Features of OCAML

- Higher order applicative language
- Call-by-value parameter passing
- Modern syntax
- Parametric polymorphism
  - Aka structural polymorphism
- Automatic garbage collection
- User-defined algebraic data types

Why learn OCAML?

- Many features not clearly in languages you have already learned
- Assumed basis for much research in programming language research
- OCAML is particularly efficient for programming tasks involving languages (eg parsing, compilers, user interfaces)
- Industrially Relevant:
  - Jane Street trades billions of dollars per day using OCaml programs
  - Major language supported at Bloomberg
- Similar languages: Microsoft F#, SML, Haskell, Scala

Session in OCAML

```ocaml
% ocaml
Objective Caml version 4.07.1
# (* Read-eval-print loop; expressions and declarations *)
  2 + 3;;  (* Expression *)
  - : int = 5
# 3 < 2;;
  - : bool = false
```

Declarations; Sequencing of Declarations

```ocaml
# let x = 2 + 3;;  (* declaration *)
  val x : int = 5
# let test = 3 < 2;;
  val test : bool = false
# let a = 1 let b = a + 4;;  (* Sequence of dec *)
  val a : int = 1
  val b : int = 5
```

Functions

```ocaml
# let plus_two n = n + 2;;
  val plus_two : int -> int = <fun>
# plus_two 17;;
  - : int = 19
```
**Functions**

```ocaml
let plus_two n = n + 2;;
```

```ocaml
plus_two 17;;
```

```ocaml
- : int = 19
```

**Environments**

- *Environments* record what value is associated with a given identifier.
- Central to the semantics and implementation of a language.
- **Notation**
  
  \[ \rho = \{ \text{name}_1 \mapsto \text{value}_1, \text{name}_2 \mapsto \text{value}_2, \ldots \} \]

  Using set notation, but describes a partial function.

  - Often stored as list, or stack.
  - To find value start from left and take first match.

**Global Variable Creation**

```ocaml
# 2 + 3;; (* Expression *)
```

```ocaml
// doesn't affect the environment
```

```ocaml
# let test = 3 < 2;; (* Declaration *)
```

```ocaml
val test : bool = false
```

```ocaml
// \( \rho_1 = \{ \text{test} \mapsto \text{false} \} \)
```

```ocaml
# let a = 1 let b = a + 4;; (* Seq of dec *)
```

```ocaml
// \( \rho_2 = \{ \text{b} \mapsto 5, \text{a} \mapsto 1, \text{test} \mapsto \text{false} \} \)
```

**New Bindings Hide Old**

```ocaml
// \( \rho_2 = \{ \text{b} \mapsto 5, \text{a} \mapsto 1, \text{test} \mapsto \text{false} \} \)
```

```ocaml
let test = 3.7;;
```

```ocaml
// What is the environment after this declaration?
```
// \( \rho_2 = \{ b \to 5, a \to 1, \text{test} \to \text{false} \} \)
let test = 3.7;;

What is the environment after this declaration?
// \( \rho_3 = \{ \text{test} \to 3.7, a \to 1, b \to 5 \} \)

Now it’s your turn
You should be able to start ACT1

// \( \rho_5 = \rho_3 = \{ \text{test} \to 3.7, a \to 1, b \to 5 \} \)
# let c =
  let b = a + a
// \( \rho_6 = \{ b \to 2 \} + \rho_3 \)
// =\{ b \to 2, \text{test} \to 3.7, a \to 1 \} \)
  in b * b;;
val c : int = 4
// \( \rho_7 = \{ c \to 4, \text{test} \to 3.7, a \to 1, b \to 5 \} \)
# b;;
- : int = 5
Local let binding

// ρ5 ⊢ ρ3 = {test → 3.7, a → 1, b → 5}
# let c =
  let b = a + a
// ρ6 = {b → 2} + ρ3
// = {b → 2, test → 3.7, a → 1}
  in b * b;
val c : int = 4
// ρ7 = {c → 4, test → 3.7, a → 1, b → 5}
# b;;
- : int = 5

Functions

let plus_two n = n + 2;;
val plus_two : int -> int = <fun>
# plus_two 17;;
- : int = 19

Nameless Functions (aka Lambda Terms)

fun n -> n + 2;;
plus_two 17;;
- : int = 19

Using a nameless function

# (fun x -> x * 3) 5;; (* An application *)
- : int = 15
# ((fun y -> y +. 2.0), (fun z -> z * 3));;
(* As data *)
- : (float -> float) * (int -> int) = (<fun>, <fun>)

Note: in fun v -> exp(v), scope of variable is only the body exp(v)
Values fixed at declaration time

# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# plus_x 3;;

What is the result?

Values fixed at declaration time

# let x = 7;; (* New declaration, not an update *)
val x : int = 7

# plus_x 3;;;

What is the result this time?

Values fixed at declaration time

# let x = 7;; (* New declaration, not an update *)
val x : int = 7

# plus_x 3;;
- : int = 15

Question

- Observation: Functions are first-class values in this language

- Question: What value does the environment record for a function variable?

- Answer: a closure
A closure is a pair of an environment and an association of a formal parameter (the input variables) with an expression (the function body), written:

\[ f \rightarrow < (v_1, \ldots, v_n) \rightarrow \text{exp}, \rho_f > \]

Where \( \rho_f \) is the environment in effect when \( f \) is defined (if \( f \) is a simple function)

* Will come back to the “formal parameter”

---

Now it’s your turn

You should be able to complete ACT1

---

When \( \text{plus}_x \) was defined, had environment:

\[ \rho_{\text{plus}_x} = \{\ldots, x \rightarrow 12, \ldots\} \]

Recall: \( \text{let } \text{plus}_x \ y = y + x \)

is really \( \text{let } \text{plus}_x = \text{fun } y -> y + x \)

Closure for \( \text{fun } y -> y + x \):

\[ < y \rightarrow y + x, \rho_{\text{plus}_x} > \]

Environment just after \( \text{plus}_x \) defined:

\[ \{\text{plus}_x \rightarrow < y \rightarrow y + x, \rho_{\text{plus}_x} >\} + \rho_{\text{plus}_x} \]