Lecture 30:

- Intro to OCaml

Announcements:

- HW-6 out, due Mon

OCaml Intro

Getting started with OCaml ... https://ocaml.org

- Download `opam` (see website, follow instructions)
- Make sure you can run `ocaml` from the command line

Basic (high-level) OCaml features:

(1) A (mostly) "**pure**" functional language

That is, pure functions:

- no **side effects** (do not modify state)
- some nice features: e.g., memoization
- can lead to smaller, reusable functions
- and programs that are easier to debug ... and better correctness guarantees

Most functions we write in OCaml will be pure

- but OCaml allows for some side effects (arrays, references, I/O)

OCaml Intro

(2) Functions are "**first-class**" objects ... like any other value

- pass functions as arguments, call within function
- return functions as values to other functions

... also allows partial functions (currying)

(3) **Static typing** ... and "strongly" typed

- Type checking done at compile time (statically)
- But also employs **type inference** (unobtrusive—w/out type annotations)

OCaml Intro

(4) A "**strict**" language that allows for "**lazy evaluation**"

- Strict languages "eagerly" evaluate function arguments
- Lazy languages defer evaluation until needed

(a) One benefit: (potentially) performance

- e.g., using quicksort can ask for first n items
- ... without sorting entire list

(b) Another benefit: "infinite" data structures

- and in particular, the ability to compute with them
- somewhat similar to iterators (or streams)

OCaml Intro

(c) Another benefit: programmer-defined control structures

- e.g., short circuit evaluation of `if-then-else`
- this means you don't need special constructs for control flow

In OCaml: ... sometimes called a "*suspension*"

```
# let x = (1/0) ;;                                        (* strict *)
Exception: Division_by_zero.

# let y = lazy (1/0) ;;                          (* lazy -- defers evaluation *)
# Lazy.force y ;;                                (* forces y to be evaluated *)
Exception: Division_by_zero.

# let z = lazy (Lazy.force y * 2) ;;            (* have to force y to do mult *)
# Lazy.force z ;;                          (* z still deferred, until forced *)
Exception: Division_by_zero.
```


OCaml Intro

(6) Plus more:

- user-defined "**algebraic types**"
- **pattern matching** with guards
- **parametric polymorphism** (with type inference)

Running OCaml (REPL) from the command line: `ocaml`

```
ocaml
OCaml version 5.2.0
Enter #help;; for help.

# ... type stuff here ...
```

- To exit type: `Control-d` or `#quit ;;`

Can run programs using: `ocaml file.ml`

(*) Can also use in VS Code (see https://ocaml.org)

OCaml Basics

Simple arithmetic

```
# 2 + 3 ;;
- : int = 5
# 5 / 2 ;;
- : int = 2
# 1.0 /. 2.0 ;;
- : float = 0.5
# 5 mod 2 ;;
- : int = 1
# 5 * -2 ;;
- : int = -10
# 2.5 +. 3.1 ;;
- : float = 5.6
# (+) 2 3 ;;
- : int = 5
```

The last one uses *infix* notation ... really (+) as a function

- ... you shouldn't write arithmetic expressions using infix notation

OCaml Basics

Comparison Operators: ... all the normal stuff

```
# 1 < 2 ;;
- : bool = true

# 1 == 1 ;;          (* can also use 1 = 1 *)
- : bool = true

# 1 != 1 ;;          (* can also use 1 <> 1 *)
- : bool = false
```

Logical operators: ... also the normal stuff

```
# not (true && (false || true)) ;;
- : bool = false
```

Q: What is wrong with the following?

```
# 1 < true ;;
Error: This expression has type bool but an expression
       was expected of type int

# 1 < 2.0 ;;
 Error: This expression has type float but an expression
        was expected of type int
```