CS61A Discussion 8: **Scheme**

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

Attendance

Form: **tinyurl.com/jerrydisc**

For the weekly question, **please complete the quiz (will be posted after discussion)**

(Of course, please only check in if you showed up!)

Agenda

1. Week in Review
2. Scheme

Week In Review

Ants - **Due Today!**

(Mini) Quiz - **Due Friday**

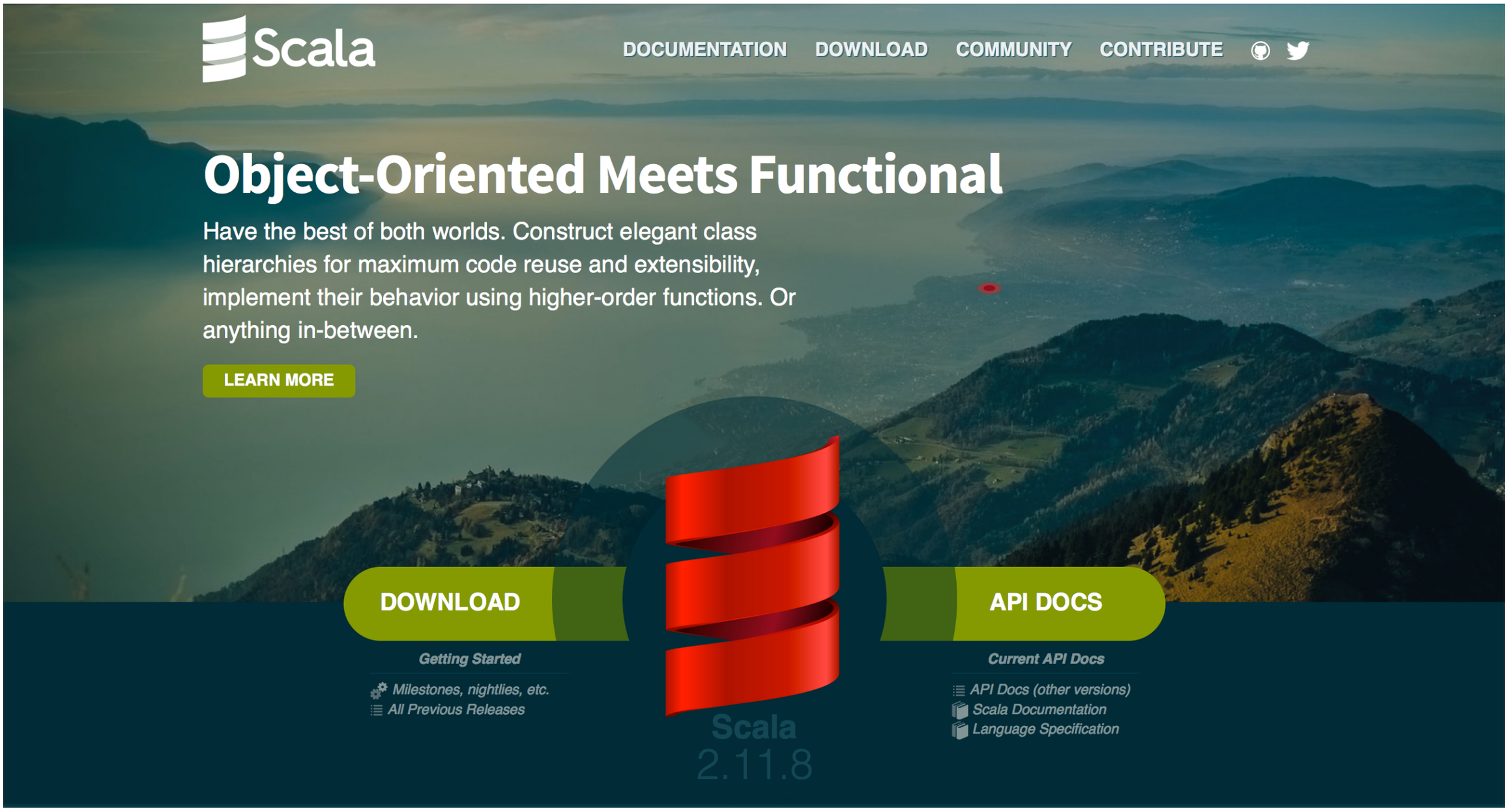
Lab 8 (Sets, Binary Trees) - **Due Friday**

Hw5 - Due Monday 3/28

Mt2 - **7-9pm, Wednesday** after Spring Break (3/30)

- ***Submit alternate time request ASAP!***

Functional Programming



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

Scheme

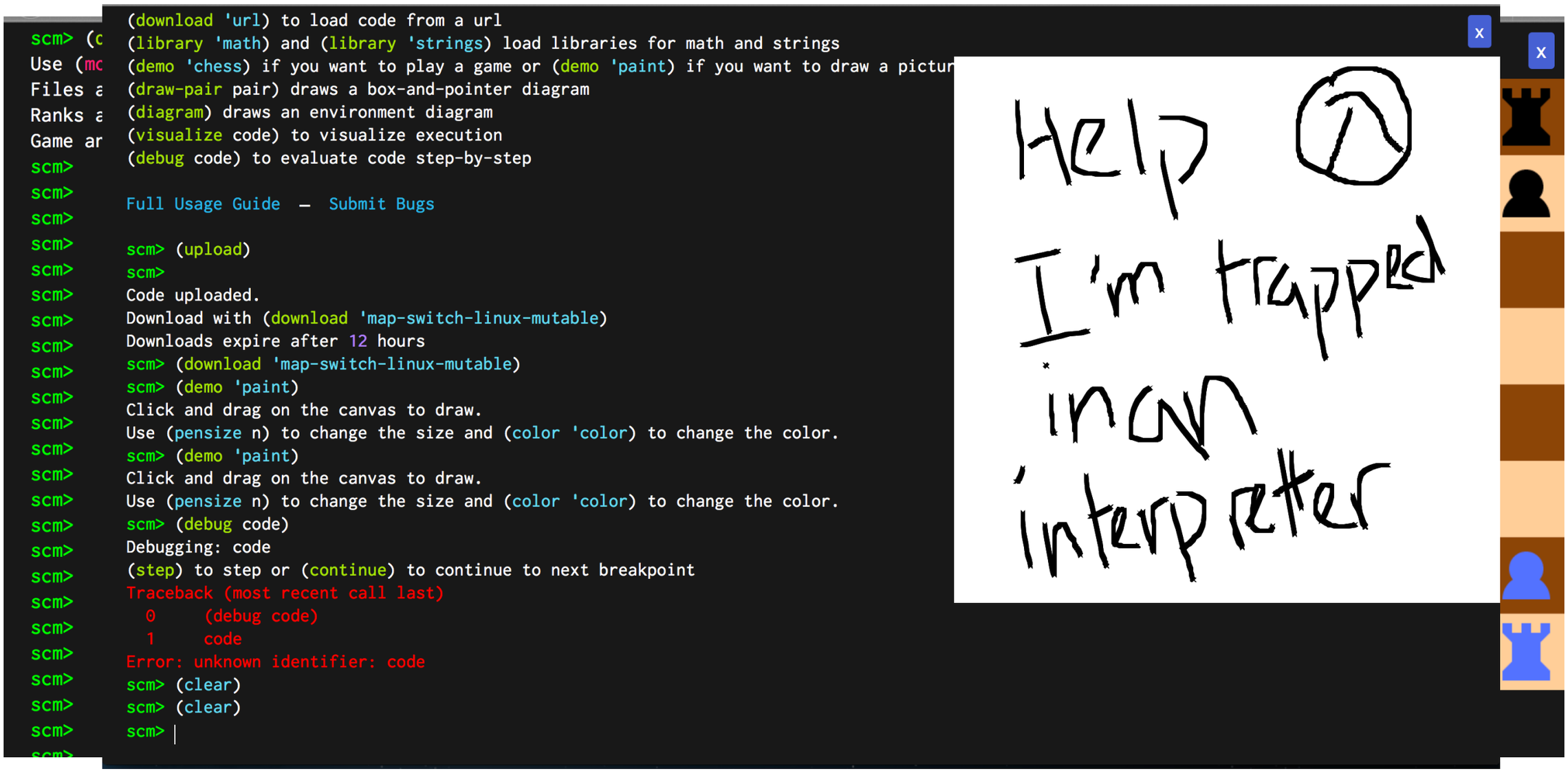
Last week: object oriented programming

This week: Scheme — a **functional** language

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

Scheme

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



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)~~

actually completely different?

- Only **expressions**!
  - Call expressions, lambdas, etc.
- There are actually quite a few similarities, however…

Scheme

Primitives

Numbers	`1, 12, 3.1416`
TRUE	`#t`
FALSE	`#f`

Scheme

Note on booleans

- The only **false value is #f** itself
- 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
`3 + 0.14 + 0.0016`	`(+ 3 0.14 + 0.0016)`
`(4 * 4) + 2000`	`(+ 2000 (* 4 4))`
`pi = 3.1416`	`(define pi 3.1416)`
`pi == 3 # evals to False`	`(= pi 3) # evals to #f`

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
`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))`

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
`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!`

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

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))
```