

# CS 3110

## Introduction to 3110

Prof. Clarkson

Fall 2015

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

# Welcome!

- Programming isn't hard
- Programming **well** is **very** hard
  - High variance among professionals' productivity:  
10x or more
  - Studying functional programming will make you a better programmer, but it requires an open mind

# 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*)

# Functional vs. imperative

## Functional languages:

- Higher level of abstraction
- Easier to develop robust software
- Immutable state: easier to reason about software

## Imperative languages:

- Lower level of abstraction
- Harder to develop robust software
- Mutable state: harder to reason about software

*You don't have to believe me now.  
If you master a functional language, you will. 😊*

# 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 wheels(Vehicle v) {  
    v.size++;  
    return v.numWheels;  
}
```

# 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
  - In math, if  $f(x)=y$ , then you can substitute  $y$  anywhere you see  $f(x)$
  - In imperative languages, you cannot:  $f$  might have side effects, so computing  $f(x)$  at time  $t$  might result in different value than at time  $t'$

...mutable programming is not well-suited to building correct code!

# Mutability

## The fantasy of mutability:

- There is a single state
- The computer does one thing at a time

## The reality of mutability:

- There is no single state
  - Programs have many threads, spread across many cores, spread across many processors, spread across many computers...  
*each with its own view of memory*
- There is no single program
  - Most applications do many things at one time

...mutable programming is not well-suited to modern computing!

# 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 and concurrent 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 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

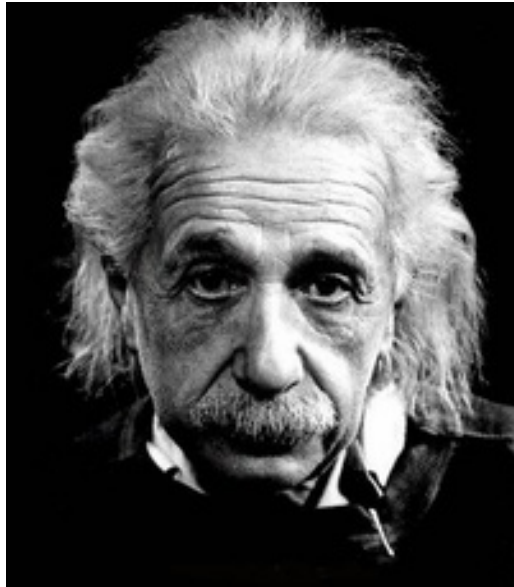
# Functional languages in the real world

- **Java 8** 
- **F#, C# 3.0, LINQ**  Microsoft
- **Scala**   **Linked in** 
- **Haskell**    at&t
- **Erlang**    T-Mobile
- **OCaml**  **Bloomberg**  **CITRIX**  
<https://ocaml.org/learn/companies.html>  Jane Street

...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

I

**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 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

**A GLIMPSE OF OCAML...**



**LOGISTICS**

# Course website

<http://www.cs.cornell.edu/Courses/cs3110/2015fa/>

- [Full syllabus](#) (required reading)
- Lecture slides
  - Typically go live within 24 hours after lecture
  - Supplement, do not replace, attendance
- Recitation notes

# Course staff

**Instructor:** Michael Clarkson

- PhD 2010 **Cornell University**
- Research areas: security and programming languages
- I go by “Prof. Clarkson” in this course



**Recitation instructor:** Mike George

- PhD ABD **Cornell University**



**TAs and consultants:** 46 at last count

- Course administrator (“head TA”): Remy Jette (rcj57)

# Course meetings

- **Lectures:** TR 10:10-11:00 am
  - attendance is semi-mandatory, about 1% of grade, measured by i>clicker
  - go buy an i>clicker and bring it on Thursday
- **Recitations:** mostly MW
  - Attendance is expected
  - TR sections are effectively MW delayed one day
  - You may attend any, regardless of registration, subject to room capacity; please pick one and stick with it
  - bring your laptop with OCaml installed
- **Consulting:** coverage nearly every day

# Email

- If it's about **content**, post a message on Piazza
  - Messages restricted only to me probably will get lost
- If it's about your own **personal logistics**, send email to [cs3110-instructors-L@cornell.edu](mailto:cs3110-instructors-L@cornell.edu)
- But better than that...come talk to me in person!

# Academic integrity

- You are bound by the Cornell Code of Academic Integrity, the CS Department Code of Academic Integrity, and the CS 1110 Explanation of Academic Integrity
  - All linked from course website
  - You are responsible for understanding them
- I use MOSS to detect plagiarism; **it works**
- If you have a question about what is or is not allowed, **please ask**
- If you fear you have committed a violation, **tell me** before grading commences

# Enrollment

- The course is at fire-code capacity
  - I cannot add any students
  - You will have to wait until someone drops
  - No auditors allowed this semester
- CS 3110 will be taught in the spring and historically is never full

# Your TODO list for tonight

1. read the syllabus
2. buy an i>clicker
3. login to CMS and Piazza
  - linked from course website
  - trouble? contact course administrator with your full name and NetID
4. install OCaml
  - we provide a virtual machine on course website
  - will be used in recitation tomorrow: download tonight
  - trouble? post on Piazza or see a consultant



# Upcoming events

- [Tue-Thur pm] Drop by my office in the afternoon if you need something immediately
- [Wednesday] Recitations begin (none today)
- [Wednesday pm] Consulting hours start; check calendar on course website for times and places
- [Thursday am] i>clickers start in lecture

...why are you still here? Get to work! 😊

**THIS IS 3110**