CSE341: Programming Languages

Lecture 28
Victory Lap

Dan Grossman

Fall 2011

Final Exam

As also indicated in class-list email:

- Next Tuesday, 2:30-4:20
- Intention is to focus primarily on material since the midterm
  - Including topics on homeworks and not on homeworks
  - Will also have a little ML, just like the course has had
- You will need to write code and English

- I hope you will pick up your exams once they are available
  - Even if you stop by in January

Victory Lap

A victory lap is an extra trip
around the track

- By the exhausted victors (us) ☺

Review course goals

- Slides from Lectures 1, 7.5
- Plus syllabus and the ACM/IEEE CS2013 draft

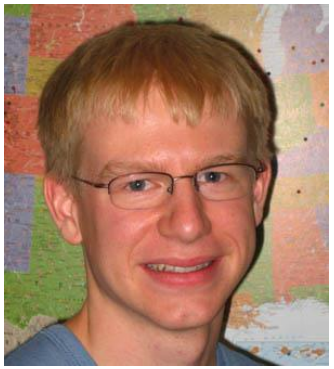
Some big themes and perspectives

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

Course evaluations: please do take some time

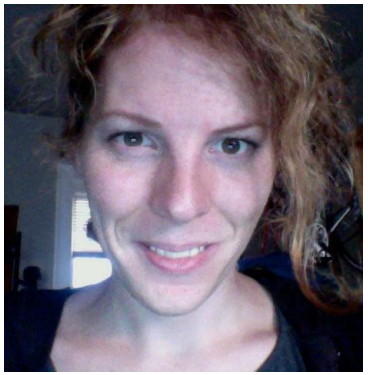
Thank you!

- **Huge** thank-you to your Tas
  - Two new homeworks
  - Section taken to the next level
  - > 500 assignments total
    - Prompt grading with very few complaints



10 sections
+ 1 lecture
+ …

Wrote Tetris
Just took 341



+ job interviews
yet moved only 1 hour

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
  - Only a few lonely office hours
  - Occasionally laughed at stuff ☺

[From Lecture 1]

We have 10 weeks to learn *the fundamental concepts* of programming languages

With hard work, patience, and an open mind, this course makes you a much better programmer

- Even in languages we won't use
- Learn the core ideas around which *every* language is built, despite countless surface-level differences and variations
- *Poor* course summary: "Uses SML, Racket, and Ruby"

Today's class:

- Course mechanics
- *[A rain-check on motivation]*
- Dive into ML: Homework 1 due end of next week

[From Lecture 1]

Homework:

- Seven total
- To be done individually
- Doing the homework involves:
  1. Understanding the concepts being addressed
  2. Writing code demonstrating understanding of the concepts
  3. Testing your code to ensure you understand and have correct programs
  4. "Playing around" with variations, incorrect answers, etc.

  We grade only (2), but focusing on (2) makes the homework harder
- Challenge problems: Low points/difficulty ratio

[From Lecture 1]

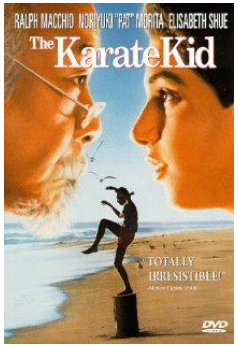
- Many essential concepts relevant in any programming language
  - And how these pieces fit together
- Use the languages ML, Racket, and Ruby because:
  - They let many of the concepts “shine”
  - Using multiple languages shows how the same concept can “look different” or actually be slightly different
  - In many ways simpler than Java
- A big focus on *functional programming*
  - Not using *mutation* (assignment statements) (!)
  - Using *first-class functions* (can’t explain that yet)

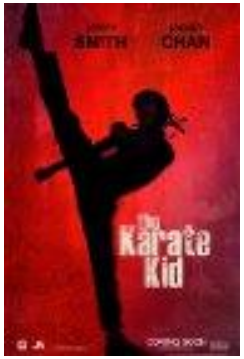
[From Lecture 1]

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

[Somewhat in the style of *The Karate Kid* movies (1984, 2010)]





[From Lecture 7.5]

- A good mechanic might have a specialty, but also understands how “cars” (not 2004 Honda Civics) work
  - And that the syntax, I mean upholstery color, isn’t essential
- A good mechanical engineer really knows how cars work, how to get the most out of them, and how to design better ones
- To learn how cars work, it may make sense to start with a classic design rather than the latest model
  - A popular car may not be a good car for learning how cars work

[From Lecture 7.5]

This course focuses as much as it can on semantics and idioms

- Correct reasoning about programs, interfaces, and compilers *requires* a precise knowledge of semantics
  - Not “I feel that conditional expressions might work like this”
  - Not “I like curly braces more than parentheses”
  - Much of software development is designing precise interfaces; what a PL means is a *really* good example

- Idioms make you a better programmer
  - Best to see in multiple settings, including where they shine
  - See Java in a clearer light even if I never show you Java

[From Lecture 7.5]

Why 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]

[From Lecture 7.5]

- No such thing as a “best” PL
- There are good general design principles for PLs
- A good language is a relevant, crisp interface for writing software
- Software leaders should know PL semantics and idioms
- Learning PLs is not about syntactic tricks for small programs
- Functional languages have been on the leading edge for decades
  - Ideas get absorbed by the mainstream, but very slowly
  - Meanwhile, use the ideas to be a better C/Java/PHP hacker

Benefits of No Mutation

[This is from memory; I may have forgotten some]

1. Can freely alias or copy values/objects: Lecture 3

2. More functions/modules are equivalent: Lectures 3, 12

3. No need to make local copies of data: Lecture 14

4. Depth subtyping is sound: Lecture 25

State updates are appropriate when you are modeling a phenomenon that is inherently state-based

- Performing a fold over a collection (e.g., summing a list) isn't!

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 OO 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

[Caveat: I wrote the goals, so not surprising I hope we met them.]

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

From the "so-called experts" ☺

- Once a decade or so, ACM/IEEE updates a "standard CS curriculum"
  - A specification of what every CS undergraduate degree should teach its students

- The next version will come out in 2013
  - The PL/Compilers area will be significantly revised
  - And, uh, apparently I'm in charge of that ☺
    - Worked with a committee of 10 "experts" and incorporated feedback from another 7 "reviewers"
  - First draft will be public February 2012
    - Let's take a sneak peek and see how we did on the "core learning outcomes" for the PL area…

Not bad

These outcomes are for a curriculum, not necessarily a course

- [In separate document; not in slides or posted]

I think this quarter's CSE341 hit 9 out of 13 directly

- Brushed up on another 2, as do other courses
- Other 2 covered in CSE351

What next?

- CSE401 Compilers is a natural choice I highly recommend
- Stay in touch
  - Tell me when 341 helps you out with something cool
  - Ask me cool PL questions
  - Don't be a stranger: let me know how the rest of your time in CSE (and beyond!) goes… I really do like to know
- Oh, but first…

Course evaluations

I am going to distribute course evaluation forms so you may rate the quality of this course. Your participation is voluntary, and you may omit specific items if you wish. To ensure confidentiality, do not write your name on the forms. There is a possibility your handwriting on the yellow written comment sheet will be recognizable; however, I will not see the results of this evaluation until after the quarter is over and you have received your grades. Please be sure to use a No. 2 PENCIL ONLY on the scannable form.

I have chosen *(name)* to distribute and collect the forms. When you are finished, he/she will collect the forms, put them into an envelope and mail them to the Office of Educational Assessment. If there are no questions, I will leave the room ~~and not return until all the questionnaires have been finished and collected~~.

Thank you for your participation.