Programming Languages (CS 234319)

Prof. Ron Pinter

Dept. of CS, Technion

Spring 2007

Why Not Study Programming Languages?

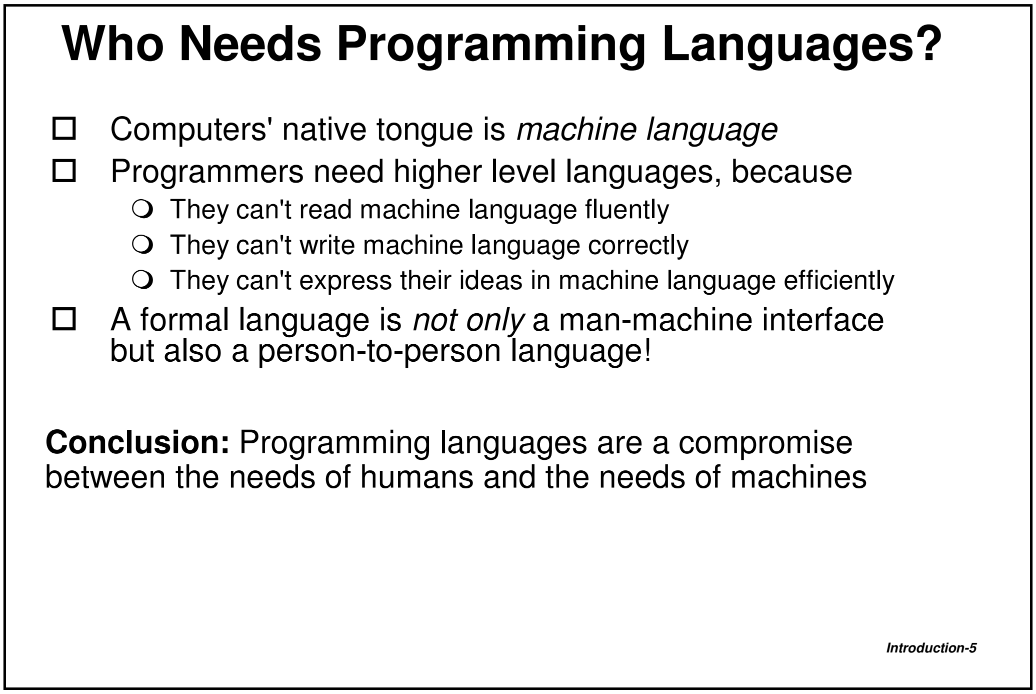
- □ The *expressive power* of all programming languages and computational devices is basically the same [the **Church-Turing Hypothesis**]:
  - ❍ The DOS batch language and Java are equivalent
  - ❍ The TRS-80 and the latest Giga Hz processors are equivalent
- □ **No** C++ or Java taught
- □ **No** theorems, proofs, lemmas, or integration by parts
- □ **No** easy grades
- □ It's at 8:30am…

Why Study Programming Languages?

- □ **Evaluation of a language’s features and usefulness**
  - ❍ Over 2,000 different languages out there
  - ❍ Common *concepts* and shared *paradigms*
  - ❍ Framework for comparative studies
- □ Because it’s fun
  - ❍ ML is neat
  - ❍ Prolog is elegant
  - ❍ There is more to it than Pascal, C, C++
  - ❍ Theoretical part will give you a new way of seeing things you thought you knew
- □ Will prove useful for other courses:
  - ❍ OOP
  - ❍ Compilation
  - ❍ Software Engineering
  - ❍ Semantics of Programming Languages

What is a Paradigm?

- □ ***par·a·digm*** (Merriam-Webster Collegiate Dictionary)
  - ❍ “a philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated”
  - ❍ Model, pattern
- □ **Thomas Kuhn** (1922-1996)
  - ❍ A set of “universally” recognized scientific achievements that for a time provide a model for a community practitioners.
- □ The Whorfian hypothesis: The language spoken influences the way reality is perceived.
- □ In programming languages: a family of languages with similar basic constructs and “mental model” of execution

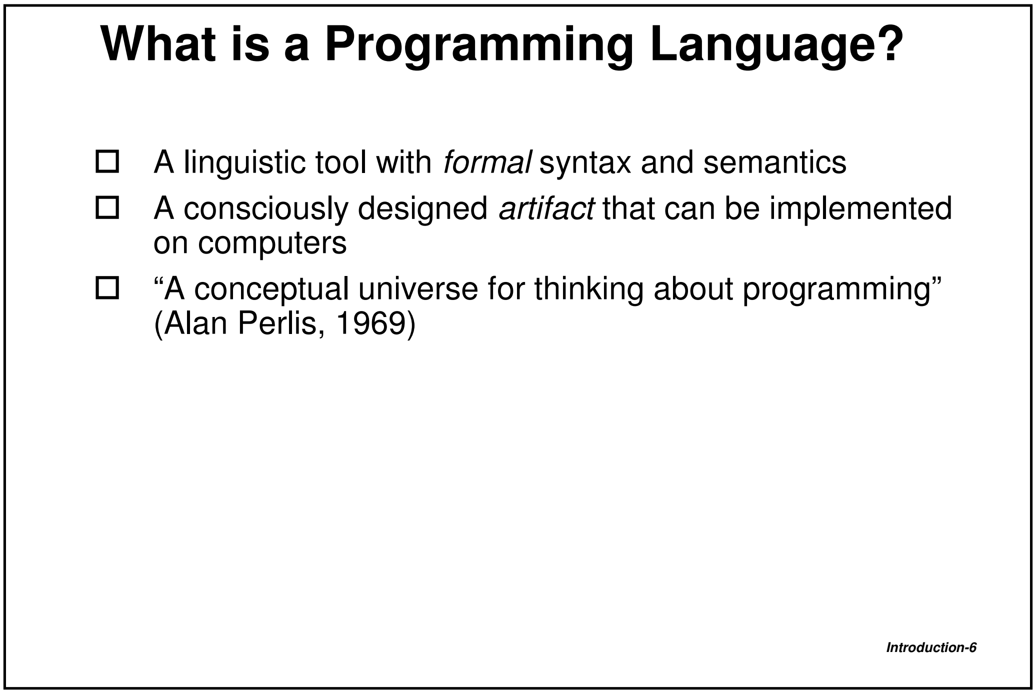


Who Needs Programming Languages?

- □ Computers' native tongue is *machine language*
- □ Programmers need higher level languages, because
  - ❍ They can't read machine language fluently
  - ❍ They can't write machine language correctly
  - ❍ They can't express their ideas in machine language efficiently
- □ A formal language is *not only* a man-machine interface but also a person-to-person language!

Conclusion: Programming languages are a compromise between the needs of humans and the needs of machines

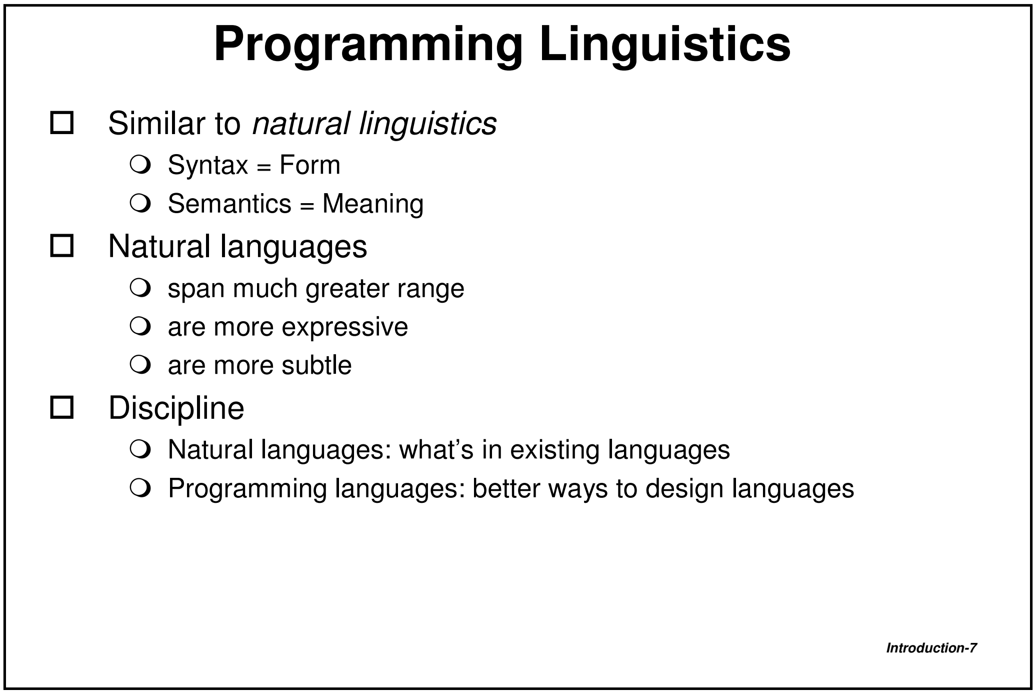
Introduction-5



What is a Programming Language?

- □ A linguistic tool with *formal* syntax and semantics
- □ A consciously designed *artifact* that can be implemented on computers
- □ “A conceptual universe for thinking about programming” (Alan Perlis, 1969)

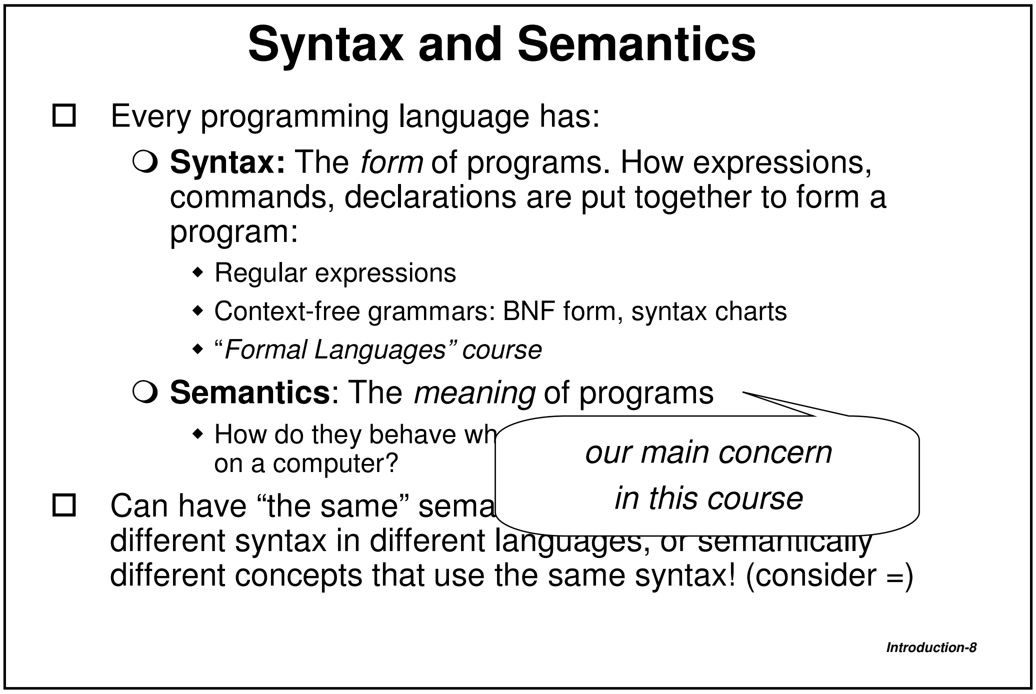
Introduction-6



Programming Linguistics

- Similar to *natural linguistics*
  - Syntax = Form
  - Semantics = Meaning
- Natural languages
  - span much greater range
  - are more expressive
  - are more subtle
- Discipline
  - Natural languages: what's in existing languages
  - Programming languages: better ways to design languages

Introduction-7



Syntax and Semantics

- Every programming language has:
  - **Syntax:** The *form* of programs. How expressions, commands, declarations are put together to form a program:
    - Regular expressions
    - Context-free grammars: BNF form, syntax charts
    - "*Formal Languages" course*
  - **Semantics**: The *meaning* of programs
    - How do they behave wh
      on a computer?
- Can have "the same" sema
  different syntax in different languages, or semantically different concepts that use the same syntax! (consider =)

our main concern
in this course

Introduction-8

Requirements from a Programming Language

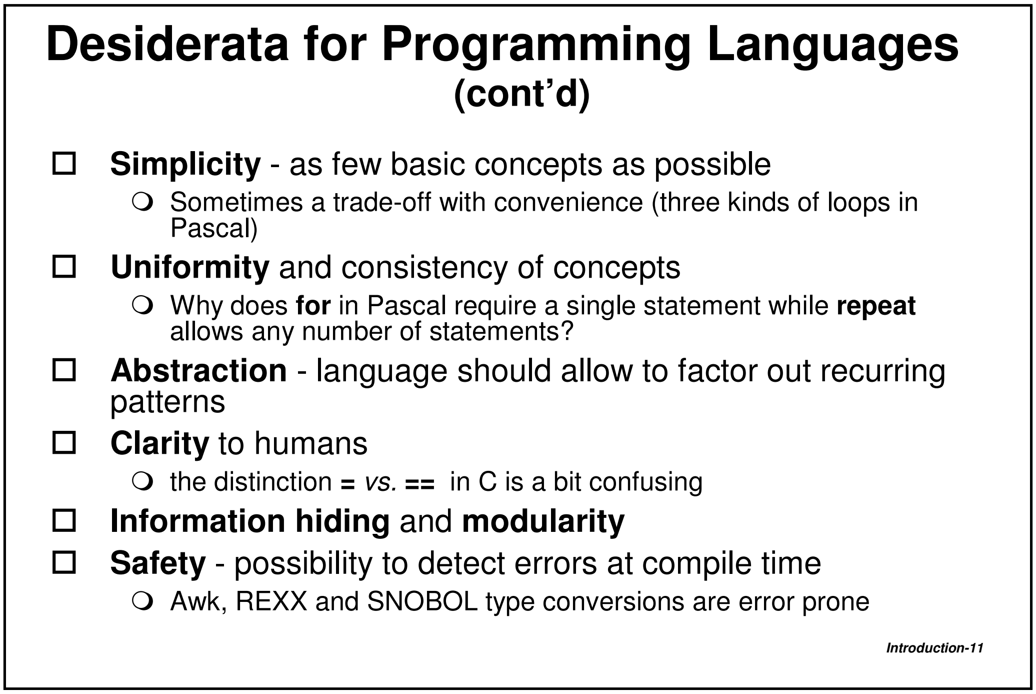
- **Universal:** every problem must have a solution
  - Express recursive functions
- **Natural:** application domain specific
  - Try writing a compiler in Cobol or a GUI in FORTRAN
- **Implementable:**
  - Neither mathematical notation
  - Nor natural language
- **Efficient:** open to debate
  - C and FORTRAN *vs.* SETL

Introduction-9

Desiderata for a Programming Language

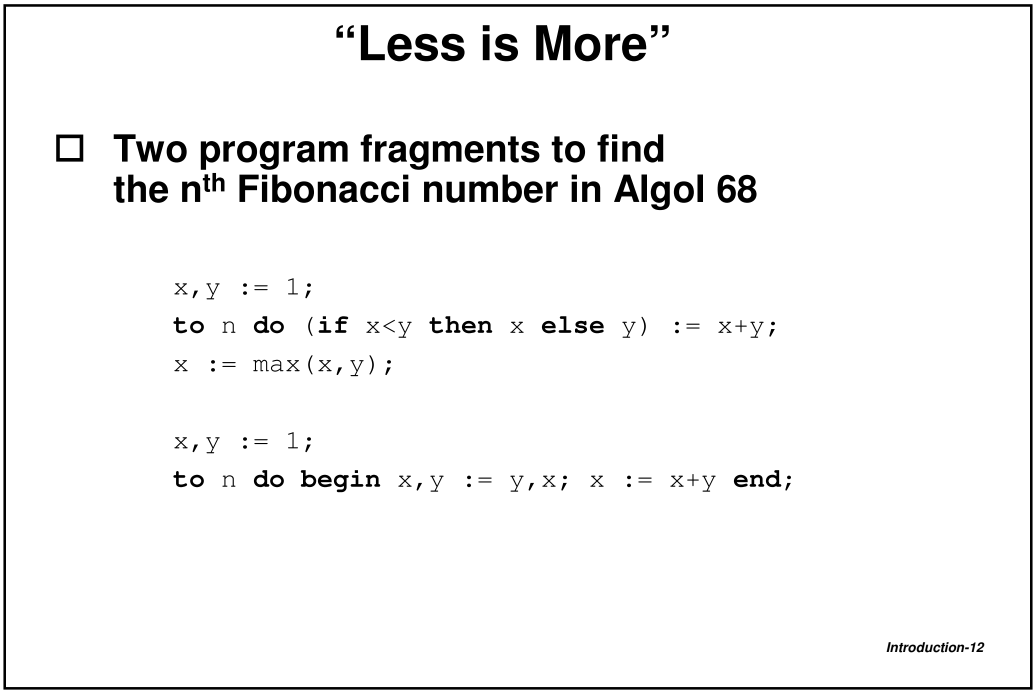
- **Expressiveness**
  - *Turing-completeness*
  - But also a practical kind of expressiveness: how easy is it to program simple concepts?
- **Efficiency**
  - Recursion in functional languages is expressive but sometimes inefficient
  - Is there an efficient way to implement the language (in machine code)?

Introduction-10



Desiderata for Programming Languages
(cont’d)

- □ **Simplicity** - as few basic concepts as possible
  - ❍ Sometimes a trade-off with convenience (three kinds of loops in Pascal)
- □ **Uniformity** and consistency of concepts
  - ❍ Why does **for** in Pascal require a single statement while **repeat** allows any number of statements?
- □ **Abstraction** - language should allow to factor out recurring patterns
- □ **Clarity** to humans
  - ❍ the distinction **=** *vs.* **==** in C is a bit confusing
- □ **Information hiding** and **modularity**
- □ **Safety** - possibility to detect errors at compile time
  - ❍ Awk, REXX and SNOBOL type conversions are error prone



“Less is More”

- □ **Two program fragments to find the nth Fibonacci number in Algol 68**

```
x,y := 1;
to n do (if x<y then x else y) := x+y;
x := max(x,y);

x,y := 1;
to n do begin x,y := y,x; x := x+y end;
```

Why Ranger 3 Missed the Moon?

An application program written in FORTRAN:

```
            DO 10 I=1.3
            …
         10 …
```

- □ Bad lexical definition – spaces are immaterial
- □ No declaration of variables (x2)
- □ Implicit typing (x2)
- □ Bad scoping rules
- □ Poor control structure specification
- □ Lack of diagnostics

What Characterizes a Programming Language

0. Formal Syntax and Semantics

1. Concepts

□ How it handles values	=> **values, types and expressions**
□ How it checks types	=> **typing systems**
□ Its entities for storing values	=> **storage**
□ Its means for storing values	=> **commands**
□ How it manages control	=> **sequencers**
□ How it attaches names to values	=> **binding**
□ How it allows generalization	=> **abstraction**

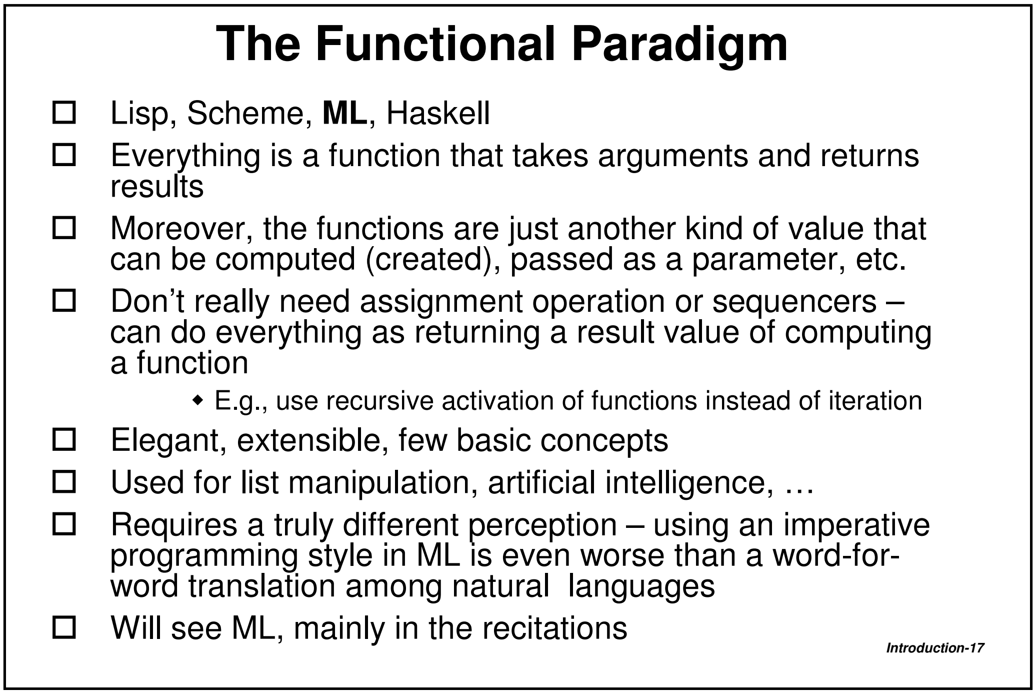
What Characterizes a Programming Language (cont'd)

2. Paradigms

- *Imperative programming:*
  - **Fortran, Cobol, Algol, PL/I, C, Pascal, Ada, C++, Icon, Modula-2, Modula-3, Oberon, Basic.**
- Concurrent programming:
  - **Ada, Occam, Par-C.**
- Object-oriented programming:
  - **Small-talk, Self, C++, Objective-C, Object Pascal, Beta, CLOS, Eiffel**
- *Functional programming:*
  - **Lisp, Scheme, Miranda, ML.**
- *Logic programming:*
  - **Prolog, Prolog-dialects, Turbo-Prolog, Icon.**
- Constraint programming:
  - **CLP, DLP**

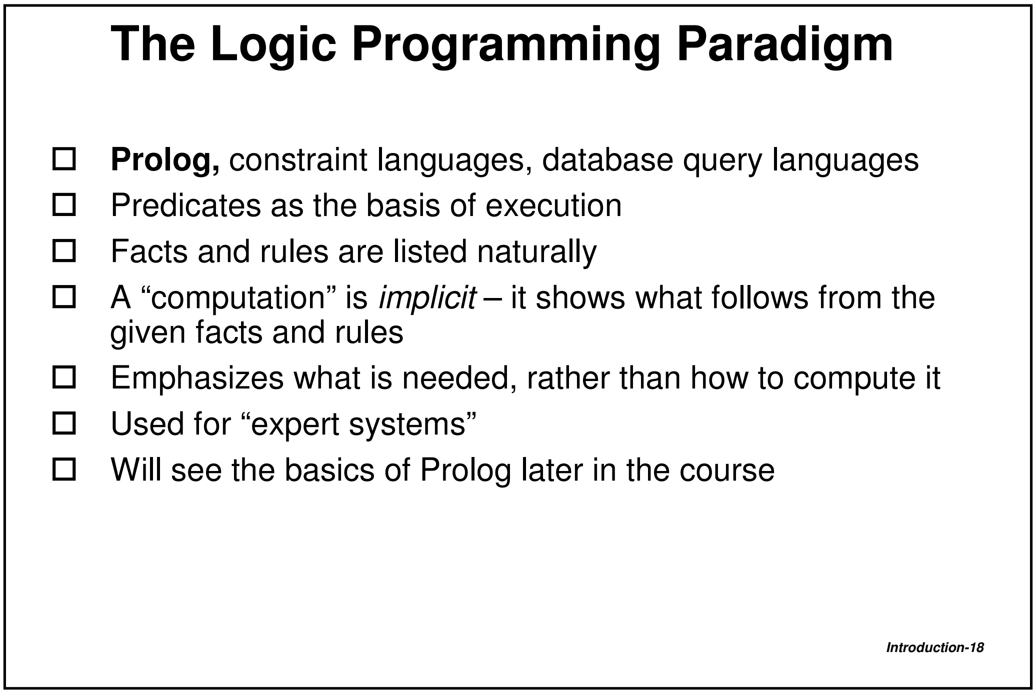
The Imperative Paradigm

- Fortran, Algol, C, **Pascal,** Ada
- The program has a state reflected by storage and location
- It comprises commands (assignments, sequencers, etc.) that update the state of the program
  - They can be grouped into procedures and functions
- There are also expressions and other functional features
- Most familiar, but a large variety of possibilities must be mastered and understood
- Models real-world processes, hence still dominant
- Lends itself to efficient processing (optimizing compilers etc.)
- Will see Pascal, mainly in examples



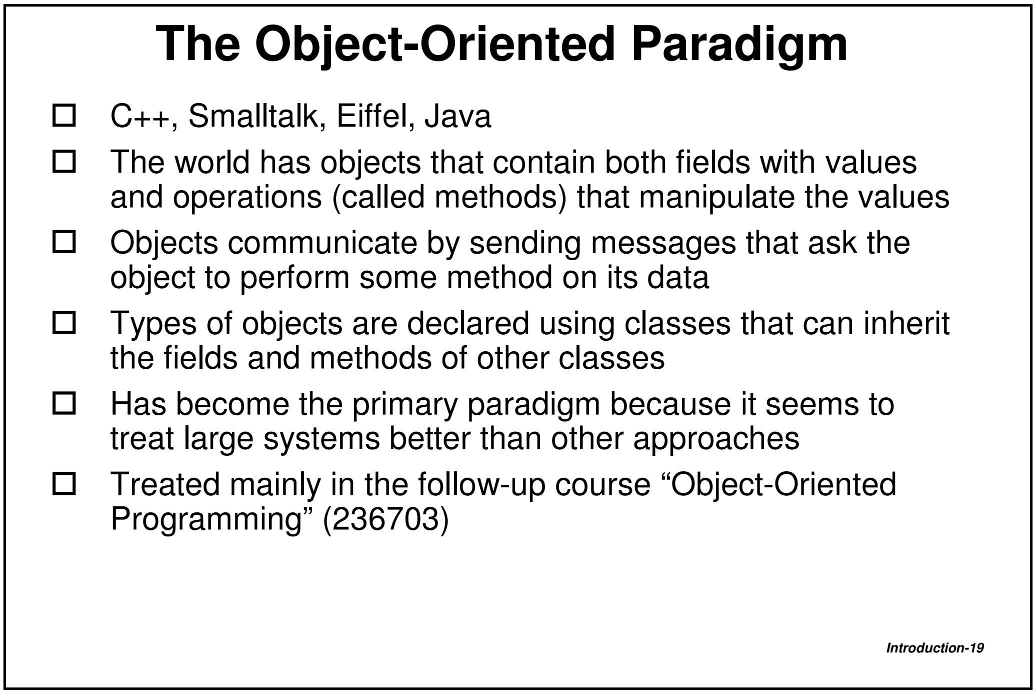
The Functional Paradigm

- □ Lisp, Scheme, **ML**, Haskell
- □ Everything is a function that takes arguments and returns results
- □ Moreover, the functions are just another kind of value that can be computed (created), passed as a parameter, etc.
- □ Don’t really need assignment operation or sequencers – can do everything as returning a result value of computing a function
  - ◆ E.g., use recursive activation of functions instead of iteration
- □ Elegant, extensible, few basic concepts
- □ Used for list manipulation, artificial intelligence, …
- □ Requires a truly different perception – using an imperative programming style in ML is even worse than a word-for-word translation among natural languages
- □ Will see ML, mainly in the recitations



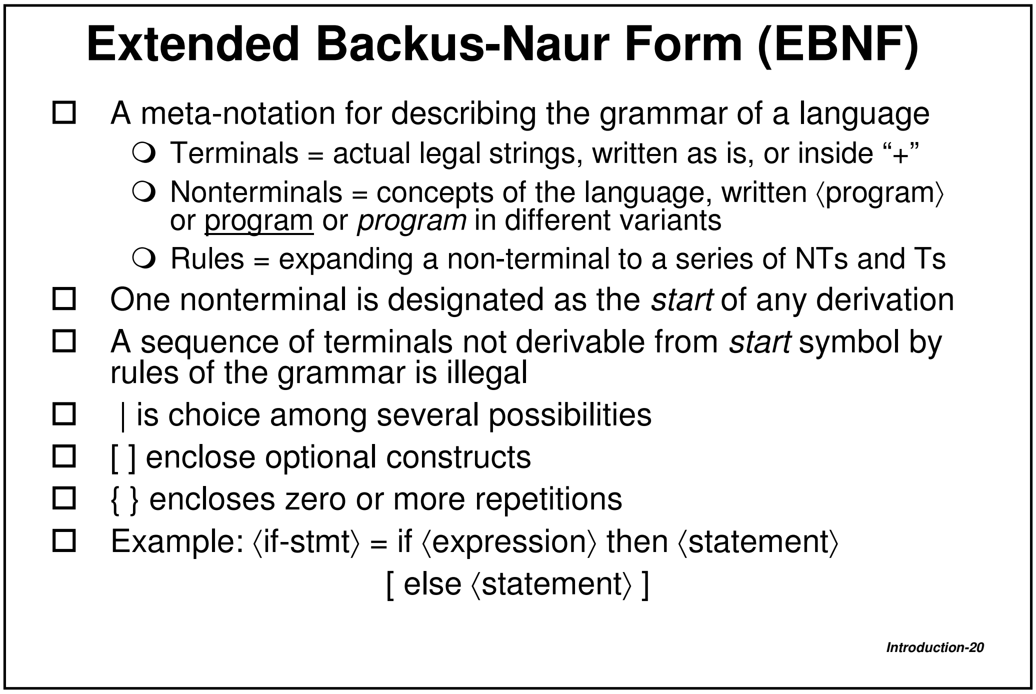
The Logic Programming Paradigm

- □ **Prolog,** constraint languages, database query languages
- □ Predicates as the basis of execution
- □ Facts and rules are listed naturally
- □ A “computation” is *implicit* – it shows what follows from the given facts and rules
- □ Emphasizes what is needed, rather than how to compute it
- □ Used for “expert systems”
- □ Will see the basics of Prolog later in the course



The Object-Oriented Paradigm

- □ C++, Smalltalk, Eiffel, Java
- □ The world has objects that contain both fields with values and operations (called methods) that manipulate the values
- □ Objects communicate by sending messages that ask the object to perform some method on its data
- □ Types of objects are declared using classes that can inherit the fields and methods of other classes
- □ Has become the primary paradigm because it seems to treat large systems better than other approaches
- □ Treated mainly in the follow-up course “Object-Oriented Programming” (236703)



Extended Backus-Naur Form (EBNF)

- □ A meta-notation for describing the grammar of a language
  - ❍ Terminals = actual legal strings, written as is, or inside “+”
  - ❍ Nonterminals = concepts of the language, written ⟨program⟩ or program or *program* in different variants
  - ❍ Rules = expanding a non-terminal to a series of NTs and Ts
- □ One nonterminal is designated as the *start* of any derivation
- □ A sequence of terminals not derivable from *start* symbol by rules of the grammar is illegal
- □ | is choice among several possibilities
- □ [ ] enclose optional constructs
- □ { } encloses zero or more repetitions
- □ Example: ⟨if-stmt⟩ = if ⟨expression⟩ then ⟨statement⟩
  [else ⟨statement⟩]

Example of EBNF

- ⟨expression⟩ = ⟨term⟩ { ⟨add-op⟩ ⟨term⟩ }
- ⟨term⟩ = ⟨factor⟩ { ⟨mult-op⟩ ⟨factor⟩ }
- ⟨factor⟩ = ⟨variable-name⟩ | ⟨number⟩
- ⟨add-op⟩ = + | -
- ⟨mult-op⟩ = ∗ | /

- Is a + 2/ b - c∗7 a legal expression?
- Yes, because there is a sequence of rule applications from ⟨expression⟩ that yields this string (these can be drawn as a “syntax tree”)

Observations on Syntax

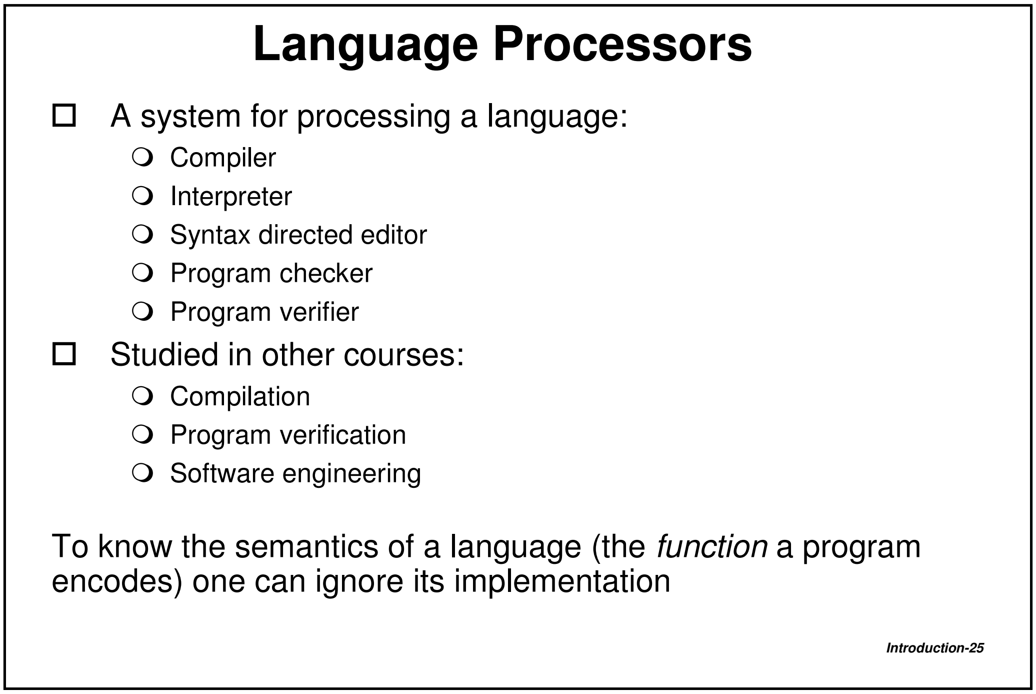
- If there are several possible syntax trees for the same sequence of terminal symbols, the sequence is syntactically **ambiguous,** and that often leads to semantic ambiguity (several possible ways to understand the meaning)
  - **Good programming language design avoids ambiguity**
- There are some syntactic rules that cannot be expressed just using EBNF (which gives *context-free* grammars)
  - **Every variable used is previously declared**
  - **The number of arguments in a procedure call equals the number of arguments in the declaration of the procedure**
- Much more on grammars and identifying legal programs you will learn in the courses *Automata and Formal Languages* and *Compilation*

Relations to Other Fields in Computer Science

- □ **Databases and Information Retrieval:** Query languages - languages for manipulating databases.
- □ **Human-Computer Interaction:** Programming Languages are designed to be written and read by humans
- □ **Operating Systems:** Input-Output support. Storage management. Shells are in fact Programming Languages.
- □ **Computer Architecture:** Programming Language design is influenced by architecture and vice versa. Instructions sets are Programming Languages. Hardware design languages.

Closely Related Topics

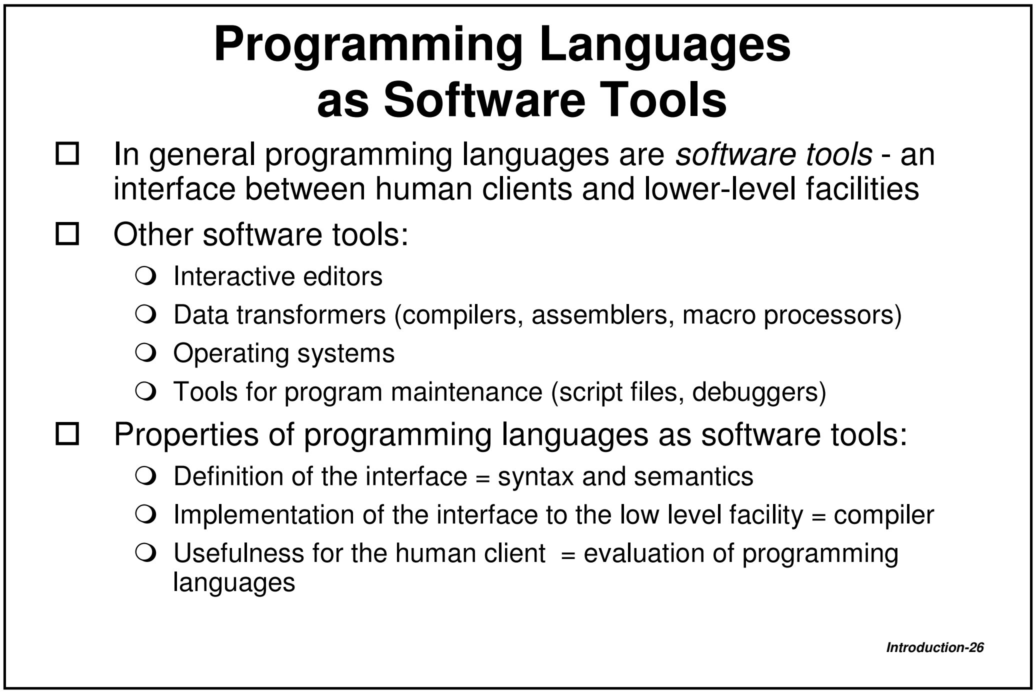
- □ **Automata and Formal Languages, Computability:** Provide the foundation for much of the underlying theory.
- □ **Compilation:** The technology of processing programming languages.
- □ **Software engineering:** The process of building software systems.



Language Processors

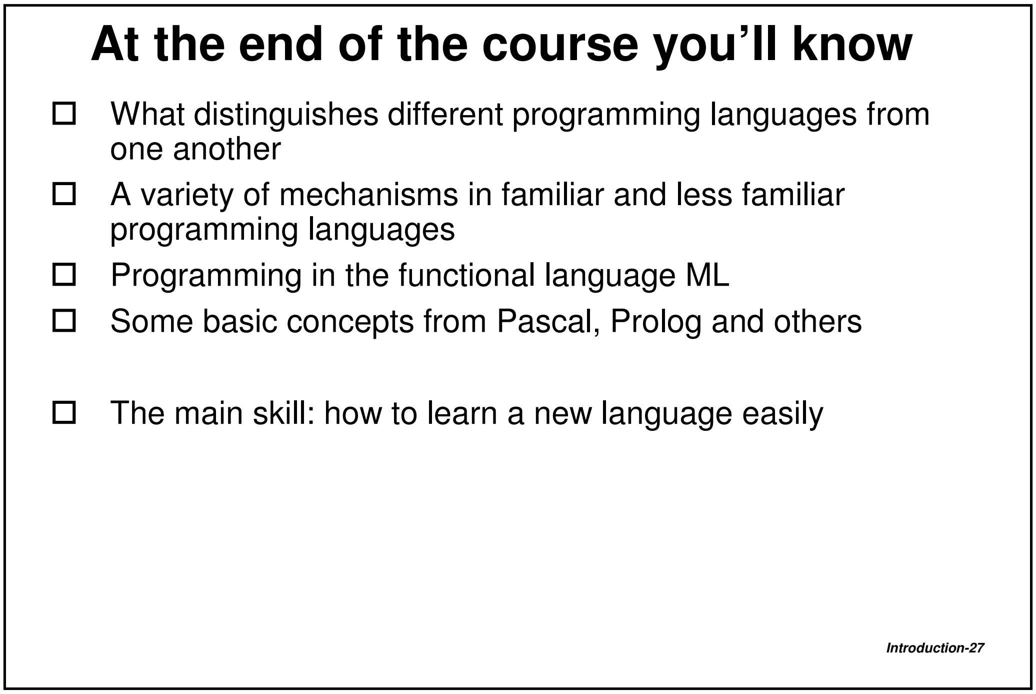
- □ A system for processing a language:
  - ❍ Compiler
  - ❍ Interpreter
  - ❍ Syntax directed editor
  - ❍ Program checker
  - ❍ Program verifier
- □ Studied in other courses:
  - ❍ Compilation
  - ❍ Program verification
  - ❍ Software engineering

To know the semantics of a language (the *function* a program encodes) one can ignore its implementation



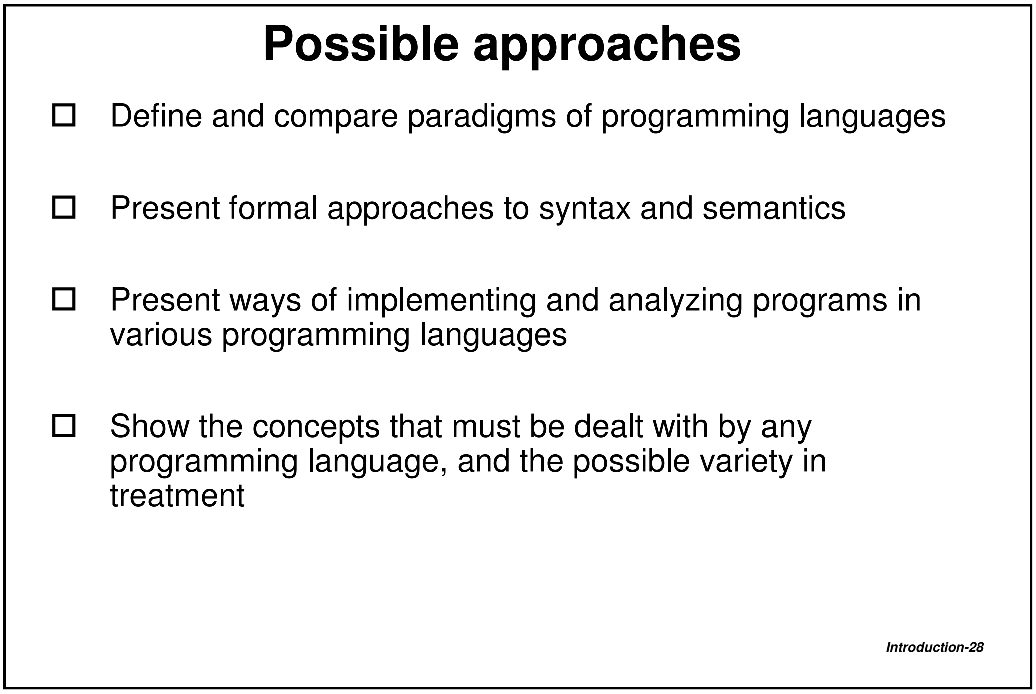
Programming Languages as Software Tools

- □ In general programming languages are *software tools* - an interface between human clients and lower-level facilities
- □ Other software tools:
  - ❍ Interactive editors
  - ❍ Data transformers (compilers, assemblers, macro processors)
  - ❍ Operating systems
  - ❍ Tools for program maintenance (script files, debuggers)
- □ Properties of programming languages as software tools:
  - ❍ Definition of the interface = syntax and semantics
  - ❍ Implementation of the interface to the low level facility = compiler
  - ❍ Usefulness for the human client = evaluation of programming languages



At the end of the course you'll know

- □ What distinguishes different programming languages from one another
- □ A variety of mechanisms in familiar and less familiar programming languages
- □ Programming in the functional language ML
- □ Some basic concepts from Pascal, Prolog and others
- □ The main skill: how to learn a new language easily



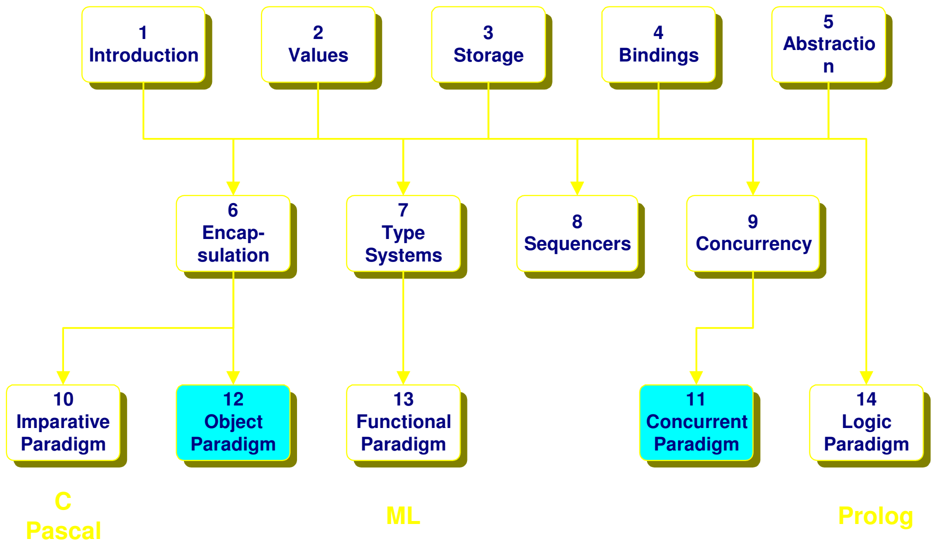
Possible approaches

- □ Define and compare paradigms of programming languages
- □ Present formal approaches to syntax and semantics
- □ Present ways of implementing and analyzing programs in various programming languages
- □ Show the concepts that must be dealt with by any programming language, and the possible variety in treatment

Syllabus

- □ **Concepts**
  - ○ values, types and expressions
  - ○ typing systems
  - ○ storage
  - ○ commands
  - ○ sequencers
  - ○ binding
  - ○ abstraction
- □ **Paradigms**
  - ○ encapsulation
  - ○ functional programming (ML)
  - ○ logic programming (Prolog)
- □ **A bit of formal semantics (as time permits)**

Course Topics*)



1 Introduction

2 Values

3 Storage

4 Bindings

5 Abstractio n

6 Encap- sulation

7 Type Systems

8 Sequencers

9 Concurrency

10 Imparative Paradigm

12 Object Paradigm

13 Functional Paradigm

11 Concurrent Paradigm

14 Logic Paradigm

C Pascal

ML

Prolog

*) according to Watt's book chapters

Text Books

- □ "*Programming Language Concepts and Paradigms",* by David A. Watt. Prentice Hall, 1990. ISBN 0-13-7228874-3. **Chapters 1-8,10,13-14.**
- □ *"Advanced Programming Language Design",* by Raphael Finkel. MIT Press, 1996. ISBN 0-8053-1191-2. **Selected material.**
- ➢ *"Programming Languages: Concepts and Constructs"* (2nd Ed), by Ravi Sethi. Addison-Wesley, 1996. ISBN 0-201-59065-4
- ➢ *"Concepts in Programming Languages"*, by John C. Mitchell. Cambridge University Press, 2002. ISBN 0-5217-8098-5.

Bibliography for Recitations

- ▪ "*Introduction to Pascal"*, by Jim Welsh and John Elder. Prentice Hall, 1979. ISBN 0-1349-1522-4.
- ▪ "*ML for the Working Programmer",* by Lawrence. C. Paulson. Cambridge University Press, 1991. ISBN 0-521-39022-2.
- ▪ "*Prolog Programming for Artificial Intelligence*", by Ivan Bratko. Addison-Wesley. ISBN 0-201-14224-4.
- ▪ "*Programming in Prolog",* by W. F. Clocksin and C. S. Mellish. Springer-Verlag, 1984. ISBN 0-387-11046-1.
- ▪ Slides

Administration

- This is not an internet course! Most material is provided in class and tutorials.
- Some material available on the web - http://webcourse.cs.technion.ac.il/234319
- Official policy will be published at the above site
- Prerequisites and tzmudim are strictly enforced
- Running announcements will be sent through the course mailing list; *please subscribe from the course's webpage*
- Requests and questions: lilas@cs.technion.ac.il
- **Course staff**
  - Prof. Ron Pinter, Taub 705, x4955, `pinter@cs`
    Office hours: Tuesday, 10-12.
  - Lila Shnaiderman (head TA), Taub 218, x5651, `lilas@cs`
  - Alexey Tsitkin, Taub 316, x4341, `lexa@cs`

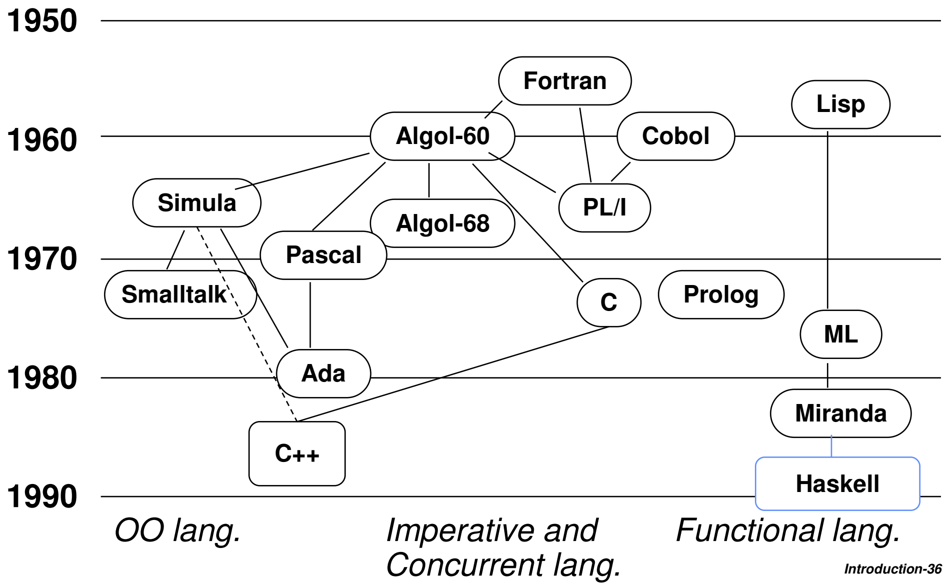
Assignments and Grading

- Assignments:
  - **Every 1-2 weeks**
  - **Weight is up to 20% (TAKEF)**
  - **teams of 2 (strict!) - matching services provided by teaching assistants**
- Final exam:
  - **Date(s) set by the authorities**
  - **Weight is 80% or more**
- Final Grade:
  - **If exam grade > 50: 80% exam, 20% assignments**
  - **Otherwise: 100% exam**

Language Inception and Evolution

- □ **Initial definition by a**
  - ❍ single person: Lisp (*McCarthy*), *APL* (*Iverson*), Pascal (*Wirth*), REXX (*Cowlishaw*), C++ (*Stroustrup*), Java (*Gosling*)
  - ❍ small team: C (Kernighan and Ritchie), ML (Milner et al.), Prolog (Clocksin and Mellish), Icon (Griswold and Griswold)
  - ❍ committee: FORTRAN, Algol, PL/1, Ada
- □ **Some survived, many more perished**
  - ❍ usability
  - ❍ compilation feasibility
  - ❍ dependence on platform
  - ❍ politics and sociology…
- □ **Most successful languages were taken over by a standards' committees** (ANSI, IEEE, ISO, …)
- □ http://www.oreilly.com/news/graphics/prog_lang_poster.pdf

Language Genealogy (till 1990)



1950

1960

1970

1980

1990

Fortran

Lisp

Algol-60

Cobol

Simula

PL/I

Algol-68

Pascal

Smalltalk

C

Prolog

ML

Ada

Miranda

C++

Haskell

OO lang.

Imperative and Concurrent lang.

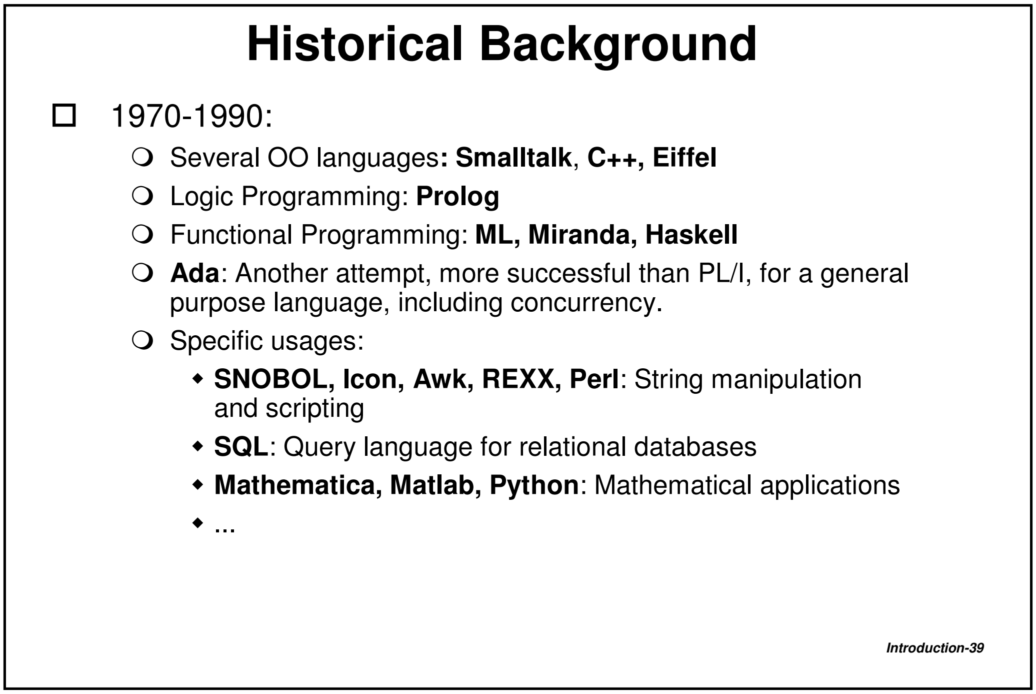
Functional lang.

Historical Background

- Until early 1950s: no real programming languages, but rather **Automatic Programming**, a mixture of assembly languages and other aids for machine code programming.
  - **Mnemonic operation codes and symbolic addresses**
  - **Subroutine libraries where addresses of operands were changed manually**
  - **Interpretive systems for floating point and indexing**
- Early 1950s: the **Laning and Zierler** System (MIT):
  a simple *algebraic* language, a library of useful functions.
- 1954: Definition of **FORTRAN** (FORmula TRANslator). Originally for numerical computing.
  - Symbolic expressions, subprograms with parameters, arrays, **for** loops, **if** statements, no blocks, weak control structures
  - 1957: first working compiler

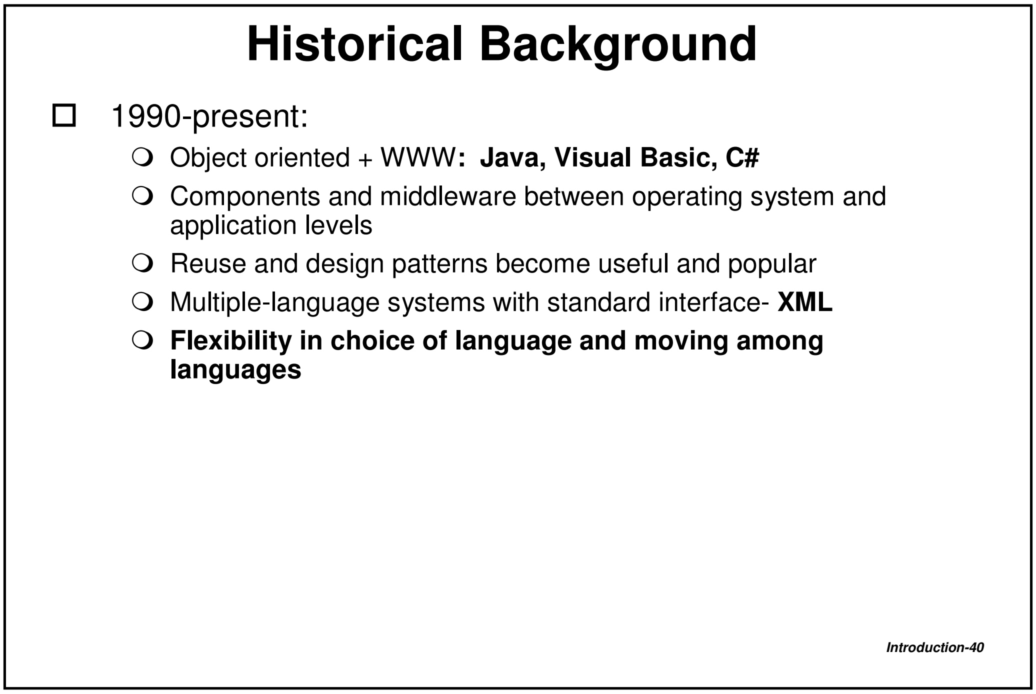
Historical Background (cont'd)

- Early 1960s**:**
  - **Cobol**: Data processing. Means for data description**.**
  - **Algol 60**: Blocks, modern control structures
    - One of the most influential imperative languages
    - Gave rise to the Algol-like languages for two decades (Pascal, PL/1, C, Algol 68; Simula, Ada)
  - **Lisp** (list processing language): symbolic expressions (rather than numerical), computation by list manipulation, garbage collection; *the first functional language*
- Mid 1960s:
  - **PL/1:** an attempt to combine concepts from numerical computation languages (FORTRAN, Algol 60) and data processing languages (Cobol).
  - **Simula**: object oriented, abstract data types



Historical Background

- 1970-1990:
  - Several OO languages**: Smalltalk**, **C++, Eiffel**
  - Logic Programming: **Prolog**
  - Functional Programming: **ML, Miranda, Haskell**
  - **Ada**: Another attempt, more successful than PL/I, for a general purpose language, including concurrency.
  - Specific usages:
    - **SNOBOL, Icon, Awk, REXX, Perl**: String manipulation and scripting
    - **SQL**: Query language for relational databases
    - **Mathematica, Matlab, Python**: Mathematical applications
    - ...



Historical Background

- 1990-present:
  - Object oriented + WWW**: Java, Visual Basic, C#**
  - Components and middleware between operating system and application levels
  - Reuse and design patterns become useful and popular
  - Multiple-language systems with standard interface- **XML**
  - **Flexibility in choice of language and moving among languages**

How Many Ways to Say "Hello, World"?

- □ In the following slides are examples of the most popular computer program "Hello, World" in various programming languages
- □ See how many you can recognize?
- □ Examples are taken (with alterations) from the Web site:

  http://www.latech.edu/~acm/HelloWorld.shtml

- □ More examples at

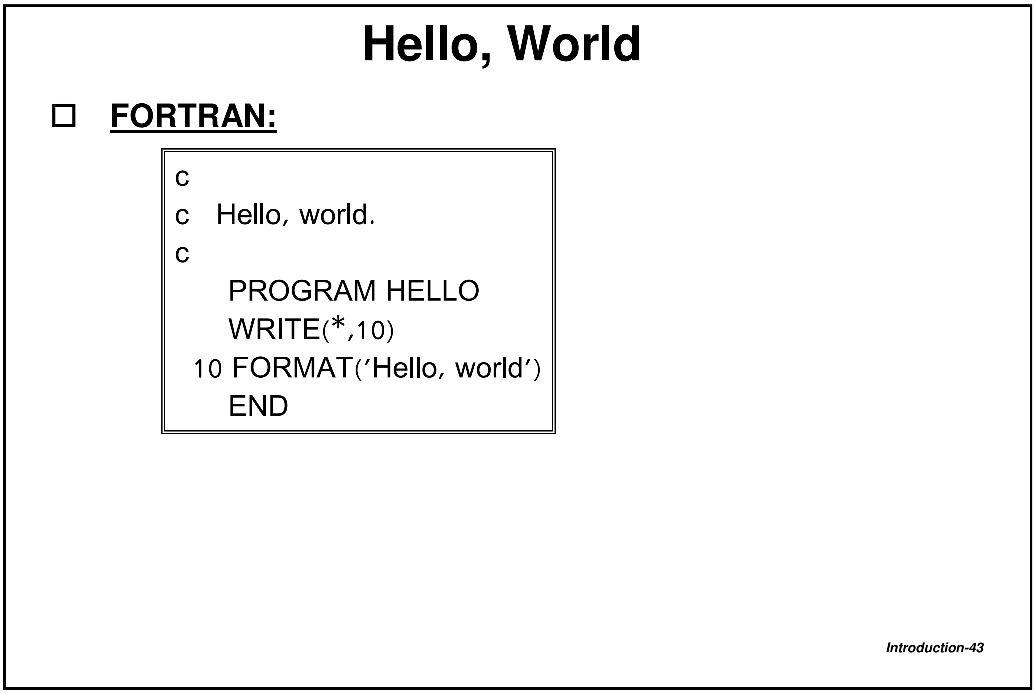
  http://99-bottles-of-beer.ls-la.net/

  (thanks to Michael Bar-Sinai!)

Hello, World

- □ **Assembly 8086:**

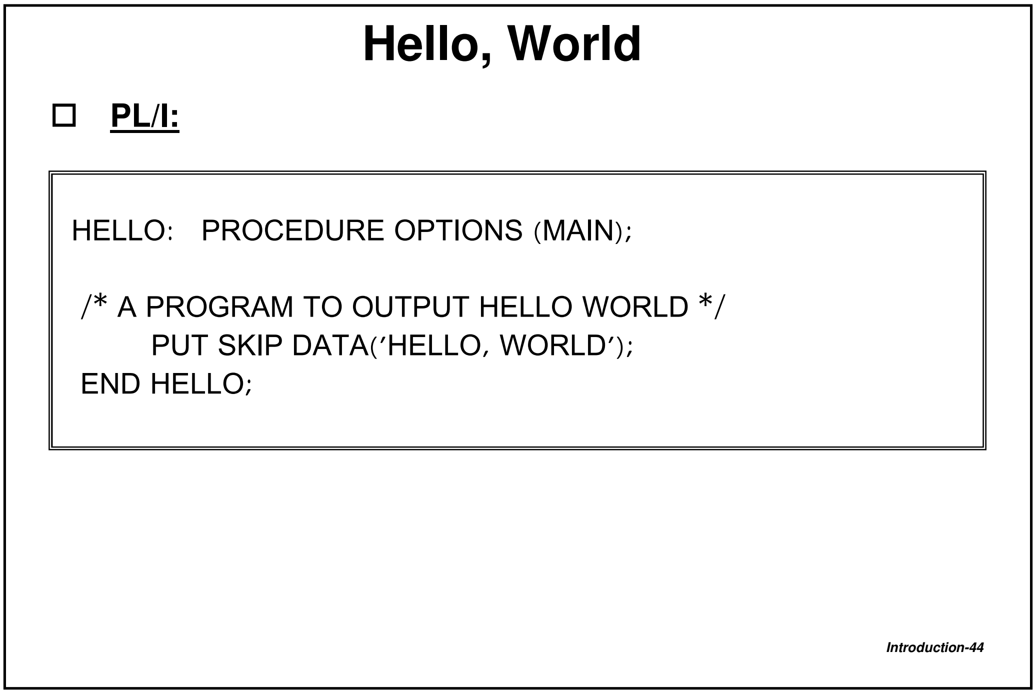
```
.model small
.stack 100h

.data
.hello_message db 'Hello, World',0dh,0ah,'$'

.code
main  proc
      mov    ax,@data
      mov    ds,ax

      mov    ah,9
      mov    dx,ofsett hello_message
      int    21h

      mov    ax,4C00h
      int    21h
main  endp
end   main
```



Hello, World

- □ **FORTRAN:**

```
c
c   Hello, world.
c
      PROGRAM HELLO
      WRITE(*,10)
   10 FORMAT('Hello, world')
      END
```



Hello, World

- □ **PL/I:**

```
HELLO:   PROCEDURE OPTIONS (MAIN);

 /* A PROGRAM TO OUTPUT HELLO WORLD */
        PUT SKIP DATA('HELLO, WORLD');
 END HELLO;
```

Hello, World

- □ **ADA:**

```
with i_o; use i_o;

procedure hello is
begin
   put (″Hello, World″);
end hello;
```

Hello, World

- □ **Prolog:**

```
hello :-
      printstring(″Hello, World″).

printstring([]).
printstring([H|T]) :-
      put(H),
      printstring(T).
```

Hello, World

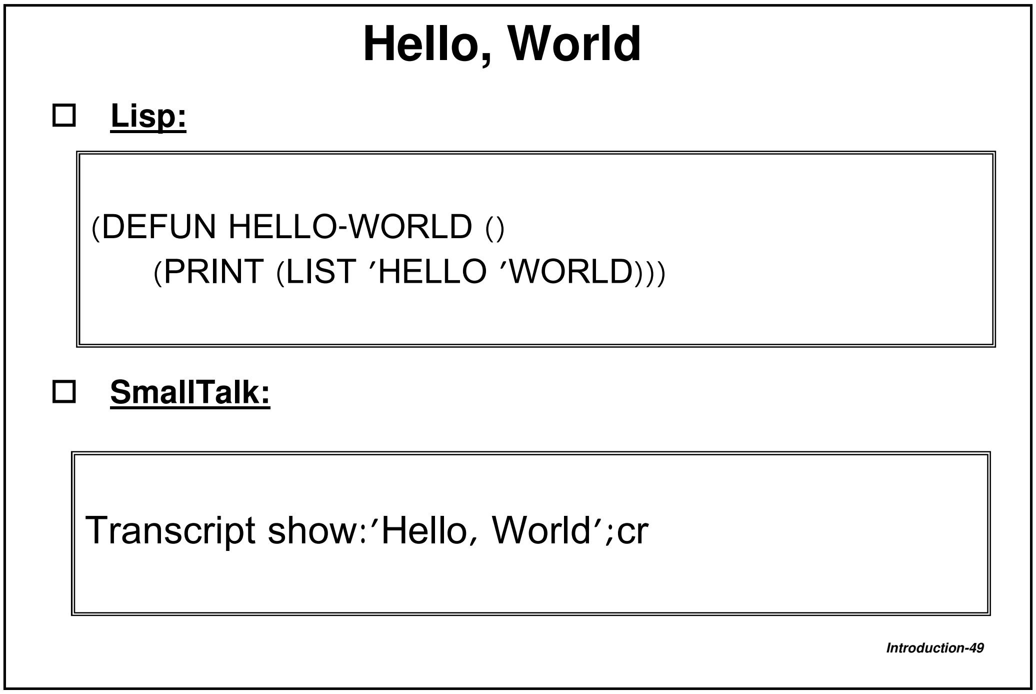
□ **SNOBOL4:**

```
OUTPUT = ’Hello, World‘
END
```

Hello, World

□ **RPG II:**

```
H
FSCREEN O F 80 80      CRT
C         EXCPT
OSCREEN E 1
O                12 ’HELLO, WORLD’
```



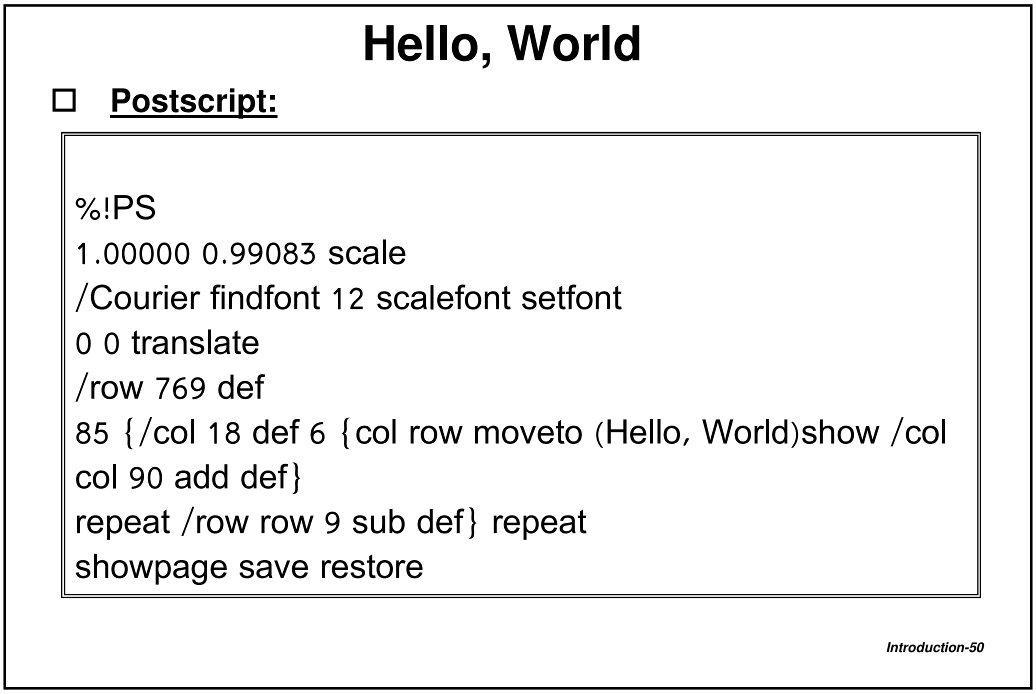
Hello, World

- □ **Lisp:**

```
(DEFUN HELLO-WORLD ()
     (PRINT (LIST ′HELLO ′WORLD)))
```

- □ **SmallTalk:**

```
Transcript show:′Hello, World′;cr
```



Hello, World

- □ **Postscript:**

```
%!PS
1.00000 0.99083 scale
/Courier findfont 12 scalefont setfont
0 0 translate
/row 769 def
85 {/col 18 def 6 {col row moveto (Hello, World)show /col
col 90 add def}
repeat /row row 9 sub def} repeat
showpage save restore
```