# CS131: Programming Languages

Lun Liu

## Administration

- Class website: https://ccle.ucla.edu/course/view/17S-COMSCI131-1
- Piazza: piazza.com/ucla/spring2017/cs131

- Office Hour: Wed 11am – 1pm, BH 2432
- Email: lunliu93@gmail.com

## OCaml: Variables

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

```
# let x = 1;;
val x : int = 1
# let y = 4;;
val y : int = 4
# let z = x + y;;
val z : int = 5
```

- **let** *<variable>* = *<expr1>* **in** *<expr2>* (local variable)

# OCaml: Functions

- Anonymous Functions

```
# (fun x -> x + 2);;
- : int -> int = <fun>
# let plustwo = (fun x -> x + 2);;
val plustwo : int -> int = <fun>
# plustwo 3;;
- : int = 5
```

## OCaml: Functions

```
# let add x y = x + y;;
val add : int -> int -> int = <fun>
# add 1 2;;
- : int = 3
```

## OCaml: Functions

- Use **function** keyword

```
# let square = function x -> x * x;;
val square : int -> int = <fun>
```

- **function** built-in pattern matching
- **function [|]<pat> -> <expr> {| <pat> -> <expr>}**

```
# let square = fun y -> match y with
  | x -> x * x;;
val square : int -> int = <fun>
```

## OCaml: Pattern Matching

```
# let imply v = match v with
    (true,true)   -> true
  | (true,false)  -> false
  | (false,true)  -> true
  | (false,false) -> true;;
val imply : bool * bool -> bool = <fun>
```

```
# let imply v = match v with
    (true,x)  -> x
  | (false,x) -> true;;
val imply : bool * bool -> bool = <fun>
```

http://caml.inria.fr/pub/docs/oreilly-book/html/book-ora016.html

## OCaml: Lists

```
# [1;2;3];;
- : int list = [1; 2; 3]
# 1::[2;3];;
- : int list = [1; 2; 3]
# [[1];[2];[3]];;
- : int list list = [[1]; [2]; [3]]
# [1]::[[2];[3]];;
- : int list list = [[1]; [2]; [3]]
```

# OCaml: Lists

```
# [1::2]::[[3; 4]];;
Error: This expression has type int but an
expression was expected of type
            int list
# [1;2]::[[3; 4]];;
-: int list list = [[1; 2]; [3; 4]]
# [1; 2] @ [3; 4];;
- : int list = [1; 2; 3; 4]
```

## OCaml: Tuples

```
# (1, 2);;
- : int * int = (1, 2)
# ("what", 2, 5.1);;
- : string * int * float = ("what", 2, 5.1)
```

## OCaml: Tuples

```
# let my_add = fun (x, y) -> x + y;;
val my_add : int * int -> int = <fun>
# my_add 1 2;;
Error: This function has type int * int -> int
       It is applied to too many arguments;
maybe you forgot a `;'.
# my_add (1,2);;
- : int = 3
```

## Exercises!

## OCaml: Higher-order Functions

- Functions are like any other expressions
- Higher-order function: function that takes another function as an argument

```
# let rec map f l =match l with
            | [] -> []
            | h::t -> (f h)::(map f t);;
val map : ('a -> 'b) -> 'a list -> 'b list =
<fun>
# map (function x -> x+1) [1;2;3];;
- : int list = [2; 3; 4]
```

## OCaml: Currying

- Passing multiple arguments one at a time,  with functions returning other functions.

```
# let add x y = x + y;;
val add : int -> int -> int = <fun>
val add : int -> (int -> int) = <fun>
# let add2_curried = add 2;;
val add2_curried :
 int -> int = <fun>
# add2_curried 3 ;;
- : int = 5
```

## Exercises

## Tail Recursion

- **The return value of any given recursive step is the same as the return value of the next recursive call**.
- Calculation first then recursive call
- Allow compiler optimization for stack

## Backup

## Environment Setup

- OCaml arrow key problems
  - UTop (fancy!)
  - ledit
  - rlwrap