Principles of programming languages (2007) Lecture 1

http://few.vu.nl/~nsilvis/PPL/2007

Natalia Silvis-Cividjian
e-mail: nsilvis@few.vu.nl

vrije Universiteit *amsterdam*

Lecture 1. Topics

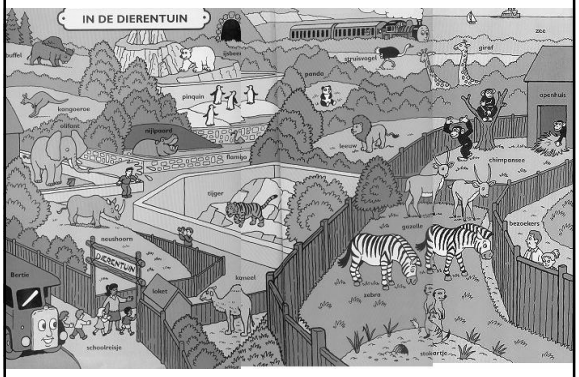
- Student survey
- Introduction
- History of major programming languages
- About this course

- **Which programming languages do you know?**
  - **Very well :**
  - **A little:**
  - **Just heard about:**
- **Which language is your favourite ?**
- **What languages you find interesting or you would like to learn?**
- **What is the first high-level programming language and when did it appear?**
- **When did Java appear and for what purposes?**
- **Can you give a definition of the following concepts? If yes give an example.**
  - **Anonymous function**
  - **Coercion**
  - **Polymorphism**
  - **Automatic garbage collection**
  - **Strong/weak type checking**
- **What is :**
  - **Python**
  - **Ruby**
  - **Eiffel**
  - **Self**
- **Explain to a layman what is OOP and why is it so important?**
- **What do you expect from this course?**

What makes programming languages such an interesting subject?

- The amazing variety
- The odd controversies
- The intriguing evolution

- **The amazing variety**



The amazing variety

There are very many, very different languages (ca 2500)

Often grouped into four families:

- Imperative
- Functional
- Logic
- Object-oriented

Imperative Languages

- Example: a factorial function in C

```
int fact(int n) {
  int sofar = 1;
  while (n>0) sofar *= n--;
  return sofar;
}
```

- Hallmarks :
  - Assignment
  - Iteration
  - Order of execution is critical

Visual languages

- Subcategory of imperative languages
- Visual Basic
- Once called **fourth-generation languages**

Object-Oriented Languages

Example: an Java object

```
public class MyInt {
  private int value;
  public MyInt(int value) {
    this.value = value;
  }
  public int getValue() {
    return value;
  }
  public MyInt getFact() {
    return new MyInt(fact(value));
  }
  private int fact(int n) {
    int sofar = 1;
    while (n > 1) sofar *= n--;
    return sofar;
  }
}
```

Object-Oriented Languages

- Hallmarks :
- Usually imperative, plus…
  - Constructs to help programmers use "objects"—little bundles of data that know how to do things to themselves

Functional Languages

- Example: a factorial function in ML

```
fun fact x =
  if x <= 0 then 1 else x * fact(x-1);
```

- Hallmarks :
  - No assignment, no side effects
  - Heavy use of recursion: no iterations

Another Functional Language

- Example: a factorial function in Lisp

```
(defun fact (x)
  (if (<= x 0) 1 (* x (fact (- x 1)))))
```

- Looks very different from ML
- But ML and Lisp are closely related

Logic Languages

- Example: a factorial function in Prolog

```
fact(X,1) :-
  X =:= 1.
fact(X,Fact) :-
  X > 1,
  NewX is X - 1,
  fact(NewX,NF),
  Fact is X * NF.
```

- Hallmark :
  - Program expressed as rules in formal logic

- The amazing variety
- **The odd controversies**

The Odd Controversies

- Programming languages are the subject of many heated debates:
  - Partisan arguments
  - Language standards
  - Fundamental definitions

The best programming language

- Java
- Fortran
- Cobol
- C
- C++
- PHP
- Javascript
- C#
- ML
- Prolog

?

No clear winner

Obviously, there is no best language for all situations. The best language might depend on many things:

- Type of program
- Reason the program is built
- Size of program
- Programmer familiarity
- Time available
- Cost
- Legacy

Language Partisans

- There is a lot of argument about the relative merits of different languages
- Every language has partisans, who praise it in extreme terms and defend it against all detractors
- To experience some of this, explore newsgroups: `comp.lang.*`

Evaluation Criteria

- Readability
- Writability
- Reliability

Evaluation criteria

Table 1.1 Language evaluation criteria and the characteristics that affect them.

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity/orthogonality	•	•	•
Control structures	•	•	•
Data types and structures	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Programming domains

- Scientific applications
- Business applications
- Artificial intelligence
- Scripting languages
- Systems programming
- Internet and Web

Language Standards

- The documents that define language standards are often drafted by international committees
- Can be a slow and complicated process
- Fortran ~~82~~ ~~8X~~ ~~88~~ 90 standard released in 1991

Basic Definitions

- Some terms refer to fuzzy concepts: all those language family names, for example
- No problem; just remember they are fuzzy
  - Bad: Is X really an *object-oriented* language?
  - Good: What aspects of X support an *object-oriented* style of programming?

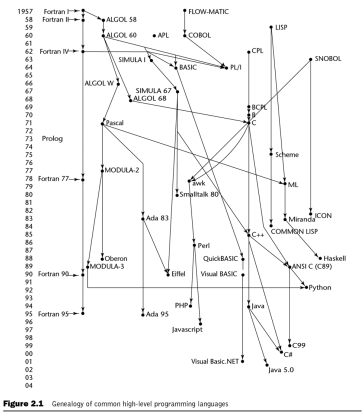
- The amazing variety
- The odd controversies
- **The intriguing evolution**

History of major programming languages

An updated genealogical diagram can be found at

http://www.levenez.com

Read Ch.2 Sebesta, ch. 24 Weber



[from Sebesta]

Figure 2.1 Genealogy of common high-level programming languages

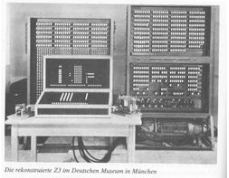
Plankalkül (1945)

Plankalkül **(Konrad Zuse)** = calculus for a computing plan

a high level language to express computations on computer Z4

Has:

- Machine independent operations, assignment statements,
- floating point data types, records, expressions with parenthesis,
- conditional statements but no else, repetition of statements , subroutines
- algorithms from array sorting to playing chess on 60 pages



Die rekonstruierte Z3 im Deutschen Museum in München

Plankalkül

| A + 1 => A

V |4 5

S |1.n 1.n

But…

- Difficult notation
- Z4 had a memory of 64 words of 32 bits each
- Never implemented
- Published vary late in 1972

Intermediate steps

- Machine code
  - Poor readability
  - Difficult to modify
  - No hardware with floating point arithmetic, no indexing
- Pseudocode

Fortran (1954)

- FORTRAN = mathematical **FOR**mula **TRAN**slating System – first compiled high level language
- IBM 704 system has floating point instructions in hardware
- Promised the efficiency of machine code and the ease of programming of pseudocodes. Almost succeeded. Code was very fast.
- Most of the calculations were numeric.
- Computers were more expensive than programmers, so no dynamic storage

```
C  FORTRAN PROGRAM TO FIND MEAN OF  N NUMBERS AND
C  NUMBER OF VALUES GREATER THAN THE MEAN


      DIMENSION A(99)
      REAL MEAN
      READ(1,5)N
    5 FORMAT(I2)
      READ(1,10) (A(I),I=1,N)
   10 FORMAT(6F10.5)
      SUM=0.0
   15 SUM=SUM+A(I)
      MEAN=SUM/FLOAT(N)
      NUMBER=0
      DO 20 I=1,N
      IF (A(I).LE.MEAN) GOTO 20
      NUMBER=NUMBER+1
   20  CONTINUE
      WRITE(2,25)  MEAN,NUMBER
   25  FORMAT(8H MEAN= ,F10.5,5X,20H NUMBER OVER MEAN = ,I5)
      STOP
      END
```

A Fortran program [from R. Clarck, Comparative programming Languages]

First step to sofistication: ALGOL 58 and ALGOL 60

Situation: Languages were developed around single architecture IBM or UNIVAC, communication was difficult.

No universal language

No portable language

In 1958 ACM (USA) + GAMM (EUR) came together to discuss the design of one international language – compromises about spheres of influence.

Goals:

Close to mathematical notations

Good for describing algorithms

Must be translatable to machine code

ALGOL 58 features

- Concept of type
- Names have any length
- Compound statements
- Semicolon as separator
- Assignment operator as :=
- Else-if clause
- But: abandoned by IBM.

ALGOL 60

- **New features:** block structure, pass by value and pass by name, subprogram recursion
- **Success:** standard way to publish algorithms for 20 years
  - All imperative languages are based on it.
  - First machine independent language
  - First language whose syntax was formally defined by BNF grammar
- **Failure:** Never widely used in USA, lack of support from IBM (Fortran compilers were faster), no I/O, formal syntax description

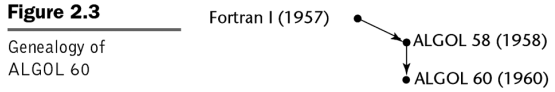


Figure 2.3

Genealogy of ALGOL 60

```
begin
        comment this program finds the mean of n numbers
        and the number of values greater than the mean ;

        integer n;

        read (n);
        begin
          real array a[1:n];
          integer i, number;
          real sum,mean;
          for i:=1 step 1 until n do
             read (a[i]);
          sum : =0.0;
          for i:=1 step 1 until n do
                    sum := sum + a[i];
          mean := sum / n ;
          number := 0 ;
          for i := 1 step 1 until n do
                    if a[i] > mean then
                    number := number + 1;
          write ("MEAN=",mean, "NUMBER OVER MEAN =", number)
        end
end
```

An ALGOL program [from R. Clarck, Comparative programming Languages]

COBOL (1959)

- First language required by DoD
- Must look like simple English (managers can read code).
- Still the most widely used business applications language.

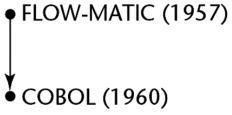
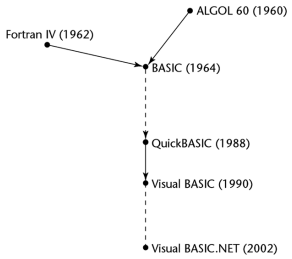


Figure 2.4

Genealogy of COBOL

Time sharing: BASIC (1964)

Figure 2.5
Genealogy of BASIC



BASIC

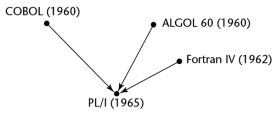
- Easy to learn and use by non-science (liberal arts) students
- Supposed to be a liberal arts programming language
- First widely used with time sharing (terminals instead of punch cards)

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

A BASIC program [from R. Clarck, Comparative programming Languages]

Everything for everybody: PL/I

Figure 2.6
Genealogy of PL/I



PL/I

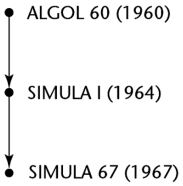
- Developed by IBM
- Built as a language for both kinds of applications: scientific computing and business
- First named Fortran VI, then NPL, PL/I
- Has:pointers, concurrency, recursivity, error handling
- Nowadays nearly dead language: too complicated

- "Using PL/I must be like flying a plane with 7,000 buttons, switches, and handles to manipulate in the cockpit". (Edsger Dijkstra)

Data abstraction: SIMULA 67

Figure 2.7

Genealogy of SIMULA 67



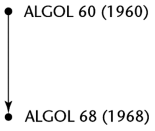
SIMULA 67

- Based on ALGOL 60 for system simulations (Norway)
- Contributions:
  - coroutines
  - A structure called class
  - Classes are base for data abstraction
  - Classes include data and functionality
  - Objects and inheritance

Orthogonal design: ALGOL68

Figure 2.8

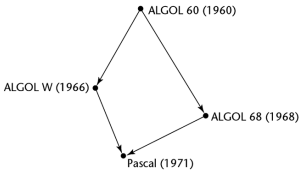
Genealogy of ALGOL 68



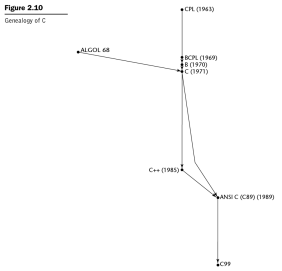
Descendants of ALGOL:Pascal

Figure 2.9

Genealogy of Pascal



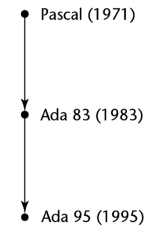
Descendants of ALGOL:C



Largest design effort: Ada 1983

Figure 2.11

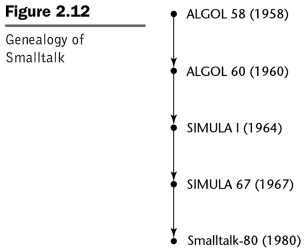
Genealogy of Ada



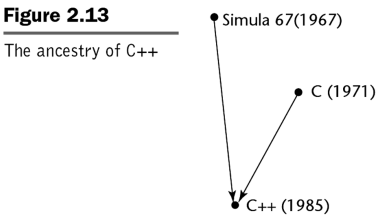
ADA

- Huge effort , much money, hundreds of people, DoD support (Honeywell/Bull)
- **Contributions:**
  - Packages for data abstraction
  - Exceptions handling
  - Generic programming units
  - Concurrency
- But: too large and complex, Compilers very difficult to build, the role of C++

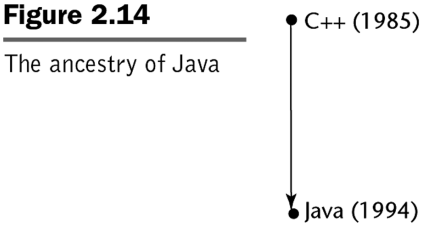
Object-oriented : Smalltalk 1980



Combining imperative with OO: C++



Imperative based OO language: Java



What makes a language successful?

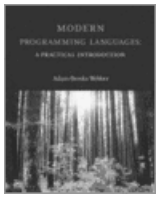
What makes a language successful?

- Expressive power
- Ease of use for a novice
- Ease of implementation
- Open source
- Excellent compilers
- Economics, patronage and inertia

About this course

Adam Brooks Weber,

Modern Programming Languages.



About this course

1. A study of the scripting language **Python** based on all the fundamental concepts discussed in Weber's book.

2. An on-going assignment: to explore and report on a less familiar language (see website for the details about this assignment).

About this course

Other recommended books:

R. Sebesta, Concepts of programming languages, 2005
M. Scott, Programming language pragmatics, 2005
R. Clarck, Comparative Programming Languages, 2001

Week	Date	Lecture topic
1	7 september	Introduction. History of programming languages evolution
2	14 September	Syntax and semantics
3	28 September	Language systems Functional programming: ML 1
4		Types Functional programming: ML 2
5		Polymorphism Functional programming: ML 3
6		Scope Memory management
7		Object oriented programming: Java 1
8		Logic programming: Prolog
9		Scripting languages: Python 1
10		Scripting languages: Python 2
11		Presentation session
12		Guests lectures

Aims

- To understand the general concepts underlying programming languages
- To experience the diversity of programming languages. Not all languages are like Java!
- To be aware of the design and implementation trade-offs
- To learn to make a fast evaluation of a language and easily learn a new language.
- To be able to argument your own language choice and understand the choice made by others.

Questions?