CS 3110

Victory Lap

Nate Foster
Spring 2020

Today's music: *We are the Champions* by Queen

I've paid my dues
Time after time.
I've done my sentence
But committed no crime.

And bad mistakes–
I've made a few.
I've had my share of sand kicked in my face
But I've come through.

It's been no bed of roses,
No pleasure cruise.
I consider it a challenge before the
whole human race,
And I ain't gonna lose.

We are the champions, my friends.
And we'll keep on fighting 'til the end.
We are the champions.
No time for losers
'Cause we are the champions of ~~the world~~ 3110.

Victory Lap

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

Thank you!

Huge thank you to TAs and consultants!

Ahad Rizvi, Alaia Solko-Breslin, Alexander Buckman, Alexander Chang, Aniroodh Ravikumar, Anthony Yang, Aryaa Pai, Calli Sabaitis, Cassandra Scarpa, Charles Dai, Chaske Yamane, Chris Lam, Chris Mulvaney, Connie Lei, Grace Zhao, Henry Zheng, Imaan Seraphine Islam Rahim, Jack Nash, Jacqueline Wu, Jake Sanders, James Goodling, Jia Jiunn Ang, John Kolesar, Joshua Kaplan, Katerina Sadov, Kerri Diamond, Kevin Wang, Kira Segenchuk, Ksenia Zhizhimontova, Laasya Renganathan, Lillian Hong, Lucia Gomez, Margaret Chan, Michael Zhao, Nathan Stack, Navin Ramsaroop, Neil Patel, Nicholas Yuwono, Noah Thompson, Olivia Lu, Olivia Zhu, Peter Buckman, Quinn Liu, Rhea Dutta, Roushi Shen, Ryan Richardson, Samuel Thomas, Samwise Parkinson, Shan Parikh, Sofya Bykova, Songyu Gu, Spencer Peters, Stephanie Hellman, Sydney Lawrence, Victoria Mao, Vivian Jiang, William Smith, Wilson Chen, Yanlam Ko, Youya Xia, Yuchen Tian, Olivia Li

Thank you!

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

- You surmounted a daunting challenge
- Your enthusiasm (even with the challenges of virtual instruction) was uplifting and contagious
- A shout out to the class of 2020 🎓

I ❤️ 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

Programming is not hard

Programming well is very hard

The Goal of 3110

Become a better programmer
though study of
programming languages

Questions we pursued

- How do you write code for and with other people?
  - Modular programming
  - Team-based final project
- How do you know your code is correct?
  - Testing
  - Verification
- How do you describe and implement a programming language?
  - Syntax and semantics
  - Interpreters

Tasks we pursued

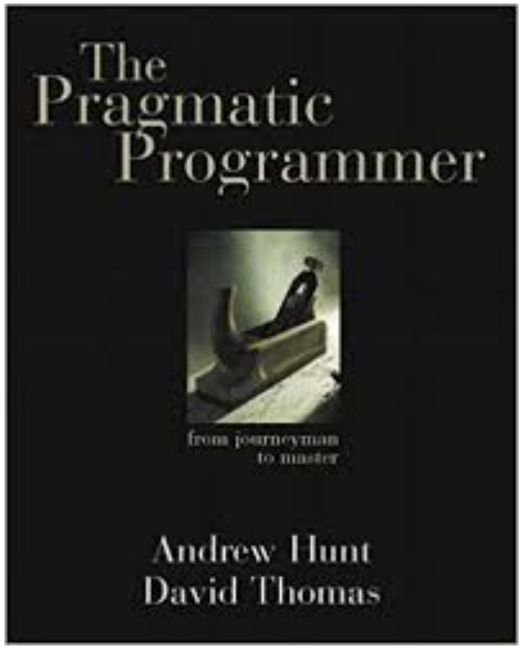
Practice of programming: read and write lots of code

Tasks we pursued

Practice of programming: coding as a (Zoom) team

Tasks we pursued

Philosophy of programming



Tasks we pursued

Learning a functional language

[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)*, by Tofte, Harper, and 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
- 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

Jeanette 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 and you can contribute

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 and you can contribute

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 ORACLE
- F#, C# 3.0, LINQ Microsoft
- Scala twitter foursquare LinkedIn
- Haskell facebook BARCLAYS at&t
- Erlang facebook amazon T-Mobile
- OCaml facebook Bloomberg CITRIX

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

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

Effective Graceful

Precise Consistent

Tasteful

Elegant

Neat Stylish

D

Beautiful

Precise Consistent

Tasteful

FINAL MATTERS

Please: keep repos private

- 11 Academic Integrity cases this fall
- 3 students failed the course as a result
- other lesser penalties

In nearly every case, plagiarism enabled because a student from a previous semester made their code public in a repo

What next?

- Follow-on courses:
  - CS 4110 Programming Languages and Logics (how to define and reason about programming languages)
  - CS 4120 Compilers (how to implement programming languages)
  - CS 4160 Formal Verification (how to prove correctness)
  - CS 5150/5152 Software Engineering (build for real clients)

- Learn another functional language?
  - Racket or Haskell

- Join the course staff?
  - CS department collects applications
  - Apply now to be on staff for Fall 2020: We seek a diverse course staff of people who want to give back to the community and can speak from their successes as well as struggles

What next?

- Stay in touch
  - Tell me when 3110 helps you out with courses (or jobs!)
  - Ask me cool PL questions
  - Drop by to tell me about the rest of your time in CS (and beyond!)… I really do like to know

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

Upcoming events

- [Tuesday] MS2 Report due
- [Tuesday] MS2 Peer Evaluation due
- [Thursday] Review Session @ 1pm
- [Friday] Course evaluations due
- [Saturday 5/16] Final Exam

This is …

This is victory.

THIS

HAS BEEN

3110