University of BRISTOL

Programming and Algorithms II

Lecture 1: Introduction

Tilo Burghardt, Ian Holyer, and Nicolas Wu

tb2935@bristol.ac.uk
ian.holyer@bristol.ac.uk
nicolas.wu@bristol.ac.uk

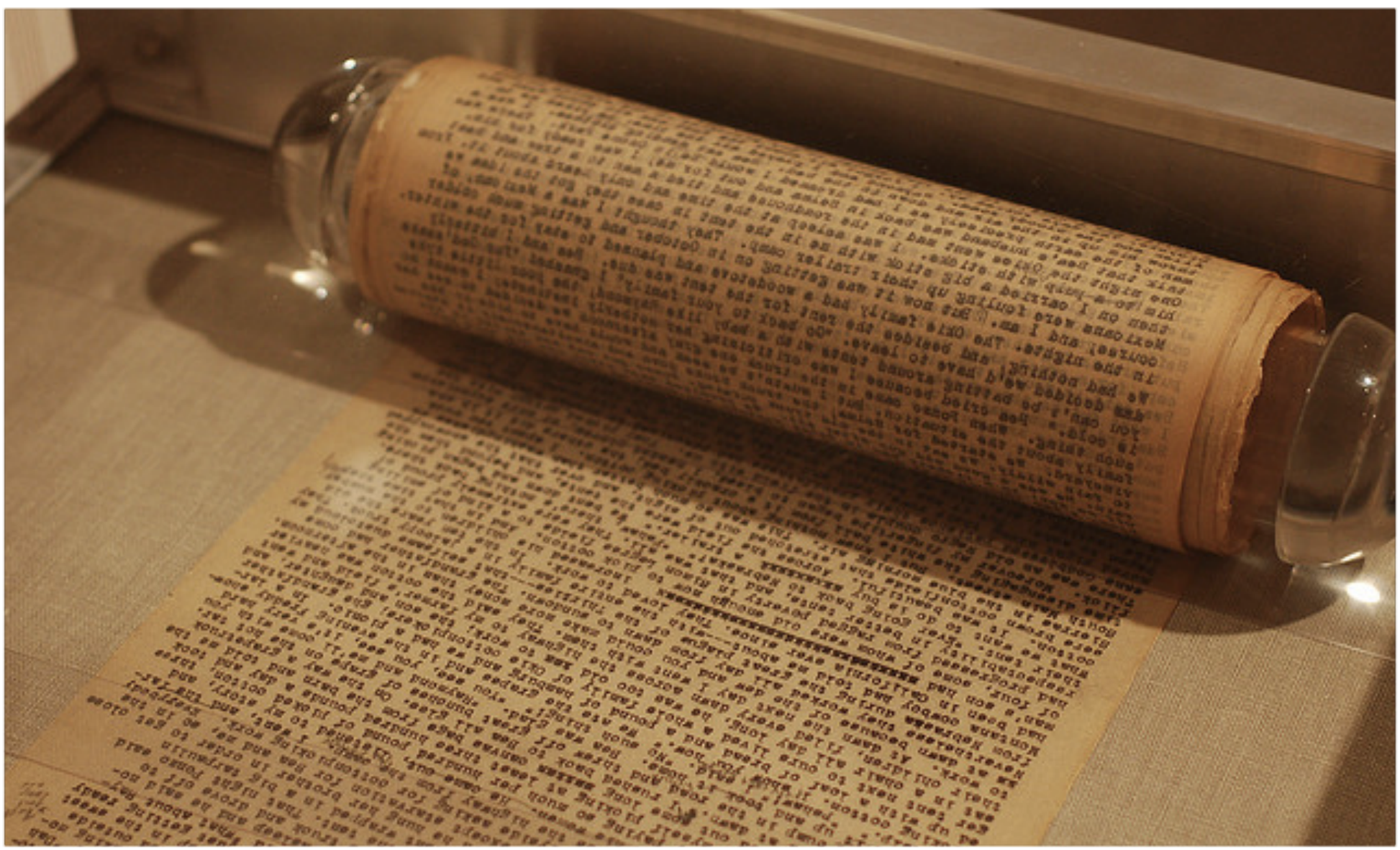
Department of Computer Science
University of Bristol

PANDA II

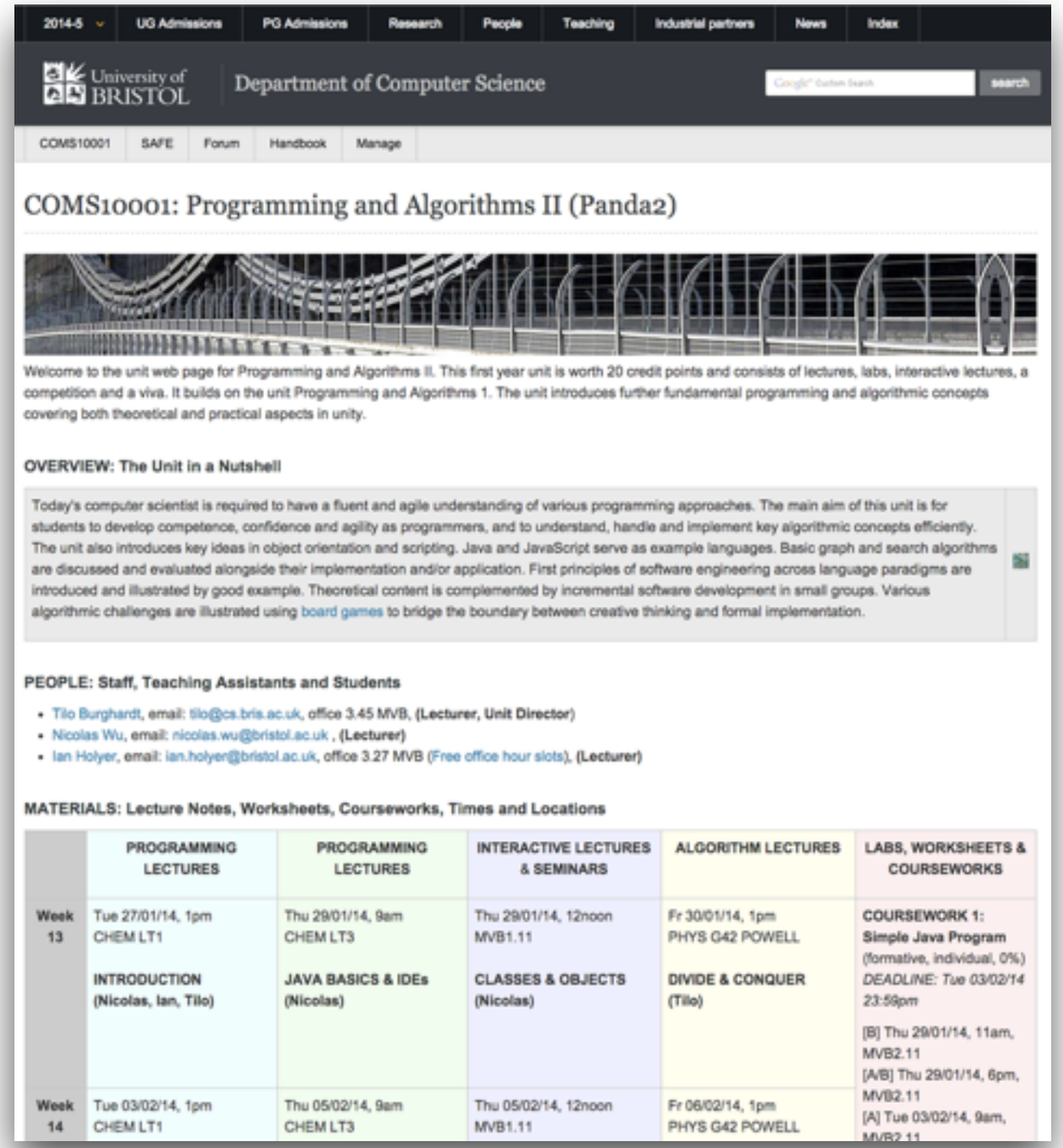
Object-Oriented Programming

Algorithms

Scripting Languages



Course Website



2014-5 | UG Admissions | PG Admissions | Research | People | Teaching | Industrial partners | News | Index

University of BRISTOL | Department of Computer Science | search

COMS10001 | SAFE | Forum | Handbook | Manage

COMS10001: Programming and Algorithms II (Panda2)

Welcome to the unit web page for Programming and Algorithms II. This first year unit is worth 20 credit points and consists of lectures, labs, interactive lectures, a competition and a viva. It builds on the unit Programming and Algorithms 1. The unit introduces further fundamental programming and algorithmic concepts covering both theoretical and practical aspects in unity.

OVERVIEW: The Unit in a Nutshell

Today's computer scientist is required to have a fluent and agile understanding of various programming approaches. The main aim of this unit is for students to develop competence, confidence and agility as programmers, and to understand, handle and implement key algorithmic concepts efficiently. The unit also introduces key ideas in object orientation and scripting. Java and JavaScript serve as example languages. Basic graph and search algorithms are discussed and evaluated alongside their implementation and/or application. First principles of software engineering across language paradigms are introduced and illustrated by good example. Theoretical content is complemented by incremental software development in small groups. Various algorithmic challenges are illustrated using board games to bridge the boundary between creative thinking and formal implementation.

PEOPLE: Staff, Teaching Assistants and Students

- Tilo Burghardt, email: tilo@cs.bris.ac.uk, office 3.45 MVB, **(Lecturer, Unit Director)**
- Nicolas Wu, email: nicolas.wu@bristol.ac.uk , **(Lecturer)**
- Ian Holyer, email: ian.holyer@bristol.ac.uk, office 3.27 MVB (Free office hour slots), **(Lecturer)**

MATERIALS: Lecture Notes, Worksheets, Courseworks, Times and Locations

	PROGRAMMING LECTURES	PROGRAMMING LECTURES	INTERACTIVE LECTURES & SEMINARS	ALGORITHM LECTURES	LABS, WORKSHEETS & COURSEWORKS
Week 13	Tue 27/01/14, 1pm CHEM LT1 **INTRODUCTION (Nicolas, Ian, Tilo)**	Thu 29/01/14, 9am CHEM LT3 **JAVA BASICS & IDEs (Nicolas)**	Thu 29/01/14, 12noon MVB1.11 **CLASSES & OBJECTS (Nicolas)**	Fr 30/01/14, 1pm PHYS G42 POWELL **DIVIDE & CONQUER (Tilo)**	**COURSEWORK 1: Simple Java Program** (formative, individual, 0%) *DEADLINE: Tue 03/02/14 23:59pm* [B] Thu 29/01/14, 11am, MVB2.11 [A/B] Thu 29/01/14, 6pm, MVB2.11
Week 14	Tue 03/02/14, 1pm CHEM LT1	Thu 05/02/14, 9am CHEM LT3	Thu 05/02/14, 12noon MVB1.11	Fr 06/02/14, 1pm PHYS G42 POWELL	[A] Tue 03/02/14, 9am, MVB2.11

https://www.cs.bris.ac.uk/Teaching/Resources/COMS10001/

Practicals


Scotland Yard
+ + + Die spannende Jagd nach Mister X quer durch London + + + Hunting Mister X + + +

Paired Programming

Doodle Poll

http://doodle.com/nezpn8uhaauxhvht

- Find a partner and sign up together on a slot, using both your full names
- First-come-first served
- Poll closes Wednesday 28th January at 4pm
- If you don't sign up, we'll allocate you

Object-Oriented Programming

Guess The Object

You have 20 questions ...

What is an Object?

to be is to do — Socrates

to do is to be — Aristotle

do be do be do — Frank Sinatra

* Actually, nobody knows if Socrates and Aristotle said these things

Object Anatomy

an object can **do** something

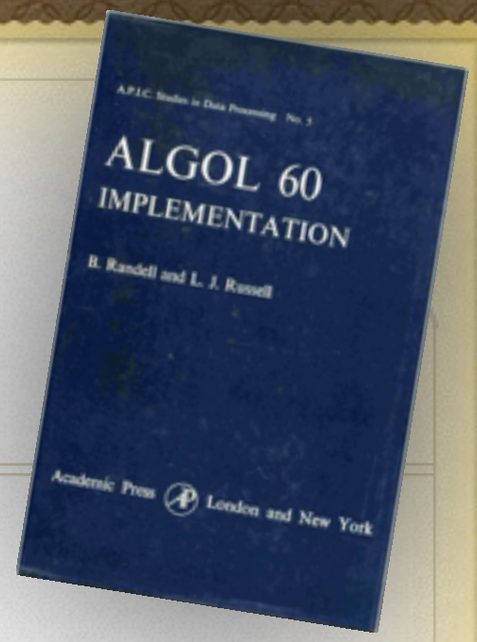
```
scooby.eat(Food food);
scooby.walk();
scooby.talk(String phrase);
```

an object can **be** something

```
scooby.name   = "Scooby";
scooby.gender = Male;
scooby.breed  = "Great Dane";
scooby.legs   = 4;
```

History

Algol 60 [1960]



- Was the standard ACM language for describing algorithms
- Introduced if-then-else, recursion, for-loops and variable scope
- Gave rise to Pascal, Simula, C, and Scheme
- Responsibility of IFIP Working Group 2.1 on Algorithmic Languages and Calculi

Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors — Tony Hoare

Simula [1967]

- 1961: Nygaard was working on simulations and systems analysis. He thinks DSLs are the way forward
- 1962: He pairs up with Dahl, an experienced programmer
- 1963: Together they proposed a simulation language
- Simula was similar to Algol 60, but supports objects with protected state, single inheritance for subtyping and code sharing, partially abstract classes and method overriding

SmallTalk [1980]

- Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace, and others at Xerox PARC
- Untyped, class based. Single inheritance, abstract classes, method overriding

C++ [1983]

- Bjarne Stroustrup at Bell Labs (1983-85)
- Extension of C, originally translated into to C
- Originally a great combination of OO and efficiency

Eiffel [1986]

- Bertrand Meyer at Interactive Software Engineering
- Object-orientation influenced by Simula
- Design by Contract influenced by software verification
- Assertions over objects via pre/post conditions. Run time exceptions when predicate booleans are false.
- Assertions are very powerful when combined with inheritance

Java [1991]

- Established by Sun, named the “Stealth project”
- Aimed at consumer electronics
- Gosling: “the goal was to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other.”
- Platform and CPU independent
- Sun's slogan: “Write Once, Run Anywhere”

The Story Continues ...

- Programming language research is far from over
- We should focus on understanding concepts and paradigms, not technologies and artefacts
- Embrace diversity in languages: innovation is good!
- Choose critically: your tools will shape you!

OOP Languages

Why Java?

Haskell-Java Controversy

To the members of the Budget Council

I write to you because of a rumour of efforts to replace in the introductory programming course of our undergraduate curriculum the functional programming language Haskell by the imperative language Java, and because I think that in this

Finally, in the specific comparison of Haskell versus Java, Haskell, though not perfect, is of a quality that is several orders of magnitude higher than Java, which is a mess (and needed an extensive advertizing campaign and aggressive salesmanship for its commercial acceptance). It is bad enough that, on the whole, in-

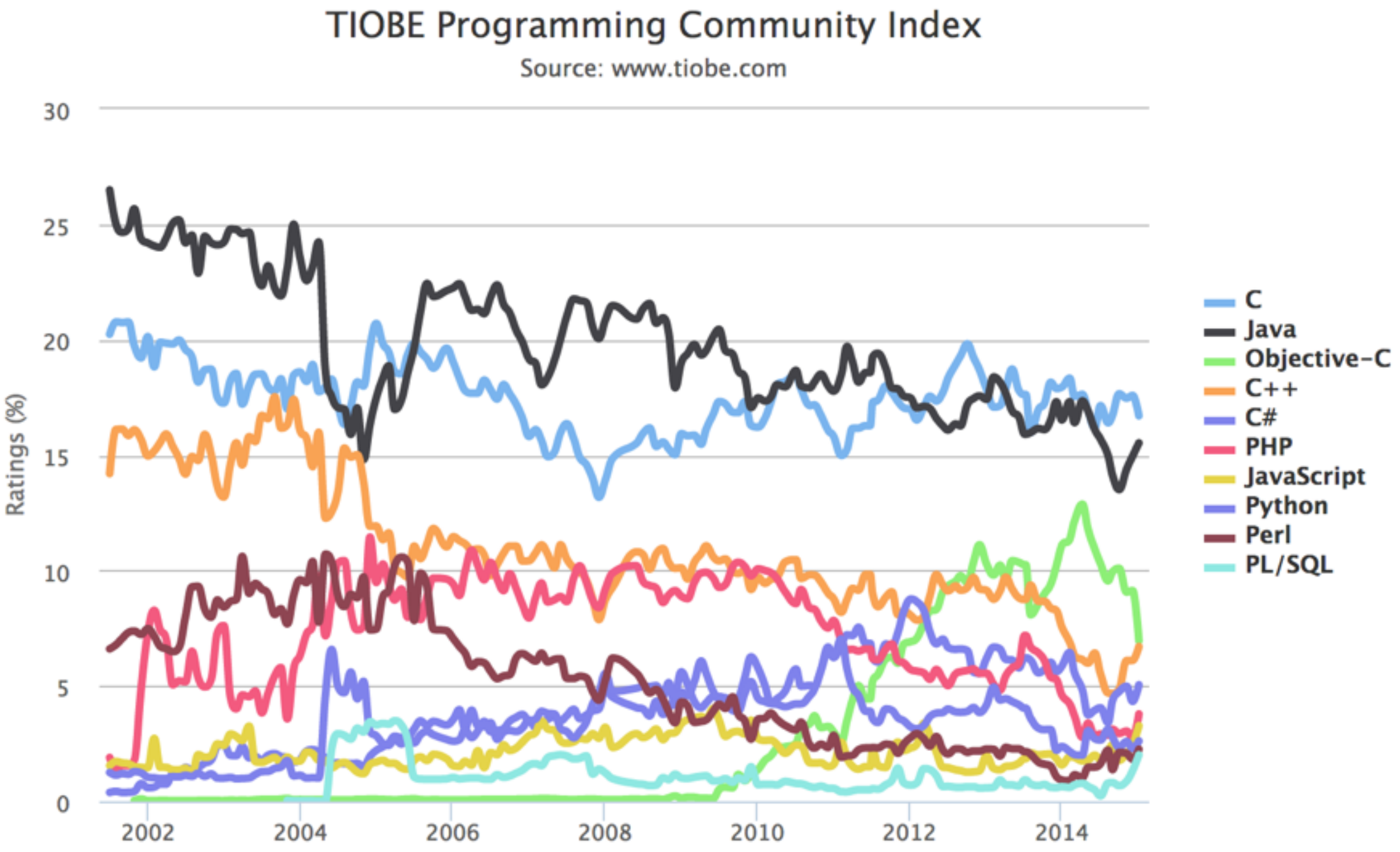
It is not only the violin that shapes the violinist, we are all shaped by the the tools we train ourselves to use, and in this respect programming languages have a devious influence: they shape our thinking habits. This circumstance makes the choice of first programming language so important. One would like to use the intro-

Austin, 12 April 2001

Edsger W. Dijkstra

Object-oriented programming is an exceptionally bad idea which could only have originated in California
— Edsger Dijkstra

Java Usage


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
C
Java
Objective-C
C++
C#
PHP
JavaScript
Python
Perl
PL/SQL

Programming In the Small

- Algorithms
- Correctness
- Efficiency
- Memory

Haskell excels at this stuff!

C excels at this stuff!

Programming In the Large

- Specification
- Requirements
- Design
- Management
- Documentation

This is where Java shines!

Java's Features

- Technical reasons:
  - Object-oriented by design
  - Exception handling
  - Garbage collection
- Non-technical reasons:
  - Platform independent
  - Widely adopted and well integrated
  - Extensive suite of libraries

Why not C++?

Memory Management

- The biggest problem with C++ is memory management
- In C++ object destructors must be called to free up memory when finished
- Memory leaks are all too easy to create
- Java uses *garbage collection* to remove objects that are no longer in use

Goals

Goals

- Extensible and modular code
- Low coupling
  - Relationship between software components should be loose
- High cohesion
  - Responsibilities of a single component should form a meaningful unit

Key Concepts

Concept	Goal	Solution
Abstraction	Code reuse	Classes and interfaces
Inheritance	Code reduction	Subclassing
Polymorphism	Specialised behaviour	Overriding
Encapsulation	Information hiding	Access modifiers
Generics	Exploit structure	Type parameters

Further Reading

Questions?