Programming Languages
1. Introduction

Oscar Nierstrasz
Roadmap

- Course Schedule
- Programming Paradigms
- A Quick Tour of Programming Language History
# Programming Languages

<table>
<thead>
<tr>
<th>Lecturer:</th>
<th>Oscar Nierstrasz</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Assistants:</td>
<td>Nevena Milojković-Lazarević, Manuel Leuenberger, Mohammadreza Hazhirpasand</td>
<td></td>
</tr>
<tr>
<td>WWW:</td>
<td><a href="http://scg.unibe.ch/teaching/pl">http://scg.unibe.ch/teaching/pl</a></td>
<td></td>
</tr>
</tbody>
</table>
Roadmap

> Course Schedule
> Programming Paradigms
> A Quick Tour of Programming Language History
Sources

> **Text:**
  

> **Other Sources:**


<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23-Feb-18</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>02-Mar-18</td>
<td>Stack-based Programming</td>
</tr>
<tr>
<td>3</td>
<td>09-Mar-18</td>
<td>Functional Programming</td>
</tr>
<tr>
<td>4</td>
<td>16-Mar-18</td>
<td>Types and Polymorphism</td>
</tr>
<tr>
<td>5</td>
<td>23-Mar-18</td>
<td>Lambda Calculus</td>
</tr>
<tr>
<td></td>
<td>30-Mar-18</td>
<td>Good Friday</td>
</tr>
<tr>
<td></td>
<td>06-Apr-18</td>
<td>Easter vacation</td>
</tr>
<tr>
<td>6</td>
<td>13-Apr-18</td>
<td>Fixed Points</td>
</tr>
<tr>
<td>7</td>
<td>20-Apr-18</td>
<td>Programming Language Semantics</td>
</tr>
<tr>
<td>8</td>
<td>27-Apr-18</td>
<td>Objects and Prototypes</td>
</tr>
<tr>
<td>9</td>
<td>04-May-18</td>
<td>Objects and Types</td>
</tr>
<tr>
<td>10</td>
<td>11-May-18</td>
<td>Logic Programming</td>
</tr>
<tr>
<td>11</td>
<td>18-May-18</td>
<td>Applications of Logic Programming</td>
</tr>
<tr>
<td>12</td>
<td>25-May-18</td>
<td>Visual Programming</td>
</tr>
<tr>
<td></td>
<td>01-Jun-18</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>
This is a note (a hidden slide). You will find some of these scattered around the PDF versions of the slides.
Roadmap

> Course Schedule
> **Programming Paradigms**
> A Quick Tour of Programming Language History
What is a Programming Language?

> A formal language for describing computation?
> A “user interface” to a computer?
> Syntax + semantics?
> Compiler, or interpreter, or translator?
> A tool to support a programming paradigm?

A programming language is a notational system for describing computation in a machine-readable and human-readable form.
— Louden
What is a Programming Language? (II)

The thesis of this course:

A programming language is a tool for developing executable models for a class of problem domains.
Themes Addressed in this Course

**Paradigms**
How do different language paradigms support problem-solving?

**Foundations**
What are the foundations of programming languages?

**Semantics**
How can we understand the semantics of programming languages?
Generations of Programming Languages

1GL: machine codes
2GL: symbolic assemblers
3GL: (machine-independent) imperative languages
4GL: domain specific application generators
5GL: AI languages …

Each generation is at a higher level of abstraction
How do Programming Languages Differ?

> **Common Constructs:**
>  — basic data types (numbers, etc.); variables; expressions; statements; keywords; control constructs; procedures; comments; errors ... 

> **Uncommon Constructs:**
>  — type declarations; special types (strings, arrays, matrices, ...); sequential execution; concurrency constructs; packages/modules; objects; general functions; generics; modifiable state; ...
A programming language is a problem-solving tool.

<table>
<thead>
<tr>
<th>Programming Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Imperative style:</strong></td>
<td>program = algorithms + data</td>
</tr>
<tr>
<td></td>
<td><em>good for decomposition</em></td>
</tr>
<tr>
<td><strong>Functional style:</strong></td>
<td>program = functions ◦ functions</td>
</tr>
<tr>
<td></td>
<td><em>good for reasoning</em></td>
</tr>
<tr>
<td><strong>Logic programming style:</strong></td>
<td>program = facts + rules</td>
</tr>
<tr>
<td></td>
<td><em>good for searching</em></td>
</tr>
<tr>
<td><strong>Object-oriented style:</strong></td>
<td>program = objects + messages</td>
</tr>
<tr>
<td></td>
<td><em>good for modeling(!)</em></td>
</tr>
</tbody>
</table>

Other styles and paradigms: blackboard, pipes and filters, constraints, lists, ...
Compilers and Interpreters

Compilers and interpreters have similar front-ends, but have different back-ends.
Roadmap

> Course Schedule
> Programming Paradigms
> A Quick Tour of Programming Language History
# A Brief Chronology

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early 1950s</td>
<td>“order codes” (primitive assemblers)</td>
<td></td>
</tr>
<tr>
<td>1957</td>
<td>FORTRAN</td>
<td>the first high-level programming language</td>
</tr>
<tr>
<td>1958</td>
<td>ALGOL</td>
<td>the first modern, imperative language</td>
</tr>
<tr>
<td>1960</td>
<td>LISP, COBOL</td>
<td>Interactive programming; business programming</td>
</tr>
<tr>
<td>1962</td>
<td>APL, SIMULA</td>
<td>the birth of OOP (SIMULA)</td>
</tr>
<tr>
<td>1964</td>
<td>BASIC, PL/I</td>
<td></td>
</tr>
<tr>
<td>1966</td>
<td>ISWIM</td>
<td>first modern functional language (a proposal)</td>
</tr>
<tr>
<td>1970</td>
<td>Prolog</td>
<td>logic programming is born</td>
</tr>
<tr>
<td>1972</td>
<td>C</td>
<td>the systems programming language</td>
</tr>
<tr>
<td>1975</td>
<td>Pascal, Scheme</td>
<td>two teaching languages</td>
</tr>
<tr>
<td>1978</td>
<td>CSP</td>
<td>Concurrency matures</td>
</tr>
<tr>
<td>1978</td>
<td>FP</td>
<td>Backus’ proposal</td>
</tr>
<tr>
<td>1983</td>
<td>Smalltalk-80, Ada</td>
<td>OOP is reinvented</td>
</tr>
<tr>
<td>1984</td>
<td>Standard ML</td>
<td>FP becomes mainstream (?)</td>
</tr>
<tr>
<td>1986</td>
<td>C++, Eiffel</td>
<td>OOP is reinvented (again)</td>
</tr>
<tr>
<td>1988</td>
<td>CLOS, Oberon, Mathematica</td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>Haskell</td>
<td>FP is reinvented</td>
</tr>
<tr>
<td>1990s</td>
<td>Perl, Python, Ruby, JavaScript</td>
<td>Scripting languages become mainstream</td>
</tr>
<tr>
<td>1995</td>
<td>Java</td>
<td>OOP is reinvented for the internet</td>
</tr>
<tr>
<td>2000</td>
<td>C#</td>
<td></td>
</tr>
</tbody>
</table>
Fortran

**History**

> John Backus (1953) sought to write programs in conventional mathematical notation, and generate code comparable to good assembly programs.

> No language design effort (made it up as they went along)

> Most effort spent on code generation and optimization

> FORTRAN I released April 1957; working by April 1958

> The current standard is FORTRAN 2008 (FORTRAN 2015 is work in progress)
Innovations

> Symbolic notation for subroutines and functions
> Assignments to variables of complex expressions
> DO loops
> Comments
> Input/output formats
> Machine-independence

Successes

> Easy to learn; high level
> Promoted by IBM; addressed large user base
> (scientific computing)
“Hello World” in FORTRAN

```fortran
PROGRAM HELLO
   DO 10, I=1,10
   PRINT *, 'Hello World'
10 CONTINUE
STOP
END
```

All examples from the ACM "Hello World" project:
www2.latech.edu/~acm/HelloWorld.shtml
ALGOL 60

History
> Committee of PL experts formed in 1955 to design universal, machine-independent, algorithmic language
> First version (ALGOL 58) never implemented; criticisms led to ALGOL 60

Innovations
> BNF (Backus-Naur Form) introduced to define syntax (led to syntax-directed compilers)
> First block-structured language; variables with local scope
> Structured control statements
> Recursive procedures
> Variable size arrays

Successes
> Highly influenced design of other PLs but never displaced FORTRAN
BEGIN
FILE F (KIND=REMOTE);
EBCDIC ARRAY E [0:11];
REPLACE E BY "HELLO WORLD!";
WHILE TRUE DO
  BEGIN
    WRITE (F, *, E);
  END;
END.
COBOL

**History**
- Designed by committee of US computer manufacturers
- Targeted business applications
- Intended to be readable by managers (!)

**Innovations**
- Separate descriptions of environment, data, and processes

**Successes**
- Adopted as de facto standard by US DOD
- Stable standard for 25 years
- *Still the most widely used PL for business applications (!)*
```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLOWORLD.
DATE-WRITTEN. 02/05/96 21:04.
* AUTHOR BRIAN COLLINS
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. RM-COBOL.
OBJECT-COMPUTER. RM-COBOL.
DATA DIVISION.
FILE SECTION.
PROCEDURE DIVISION.
MAIN-LOGIC SECTION.
BEGIN.
DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
DISPLAY "HELLO, WORLD." LINE 15 POSITION 10.
STOP RUN.
MAIN-LOGIC-EXIT.
EXIT.
```


**History**
- Designed by committee of IBM and users (early 1960s)
- Intended as (large) general-purpose language for broad classes of applications

**Innovations**
- Support for concurrency (but not synchronization)
- Exception-handling on conditions

**Successes**
- Achieved both run-time efficiency and flexibility (at expense of complexity)
- First “complete” general purpose language
“Hello World” in PL/1

HELLO:  PROCEDURE OPTIONS (MAIN);
     /* A PROGRAM TO OUTPUT HELLO WORLD */
     FLAG = 0;

LOOP:  DO WHILE (FLAG = 0);
     PUT SKIP DATA('HELLO WORLD!');
     END LOOP;

END HELLO;
Functional Languages

> ISWIM (If you See What I Mean)
  — Peter Landin (1966) — paper proposal

> FP
  — John Backus (1978) — Turing award lecture

> ML
  — Edinburgh
  — initially designed as meta-language for theorem proving
  — Hindley-Milner type inference
  — “non-pure” functional language (with assignments/side effects)

> Miranda, Haskell
  — “pure” functional languages with “lazy evaluation”
“Hello World” in Functional Languages

**SML**

```sml
print("hello world!\n");
```

**Haskell**

```haskell
hello() = print "Hello World"
```
Prolog

**History**

> Originated at U. Marseilles (early 1970s), and compilers developed at Marseilles and Edinburgh (mid to late 1970s)

**Innovations**

> Theorem proving paradigm
> Programs as sets of clauses: facts, rules and questions
> Computation by “unification”

**Successes**

> Prototypical logic programming language
> Used in Japanese Fifth Generation Initiative
“Hello World” in Prolog

```prolog
hello :- printstring("HELLO WORLD!!!!").

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

**History**

> **Simula** was developed by Nygaard and Dahl (early 1960s) in Oslo as a language for simulation programming, by adding classes and inheritance to ALGOL 60

```plaintext
Begin
    while 1 = 1 do begin
        outtext("Hello World!");
        outimage;
    end;
End;
```

> **Smalltalk** was developed by Xerox PARC (early 1970s) to drive graphic workstations

```plaintext
Transcript show:'Hello World';cr
```
Object-Oriented Languages

**Innovations**
> Encapsulation of data and operations (contrast ADTs)
> Inheritance to share behaviour and interfaces

**Successes**
> Smalltalk project pioneered OO user interfaces
> Large commercial impact since mid 1980s
> Countless new languages: C++, Objective C, Eiffel, Beta, Oberon, Self, Perl 5, Python, Java, Ada 95 ...
Interactive Languages

> Made possible by advent of time-sharing systems (early 1960s through mid 1970s).

*BASIC*

> Developed at Dartmouth College in mid 1960s
> Minimal; easy to learn
> Incorporated basic O/S commands (NEW, LIST, DELETE, RUN, SAVE)

```
10 print "Hello World!"
20 goto 10
```

> ...
Interactive Languages ...

**APL**

- Developed by Ken Iverson for concise description of numerical algorithms
- Large, non-standard alphabet (52 characters in addition to alphanumerics)
- Primitive objects are arrays (lists, tables or matrices)
- Operator-driven (power comes from composing array operators)
- No operator precedence (statements parsed right to left)

'HELLO WORLD'
Special-Purpose Languages

**SNOBOL**

- First successful string manipulation language
- Influenced design of text editors more than other PLs
- String operations: pattern-matching and substitution
- Arrays and associative arrays (tables)
- Variable-length strings

```
OUTPUT = 'Hello World!'
END
```

> ...
Symbolic Languages ...

Lisp
> Performs computations on symbolic expressions
> Symbolic expressions are represented as *lists*
> Small set of constructor/selector operations to create and manipulate lists
> Recursive rather than iterative control
> *No distinction between data and programs*
> First PL to implement storage management by garbage collection
> Affinity with lambda calculus

(DEFUN HELLO-WORLD ()
  (PRINT (LIST 'HELLO 'WORLD)))
4GLs

“Problem-oriented” languages
- PLs for “non-programmers”
- Very High Level (VHL) languages for specific problem domains

Classes of 4GLs (no clear boundaries)
- Report Program Generator (RPG)
- Application generators

Query languages
- Decision-support languages
- Successes
- Highly popular, but generally ad hoc
“Hello World” in RPG

H
FSCREEN O F 80 80
C
    EXCPT
OSCREEN E 1
O
    CRT
12 'HELLO WORLD!'
"Hello World" in SQL

CREATE TABLE HELLO (HELLO CHAR(12))
UPDATE HELLO
    SET HELLO = 'HELLO WORLD!'
SELECT * FROM HELLO
Scripting Languages

**History**
Countless “shell languages” and “command languages” for operating systems and configurable applications

Unix shell (ca. 1971) developed as user shell and scripting tool

HyperTalk (1987) was developed at Apple to script HyperCard stacks

TCL (1990) developed as embedding language and scripting language for X windows applications (via Tk)

Perl (~1990) became de facto web scripting language

```bash
echo "Hello, World!"
```

```swift
on OpenStack
    show message box
    put "Hello World!" into message box
end OpenStack
```

```bash
puts "Hello World 
```

```bash
print "Hello, World!\n";
```
Scripting Languages ...

Innovations
> Pipes and filters (Unix shell)
> Generalized embedding/command languages (TCL)

Successes
> Unix Shell, awk, emacs, HyperTalk, AppleTalk, TCL, Python, Perl, VisualBasic ...
The future?

> Dynamic languages
  — very active

> Domain-specific languages
  — very active

> Visual languages
  — many developments, but still immature

> Modeling languages
  — emerging from UML and MDE …
What you should know!

> What, exactly, is a programming language?
> How do compilers and interpreters differ?
> Why was FORTRAN developed?
> What were the main achievements of ALGOL 60?
> Why do we call C a “Third Generation Language”?
> What is a “Fourth Generation Language”? 
Can you answer these questions?

> Why are there so many programming languages?
> Why are FORTRAN and COBOL still important programming languages?
> Which language should you use to implement a spelling checker?
> A filter to translate upper-to-lower case?
> A theorem prover?
> An address database?
> An expert system?
> A game server for initiating chess games on the internet?
> A user interface for a network chess client?
Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:
- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

http://creativecommons.org/licenses/by-sa/4.0/