Shrinking JavaScript for an SICP-based First-Year Course

Martin Henz

joint work with Boyd Anderson, Kok-Lim Low, and Daryl Tan

National University of Singapore (NUS)

https://www.comp.nus.edu.sg/~henz/publications/pdf/Shrinking_JavaScript_Slides.pdf

Overview

- Motivation for using SICP and JavaScript
- Shrinking JavaScript for SICP
- Source Academy and Source Academy @ NUS
- Outcomes so far
- Outlook: Global scaling of experiential learning with SICP

But first…

…thanks to Jerry and Hal for hosting me here, to Julie for her kind support, and to the CSAIL and the MIT administration for making my visit happen! Thanks to Cynthia Rosenthal from CSAIL for organizing this seminar.

Motivation

About me

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

- Training: Programming language design and implementation
- PhD in 1997: Objects in Oz, Concurrent Constraint Programming
- Research in discrete optimization, tournament scheduling (ACC97/98)
- Co-founded Workforce Optimizer Pte Ltd in 2001
- Teaching programming language design and implementation at NUS since 1997
- “Discovered” experiential learning in the 2000s and 2010s

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Background of SICP JS

- 1970s-90s: Hal Abelson and Gerald Jay Sussman introduce principled CS1 education with **S**tructure and **I**nterpretation of **C**omputer **P**rograms
- 1997: NUS adopts SICP in a small opt-in course called CS1101S

Why do we (still) use SICP for CS1 at NUS?

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Students benefit from SICP's emphasis on:

- Mental models for computation
- Programming as "communicating computational processes"
- Functional-programming-first approach
- "Roll your own language"

…as opposed to:

- Learning a particular programming language
- Solving problems using programming
- Software development

Studios with at most 8 students

Studios with at most 8 students (plus their “Avenger”)

Conversion of CS1101S to JavaScript

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

- 2008: MIT moves away from SICP and 6.001
- 2008: JavaScript adaptation of SICP starts
- 2012: CS1101S converts from Scheme to JavaScript
- 2015: EcmaScript 2015 enables full adaptation of SICP to JavaScript
- 2018: CS1101S gets adopted for all CS first-year students

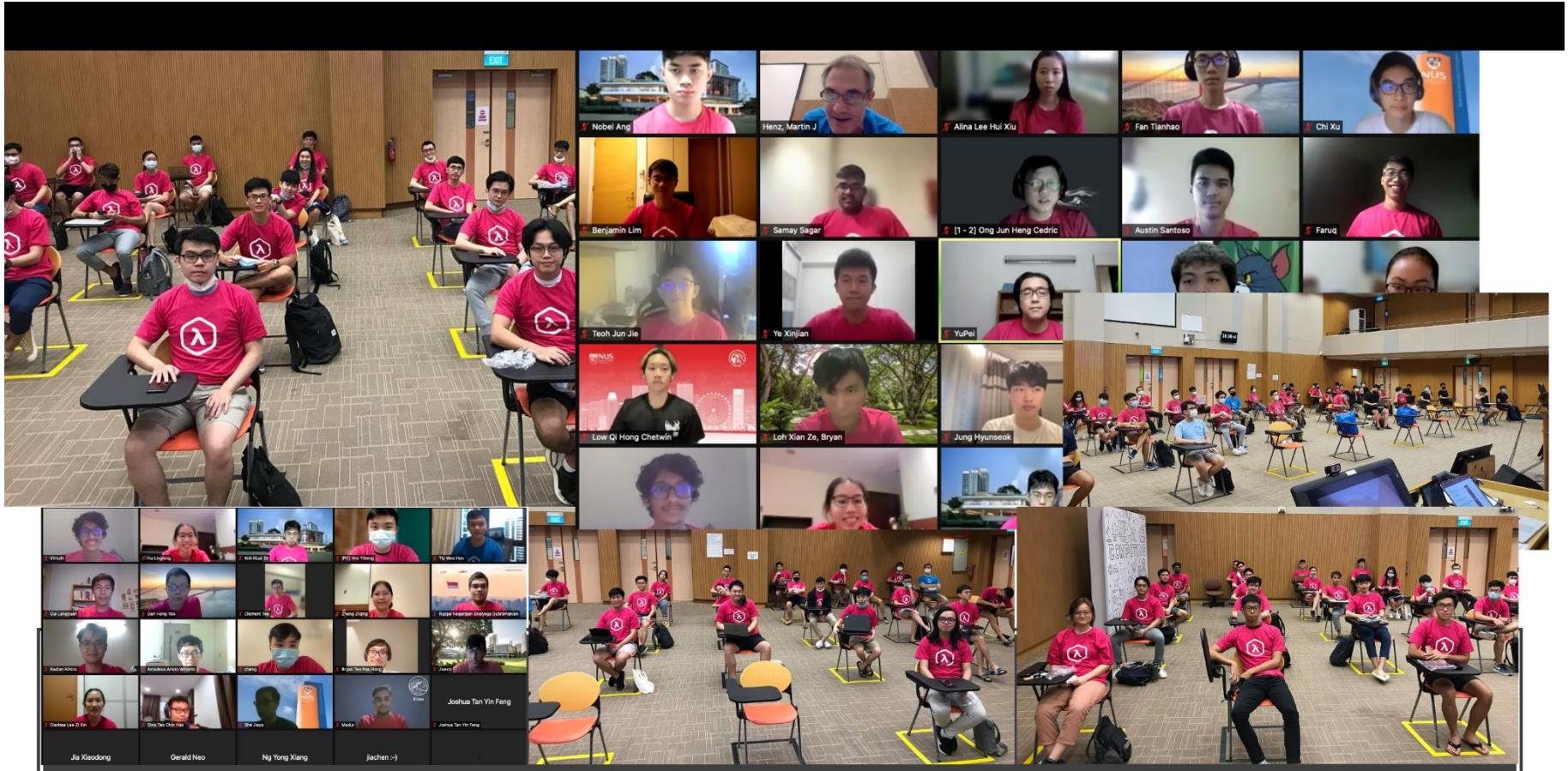
What did we get ourselves into?

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

- The task: scaling from 120 student in 2017
  420 students in 2018
- First challenge: How to keep group size of 8 students?
- Our asset: a core group of dedicated Avengers who volunteered to help in recruiting 50+ new Avengers
- Funding?

2018: 412 students, 55 Avengers

2021: 667 students, 88 Avengers



Shrinking JavaScript

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

- Second challenge: How to **manage**
  Avengers and students, and grade assessments?
- Our asset: the core group of Avengers volunteered to
  build a system for teaching CS1101S that we called “Source Academy”
- Guiding principle: KISS: JavaScript is too big for us: **we need to shrink it!**

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

What did we mean by *shrinking* JavaScript?

- We *force* students to use very small JavaScript *sub*languages
- Language features not in sublanguages are *not available in our implementation*

Similar to approaches in teaching PL/I, DrScheme, Racket, Grace

For references, see “Shrinking JavaScript for CS1” SPLASH-E 2021

Why *shrink* the CS1 language?

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

- Lower the barrier of entry
- Focus on learning objectives
- Simplify implementation of tools

Examples:

```
if (test(x) === true) { … } else { … }
if (test(x)) { … } else { … }
```

bad: is not in first sublanguage
good (if `test` returns boolean)

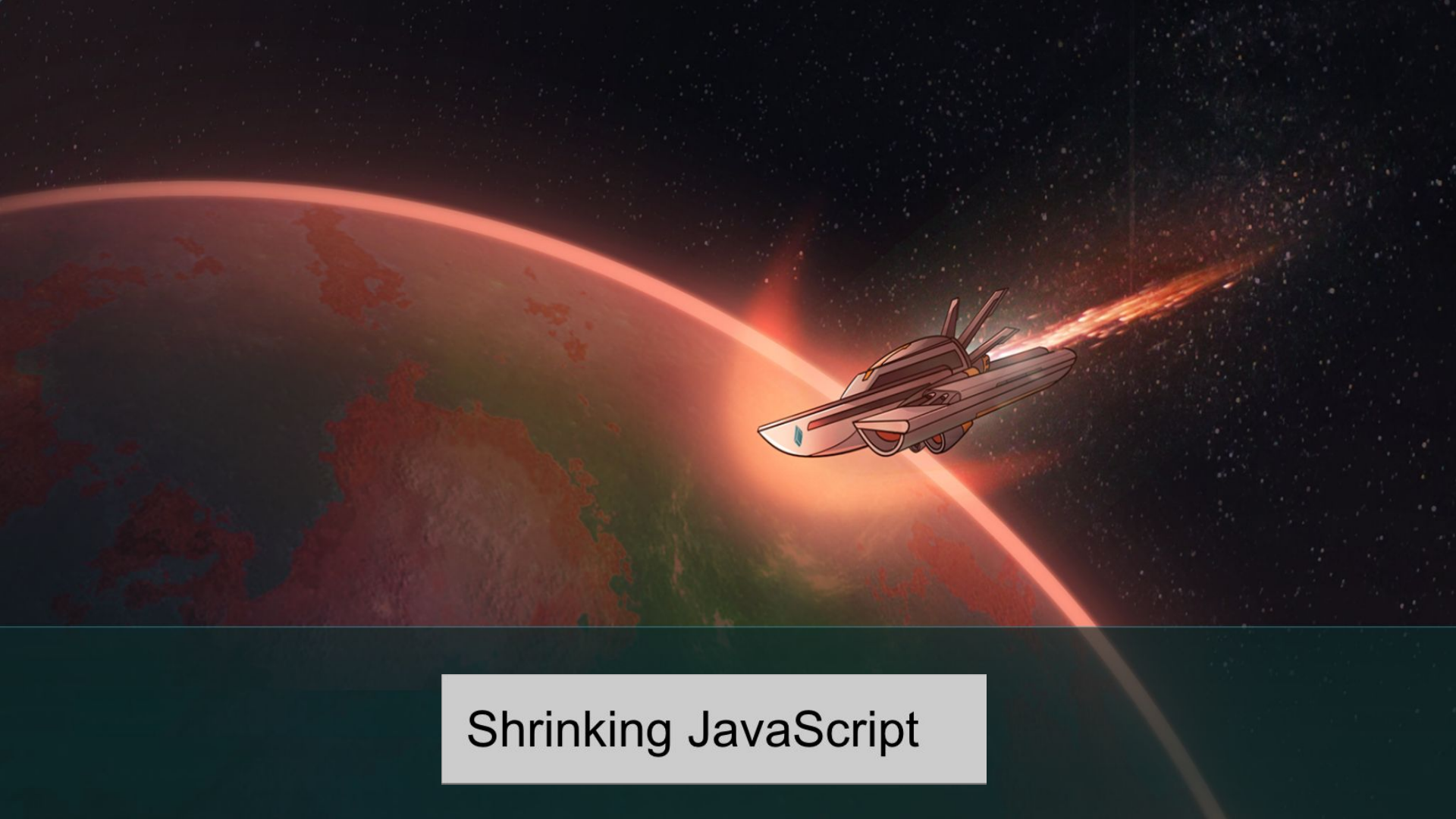
JavaScript's == operator is weird
⇒ Our JavaScript sublanguages do not have ==

OOP not introduced in our CS1
⇒ Our JavaScript sublanguages do not have OOP

SICP JS book project

- Third challenge: How to communicate course content effectively in a team of ~100 persons in total?
- Solution: get serious about adapting SICP to JavaScript
- Key assets: Tobias Wrigstad who visited NUS on a teaching sabbatical in 2019, and Julie Sussman, who got involved as MIT Press editor in August 2020
- Result: SICP JavaScript Edition

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook



Shrinking JavaScript

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Language progression in our CS1 course

- Source §1: JavaScript sublanguage for SICP JS Chapter 1
  - Lambda calculus plus statements, primitive values, explicit recursion
- Source §2: for SICP JS Chapter 2
  - Source §1 plus pairs
- Source §3: for SICP JS Chapter 3
  - Source §2 plus variables and assignment (our CS1 course also adds arrays and loops)
- Source §4: for SICP JS Chapter 4
  - Source §3 plus a parse function

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Source §1

program	::=	*statement* ...	program
statement	::=	**const** *name* **=** *expression* **;**	constant declaration
	|	**function** *name* **(** *names* **)** *block*	function declaration
	|	**return** *expression* **;**	return statement
	|	*if-statement*	conditional statement
	|	*block*	block statement
	|	*expression* **;**	expression statement
if-statement	::=	**if (** *expression* **)** *block*	
		else (*block* | *if-statement*)	conditional statement
block	::=	**{** *statement* ... **}**	block statement

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Source §1 (continued)

expression	::=	*number* | **true** | **false** | *string*	primitive literal expression
	|	*name*	name expression
	|	*expression binary-operator expression*	binary operator combination
	|	*unary-operator expression*	unary operator combination
	|	*expression* **(** *expressions* **)**	function application
	|	(*name* | **(** *names* **)**) **=>** *expression*	lambda expression (expression body)
	|	(*name* | **(** *names* **)**) **=>** *block*	lambda expression (block body)
	|	*expression* **?** *expression* **:** *expression*	conditional expression
	|	**(** *expression* **)**	parenthesised expression
binary-operator	::=	**+** | **-** | ***** | **/** | **%** | **===** | **!==**	
	|	**>** | **<** | **>=** | **<=** | **&&** | **||**	binary operator
unary-operator	::=	**!** | **-**	unary operator
expressions	::=	ϵ | *expression* (**,** *expression*) ...	argument expressions

Some fun with Source §1

Runes: https://share.sourceacademy.org/rightsplit

Curves: https://share.sourceacademy.org/funwithcurves

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Source §2

- Add primitive expression `null` for empty list (Scheme's `nil`)
- Add `pair`, `head`, `tail` (Scheme's `cons`, `car`, `cdr`)
- Add library for list processing (map/reduce/filter)

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Some fun with Source §2

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Functional audio processing: https://share.sourceacademy.org/echo

Sound contest 2019 winner: https://share.sourceacademy.org/0iz2g

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Source §3

- Required by SICP:

statement ::= ...
| **let** *name* = *expression* ; variable decl.

expression ::= ...
| *name* = *expression* variable assgmt

- Required by our CS1:
  - while loops, for loops
  - Arrays:

expression ::= ...
| *expression*[*expression*] array access
| *expression*[*expression*] = *expression* array assignment
| [*expressions*] literal array expression

Some fun with Source §3

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

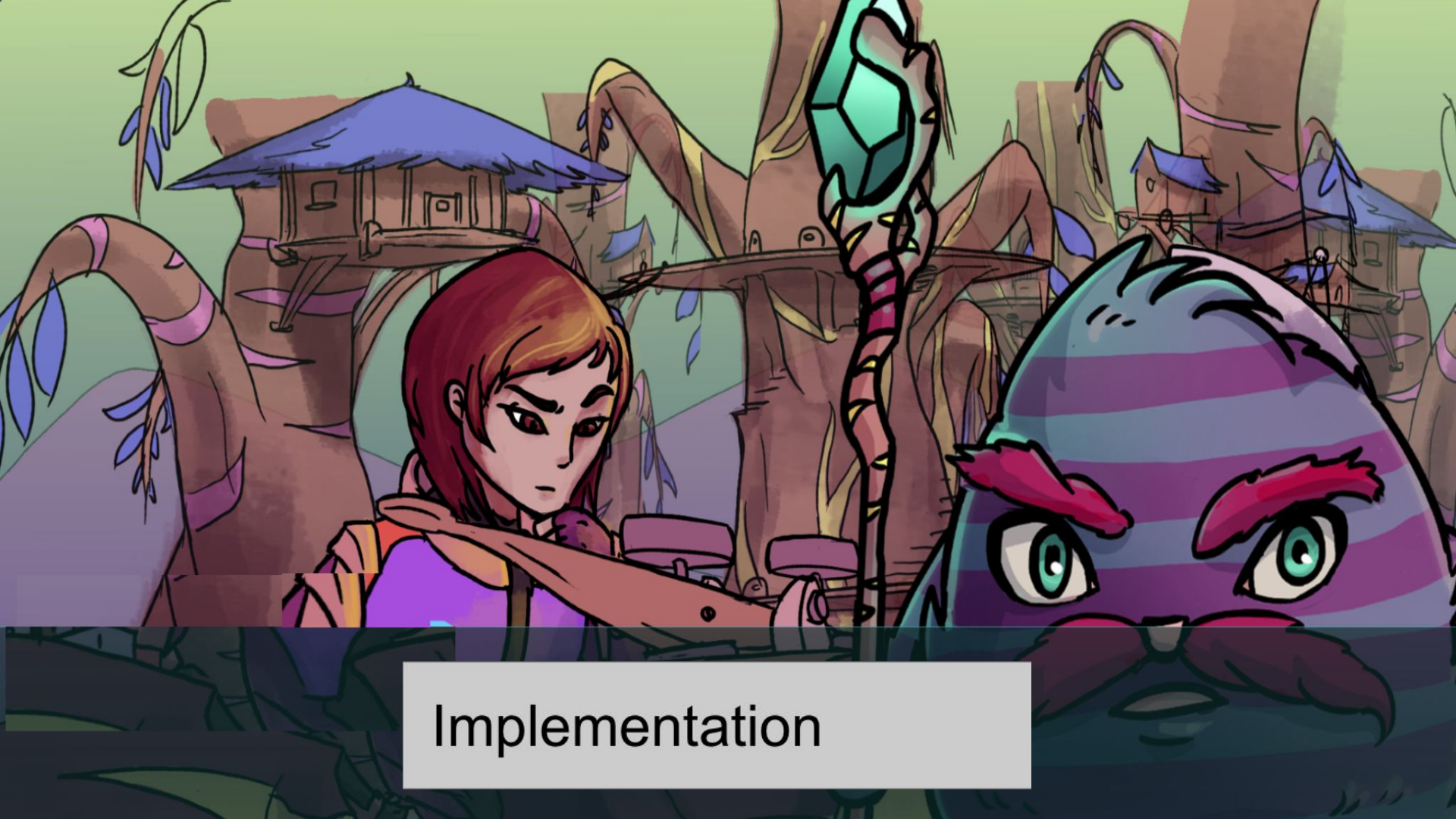
Composing video filters: https://share.sourceacademy.org/funwithfilters

Motion detector: https://share.sourceacademy.org/motiondetector

Source §4

- Add function `parse` for meta programming

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook


Implementation

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Source Academy

Open-source, developed *for* students *by* students:
First-year projects, Prog. Lang. Implementation term projects, Final-Year Projects

- Source Academy: https://sourceacademy.org server-less, on Github pages
- Source Academy @ NUS: https://sourceacademy.nus.edu.sg adds:
  - Scalable backend (written in Elixir, currently hosted on AWS)
  - Game
  - Achievements
  - Assignments (uploading, submission, manual and automatic grading)
  - Contests
  - Course management support

In-browser language implementations (js-slang)

- Parser: restricts students to chosen sublanguage
- Transpiler: JavaScript-to-JavaScript translation ensures proper tail calls (PTC) even when the browser does not implement PTC, adds pedagogical error messages
- Stepper: based on small-step reduction semantics
- Compilers from Source to SMVL virtual machine language: used for robotics and SICP 3.4
- Interpreters: used for environment visualizer and SICP 4.3

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook


Ershk
Wow... Can't believe that worked!
Outcomes

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Outcome: Shrinking languages

Shrinking the CS1 language is **liberating** everyone involved:

- Students: “I can achieve what my ‘expert programmer’ peers can achieve.”
- Instructor: “I don’t need to worry about language features that I don’t cover.”
- Implementer: “I can design and implement new tools in a semester project.”

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Outcome: Source Academy

91% of CS1101S students in 2021 said they
Agree/Strongly Agree that the Source Academy helped them
“understand the structure and interpretation of computer programs”

Some anonymous CS1101S student feedback:

- “Source Academy was a brilliant and fun platform to use. The format of paths, missions, and quests kept my interest up throughout the course.”
- “The Source Academy was nothing short of a marvel; I cannot imagine the amount of effort and resources that were needed to make it a success…”

Outlook

Motivation
Shrinking JavaScript
Implementation
Outcome
Outlook

Outlook: JavaScript for CS1

- EcmaScript 2015 enabled seamless use of JavaScript in SICP-based courses

- JavaScript keeps improving while retaining the functional core used in SICP

Any application that can be written in JavaScript,

will eventually be written in JavaScript.

Atwood's Law

Motivation
Shrinking JavaScript
Implementation
Outcome
Outlook

Outlook: Shrinking languages

You can **roll your own** web-based shrunken language implementation using Source Academy infrastructure

Examples:

- Scheme in Source Academy
- SICPy

Motivation
Shrinking JavaScript
Implementation
Outcome
Outlook

Outlook: Entry-level CS Education

SICP is still, after 50 years, the best computer science book in the world.

Brian Harvey, Berkeley

- SICP JS translation to Chinese under way
- Synergy between textbook and Source Academy


Can we build an inclusive global community of learners of entry-level computer science?

Some fun with Source §1

Runes: https://share.sourceacademy.org/rightsplit

Curves: https://share.sourceacademy.org/funwithcurves

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Some fun with Source §2

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Functional audio processing: https://share.sourceacademy.org/echo

Sound contest 2019 winner: https://share.sourceacademy.org/0iz2g

Some fun with Source §3

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Composing video filters: https://share.sourceacademy.org/funwithfilters

Motion detector: https://share.sourceacademy.org/motiondetector

The Solution (in Scheme and C)

```
(define (range bst low high)
List *range(BST *bst, int low, int high) {
  (cond ((< (datum bst) low)
  if (bst->datum < low)
         (range (right-branch bst) low high))
          return range(bst->right, low, high);
        ((> (datum bst) high)
         else if (bst->datum > high)
         (range (left-branch bst) low high))
          return range(bst->left, low, high);
        (else
         else return
         (append (range (left-branch bst) low high)
          append(range(bst->left, low, high),
                 (cons (datum bst)
                  cons(bst->datum,
                       (range (right-branch bst) low high))))))
                        range(bst->right, low, high))); }
```

From:
Brian Harvey's
"Last Lecture"
at Berkeley,
May 3 2013

Parser

The Source Academy uses Acorn[1], an open-source JavaScript parser, to build the Abstract Syntax Tree (AST).

We also check for any disallowed JavaScript syntax and return an error if any is found. What we get at the end is a valid Source AST.

[1]https://github.com/acornjs/acorn

Is SICP JS more complex than the original? If so: why?

Apart from the superficial syntax issues, SICP JS differs from SICP in two major ways:
(1) It adds return statements to the language: you can return from a function anywhere in the body
(2) It adds the notion of parsing: the text of a program can be transformed into a data structure

But the question is: What are the concepts that need to be covered today, when the ambition is “Structure and Interpretation of Computer Programs”?

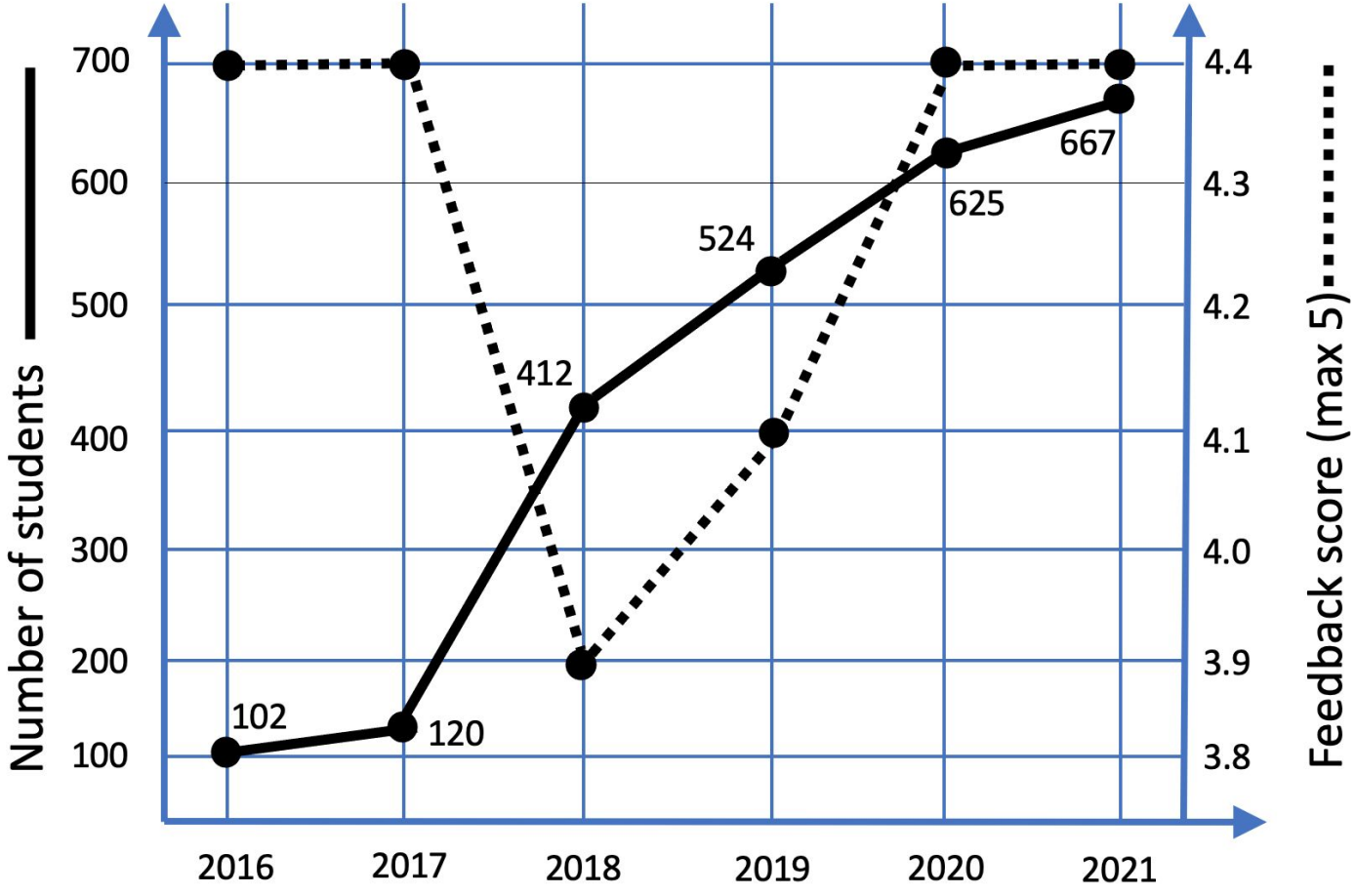
- Return statements?
- Language processing of non-Lisp-like languages?

If the answer in these two cases is “Yes” then adding Return statements and Parsing is not a bug but a feature:

A reader who is interested in the “structure and interpretation of computer programs” should learn about return statements and what they mean, because they occur in most languages that are in popular use today.

Similarly, a reader should be exposed to parsing because it is the key to implementing any language that is not Lisp-like.

Outcome: CS1101S student # and feedback



Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Learning experiences

- Motivation
- Shrinking JavaScript
- Implementation
- Outcomes
- Outlook

Learning experiences

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Background

- 1970s-90s: Hal Abelson and Gerald Jay Sussman spearhead education with Structure and Interpretation of Computer Programs
- 1997: NUS adopts SICP in a CS1 course called CS1101S
- 2008: JavaScript adaptation of SICP starts
- 2012: CS1101S converts from Scheme to JavaScript
- 2015: EcmaScript 2015 enables “serious” work on SICP JS
- **2018: CS1101S becomes compulsory for all CS first-year students**

The challenge: scaling from 120 student in 2017 to 667 students in 2021

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Why use JavaScript rather than Python?

- Proper tail calls (PTC) is in JavaScript standard (ES2021).
- Python does not specify PTC.

- Functional programming is at least as elegant in JavaScript as in Scheme.
- Python imposes syntactic restrictions on lambda expressions.

- JavaScript clearly distinguishes assignment from declaration (since ES2015).
- Python does not syntactically distinguish between assignment and declaration.

Plus: All the fun in the World Wide Web**!**

Stepper

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Processes for factorial: https://share.sourceacademy.org/factorialinstepper

Data Viz

Data visualization: SICP JS 2.2.2

Debugging append: https://share.sourceacademy.org/66ymt

Motivation

Shrinking JavaScript

Implementation

Outcomes

Outlook

Motivation
Shrinking JavaScript
Implementation
Outcomes
Outlook

Environment Visualizer

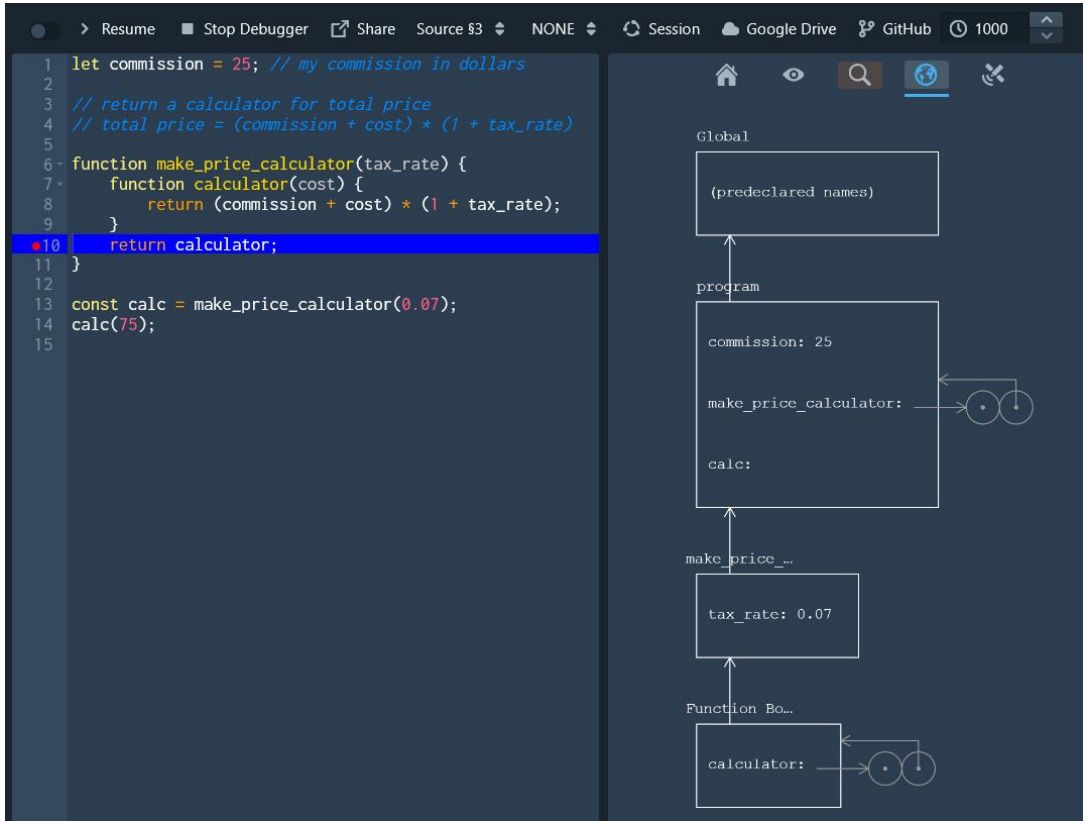
Debugging a bank account: https://share.sourceacademy.org/bankaccount

Debugging cps: https://share.sourceacademy.org/appendcps

Learning Tools: Environment Visualiser

Allows students to inspect a Source program's current execution state by setting breakpoints before the relevant program lines.

It uses a CPS-style interpreter (rather than Source transpiler)



Why did instructors stop using Scheme for CS1?

- Programming has become a practically useful skill for students: internships, summer jobs, startups,...
- Student motivation increases when they *perceive* the language as “useful” to them
- Syntax not very important…except:
  Scheme syntax is ***so*** different from the rest