

# Programming Languages and Compilers (CS 421)

Elsa L Gunter  
2112 SC, UIUC



<https://courses.engr.illinois.edu/cs421/fa2023/CS421D>

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



## Functions with more than one argument

---

```
# let add_three x y z = x + y + z;;
```

```
val add_three : int -> int -> int -> int = <fun>
```

```
# let t = add_three 6 3 2;;
```

```
val t : int = 11
```

```
# let add_three =
```

```
  fun x -> (fun y -> (fun z -> x + y + z));;
```

```
val add_three : int -> int -> int -> int = <fun>
```

Again, first syntactic sugar for second



# Functions with more than one argument

---

```
# let add_three x y z = x + y + z;;
```

```
val add_three : int -> int -> int -> int = <fun>
```

- What is the value of add\_three?
- Let  $\rho_{\text{add\_three}}$  be the environment before the declaration
- Remember:

```
let add_three =
```

```
  fun x -> (fun y -> (fun z -> x + y + z));;
```

```
Value: <x ->fun y -> (fun z -> x + y + z),  $\rho_{\text{add\_three}}$  >
```



# Partial application of functions

---

```
let add_three x y z = x + y + z;;
```

```
# let h = add_three 5 4;;
```

```
val h : int -> int = <fun>
```

```
# h 3;;
```

```
- : int = 12
```

```
# h 7;;
```

```
- : int = 16
```



# Partial application of functions

---

```
let add_three x y z = x + y + z;;
```

```
# let h = add_three 5 4;;
```

```
val h : int -> int = <fun>
```

```
# h 3;;
```

```
- : int = 12
```

```
# h 7;;
```

```
- : int = 16
```

```
- Partial application also called sectioning
```



# Functions as arguments

---

```
# let thrice f x = f (f (f x));;
```

```
val thrice : ('a -> 'a) -> 'a -> 'a = <fun>
```

```
# let g = thrice plus_two;;
```

```
val g : int -> int = <fun>
```

```
# g 4;;
```

```
- : int = 10
```

```
# thrice (fun s -> "Hi! " ^ s) "Good-bye!";;
```

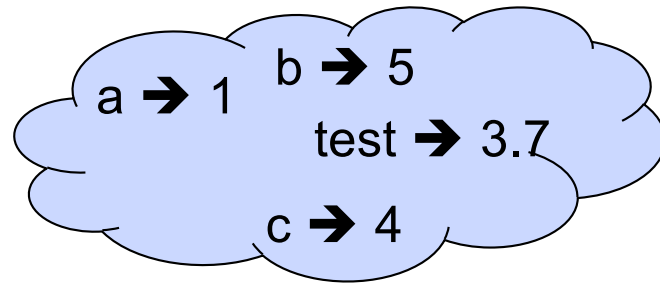
```
- : string = "Hi! Hi! Hi! Good-bye!"
```

# Tuples as Values

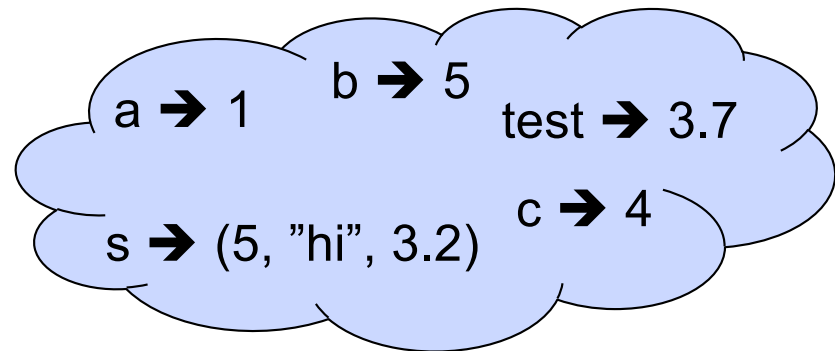
```
// ρ7 = {c → 4, test → 3.7,  
          a → 1, b → 5}
```

```
# let s = (5, "hi", 3.2);;
```

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

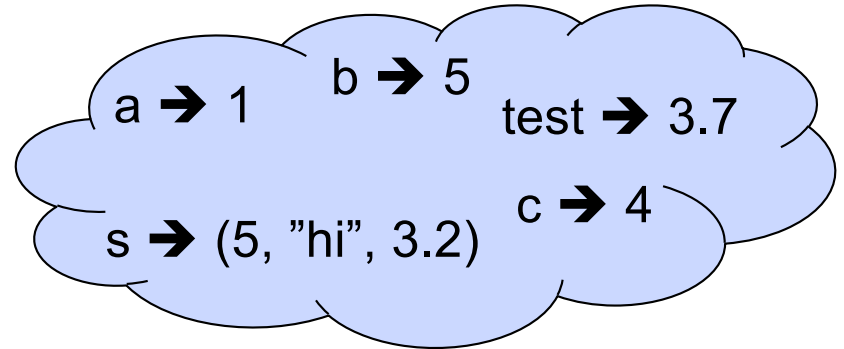


```
// ρ8 = {s → (5, "hi", 3.2),  
          c → 4, test → 3.7,  
          a → 1, b → 5}
```



# 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 *)
```

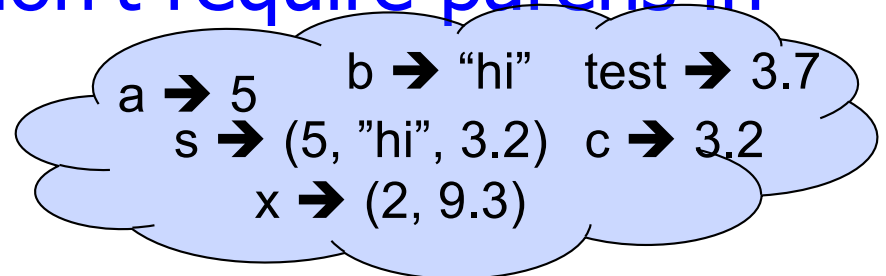
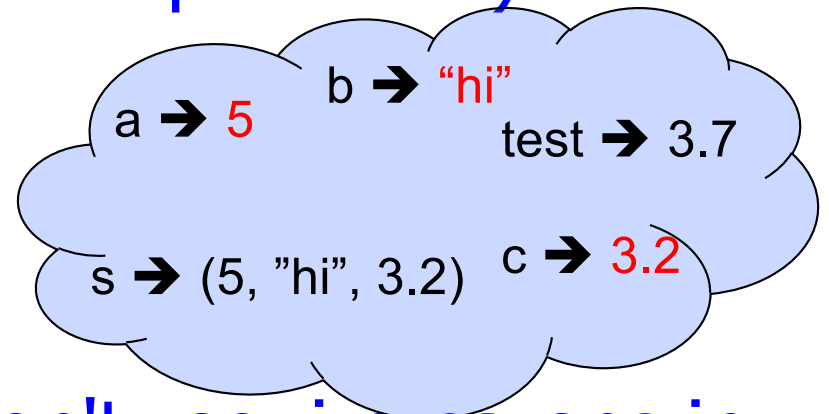
```
val a : int = 5
```

```
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")
```



# Curried vs Uncurried

---

- Recall

```
val add_three : int -> int -> int -> int = <fun>
```

- How does it differ from

```
# let add_triple (u,v,w) = u + v + w;;
```

```
val add_triple : int * int * int -> int = <fun>
```

- add\_three is *curried*;
- add\_triple is *uncurried*



# Curried vs Uncurried

---

```
# add_triple (6,3,2);;
```

```
- : int = 11
```

```
# add_triple 5 4;;
```

```
Characters 0-10:
```

```
add_triple 5 4;;
```

```
^^^  
^^^  
^^^
```

This function is applied to too many arguments,  
maybe you forgot a `;`

```
# fun x -> add_triple (5,4,x);;
```

```
: int -> int = <fun>
```



# 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



# Save the Environment!

---

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

$$\langle (v_1, \dots, v_n) \rightarrow \underline{\text{exp}}, \rho \rangle$$

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



# Closure for plus\_pair

---

- Assume  $\rho_{\text{plus\_pair}}$  was the environment just before `plus_pair` defined

- Closure for `fun (n,m) -> n + m`:

$$\langle (n,m) \rightarrow n + m, \rho_{\text{plus\_pair}} \rangle$$

- Environment just after `plus_pair` defined:

$$\{\text{plus\_pair} \rightarrow \langle (n,m) \rightarrow n + m, \rho_{\text{plus\_pair}} \rangle\}$$

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



# 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  $x \rightarrow v$ :  $\{x \rightarrow v\} + \rho$





# Evaluating declarations

---

- Evaluation uses an environment  $\rho$
- To evaluate a (simple) declaration  $\text{let } x = e$ 
  - 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 =



# 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  $\rho$ :  $\rho(v)$



# 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  $(e_1, \dots, e_n)$ ,
  - Evaluate each  $e_i$  to  $v_i$ , right to left for OCaml
  - Then make value  $(v_1, \dots, v_n)$



# Evaluating expressions in OCaml

---

- To evaluate uses of  $+$ ,  $-$ , etc, eval args, then do operation



# Evaluating expressions in OCaml

---

- To evaluate uses of  $+$ ,  $-$ , etc, eval args, then do operation
- Function expression evaluates to its closure



# 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:  $\text{let } x = e1 \text{ in } e2$ 
  - Eval  $e1$  to  $v$ , then eval  $e2$  using  $\{x \rightarrow v\} + \rho$



# Evaluating expressions in OCaml

---

- To evaluate uses of  $+$ ,  $-$ , etc, eval args (right to left for Ocaml), 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`

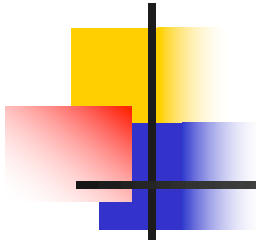




# Evaluation of Application with Closures

---

- Given application expression  $f e$
- In Ocaml, evaluate  $e$  to value  $v$
- In environment  $\rho$ , evaluate left term to closure,  
 $c = \langle (x_1, \dots, x_n) \rightarrow b, \rho' \rangle$ 
  - $(x_1, \dots, x_n)$  variables in (first) argument
  - $v$  must have form  $(v_1, \dots, v_n)$
- Update the environment  $\rho'$  to  
 $\rho'' = \{x_1 \rightarrow v_1, \dots, x_n \rightarrow v_n\} + \rho'$
- Evaluate body  $b$  in environment  $\rho''$



# Extra Material for Extra Credit



# Evaluating expressions in OCaml

---

- Evaluation uses an environment  $\rho$ 
  - $\text{Eval}(e, \rho)$
- A constant evaluates to itself, including primitive operators like  $+$  and  $=$ 
  - $\text{Eval}(c, \rho) \Rightarrow \text{Val } c$
- To evaluate a variable  $v$ , look it up in  $\rho$ :
  - $\text{Eval}(v, \rho) \Rightarrow \text{Val}(\rho(v))$



# Evaluating expressions in OCaml

---

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



# Evaluating expressions in OCaml

---

- To evaluate uses of  $+$ ,  $-$ , etc, eval args, then do operation  $\odot$  ( $+$ ,  $-$ ,  $*$ ,  $+$ ., ....)
  - $\text{Eval}(e_1 \odot e_2, \rho) \Rightarrow \text{Eval}(e_1 \odot \text{Eval}(e_2, \rho), \rho)$
  - $\text{Eval}(e_1 \odot \text{Val } v_2, \rho) \Rightarrow \text{Eval}(\text{Eval}(e_1, \rho) \odot \text{Val } v_2, \rho)$
  - $\text{Eval}(\text{Val } v_1 \odot \text{Val } v_2) \Rightarrow \text{Val } (v_1 \odot v_2)$
- Function expression evaluates to its closure
  - $\text{Eval}(\text{fun } x \rightarrow e, \rho) \Rightarrow \text{Val } \langle x \rightarrow e, \rho \rangle$



# Evaluating expressions in OCaml

---

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



# 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  $e_2$
- $\text{Eval}(\text{if } b \text{ then } e_1 \text{ else } e_2, \rho) \Rightarrow$   
 $\text{Eval}(\text{if } \text{Eval}(b, \rho) \text{ then } e_1 \text{ else } e_2, \rho)$
- $\text{Eval}(\text{if Val true then } e_1 \text{ else } e_2, \rho) \Rightarrow \text{Eval}(e_1, \rho)$
- $\text{Eval}(\text{if Val false then } e_1 \text{ else } e_2, \rho) \Rightarrow \text{Eval}(e_2, \rho)$



# Evaluation of Application with Closures

---

- Given application expression  $f e$
- In Ocaml, evaluate  $e$  to value  $v$
- In environment  $\rho$ , evaluate left term to closure,  
 $c = \langle (x_1, \dots, x_n) \rightarrow b, \rho' \rangle$ 
  - $(x_1, \dots, x_n)$  variables in (first) argument
  - $v$  must have form  $(v_1, \dots, v_n)$
- Update the environment  $\rho'$  to  
 $\rho'' = \{x_1 \rightarrow v_1, \dots, x_n \rightarrow v_n\} + \rho'$
- Evaluate body  $b$  in environment  $\rho''$





# Evaluation of Application with Closures

---

- $\text{Eval}(f\ e, \rho) \Rightarrow \text{Eval}(f\ (\text{Eval}(e, \rho)), \rho)$
- $\text{Eval}(f\ (\text{Val } v), \rho) \Rightarrow \text{Eval}((\text{Eval}(f, \rho))\ (\text{Val } v), \rho)$
- $\text{Eval}((\text{Val } \langle (x_1, \dots, x_n) \rightarrow b, \rho' \rangle)\ (\text{Val } (v_1, \dots, v_n)), \rho) \Rightarrow$   
 $\text{Eval}(b, \{x_1 \rightarrow v_1, \dots, x_n \rightarrow v_n\} + \rho')$



# Evaluation of Application of plus\_x;;

---

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}(\text{plus\_x } z, \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x } (\text{Eval}(z, \rho))) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}(\text{plus\_x } z, \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x } (\text{Eval}(z, \rho)), \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x } (\text{Val } 3), \rho) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}(\text{plus\_x } z, \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x}(\text{Eval}(z, \rho)), \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x}(\text{Val } 3), \rho) \Rightarrow$
- $\text{Eval}((\text{Eval}(\text{plus\_x}, \rho))(\text{Val } 3), \rho) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}(\text{plus\_x } z, \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x}(\text{Eval}(z, \rho)), \rho) \Rightarrow$
- $\text{Eval}(\text{plus\_x}(\text{Val } 3), \rho) \Rightarrow$
- $\text{Eval}((\text{Eval}(\text{plus\_x}, \rho))(\text{Val } 3), \rho) \Rightarrow$
- $\text{Eval}((\text{Val} \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle)(\text{Val } 3), \rho) \Rightarrow \dots$



# Evaluation of Application of plus\_x;;

---

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- Eval  $((\text{Val} \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle)(\text{Val } 3), \rho)$   
 $\Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- Eval  $((\text{Val} \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle)(\text{Val } 3), \rho) \Rightarrow$
- Eval  $(y + x, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}((\text{Val}\langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle)(\text{Val } 3), \rho) \Rightarrow$
- $\text{Eval}(y + x, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval}(y + \text{Eval}(x, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}), \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow \dots$



# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{ \text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots \}$$

where  $\rho_{\text{plus\_x}} = \{ x \rightarrow 12, \dots, y \rightarrow 24, \dots \}$

- $\text{Eval} ((\text{Val} \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle) (\text{Val } 3), \rho) \Rightarrow$
- $\text{Eval} (y + x, \{ y \rightarrow 3 \} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval} (y + \text{Eval} (x, \{ y \rightarrow 3 \} + \rho_{\text{plus\_x}}), \{ y \rightarrow 3 \} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval} (y + \text{Val } 12, \{ y \rightarrow 3 \} + \rho_{\text{plus\_x}}) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{\text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots\}$$

where  $\rho_{\text{plus\_x}} = \{x \rightarrow 12, \dots, y \rightarrow 24, \dots\}$

- $\text{Eval}(y + \text{Eval}(x, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}), \\ \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval}(y + \text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval}(\text{Eval}(y, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) + \\ \text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{ \text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots \}$$

where  $\rho_{\text{plus\_x}} = \{ x \rightarrow 12, \dots, y \rightarrow 24, \dots \}$

- $\text{Eval}(\text{Eval}(y, \{y \rightarrow 3\}) + \rho_{\text{plus\_x}}) +$   
 $\text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval}(\text{Val } 3 + \text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow \dots$

# Evaluation of Application of plus\_x;;

- Have environment:

$$\rho = \{ \text{plus\_x} \rightarrow \langle y \rightarrow y + x, \rho_{\text{plus\_x}} \rangle, \dots, \\ y \rightarrow 19, x \rightarrow 17, z \rightarrow 3, \dots \}$$

where  $\rho_{\text{plus\_x}} = \{ x \rightarrow 12, \dots, y \rightarrow 24, \dots \}$

- $\text{Eval}(\text{Eval}(y, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) + \text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Eval}(\text{Val } 3 + \text{Val } 12, \{y \rightarrow 3\} + \rho_{\text{plus\_x}}) \Rightarrow$
- $\text{Val } (3 + 12) = \text{Val } 15$



# Evaluation of Application of `plus_pair`

- Assume environment

$\rho = \{x \rightarrow 3, \dots,$

$\text{plus\_pair} \rightarrow \langle (n, m) \rightarrow n + m, \rho_{\text{plus\_pair}} \rangle \} + \rho_{\text{plus\_pair}}$

- $\text{Eval} (\text{plus\_pair} (4, x), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((4, x), \rho)), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((4, \text{Eval} (x, \rho)), \rho)), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((4, \text{Val } 3), \rho)), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((\text{Eval} (4, \rho), \text{Val } 3), \rho)), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((\text{Val } 4, \text{Val } 3), \rho)), \rho) \Rightarrow$

# Evaluation of Application of `plus_pair`

- Assume environment

$\rho = \{x \rightarrow 3, \dots,$

$\text{plus\_pair} \rightarrow \langle (n, m) \rightarrow n+m, \rho_{\text{plus\_pair}} \rangle \} + \rho_{\text{plus\_pair}}$

- $\text{Eval} (\text{plus\_pair} (\text{Eval} ((\text{Val } 4, \text{Val } 3)), \rho)), \rho) \Rightarrow$
- $\text{Eval} (\text{plus\_pair} (\text{Val } (4, 3)), \rho) \Rightarrow$
- $\text{Eval} (\text{Eval} (\text{plus\_pair}, \rho), \text{Val } (4, 3)), \rho) \Rightarrow \dots$
- $\text{Eval} ((\text{Val} \langle (n, m) \rightarrow n+m, \rho_{\text{plus\_pair}} \rangle) (\text{Val} (4, 3)), \rho) \Rightarrow$
- $\text{Eval} (n + m, \{n \rightarrow 4, m \rightarrow 3\} + \rho_{\text{plus\_pair}}) \Rightarrow$
- $\text{Eval} (4 + 3, \{n \rightarrow 4, m \rightarrow 3\} + \rho_{\text{plus\_pair}}) \Rightarrow 7$



# 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);;`

$\rho_1 = \{\text{pair\_map} \rightarrow$   
     $\langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$   
     $\{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\} \rangle,$   
     $f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$



# 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

---

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

$\rho_1 = \{\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m), \rho_0 \rangle,$   
 $f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$

let f = pair\_map f;;



# Evaluate pair\_map f

---

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

$$\rho_1 = \{\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n, m) \rightarrow (g \ n, g \ m), \rho_0 \rangle, \\ f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$$

$$\text{Eval}(\text{pair\_map } f, \rho_1) =$$



# Evaluate pair\_map f

---

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

$\rho_1 = \{\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g \ n, g \ m), \rho_0 \rangle,$   
 $f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$

$\text{Eval}(\text{pair\_map } f, \rho_1) \Rightarrow$

$\text{Eval}(\text{pair\_map } (\text{Eval}(f, \rho_1)), \rho_1) \Rightarrow$

$\text{Eval}(\text{pair\_map } (\text{Val} \langle n \rightarrow n + 5, \{ \} \rangle), \rho_1) \Rightarrow$

$\text{Eval}((\text{Eval}(\text{pair\_map}, \rho_1))(\text{Val} \langle n \rightarrow n + 5, \{ \} \rangle), \rho_1) \Rightarrow$

$\text{Eval}((\text{Val } (\langle g \rightarrow \text{fun } (n,m) \rightarrow (g \ n, g \ m), \rho_0 \rangle)$

$(\text{Val } \langle n \rightarrow n + 5, \{ \} \rangle), \rho_1) \Rightarrow$

$\text{Eval}(\text{fun } (n,m) \rightarrow (g \ n, g \ m), \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\} + \rho_0)$   
 $\Rightarrow$



# Evaluate pair\_map f

$$\rho_0 = \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$$
$$\rho_1 = \{\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n, m) \rightarrow (g \ n, g \ m), \rho_0 \rangle, \\ f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}$$
$$\text{Eval}(\text{pair\_map } f, \rho_1) \Rightarrow \dots \Rightarrow$$
$$\text{Eval}(\text{fun } (n, m) \rightarrow (g \ n, g \ m), \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\} + \rho_0) \\ =$$
$$\text{Eval}(\text{fun } (n, m) \rightarrow (g \ n, g \ m),$$
$$\{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle, f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\}) \Rightarrow$$
$$\text{Val } (\langle (n, m) \rightarrow (g \ n, g \ m),$$
$$\{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle, f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle\})$$

# Answer

$\rho_1 = \{\text{pair\_map} \rightarrow$   
 $\langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m), \{f \rightarrow \langle n \rightarrow n + 5, \{\}\rangle\}\rangle,$   
 $f \rightarrow \langle n \rightarrow n + 5, \{\}\rangle\}$

**let f = pair\_map f;;**

$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$   
 $\{g \rightarrow \langle n \rightarrow n + 5, \{\}\rangle,$   
 $f \rightarrow \langle n \rightarrow n + 5, \{\}\rangle\}\rangle,$   
 $\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$   
 $\{f \rightarrow \langle n \rightarrow n + 5, \{\}\rangle\}\rangle\}$

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





# 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 Evaluation?

---

$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$   
     $\{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$   
     $f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle\},$   
     $\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$   
     $\{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \rangle\}$

let a = f (4,6);;



# Evaluate $f(4,6)$ ;;

---

$$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle\},$$
$$\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \rangle\}$$

$\text{Eval}(f(4,6), \rho_2) =$



# Evaluate $f(4,6)$ ;;

$$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle,$$
$$\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \}$$
$$\text{Eval}(f(4,6), \rho_2) \Rightarrow \text{Eval}(f(\text{Eval}((4,6), \rho_2)), \rho_2) \Rightarrow$$
$$\text{Eval}(f(\text{Eval}(4, \text{Eval}(6, \rho_2)), \rho_2), \rho_2) \Rightarrow$$
$$\text{Eval}(f(\text{Eval}(4, \text{Val } 6), \rho_2), \rho_2) \Rightarrow$$
$$\text{Eval}(f(\text{Eval}(\text{Eval}(4, \rho_2), \text{Val } 6), \rho_2), \rho_2) \Rightarrow$$
$$\text{Eval}(f(\text{Eval}(\text{Val } 4, \text{Val } 6), \rho_2), \rho_2) \Rightarrow$$



# Evaluate $f(4,6)$ ;;

---

$$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle,$$
$$\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \}$$

$\text{Eval}(f(4,6), \rho_2) \Rightarrow \dots \Rightarrow$

$\text{Eval}(f(\text{Eval}(\text{Val } 4, \text{Val } 6), \rho_2), \rho_2) \Rightarrow$

$\text{Eval}(f(\text{Val } (4, 6)), \rho_2) \Rightarrow$

$\text{Eval}(\text{Eval}(f, \rho_2)(\text{Val } (4, 6)), \rho_2) \Rightarrow$



# Evaluate $f(4,6)$ ;;

---

$$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle\},$$
$$\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \rangle\}$$
$$\text{Eval}(f(4,6), \rho_2) \Rightarrow \dots \Rightarrow$$
$$\text{Eval}(\text{Eval}(f, \rho_2) (\text{Val } (4, 6)), \rho_2) \Rightarrow$$
$$\text{Eval}((\text{Val } \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle) (\text{Val}(4,6)) ), \rho_2) \Rightarrow$$



# Evaluate $f(4,6)$ ;;

$$\rho_2 = \{f \rightarrow \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle\},$$
$$\text{pair\_map} \rightarrow \langle g \rightarrow \text{fun } (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle \rangle\}$$
$$\text{Eval}((\text{Val } \langle (n,m) \rightarrow (g\ n, g\ m),$$
$$\quad \{g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \rangle)(\text{Val}(4,6)) \ ), \rho_2) \Rightarrow$$
$$\text{Eval}((g\ n, g\ m), \{n \rightarrow 4, m \rightarrow 6, g \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle,$$
$$\quad f \rightarrow \langle n \rightarrow n + 5, \{ \} \rangle \}) \Rightarrow$$



# Evaluate $f(4,6)$ ;;

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

$\text{Eval}((g\ n, g\ m), \{n \rightarrow 4, m \rightarrow 6, g \rightarrow \langle n \rightarrow n+5, \{ \} \rangle,$   
 $f \rightarrow \langle n \rightarrow n+5, \{ \} \rangle\}) =$

$\text{Eval}(g\ n, g\ m), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(g\ m, \rho')), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(g\ (\text{Eval}(m, \rho')), \rho')), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(g\ (\text{Val } 6), \rho')), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}((\text{Eval}(g, \rho'))(\text{Val } 6), \rho')), \rho') \Rightarrow$





# Evaluate $f(4,6)$ ;;

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

$\text{Eval}(g\ n, \text{Eval}(\text{Eval}(g, \rho')(\text{Val } 6), \rho'), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(\text{Val} \langle n \rightarrow n+5, \{ \} \rangle (\text{Val } 6), \rho'), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(n+5, \{n \rightarrow 6\} + \{ \})), \rho' =$

$\text{Eval}(g\ n, \text{Eval}(n+5, \{n \rightarrow 6\})), \rho' \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(n + (\text{Eval}(5, \{n \rightarrow 6\})), \{n \rightarrow 6\})), \rho' \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(n + (\text{Val } 5), \{n \rightarrow 6\})), \rho' \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(\text{Eval}(n, \{n \rightarrow 6\}) + (\text{Val } 5), \{n \rightarrow 6\}), \rho') \Rightarrow$

$\text{Eval}(g\ n, \text{Eval}(\text{Val } 6 + (\text{Val } 5), \{n \rightarrow 6\}), \rho') \Rightarrow$



# Evaluate $f(4,6)$ ;;

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

$\text{Eval}(g\ n, \text{Eval}(\text{Val } 6) + (\text{Val } 5), \{n \rightarrow 6\}), \rho' \Rightarrow$

$\text{Eval}(g\ n, \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(g\ n, \rho'), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(g\ (\text{Eval}(n, \rho')), \rho'), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(g\ (\text{Val } 4), \rho'), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(\text{Eval}(g, \rho')(\text{Val } 4), \rho'), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(\text{Val} \langle n \rightarrow n+5, \{ \} \rangle (\text{Val } 4), \rho'), \text{Val } 11), \rho'$   
 $\Rightarrow$



# Evaluate $f(4,6)$ ;;

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

$\text{Eval}(\text{Eval}(\text{Val} \langle n \rightarrow n+5, \{ \} \rangle (\text{Val } 4), \rho'), \text{Val } 11), \rho')$   
 $\Rightarrow$

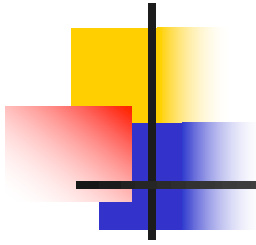
$\text{Eval}(\text{Eval}(n+5, \{n \rightarrow 4\} + \{\}), \text{Val } 11), \rho' =$

$\text{Eval}(\text{Eval}(n+5, \{n \rightarrow 4\}), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(n + \text{Eval}(5, \{n \rightarrow 4\}), \{n \rightarrow 4\}), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(n + (\text{Val } 5), \{n \rightarrow 4\}), \text{Val } 11), \rho' \Rightarrow$

$\text{Eval}(\text{Eval}(\text{Eval}(n, \{n \rightarrow 4\}) + (\text{Val } 5), \{n \rightarrow 4\}),$   
 $\text{Val } 11), \rho' \Rightarrow$



# 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"]
```



# Structural Recursion

---

- Functions on recursive datatypes (eg lists) tend to be recursive
- Recursion over recursive datatypes generally by structural recursion
  - Recursive calls made to components of structure of the same recursive type
  - Base cases of recursive types stop the recursion of the function



# Question: Length of list

---

- Problem: write code for the length of the list
  - How to start?

let rec length list =





# Question: Length of list

---

- Problem: write code for the length of the list
  - How to start?

let rec length list =  
 match list with



# Question: Length of list

---

- Problem: write code for the length of the list
  - What patterns should we match against?

let rec length list =  
 match list with



# Question: Length of list

---

- Problem: write code for the length of the list
  - What patterns should we match against?

```
let rec length list =  
  match list with [] ->  
  | (a :: bs) ->
```



# Question: Length of list

---

- Problem: write code for the length of the list
  - What result do we give when `list` is empty?

```
let rec length list =  
  match list with [] -> 0  
  | (a :: bs) ->
```



# Question: Length of list

---

- Problem: write code for the length of the list
  - What result do we give when `list` is not empty?

```
let rec length list =  
  match list with [] -> 0  
  | (a :: bs) ->
```



## Question: Length of list

---

- Problem: write code for the length of the list
  - What result do we give when `list` is not empty?

let rec length list =

match list with [] -> 0

| (a :: bs) -> 1 + length bs

# Structural Recursion : List Example

```
# let rec length list = match list
  with [ ] -> 0   (* Nil case *)
       | a :: bs -> 1 + length bs;; (* Cons case *)
val length : 'a list -> int = <fun>
# length [5; 4; 3; 2];;
- : int = 4
```

- Nil case [ ] is base case
- Cons case recurses on component list **bs**



# Same Length

---

- How can we efficiently answer if two lists have the same length?





# Same Length

---

- How can we efficiently answer if two lists have the same length?

```
let rec same_length list1 list2 =  
  match list1 with [] ->  
    (match list2 with [] -> true  
     | (y::ys) -> false)  
  | (x::xs) ->  
    (match list2 with [] -> false  
     | (y::ys) -> same_length xs ys)
```



Your turn: `doubleList : int list -> int list`

---

- Write a function that takes a list of int and returns a list of the same length, where each element has been multiplied by 2

`let rec doubleList list =`



Your turn: `doubleList : int list -> int list`

---

- Write a function that takes a list of `int` and returns a list of the same length, where each element has been multiplied by 2

```
let rec doubleList list =
```

```
  match list
```

```
    with [] -> []
```

```
      | x :: xs -> (2 * x) :: doubleList xs
```



Your turn: `doubleList : int list -> int list`

---

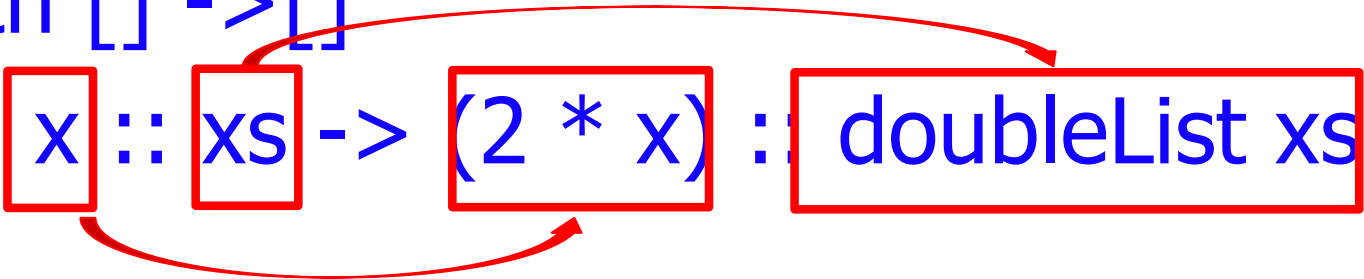
- Write a function that takes a list of `int` and returns a list of the same length, where each element has been multiplied by 2

`let rec doubleList list =`

`match list`

`with [] -> []`

`| x :: xs -> (2 * x) :: doubleList xs`





# Higher-Order Functions Over Lists

---

```
# let rec map f list =  
  match list  
  with [] -> []  
       | (h::t) -> (f h) :: (map f t);;  
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>  
# map plus_two fib5;;  
- : int list = [10; 7; 5; 4; 3; 3]  
# map (fun x -> x - 1) fib6;;  
: int list = [12; 7; 4; 2; 1; 0; 0]
```

# Higher-Order Functions Over Lists

```
# let rec map f list =
```

```
  match list
```

```
  with [] -> []
```

```
  | (h::t) -> (f h) :: (map f t);;
```

```
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

```
# map plus_two fib5;;
```

```
- : int list = [10; 7; 5; 4; 3; 3]
```

```
# map (fun x -> x - 1) fib6;;
```

```
: int list = [12; 7; 4; 2; 1; 0; 0]
```



# Mapping Recursion

---

- Can use the higher-order recursive map function instead of direct recursion

```
# let doubleList list =  
  List.map (fun x -> 2 * x) list;;  
val doubleList : int list -> int list = <fun>  
# doubleList [2;3;4];;  
- : int list = [4; 6; 8]
```



# Mapping Recursion

---

- Can use the higher-order recursive map function instead of direct recursion

```
# let doubleList list =  
  List.map (fun x -> 2 * x) list;;  
val doubleList : int list -> int list = <fun>  
# doubleList [2;3;4];;  
- : int list = [4; 6; 8]
```

- Same function, but no explicit recursion



# Folding Recursion

- Another common form “folds” an operation over the elements of the structure

```
# let rec multList list = match list
  with [ ] -> 1
       | x::xs -> x * multList xs;;
val multList : int list -> int = <fun>
# multList [2;4;6];;
- : int = 48
```

- Computes  $(2 * (4 * (6 * 1)))$



# Folding Recursion : Length Example

---

```
# let rec length list = match list
  with [ ] -> 0 (* Nil case *)
       | a :: bs -> 1 + length bs;; (* Cons case *)
```

```
val length : 'a list -> int = <fun>
```

```
# length [5; 4; 3; 2];;
```

```
- : int = 4
```

- Nil case [ ] is base case, 0 is the base value
- Cons case recurses on component list bs
- What do `multList` and `length` have in common?