# CS131: OCaml (1)

Zhiping (Patricia) Xiao

Discussion 1C Week 0

# Why discussions?

- **Help you with your homework & more practicing.**
- Not everything is covered in the lectures.
- There are some common questions most of you have.
- An important source of feedback.

# Introduction

- Course website
  http://web.cs.ucla.edu/classes/fall19/cs131/
- Piazza https://piazza.com/class/k111z7l37yqid
- CCLE
  https://ccle.ucla.edu/course/view/19F-COMSCI131-1
- SEASnet
  https://www.seas.ucla.edu/acctapp/
- Previous years course websites easily accessed through Google / by changing the URLs.
- Professor / TAs' information on course website & on CCLE.
- Office hours 2 hours / TA / week, posted on CCLE.

- Why learning this course?
  - http://www.cs.pomona.edu/~kim/why.pdf
  - Essential skill in CS
  - More efficient implementation
  - Choose the right language for a certain project
  - The ability of learning new tools faster
  - Design & develop programming languages in the future

# Homework 1 overview

- Due **Oct 8th**, submit on **CCLE**
- Spec: [http://web.cs.ucla.edu/classes/fall19/cs131/hw/hw1.html](http://web.cs.ucla.edu/classes/fall19/cs131/hw/hw1.html)
  - **SEASnet** server usage mentioned, p.s. *Ssh*
  - Submit 3 files: **hw1.ml**, **hw1test.ml**, **hw1.txt**
- **DO NOT CHEAT** - we are running plagiarism checker on your submissions.
- Graded automatically with scripts
- Lateness policy: -1%, -2%, -4%, -8%, etc.
- Feel free to debug on local machines, but make sure it runs properly on the server (SEASnet).

# OCaml resources

- The official website
- Local Installation
- Documentation, among which only Pervasives and List modules are allowed in HW1.
- Try OCaml online

# Recommended Order to Dive-in

1. Setup your SEASnet account.
2. Local environment installation (online version to play with).
   a. Once installed, you can play with its console, similar with Python, by typing `ocaml` to start it, tryout any command such as `let a:int = 10;;`, type in `#use “myscript.ml”;;` to run code in the file named ***myscript.ml*** and `#quit;;` to exit.
3. Go through the basic tutorial.
4. Go through the structure tutorial.
5. Go through the if, loop and recursion tutorial.
6. Start coding while looking up the Pervasives and List modules’ documentations. If there’s any question: **first, search for help on Google**; next, if solved, you could share your problem & solution on Piazza, if not solved, you could ask for help on Piazza / come to any TA’s office hours.
7. Check debugging tutorial if needed.

# OCaml types and ranges

| **OCaml Type** | **Range** |
|---|---|
| int | 31-bit signed int (roughly +/- 1 billion) on 32-bit processors, or 63-bit signed int on 64-bit processors |
| float | IEEE double-precision floating point, equivalent to double in C |
| bool | A boolean, written either true or false |
| char | An 8-bit character, ONLY ascii supported |
| string | A string |
| unit | Written as () |

# OCaml - introduction

- A typical **functional** programming language.
  - Functions are  “first-class objects”, you can pass them into functions / treat them as if they were any other variables.
- Variables are **immutable**.
  - Once declared, no way to modify it; thus less side-effects.
  - **Lists** are immutable as well.
- It uses **type inference** to work out the types automatically
  - we can manually define the variable types (not necessary) like `let a:int = 3;;`
  - As a side-effect of type inference, functions / operators can't have overloaded definitions, that’s why we have, e.g. **+** for integer plus integer, and **+.** for float plus float.
- It never does **implicit** casts.
  - Be careful the difference between 2 and 2.0
  - Explicit cast example: *float_of_int*, *int_of_char*, etc.
- OCaml function **never returns**.
  - The last expression in a function becomes the result of the function automatically.

# OCaml - introduction

- Functional language prefer **recursion** than for/while loop.
    - No such supports like *break, continue, last*
    - The loops are second-class citizens with fairly limited use.
    - Homework 1 **requires** using recursion instead of loops.
- The basic algebraic operators only applies to variables of the **same** data type.
    - **+, -, *, /** are all operations on integers.
    - **+., -., *., /.** are all operations on floats.
    - **mod** is the modulo operator.
    - **>, >=, <, <=** are integer or float comparison operators
    - **=** compares two values and returns if they are equal
    - ^ is the string concatenation operator
    - **&&, not, ||** are the most-common logic operators.
    - ** is the power operation, only applies to float.
    - Etc. For more, visit the Pervasives module.
- **Lists** are **homogenous**, **tuples** are **heterogenous**.
    - Every list element must be of the same type, no limit on tuple elements' type.

# OCaml basic syntax and function calls

- End of line: `;;`
- Comments: `(* the content of comments *)`
  - There’s no single-line comment
- Variables: `let a = 10;;`
- If statement: `if a < 5 then a + 1 else a * 2;;`
- Functions:
  - Defining and calling an ordinary function

    ```
    let average a b =
        (a +. b) /. 2.0;;
    average 1. 5.;;
    ```

  - Defining and calling a recursive function

    ```
    let rec range a b =
        if a > b then []
        else a :: range (a+1) b;;
    range 1 10;;
    ```

  - Polymorphic functions (somewhat similar with ***templates***, no specific type)

    ```
    let identical x = x;;
    ```

  - Defining and using anonymous function

    ```
    (* two times x *)
    fun x -> x * 2;;
    (* sum of x, y *)
    fun x y -> x + y;;
    (* usage *)
    (fun x -> x * 2) 3;;
    (fun x y -> x + y) 1 2;;
    ```

  - Function with local variable

    ```
    let plus_3_plus_5_times_8 x =
        let a = 3
        and b = 5 in
        let c = a + b in
        (x + a + b) * c;;
    plus_3_plus_5_times_8 0;;
    ```

# OCaml basic data structure - List and tuples

```
(* lists are homogenous, immutable *)
(* empty list *)
[];;
(* defining lists by ::
   a::b is equivalent with cons a b
   where a is a variable *)
let a1 = cons 1 [];;       (* => [1]*)
let b1 = 1::2::[];;        (* => [1;2]*)
(* defining lists by directly assigning values *)
let a2 = [1];;             (* => [1]*)
let b2 = [1;2];;           (* => [1;2]*)
(* they are equivalent
   note that this comparison compare values *)
a1 = a2;;                  (* true *)
b1 = b2;;                  (* true *)
(* append (@) is also very useful
   a@b is equivalent with append a b
   and also equivalent with concat [a;b]
   Where a and b are both lists *)
b1@b2;;                    (* => [1;2;1;2]*)
append b1 b2;;             (* => [1;2;1;2]*)
concat b1 b2;;             (* => [1;2;1;2]*)
(* separate the list into head and the rest *)
List.hd [1;2;3];;          (* => 1 *)
List.tl [1;2;3];;          (* => [2;3] *)
```

```
(* tuples are heterogenous, immutable *)
("3", 3);;
("3", 4, 0.5, "turtles");;
(* get the first item in tuple with ‘fst’
   get the second item in tuple with `snd’
   note that they only work on tuples with size 2 *)
fst ("1", 2);;                    (* => "1" *)
snd ("1", 2);;                    (* => 2 *)

(* two ways of including library functions
   from List module *)
List.length [1;2;3];;             (* 3 *)
(* or option 2: *)
open List;;
length [1;2;3];;                  (* 3 *)
```

# OCaml Pattern Matching

```
(* syntax
  match v with
    | patter -> …
    | _ -> ...
  _ is the placeholder,
    and it matches with everything.
  can be used for iterate over lists.*)
(* example 1 *)
(* multiply two items in a tuple*)
let mult_tuple t =
  match t with
    | (a, b) -> a * b;;
mult_tuple (3, 2);;            (* => 6 *)
(* example 2 *)
(* iterate through list and sum the elements *)
let rec sum_list l = match l with
  | [] -> 0
  | head::rest -> head + (sum_list rest);;
  (* in this case, head is the first element
     rest is a list, the remaining part of the list
   *)
sum_list [1;2;3];;             (* => 6 *)
```

```
(*
  note that the order of pattern matching
         matters a lot.
  e.g. if we put | _ -> … as the first pattern
       then it’ll always be matched
           and the rest patterns will remain unused.
 *)
```

# OCaml Debug

- Observing type error by understanding the **print-back** from OCaml.
  - `int * float * char`: a 3-element tuple has elements of types int, float, char. E.g. `(1, 1.0, ‘1’)`
  - `int -> float -> char -> bool = <fun>`: a function with 3 parameters of types int, float, char, and “returns” a boolean value.
- Debugging tools provided by OCaml.
  - Trace: from the ***interactive toplevel***.
    - `let rec fib x = if x <= 1 then 1 else fib (x - 1) + fib (x - 2);;`
    - `#trace fib;;                  (* traced *)`
    - `fib 3;;                       (* will print the traced info *)`
    - `#untrace fib;;                (* untraced *)`
  - OCaml debugger which allows analysing programs compiled with **ocamlc** (refer to official documentation for more details).

# Understanding HW1: Grammars

- Symbol
  - Terminal: A symbol which you cannot replace with other symbols
  - Non-terminal: A symbol which you can replace with other symbols
- Rule
  - From a non terminal symbol, derive a list of symbols
- Grammar
  - A starting symbol, and a set of rules that describe what symbols can be derived from a non terminal symbol

- Example of a simple grammar:
  - symbols: S, A, B, a, b
  - Non-terminals: S, A, B
  - Terminals: a, b
  - **Starting symbols: S**
  - Rules:
    - S -> A
    - S -> B
    - A ->aA
    - A -> a
    - B -> bB
    - B -> b
  - Question: How to derive **aaa** ?

# Requirements: implementing & testing

- In **hw1.ml** implementing functions:
  - subset
  - equal_sets
  - set_union
  - set_intersection
  - set_diff
  - computed_fixed_point
  - Filter_reachable
  - any auxiliary types and functions (if needed)
- In **hw1test.ml**
  - Supply **at least one** test case for each of the above functions in the style shown in the sample test cases.

- **Hw1.txt** is an after-action report
  - Assessment
    - why you solved it this way
    - other approaches that you considered and rejected (why)
    - any weaknesses
  - Plain text file, ≤ 2000 bytes long
  - Instructions & advise on how to write a good report etc.
- **Attention:** Please **do not** put your name, student ID, or other personally identifying information in your files.

# Coding: encouraging “base on” other functions

- Code-reuse is better than pasting the same logic everywhere.
- It further encourages modularize your code.
- Some hints:
    - **equal_sets** could use **subset**
    - **set_union** could use **set_diff**
    - **filter_reachable** could use
        - **equal_sets**
        - **computed_fixed_point**