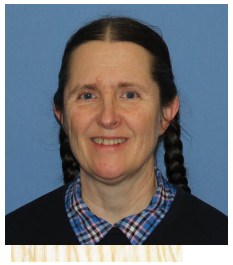
Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC



https://courses.engr.illinois.edu/cs421/sp2023

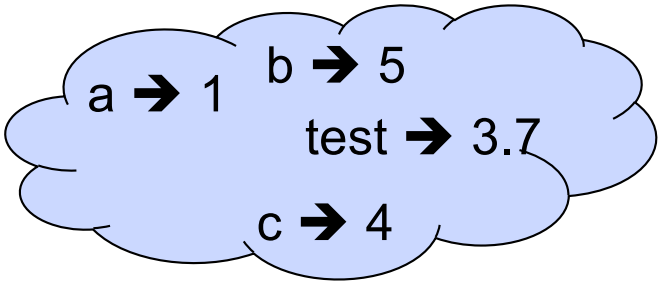
Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Now it’s your turn

You should be able to complete ACT1

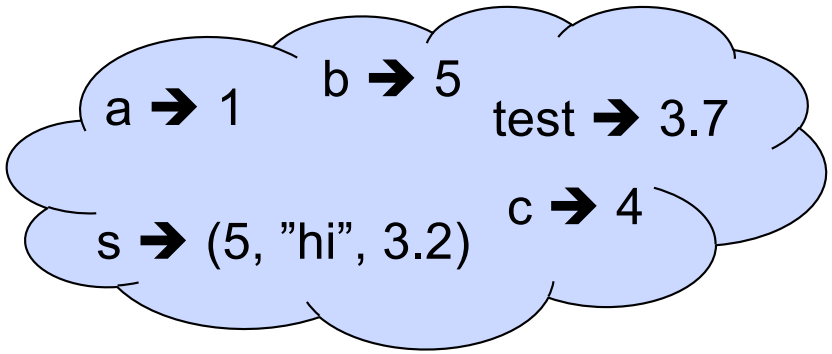
Tuples as Values

// ρ_7 = {c → 4, test → 3.7,
a → 1, b → 5}



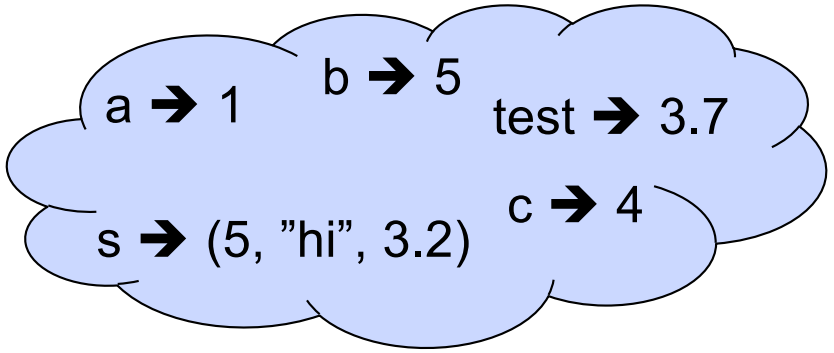

```
# let s = (5,"hi",3.2);;
val s : int * string * float = (5, "hi", 3.2)
```

// ρ_8 = {s → (5, "hi", 3.2),
c → 4, test → 3.7,
a → 1, b → 5}

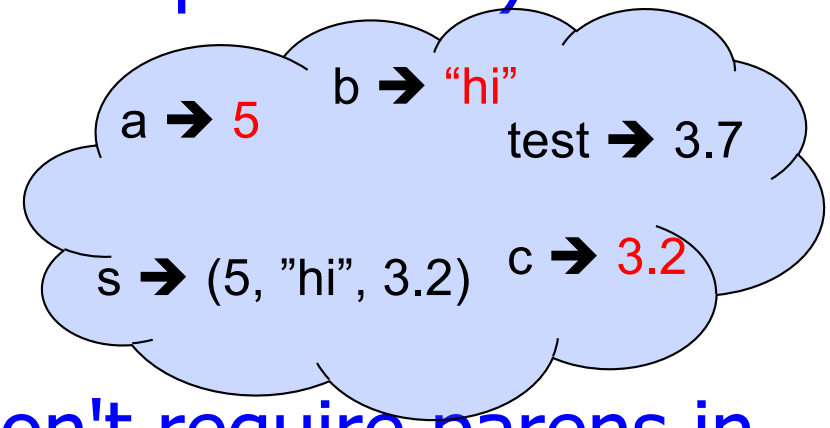


Pattern Matching with Tuples

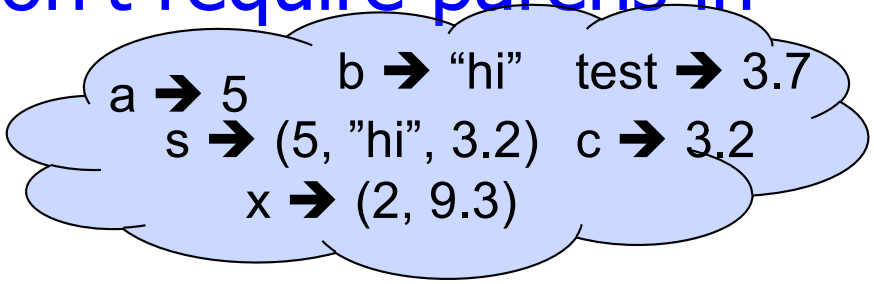
/ ρ_8 = {s → (5, "hi", 3.2),
c → 4, test → 3.7,
a → 1, b → 5}

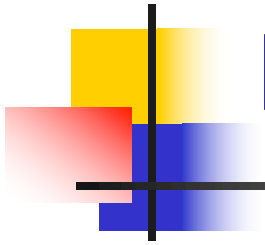



```
# let (a,b,c) = s;;  (* (a,b,c) is a pattern *)
val a : int = 5
val b : string = "hi"
val c : float = 3.2
```



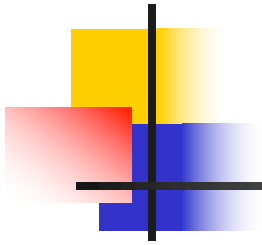

```
# let x = 2, 9.3;; (* tuples don't require parens in
   Ocaml *)
val x : int * float = (2, 9.3)
```





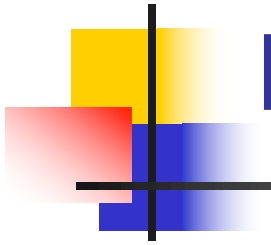
Nested Tuples

```
#  (*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")
```



Match Expressions

```
# let triple_to_pair triple =
    match triple
    with (0, x, y) -> (x, y)
    | (x, 0, y) -> (x, y)
    | (x, y, _) -> (x, y);;
val triple_to_pair : int * int * int -> int * int =
    <fun>
```

- Each clause: pattern on left, expression on right
- Each x, y has scope of only its clause
- Use first matching clause

Closure for plus_pair

- Assume ρ_{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:

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

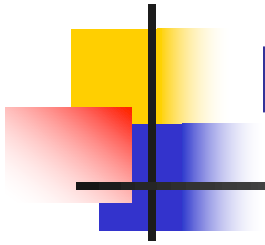
$$+ \rho_{plus\_pair}$$

Save the Environment!

- A *closure* is a pair of an environment and an association of a pattern (e.g. (v1,…,vn) giving the input variables) with an expression (the function body), written:

$$< (v1,\ldots,vn) \rightarrow exp, \rho >$$

- Where ρ is the environment in effect when the function is defined (for a simple function)



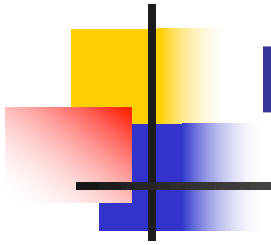
Evaluating declarations

- Evaluation uses an environment ρ
- To evaluate a (simple) declaration let x = e
  - Evaluate expression e in ρ to value v
  - Update ρ with x v: $\{x \rightarrow v\} + \rho$

- Update: $\rho_1 + \rho_2$ has all the bindings in ρ_1 and all those in ρ_2 that are not rebound in ρ_1

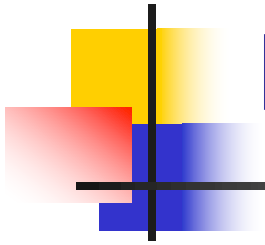
$\{x \rightarrow 2, y \rightarrow 3, a \rightarrow \text{“hi”}\} + \{y \rightarrow 100, b \rightarrow 6\}$

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



Evaluating expressions in OCaml

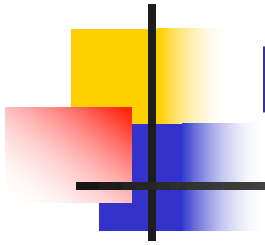
- Evaluation uses an environment ρ
- A constant evaluates to itself, including primitive operators like + and =
- To evaluate a variable, look it up in ρ: $\rho(v)$
- To evaluate a tuple $(e_1,\ldots,e_n)$,
  - Evaluate each e_i to v_i, right to left for Ocaml
  - Then make value $(v_1,\ldots,v_n)$



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

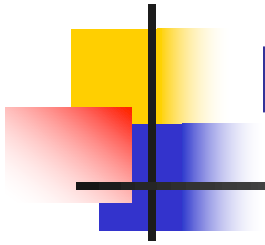
Evaluation of Application with Closures



- Given application expression f e
- In Ocaml, evaluate e to value v
- In environment ρ, evaluate left term to closure, $c = <(x_1,\ldots,x_n) \rightarrow b, \rho'>$
  - $(x_1,\ldots,x_n)$ variables in (first) argument
  - v must have form $(v_1,\ldots,v_n)$
- Update the environment ρ' to

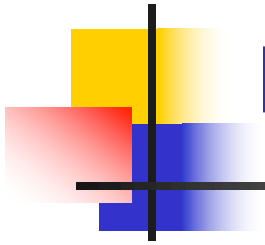
  $\rho'' = \{x_1 \rightarrow v_1,\ldots, x_n \rightarrow v_n\} + \rho'$
- Evaluate body b in environment ρ''

Extra Material for Extra Credit



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))



Evaluating expressions in OCaml

- To evaluate a tuple $(e_1,\ldots,e_n)$,
  - Evaluate each e_i to v_i, right to left for Ocaml
  - Then make value $(v_1,\ldots,v_n)$
  - Eval$((e_1,\ldots,e_n),\rho)$=> Eval$((e_1,\ldots,$Eval $(e_n, \rho)), \rho)$
  - Eval$((e_1,\ldots,e_i,$ Val $v_{i+1},\ldots,$ Val $v_n)$, $\rho)$ =>
    Eval$((e_1,\ldots,$Eval$(e_i, \rho),$ Val $v_{i+1},\ldots,$ Val $v_n)$, $\rho)$
  - Eval$((\text{Val } v_1,\ldots,\text{Val } v_n)$, $\rho)$ => Val $(v_1,\ldots,v_n)$

Evaluating expressions in OCaml

- To evaluate uses of +, - , etc, eval args, then do operation ⦿ (+, -, *, +., ….)
  - Eval(e_1⦿e_2, ρ) => Eval(e_1⦿Eval(e_2, ρ), ρ))
  - Eval(e_1⦿Val e_2, ρ)=>Eval(Eval(e_1, ρ)⦿Val v_2, ρ))
  - Eval(Val v_1⦿ Val v_2) => Val (v_1⦿ v_2)

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

Evaluating expressions in OCaml

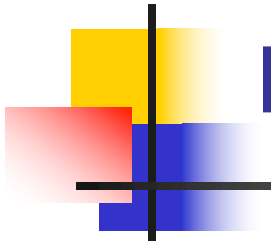
- To evaluate a local dec: let x = e1 in e2
  - Eval e1 to v, then eval e2 using $\{x \rightarrow v\} + \rho$
  - Eval(let x = e_1 in e_2, ρ) =>
    Eval(let x = Eval(e_1, ρ) in e_2, ρ)
  - Eval(let x = Val v in e_2, ρ) =>
    Eval(e_2, $\{x \rightarrow v\} + \rho$)

Evaluating expressions in OCaml

- To evaluate a conditional expression:
  if b then e_1 else e_2
  - Evaluate b to a value v
  - If v is True, evaluate e_1
  - If v is False, evaluate e_2

  - Eval(if b then e_1 else e_2, ρ) =>
    Eval(if Eval(b, ρ) then e_1 else e_2, ρ)
  - Eval(if Val true then e_1 else e_2, ρ) =>Eval(e_1, ρ)
  - Eval(if Val false then e_1 else e_2, ρ) =>Eval(e_2, ρ)

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, $c = <(x_1,\ldots,x_n) \rightarrow b, \rho'>$
  - $(x_1,\ldots,x_n)$ variables in (first) argument
  - v must have form $(v_1,\ldots,v_n)$
- Update the environment ρ' to

  $\rho'' = \{x_1 \rightarrow v_1,\ldots, x_n \rightarrow v_n\} + \rho'$
- Evaluate body b in environment 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 <(x_1,...,x_n) $\rightarrow$ b, ρ'>)(Val (v_1,...,v_n)), ρ)=>
  Eval(b, {$x_1 \rightarrow v_1$,..., $x_n \rightarrow v_n$}+ρ')

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval (plus_x z, ρ) =>
- Eval(plus_x (Eval(z, ρ))) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 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 $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 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 $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 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) =>
- Eval ((Val<y $\rightarrow$ y + x, ρ_{plus_x} >)(Val 3), ρ) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval ((Val<y $\rightarrow$ y + x, ρ_{plus_x} >)(Val 3), ρ) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval ((Val<y $\rightarrow$ y + x, ρ_{plus_x} >)(Val 3), ρ) =>
- Eval (y + x, {y $\rightarrow$ 3} +ρ_{plus_x}) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval ((Val<y $\rightarrow$ y + x, ρ_{plus_x} >)(Val 3), ρ) =>
- Eval (y + x, {y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(y+Eval(x, {y $\rightarrow$ 3} +ρ_{plus_x}),
  {y $\rightarrow$ 3} +ρ_{plus_x}) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval ((Val<y $\rightarrow$ y + x, ρ_{plus_x} >)(Val 3), ρ) =>
- Eval (y + x, {y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(y+Eval(x, {y $\rightarrow$ 3} +ρ_{plus_x}),
  {y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(y+Val 12,{y $\rightarrow$ 3} +ρ_{plus_x}) => …

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval(y+Eval(x, {y $\rightarrow$ 3} +ρ_{plus_x}),
  {y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(y+Val 12,{y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(Eval(y, {y $\rightarrow$ 3} +ρ_{plus_x}) +
  Val 12,{y $\rightarrow$ 3} +ρ_{plus_x}) =>…

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}

- Eval(Eval(y, {y $\rightarrow$ 3} +ρ_{plus_x}) +
  Val 12,{y $\rightarrow$ 3} +ρ_{plus_x}) =>

- Eval(Val 3 + Val 12 ,{y $\rightarrow$ 3} +ρ_{plus_x}) =>…

Evaluation of Application of plus_x;;

- Have environment:

  ρ = {plus_x $\rightarrow$ <y $\rightarrow$ y + x, ρ_{plus_x} >, … ,
  y $\rightarrow$ 19, x $\rightarrow$17, z $\rightarrow$3, …}

  where ρ_{plus_x} = {x $\rightarrow$ 12, … , y $\rightarrow$ 24, …}
- Eval(Eval(y, {y $\rightarrow$ 3} +ρ_{plus_x}) +
  Val 12,{y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Eval(Val 3 + Val 12 ,{y $\rightarrow$ 3} +ρ_{plus_x}) =>
- Val (3 + 12) = Val 15

Evaluation of Application of plus_pair

- Assume environment

ρ = {x $\rightarrow$ 3…,
plus_pair $\rightarrow$<(n,m) $\rightarrow$n + m, ρ_{plus_pair}>} + ρ_{plus_pair}

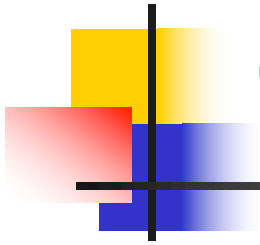
- Eval (plus_pair (4,x), ρ)=>
- Eval (plus_pair (Eval ((4, x), ρ)), ρ) =>
- Eval (plus_pair (Eval ((4, Eval (x , ρ)), ρ)), ρ) =>
- Eval (plus_pair (Eval ((4, Val 3), ρ)), ρ) =>
- Eval (plus_pair (Eval ((Eval (4, ρ), Val 3), ρ)), ρ) =>
- Eval (plus_pair (Eval ((Val 4, Val 3), ρ)), ρ) =>

Evaluation of Application of plus_pair

- Assume environment

ρ = {x $\rightarrow$ 3…,
plus_pair $\rightarrow$<(n,m) $\rightarrow$n+m, ρ_{plus_pair}>} + ρ_{plus_pair}

- Eval (plus_pair (Eval ((Val 4, Val 3), ρ)), ρ) =>
- Eval (plus_pair (Val (4, 3)), ρ) =>
- Eval (Eval (plus_pair, ρ), Val (4, 3)), ρ) => …
- Eval ((Val<(n,m)$\rightarrow$n+m, ρ_{plus_pair}>)(Val(4,3)) , ρ)=>
- Eval (n + m, {n -> 4, m -> 3} + ρ_{plus_pair}) =>
- Eval (4 + 3, {n -> 4, m -> 3} + ρ_{plus_pair}) => 7

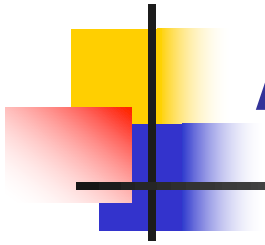


Closure question

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

```
let f = fun n -> n + 5;;
(* 0 *)
let pair_map g (n,m) = (g n, g m);;
let f = pair_map f;;
let a = f (4,6);;
```

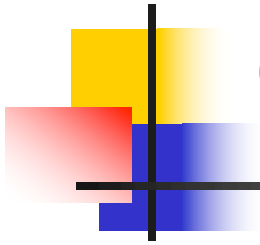
What is the environment at (* 0 *)?



Answer

```
let f = fun n -> n + 5;;
```

$\rho_0 = \{f \rightarrow \langle n \rightarrow n + 5, \{\ \}\rangle\}$

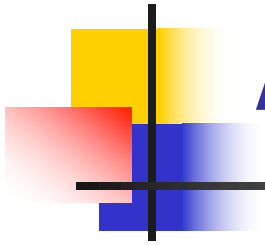


Closure question

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

```
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 *)?



Answer

$\rho_0 = \{f \rightarrow \langle n \rightarrow n + 5, \{\ \}\rangle\}$

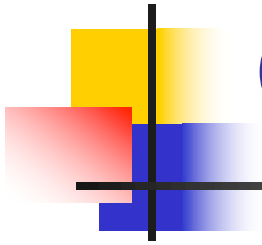
let pair_map g (n,m) = (g n, g m);;

$\rho_1 = \{$pair_map $\rightarrow$

$\langle g \rightarrow$ fun (n,m) -> (g n, g m),

$\{f \rightarrow \langle n \rightarrow n + 5, \{\ \}\rangle\}\rangle$,

$f \rightarrow \langle n \rightarrow n + 5, \{\ \}\rangle\}$

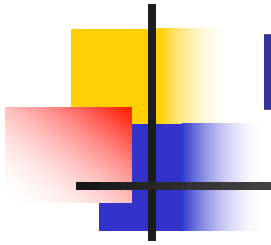


Closure question

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

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

What is the environment at (* 2 *)?

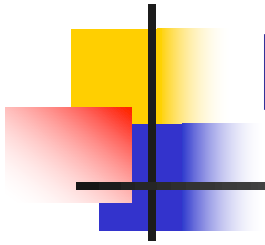


Evaluate pair_map f

ρ_0 = {f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

ρ_1 = {pair_map $\rightarrow$<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>,
f$\rightarrow$<n $\rightarrow$ n + 5, { }>}

let f = pair_map f;;

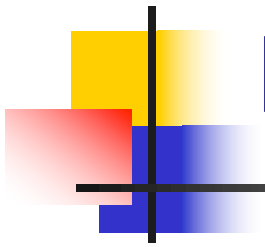


Evaluate pair_map f

ρ_0 = {f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

ρ_1 = {pair_map $\rightarrow$<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>,
f$\rightarrow$<n $\rightarrow$ n + 5, { }>}

Eval(pair_map f, ρ_1) =



Evaluate pair_map f

ρ_0 = {f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

ρ_1 = {pair_map $\rightarrow$<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>,
f$\rightarrow$<n $\rightarrow$ n + 5, { }>}

Eval(pair_map f, ρ_1) =>

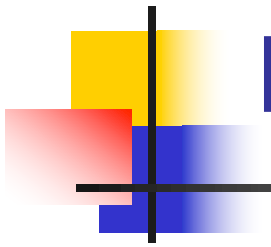
Eval(pair_map (Eval(f, ρ_1)), ρ_1) =>

Eval(pair_map (Val<n $\rightarrow$ n + 5, { }>), ρ_1) =>

Eval((Eval(pair_map, ρ_1))(Val<n $\rightarrow$ n+5, { }>), ρ_1) =>

Eval((Val (<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>)
(Val <n $\rightarrow$ n + 5, { }>), ρ_1) =>

Eval(fun (n,m)->(g n, g m), {g$\rightarrow$<n$\rightarrow$n + 5, { }>}+ρ_0)
=>



Evaluate pair_map f

ρ_0 = {f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

ρ_1 = {pair_map $\rightarrow$<g$\rightarrow$fun (n,m) -> (g n, g m), ρ_0>,
f$\rightarrow$<n $\rightarrow$ n + 5, { }>}

Eval(pair_map f, ρ_1) => ... =>

Eval(fun (n,m)->(g n, g m), {g$\rightarrow$<n$\rightarrow$n + 5, { }>}+ρ_0)
=

Eval(fun (n,m)->(g n, g m),
{g$\rightarrow$<n$\rightarrow$n + 5, { }>, f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}) =>

Val (<(n,m) $\rightarrow$(g n, g m),
{g$\rightarrow$<n$\rightarrow$n + 5, { }>, f $\rightarrow$ <n $\rightarrow$ n + 5, { }>})

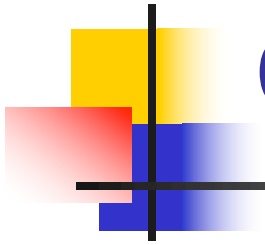
Answer

ρ_1 = {pair_map $\rightarrow$
<g $\rightarrow$ fun (n,m) -> (g n, g m),{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}

let f = pair_map f;;

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

(*Remember: the original f is now removed from ρ_2 *)



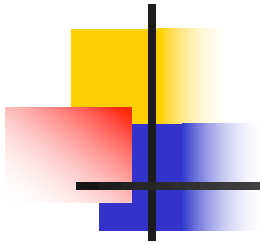
Closure question

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

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

(* 3 *)

What is the environment at (* 3 *)?



Final Evalution?

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

let a = f (4,6);;

Evaluate f (4,6);;

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),

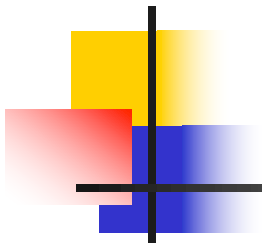
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,

f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,

pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),

{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

Eval(f (4,6), ρ_2) =



Evaluate f (4,6);;

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

Eval(f (4,6), ρ_2) => Eval(f (Eval((4,6), ρ_2)), ρ_2) =>

Eval(f (Eval((4,Eval(6, ρ_2)), ρ_2)), ρ_2) =>

Eval(f (Eval((4,Val 6), ρ_2)), ρ_2) =>

Eval(f (Eval((Eval(4, ρ_2),Val 6), ρ_2)), ρ_2) =>

Eval(f (Eval((Val 4,Val 6), ρ_2)), ρ_2) =>

Evaluate f (4,6);;

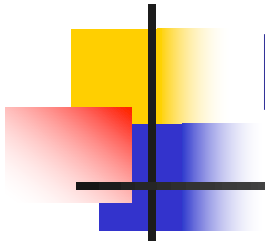
ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

Eval(f (4,6), ρ_2) => ... =>

Eval(f (Eval((Val 4,Val 6), ρ_2)), ρ_2) =>

Eval(f (Val (4, 6)), ρ_2) =>

Eval(Eval(f, ρ_2) (Val (4, 6)), ρ_2) =>



Evaluate f (4,6);;

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

Eval(f (4,6), ρ_2) => ... =>

Eval(Eval(f, ρ_2) (Val (4, 6)), ρ_2) =>

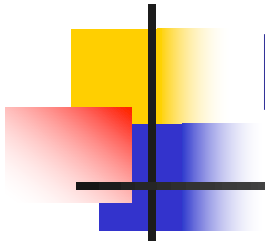
Eval((Val <(n,m)$\rightarrow$(g n, g m),
{g$\rightarrow$<n$\rightarrow$n+5, { }>,
f$\rightarrow$<n$\rightarrow$n+5, { }>}>)(Val(4,6)))),ρ_2)=>

Evaluate f (4,6);;

ρ_2 = {f $\rightarrow$ <(n,m) $\rightarrow$(g n, g m),
{g $\rightarrow$ <n $\rightarrow$ n + 5, { }>,
f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>,
pair_map $\rightarrow$ <g $\rightarrow$ fun (n,m) -> (g n, g m),
{f $\rightarrow$ <n $\rightarrow$ n + 5, { }>}>}

Eval((Val <(n,m)$\rightarrow$(g n, g m),
{g$\rightarrow$<n$\rightarrow$n+5, { }>,
f$\rightarrow$<n$\rightarrow$n+5, { }>}>)(Val(4,6)))),ρ_2)=>

Eval((g n, g m), {n $\rightarrow$ 4, m $\rightarrow$ 6, g$\rightarrow$<n$\rightarrow$n+5, { }>,
f$\rightarrow$<n$\rightarrow$n+5, { }>}) =>



Evaluate f (4,6);;

Let $\rho' = \{n \rightarrow 4, m \rightarrow 6, g \rightarrow <n \rightarrow n+5, \{\}>,$
$f \rightarrow <n \rightarrow n+5, \{\}>\})$

Eval((g n, g m), $\{n \rightarrow 4, m \rightarrow 6, g \rightarrow <n \rightarrow n+5, \{\}>,$
$f \rightarrow <n \rightarrow n+5, \{\}>\}) =$

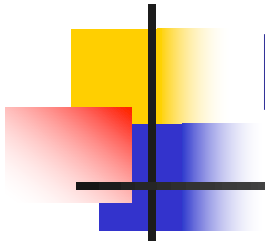
Eval((g n, g m), ρ') =>

Eval((g n, Eval(g m, ρ')), ρ') =>

Eval((g n, Eval(g (Eval (m, ρ')), ρ')), ρ') =>

Eval((g n, Eval(g (Val 6), ρ')), ρ') =>

Eval((g n, Eval((Eval(g, ρ'))(Val 6), ρ')), ρ') =>



Evaluate f (4,6);;

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

Eval((g n, Eval((Eval(g, ρ'))(Val 6), ρ')), ρ') =>

Eval((g n, Eval((Val<n→n+5,{ }>)(Val 6), ρ')), ρ') =>

Eval((g n, Eval(n+5, {n→6}+{ })), ρ') =

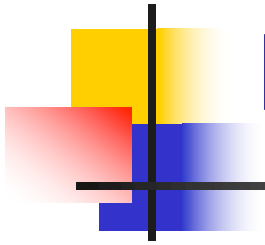
Eval((g n, Eval(n+5, {n→6})), ρ') =>

Eval((g n, Eval(n+(Eval(5, {n→6})), {n→6})), ρ') =>

Eval((g n, Eval(n+(Val 5), {n→6})), ρ') =>

Eval((g n, Eval((Eval(n,{n→6}))+(Val 5),{n→6}), ρ')=>

Eval((g n, Eval((Val 6)+(Val 5),{n→6}), ρ')=>



Evaluate f (4,6);;

Let $\rho' = \{n \rightarrow 4, m \rightarrow 6, g \rightarrow <n \rightarrow n+5, \{\}>,$
$f \rightarrow <n \rightarrow n+5, \{\}>\})$

Eval((g n, Eval((Val 6)+(Val 5),{n→6}), ρ') =>

Eval((g n, Val 11), ρ') =>

Eval((Eval(g n, ρ'), Val 11), ρ') =>

Eval((Eval(g (Eval(n, ρ')), ρ'), Val 11), ρ') =>

Eval((Eval(g (Val 4), ρ'), Val 11), ρ') =>

Eval((Eval(Eval(g, ρ')(Val 4), ρ'), Val 11), ρ') =>

Eval((Eval((Val<n→n+5, { }>)(Val 4), ρ'), Val 11), ρ') =>

Evaluate f (4,6);;

Let ρ' = {n $\rightarrow$ 4, m $\rightarrow$ 6, g$\rightarrow$<n$\rightarrow$n+5, { }>,
f$\rightarrow$<n$\rightarrow$n+5, { }>})

Eval((Eval((Val<n$\rightarrow$n+5, { }>)(Val 4), ρ'), Val 11), ρ')
=>

Eval((Eval(n+5, {n $\rightarrow$ 4}+{})), Val 11), ρ') =

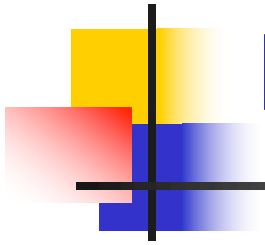
Eval((Eval(n+5, {n $\rightarrow$ 4})), Val 11), ρ') =>

Eval((Eval(n+Eval(5,{n $\rightarrow$ 4}),{n $\rightarrow$ 4}), Val 11),ρ') =>

Eval((Eval(n+(Val 5),{n $\rightarrow$ 4}), Val 11),ρ') =>

Eval((Eval(Eval(n,{n $\rightarrow$ 4})+(Val 5),{n $\rightarrow$ 4}),
Val 11),ρ') =>

End of
Extra Material for Extra Credit



Recursive Functions

```
# let rec factorial n =
    if n = 0 then 1 else n * factorial (n - 1);;
  val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
# (* rec  is needed for recursive function
   declarations *)
```

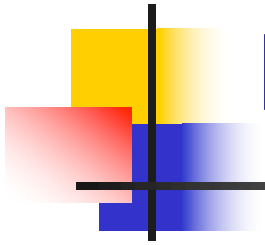
Recursion Example

Compute n^2 recursively using:

$$n^2 = (2 * n - 1) + (n - 1)^2$$

```
# let rec nthsq n =          (* rec for recursion *)
   match n                 (* pattern matching for cases *)
   with 0 -> 0                  (* base case *)
   | n -> (2 * n -1)            (* recursive case *)
          + nthsq (n -1);;   (* recursive call *)
val nthsq : int -> int = <fun>
# nthsq 3;;
- : int = 9
```

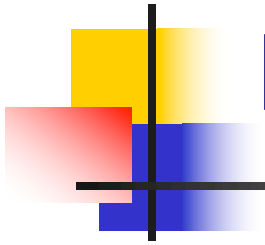
Structure of recursion similar to inductive proof



Recursion and Induction

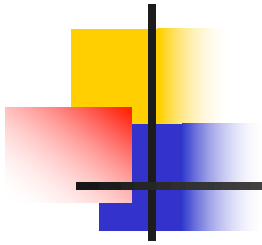
```
# let rec nthsq n = match n with 0 -> 0
      | n -> (2 * n - 1) + nthsq (n - 1) ;;
```

- Base case is the last case; it stops the computation
- Recursive call must be to arguments that are somehow smaller - must progress to base case
- **if** or **match** must contain base case
- Failure of these may cause failure of termination



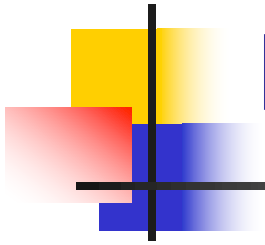
Lists

- 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; …; xn] == x1 :: x2 :: … :: xn :: [ ]



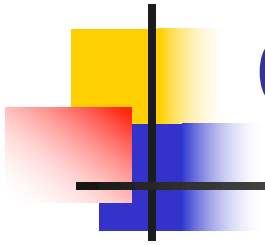
Lists

```
# 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; 1]
# (8::5::3::2::1::1::[ ]) = fib5;;
- : bool = true
# fib5 @ fib6;;
- : int list = [8; 5; 3; 2; 1; 1; 13; 8; 5; 3; 2; 1;
   1]
```



Lists are Homogeneous

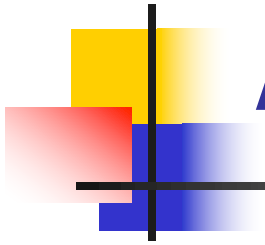
```
# let bad_list = [1; 3.2; 7];;
Characters 19-22:
  let bad_list = [1; 3.2; 7];;
                     ^^^
This expression has type float but is here
    used with type int
```



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”]]

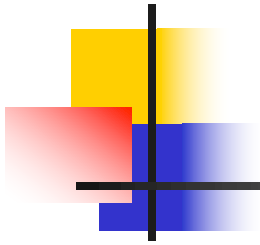


Answer

- Which one of these lists is invalid?

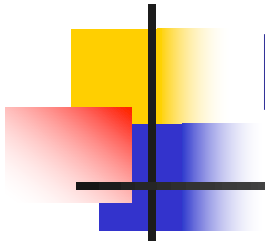
1. [2; 3; 4; 6]
2. [2,3; 4,5; 6,7]
3. [(2.3,4); (3.2,5); (6,7.2)]
4. [[“hi”; “there”]; [“wahcha”]; [ ]; [“doin”]]

- 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"]
```