Final Exam

As also indicated in class-list email:

- **Next Tuesday, 8:30-10:20AM**
- Intention is to focus primarily on material since the midterm
  - Including topics on homeworks and not on homeworks
  - May also have a little ML, just like the course has had
- You will need to write code and English, but the focus will be on code
**Victory Lap**

A victory lap is an extra trip around the track

- By the exhausted victors (us) 😊

Review course goals

- Slides from Introduction and Course-Motivation

Some big themes and perspectives

- Stuff for five years from now more than for the final

Time for open Q&A

Do your course evaluations!!!
Thank you!

- **Huge** thank-you to your TAs
  - Great team effort
  - Really invested in a successful course
Thank you!

- And a huge thank you to all of you
  - Great attitude about a very different view of software
  - Good class attendance and questions
  - Occasionally laughed at stuff 😊

- Computer science ought to be challenging and fun!
[From Lecture 1]

• Many essential concepts relevant in any programming language
  – And how these pieces fit together

• Use ML, Racket, and Ruby languages:
  – They let many of the concepts “shine”
  – Using multiple languages shows how the same concept can “look different” or actually be slightly different
  – In many ways simpler than Java

• Big focus on functional programming
  – Not using mutation (assignment statements) (!)
  – Using first-class functions (can’t explain that yet)
  – But many other topics too
Learning to think about software in this “PL” way will make you a better programmer even if/when you go back to old ways.

It will also give you the mental tools and experience you need for a lifetime of confidently picking up new languages and ideas.
[From Course Motivation]

• No such thing as a “best” PL

• Fundamental concepts easier to teach in some (multiple) PLs

• A good PL is a relevant, elegant interface for writing software
  – There is no substitute for precise understanding of PL semantics

• Functional languages have been on the leading edge for decades
  – Ideas have been absorbed by the mainstream, but very slowly
  – First-class functions and avoiding mutation increasingly essential
  – Meanwhile, use the ideas to be a better C/Java/PHP hacker

• Many great alternatives to ML, Racket, and Ruby, but each was chosen for a reason and for how they complement each other
[From Course Motivation]

SML, Racket, and Ruby are a useful combination for us

<table>
<thead>
<tr>
<th></th>
<th>dynamically typed</th>
<th>statically typed</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional</td>
<td>Racket</td>
<td>SML</td>
</tr>
<tr>
<td>object-oriented</td>
<td>Ruby</td>
<td>Java</td>
</tr>
</tbody>
</table>

**ML:** polymorphic types, pattern-matching, abstract types & modules

**Racket:** dynamic typing, “good” macros, minimalist syntax, eval

**Ruby:** classes but not types, very OOP, mixins

[and much more]

Really wish we had more time:

**Haskell:** laziness, purity, type classes, monads

**Prolog:** unification and backtracking

[and much more]
Benefits of No Mutation

[An incomplete list]

1. Can freely alias or copy values/objects: Unit 1

2. More functions/modules are equivalent: Unit 4

3. No need to make local copies of data: Unit 5

4. Depth subtyping is sound: Unit 8

State updates are appropriate when you are modeling a phenomenon that is inherently state-based
  – A fold over a collection (e.g., summing a list) is not!
Some other highlights

• Function closures are really powerful and convenient…
  – … and implementing them is not magic

• Datatypes and pattern-matching are really convenient…
  – … and exactly the opposite of OOP decomposition

• Sound static typing prevents certain errors…
  – … and is inherently approximate

• Subtyping and generics allow different kinds of code reuse…
  – … and combine synergistically

• Modularity is really important; languages can help
From the syllabus

Successful course participants will:

• Internalize an accurate understanding of what functional and object-oriented programs mean

• Develop the skills necessary to learn new programming languages quickly

• Master specific language concepts such that they can recognize them in strange guises

• Learn to evaluate the power and elegance of programming languages and their constructs

• Attain reasonable proficiency in the ML, Racket, and Ruby languages and, as a by-product, become more proficient in languages they already know
What now?

• Use what you learned whenever you reason about software!
• CSE 401
• CSE 402

Does PL research (cf. uwplse.org) design new general-purpose languages? *Not really; it does cool stuff with same intellectual tools!*

Some current UW projects
  – 3D-printing tools
  – Checker framework
  – Rosette
  – Language for microfluidics
  – Verified software written in Coq (which is quite SML-like)
The End

This really is my favorite course and it probably always will be

Don’t be a stranger!

Time for ask-me-anything questions?