Course Introduction

2020 / 9 / 24

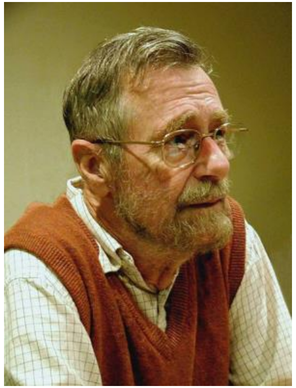
Slides adapted from Berkeley CS61a

What is Computer Science?

- What problems can be solved using computation?
- How to solve those problems?
- What techniques lead to effective solutions?

Computer Science is no more about computers than astronomy is about telescopes.

Edsger W. Dijkstra



What is Computer Science?

- Systems
- Artificial Intelligence
- Graphics
- Security
- Networking
- Programming Languages
- Theory
- Scientific Computing

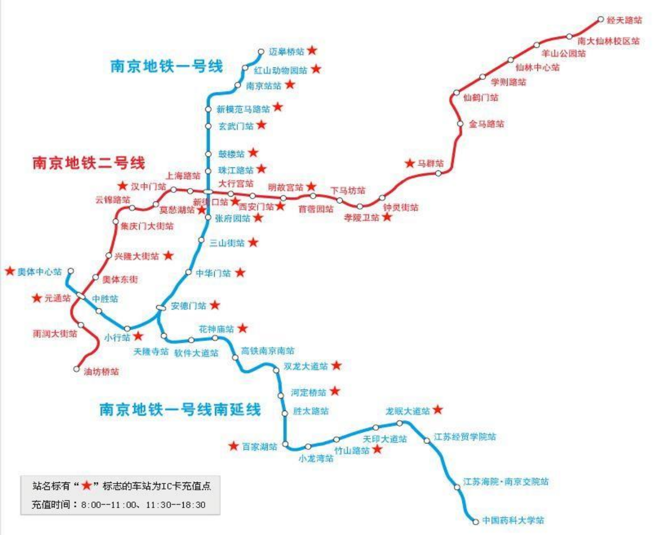
...

What is this course about?

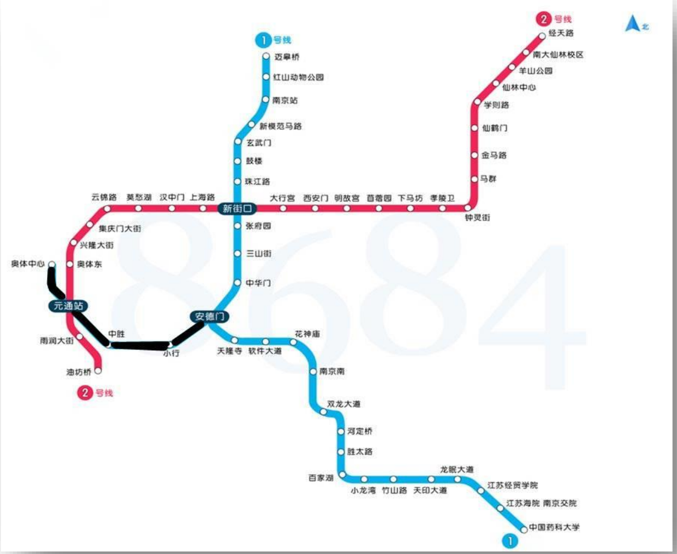
- Introduction to Programming
  - Full understanding of Python fundamentals
  - Combining multiple ideas in large projects
  - How computers interpret programming languages
  - More …

What is this course about?

- Introduction to Programming
- Managing Complexity
    - Mastering **Abstraction**



站名标有“★”标志的车站为IC卡充值点
充值时间：8:00--11:00、11:30--18:30



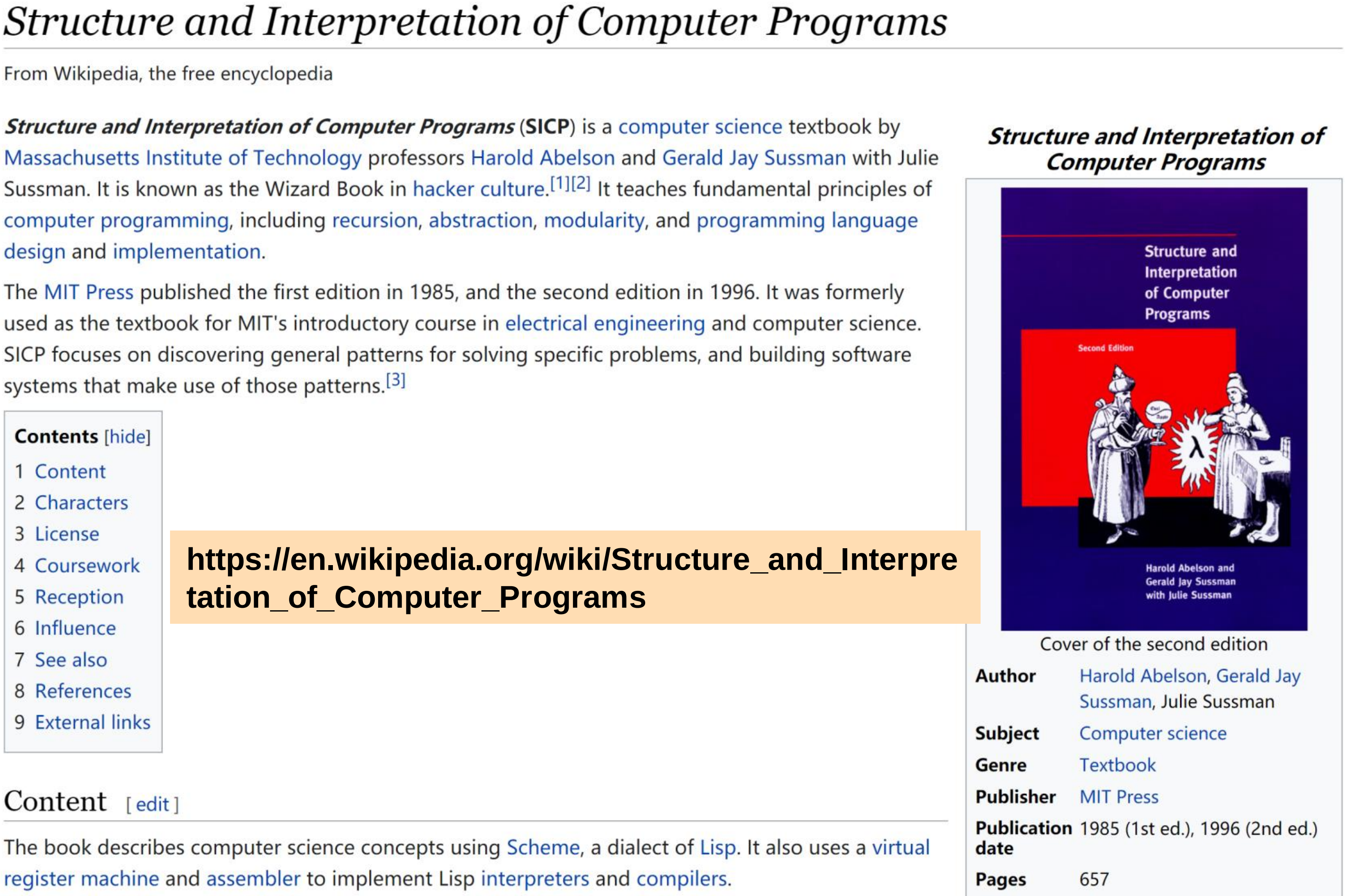
What is this course about?

- Introduction to Programming
- Managing Complexity
  - Mastering Abstraction
  - Programming Paradigms

- A challenging course that will demand a lot from you

Alternative to this course

- 程序☒ ☒ 基☒
    - Programming in C
    - Similar goals, different textbooks and languages



Structure and Interpretation of Computer Programs

From Wikipedia, the free encyclopedia

Structure and Interpretation of Computer Programs (**SICP**) is a computer science textbook by Massachusetts Institute of Technology professors Harold Abelson and Gerald Jay Sussman with Julie Sussman. It is known as the Wizard Book in hacker culture.[1][2] It teaches fundamental principles of computer programming, including recursion, abstraction, modularity, and programming language design and implementation.

The MIT Press published the first edition in 1985, and the second edition in 1996. It was formerly used as the textbook for MIT's introductory course in electrical engineering and computer science. SICP focuses on discovering general patterns for solving specific problems, and building software systems that make use of those patterns.[3]

Contents [hide]

Content [edit]

The book describes computer science concepts using Scheme, a dialect of Lisp. It also uses a virtual register machine and assembler to implement Lisp interpreters and compilers.

Structure and Interpretation of Computer Programs

Structure and Interpretation of Computer Programs
Second Edition
λ
Harold Abelson and Gerald Jay Sussman with Julie Sussman

Cover of the second edition

Author	Harold Abelson, Gerald Jay Sussman, Julie Sussman
Subject	Computer science
Genre	Textbook
Publisher	MIT Press
Publication date	1985 (1st ed.), 1996 (2nd ed.)
Pages	657

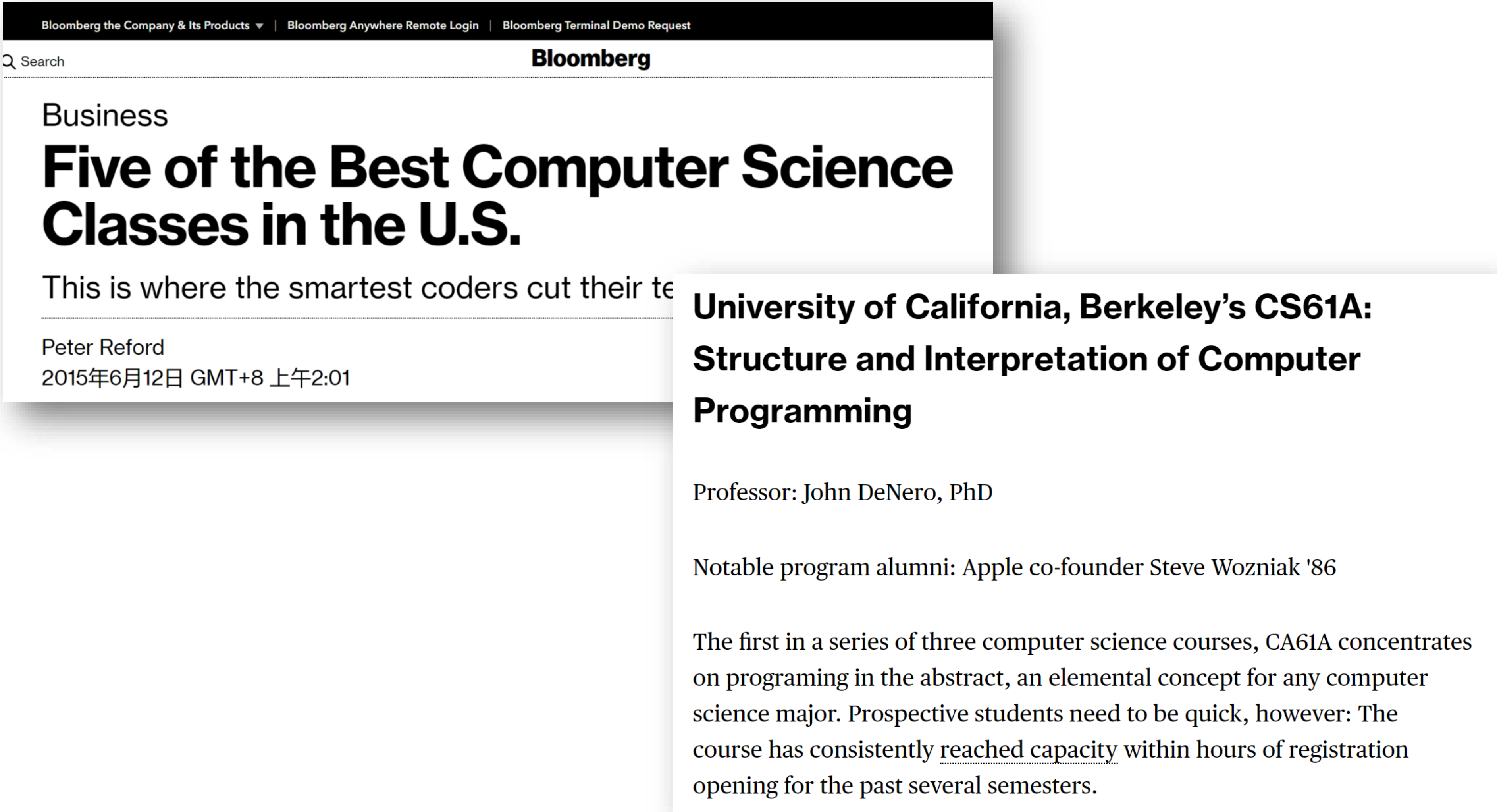
https://en.wikipedia.org/wiki/Structure_and_Interpretation_of_Computer_Programs

This Course: A Clone of BerkeleyCS61A

https://cs61a.org/

- 教材： Composing Programs，SICP的Python版
  - https://composingprograms.com/
- 全美最受欢迎的5门计算机课程之一

https://cs61a.org/resources.html#advice



Bloomberg the Company & Its Products ▾ | Bloomberg Anywhere Remote Login | Bloomberg Terminal Demo Request
Search
Bloomberg

Business

Five of the Best Computer Science Classes in the U.S.

This is where the smartest coders cut their te

Peter Reford
2015年6月12日 GMT+8 上午2:01

University of California, Berkeley's CS61A: Structure and Interpretation of Computer Programming

Professor: John DeNero, PhD

Notable program alumni: Apple co-founder Steve Wozniak '86

The first in a series of three computer science courses, CA61A concentrates on programing in the abstract, an elemental concept for any computer science major. Prospective students need to be quick, however: The course has consistently reached capacity within hours of registration opening for the past several semesters.

Course Format

Lecture	Th/F 14:00 -15:50，仙I-106
Lab section	Th 16:00 – 18:00，基▯ ▯ ▯ 楼 乙124
Office hours	Th 19:00 – 21:00
Course webpage	https://cs.nju.edu.cn/xyfeng/teaching/SICP
Online textbook	https://composingprograms.com

- homework assignments
- programming projects
- A midterm and a final
- Lots of course support

Grading

- Homework, 15%

Homeworks

- Will be graded on “effort”
- This approximately means, completing most of the problems and at least attempting to solve the rest
- This means there’s no reason to cheat!
- Ask for help if you are stuck and make a good effort on all of the homework

Grading

- Homework, 15%
- Labs, 10%
  - Graded on correct completion
  - Need to complete in the lab section
- Projects, 25%

Projects

- Will be graded on correctness and composition
- Several of the programming projects will be partnered
- Larger than homeworks

Grading

- Homework, 15%
- Labs, 10%
- Projects, 25%
- Midterm, 25%
- Final, 25%

Collaboration

- We **highly** encourage discussing / sharing ideas with each other
- **Limitations**
  - Do not share code
  - The only circumstance in which a student should be looking at another student's code is if they are project partners

Questions?

What is programming about, really?

Expressions

Types of Expressions

An expression describes a computation and evaluates to a value

$$18 + 45$$

$$\boxed{f(x)}$$

$$\frac{6}{23}$$

$$\binom{45}{18}$$

$$\sqrt{2323478}$$

$$2^{100}$$

$$\sin \pi$$

$$\sum_{i=1}^{100} i$$

$$|-1253|$$

$$\lim_{x \to \infty} \frac{1}{x}$$

$$7 \bmod 2$$

$$\log_2 1024$$

Call Expressions in Python

All expressions can use function call notation

Demo

Anatomy of a Call Expression

$$\underbrace{\texttt{add}}_{\text{Operator}}\ \texttt{(}\ \underbrace{\texttt{2}}_{\text{Operand}}\ \texttt{,}\ \underbrace{\texttt{3}}_{\text{Operand}}\ \texttt{)}$$

Operators and operands are also expressions

Evaluation of a Call Expression

$$\underbrace{\texttt{add}}_{\text{Operator}}\ \texttt{(}\ \underbrace{\texttt{2}}_{\text{Operand}}\ \texttt{,}\ \underbrace{\texttt{3}}_{\text{Operand}}\ \texttt{)}$$

1. Evaluate
   a. Evaluate the operator subexpression
   b. Evaluate each operand subexpression
2. Apply
   a. Apply the value of the operator subexpression to the values of the operand subexpression

```
add(add(6, mul(4, 6)), mul(3, 5))
```

Humans

We like to inside inside-out

```
add(add(6, mul(4, 6)), mul(3, 5))
add(add(6,    24    ), mul(3, 5))
add(add(6,    24    ), mul(3, 5))
add(        30       , mul(3, 5))
add(        30       , mul(3, 5))
add(        30       ,     15   )
add(        30       ,     15   )
                45
```

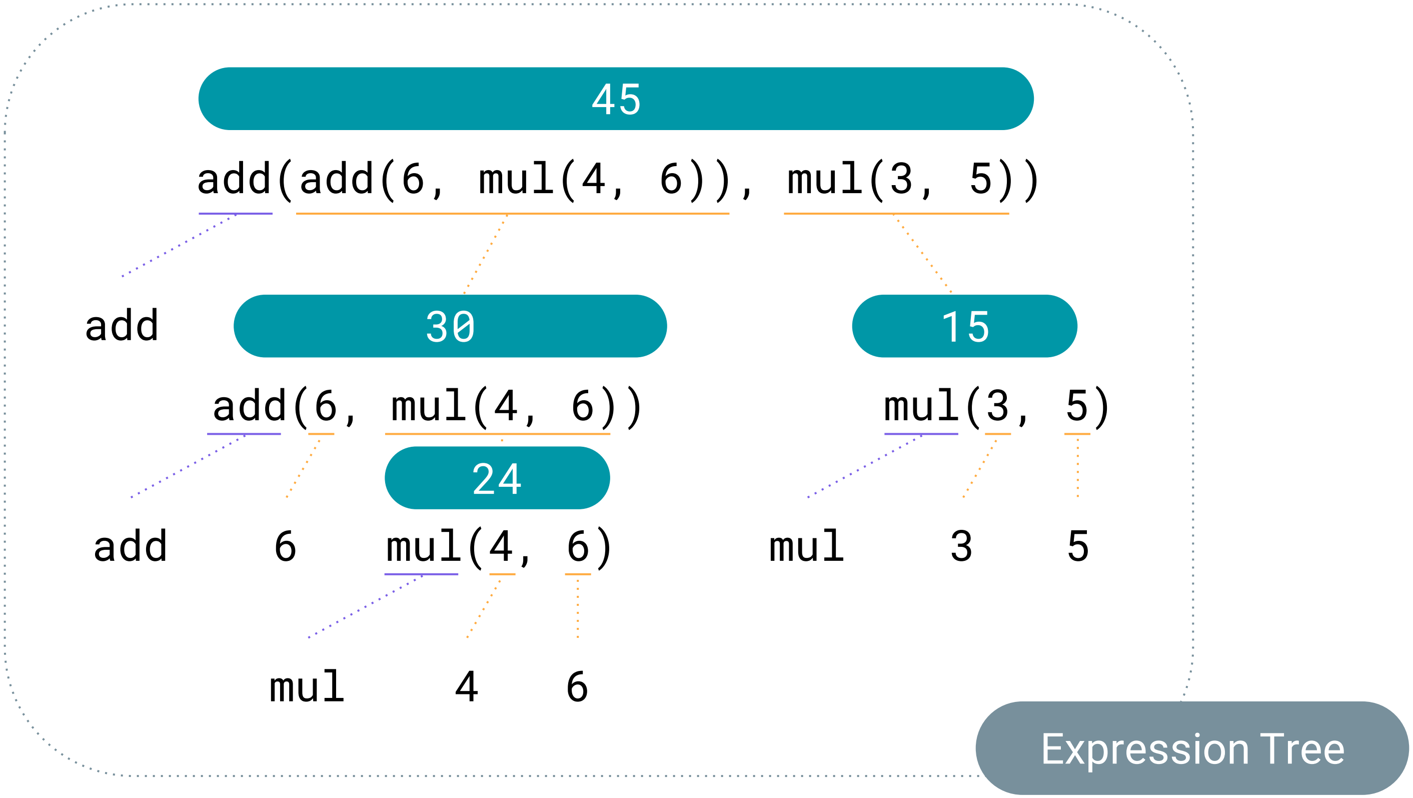
Python can’t jump around in the same way we do

Nested Call Expression

1 Evaluate operator

2 Evaluate operands

3 Apply!



45

add(add(6, mul(4, 6)), mul(3, 5))

add

30

15

add(6, mul(4, 6))

mul(3, 5)

24

add

6

mul(4, 6)

mul

3

5

mul

4

6

Expression Tree

Functions, Values, Objects, Interpreters and Data

Demo