

# **CSE216**

## **Foundations of Computer Science**

**Instructor: Zhoulai Fu**

**State University of New York, Korea**

Some slides taken from Prof. YoungMin Kwon. Thanks!

# Plan

- Today 1: Revision on Ocaml
- Today 2: Revision for Midterm 1
- Today 3: More clarification on Ocaml
- Thursday: Midterm 1

# Midterm 1

## (Same as in the announcement)

- - Date & Time: Thursday, Oct 12, from 9:00 AM to 10:30 AM.
- - Location: B203 (our regular Thursday morning lecture room).
- - Coverage: It includes all the topics discussed in the lectures up to next Tuesday.
- - Format: In-person. Unlimited physical notes. All answers must be submitted on BrightSpace. While I recognize this online submission method is cumbersome, it ensures our alignment with the school's ABET accreditation standards

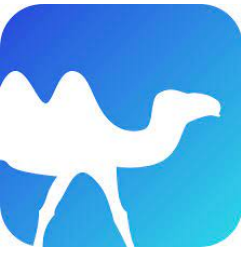
# Revision of our last lecture

# Revision

- Ocaml program = Definitions + Expressions
- `let x = 3` is a definition
- Everything else is an expression, and has a type before runtime
- An expression can be evaluated at runtime following beta-reduction

# Exercises on definitions/expressions and types

- **let  $x = 3$  in  $x+5$**  is an \_\_\_\_ of type \_\_\_\_
- **if  $b$  then  $3$  else  $5$**  is an \_\_\_\_ of type \_\_\_\_
- **let  $f\ x = x+1$**  is a \_\_\_\_ that associates \_\_\_\_ with an expression of type \_\_\_\_
- **let  $f\ x\ y = x + y$**  is a \_\_\_\_ that associates \_\_\_\_ with an expression of type \_\_\_\_



# Unit type

Expressions with no meaningful values have type **unit**.

Example: `print_string "okay"`

The type has a single value, written `()`

# Exercises

- `print_endline "hello"` is of type \_\_\_\_
- `print_end` is of type \_\_\_\_



# Exercises

- Evaluate **let x = 3 in let y = x + 2 in y\*y**
- Evaluate **let x= 3 in (let y= x+1 in y) \* (let z = x \* x in z)**

# Exercises

- For each of these expressions, what is its value and type. Write the value of a function by “<fun>”:
  - `let x = 3 in x * x`
  - `print_string "hello"`
  - `print_string`
  - `let f x y = x + y in f 5 6`
  - `let x = 5 in (if x > 0 then "pos" else "neg")`
  - `let f x y = x + y in f 5`

# Midterm Revision exercises

- Regular expressions
- Context-free grammar
- lambda calculus
- Ocaml

# Regular expression

In our class, we have studied the core regular expressions and some abbreviations based on those core regular expressions.

$r$	Meaning	Language $\mathcal{L}(r)$	Abbrev.	Meaning	Expansion
$a$	Character $a$	$\{ "a" \}$	$[aeiou]$	Set	$a e i o u$
$\epsilon$	Empty string	$\{ "" \}$	$[0-9]$	Range	$0 1 \dots 8 9$
$r_1 r_2$	$r_1$ followed by $r_2$	$\{ s_1 s_2 \mid s_1 \in \mathcal{L}(r_1), s_2 \in \mathcal{L}(r_2) \}$	$[0-9a-z]$	Ranges	$0 1 \dots 8 9 a b \dots y z$
$r^*$	Zero or more $r$	$\{ s_1 \dots s_n \mid s_i \in \mathcal{L}(r), n \geq 0 \}$	$r?$	Zero or one $r$	$r \epsilon$
$r_1 r_2$	Either $r_1$ or $r_2$	$\mathcal{L}(r_1) \cup \mathcal{L}(r_2)$	$r^+$	One or more $r$	$r r^*$

Give a regular expression over  $\{a, b\}$  that has  $aab$  as a substring

# Regular expression

(1) Write a regular expression pattern to match valid music notes according to the criteria below: A music note is represented by a capital letter A to G (inclusive) followed by an optional symbol: sharp (#), flat (b), or natural (n).

Example valid inputs: C, D#, Fb, Gn

Example invalid inputs: H, C##, Fm, C#b

Note. The sharp symbol ("#") is not a special character in regular expressions. So you do *not* need to escape it with a backslash.

# Context-free grammar

(2)\*\* What is the language generated by the following grammar? Select one answer from the four choices.

$S \rightarrow aSbb \mid \epsilon$

- A. The set of all strings that start with 'a' and end with two 'b'.
- B. The set of all strings that contain twice as many 'b's as 'a's.
- C. The set of all strings that contain an odd number of 'a's followed by an even number of 'b's.
- D. The set of all strings that contain  $n$  'a's followed by  $m$  'b's, where  $m = 2n \geq 0$

# lambda calculus

- Draw the parse tree of the following lambda terms, and reduce them to the normal form. Write the reduction in details.
- $(\lambda x.(x\ y))(\lambda z.z)$

# lambda calculus

- Draw the parse tree of the following lambda terms, and reduce them to the normal form. Write the reduction in details.
- $((\lambda x.((\lambda y.(x\ y))x))(\lambda z.w))$



# Ocaml

- Determine the type of the following Ocaml expression
  - `print_endline "hangul"`
  - `let f x y = x + y in f 3`

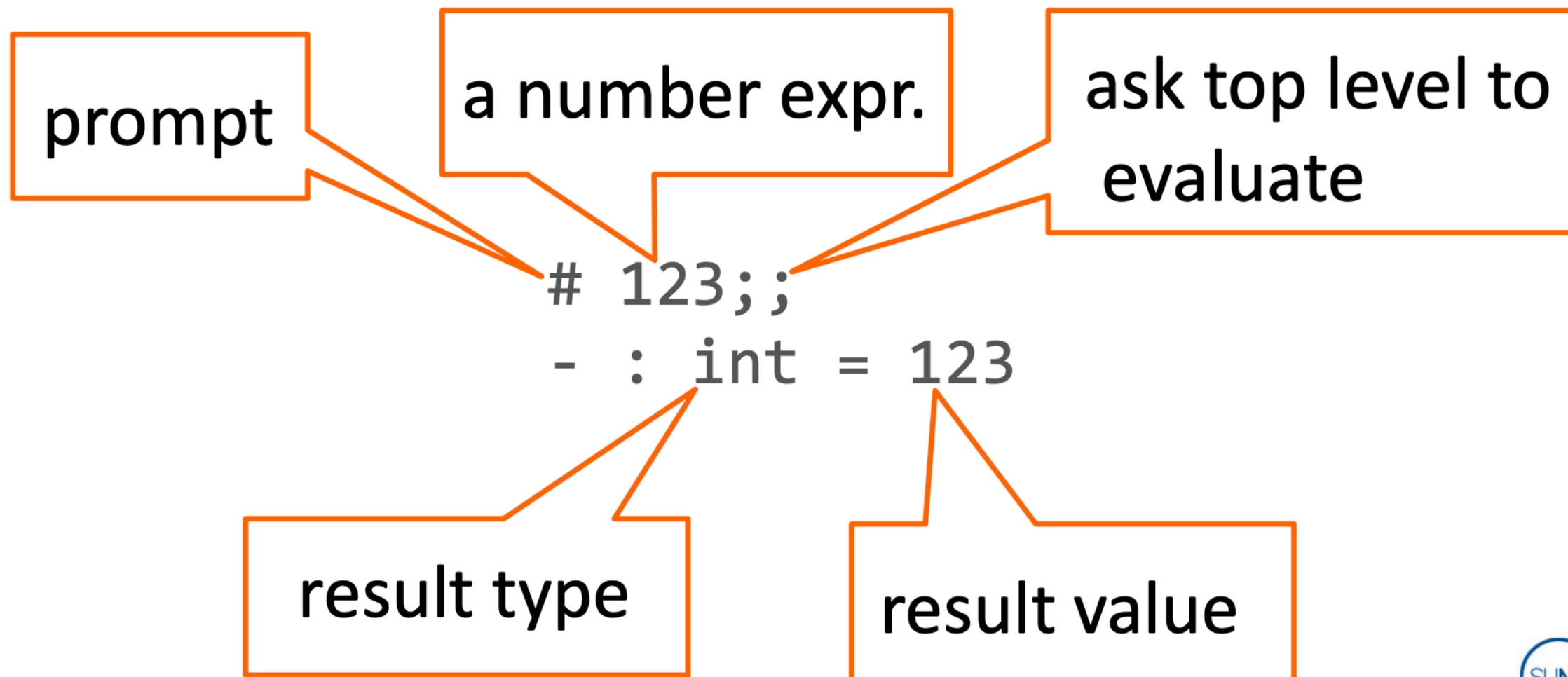
# Ocaml

- Determine the type of the following defined Ocaml functions
  - `let f x y = x + y`
  - `let f x = if x then "hello" else "annyeong haseyo"`

# Ocaml Basics

# Numbers

- A primitive expression
- Enter **123;;** in the OCaml interactive system (a.k.a. toplevel)



# Arithmetic Operators

- Using arithmetic operators

■ `+ - * / mod`      `+. -. *. /. **`

- Type `1 + 2 * 3` in the OCaml top level

```
# 1 + 2 * 3;;  
- : int = 7
```

- Type `1. +. 2. *. 3.`

```
# 1. +. 2. *. 3.;;  
- : float = 7.
```

# Arithmetic Operators (2)

- For each operator, there is a corresponding function

```
# (+);;
```

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

- Function application

- No parenthesis

- Arguments are separated by spaces

```
# (+) 1 2;;
```

```
- : int = 3
```

# Arithmetic Operators (3)

- Type coercion is not automatic in OCaml

```
# 1.0 + 2.0;;  
Characters 0-3:  
  1.0 + 2.0;;  
  ^^^
```

Error: This expression has type float but an expression  
was expected of type int

```
# 1.0 +. 2.0;;  
- : float = 3.  
# (+.);;  
- : float -> float -> float = <fun>
```

```
# float_of_int 1;; (* or float 1 *)  
- : float = 1.  
# int_of_float 1.5;;  
- : int = 1
```

# Abstraction by Names

- Create a variable to name a value
  - *let binding*

*let* *<variable>* = *<expr>*

```
# let x = 1 + 2;;  
val x : int = 3
```

```
# let add = (+);;  
val add : int -> int -> int = <fun>
```



# Abstraction by Names (2)

- Environment

- A data structure that keeps track of name-value pairs

```
# x;;  
- : int = 3
```

```
# add;;  
- : int -> int -> int = <fun>
```

```
# add x 1;;  
- : int = 4
```

# Evaluating Combinations

## ■ Example

```
# let add = (+);;  
val add : int -> int -> int = <fun>
```

```
# let mul = ( * );;  
val mul : int -> int -> int = <fun>
```

```
# let x = 5;;  
val x : int = 5
```

*notice the space:  
(\* would start a **comment***

```
# mul (add 1 (mul 2 3))  
      (add 4 x);;  
- : int = 63
```

# Lab exercise

- Try combining x and y using the ^ operator. What error do you see? Choose your own x and y
- Debug and make things right