PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

CSE341: Programming Languages

Lecture 26
Course Victory Lap

Brett Wortzman

Spring 2020

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

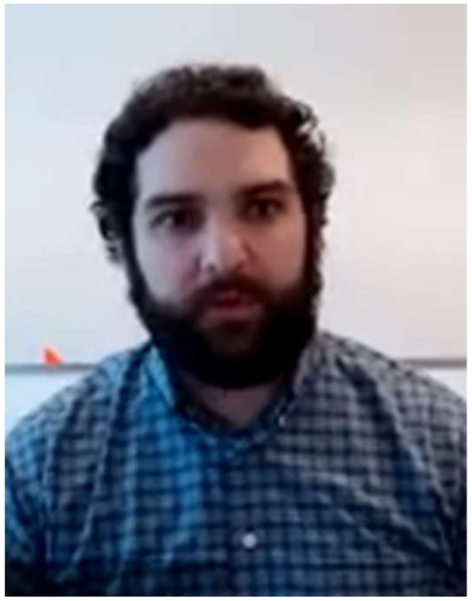
- Stuff for five years from now more than for the final

Maybe time for open Q&A

Please fill out the course evaluation!!!

We’ve come a long way

First Day of Class
March 30


(Almost) Last Day of Class
June 3

Thank you!

- **Huge** thank-you to your TAs
  - Great team effort
  - Really invested in a successful course
  - Many message boards posts, assignments graded
  - Many hours of teaching and prepping sections
  - SUPER hard working and high energy team ☺

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
  - Willingness to work with us during this crazy quarter

- Computer science ought to be challenging and fun!

[From Lecture 1]

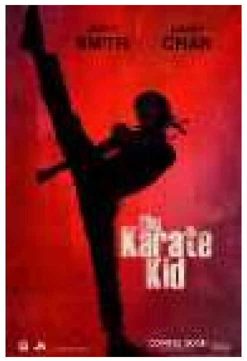
- Many essential concepts relevant in any programming language
  - And how these pieces fit together
- Use ML, Racket, and Ruby languages:
  - They let many of the concepts “shine”
  - Using multiple languages shows how the same concept can “look different” or actually be slightly different
  - In many 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]

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 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

More high-level takeaways

- Every choice involves tradeoffs
  - Type systems: Convenience vs. protection
  - Syntax: Conciseness vs. precision
  - Eagerness: Simplicity vs. performance
  - Purity: Clarify vs. usefulness

- Just because you can, doesn't mean you should (and vice versa!)
  - Mutation: makes reasoning harder
  - Wildcards/defaults: hides errors
  - Depth subtyping: prevents soundness (only if mutation allowed!)

- Programming languages are *hard*
  - Have sympathy next time you wonder "why can't Language X just allow this?"

Wat?

Wat

@garybernhardt

https://www.destroyallsoftware.com/talks/wat

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

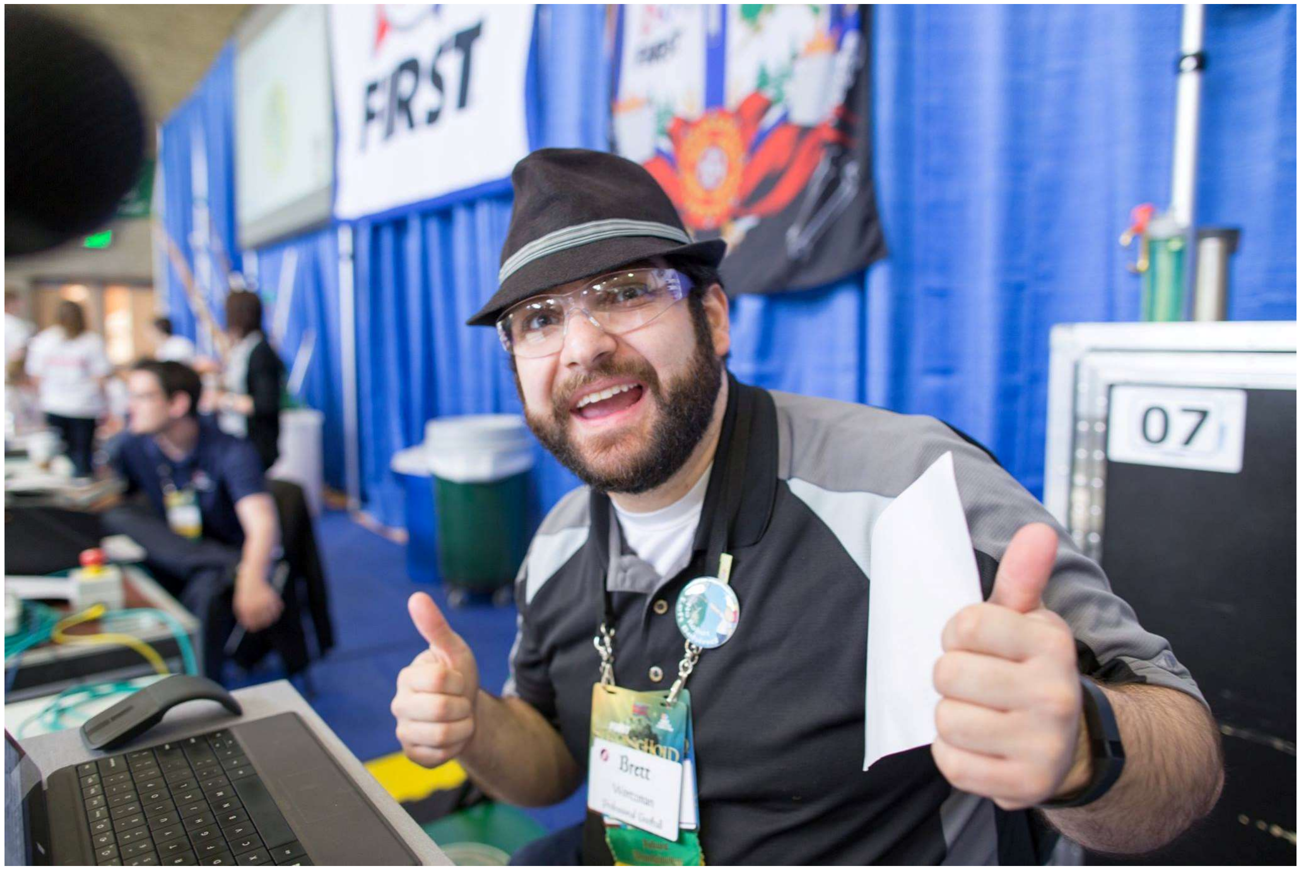
What now?

- Use what you learned whenever you reason about software!
- CSE 401 – Compilers
- CSE 402 – Domain-specific Languages
- CSE 490P – Advanced PLs and Verification (lots of proofs)
- CSE 505 – Principles of PLs (formal semantics, more proofs)

Does PL research design new general-purpose languages?

- *Not really; it does cool stuff with same intellectual tools!*
- Check out http://www.uwplse.org

The End



Don’t be a stranger!