CSE 4102: Programming Languages

Lecture 7: Functional programming wrap-up

Spring 2026

OCaml/functional programming overview

- Immutability/lack of side effects (but not purely)
- Higher-order
- Rich, strong static type system
  - ADTs
  - Polymorphism

Functional – lack of side effects

- What's a side effect?
  - Mutation/destructive updates to data structures – changing values in memory (e.g. "x = 5;", a reverse function that reverses the list in-line)
  - I/O
  - Exceptions
  - Anything a program does other than return a value

  - OCaml is not purely functional!

Benefits of functional programming

- Sharing

```
let l = [1; 2; 3; 4; 5]
let l’ = 0::l
```

  - Really good for implementing search problems (e.g., Hanoi on HW1, n-queens on HW2)

- Referential transparency (e.g., replace fib 5 with 8 everywhere)

- Don’t need to worry about evaluation order, race conditions, etc.

FP in other languages

- There's nothing stopping you!
  - (OK, this gets tricky when implementing non-destructive data structures)

```
int fib(int n) {
    return n <= 1 ? 1 : fib (n - 2) + fib (n - 1);
}
```

Higher-order programming

- Functions are just like any other value
- Functions can take and return functions (higher-order functions)
- Anonymous functions

- Allows code reuse, simplicity of language (e.g., currying instead of multi-argument functions)

HOP in other languages

- Python list documentation:
  - `squares = list(map(lambda x: x**2, range(10)))`

OCaml's type system

- ADTs allow expressing fairly detailed properties
  - Remember twothreetree from last time: every 2-3 tree can be a twothreetree, every twothreetree is a 2-3 tree (no "garbage")
  - In other languages: TypeScript, Swift
- Polymorphism
  - Code reuse
  - In other languages: many at least come close!