计算概论A（实验班）

函数式程序设计

胡振江，张伟

信息学院计算机科学技术系

2021年9月15日

授课教师

胡振江

- 1988: 上海交通大学 计算机系 本科毕业
- 1996: 日本东京大学 信息工学 博士学位
- 1997: 日本东京大学 工学部 讲师
- 2000: 日本东京大学 工学部 副教授
- 2008: 日本国立信息学研究所 教授
- 2018: 日本东京大学 信息科学技术学院 教授
- 2019: 北京大学 计算机系 讲席教授

日本工学会会士、ACM杰出科学、IEEE Fellow
日本工程院院士、欧洲科学院院士

研究简介

- Functional Programming (1985-now)
  - **Calculating Efficient Functional Programs**
  - ACM ICFP Steering Committee Co-Chair (2012-2013)
- Algorithmic Languages and Calculi (1992-now)
  - **Parallel programming and Automatic Parallelization**
  - IFIP WG 2.1 Member
- Bidirectional Languages (2003-now)
  - **Bidirectional languages for system/data interoperability**
  - Steering Committee Member of MODELS, ICMT, BX

胡振江

职称：教授

研究所：软件研究所

研究领域：程序设计语言，函数式语言，软件工程，程序演算

办公电话：86-10-62757974

电子邮件：huzj@pku.edu.cn

个人主页：http://sei.pku.edu.cn/~hu

燕园校区：
理科1号楼1247室
昌平新校区：
计算机大楼449室

张伟

职称：副教授

研究所：软件研究所

研究领域：群体软件开发，群体智能系统设计

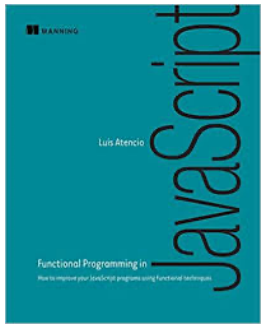
电子邮件：zhangw.sei@pku.edu.cn

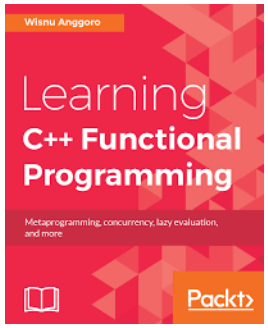
燕园校区：
理科1号楼1803室
昌平新校区：
计算机大楼437室

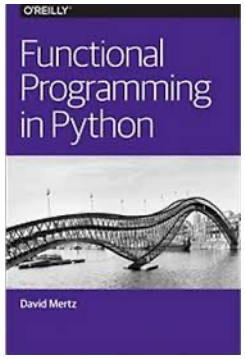
- 助教：谢睿峰
- 电子邮件： xrf@pku.edu.cn

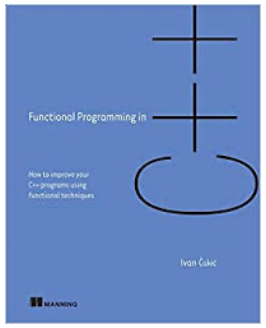
学习函数式程序设计的N个理由

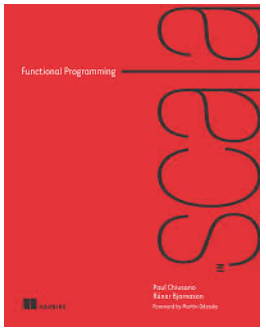
函数式程序设计 … 火了？

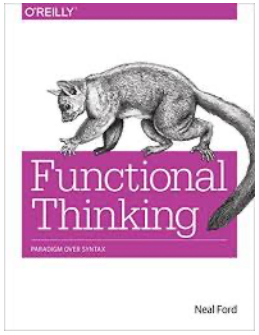


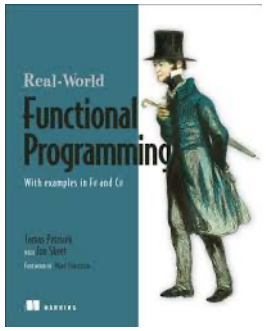


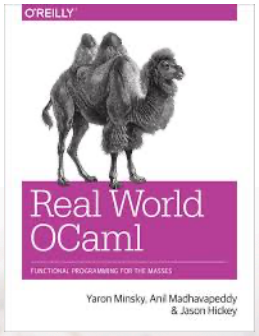


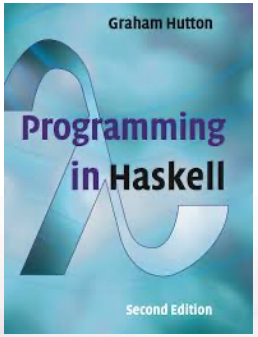


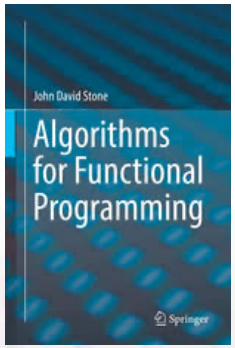


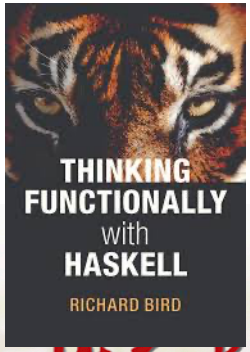








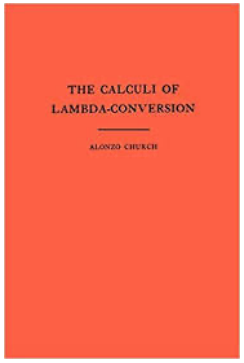




函数式语言: 讨论“计算”的鼻祖语言

Alonzo Church (1903-1995)

Lambda Calculus



Church-Turing Thesis

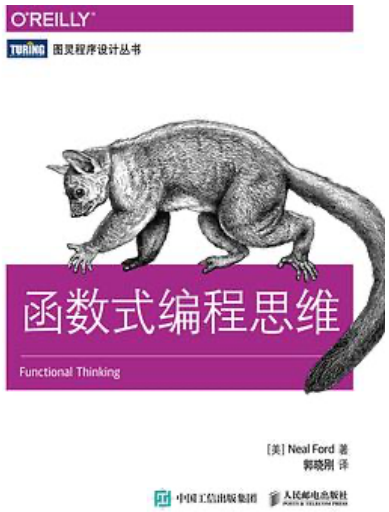
If an algorithm exists, then there is an equivalent Turing Machine or applicable Lambda-function for that algorithm.

John McCarthy (1927-2011)

Father of AI and Lisp

Operators + Notions for Functions
=
Whole Programming Languages

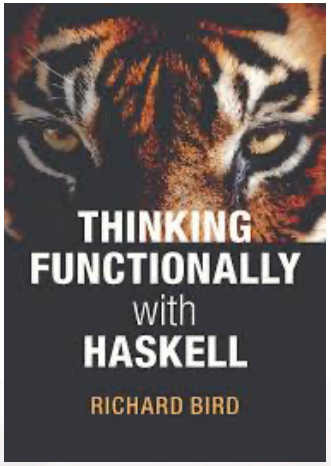
适合现代软件开发的函数式编程思维



复合形: "simple ➔ easy!"

抽象性: 将琐碎的细节交给运行时

简洁透明性: 不是封装不确定因素，
而是减少不确定因素



程序设计

=

正确而简明的实现

+

基于程序推理的优化

很多顶级大学采用的第一门程序设计课

We aim to teach the core principles so that students can quickly grasp any new language that comes along.

- 剑桥大学
- 牛津大学
- 东京大学
- 爱丁堡大学
- Chalmers University of Technology
- University of New South Wales
- Australian National University

Dijstra呼吁在Austin大学继续教授FP

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 case the Budget Council has to take responsibility lest the decision be taken at the wrong level.

You see, it is no minor matter. Colleagues from outside the state (still!) often wonder how I can survive in a place like Austin, Texas, automatically assuming that Texas's solid conservatism guarantees equally solid mediocrity. My usual answer is something like "Don't worry. The CS Department is quite an enlightened place, for instance for introductory programming we introduce our freshmen to Haskell"; they react first almost with disbelief, and then with envy --usually it turns out that their undergraduate curriculum has not recovered from the transition from Pascal to something like C++ or Java.

A very practical reason for preferring functional programming in a freshman course is that most students already have a certain familiarity with imperative programming. Facing them with the novelty of functional programming immediately drives home the message that there is more to programming than they thought. And quickly they will observe that functional programming elegantly admits solutions that are very hard (or impossible) to formulate with the programming vehicle of their high school days.

A fundamental reason for the preference is that functional programs are much more readily appreciated as mathematical objects than imperative ones, so that you can teach what rigorous reasoning about programs amounts to. The additional advantage of functional programming with "lazy evaluation" is that it provides an environment that discourages operational reasoning.

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-

dustry accepts designs of well-identified lousiness as "de facto" standards. Personally I think that the University should keep the healthier alternatives alive.

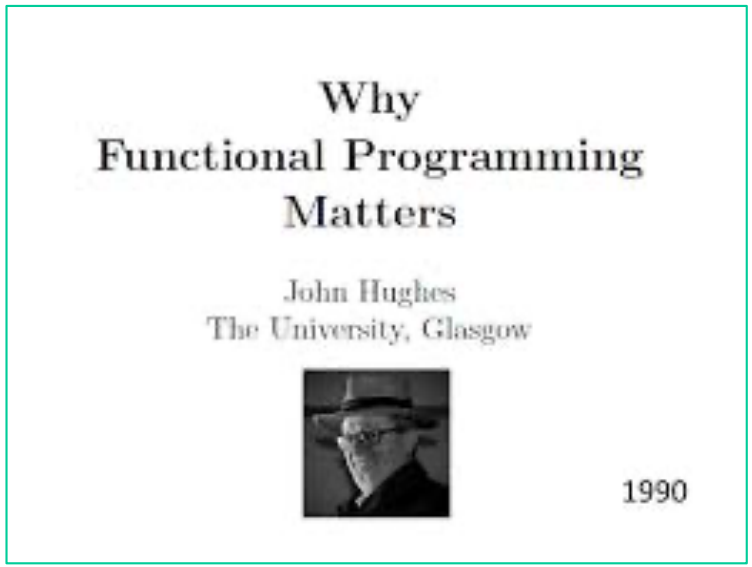
* * *

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 introductory programming course as a means of creating a culture that can serve as a basis for a computing science curriculum, rather than be forced to start that with a lot of unlearning (if that is possible at all: what has become our past, forever remains so). The choice implies a grave responsibility towards our undergraduate students, and that is why it can not be left to a random chairman of something but has to be done by the Budget Council. This is not something that can be left to the civil servants or the politicians; here statesmen are needed.

Austin, 12 April 2001

Edsger W. Dijkstra

FP专家说的理由



Volume 2, Issue 3
September 2015

Article Contents

How functional programming mattered

Zhenjiang Hu ✉, John Hughes, Meng Wang

National Science Review, Volume 2, Issue 3, September 2015, Pages 349–370,
https://doi.org/10.1093/nsr/nwv042
Published: 13 July 2015 **Article history** ▼

PDF Split View Cite Permissions Share ▼

Abstract

In 1989 when functional programming was still considered a niche topic, Hughes wrote a visionary paper arguing convincingly 'why functional programming matters'. More than two decades have passed. Has functional programming really mattered? Our answer is a resounding 'Yes!'. Functional programming is now at the forefront of a new generation of programming technologies, and enjoying increasing popularity and influence. In this paper, we review the impact of functional programming, focusing on how it has changed the way we may construct programs, the way we may verify programs, and fundamentally the way we may think about programs.

Keywords: functional programming, functional languages, equational reasoning, *monad*, high order function

John Hughes: **Why Functional Programming Matters.** Comput. J. 32(2): 98-107 (1989)

Zhenjiang Hu, John Hughes, Meng Wang: **How functional programming mattered.** *National Science Review*, Volume 2, Issue 3, September 2015, Pages 349–370

课程目的

课程目的

- 通过教授函数式程序设计的基本概念（例如递归，抽象，高阶函数和数据类型，程序推理）并说明其实际使用，使学生能够
  - 熟悉程序设计的基本技术和函数式思维方式
  - 理解“计算”的基本概念
  - 培养学生对程序设计问题的分析和解决能力

选课的同学要求具备良好的数学基础
有一定的编程经验

教学内容

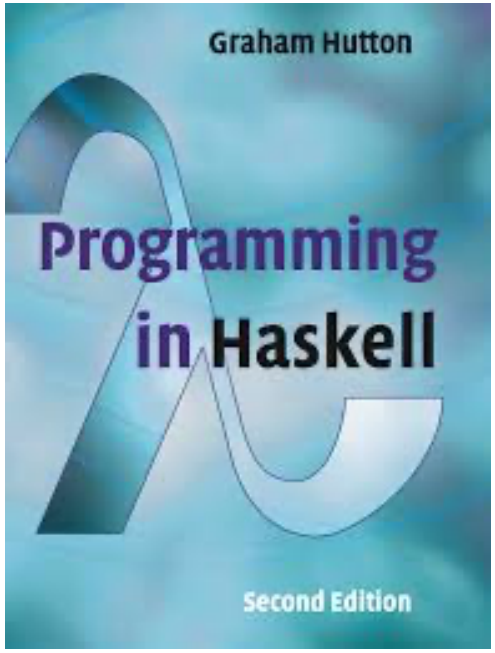
主要包括三部分

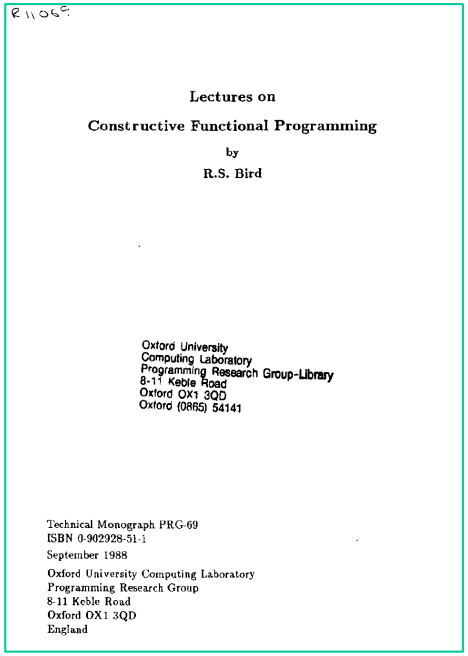
- Haskell 函数式程序设计: 约11次课
  - 类型，函数定义，递归定义，高级函数，类型定义，描述副作用的函数，惰性计算，各种应用

- （基于Agda的）程序推理与演算： 约11次课
  - 程序的结构化
  - 程序推理与演算理论
  - 程序推导与程序综合

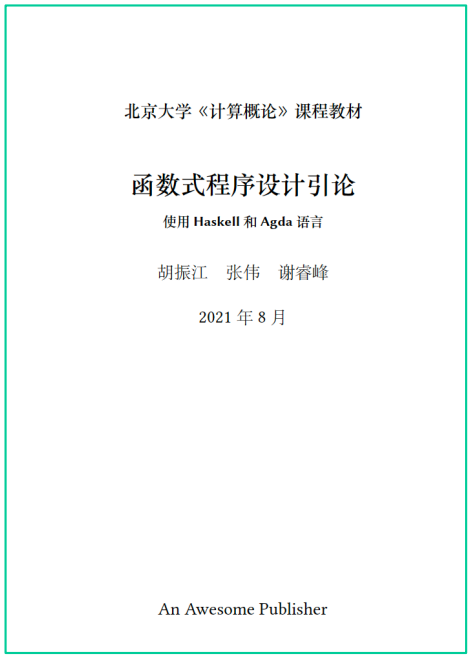
- 计算概论大课: 7次课

主要教材




Lectures on
Constructive Functional Programming
by
R.S. Bird

Technical Monograph PRG-69
ISBN 0-902928-51-1
September 1988
Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England


北京大学《计算概论》课程教材

函数式程序设计引论

使用 Haskell 和 Agda 语言

胡振江　张伟　谢睿峰

2021 年 8 月

An Awesome Publisher

Graham Hutton, Programming in Haskell, Cambridge University Press, 2016.

Richard Bird, Lecture Notes on Constructive Functional Programming, Technical Monograph PRG-69, Oxford University, 1988.

评分标准

评分标准

- 平时：30分
- 期中：30分
- 期末：40分

- 期中的形式未定，期末是考试。