Principles of Programming Languages
Lecture 1: Introduction

Andrei Arusoaie[1]

[1]Department of Computer Science

October 3, 2017

Outline

Principles of Programming Languages - Motivation

- Learning a new PL is nowadays a requirement
- “Programmers need to move from one PL to another with naturalness and speed” [Gabbrielli2010]
- PLs have their similarities, analogies, inherited characteristics

GOAL:

understand the basic mechanisms behind the creation of a PL

Principles of Programming Languages - Motivation

- Learning a new PL is nowadays a requirement
- “Programmers need to move from one PL to another with naturalness and speed” [Gabbrielli2010]
- PLs have their similarities, analogies, inherited characteristics

GOAL:

understand the basic mechanisms behind the creation of a PL

Principles of Programming Languages - Motivation

- Learning a new PL is nowadays a requirement
- “Programmers need to move from one PL to another with naturalness and speed” [Gabbrielli2010]
- PLs have their similarities, analogies, inherited characteristics

GOAL:

understand the basic mechanisms behind the creation of a PL

Principles of Programming Languages - Motivation

- Learning a new PL is nowadays a requirement
- “Programmers need to move from one PL to another with naturalness and speed” [Gabbrielli2010]
- PLs have their similarities, analogies, inherited characteristics

GOAL:

understand the basic mechanisms behind the creation of a PL

Skills to develop

- The definition of a programming language:

  Syntax + *Semantics*

- Have a better understanding of the PLs you already know
- Ability to learn a new PL fast
- Get familiar with a framework for defining languages

Skills to develop

- The definition of a programming language:

  Syntax + *Semantics*

- Have a better understanding of the PLs you already know
- Ability to learn a new PL fast
- Get familiar with a framework for defining languages

Skills to develop

- The definition of a programming language:

$$\textit{Syntax} + \textit{Semantics}$$

- Have a better understanding of the PLs you already know
- Ability to learn a new PL fast
- Get familiar with a framework for defining languages

Skills to develop

- The definition of a programming language:

$$\textit{Syntax} + \textit{Semantics}$$

- Have a better understanding of the PLs you already know
- Ability to learn a new PL fast
- Get familiar with a framework for defining languages

Principles of Programming Languages - Organisation

- People: Andrei Arusoaie
- Period: 1^{st} semester (Fall + Winter 2017), 2^{nd} year
- http://www.uaic.ro/wp-content/uploads/2013/12/Structura_an_univ_2017-2018.pdf
- 14 (full activity) weeks
- Final grade = 50 (midterm) + 50 (project)
- Midterm: 50 points, week 7 - lab test ("open book" exam)
- Project: 50 points, 2 stages: 20 (Dec 20) + 30 (Jan 17);
- Bonus: presentations or very interesting projects
- Criteria: min 50 points

Principles of Programming Languages - Organisation

- People: Andrei Arusoaie
- Period: 1^{st} semester (Fall + Winter 2017), 2^{nd} year
- `http://www.uaic.ro/wp-content/uploads/2013/12/Structura_an_univ_2017-2018.pdf`
- 14 (full activity) weeks
- Final grade = 50 (midterm) + 50 (project)
- Midterm: 50 points, week 7 - lab test ("open book" exam)
- Project: 50 points, 2 stages: 20 (Dec 20) + 30 (Jan 17);
- Bonus: presentations or very interesting projects
- Criteria: min 50 points

Principles of Programming Languages - Organisation

- People: Andrei Arusoaie
- Period: 1^{st} semester (Fall + Winter 2017), 2^{nd} year
- `http://www.uaic.ro/wp-content/uploads/2013/12/Structura_an_univ_2017-2018.pdf`
- 14 (full activity) weeks
- Final grade = 50 (midterm) + 50 (project)
- Midterm: 50 points, week 7 - lab test (“open book“ exam)
- Project: 50 points, 2 stages: 20 (Dec 20) + 30 (Jan 17);
- Bonus: presentations or very interesting projects
- Criteria: min 50 points

Lectures - Organisation

- Main course material:

  "*Programming Languages: Principles and Paradigms*"

  Authors: Maurizio Gabbrielli and Simone Martini (Springer-Verlag 2010)

- Available online here:

  `http://websrv.dthu.edu.vn/attachments/newsevents/content2415/Programming_Languages_-_Principles_and_Paradigms_thereds1106.pdf`

Labs - Organisation

- Lab: the K framework (version 4.0)
- Web: `http://www.kframework.org/`
- K-4.0 release:

  `https://github.com/kframework/k/releases/tag/v4.0.0`
- Video tutorials*:

  `https://youtu.be/3ovulLNCEQc?list=`
  `PLQMvp5V6ZQjOm4JZK15s-WJtQHxOmb2h7`

Labs - Organisation

- Lab: the K framework (version 4.0)
- Web: `http://www.kframework.org/`
- K-4.0 release:

  `https://github.com/kframework/k/releases/tag/v4.0.0`
- Video tutorials*:

  `https://youtu.be/3ovulLNCEQc?list=`
  `PLQMvp5V6ZQjOm4JZK15s-WJtQHxOmb2h7`

Principles of Programming Languages - Communication

- Course page [DEMO]:

  `https://profs.info.uaic.ro/~arusoaie.andrei/lectures/PLP/2017/plp.html`

- Facebook group:

  `https://www.facebook.com/groups/130038550972094/`

- Email: arusoaie.andrei@info.uaic.ro
- Slack: `https://plp-fii.slack.com/signup` or send me an email to send you an invitation

Today's lecture

1. Preliminary discussion
2. History
3. Programming paradigms
4. Debate & preparing lab

Today's lecture

1. Preliminary discussion
2. History
3. Programming paradigms
4. Debate & preparing lab

Today's lecture

1. Preliminary discussion
2. History
3. Programming paradigms
4. Debate & preparing lab

Today's lecture

1. Preliminary discussion
2. History
3. Programming paradigms
4. Debate & preparing lab

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

Quick questions, debatable answers

- What is a programming language?
- Why do we need programming languages?
- Which languages have you heard about?
- Have you ever created your own programming language?
- How to design a programming language?
- What is the best programming language?

History - I

- Programming computers
- 1944: ASCC/MARK I
  - built by IBM and Harward Univ.
  - programmed using punched tapes and external physical media (like switches)
- 1946: ENIAC
  - J. Mauchly and J.P. Eckert, early (design) J. von Neumann
  - no storage
  - programmed using external physical media (electrical cables)
  - used to calculate ballistic trajectories

History - I

- Programming computers
- 1944: ASCC/MARK I
  - built by IBM and Harward Univ.
  - programmed using punched tapes and external physical media (like switches)
- 1946: ENIAC
  - J. Mauchly and J.P. Eckert, early (design) J. von Neumann
  - no storage
  - programmed using external physical media (electrical cables)
  - used to calculate ballistic trajectories

History - II

- 1GL (first generation languages): machine language
  - binary code
  - hard to use, too “machine connected”
  - people realized that they need something closer to the user’s natural language
- 2GL: (second generation languages): assembly language
  - introduced to ease the development
  - translated into machine code
  - downside: each machine has its own assembly language

History - II

- 1GL (first generation languages): machine language
  - binary code
  - hard to use, too “machine connected”
  - people realized that they need something closer to the user’s natural language
- 2GL: (second generation languages): assembly language
  - introduced to ease the development
  - translated into machine code
  - downside: each machine has its own assembly language

History - III

- 1950 – 3GL: high-level languages
  - abstract languages
  - ignore the physical characteristics of the computer
  - well suited to express *algorithms*
- 1950: ALGOL = ALGOrithmic Languages (family of languages)
- 1957: FORTRAN = FORmula TRANslation
- 1960: LISP = LISt Processor
- However, the development was still an issue
- Lots of human resources required, expensive hardware

History - III

- 1950 – 3GL: high-level languages
  - abstract languages
  - ignore the physical characteristics of the computer
  - well suited to express *algorithms*
- 1950: ALGOL = ALGOrithmic Languages (family of languages)
- 1957: FORTRAN = FORmula TRANslation
- 1960: LISP = LISt Processor
- However, the development was still an issue
- Lots of human resources required, expensive hardware

History - III

- 1950 – 3GL: high-level languages
  - abstract languages
  - ignore the physical characteristics of the computer
  - well suited to express *algorithms*
- 1950: ALGOL = ALGOrithmic Languages (family of languages)
- 1957: FORTRAN = FORmula TRANslation
- 1960: LISP = LISt Processor
- However, the development was still an issue
- Lots of human resources required, expensive hardware

History IV

- 1970's: microprocessor; batch processing
- Further abstractions pushed PL development to what PL are today
- New paradigms: object-oriented, declarative programming
- The C language: Dennis Ritchie and Ken Thompson
  - Initially designed for the UNIX operating system
  - Successor of B
  - Followed by: Pascal, SmallTalk
- Declarative languages: ML (Meta Language), PROLOG

History IV

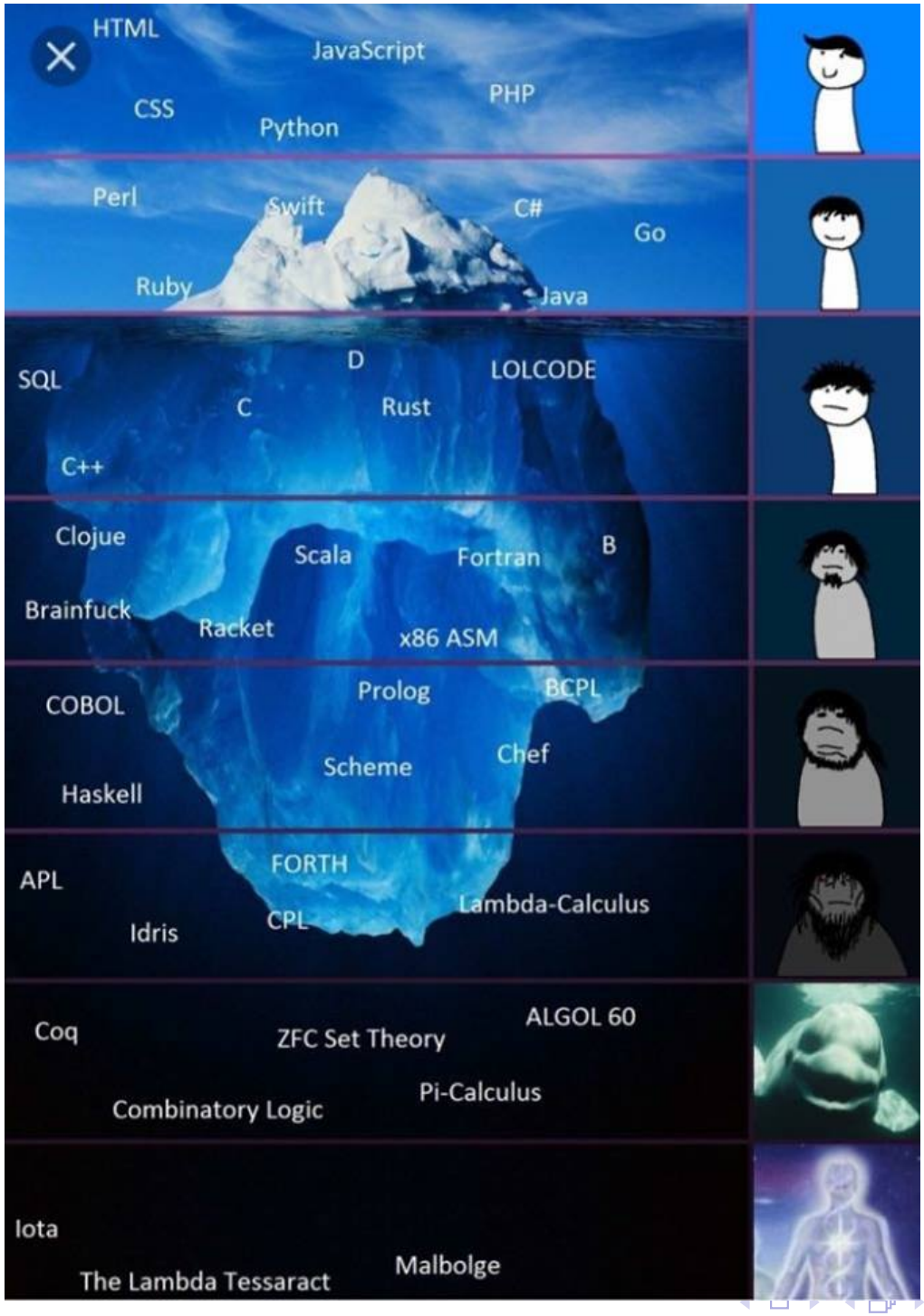
- 1970's: microprocessor; batch processing
- Further abstractions pushed PL development to what PL are today
- New paradigms: object-oriented, declarative programming
- The C language: Dennis Ritchie and Ken Thompson
  - Initially designed for the UNIX operating system
  - Successor of B
  - Followed by: Pascal, SmallTalk
- Declarative languages: ML (Meta Language), PROLOG

History IV

- 1970's: microprocessor; batch processing
- Further abstractions pushed PL development to what PL are today
- New paradigms: object-oriented, declarative programming
- The C language: Dennis Ritchie and Ken Thompson
  - Initially designed for the UNIX operating system
  - Successor of B
  - Followed by: Pascal, SmallTalk
- Declarative languages: ML (Meta Language), PROLOG

History - V

- 1980's: the PC era
- 1986: C++ - Bjarne Stroustrup
- 1989: HTML - T. Berners-Lee, Python - Guido van Rossum
- 1990's: internet, WWW
- 1990-1995: Java - Jim Gosling at SUN
- 1994 : PHP - Rasmus Lerdorf
- 2000 : C# - Anders Hejlsberg
- 2004 : Scala - Martin Odersky
- 2012 : Go - at Google Inc. (started in 2007 by R. Griesemer, R. Pike, K. Thompson)
- 2014 : Swift - at Apple Inc. (started in 2010 by Chris Lattner)


HTML
JavaScript
CSS
PHP
Python
Perl
Swift
C#
Go
Ruby
Java
SQL
D
LOLCODE
C
Rust
C++
Clojue
Scala
Fortran
B
Brainfuck
Racket
x86 ASM
COBOL
Prolog
BCPL
Scheme
Chef
Haskell
FORTH
APL
Lambda-Calculus
Idris
CPL
Coq
ALGOL 60
ZFC Set Theory
Pi-Calculus
Combinatory Logic
Iota
Malbolge
The Lambda Tessaract

Main programming paradigms

1. Imperative programming
   - *First do this and then do that*
2. Object-oriented programming
   - *Model the world using objects that exchange messages*
3. Functional programming[1]
   - *Evaluate an expression and pass the result*
4. Logic programming[1]
   - *Answer questions by searching a solution*

[1] Part of the Declarative programming paradigm

Main programming paradigms

1. Imperative programming
   - *First do this and then do that*
2. Object-oriented programming
   - *Model the world using objects that exchange messages*
3. Functional programming[1]
   - *Evaluate an expression and pass the result*
4. Logic programming[1]
   - *Answer questions by searching a solution*

[1] Part of the Declarative programming paradigm

Main programming paradigms

1. Imperative programming
   - *First do this and then do that*
2. Object-oriented programming
   - *Model the world using objects that exchange messages*
3. Functional programming[1]
   - *Evaluate an expression and pass the result*
4. Logic programming[1]
   - *Answer questions by searching a solution*

[1] Part of the Declarative programming paradigm

Main programming paradigms

1. Imperative programming
   - *First do this and then do that*
2. Object-oriented programming
   - *Model the world using objects that exchange messages*
3. Functional programming[1]
   - *Evaluate an expression and pass the result*
4. Logic programming[1]
   - *Answer questions by searching a solution*

---

[1] Part of the Declarative programming paradigm

Imperative programming

Example program:

```
read(n);
s = 0;
if (n < 0) {
   print "error";
} else {
  while (n > 0) do {
    s = s + n;
    n = n - 1;
  }
}
```

- Requires a *program state*
- Execution is (incrementally) changing the program state
- Instructions: assignment, decisional, loops
- *First class value*: variable

Imperative programming

Example program:

```
read(n);
s = 0;
if (n < 0) {
   print "error";
} else {
  while (n > 0) do {
    s = s + n;
    n = n - 1;
  }
}
```

- Requires a *program state*
- Execution is (incrementally) changing the program state
- Instructions: assignment, decisional, loops
- *First class value*: variable

Imperative programming

Example program:

```
read(n);
s = 0;
if (n < 0) {
   print "error";
} else {
  while (n > 0) do {
    s = s + n;
    n = n - 1;
  }
}
```

- Requires a *program state*
- Execution is (incrementally) changing the program state
- Instructions: assignment, decisional, loops
- *First class value*: variable

Imperative programming

Example program:

```
read(n);
s = 0;
if (n < 0) {
   print "error";
} else {
  while (n > 0) do {
    s = s + n;
    n = n - 1;
  }
}
```

- Requires a *program state*
- Execution is (incrementally) changing the program state
- Instructions: assignment, decisional, loops
- *First class value*: variable

Imperative programming

Example program:

```
read(n);
s = 0;
if (n < 0) {
   print "error";
} else {
  while (n > 0) do {
    s = s + n;
    n = n - 1;
  }
}
```

- Requires a *program state*
- Execution is (incrementally) changing the program state
- Instructions: assignment, decisional, loops
- *First class value*: variable

Object-Oriented Programming

Example (Java):

```
public class Person {
  private String name;
  Person(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
}

// in the main function
  Driver john = new Driver("John");
  john.startDriving();
  boolean status = john.getStatus();
```

```
public class Driver extends Person
{
  private boolean isDriving;
  Driver(String name) {
    super(name);
    isDriving = false;
  }
  public void startDriving() {
    isDriving = true;
  }
  public boolean getStatus() {
    return isDriving;
  }
}
```

Object-Oriented Programming

- Model using *objects* and *classes*
- Objects exchange messages (dispatch/message passing)
- Abstraction, Encapsulation
- Inheritance, Polymorphism, Overriding, Overloading

Functional programming

Example:
(Haskell)

```
mysum :: Integer -> Integer
mysum n = if (n < 0)
          then 0
          else n + mysum (n - 1)
```

Run:

```
> mysum 10
55
> mysum 0
0
> mysum (-3)
0
```

- Computations = evaluation of mathematical functions
- No state!
- *Non-mutable* values
- First class value: function

Functional programming

Example:
(Haskell)

```
mysum :: Integer -> Integer
mysum n = if (n < 0)
          then 0
          else n + mysum (n - 1)
```

Run:

```
> mysum 10
55
> mysum 0
0
> mysum (-3)
0
```

- Computations = evaluation of mathematical functions
- No state!
- *Non-mutable* values
- First class value: function

Functional programming

Example:
(Haskell)

```
mysum :: Integer -> Integer
mysum n = if (n < 0)
          then 0
          else n + mysum (n - 1)
```

Run:

```
> mysum 10
55
> mysum 0
0
> mysum (-3)
0
```

- Computations = evaluation of mathematical functions
- No state!
- *Non-mutable* values
- First class value: function

Logic Programming

Example (Prolog):

```
man(john).
man(dan).
man(mark).
father(john, mark).
father(mark, dan).

grandfather(X,Y) :- man(X),
 man(Y),father(X,Z),father(Z,Y).
```

Search solution:

```
1 ?- father(mark, dan).
true.
2 ?- grandfather(mark, dan).
false.
3 ?- grandfather(john, dan).
true.
4 ?- grandfather(X, dan).
X = john .
```

- Algorithm = Logic + Control
- Programmer provides a logical specification
- An interpreter searches for the solution using resolution

Logic Programming

Example (Prolog):

```
man(john).
man(dan).
man(mark).
father(john, mark).
father(mark, dan).

grandfather(X,Y) :- man(X),
 man(Y),father(X,Z),father(Z,Y).
```

Search solution:

```
1 ?- father(mark, dan).
true.
2 ?- grandfather(mark, dan).
false.
3 ?- grandfather(john, dan).
true.
4 ?- grandfather(X, dan).
X = john .
```

- Algorithm = Logic + Control
- Programmer provides a logical specification
- An interpreter searches for the solution using resolution

Logic Programming

Example (Prolog):

```
man(john).
man(dan).
man(mark).
father(john, mark).
father(mark, dan).

grandfather(X,Y) :- man(X),
 man(Y),father(X,Z),father(Z,Y).
```

Search solution:

```
1 ?- father(mark, dan).
true.
2 ?- grandfather(mark, dan).
false.
3 ?- grandfather(john, dan).
true.
4 ?- grandfather(X, dan).
X = john .
```

- Algorithm = Logic + Control
- Programmer provides a logical specification
- An interpreter searches for the solution using resolution

Domain Specific Languages

- Web: HTML, CSS, scripting (e.g., Javascript, PHP)
- Databases: SQL
- Algebraic: Maude, CafeOBJ
- Modelling: UML
- WSC: BPEL
- Regex text processing: Perl
- Programming languages: Rascal, Spoofax, **K**, Racket
- ...

Debate

Which paradigm is better?

Indeed, this question is stupid...

Debate

Which paradigm is better?

Indeed, this question is stupid...

Next - what is a programming language?

- PL = Syntax + Semantics
- Syntax:
  - how can we combine symbols to create correct programs?
  - Grammars (BNF = Backus-Naur Form)
- Semantics: what programs *mean*?
- Other things related to PLs: pragmatics, compilers, etc.

Next - what is a programming language?

- PL = Syntax + Semantics
- Syntax:
  - how can we combine symbols to create correct programs?
  - Grammars (BNF = Backus-Naur Form)
- Semantics: what programs *mean*?
- Other things related to PLs: pragmatics, compilers, etc.

Next - what is a programming language?

- PL = Syntax + Semantics
- Syntax:
  - how can we combine symbols to create correct programs?
  - Grammars (BNF = Backus-Naur Form)
- Semantics: what programs *mean*?
- Other things related to PLs: pragmatics, compilers, etc.

Next - what is a programming language?

- PL = Syntax + Semantics
- Syntax:
  - how can we combine symbols to create correct programs?
  - Grammars (BNF = Backus-Naur Form)
- Semantics: what programs *mean*?
- Other things related to PLs: pragmatics, compilers, etc.

Questions?

Bibliography:

- *Chapter 13* from the main course material.
- *Paper:* “Executing Formal Semantics with the K Tool”