Introduction to OCaml

Prof. Clarkson
Summer 2016

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:

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

Functions/methods have **side effects**:

```plaintext
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 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
“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 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 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

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

1879-1955
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
Elegant

Stylish

Neat

Dignified

Refined

Simple

Effective

Graceful

Precise

Consistent

Tasteful
Elegant

Beautiful
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
- Modules
  - Advanced system for structuring large systems
A GLIMPSE OF OCAMLR...
This summer course

- Based on Cornell CS 3110
  - 15 week course
  - Required of all CS majors
  - Usually taken by sophomores
  - Third and last programming course

- We'll cover about 1/3 of CS 3110 in one week
Lab

• Schedule:
  – Lecture 10:10-11:00 am
  – Break 11:00-11:10 am
  – Lab 11:10-12:00 pm
  – Lunch 12:00-1:30 pm
  – Lecture 1:30-2:20 pm
  – Break 2:20-2:30 pm
  – Lab 2:30-3:20 pm

• Lab is Gates G23

• Lab (aka recitation) materials are linked from course webpage