

Discussion 07: Scheme

TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**

Agenda

1. Attendance
2. Feedback
3. Announcements
4. Scheme (fast)
5. Check Your Understanding

Attendance

Sign in at bit.do/jerrydisc

OR

Come to me for check-in

Also, for David: <http://tinyurl.com/CS61A-110-feedback>

Announcements

MT 2 Grades are out

- As always, feel free to email me to chat!

Hw 9 due Halloween (OoOooOo0000Ooo)

All other grades on Ok! P/NP deadline is this Friday

1958

MIT

~~A long time ago in a galaxy far,
far away....~~

LSP

Functional Programming

The image shows the official Scala website homepage. The background is a scenic photograph of a mountain range with a lake. At the top left is the Scala logo. To its right are navigation links: DOCUMENTATION, DOWNLOAD, COMMUNITY, and CONTRIBUTE, along with social media icons for GitHub and Twitter. The main headline reads "Object-Oriented Meets Functional". Below it is a descriptive paragraph: "Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between." A "LEARN MORE" button is located below this text. In the center is the large red Scala logo with the text "Scala 2.11.8" underneath. On either side of the logo are green call-to-action buttons: "DOWNLOAD" on the left and "API DOCS" on the right. Below these buttons are sections for "Getting Started" and "Current API Docs", each with a list of links.

Scala

DOCUMENTATION DOWNLOAD COMMUNITY CONTRIBUTE

Object-Oriented Meets Functional

Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between.

LEARN MORE

DOWNLOAD

Getting Started

- Milestones, nightlies, etc.
- All Previous Releases

Scala 2.11.8

API DOCS

Current API Docs

- API Docs (other versions)
- Scala Documentation
- Language Specification

<http://www.scala-lang.org/>

Scheme

Scheme — a **functional** language

- Dialect of the popular **Lisp** programming language



Scheme

Note: staff-provided scheme interpreter available at
scheme.cs61a.org

Based on CS 61A's Scheme Project

```
scm> (demo 'songs)
(demo-song <name> [times] [tempo]) to play a song
Available songs: ode-to-joy, sarias-song, kakariko-village,
song-of-storms, fight-for-california

To load a song from a GitHub Gist, use:
(gist-song <gist-id> <name> [times] [tempo])
scm> (demo-song 'song-of-storms)
Preparing song...
Loading accordion...
Loading tangoAccordion...
Loading oboe...
Loading vibraphone...
Loading percussion...
Playing...
scm> |
(debug code) evaluates code step-by-step
Full Usage Guide - Code Editor - Submit Bugs
scm> (demo 'paint)
Click and drag on the canvas to draw.
Use (pensize n) to change the size and (color 'color) to change the color.
scm> |
```



Scheme

Like Python, but...

harder?

- No iteration — recursion only!
- No mutation/mutable structures

Scheme

Like Python, but...

better?

- No finicky indentation
- No mutation/mutable structures (yup, this is both good and bad!) — **simpler code and behavior**

Scheme

Like Python, but... (faster, stronger)

not actually like Python?

- Where's iteration? (only expressions!)
- Where are objects?
- There are actually quite a few similarities, however...

Scheme

Primitives

Numbers	1, 12, 3.1416
Truthy values	#t, everything else
Falsy values	#f

Scheme

Note on booleans

- The only **false value is #f** itself (our interpreter also supports "false")
- Everything else is **“truthy”** (#t, 0, empty list, etc.)

Scheme

Functions

- Like Python, **parentheses** denote a function call
 - **Eval operator, eval operands, apply**
- We use **polish prefix notation** (you'll get used to it!)

Scheme

Python	Scheme
<code>3 + 0.14 + 0.0016</code>	<code>(+ 3 0.14 0.0016)</code>
<code>(4 * 4) + 2000</code>	<code>(+ (* 4 4) 2000)</code>
<code>pi = 3.1416</code>	<code>(define pi 3.1416)</code>
<code>pi == 3 # evals to False</code>	<code>(= pi 3) # evals to false</code>

Scheme

Symbols

- **Quoted** expressions are not evaluated
- Allow us to talk about Scheme, in Scheme! (more on this in the proj)
- Also allow compound objects (more on this when we talk about pairs)

Scheme

Python	Scheme
<code>1 and 2 and 3</code>	<code>(and 1 2 3)</code>
<code>not 1 or 2 or 1 / 0</code>	<code>(or (not 1) 2 (/ 1 0))</code>
<code>if pi > 3: return 1 else: return 0</code>	<code>(if (> pi 3) 1 0)</code>

Scheme

Python	Scheme
<code>lambda x, y: x + y</code>	<code>(lambda (x y) (+ x y))</code>
<code>square = lambda x: x * x</code>	<code>(define square (lambda (x) (* x x)))</code>
<code># Same as above</code>	<code>(define (square x) (* x x))</code>

Scheme

Pairs

- A **Scheme abstract data type**
- Much like **linked lists** in Python
- Pairs have a first (`car`) and a rest (`cdr`)
- Build pairs by linking (`cons`) together two things

Scheme

Python	Scheme
<code>Link(1, empty)</code>	<code>(cons 1 nil)</code>
<code>Link(1, Link(2, empty))</code>	<code>(cons 1 (cons 2 nil))</code>
<code>Link(1, 2) # Not allowed!</code>	<code>(cons 1 2) ; Allowed!</code>

Scheme

Well-formed (“good looking”) lists end in nil

```
scm> (cons 1 (cons 2 nil))
```

```
(1 2)
```

Malformed lists are denoted by a dot

```
scm> (cons 1 2)
```

```
(1 . 2)
```

Scheme

Cons vs List

Scheme

Quotes allow us to not evaluate a list, and just simplify it instead:

```
scm> '(1 . (2 . (3)))
```

```
(1 2 3)
```

The **list** function creates lists out of anything!

```
scm> (list 'list 1 ''(2))
```

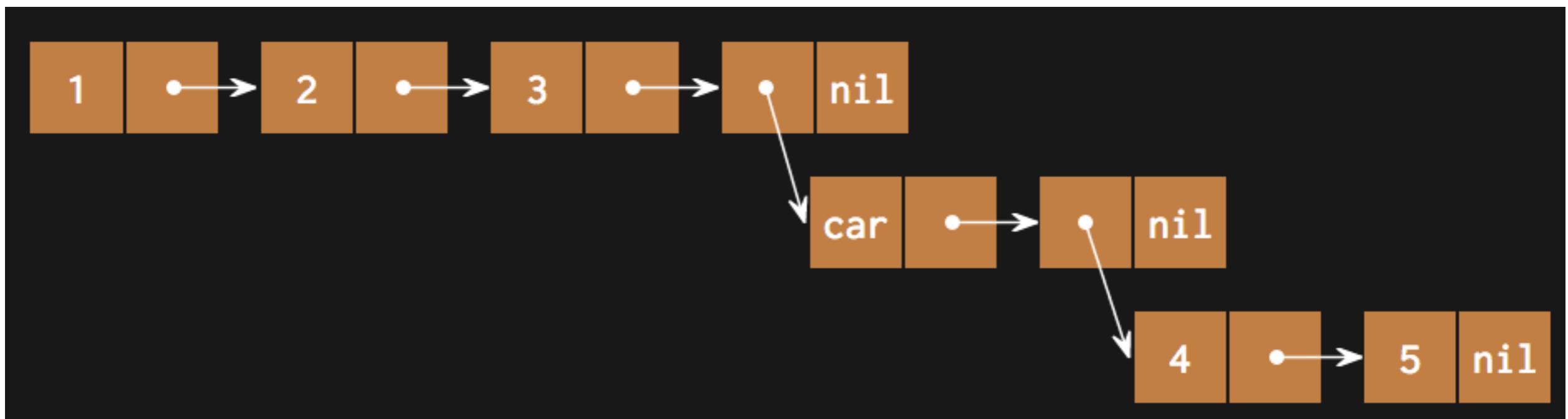
```
(list 1 '(2))
```

Check Your Understanding

Draw the diagram for the following:

```
>(list 1 '(2 . (3)) ' (4) 5)
```

Convert the following diagram into a list:



WWSD?

```
scm> (+ 1)  
1  
scm> (* 3)  
3  
scm> (+ (* 3 3) (* 4 4))  
25  
scm> (define a (define b 3))  
a  
scm> a  
b  
scm> b  
3
```

WWSD?

```
scm> (if (or #t (/ 1 0)) 1 (/ 1 0))  
1  
scm> (if (> 4 3)  
(+ 1 2 3 4) (+ 3 4 (* 3 2)))  
10  
scm> ((if (< 4 3) + -) 4 100)  
-96  
scm> (if 0 1 2)  
1
```