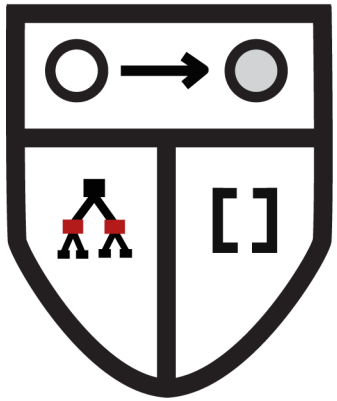


CS 3110

Introduction to 3110

Prof. Clarkson

Fall 2018

Today's music: Prelude from Final Fantasy VII
by Nobuo Uematsu (remastered by Sean Schafianski)

Programming is not hard

Programming well is very hard

Folklore:

10x

variation in professional programmer productivity

[Grant and Sackman, 1967]: 28x
[Prechelt 1999]: 2-4x

The Goal of 3110

Become a better programmer

though study of

programming languages

Programming Languages

Java is to Programming Languages
as
Japanese is to Linguistics

Programming Languages: Language design, implementation, semantics, compilers, interpreters, runtime systems, programming methodology, testing, verification, security, reliability…

Adjacent to **Software Engineering** in the CS family tree.

Questions we'll pursue

- How do you write code for and with other people?
- How do you know your code is correct?
- How do you describe and implement a programming language?

Tasks we'll pursue

Practice of programming: read and write lots of code

11 programming assignments:
about 100-400 LoC each, excluding testing and documentation

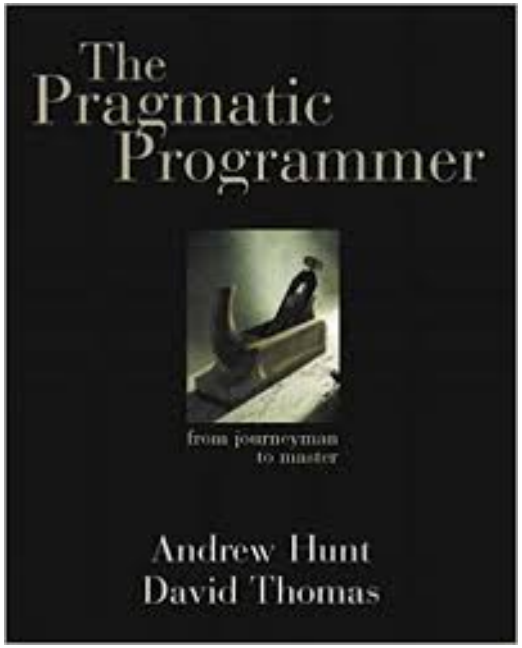
Tasks we'll pursue

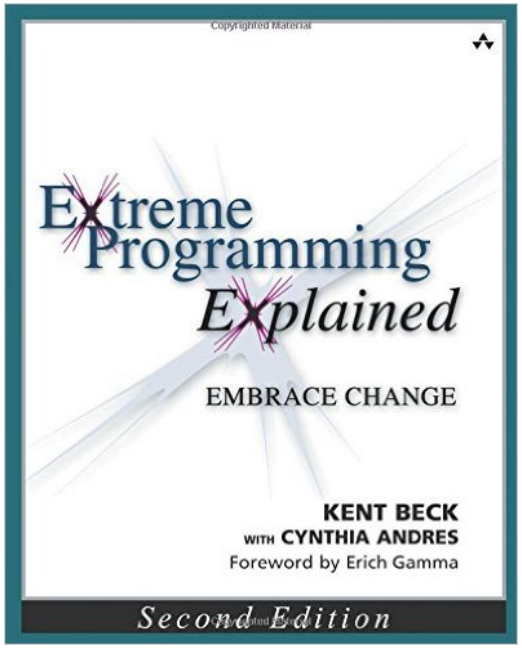
Practice of programming: coding as a team

Starting with 3rd assignment: instructor-formed teams of 3 or 4 students

Tasks we'll pursue

Concepts of programming: written assignments





Essay on the first book above; bonus essay on second book
Weekly written recitation assignments (no more than 1 page per recitation)

Tasks we'll pursue

Learning a functional language

Why? What does that even mean?

What is a functional language?

A functional language:

- defines computations as mathematical functions
- avoids mutable state

State: the information maintained by a computation

Mutable: can be changed (antonym: *immutable*)

Mutability

The fantasy of mutability:

- It's easy to reason about: the machine does this, then this...

The reality of mutability:

- Machines are good at complicated manipulation of state
- Humans are not good at understanding it!

  Mutability breaks *referential transparency*: ability to replace expression with its value without affecting result of computation

Imperative programming

Commands specify how to compute by destructively changing state:

```
x = x+1;
a[i] = 42;
p.next = p.next.next;
```

Functions/methods have **side effects**:

```
int x = 0;
int incr_x() {
  x++;
  return x;
}
```

Functional programming

Expressions specify what to compute

- Variables never change value
- Functions never have side effects

The reality of immutability:

- No need to think about state
- Powerful ways to build correct programs

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

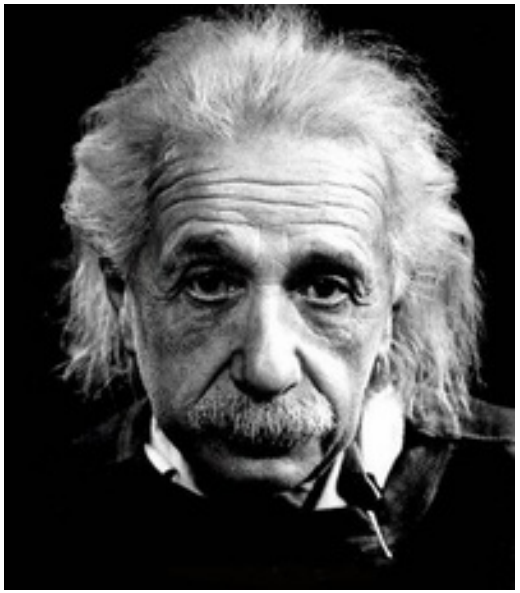
Functional languages in the real world

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

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

...but Cornell CS (et al.) require functional programming for your *education*, not to get you a job

Albert Einstein



1879-1955

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

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

Do aesthetics matter?

YES!

Who reads code?

- Machines
- Humans

- Elegant code is easier to read and maintain
- Elegant code might (not) be easier to write

OCaml

A pretty good language for writing beautiful programs

O = Objective, Caml=not important
ML is a family of languages; originally the “meta-language” for a tool

OCaml is awesome

- 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

But no language is perfect…

Languages are tools

Languages are tools

- There's no universally perfect tool
- There's no universally perfect language
- **OCaml is good for this course** because:
  - good mix of functional & imperative features
  - relatively easy to reason about meaning of programs
- **But OCaml isn't perfect**
  - there will be features you miss from language X
  - there will be annoyances based on your expectations
  - keep an open mind, try to have fun

LOGISTICS

Course website

cs3110.org

or

https://www.cs.cornell.edu/courses/cs3110/2018fa/

One-page course summary

- Pick up a hardcopy on your way out (also posted on website)
- Includes your TODO list before Monday

Course staff

Professor: Michael Clarkson

- PhD 2010 Cornell University
- I go by "Professor Clarkson" in this course
- Research background: security and programming languages
- Now I'm 100% teaching focused
- Interests that will show up in lecture: memes, sci-fi and fantasy, video games, music
- This is my 10-year anniversary with CS 3110

TAs and consultants: 53 at last count

Registration

- If you still want in, follow instructions on course website to **add yourself to Standby List**
- Deadline: Sunday noon

Upcoming events

- [today] Drop by my office in the afternoon if you need something immediately
- [tomorrow] Consulting hours start; check calendar on course website
- [Monday] Recitations begin (none today)
- [Tuesday] Bring iClicker

…why are you still here? Get to work! ☺

THIS IS 3110