1/24/23
Now it’s your turn
You should be able to complete ACT1

Tuples as Values
// ρ7 = {c → 4, test → 3.7, 
# a → 1, b → 5}
# let s = (5,"hi",3.2);;
val s : int * string * float = (5, "hi", 3.2)
// ρ8 = {s → (5, "hi", 3.2), 
# c → 4, test → 3.7, 
# a → 1, b → 5}

Pattern Matching with Tuples
/ ρ8 = {s → (5, "hi", 3.2), 
# c → 4, test → 3.7, 
# a → 1, b → 5}
# let (a,b,c) = s;;  (* (a,b,c) is a pattern *)
val a : int = 5
val b : string = "hi"
val c : float = 3.2
# let x = 2, 9.3;; (* tuples don't require parens in Ocaml *)
val x : int * float = (2, 9.3)

Nested Tuples
# (*Tuples can be nested *)
let d = ((1,4,62),("bye",15),73.95);;
val d : (int * int * int) * (string * int) * float = ((1, 4, 62), ("bye", 15), 73.95)
# (*Patterns can be nested *)
let (p,(st,_),_) = d;; (* _ matches all, binds nothing *)
val p : int * int * int = (1, 4, 62)
val st : string = "bye"

Functions on tuples
# let plus_pair (n,m) = n + m;;
val plus_pair : int * int -> int = <fun>
# plus_pair (3,4);;
- : int = 7
# let double x = (x,x);;
val double : 'a -> 'a * 'a = <fun>
# double 3;;
- : int * int = (3, 3)
# double "hi";;
- : string * string = ("hi", "hi")
Each clause: pattern on left, expression on right
Each x, y has scope of only its clause
Use first matching clause

```ocaml
let triple_to_pair triple = match triple with
    (0, x, y) -> (x, y)
  | (x, 0, y) -> (x, y)
  | (x, y, _) -> (x, y);;
```

Closure for `plus_pair`
Assume $\rho_{plus\_pair}$ was the environment just before `plus_pair` defined
Closure for `plus_pair`:

$$<(n,m) \rightarrow n + m, \rho_{plus\_pair} >$$

Environment just after `plus_pair` defined:

$$\{ plus\_pair \rightarrow <(n,m) \rightarrow n + m, \rho_{plus\_pair} > \} + \rho_{plus\_pair}$$

Save the Environment!
A **closure** is a pair of an environment and an association of a pattern (e.g. $(v_1, \ldots, v_n)$ giving the input variables) with an expression (the function body), written:

$$< (v_1, \ldots, v_n) \rightarrow exp, \rho >$$

Where $\rho$ is the environment in effect when the function is defined (for a simple function)

Evaluating declarations
Evaluation uses an environment $\rho$
To evaluate a (simple) declaration `let x = e`

- Evaluate expression $e$ in $\rho$ to value $v$
- Update $\rho$ with $x$ $v$: $\{x \rightarrow v\} + \rho$

Update: $\rho_1 + \rho_2$ has all the bindings in $\rho_1$ and all those in $\rho_2$ that are not rebound in $\rho_1$

$$\{x \rightarrow 2, y \rightarrow 3, a \rightarrow \text{“hi”}\} + \{y \rightarrow 100, b \rightarrow 6\} = \{x \rightarrow 2, y \rightarrow 3, a \rightarrow \text{“hi”}, b \rightarrow 6\}$$

Evaluating expressions in OCaml
Evaluation uses an environment $\rho$
A constant evaluates to itself, including primitive operators like $+$ and $=$
To evaluate a variable, look it up in $\rho$: $\rho(v)$
To evaluate a tuple $(e_1, \ldots, e_n)$,
- Evaluate each $e_i$ to $v_i$, right to left for Ocaml
- Then make value $(v_1, \ldots, v_n)$

Evaluating expressions in OCaml
To evaluate uses of $+$, $\_\_\_$, etc, eval args, then do operation
Function expression evaluates to its closure
To evaluate a local dec: `let x = e1 in e2`
- Eval $e1$ to $v$, then eval $e2$ using $\{x \rightarrow v\} + \rho$
To evaluate a conditional expression:
  - if $b$ then $e1$ else $e2$
    - Evaluate $b$ to a value $v$
    - If $v$ is True, evaluate $e1$
    - If $v$ is False, evaluate $e2$
Evaluation of Application with Closures

- Given application expression f e
- In Ocaml, evaluate e to value v
- In environment ρ, evaluate left term to closure, c = <(x₁,...,xₙ) → b, ρ'>
  - (x₁,...,xₙ) variables in (first) argument
  - v must have form (v₁,...,vₙ)
- Update the environment ρ'' to
  ρ'' = {x₁ → v₁,..., xₙ → vₙ} + ρ'
- Evaluate body b in environment ρ''

Evaluating expressions in OCaml

- Evaluation uses an environment ρ
  - Eval (e, ρ)
- A constant evaluates to itself, including primitive operators like + and =
  - Eval (c, ρ) => Val c
- To evaluate a variable v, look it up in ρ:
  - Eval (v, ρ) => Val (ρ(v))

To evaluate a tuple (e₁,...,eₙ),

- Evaluate each eᵢ to vᵢ, right to left for Ocaml
- Then make value (v₁,...,vₙ)

Function expression evaluates to its closure
- Eval (fun x -> e, ρ) => Val < x -> e, ρ>

Extra Material for Extra Credit
Evaluating expressions in OCaml

To evaluate a conditional expression:
if b then e₁ else e₂
- Evaluate b to a value v
- If v is True, evaluate e₁
- If v is False, evaluate e₂

Eval(if b then e₁ else e₂, r) =>
Eval(if Val true then e₁ else e₂, r) => Eval(e₁, r)
Eval(if Val false then e₁ else e₂, r) => Eval(e₂, r)

Evaluation of Application with Closures

Given application expression f e
In Ocaml, evaluate e to value v
In environment ρ, evaluate left term to closure,
c = (x₁,...,xₙ) → b, ρ’
(x₁,...,xₙ) variables in (first) argument
v must have form (v₁,...,vₙ)
Update the environment ρ’ to
ρ” = {x₁ → v₁,..., xₙ → vₙ} + ρ’
Evaluate body b in environment ρ”

Evaluation of Application of plus_x;;

Have environment:
ρ = {plus_x → <y → y + x, ρ_{plus_x} >, ..., y → 19, x → 17, z → 3, ...}
where ρ_{plus_x} = {x → 12, ..., y → 24, ...}
Eval (plus_x z, ρ) =>
Eval(plus_x (Eval(z, ρ)), ρ) =>
Eval(plus_x (Val 3), ρ) => ...

Evaluation of Application of plus_x;;

Have environment:
ρ = {plus_x → <y → y + x, ρ_{plus_x} >, ..., y → 19, x → 17, z → 3, ...}
where ρ_{plus_x} = {x → 12, ..., y → 24, ...}
Eval (plus_x z, ρ) =>
Eval (plus_x (Eval(z, ρ)), ρ) =>
Eval (plus_x (Val 3), ρ) =>
Eval ((Eval(plus_x, ρ)) (Val 3), ρ) => ...

Evaluation of Application of plus_x;;

Have environment:
ρ = {plus_x → <y → y + x, ρ_{plus_x} >, ..., y → 19, x → 17, z → 3, ...}
where ρ_{plus_x} = {x → 12, ..., y → 24, ...}
Eval (plus_x z, ρ) =>
Eval (plus_x (Eval(z, ρ)), ρ) =>
Eval (plus_x (Val 3), ρ) =>
Eval ((Eval(plus_x, ρ)) (Val 3), ρ) => ...

Evaluation of Application with Closures

Eval(f e, ρ) => Eval(f (Eval(e, ρ)), ρ)
Eval(f (Val v), ρ) => Eval((Eval(f, ρ)) (Val v), ρ)
Eval((Val <(x₁,...,xₙ) → b, ρ'>)(Val (v₁,...,vₙ)), ρ) =>
Eval(b, {x₁ → v₁,..., xₙ → vₙ} + ρ')
Evaluation of Application of plus_x;;

Have environment:
\[ \rho = \{ \text{plus}_x \rightarrow y \rightarrow y + x, \rho_{\text{plus}_x} >, \ldots, y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \ldots \} \]

where \( \rho_{\text{plus}_x} = \{ x \rightarrow 12, \ldots, y \rightarrow 24, \ldots \} \)

\[ \text{Eval (plus}_x(\text{Val 3}, \rho) \Rightarrow \]
\[ \text{Eval ((Val<y \rightarrow y + x, \rho_{\text{plus}_x}>)(Val 3), \rho) } \Rightarrow \ldots \]

1/25/23 28

Evaluation of Application of plus_x;;

Have environment:
\[ \rho = \{ \text{plus}_x \rightarrow y \rightarrow y + x, \rho_{\text{plus}_x} >, \ldots, y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \ldots \} \]

where \( \rho_{\text{plus}_x} = \{ x \rightarrow 12, \ldots, y \rightarrow 24, \ldots \} \)

\[ \text{Eval ((Val<y \rightarrow y + x, \rho_{\text{plus}_x}>)(Val 3), \rho) } \Rightarrow \ldots \]

1/25/23 29

Evaluation of Application of plus_x;;

Have environment:
\[ \rho = \{ \text{plus}_x \rightarrow y \rightarrow y + x, \rho_{\text{plus}_x} >, \ldots, y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \ldots \} \]

where \( \rho_{\text{plus}_x} = \{ x \rightarrow 12, \ldots, y \rightarrow 24, \ldots \} \)

\[ \text{Eval ((Val<y \rightarrow y + x, \rho_{\text{plus}_x}>)(Val 3), \rho) } \Rightarrow \ldots \]
\[ \text{Eval (y + x, \{y \rightarrow 3\} } + \rho_{\text{plus}_x} \} \Rightarrow \ldots \]

1/25/23 30

Evaluation of Application of plus_x;;

Have environment:
\[ \rho = \{ \text{plus}_x \rightarrow y \rightarrow y + x, \rho_{\text{plus}_x} >, \ldots, y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \ldots \} \]

where \( \rho_{\text{plus}_x} = \{ x \rightarrow 12, \ldots, y \rightarrow 24, \ldots \} \)

\[ \text{Eval (y + Eval(x, \{y \rightarrow 3\} } + \rho_{\text{plus}_x} \} + \rho_{\text{plus}_x} ) \Rightarrow \]
\[ \text{Eval(y + Val 12, } \{y \rightarrow 3\} + \rho_{\text{plus}_x} ) \Rightarrow \ldots \]

1/25/23 31

Evaluation of Application of plus_x;;

Have environment:
\[ \rho = \{ \text{plus}_x \rightarrow y \rightarrow y + x, \rho_{\text{plus}_x} >, \ldots, y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \ldots \} \]

where \( \rho_{\text{plus}_x} = \{ x \rightarrow 12, \ldots, y \rightarrow 24, \ldots \} \)

\[ \text{Eval(y + Val 12, } \{y \rightarrow 3\} + \rho_{\text{plus}_x} ) \Rightarrow \]
\[ \text{Eval(Eval(y, } \{y \rightarrow 3\} + \rho_{\text{plus}_x} ) + \]
\[ \text{Val 12, } \{y \rightarrow 3\} + \rho_{\text{plus}_x} ) \Rightarrow \ldots \]

1/25/23 32
Evaluation of Application of plus_x;;
- Have environment:
  \[ \rho = \{ \text{plus}_x \to <y \to y + x, \rho_{\text{plus}_x}>, \ldots, y \to 19, x \to 17, z \to 3, \ldots \} \]
  where \[ \rho_{\text{plus}_x} = \{ x \to 12, \ldots, y \to 24, \ldots \} \]
- \[ \text{Eval}((\text{Eval}(y, \{ y \to 3 \} + \rho_{\text{plus}_x}) + \text{Val}(12, \{ y \to 3 \} + \rho_{\text{plus}_x})) \Rightarrow \]
- \[ \text{Eval}((\text{Val}(3) + \text{Val}(12), \{ y \to 3 \} + \rho_{\text{plus}_x})) \Rightarrow \]
- \[ \text{Val}(3 + 12) = \text{Val}(15) \]

Evaluation of Application of plus_pair
- Assume environment
  \[ \rho = \{ x \to 3, \ldots, \text{plus}_\text{pair} \to <(n,m) \to n + m, \rho_{\text{plus}_\text{pair}}>, \rho_{\text{plus}_\text{pair}} > \} \]
  where \[ \rho_{\text{plus}_\text{pair}} = \{ x \to 3, \ldots, \text{plus}_\text{pair} \to <(n,m) \to n + m, \rho_{\text{plus}_\text{pair}} > \} \]
- \[ \text{Eval}((\text{Eval}(\text{Val}(4), \text{Val}(3)), \rho_{\text{plus}_\text{pair}}) + \rho_{\text{plus}_\text{pair}}) \Rightarrow \]
- \[ \text{Eval}((\text{Val}(4) + \text{Val}(3), \rho_{\text{plus}_\text{pair}}) + \rho_{\text{plus}_\text{pair}}) \Rightarrow 7 \]

Closure question
- If we start in an empty environment, and we execute:
  \[ \text{let } f = \text{fun } n \to n + 5; \]
  (*) 0 *)
  \[ \text{let } \text{pair}_\text{map} g (n,m) = (g n, g m); \]
  \[ \text{let } f = \text{pair}_\text{map} f; \]
  \[ \text{let } a = f(4,6); \]
  What is the environment at (*) 0 *)?

Answer
- \[ \rho_0 = \{ f \to <n \to n + 5, \{ \} > \} \]
Closure question

If we start in an empty environment, and we execute:

```plaintext
let f = fun => n + 5;;
let pair_map g (n,m) = (g n, g m);;
(* 1 *)
let f = pair_map f;;
let a = f (4,6);;
```

What is the environment at (* 1 *)?

```
ρ₀ = {f → <n → n + 5, { }}>
let pair_map g (n,m) = (g n, g m);;
ρ₁ = {pair_map →
   <g → fun (n,m) -> (g n, g m),
   {f → <n → n + 5, { }}>,
   f → <n → n + 5, { }}>}
```

Closure question

If we start in an empty environment, and we execute:

```plaintext
let f = fun => n + 5;;
let pair_map g (n,m) = (g n, g m);;
(* 2 *)
let f = pair_map f;;
let a = f (4,6);;
```

What is the environment at (* 2 *)?

```
ρ₀ = {f → <n → n + 5, { }}> 
ρ₁ = {pair_map → <g→fun (n,m) -> (g n, g m), 
       {f → <n → n + 5, { }}>},
       f → <n → n + 5, { }}>}
```

Evaluate pair_map f

```
ρ₀ = {f → <n → n + 5, { }}> 
ρ₁ = {pair_map →<g→fun (n,m) -> (g n, g m), ρ₀>,
       f→<n → n + 5, { }}>}
let f = pair_map f;;
```

```
Eval(pair_map f, ρ₁) =
```

Evaluate pair_map f

```
ρ₀ = {f → <n → n + 5, { }}> 
ρ₁ = {pair_map →<g→fun (n,m) -> (g n, g m), ρ₀>,
       f→<n → n + 5, { }}>}
Eval(pair_map f, ρ₁) =>
Eval(pair_map (Eval(f, ρ₁)), ρ₁) =>
Eval(pair_map (Val<n → n + 5, { }>, ρ₁)) =>
Eval((Eval(pair_map, ρ₁))(Val<n → n+5, { }>, ρ₁)) =>
Eval((Val (<g→fun (n,m) -> (g n, g m), ρ₀>))
   (Val <n → n + 5, { }>, ρ₁)) =>
Eval(fun (n,m)->(g n, g m), {g→<n→n + 5, { }>}+ρ₀)
=>
```
Evaluate \( \text{pair}_\text{map} \, f \)

\[
\rho_0 = \{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\rho_1 = \{ \text{pair}_\text{map} \rightarrow g \rightarrow \text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m), \]
\[
f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{Eval}(\text{pair}_\text{map} \, f, \rho_1) \Rightarrow \ldots \Rightarrow \text{Eval}(\text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m), \{ g \rightarrow n \rightarrow n + 5, \{ \} \} + \rho_0) = \text{Val} \, (\langle n, m \rangle \rightarrow (g \, n, \, g \, m), \{ g \rightarrow n \rightarrow n + 5, \{ \} \})
\]

\[
\rho_2 = \{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

(*Remember: the original \( f \) is now removed from \( \rho_2 \)*)

Answer

\[
\rho_2 = \{ \text{pair}_\text{map} \rightarrow g \rightarrow \text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m), \}
\]

\[
f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{let} \, f = \text{pair}_\text{map} \, f;;
\]

\[
\rho_2 = \{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{pair}_\text{map} \rightarrow g \rightarrow \text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m),
\]

\[
\{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

(*Remember: the original \( f \) is now removed from \( \rho_2 \)*)

Closure question

If we start in an empty environment, and we execute:

\[
\text{let} \, f = \text{fun} \Rightarrow n + 5;;
\]

\[
\text{let} \, \text{pair}_\text{map} \, g \, (n,m) = (g \, n, \, g \, m);;
\]

\[
\text{let} \, f = \text{pair}_\text{map} \, f;;
\]

\[
\text{let} \, a = f \, (4,6);;
\]

(* 3 *)

What is the environment at (* 3 *)?

Final Evaluation?

\[
\rho_2 = \{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{pair}_\text{map} \rightarrow g \rightarrow \text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m),
\]

\[
\{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{let} \, a = f \, (4,6);;
\]

Evaluate \( f \, (4,6);; \)

\[
\rho_2 = \{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{pair}_\text{map} \rightarrow g \rightarrow \text{fun} \, (n,m) \rightarrow (g \, n, \, g \, m),
\]

\[
\{ f \rightarrow n \rightarrow n + 5, \{ \} \}
\]

\[
\text{Eval}(f \, (4,6), \rho_2) = \ldots
\]
Let $\rho = \{ n \rightarrow 4, \ m \rightarrow 6, \ g \rightarrow n \rightarrow n + 5, \ \{ \}>, \ f \rightarrow n \rightarrow n + 5, \ \{ \} \}> \}$

\[
\begin{align*}
\text{Eval}((g n, \text{Eval}((\text{Val}(6), \rho'))), \rho') &=> \\
\text{Eval}((g n, \text{Eval}((\text{Val}(n \rightarrow n + 5, \{ \}>)(\text{Val}(6), \rho'))), \rho') &=> \\
\text{Eval}((g n, \text{Eval}(n \rightarrow n + 5, \{ n \rightarrow 6 \})), \rho') &=> \\
\text{Eval}((g n, \text{Eval}(n \rightarrow n + 5, \{ n \rightarrow 6 \})), \rho') &=> \\
\text{Eval}((g n, \text{Eval}((\text{Val}(5), \{ n \rightarrow 6 \})), \rho')) &=> \\
\text{Eval}((g n, \text{Eval}((\text{Val}(5), \{ n \rightarrow 6 \})), \rho')) &=> \\
\text{Eval}((g n, \text{Eval}((\text{Val}(n, \{ n \rightarrow 6 \})) + (\text{Val}(5), \{ n \rightarrow 6 \})), \rho')) &=> \\
\text{Eval}((g n, \text{Eval}((\text{Val}(6) + (\text{Val}(5), \{ n \rightarrow 6 \})), \rho')) &=>
\end{align*}
\]
Evaluate \( f(4,6) \);

Let \( \rho' = \{ n \mapsto 4, m \mapsto 6, g \mapsto \langle n \mapsto n+5, \{ \rangle, f \mapsto \langle n \mapsto n+5, \{ \rangle \} \}\) 

\[
\begin{align*}
\text{Eval((Eval((Val<n\mapsto n+5, \{ \}))(Val 4), \rho'), Val 11), \rho') =} \\
\text{Eval((Eval(n+5, \{n \mapsto 4\}))(Val 11), \rho')) =>} \\
\text{Eval((Eval(n+(Val 5),\{n \mapsto 4\}))(Val 11), \rho')) =>} \\
\text{Eval((Eval(n),(Val 11)), \rho')) =>} \\
\text{1/25/23 58}
\end{align*}
\]

End of Extra Material for Extra Credit

Recursive Functions

\[
\begin{align*}
\text{let rec factorial n =} \\
\text{if n = 0 then 1 else n * factorial (n - 1);;} \\
\text{val factorial : int -> int = <fun>} \\
\text{# factorial 5;;} \\
\text{- : int = 120} \\
\text{# (* rec is needed for recursive function declarations *)}
\end{align*}
\]

Recursion Example

Compute \( n^2 \) recursively using:

\[ n^2 = (2 \times n - 1) + (n - 1)^2 \]

\[
\begin{align*}
\text{let rec nthsq n =} \\
\text{match n with} \\
\text{0 -> 0} \\
\text{| n -> (2 \times n - 1) + nthsq (n - 1);} \\
\text{val nthsq : int -> int = <fun>} \\
\text{# nthsq 3;;} \\
\text{- : int = 9}
\end{align*}
\]

Structure of recursion similar to inductive proof

Recursion and Induction

\[
\begin{align*}
\text{let rec nthsq n = match n with} \\
\text{0 -> 0} \\
\text{| n -> (2 \times n - 1) + nthsq (n - 1);} \\
\text{val nthsq : int -> int = <fun>} \\
\text{# nthsq 3;;} \\
\text{- : int = 9}
\end{align*}
\]

Lists

- List can take one of two forms:
  - Empty list, written \([\ ]\)
  - Non-empty list, written \(x :: xs\)
    - \(x\) is head element, \(xs\) is tail list, :: called “cons”
    - Syntactic sugar: \([x] == x :: [\ ]\)
  - \([x1; x2; \ldots; xn] == x1 :: x2 :: \ldots :: xn :: [\ ]\)
Lists

# let fib5 = [8;5;3;2;1;1];;
val fib5 : int list = [8; 5; 3; 2; 1; 1]
# let fib6 = 13 :: fib5;;
val fib6 : int list = [13; 8; 5; 3; 2; 1]

Fibonacci sequence implemented in OCaml.

Lists are Homogeneous

# let bad_list = [1; 3.2; 7];;
Characters 19-22:

This expression has type float but is here used with type int

Question

Which one of these lists is invalid?

1. [2; 3; 4; 6]
2. [2,3; 4,5; 6,7]
3. [(2.3,4); (3.2,5); (6,7.2)]
4. ["hi"; "there"; "wahcha"; [ ]; [“doin”]]

Answer

Which one of these lists is invalid?

1. [2; 3; 4; 6]
2. [2,3; 4,5; 6,7]
3. [(2.3,4); (3.2,5); (6,7.2)]
4. ["hi"; "there"; "wahcha"; [ ]; [“doin”]]

3 is invalid because of last pair

Functions Over Lists

# let rec double_up list =
    match list with
    [ ] -> [ ] /* pattern before ->, expression after * /
    | (x :: xs) -> (x :: x :: double_up xs);;
val double_up : 'a list -> 'a list = <fun>

# let fib5_2 = double_up fib5;;
val fib5_2 : int list = [8; 8; 5; 5; 3; 3; 2; 2; 1; 1; 1; 1]

Functions Over Lists

# let silly = double_up ["hi"; "there"];;
val silly : string list = ["hi"; "hi"; "there"; "there"]

# let rec poor_rev list =
    match list with
    [] -> []
    | (x::xs) -> poor_rev xs @ [x];;
val poor_rev : 'a list -> 'a list = <fun>

# poor_rev silly;;
- : string list = ["there"; "there"; "hi"; "hi"]