Functional Programming

CS3100

Recap

Last Time:

- Why study programming languages?

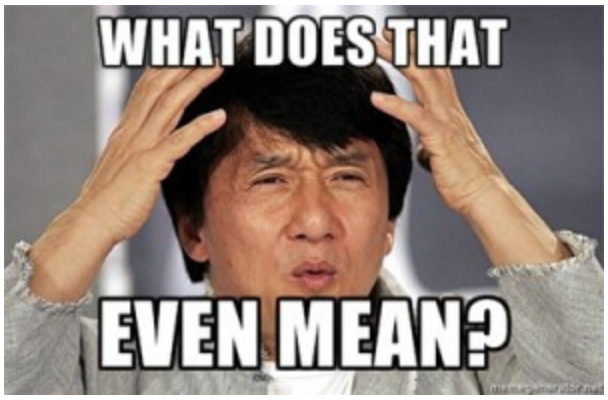
Today:

- Why functional programming matters?

See also the famous paper titled "Why Functional Programming Matters?" (https://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf) by John Huges.

In this part of the course, we will learn

🌟🌟 *Functional Programming* 🌟🌟



What is a functional language?

A functional language:

- defines computations as **mathematical functions**
- avoids mutable **state**

State: 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

- Functional never have side effects

The power of immutability:

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

Why study functional programming?

1. Functional programming languages predict the future.

1. Functional programming 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 programming languages predict the future.
2. Functional programming languages are *sometimes* used in the industry.

2. Functional Programmming in Industry

- Java 8 -- Oracle
- F#, C# 3.0, LINQ -- Microsoft
- Scala -- Twitter, Foursquare, LinkedIn
- Haskell -- Facebook, Barclays, AT&T
- Erlang -- Facebook, Amazon, WhatsApp
- OCaml -- Facebook, Bloomberg, Citrix, JaneStreet

Why study functional programming?

1. Functional programming languages predict the future.
2. Functional programming languages are *sometimes* used in the industry.
3. Functional programming languages are **elegant**.

Does aesthetics matter?

You'll often hear that functional programming code is beautiful, concise, stylish, refined, etc. But does it 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
- Algebraic datatypes and pattern matching
- First-class functions
- Static type-checking
- Automatic type inference
- Parametric polymorphism
- Garbage collection
- Modules

But no language is perfect...

- 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

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**

Five aspects of learning a PL

1. **Syntax:** How do you write language constructs?
2. **Semantics:** What do programs mean? (Type checking, evaluation rules)
3. **Idioms:** What are typical patterns for using language features to express your computation?
4. **Libraries:** What facilities does the language (or a third-party project) provide as “standard”? (E.g., file access, data structures)
5. **Tools:** What do language implementations provide to make your job easier? (E.g., top-level, debugger, GUI editor, ...)

- Breaking a new PL down into these pieces makes it easier to learn.

Our Focus

We focus on **semantics** and **idioms** for OCaml

- **Semantics** is like a meta-tool: it will help you learn
- **Idioms** will make you a better programmer in those languages

Libraries and **tools** are a secondary focus: throughout your career you’ll learn new ones on the job every year

- **Syntax** is a "fact"; almost always boring
  - People obsess over subjective preferences {yawn}
  - Class rule: We don’t complain about syntax

Fin.