Lecture 1: Introduction & Programming Paradigms

V3, 17th January 2025

Programming Languages (H)

Simon Fowler & Michele Sevegnani

Semester 2, 2024/2025

sli.do 4793883

University of Glasgow

sli.do 4793883

Administrivia

- **Course co-ordinators**
  - Simon Fowler simon.fowler@glasgow.ac.uk, SAWB 510c
  - Michele Sevegnani michele.sevegnani@glasgow.ac.uk, S144 Lilybank Gardens
- **Office Hours:**
  - By e-mail appointment
- **GTAs**: Matthew Alan Le Brun & Duncan Lowther
- **Schedule:**
  - **Lectures**: Fridays 11:00 – 13:00, Boyd Orr 222
  - **Labs**: Fridays 15:00 – 16:00 and 16:00 – 17:00, Boyd Orr 720 from Week 2 onwards (not this week!)
- Materials will be posted on Moodle before the lecture / lab

sli.do 4793883

About Me

- Lecturer in Programming Language Foundations here at Glasgow
  - You may remember me from AF2 and FP ☺
- PhD in PL from Edinburgh back in 2019
- Research work on programming language design, particularly around type systems, concurrency, and domain-specific languages
- Happy to discuss UG4/MSci/PhD projects

sli.do 4793883

Teams

- You should have been added to the PL Teams channel
  - The link is also on the Moodle page
- This will be our **main method of communication** with you, so please monitor it
- Please use Teams to ask questions or discuss with your peers
  - We have already created channels for technical support and the coursework
  - We're happy to answer DMs / emails, but consider posting (non-sensitive) questions publicly so your peers can learn too

sli.do 4793883

Assessment

- **Written Exam,** Semester 2 Diet (80%)
  - Online, invigilated, lab-based exam
- **Programming Assignment** (20%)
  - Released W7

sli.do 4793883

Course Aims

This course has two main aims:

- To introduce you to **fundamental concepts** behind the design of programming languages (paradigms, formal specifications, research directions)
- To teach you how to **implement a basic compiler** using **compiler generation tools**

sli.do 4793883

We have changed the course this year

- We **have rewritten the first half of the course**
  - Goal: after this course, you should be able to read a PL paper
  - Much more concentration on fundamental concepts, getting to grips with the essence of programming language design
  - I have taken some inspiration from James Cheney's *Elements of Programming Languages* taught at the University of Edinburgh
- Don't worry, we will add a sample exam and plenty of revision questions for the new material
- Labs begin earlier **(week 2)**, allowing you to get to grips with tools (ANTLR) earlier so that you can practice course concepts

sli.do 4793883

Moodle

- Course resources will be on **Moodle** (let us know if you're not enrolled!)
- Each week:
  - Lecture slides
  - Lecture recordings
  - Lab sheet
- Also: coursework handouts and solutions collected through Moodle

sli.do 4793883

Course Content

- **Programming Language Design (Simon Fowler)**
  - Programming paradigms, syntax, variables and binding, semantics, type systems, imperative programming and the Fun language
- **Programming Language Implementation (Michele Sevegnani)**
  - Parsing, VM code generation, { procedural, data, generic } abstraction, run-time organisation, native code generation

sli.do 4793883

Intended Learning Outcomes

1. Describe the relationships between the **imperative, object-oriented, and functional programming paradigms**;

2. Explain fundamental programming language concepts such as **values and types, variables and lifetime, bindings and scope, procedural abstraction, data abstraction, and generic abstraction**;

3. Explain the distinction between **syntax and semantics**, and read and write syntactic **specifications of programming languages**;

4. Explain the functions **of compilers and interpreters**, and compiler-interpreter interactions such as multi-stage compilers, interpretive compilers, and just-in-time compilers;

5. **Implement a compiler and interpreter** for a simple programming language using compiler-generation tools.

sli.do 4793883

Labs

- Every week is accompanied by an **exercise sheet** that will be released on Mondays (from W2 onwards)
- **Labs** every Friday afternoon
  - Optional but recommended
- Opportunity to do previous week's exercise sheet and get help from course staff
- Later in the course: opportunity to get help with the coursework

sli.do 4793883

Weekly Reflections

- New this year: each week will have an optional ‘weekly reflection’ form on Moodle
- Typical week
  - What did you enjoy most about this week?
  - What did you find most difficult?
  - How did you find the lab sheet? (W2 onwards)
  - Is there something that we could have explained better, or done differently?
  - ...and another silly question, to keep things interesting – best / funniest answers shown at the following lecture
- These will be anonymous but will help us refine future weeks

sli.do 4793883

Coursework Assignment

- In W7 we will release a **coursework assignment**, worth 20% of your course grade
- Will put theory into practice
- There will be a lecture introducing the coursework in W7

...hopefully you will enjoy it!

sli.do 4793883

Feedback

- In PL you will receive several types of feedback:
  - **Individual Written Feedback**: You will receive individual written feedback on your coursework assignment.
  - **Aggregated Feedback:** After the assignment and exam, we will provide aggregated feedback concentrating on common patterns we have noticed over student submissions.
  - **Verbal Feedback in Labs:** The lecturers and demonstrators are happy to provide verbal feedback on any lab solutions.
- We aim to provide grades and feedback to summative assessments three weeks after submission.
- Students are welcome to request any additional feedback explicitly from the course staff.

sli.do 4793883

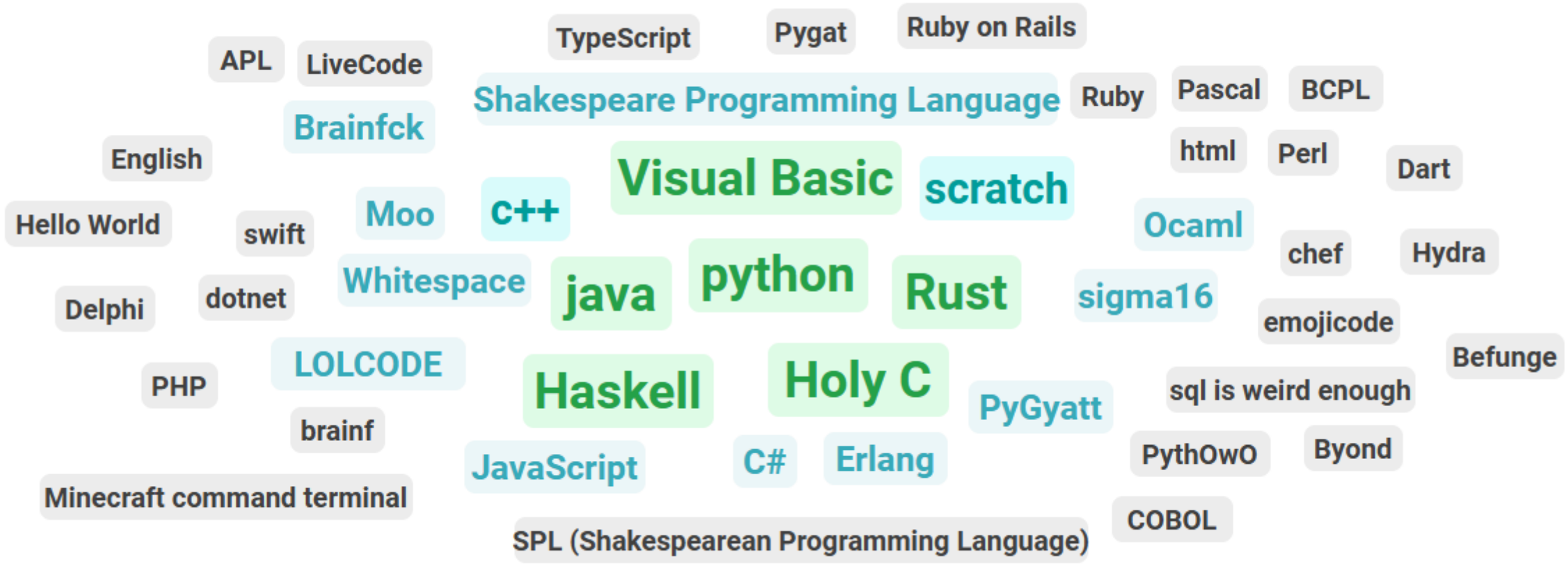
Programming Languages?

- Programming languages allow us to **communicate our intentions** to a computer
- Originally, computers were programmed using **punch cards**
  - Perforated pieces of paper containing binary data
  - Image on the left is of Margaret Hamilton, next to a stack of punch cards that she helped write and were used to power Apollo mission
- Modern day equivalent would be assembly
- PLs give us **abstraction**: concentrate on what we **mean** rather than the low-level details

sli.do 4793883

Sli.do Break: Name any programming language. The weirder the better!

sli.do 4793883



sli.do 4793883

Typical Programming Language Pipeline

- Code in a PL cannot be run directly by a processor!
- Instead, must be **interpreted**, or **compiled** into a different (often more basic) language
  - More on this distinction later

Transform program source code (strings) into a structured representation

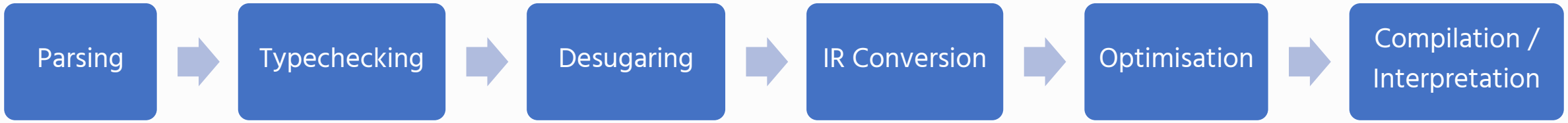
(In a typed language) Check that source code does not have type errors

Either interpret the code, or emit lower—level code (e.g., assembly)

sli.do 4793883

Typical Programming Language Pipeline

- More realistic compilers are typically more complex



- "Compilation" itself has many steps: for example, liveness analysis, register allocation, instruction selection… Michele will talk about this later in the course
- Also often require some **runtime** support (e.g., garbage collection)

sli.do 4793883

Components of a PL Design

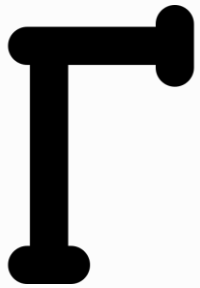
Syntax

{ }

What are the constructs of the language? How do expressions or statements look?

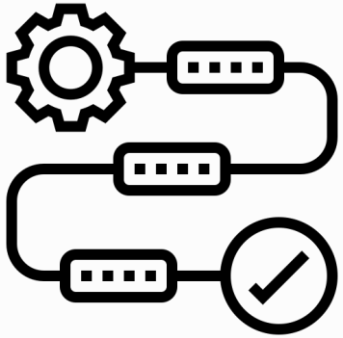
Typing Rules

(if language is typed)



Which programs are sensible? Can we rule out some errors before the program is run?

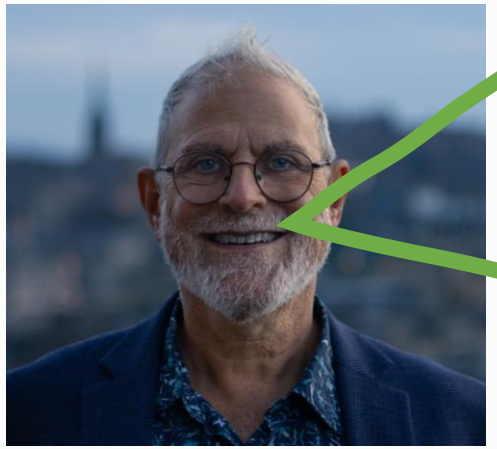
Semantics



What does the program mean? What should it do when it is run?

sli.do 4793883

Aside: Wadler's Law



In any language design, the total time spent discussing a feature in this list is proportional to two raised to the power of its position.

0. Semantics
1. Syntax
2. Lexical syntax
3. Lexical syntax of comments

(That is, twice as much time is spent discussing syntax than semantics, twice as much time is spent discussing lexical syntax than syntax, and twice as much time is spent discussing syntax of comments than lexical syntax.)

sli.do 4793883

Surely it's all been done already!

- You'd think, right?
- Programming languages research is a vibrant and exciting field where concepts and ideas find their way from theory, to practice, and (sometimes!) to the mainstream

Programming languages research is about so much more than parsing!

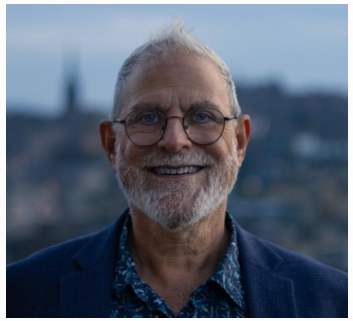
sli.do 4793883

LINEAR LOGIC*

Jean-Yves GIRARD

Équipe de Logique Mathématique, UA 753 du CNRS, UER de Mathématiques, Université de Paris VII, 75251 Paris, France

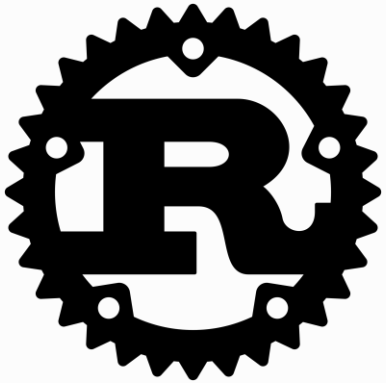
Communicated by M. Nivat
Received October 1986



Linear types can change the world!

Philip Wadler
University of Glasgow*

Several decades of work on affine types, uniqueness types, borrowing etc. later…



sli.do 4793883

Bringing the Web up to Speed with WebAssembly

Andreas Haas Andreas Rossberg Derek L. Schuff* Ben L. Titzer
Google GmbH, Germany / *Google Inc, USA
{ahaas,rossberg,dschuff,titzer}@google.com

Michael Holman
Microsoft Inc, USA
michael.holman@microsoft.com

Dan Gohman Luke Wagner Alon Zakai
Mozilla Inc, USA
{sunfishcode,luke,azakai}@mozilla.com

JF Bastien
Apple Inc, USA
jfbastien@apple.com

Abstract

The maturation of the Web platform has given rise to sophisticated and demanding Web applications such as interactive 3D visualization, audio and video software, and games. With that, efficiency and security of code on the Web has become more important than ever. Yet JavaScript as the only built-in language of the Web is not well-equipped to meet these requirements, especially as a compilation target.

Engineers from the four major browser vendors have risen to the challenge and collaboratively designed a portable low-level bytecode called WebAssembly. It offers compact representation, efficient validation and compilation, and safe low to no-overhead execution. Rather than committing to a device types. By historical accident, JavaScript is the only natively supported programming language on the Web, its widespread usage unmatched by other technologies available only via plugins like ActiveX, Java or Flash. Because of JavaScript's ubiquity, rapid performance improvements in modern VMs, and perhaps through sheer necessity, it has become a compilation target for other languages. Through Emscripten [43], even C and C++ programs can be compiled to a stylized low-level subset of JavaScript called asm.js [4]. Yet JavaScript has inconsistent performance and a number of other pitfalls, especially as a compilation target.

WebAssembly addresses the problem of safe, fast, portable low-level code on the Web. Previous attempts at solving it, from ActiveX to Native Client to asm.js, have fallen short of

sli.do 4793883

Fundamentals are important!

JavaScript: widely-popular language, but to this day must still support very odd scoping and semantics.

(I don't mean to dunk on JavaScript. A lot has been done to fix these issues in the last few years. But a lot of the strangeness could have been prevented through knowledge of PL design fundamentals.)

- Programming language design concepts have been around for **decades**, and show us how to write programming languages with sensible meanings
- Why reinvent the wheel (badly) when the groundwork has been done in a principled way already?

https://www.destroyallsoftware.com/talks/wat

sli.do 4793883

Programming Languages Research, Locally

- Scotland is a hotbed for PL research
  - No less than 7 Scottish universities have PL groups
  - An esteemed history: Haskell, monads, effect handlers, dependent types…
  - Scottish Programming Languages Institute (https://spli.scot) organises community events (seminars, courses, summer schools)

- We have our own PL Research Theme at Glasgow
  - https://tinyurl.com/pl-theme
  - You'd be very welcome to come along to our seminars / join our Team
  - Academics / researchers in the theme all interested in supervising projects across the theory-practice spectrum

sli.do 4793883

Resources (Useful, not required)

- *Practical Foundations for Programming Languages*. Robert Harper, 2016. Cambridge University Press.
- *Types and Programming Languages*. Benjamin C. Pierce, 2002. MIT Press.
- *Programming Language Design Concepts*. David Watt, 2010. Wiley.

- Links to some eBooks on the Moodle page

Programming Paradigms

sli.do 4793883

Programming Paradigms

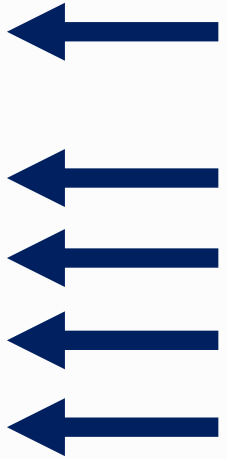
- A **programming paradigm** is a given style of programming that dictates the principles, techniques, and methods used to solve problems in that language
- You often want the **right tool for the job**: SQL is useless for systems programming!
- We will look at the **imperative**, **functional**, and **object-oriented** paradigms, but there are others:
  - Logic programming, concurrent programming, …
- Some PLs (e.g., Scala) are **multi-paradigm**

sli.do 4793883

Imperative Programming Languages

- Imperative language: **Program Counter + Call Stack + State**
- We record our **current position** in the program
- Statements can alter that position (and perform side effects)
- Variable assignments alter some **store**

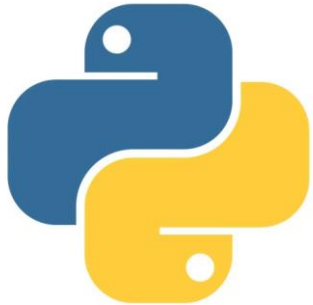
```
def addOne(x):
  return x + 1

xs = []
for x in range(1,3):
  xs.append(addOne(x))
return xs
```

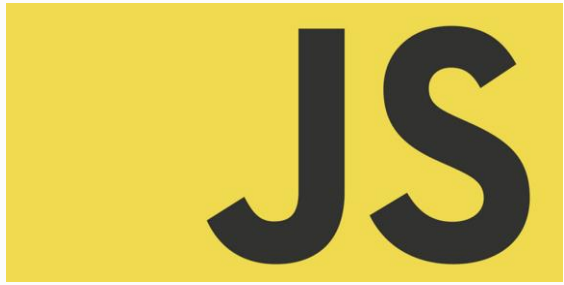


sli.do 4793883

Imperative Programming Languages

- Most of you will program in imperative languages: examples include C, Python, JavaScript…
- Java is also imperative, and also object-oriented (as we'll see in a minute)
- Imperative paradigm **closer to the underlying hardware** as it is built around mutability





sli.do 4793883

Expressions vs. Statements

Statement:

An instruction / computation step

Doesn't return anything

Expression:

A term in the language that eventually reduces to a **value** (e.g., a string, integer, …)

Can be **contained** within a statement

```
def addOne(x):
    return x + 1

xs = []
for x in range(1,3):
    xs.append(addOne(x))
return xs
```

sli.do 4793883

Functional Programming Languages

- Main difference: **everything is an expression**
- Often have **first-class functions**: can create, apply, and pass functions just like any other expression
- Evaluation is **reduction of a complex expression to a value**

```
map addOne [1,2,3]
→ 2 :: (map addOne [2, 3])
→  2 :: 3 :: (map addOne [3])
→ 2 :: 3 :: 4 :: (map addOne [])
= [2, 3, 4]
```

sli.do 4793883

Functional Programming Languages

- Functional programming languages used to be niche, but are now more popular
- You can program in imperative languages in a **functional style** (e.g., by using `const` rather than `var` or `let` bindings in JavaScript)
- Some FP languages (e.g., Haskell / Idris) are **pure**: separation of side-effecting code

sli.do 4793883

Object-Oriented Programming Languages

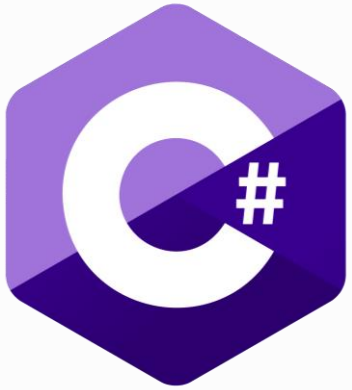
- Key concept is an **object**: consists of some state (fields), and some functions that operate on the state (methods)
- Key concept is **encapsulation**: don't expose internal state unnecessarily
- Also supports **inheritance**: ability to extend previously-defined objects to make new ones

```
public class MyListAdder {

  private List<Int> list;
  public MyListAdder(List<Int> lst) {
    this.list = lst;
  }

  public int addList() {
    int res = 0;
    foreach (int i : list) {
      res = res + i;
    }
    return res;
  }
}
```

sli.do 4793883

Object-Oriented Programming Languages

- OOP is still immensely popular and used heavily in industry
- Many **design patterns** to allow complex functionality in an idiomatic way
- Even in some functional languages (e.g., OCaml), OO concepts can be useful (e.g., for defining visitors)
- Similarly, Java now includes FP concepts (e.g., lambdas, streams)



sli.do 4793883

End of Part 1

- **So far:**
  - **Course Details**
    - Lectures, labs, coursework…
  - **Dipping our toes in the water**
    - What is a programming language?
    - Key components of a PL
    - Modern PL research
    - Programming Paradigms
- **In 10 minutes or so**
  - Syntax: concrete and abstract syntax, regular expressions, BNF grammars

sli.do 4793883

Questions?