Introduction to OCaml

Prof. Clarkson
Summer 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**:

```c
int wheels(Vehicle v) {
  v.size++;
  return v.numWheels;
}
```
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 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
Analogy: studying cooking

• Understand the structure behind recipes
• Cook without a manual
• Express yourself
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

- F#, C# 3.0, LINQ (Microsoft)
- Scala (Twitter, LinkedIn, FourSquare)
- Java 8
- Haskell (dozens of small companies/teams)
- Erlang (distributed systems, Facebook chat)
- OCaml (Jane Street)
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
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
Example 1: Sum Squares

// returns: Σ₁≤i≤n i²
int sum_squares(int n) {
    sum=0;
    for (int x = 1; x <= n; x++) {
        sum = sum + x*x
    }
    return sum;
}
Example 1: Sum Squares

// returns: $\sum_{1 \leq i \leq n} i^2$
int sum_squares(int n) {
    if (n==0) {
        return 0;
    } else {
        return n*n + sum_squares(n-1)
    }
}
**Example 1: Sum Squares**

(* returns: $\sum_{1 \leq i \leq n} i^2$ *)

```ml
let rec sum_squares n =
  if n=0 then 0
  else n*n + sum_squares (n-1)
```

*Closed form is better yet:* \[ \frac{n(n + 1)(2n + 1)}{6} \]
Example 2: Reverse List

// return a copy of x, // with the order of its elements reversed
List reverse(List x) {
    List y = null;
    while (x != null) {
        List t = x.next;
        x.next = y;
        y = x;
        x = t;
    }
    return y;
}
Example 2: Reverse List

(* return the reverse of lst *)

let rec reverse lst =
  match lst with
  | [] -> []
  | h::t -> (reverse t) @ [h]

This is not the most efficient algorithm
Example 3: Quicksort

- Describe quicksort in English.
- Describe quicksort in Java. (No.)
- Describe quicksort in OCaml:

```
(* returns lst sorted according to < *)
let rec qsort lst =
  match lst with
  | [] -> []
  | pivot::rest -> (* poor choice of pivot *)
    let (left,right) = partition ((<) pivot) rest
  in (qsort left) @ [pivot] @ (qsort right)
```

But definitely don't use this exact algorithm
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 BRIEF TOUR OF OCAMAL...