Announcements

• Assignment #1 is up on the web page
  - prep configuration doesn’t work yet
  - turnin does not work yet

• Deadline Extensions for the quarter:
  - Four “late days”, used as “whole unit”
  - 5 mins late = 1 late day
  - Plan ahead, no other extensions

Recap: ML’s holy trinity

• Everything is an expression
• Everything has a value
• Everything has a type

Word from our sponsor ...

• Programming Assignments done ALONE
• We use plagiarism detection software
  - Have code from all previous classes
  - MOSS is fantastic, plagiarize at your own risk
• Zero Tolerance
  - offenders punished ruthlessly
• Please see academic integrity statement
Recap: Interacting with ML

“Read-Eval-Print” Loop

Repeat:
1. System reads expression e
2. System evaluates e to get value v
3. System prints value v and type t

What are these expressions, values and types?
- int, bool, products (tuples), lists, …

Recap: Tuples vs. Lists?

What’s the difference?
- Tuples:
  - Different types, but fixed number:
    - pair = 2 elts
      - (3, “abcd”) (int * string)
    - triple = 3 elts
      - (3, “abcd”, (3.5,4.2)) (int * string * (real * real))
- Lists:
  - Same type, unbounded number:
    - [3;4;5;6;7] int list
- Syntax:
  - Tuples = comma
  - Lists = semicolon

More on Lists

Goto Ocaml interpreter…

So far, a fancy calculator…

... what do we need next?
Variables and bindings

\[
\text{let } x = e;
\]
“Bind the value of expression \(e\) to the variable \(x\)”

Later declared expressions can use \(x\)
- Most recent “bound” value used for evaluation

\[
\# \text{ let } x = 2+2;; \\
\text{val } x : \text{ int} = 4
\]

Undeclared variables (i.e. without a value binding) are not accepted!

\[
\# \text{ let } p = a + 1;; \\
\text{Characters 8-9:} \\
\text{ let } p = a + 1 ;; \\
\wedge \\
\text{Unbound value } a
\]

Catches many bugs due to typos

Local bindings

... for expressions using “temporary” variables

\[
\text{let} \\
\text{tempVar } = x + 2 \ast y \\
\text{in} \\
\text{tempVar } \ast \text{tempVar};;
\]

- \(\text{tempVar}\) is bound only inside expr body from \(\text{in} [...];\)
- Not visible (“in scope”) outside
Binding by Pattern-Matching

Simultaneously bind several variables

```ocaml
# let (x, y, z) = (2+3,"a"^"b", 1::[2]);;
val x : int = 5
val y : string = "ab"
val z : int list = [1;2]
```

But what of:

```ocaml
# let h::t = [1;2;3];;
Warning P: this pattern-matching not exhaustive.
val h : int = 1
val t : int list = [2,3]
```

Why is it whining?

```ocaml
# let h::t = [];
Exception: Match_failure
# let l = [1;2;3];
val l = [1;2;3] : list
# let h::t = l;
Warning: Binding not exhaustive
val h = 1 : int
val t = [2,3] : int
```

In general l may be empty (match failure!)
Another useful early warning

Next: functions, but remember...

Expression Value

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Body Expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>int -&gt; int</td>
<td>fn</td>
</tr>
</tbody>
</table>
```

A function is a value!

Complex types: Functions!

How a call (“application”) is evaluated:
1. Evaluate argument
2. Bind formal to arg value
3. Evaluate “Body expr”

Everything is an expression
Everything has a value
Everything has a type
A Problem

Parameter (formal)  Body  
\[
\text{fun } x \rightarrow x + 1; \quad \text{fn}
\]

Can functions only have a single parameter?

How a call (“application”) is evaluated:
1. Evaluate argument
2. Bind formal to arg value
3. Evaluate “Body expr”

A Solution: Simultaneous Binding

Parameter (formal)  Body  
\[
\text{fun } (x, y) \rightarrow x < y; \quad \text{fn}
\]

Can functions only have a single parameter?

How a call (“application”) is evaluated:
1. Evaluate argument
2. Bind formal to arg value
3. Evaluate “Body expr”

Another Solution

Parameter (formal)  Body  
\[
\text{fun } x \rightarrow \text{fun } y \rightarrow x < y; \quad \text{fn}
\]

Whoa! A function can return a function

```
# let lt = fun x -> fn y -> x < y;
val lt : int -> int -> bool = fn
# let is5Lt = lt 5;
val is5Lt : int -> bool = fn
# is5Lt 10;
val it : bool = true;
# is5Lt 2;
val it : bool = false;
```

And how about...

Parameter (formal)  Body  
\[
\text{fun } f \rightarrow \text{fun } x \rightarrow \text{not}(f x); \quad \text{fn}
\]

A function can also take a function argument

```
# let neg = fun f -> fun x -> not(f x);
val neg : ('a -> bool) -> ('a -> bool) = fn
# let is5gte = neg is5lt;
val is5gte : int -> bool = fn
# is5gte 10;
val it : bool = false;
# is5gte 2;
val it : bool = true;
(*...odd, even...*)
```
A shorthand for function binding

```ocaml
# let neg = fun f -> fun x -> not (f x);
... # let neg f x = not (f x);
val neg : int -> int -> bool = fn
# let is5gte = neg is5lt;
val is5gte : int -> bool = fn;
# is5gte 10;
val it : bool = false;
# is5gte 2;
val it : bool = true;
```

Put it together: a “filter” function

```ocaml
# let rec filter f l =
  match l with
  | [] -> []
  | (h::t) -> if f h then h::(filter f t)
    else (filter f t);
val filter : ('a->bool)->'a list->'a list = fn
# let list1 = [1,31,12,4,7,2,10];
# is5gte = neg is5lt;
val is5gte : int -> bool = fn;
# is5gte 10;
val it : bool = false;
# is5gte 2;
val it : bool = true;
```

Put it together: a “partition” function

```ocaml
# let partition f l = (filter f l, filter (neg f) l);
val partition : ('a->bool)->'a list->'a list * 'a list = fn
# let partition fl= (filter f l filter (neg f) l);
val partition : ('a->bool)->'a list->'a list * 'a list = fn
```

A little trick ...

```ocaml
# 2 <= 3;; ... val it : bool = true
# "ba" <= "ab";; val it : bool = false
# 2 <= 3;; ... val it : bool = true
# "ba" <= "ab";; val it : bool = false
```

A little trick ...

```ocaml
# let list1 = [1,31,12,4,7,2,10];
# let list1 = [1,31,12,4,7,2,10];
# partition is5lt list1
val it : (int list * int list) = ([31,12,7,10], [1,2,10])
# partition even list1;
val it : (int list * int list) = ([31,12,7,10], [1,2,10])
```
Put it together: a “quicksort” function

```ml
let rec sort l =
  match l with
  | [] -> []
  | (h::t) ->
    let (l,r) = partition ((<) h) t in
    (sort l)@(h::(sort r))
```