# Programming Languages and Compilers (CS 421)

Talia Ringer (they/them)

4218 SC, UIUC

https://courses.grainger.illinois.edu/cs421/fa2023/

Based heavily on slides by Elsa Gunter, which were based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Course Logistics

# Assignments and Deadlines

- **MP1** is “due” on **Tuesday**
  - Not *directly* worth points
  - *But* **first quiz** is on **Tuesday**
  - Questions on first quiz are **literally from MP1**
  - All quizzes and the MPs before them are like this
  - Sorry for confusion
  - Quiz happens **in person**—please show up!
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help
- Any questions about this?

# Assignments and Deadlines

- **MP1** is "due" on **Tuesday**
  - Not *directly* worth points
  - *But* **first quiz** is on **Tuesday**
  - Questions on first quiz are **literally from MP1**
  - All quizzes and the MPs before them are like this
  - Sorry for confusion
  - Quiz happens **in person**—please show up!
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help
- Any questions about this?

See Lecture 1 Follow-up on Piazza for info on first question that I forgot to share Tuesday

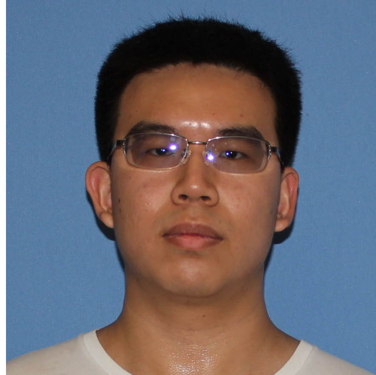
# Assignments and Deadlines

There are **5 quizzes**, not 4.
Slides last class had a typo.
Website is correct here!

- **MP1** is “due” on **Tuesday**
  - Not *directly* worth points
  - *But* **first quiz** is on **Tuesday**
  - Questions on first quiz are **literally from MP1**
  - All quizzes and the MPs before them are like this
  - Sorry for confusion
  - Quiz happens **in person**—please show up!
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help
- Any questions about this?

# Course TAs - Our Sections

Paul Krogmeier


James Luo

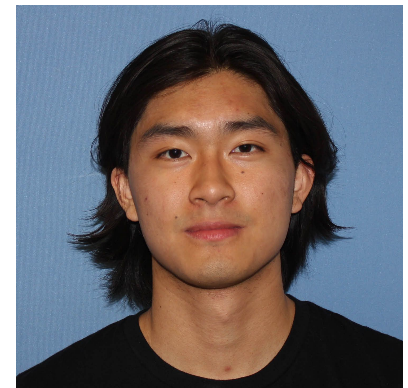
# Course TAs - Other Sections

Yerong Li

Shaurya Gomber

Deeya Bansal

Benjamin Darnell



Alan Yao

Questions about OCaml so far?

# Objectives for Today

- On Tuesday, you got started with OCaml
- Today, you will start to learn what actually happens when you run OCaml, like:
  - What happens when you **evaluate** an expression in OCaml?
  - How does OCaml **keep track** of values?
- This captures concepts present in **many languages**, so it is pretty broadly useful
  - Though there are some language-specific quirks

# Environments

# Environments

- **Environments** keep track of what value is associated with a given identifier
- Central to the semantics (meaning) and implementation of a language
- Notation:

  $$\rho = \{name_1 \rightarrow value_1, name_2 \rightarrow value_2, \ldots\}$$

  Using set notation, but describes a partial function
- Often stored as list, or stack
  - To find value start from left and take first match

# Environments

- **Environments** keep track of what value is associated with a given identifier
- Central to the semantics (meaning) and implementation of a language
- Notation:

  $$\boldsymbol{\rho} = \{name_1 \rightarrow value_1,\ name_2 \rightarrow value_2,\ \ldots\}$$

  Using set notation, but describes a partial function
- Often stored as list, or stack
  - To find value start from left and take first match

# Environments

X → 3

name → “Steve”

y → 17

b → true

Environments

# Environments

X → 3

name → “Steve”

y → 17

**region → (5.4, 3.7)**

b → true

Environments

# Environments

X → 3

name → "Steve"

y → 17

region → (5.4, 3.7)

**f** → …

b → true

# Environments

**X → 5**

name → “Steve”

y → 17

region → (5.4, 3.7)

f → …

b → true

Environments

# Changing the Environment

\# **2 + 3;;** **(* Expression *)**

**// doesn't affect the environment**

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

val test : bool = false

// $\rho_1$ = {test → false}

\# let a = 1

let b = a + 4;; (* Sequence *)

// $\rho_2$ = {b → 5, a → 1, test → false}

# Changing the Environment

```
# 2 + 3;;      (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;;      (* Declaration *)
val test : bool = false
```

// $\rho_1$ = {test → false}

```
# let a = 1
  let b = a + 4;; (* Sequence *)
```

// $\rho_2$ = {b → 5, a → 1, test → false}

# Changing the Environment

```
# 2 + 3;;      (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;;      (* Declaration *)
val test : bool = false
//  ρ1 = {test → false}
# let a = 1
  let b = a + 4;; (* Sequence *)
//  ρ2 = {b → 5, a → 1, test → false}
```

# Changing the Environment

```
# 2 + 3;;     (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;;     (* Declaration *)
val test : bool = false
//  ρ1 = {test → false}
# let a = 1
  let b = a + 4;; (* Sequence *)
//  ρ2 = {b → 5, a → 1, test → false}
```

# Changing the Environment

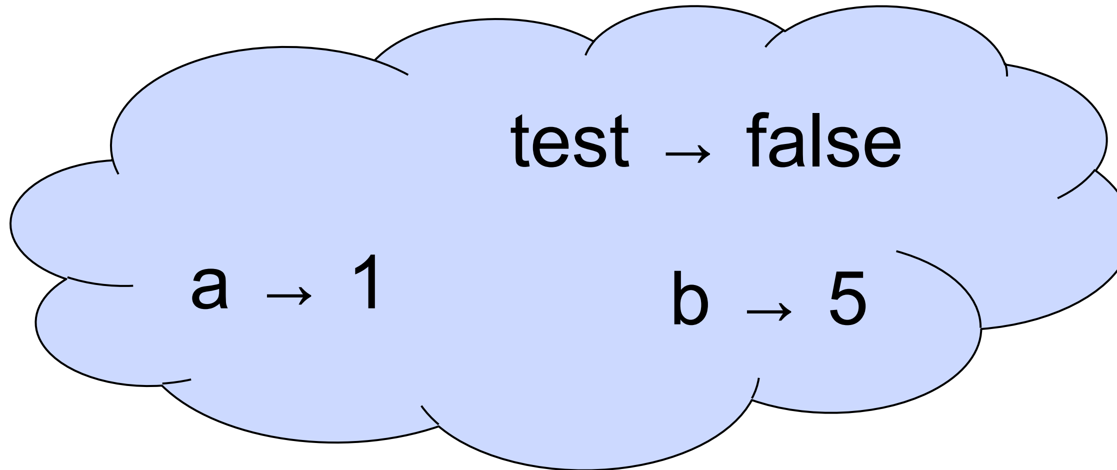
```
# 2 + 3;;     (* Expression *)
// doesn't affect the environment
# let test = 3 < 2;;     (* Declaration *)
val test : bool = false
```

// $\rho_1$ = {test → false}

```
# let a = 1
  let b = a + 4;; (* Sequence *)
```

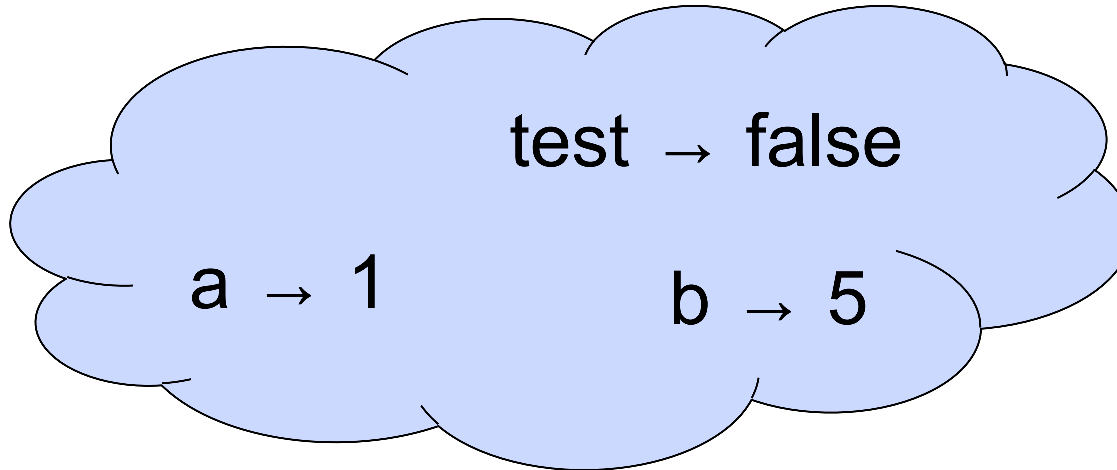
// $\rho_2$ = {**b** → **5**, **a** → **1**, test → false}

# Changing the Environment




// $\rho_2 = \{b \rightarrow 5, a \rightarrow 1, test \rightarrow false\}$

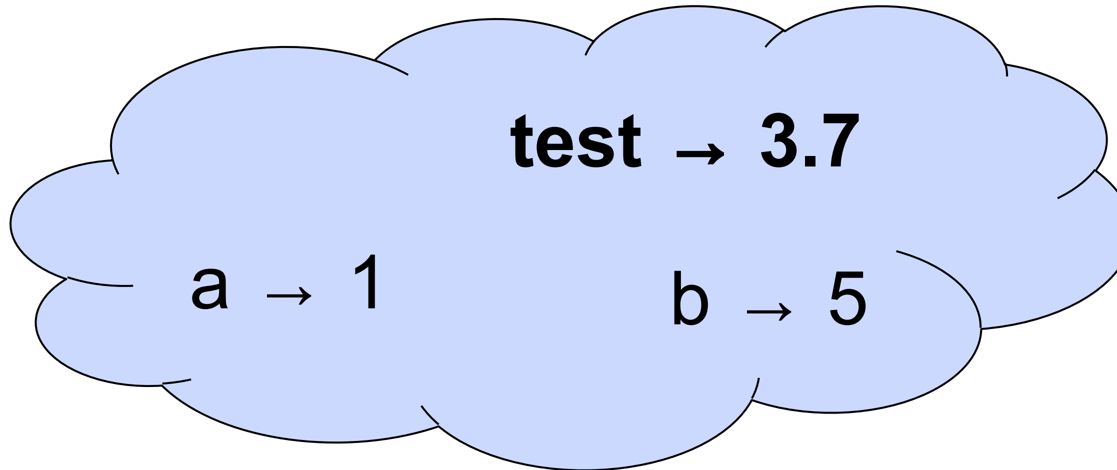
# Changing the Environment




(* Updating bindings *)

let test = 3.7;;

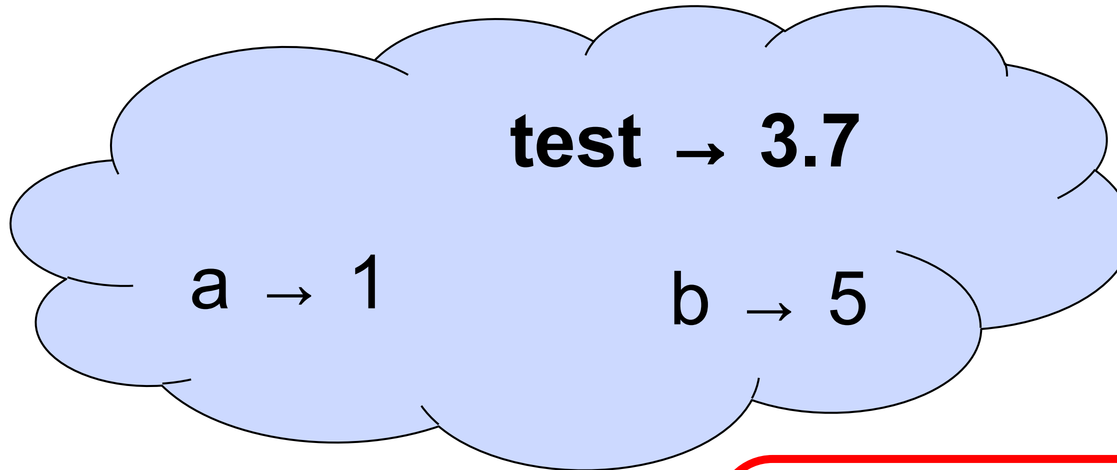
# New Bindings Hide Old




```
(* Updating bindings *)
let test = 3.7;;
```

// $\rho_3$ = {**test** → **3.7**, a → 1, b → 5}

# New Bindings Hide Old




**Aside**: It is common to implement as association lists, and just ignore rather than remove old bindings

(* Updating bindings *)
let test = 3.7;;
// $\rho_3$ = {**test** → **3.7**, a → 1, b → 5}

Let’s start WA1-IC together!

(This will help you with WA1.)

Questions so far?

# Variables and Environments

# **Global** Versus Local Variables

```
# let a = 1
  let b = a + 4;;
//  ρ = {b → 5, a → 1, …}
```

# Global Versus **Local** Variables

```
# let a = 1 in
  let b = a + 4 in
  b;;
- : int = 5
```

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1, …}
  let b = a + 4 in
  b;;
- : int = 5
```

Variables and Environments

# Global Versus **Local** Variables

\# let **a** = 1 **in** // $\rho_2 = \{\mathbf{a} \rightarrow \mathbf{1}, \ldots\}$

let **b** = a + 4 **in** // $\rho_3 = \{\mathbf{b} \rightarrow \mathbf{5},\ a \rightarrow 1, \ldots\}$

b;;

\- : int = 5

```
# let a = 1 in // ρ2 = {a → 1, …}
  let b = a + 4 in // ρ3 = {b → 5,  a → 1, …}
  b;;
- : int = 5
```

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1, …}
  let b = a + 4 in // ρ3 = {b → 5,  a → 1, …}
  b;;  // ρ4 = {…}
- : int = 5
```

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1, …}
  let b = a + 4 in // ρ3 = {b → 5,  a → 1, …}
  b;;  // ρ4 = {…}
- : int = 5
```

Local variables are not accessible outside of their local scope!

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1}
  let b = a + 4 in // ρ3 = {b → 5, a → 1}
  b;; // ρ4 = {}
- : int = 5
```

So imagine we started with an empty environment …

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1}
  let b = a + 4 in // ρ3 = {b → 5, a → 1}
  b;;  // ρ4 = {}
- : int = 5
# b;;
```

What is the result?

# Global Versus **Local** Variables

```
# let a = 1 in // ρ2 = {a → 1}
  let b = a + 4 in // ρ3 = {b → 5, a → 1}
  b;; // ρ4 = {}
- : int = 5
# b;;
Error: Unbound value b
```

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

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

# Values Fixed at Declaration Time

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

What is the result?

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# plus_x 3;;
- : int = 15
```

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# let x = 7;;
val x : int = 7
# plus_x 3;;
```

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# let x = 7;;
val x : int = 7
# plus_x 3;;
```

What is the result?

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# let x = 7;;
val x : int = 7
# plus_x 3;;
- : int = 15
```

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# let x = 7;;
val x : int = 7
# plus_x 3;;
- : int = 15
```

# Values Fixed at Declaration Time

```
# let x = 12;;
val x : int = 12
# let plus_x y = y + x;;
val plus_x : int -> int = <fun>
# let x = 7;;
val x : int = 7
# plus_x 3;;
- : int = 15
```

How does the environment keep track of functions? What's actually happening inside of the environment here?

# Closures

# Motivating Closures

- **Functions** are **first-class values** in this language
- What value does the environment record for a **function variable**, like plus_x?
- The answer is what we call a **closure**

# Defining Closures

- A **closure** is a pair of:
  - an **environment**, and
  - an **association** mapping:
    - a sequence of **variables** (input variables) to
    - an **expression** (the function body),
- written:

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

- where $\rho_f$ is the environment in effect when f is defined (if f is a simple function).

# Defining Closures

- A **closure** is a pair of:
  - an **environment**, and
  - an **association** mapping:
    - a sequence of **variables** (input variables) to
    - an **expression** (the function body),
- written:

$$f \rightarrow \langle\, (v1,\ldots,vn) \rightarrow exp,\ \rho_f \,\rangle$$

- where $\rho_f$ is the environment in effect when $f$ is defined (if $f$ is a simple function).

# Closure for plus_x

- When plus_x was defined, we had environment:

  $\rho_{plus_x} = \{\ldots, x \rightarrow 12, \ldots\}$

- Recall: let plus_x y = y + x

  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:

  $<y \rightarrow y + x, \rho_{plus_x}>$

- Environment just after plus_x defined:

  $\{plus_x \rightarrow <y \rightarrow y + x, \rho_{plus_x}>\} + \rho_{plus_x}$

# Closure for plus_x

- When plus_x was defined, we had environment:

  $\rho_{plus_x} = \{\ldots, x \to 12, \ldots\}$

- Recall: let plus_x y = y + x

  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:

  $<y \to y + x, \rho_{plus_x}>$

- Environment just after plus_x defined:

  $\{plus_x \to <y \to y + x, \rho_{plus_x}>\} + \rho_{plus_x}$

# Closure for plus_x

- When plus_x was defined, we had environment:

  $\rho_{plus_x} = \{\ldots, x \rightarrow 12, \ldots\}$

- Recall: let plus_x y = y + x

  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:

  $< y \rightarrow y + x, \rho_{plus_x} >$

- Environment just after plus_x defined:

  $\{plus_x \rightarrow < y \rightarrow y + x, \rho_{plus_x} >\} + \rho_{plus_x}$

# Closure for plus_x

- When plus_x was defined, we had environment:

  $\rho_{plus_x} = \{\ldots, x \to 12, \ldots\}$

- Recall: let plus_x y = y + x

  is really let plus_x = fun y -> y + x

- Closure for fun y -> y + x:

  $<y \to y + x, \rho_{plus_x}>$

- Environment just after plus_x defined:

  $\{plus_x \to <y \to y + x, \rho_{plus_x}>\} + \rho_{plus_x}$

(This will help you with WA1.)

Questions?

# Takeaways

- Languages (including OCaml) map variables to values in an **environment**.
- Functions in OCaml are **first-class** values—so in environments, function variables map to values.
- The particular values they map to are called **closures**. These store environments, as well as a map from input variables to the function body.
- In OCaml, the environment stored in a closure is the one from when the function was **first defined**.
- Doing **WA1** will help you develop more intuition for this—please ask for help if you need it!

# Next Class:
# Evaluating Expressions in OCaml

# Reminder: Also Next Class

- **MP1** is “due” on **Tuesday**
  - Not *directly* worth points
  - *But* **first quiz** is on **Tuesday**
  - Questions on first quiz are **literally from MP1**
  - All quizzes and the MPs before them are like this
  - Sorry for confusion
  - Quiz happens **in person**—please show up!
- All deadlines can be found on **course website**
- Use **office hours** and **class forums** for help

Next Class