LUND
UNIVERSITY

EDAP05: Concepts of Programming Languages

LECTURE 1: INTRODUCTION

Christoph Reichenbach

Contents

- Programming languages: structure and semantics
- Some language implementation considerations
- How to evaluate and compare languages
- Advanced material:
  - Compilers: **EDAN65**
  - Optimising Compilers: **EDAN75**
  - Program Analysis: **EDAP15**
  - Project Course: (**EDAN70**, **EDAN90**)

What we will not be covering

- Concurrency
- Software tools
- How to build a compiler

Course Structure

Information

- Today's lecture
- Our Textbook
- Course Supplements

Interaction

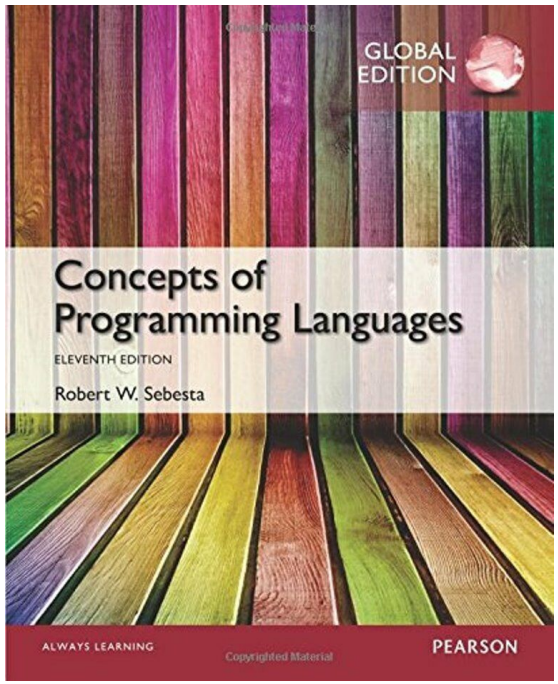
- 2× per week: Class Sessions
- Exercises
- Online course system
- Zoom office hours
- Online discussion system (tbd)
- e-mail:
  `christoph.reichenbach@cs.lth.se`
- TAs (faster replies!):
  - Anton:
    `anton.risberg_alakula@cs.lth.se`
  - Idriss:
    `idriss.riouak@cs.lth.se`

Skills

- Skill-based learning:
  - Enumerated list of skills that you need to pass the exam
  - Skill numbers connected to book, supplements, exercises

Conversational Classroom

- Future lectures are based on the textbook:



(+ Supplements)

- **Read the sections of the book listed on the weekly schedule, prepare your questions ahead of time!**
- Lecture slots *interactive* Q&A

Bring your questions!

Online Systems

All accessible via `http://cs.lth.se/EDAP05` :

- Schedule and Skillset overview
  - What skills are you supposed to know?
  - What lecture / reading material helps you with those skills?
- Group and Homework management via the *Course Online system* (Online Tomorrow)
- Discussions?

Exercises

- Five weekly exercises
  - Starting next week
  - **Available**: Thursday 08:00
  - **Deadline**: Wednesday 14:00 the week after
  - One exception per group can be handed in late
  - **Submission**: Course online system
- Done in groups of two (group selection in online system)
- Get help from TAs during labs (sign-up: online system)

Thu	08:15–10:00
Thu	13:15–15:00
Fri	15:15–17:00

- Need 50% on each assignment to be admitted to final exam
- Bonus on final exam if you get 80% or better right:
  - 1% for 80% to $< 90\%$
  - 2% for 90% or more
- *Late exceptions don't count towards bonus points*

Exam

15 January (Sat), 08:00–13:00, in Vic:2a / 2b

- All exam questions based on the skills from our skill list
- No more than 25% of points based on *synthesis*:
  - Interaction between two or more skills
- Alternative option (only for exchange students whose exchange will have ended by the time of the exam): Project + Report + Presentation

Week Overview

Mo	Tu	We	Th	Fr	
	Class Session 15:15 E:A	Class Session 15:15 **V**:A	New Exercise Labs	Labs	

Mo	Tu	We	Th	Fr	
		Submit exercise solution			

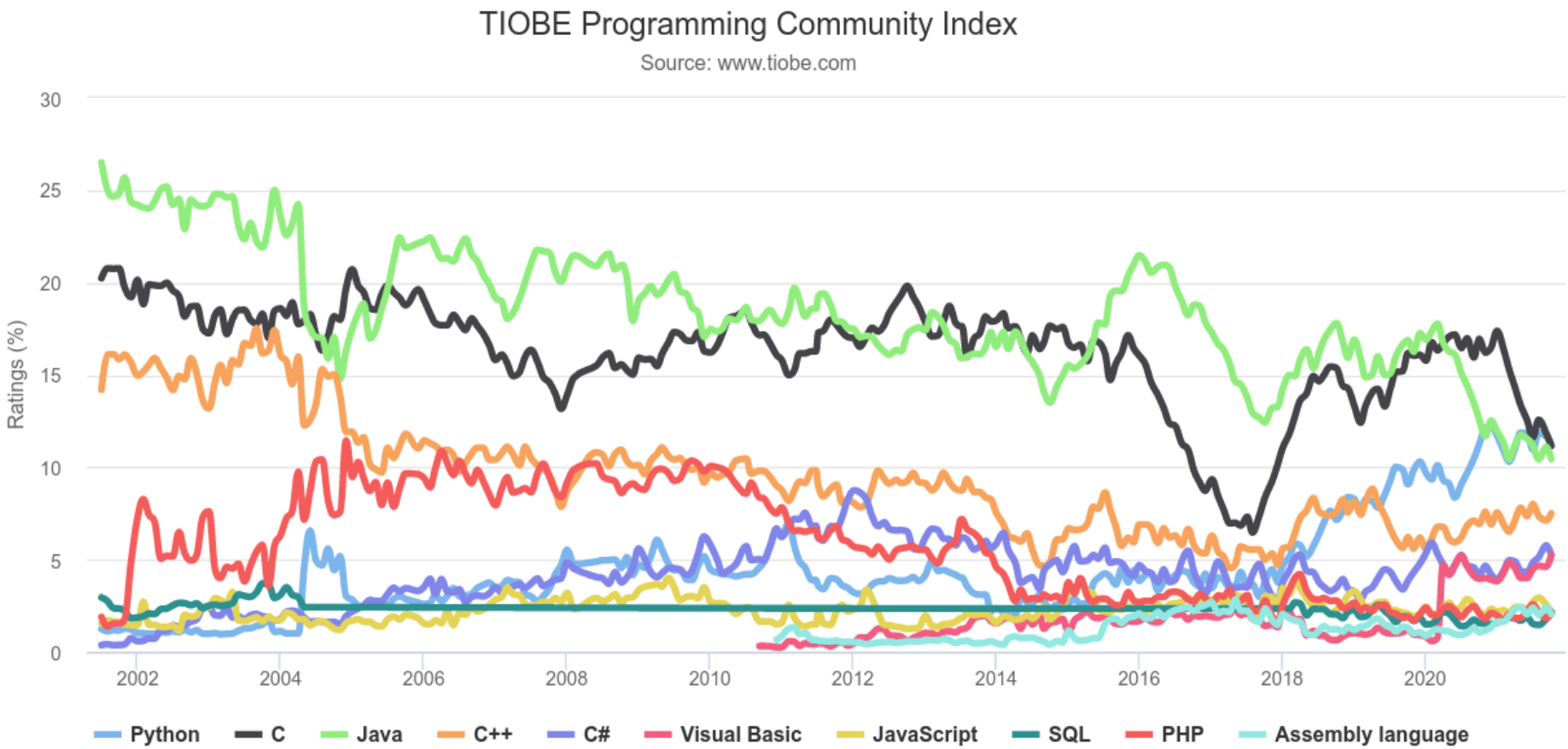
Why Study Programming Languages?

TIOBE Programming Language Index

Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	↑	Python	11.27%	-0.00%
2	1	↓	C	11.16%	-5.79%
3	2	↓	Java	10.46%	-2.11%
4	4		C++	7.50%	+0.57%
5	5		C#	5.26%	+1.10%
6	6		Visual Basic	5.24%	+1.27%
7	7		JavaScript	2.19%	+0.05%
8	10	↑	SQL	2.17%	+0.61%
9	8	↓	PHP	2.10%	+0.01%
10	17	⇈	Assembly language	2.06%	+0.99%
11	19	⇈	Classic Visual Basic	1.83%	+1.06%
12	14	↑	Go	1.28%	+0.13%
13	15	↑	MATLAB	1.20%	+0.08%
14	9	⇊	R	1.20%	-0.79%
			Groovy	1.18%	-0.05%

Source: `tiobe.com`

TIOBE Programming Language Chart



Today

- What are programming languages?
- How can we describe languages?
- How can we compare language features?
- Exploring language features:
  - Meaning
  - Impact on language implementation

Purpose of a Programming Language?

Some Languages

Some Differences between Languages

Expressive Power

- Can language A compute more than language B?
  - **Church-Turing Thesis**:
    Anything that can be computed by some machine in a finite number of steps can be computed by one of the following, and vice-versa:
    - Turing Machines
    - Church's untyped Lambda calculus
    - Gödel's generally recursive functions
    - The Random Access Machine (Goldstine, Burks, von Neumann)
    - The 110 cellular automaton
    - Semi-Thue systems
    - JavaScript

      ...
  - Such languages are called **Turing-Complete**

Sub-Turing Languages

- Some languages are not Turing-Complete:
  - Regular expressions
  - SQL queries
  - *Backus-Naur Form* (BNF)

General-Purpose Languages

General-Purpose Languages

- Turing-Complete
- Suitable for arbitrary tasks

Domain-Specific Languages

- *Possibly* Sub-Turing
- Focus on specialised applications
  - Application-specific syntax
  - Application-specific error checking
  - Application-specific optimisations

Domain-Specific Language example: Regular expression:

```
.*Test(ing)?$
```

- Match any string that ends in either Test or Testing
- Can be compiled into high-performance matching algorithm

Language Critique

- What is the best programming language?
  - Best for *what task*?
  - Measured by *what criteria*?
  - Measurements obtained *how*?
    (For most criteria, we don't have good measurement tools!)
- Qualities of:
  - the language
  - the implementation(s)
  - the available tooling
  - the available libraries
  - other infrastructure (user groups, books, . . .)

Criterion: Readability

- How easy is it to read software in the language?

A Program

```
++++++++[>++++[
>++>+++>+++>+<<
<<-]>+>+>->>+[<
]<-]>>.>---.+++
++++..+++.>>.<-
.<.+++.------.-
-------.>>+.>++.
```

- Program 1:

$$\sqrt{\sum_{v \in S} v^2}$$

- Program 2:
  Multiply each number in S with itself, add up all the results to compute a *sum*, and then give me the nonnegative number that, when multiplied with itself, is equal to that *sum*.

- Readability depends on:
  - Problem domain (typical notation?)
  - Reader's background
- Multiple general *characteristics* help us understand readability

Simplicity

- Small number of features
- Minimal redundancy

Example

- Modula-3 language:
  Design deliberately limited to 50 pages

Counter-Example

Python

```
def d(x):
    r = x[::-1]
    return x == r
```

Orthogonality

- Features can be combined freely
- Minimal overlap between features

Example

- loops / conditionals may contain other loops / conditionals
- Many functional languages: 'Everything is a value'

Counter-Example

C

```
// global variable section

float f1 = 2.0f * 2.0f;
float f2 = sqrt(2.0f); // error
```

Syntax Design

Example

C

```
if (cond)
  print(a);
  print(b);
```

Go

```
if cond {
  print(a);
  print(b);
}
```

Counter-Example

Fortran 95

```
program hello
  implicit none
  integer end, do
  do = 0
  end = 10
  do do=do,end
    print *,do
  end do
end program hello
```

Data Types

- Datatypes can communicate intent
- Possibly enforce checking

Java

```
enum Color {
  Red, Green, Blue
};
...
Color c = readColorFromUser();
```

Summary: Readability Characteristics

- **Readability** helps us understand code
- Core *characteristics*:
  - Simplicity
  - Orthogonality
  - Syntax Design
  - Datatypes

Criterion: Writability

- How easy is it to write software in the language?
- Characteristics that contribute to **Readability** contribute to **Writability**
- Further criteria for **Writability**:
  - **Support for Abstraction**
    - over values (via variables)
    - over expressions (via functions)
    - over statements (via subprograms)
    - over types...
  - **Expressivity**

Criterion: Reliability

- How easy is it to write *reliable* software in the language?
- Criteria that contribute to **Readability** or **Writability** also contribute to **Reliability**
- Further criteria:
  - **Type Checking**
    - The language prevents *type errors* ($\rightarrow$ in two weeks)
  - **Exception Handling**
    - The language allows errors during execution to be systematically escalated ($\rightarrow$ later)
  - **Restricted Aliasing**

Restricted Aliasing

Java

```
public static <T> void
concat(List<T> lhs, List<T> rhs) {
    for (int i = 0; i < rhs.size(); i++) {
        lhs.add(rhs.get(i));
    }
}

concat(a, a);
```

- Attach *rhs* to the end of *lhs*
- This code misbehaves (infinite loop) when passed the same list for both parameters
- **Aliasing**: two different names mean the same thing

Criterion: Cost

- **Cost** explains the investment needed to use a language:
  - Training time
  - Programming time
  - Compilation time
  - Run time
  - Financial cost of special software
  - Cost of limited reliability
    - Maintenance time
    - Insurance cost

Language Evaluation Summary

	Readability	Writability	Reliability
Simplicity	+	+	+
Orthogonality	+	+	+
Types	+	+	+
Syntax Design	+	+	+
Abstraction Support		+	+
Expressivity		+	+
Type Checking			+
Exception Handling			+
Restricted Aliasing			+

(this is Robert W. Sebesta, "Concepts of Programming Languages", Table 1.1)

- Separate dimension: **Cost**
- Alternative (more detailed) model: Green and Petre, "Cognitive Dimensions of Notation"

Tomorrow

- Computer Systems Background
- Memory
- Compilation
- Run-Time Systems
- Primitive Types

Read the listed parts of the book, bring your questions!