

# CS61A Discussion 8:

# **Scheme**

TA: **Jerry Chen**

Email: **[jerry.c@berkeley.edu](mailto:jerry.c@berkeley.edu)**

TA Website: **[jerryjrchen.com/cs61a](http://jerryjrchen.com/cs61a)**

# Attendance

Form: **[tinyurl.com/jerrydisc](https://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**

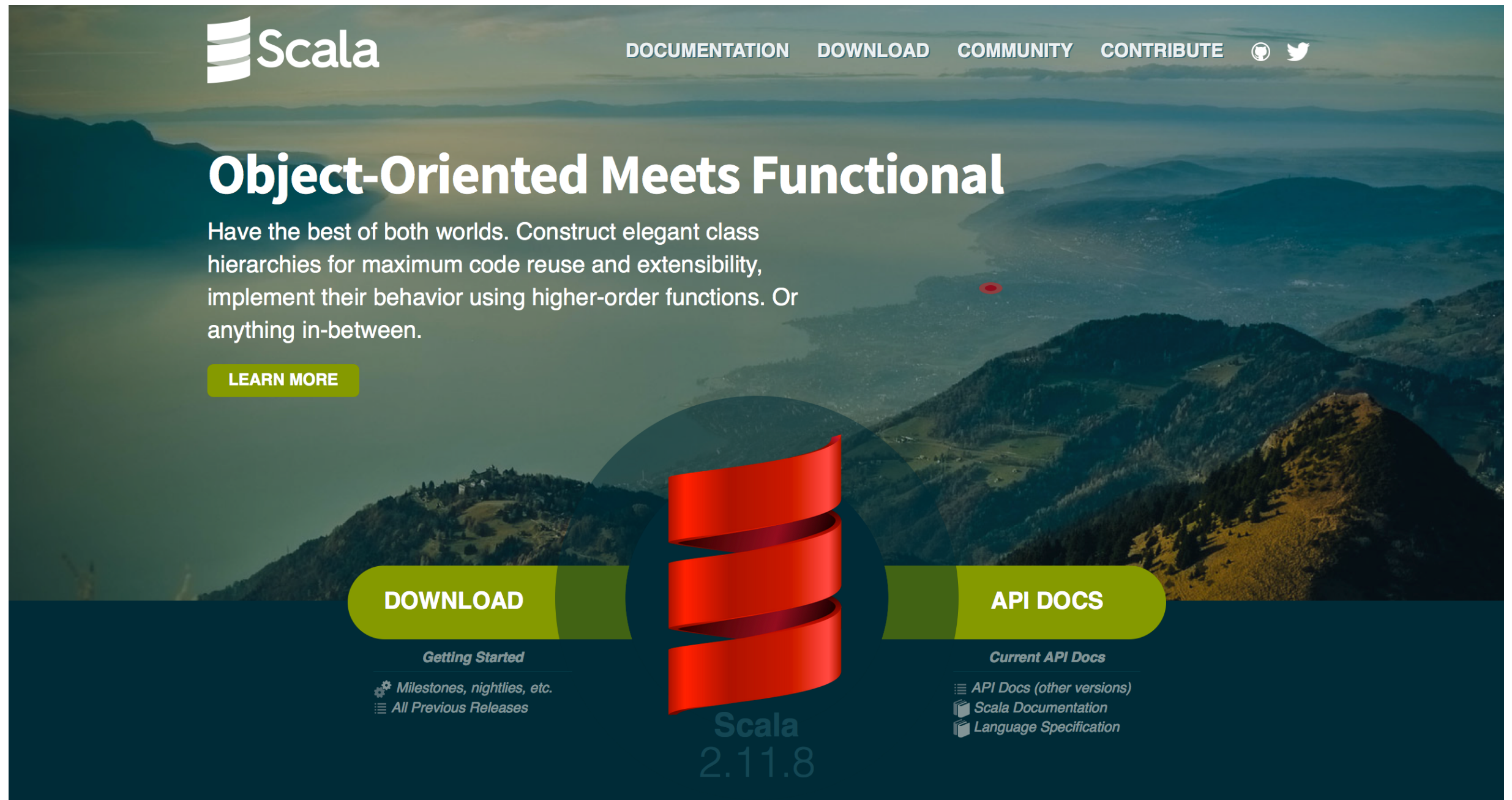
Lab 8 (Sets, Binary Trees) - **Due Friday**

Hw5 - Due Monday 3/28

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

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

# Functional Programming

The image shows the homepage of the Scala website. The background is a scenic landscape with mountains and a lake. At the top left is the Scala logo. To the right of the logo are navigation links: DOCUMENTATION, DOWNLOAD, COMMUNITY, and CONTRIBUTE, followed by GitHub and Twitter icons. The main heading is "Object-Oriented Meets Functional". Below it is a paragraph describing Scala's features. A "LEARN MORE" button is positioned below the paragraph. In the center, there is a large red ribbon graphic. Below the ribbon, the text "Scala 2.11.8" is visible. On either side of the ribbon are two green buttons: "DOWNLOAD" on the left and "API DOCS" on the right. Below the "DOWNLOAD" button is a section titled "Getting Started" with links for "Milestones, nightlies, etc." and "All Previous Releases". Below the "API DOCS" button is a section titled "Current API Docs" with links for "API Docs (other versions)", "Scala Documentation", and "Language Specification".

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

API DOCS

Scala 2.11.8

Getting Started

- Milestones, nightlies, etc.
- All Previous Releases

Current API Docs

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

<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](http://scheme.cs61a.org)

The image shows a screenshot of the Scheme interpreter interface. On the left, a terminal window displays a series of commands and their outputs. The commands include `(download 'url)`, `(library 'math)`, `(library 'strings)`, `(demo 'chess)`, `(demo 'paint)`, `(draw-pair pair)`, `(diagram)`, `(visualize code)`, `(debug code)`, `(upload)`, `(download 'map-switch-linux-mutable)`, `(demo 'paint)`, `(debug code)`, `(step)`, `(continue)`, `(clear)`, and `(clear)`. The outputs include "Code uploaded.", "Downloads expire after 12 hours", "Click and drag on the canvas to draw.", "Use (pensize n) to change the size and (color 'color) to change the color.", "Debugging: code", "Traceback (most recent call last)", "0 (debug code)", "1 code", and "Error: unknown identifier: code".

On the right, a whiteboard area contains a handwritten note in black ink that reads: "Help I'm trapped in my interpreter". To the right of the text is a circled number "1". The whiteboard has a vertical toolbar on its right side with icons for a king, a person, and a king.

# 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	<code>1, 12, 3.1416</code>
TRUE	<code>#t</code>
FALSE	<code>#f</code>

# 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
<code>3 + 0.14 + 0.0016</code>	<code>(+ 3 0.14 + 0.0016)</code>
<code>(4 * 4) + 2000</code>	<code>(+ 2000 (* 4 4))</code>
<code>pi = 3.1416</code>	<code>(<b>define</b> pi 3.1416)</code>
<code>pi == 3 # evals to False</code>	<code>(= pi 3) # evals to #f</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 &gt; 3:     return 1 else:     return 0</code>	<code>(if (&gt; 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>(<b>cons</b> 1 nil)</code>
<code>Link(1, Link(2, empty))</code>	<code>(<b>cons</b> 1 (<b>cons</b> 2 nil))</code>
<code>Link(1, 2) # Not allowed!</code>	<code>(<b>cons</b> 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

**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) )
```