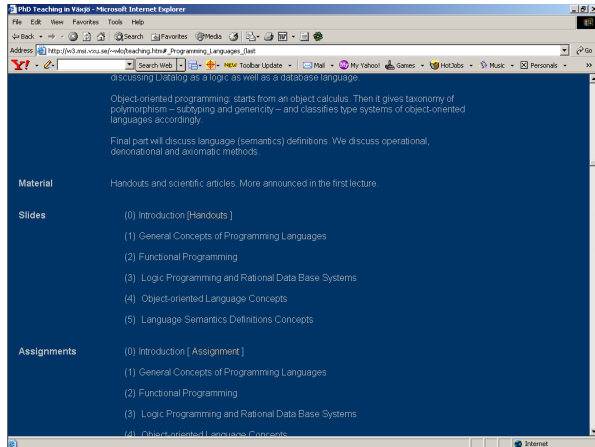
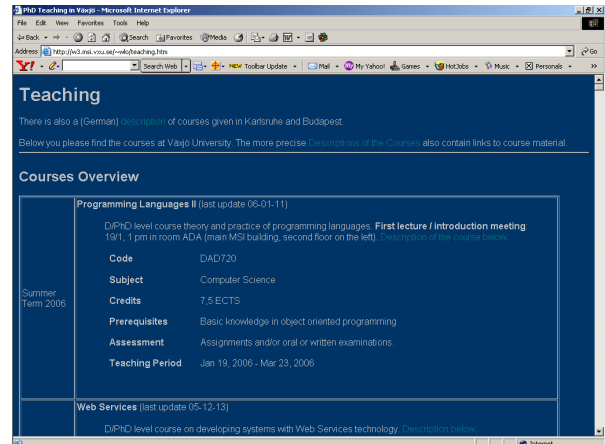


# Programming Languages



Welf Löwe  
MSI Building 3<sup>rd</sup> floor (left)  
Tel. 0047 - 70 8495  
Welf.Lowe@msi.vxu.se  
<http://www.msi.vxu.se/~wlo/teaching.htm>



## Goals of the Course

- Concepts of programming languages
  - Understand them, understand implementation variants
  - Recognize them in individual concrete languages
- Overview of paradigms
  - Functional, logic, (imperative) object-oriented
  - Ideas, advantages, disadvantages
  - When use which paradigm?
- Definition of languages
  - Syntax definition (briefly)
  - Static semantics (typing, consistency rules)
  - Dynamic semantic definitions
- **No** programming lesson.

## Course Structure

1. Basic concepts of programming languages
  - Paradigms
  - Syntax, Semantics, Pragmatics
  - Binding, lifetime and scope of definitions
  - types and typing, abstract data types
2. Paradigms
  - Functional programming ( $\lambda$ -calculus, LISP)
  - Logic programming (Prolog, Datalog)
  - Imperative object-oriented programming (Java, Small Talk, Cecil, Sather, Eiffel, CLOS and C++ compared)
3. Specification of programming languages
  - Operational semantics
  - Denotational semantics
  - Axiom semantics

## Course Material (3 classes)

1. General textbooks on programming languages and programming paradigms
  - Similar goals as this course
2. Reference Manuals & Standards
  - To introduce a language
  - Short and precise
  - No easy reading
  - For tool (e.g. compiler) constructors
3. Textbooks teaching single languages
  - Undergraduate courses
  - Easy reading

## General Programming Languages textbooks

- R. W. Sebesta: *Concepts of Programming Languages: Addison-Wesley (7th Edition) 2005*
- J. C. Reynolds: *Theories of Programming Languages*, Cambridge University Press, 1998
- C. Ghezzi, M. Jazayeri: *Programming Language Concepts*, John Wiley & Sons, 1998
- R. Sethi: *Programming Languages – Concepts and Constructs*, Addison-Wesley 1996.
- J. C. Mitchell: *Foundations of Programming Languages*, MIT Press, 1996
- R. Cezzar: *A Guide to Programming Languages*, Artech House, 1995.
- K.C. Louden: *Programming Languages: Principles and Practice*, PWS Publ. Comp. 1993.
- D. P. Friedman, M. Wand, C. T. Haynes: *Essentials of Programming Languages*, MIT-Press, 1992
- E. Horowitz: *Fundamentals of Programming Languages*. Computer Science Press 1984.

7

## History of Programming Languages

- E. Horowitz (Ed.): *Programming Languages, A Grand Tour*, Springer 1983.
- R. L. Wexelblat: *History of Programming Languages*. Academic Press, 1981.
- J. E. Sammet: *Programming Languages: History and Fundamentals*. Prentice-Hall, 1969.
- M. Fogiel: *Handbook and guide for comparing and selecting computer languages BASIC, FORTRAN, PASCAL, COBOL, PL/1, APL, ALGOL-60, C*. Research and Education Association, 1985.

8

## Textbooks on Special Paradigms

- R. Bird, P.L. Wadler: *Introduction to Functional Programming*. Prentice Hall 1988.
- J.W. Lloyd: *Foundations of Logic Programming*, Springer 1984.
- B. Meyer: *Object-oriented Software Construction*, Prentice Hall 1988. (practical)
- M. Arbadi, L. Cardelli: *A Theory of Objects*, Springer 1996. (theoretical)

9

## Reference Manuals & Standards

- **Ada95** Reference Manual ANSI/ISO/IEC 8652:1995.
- **Cobol85** ANSI X3.23-1985, ISO 1989:1985
- **Fortran**:
  - Fortran77 ANSI X.3.9 - 1978
  - Fortran90 ISO/IEC 1539:1991 und ANSI X3.198-1992
  - Fortran95 ISO/IEC 1539:1997
  - Adams et. al.: *Fortran95 Handbook*, MIT-Press, 1997.
- **C/C++**
  - C ISO/IEC 9899:1990
  - C++ ISO/IEC 14882:1998
  - B. Stroustrup: *The C++ Programming Language*, 3rd Edition, Addison-Wesley, 1997.
- J. Goslin, B. Joy, G. Steele: *The Java™ Language Specification*, Addison-Wesley, 1996.
- **Further standards**: search <http://www.iso.org> for "programming languages"

10

## What is a programming language

- **What is a program:**
  - Definition of an algorithm:  
Function: input data → output data
  - Definition of reactive system:  
Function: Stream of input data → Stream of output data
- **What is a programming language:**
  - Notation to define programs in a form that is readable for machines and humans
- **Readable for machines:** formal (each programming language is a formal specification language)
- **Readable for humans:** abstracts from machine properties

11

## Basic Language Paradigms

- **Imperative (procedural, operational) Languages**
  - States
  - State transitions (Assignments)
- **Functional Languages**
  - nested function evaluation
  - functions as data (in theory)
  - (Applicative: link of imperative+functional)
- **Logical Languages**
  - Predicate logic formulas
  - Compute the model (Resolution, Unification)
- All are Turing complete (any algorithm can be defined)
- All can be mapped to imperative concepts (i.e. compiled to machine programs)
- Differences: suitability for solving certain problems, libraries and components, i.e. differences engineering systems

12

## Imperative Languages

- Basis: von-Neumann Model
- Variables: mapping names to memory cells
- Statements:
  - Assign changes value of memory cells (contain expressions)
  - Control structures (loops, conditions, calls and procedural abstraction) for defining sequences of assignments
- Execution of statement changes state
- State:
  - Values of variables
  - Procedures being executed (call stack)
  - Statement being executed next (program counter)

13

## Functional Languages

- Basis: mathematical functions
- Functions (side effect free expressions) also as parameters and results of other functions
- Variable definitions
  - Bind names to function definitions
  - Do not change during program execution
- Program is a set of variable definitions and expressions being executed
- There is actually **no state** in pure functional paradigm, but many functional languages are not quite consequent here.

14

## Logical Languages

- Basis: logic formulas
- Program is a set of predicate logics formulas (e.g. Prolog Horn clauses) and a query (predicate)
- If satisfiable a truth assignment to logic variables is the output
- Execution: logics calculus (resolution, unification)
- Pragmatic extension to these principles in languages like Prolog

15

## Orthogonal Classifications

- Goal of languages is to support software engineering
- Hence languages provide further concepts orthogonal to imperative, functional and logical paradigms:
  - Structured languages (e.g. Algol, Pascal)
    - Procedural abstraction
    - Blocks statically defining scopes
  - Modular languages (e.g. ADA, Modula)
    - Supports the concept of modules (Modular Abstraction)
    - Information hiding and scoping
    - Separation of implementation and interface
  - Object oriented languages (e.g. Java, C#)
    - Data abstraction (objects) encapsulating data and applicable operations
    - Inheritance and polymorphism
    - Often also modular concepts: a class can be understood as a module

16

## Example: TPK Algorithm

- The TPK algorithm reads in an array of 11 values, applies a function to each value, and then writes the result in reverse order. It serves just to illustrate some of the usual actions that programming languages must perform.
- It introduced in Luis Trabb Pardo and Donald Ervin Knuth: "The early development of programming languages." In Encyclopedia of Computer Science and Technology, Marcel Dekker, New York, 1977, pages 419-96.
- More on <http://www.cs.fit.edu/~ryan/compare/>

17

## TPK in Algol 68 (imperative, structured)

```
begin
  int i;
  [0:10] real a;
  proc (real) real
    f := (real t) : sqrt(abs(t)) + 5 * t ^ 3;
  for i from 0 to 10 by 1 do
    read(a[i])
  od;
  for i from 10 to 0 by -1 do
    y := f(a[i]);
    if y > 400 then write(i); write("too large");
    else write(i); write(y);
  fi
od
end
```

18

## TPK in Basic (imperative, unstructured)

```
10 REM BASIC PROGRAM FOR TPK ALGORITHM
20 DIM A(11)
30 FOR I = 1 TO 11
40 INPUT A(I)
50 NEXT I
60 FOR J = 1 TO 11
70 LET I = 11 - J
80 LET Y = SQR(ABS(A(I+1))) + 5 ** A(I+1)
90 IF Y > 400.0 THEN 120
100 PRINT I+1, Y
110 GO TO 130
120 PRINT I, "TOO LARGE"
130 NEXT J
140 STOP
150 END
```

19

## TPK in Fortran 77 (imperative, unstructured)

```
      DIMENSION A(11)
      READ A
2     DO 3,8,10 J=1, 11
3     I=11-J
      Y=SQRT(ABS(A(I+1)))+5*A(I+1)**3
      IF (400. >= Y) 8, 4
4     PRINT I, 999.
      GO TO 2
8     PRINT I, Y
10    STOP
```

20

## TPK in Java (imperative, object-oriented)

```
import java.util.Scanner; // Input class
import static java.lang.Math.*; // Standard math library
public class TPK {
    // f(x) = sqrt(|x|) + 5*x**3
    static double f (double x) {
        return sqrt (abs(x)) + 5.0 * pow(x,3.0);
    }
    public static void main (String args[]) {
        final double A[] = new double[11];
        final Scanner scan = new Scanner (System.in);
        // Read in the values of the array "A"; one per line
        for (int i=0; i<A.length; i++) {
            A[i] = scan.nextDouble (); }
        // In reverse order, apply "f" to each element of "A" and print
        for (int i=A.length-1; i>=0; i--) {
            final double y = f (A[i]);
            if (y > 400.0) System.out.format ("%d TOO LARGE%n", i);
            else System.out.format ("%d %f%n", i, y); }
    }
}
```

21

## TPK in Lisp (functional)

```
(DEFUN SQR (X) ...)
(DEFUN EXP (X T) ...)
(DEFUN ABS (X)
  (COND (GREATER X 0) (X)
        (TRUE) (SUB 0 X)))
(DEFUN F (T) (PLUS (SQR (ABS T)) (TIMES 5 (EXP T 3))))
(DEFUN READATA () (READINPUT 10))
(DEFUN READINPUT (I)
  (COND (GREATER I 0) (CONS (READ) READINPUT (MINUS I 0))
        (EQ I 0) (CONS READ NIL)))
(DEFUN WRITE (L)
  (COND ((EMPTY L) NIL)
        (TRUE) (CONS (PRINT (CAR X) (WRITE (CDR X))))))
(DEFUN PROG ()
  (WRITE (MAP (READATA) F)))
```

22

## History

- 42 Zuse's *Plankalkül*: Z4 machine, published only in 1972
- 40ies machine languages (sedecimal)
- 50ies machine languages (symbolic addressed) like auto code and Formula code: Expressions, Jumps, Loops, Parameterless Procedures
- 54 **Fortran**: Parameter procedures
- 58 **Lisp**: Functional programming with garbage collection, meta-programming  
**Algol-58**: programming in the small feels pretty much as today
- 60 **Cobol**: Records, Case, Separated I/O  
**Algol-60**: Naur Report, ACM Specification for algorithms
- 62 **APL**: applicative language, interactive programming, scripting
- 64 **Snobol**: text processing, patterns, search trees
- 67 **Simula**: object-oriented language

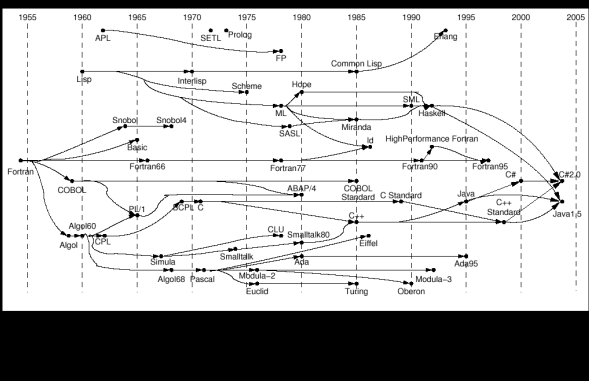
23

## History

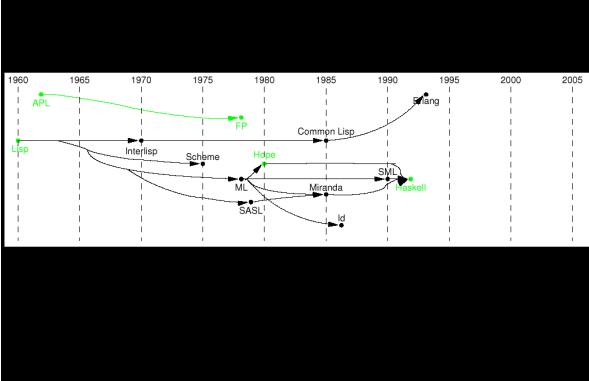
- 67 **Algol-W**: Heap, Pointers, User declared types
- 68 **Algol-68**: Simplification and Standardization
- 70 **CPL**, **BCPL**, **C**: Pointer arithmetic
- 71 **Pascal**: A clean simple Algol
- 72ff **Prolog**: SLD Resolution, Unification, Term rewriting
- 74 **Smalltalk**: everything is an object
- 75 **Modula**: notion of a module
- 76ff **Scheme**, **ML**: practical functional programming
- 80ff **SQL**, **ABAP4**: 4GL
- 83 **ADA**: Modular and distributed programming
- 85 **C++**: OO and C
- 95 **Java**: A clean simple C++
- 99 **C#**: Jet another Java

24

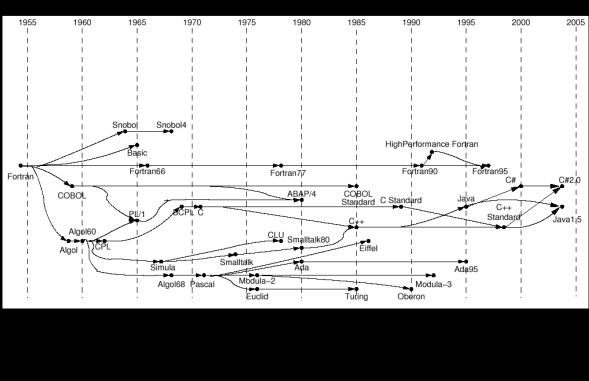
# History of Programming Languages



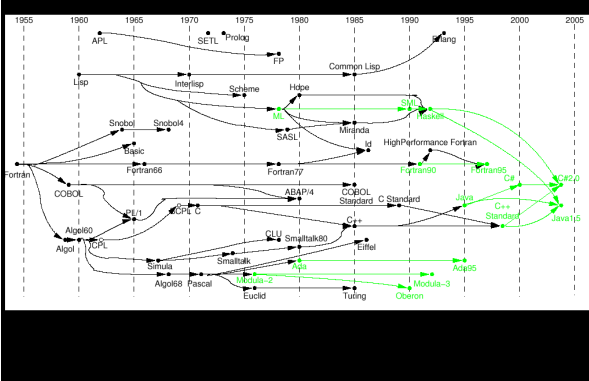
# History of Functional Languages



# History of Imperative Languages



# History of Modular Languages



# History of Object-Oriented Languages

