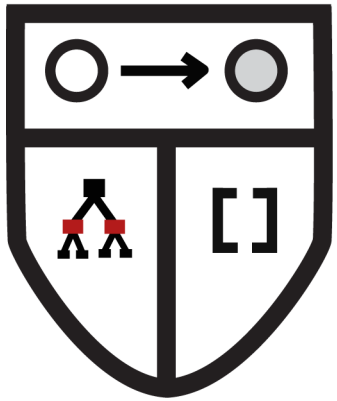


CS 3110

Victory Lap

Dietrich Geisler & Alan Cheng

Spring 2018

Victory Lap

Extra trip around the track by the exhausted victors – WE are the champions

Thank you!

Huge thank you to Prof. Foster and the course staff:

Harrison Goldstein, Richa Despande, Anmol Kabra, Adina Walzer, Anna Fang, Drew Dunne, Sitar Harel, Theodore Bauer, Tyler Etzel, Yuchen Shen, Cassandra Scarpa, Newton Ni, Devin Lehmacher, Irene Yoon, Frank Li, Wilson Chen, Jialu Bao, Iris Zheng, Kevin Gao, Kevin Wang, Kimberly Guo, Meghan Chen, Megan Le, Nina Ray, Ray Zeng, Shumei Guan, Sonia Appasamy, Xinyu Zhao, Tin Kuo, Ning Ning Sun

Thank you!

Thank you to Discourse heroes!

	Received	Given	Topics	Replies
DavidH David Huang	107	23	42	328
Rishi Rishi Bommasani	29	31	3	71
awt	6	3	36	60
ani1729cyber Aniroodh Ravikumar	8	27	4	59
zs225 Zack Shen	12	4	21	56
 **tz277** Timothy Zhu	9	81	38	49
ekumar Eashaan Kumar	11	24	101	48
alj55 Alex Jiang	11	14	17	30
jaclynhuang	13	17	35	28
sahamanish99 Manish Saha	8	6	11	24

Thank you!

And a huge thank you to all of **you!**

You surmounted a daunting challenge!

We ❤️ this course. You make it all worthwhile.

What did we learn?

- You feel exhausted...
- You're tired of coding...

...step back and think about what happened along the way

[Lec 1]
OCaml is awesome because of…

- Immutable programming
  - Variable's values cannot destructively be changed; makes reasoning about program easier!
- Algebraic datatypes and pattern matching
  - Makes definition and manipulation of complex data structures easy to express
- First-class functions
  - Functions can be passed around like ordinary values
- Static type-checking
  - Reduce number of run-time errors
- Automatic type inference
  - No burden to write down types of every single variable
- Parametric polymorphism
  - Enables construction of abstractions that work across many data types
- Garbage collection
  - Automated memory management eliminates many run-time errors
- Modules
  - Advanced system for structuring large systems

BIG IDEAS

1. Languages can be learned systematically

- Every language feature can be defined in isolation from other features, with rules for:
  - syntax
  - static semantics (typing rules)
  - dynamic semantics (evaluation rules)

- Divide-and-conquer!
- Entire language can be defined mathematically and precisely
  - SML is. Read *The Definition of Standard ML (Revised)* (Tofte, Harper, MacQueen, 1997).

- Learning to think about software in this "PL" way has made you a better programmer even when you go back to old ways
  - And given you the mental tools and experience you need for a lifetime of confidently picking up new languages and ideas

2. Immutability is an advantage

- No need to think about pointers or draw memory diagrams
- Think at a higher level of abstraction
- Programmer can alias or copy without worry
- Concurrent programming easier with immutable data
- But mutability is appropriate when
  - you need to model inherently state-based phenomena
  - or implement some efficient data structures

3. Programming languages aren't magic

- Interpretation of a (smallish) language is something you can implement yourself

- Domain specific languages (DSL): something you probably *will* implement for some project(s) in your career

4. Elegant abstractions *are* magic

From a small number of simple ideas...

...an explosion of code!

- language features: product types, union types
- higher order functions: map, fold, ...
- data structures: lists, trees, dictionaries, monads
- module systems: abstraction, functors

Computational Thinking

Jeannette Wing

- *Computational thinking is using abstraction and decomposition when... designing a large, complex system.*
- *Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.*

https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
https://www.microsoft.com/en-us/research/video/computational-thinking/

5. Building software is more than hacking

- **Design**: think before you type
- **Empathy**: write code to communicate
- **Assurance**: testing and verification
- **Teamwork**: accomplish more with others

6. CS has an intellectual history created by people like you

Big ideas

1. Languages can be learned systematically
2. Immutability is an advantage
3. Programming languages aren't magic
4. Elegant abstractions are magic
5. Building software is more than hacking
6. CS has an intellectual history created by people like you

FAQs

- Why OCaml?
- When will I use FP again?

Languages are tools

Languages are tools

- There's no universally perfect tool
- There's no universally perfect language
- **OCaml was good for this course** because:
  - good mix of functional & imperative features
  - relatively easy to reason about meaning of programs
  - From the Turing Award citation for Robin Milner:
    ML was way ahead of its time. It is built on clean and well-articulated mathematical ideas, teased apart so that they can be studied independently and relatively easily remixed and reused. ML has influenced many practical languages, including Java, Scala, and Microsoft's F#. Indeed, no serious language designer should ignore this example of good design.
- **But OCaml isn't perfect** (see above)

FAQs

- Why OCaml?
- When will I use FP again?

FAQs

- Why OCaml?
- ~~When will I use FP again?~~ Why did I study FP?

Why study functional programming?

1. Functional languages teach you that programming transcends programming in a language (assuming you you have only programmed in imperative languages)
2. Functional languages predict the future
3. (Functional languages are *sometimes* used in industry)
4. Functional languages are elegant

Why study functional programming?

1. Functional languages teach you that programming transcends programming in a language (assuming you you have only programmed in imperative languages)
2. Functional languages predict the future
3. (Functional languages are *sometimes* used in industry)
4. Functional languages are elegant

Analogy: studying a foreign language

- Learn about another culture; incorporate aspects into your own life
- Shed preconceptions and prejudices about others
- Understand your native language better

Alan J. Perlis

1922-1990

"A language that doesn't affect the way you think about programming is not worth knowing."

First recipient of the Turing Award

for his "influence in the area of advanced programming techniques and compiler construction"

Why study functional programming?

1. Functional languages teach you that programming transcends programming in a language (assuming you you have only programmed in imperative languages)
2. Functional languages predict the future
3. (Functional languages are *sometimes* used in industry)
4. Functional languages are elegant

Functional languages predict the future

- Garbage collection
  Java [1995], LISP [1958]
- Generics
  Java 5 [2004], ML [1990]
- Higher-order functions
  C#3.0 [2007], Java 8 [2014], LISP [1958]
- Type inference
  C++11 [2011], Java 7 [2011] and 8, ML [1990]
- **What's next?**

Why study functional programming?

1. Functional languages teach you that programming transcends programming in a language (assuming you you have only programmed in imperative languages)
2. Functional languages predict the future
3. (Functional languages are *sometimes* used in industry)
4. Functional languages are elegant

Functional languages in the real world

- Java 8 
- F#, C# 3.0, LINQ 
- Scala   
- Haskell   
- Erlang   
- OCaml  

  https://ocaml.org/learn/companies.html 

Why study functional programming?

1. Functional languages teach you that programming transcends programming in a language (assuming you you have only programmed in imperative languages)
2. Functional languages predict the future
3. (Functional languages are *sometimes* used in industry)
4. Functional languages are elegant

Elegant


Neat
Stylish
Dignified
Refined
Simple
Graceful
Effective
Precise
Consistent
Tasteful

Elegant

Neat Stylish

Beautiful

Precise Consistent

Tasteful

Q&A

FINAL MATTERS

What next?

- Follow-on courses:
  - CS 4110/6110 Programming Languages and Logics (how to define and reason about programming languages)
  - CS 4120 Compilers (how to implement programming languages)
  - CS 4160 Formal Verification (new course on verification with Coq)
- Learn another functional language?
  - Racket or Haskell
- Join the course staff?
  - CS department collects applications
  - Apply to be on staff next semester!

What next?

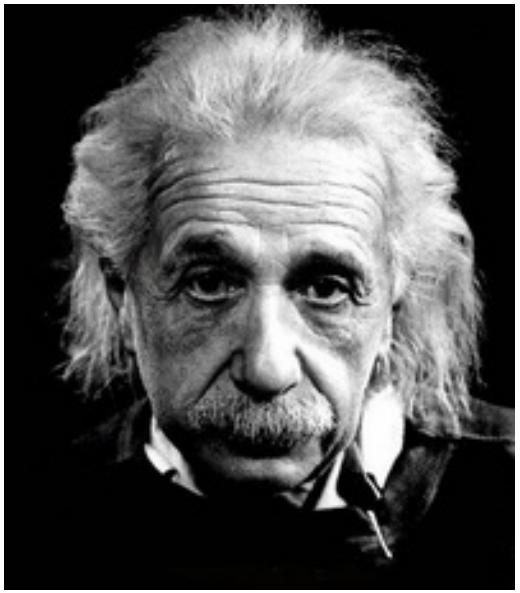
- Stay in touch
  - Tell Prof. Foster when 3110 helps you out with future courses (or jobs!)
  - Ask us cool PL questions
  - Drop by Prof. Foster's office to tell him about the rest of your time in CS (and beyond!)… He really does like to know

- Crossing the finish line is just the beginning of the next race…
  DO AMAZING THINGS WITH YOUR LIFE

Final Exam

- Thursday, 5/17/18, 7:00-9:30 pm, Statler Auditorium
- Covers everything in the course
- Closed-book exam, no notes
- (remaining details will be posted on Discourse)
- Final grades will be in CMS about 7-10 days after exam

Finally



1879-1955

"Education is what remains after one has forgotten everything one learned in school." –Albert Einstein

Finally

- The most important idea of this course:
  - complicated artifacts can be broken down into small pieces
  - you can then study those small pieces and understand how they work in isolation
  - then you can understand why their aggregation achieves some goals

- Examples: a module you designed, or the OCaml language

- That kind of analysis is applicable anywhere, not just programming

Upcoming events

- [tomorrow] A5 due
- [by 5/14/17 8:00 am] submit a course eval; worth 1% of final grade; take time on this, especially the free response questions