CSE341: Programming Languages

Lecture 27ish
Course Victory Lap

James Wilcox

Winter 2017

Administrivia

- IP2 is out as of this morning; due in a week
- Intention is to focus primarily on material since the midterm
  - More fun to show you new stuff than ask you about old stuff
  - But will still test your understanding of old stuff "on the way"
- You will need to write code and English
- Please do course evals

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 IP2

Thank you!

- **Huge** thank-you to your TAs
  - Great team effort
  - Deep understanding of material despite all having different 341 instructors
  - Put up with me
  - Great sections, timely grading, etc., etc.

Thank you!

- And a huge thank you to all of **you**
  - Great attitude about a very different view of software
  - Good questions
  - Put up with me
  - 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

[From Lecture 1] Why learn this?

To free our minds from the shackles
of imperative programming.

[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

	dynamically typed	statically typed
functional	Racket	SML
object-oriented	Ruby	Java

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

Where to go from here

- Consider taking further PL courses
  - 401, 505, 507, …
- Consider picking up a cool new language on your own
  - Haskell, Rust, Agda, ...
- Understand an X by building your own X
  - PL, OS, website, app, ...
- Consider getting involved in research
- Consider going to grad school

The End

I've really enjoyed teaching this course (learning some of it as I go!)

Don't be a stranger!