Discussion 08: **Scheme**

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

Agenda

Scheme

1. Syntax

2. Scheme lists

Announcements

MT 2 Grades are out. Submit regrades by next Sunday

Ants due Thursday

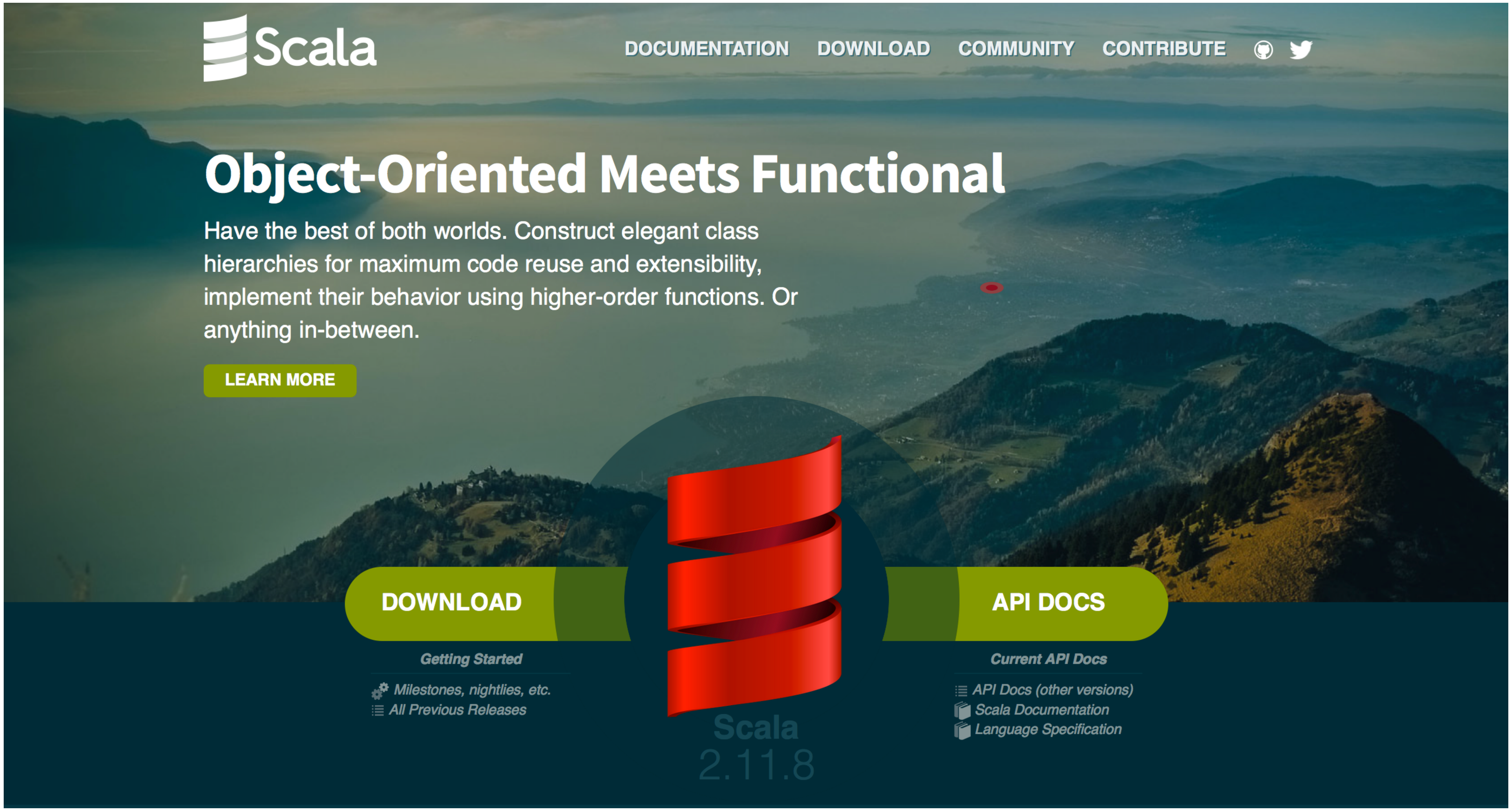
HW 6 due Friday

- HW Party Th 6:30-8:30pm, 247 Cory

Scheme Preparedness

What's up with those parentheses amirite	1
	2
I could do a problem after looking at an example or two	3
	4
I could teach this to someone	5

Functional Programming



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.

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

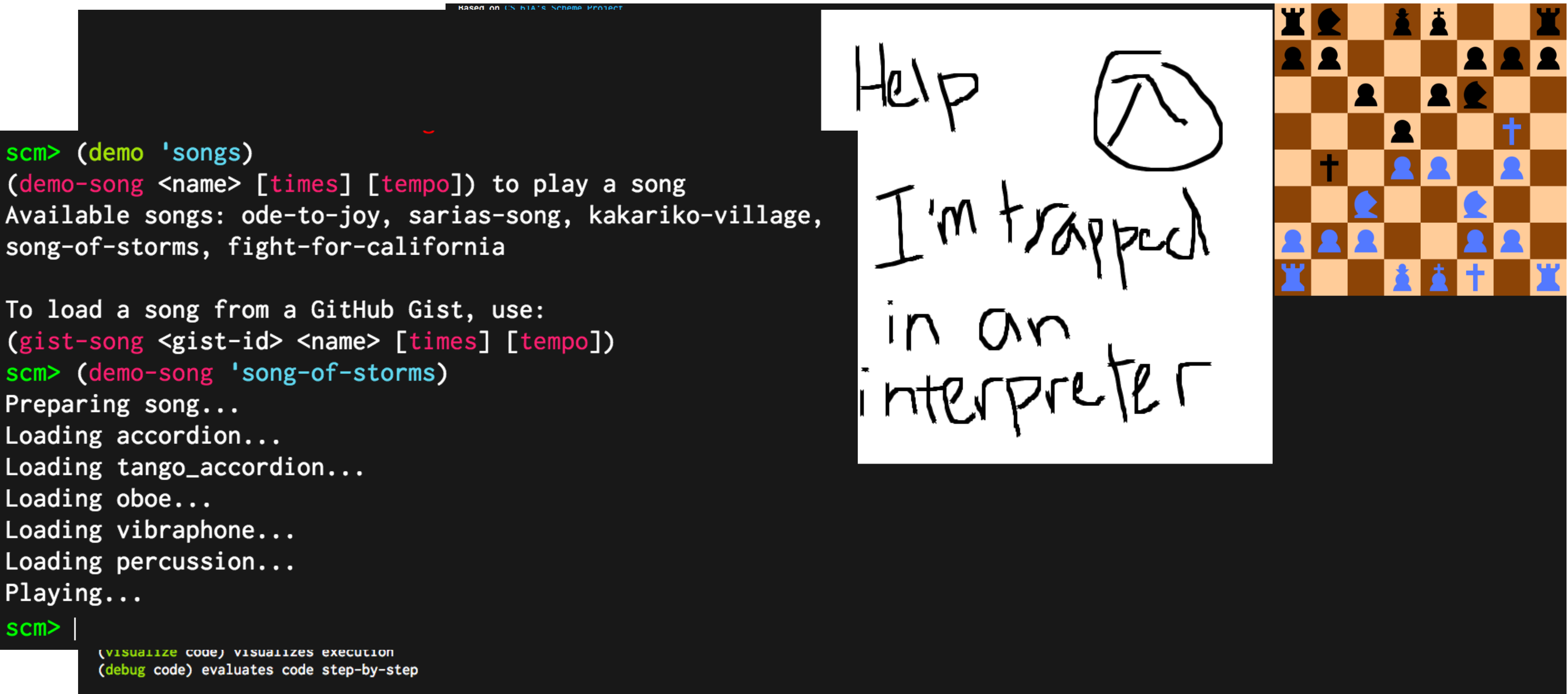
Scheme

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 tango_accordion...
Loading oboe...
Loading vibraphone...
Loading percussion...
Playing...
scm> |
```

```
(visualize code) visualizes execution
(debug code) evaluates code step-by-step
```

vs Python

Like Python, but…

harder?

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

vs Python

Like Python, but…

better?

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

vs Python

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

Note "prefix notation" for operators

```
(square (+ 5 5))
```

1. Eval operator

2. Eval operand(s)

3. Apply operator to operands

Scheme

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

Scheme

Python	Scheme
`1 and 2 and 3`	`(and 1 2 3)`
`not 1 or 2 or 1 / 0`	`(or (not 1) 2 (/ 1 0))`

```
if pi > 3:
 return 1
else:
 return 0
```
	`(if (> pi 3) 1 0)`

Scheme

Python	Scheme
`lambda x, y: x + y`	`(lambda (x y) (+ x y))`

`square = lambda x: x * x`	`(define square`
	` (lambda (x) (* x x)))`

`# Same as above`	`(define (square x) (* x x))`

Pairs

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

Scheme

Python	Scheme
`Link(1, empty)`	`(cons 1 nil)`
`Link(1, Link(2, empty))`	`(cons 1 (cons 2 nil))`
`Link(1, 2) # Not allowed!`	`(cons 1 2) ; Allowed!`

Lists

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

Symbols

- **Quoted** expressions are not evaluated
- Allow us to talk about Scheme, in Scheme!
- Also allow typing in "compound objects" (basically, scheme lists)

Lists

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))
```

Lists

List is not (always) your friend

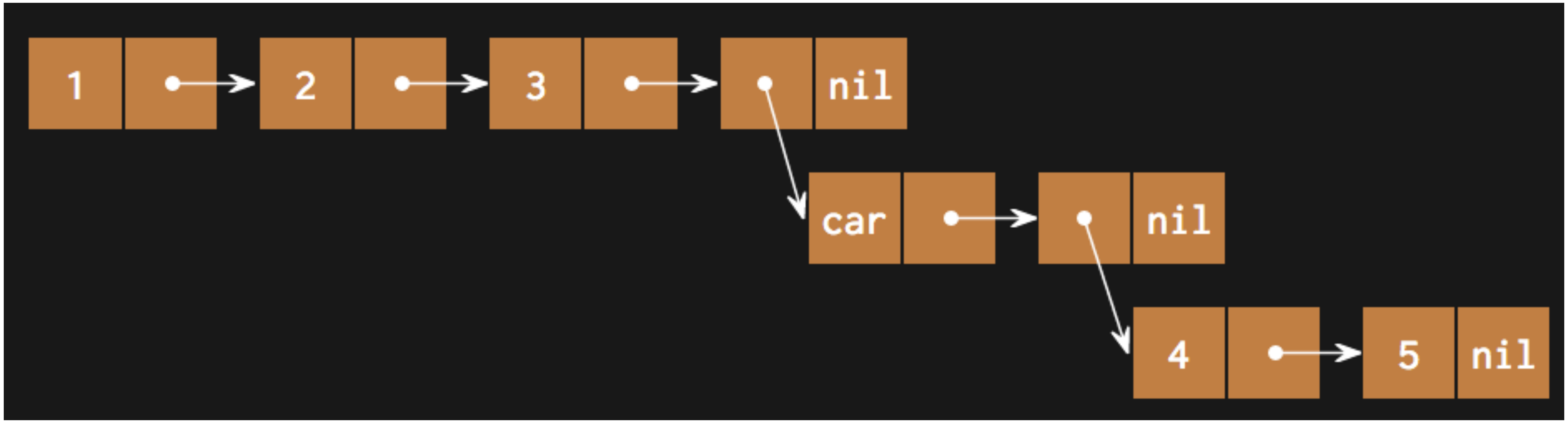
```
scm> (cons 1 '(2 3))

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

Check Your Understanding

1. Draw the diagram for the following:

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

2. Convert the following diagram into a list:



WWSD? Q1

```
scm> (define a 1)
a
scm> a
1
scm> (define b a)
b
scm> b
1
scm> (define c 'a)
c
scm> c
a
```

WWSD? Q2

```
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? Q3

```
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
```