CS 162: Programming Languages
Discussion Session 2
More on OCaml

Yanju Chen

Office Hour: Wednesdays 1-3pm, TA Trailer

Quick Start

REPL Online (Read-Eval-Print Loop)

https://repl.it/languages/ocaml

Compile and Run

```
ocamlopt -o <executable> <source_code>
```

Reference

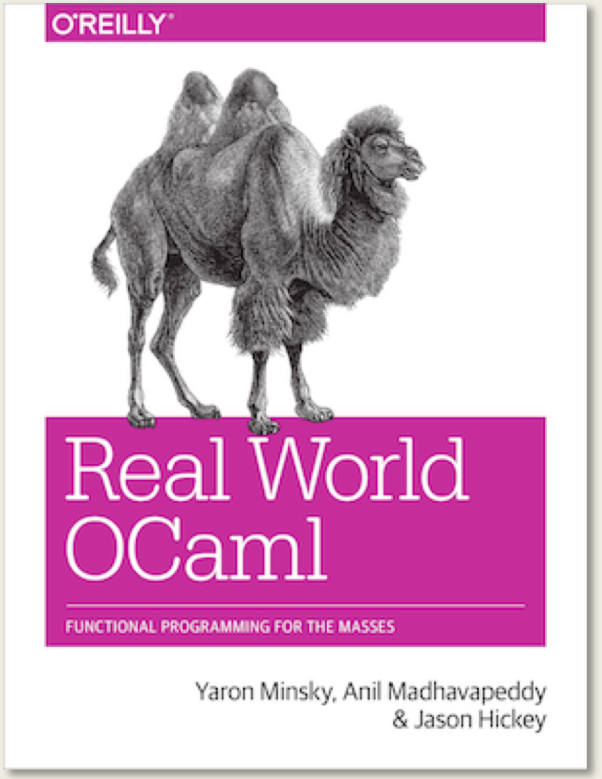
https://v1.realworldocaml.org/v1/en/html/index.html
https://ocaml.org/learn/tutorials/99problems.html

Notes

CSIL and Phelps machines have OCaml installed already.

Comment

```
(* The following code is broken! *)
```



Input and Output Functions

Output Examples

```
# print_char 'a' ;;
# print_string "apple" ;;
# print_int 5 ;;
# print_float 3. ;;
# print_newline () ;;
```

Type Conversion

```
# float_of_int 4 ;;
# int_of_float 4. ;;
# float_of_string "4." ;;
# int_of_string "4" ;;
```

Read from Standard Input

```
# read_line () ;;
# read_int () ;;
# read_float () ;;
```

Example: Read, Plus One and Return

```
# let line = read_int () in
    print_int (line +1)
;;
```

let expression

let Expression

```
let <variable> = <expr>
let <variable> = <expr1> in <expr2>
```

Notes

A let binding in an inner scope can shadow, or hide, the definition from an outer scope.

Example of let Expressions

```
# let x = 3;;
# let y = 4;;
# let z = x + y;;
```

Useful String Operations

String Functions

```
# String.length "apple" ;;
# String.get "peach" 1 ;;
# String.sub "pineapple" 4 5 ;;
```

String Operations

```
# "peach".[1] ;;
# "pine" ^ "apple" ;;
# "apple" == "apple" ;; (* Is this correct? *)
# "apple" = "apple" ;; (* How about this? *)
# "apple" <> "banana" ;;
```

Notes

Use “=“ for comparison unless you know what you are doing using “==“.
See https://stackoverflow.com/a/25901920

Reference

https://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html

Useful List Operations

List Operations

```
# [1;2;3;4] ;; (* list notation *)
# 1::(2::(3::[])) ;; (* list notation *)
# [1;2;3]@[4;5;6] ;; (* list concatenation *)
# 9::[1;2;3] ;; (* add an element to the list *)
# List.hd [1;2;3] ;; (* head element *)
# List.tl [1;2;3] ;; (* tail element *)
```

Question

How do we add elements to the end of a list?

List Operations

```
# [1;2;3]@[4] ;;
```

Question

How do we loop over lists?

Looping over Lists

```
List.iter <function> <list>
List.map <function> <list>
(* or use pattern matching *)
```

Looping over Lists

```
# let my_list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10];;
# let f elem = Printf.printf "I'm looking at element %d now\n" elem in
    List.iter f my_list
;;
```

Pattern Matching (of List)

Pattern Matching

```
# let plus_one_match x =
    match x with
    | 0 -> 1
    | 1 -> 2
    | _ -> x + 1
;;
```

Pattern Matching

```
# let plus_one_if x =
    if x=0 then 1
    else if x=1 then 2
    else x+1
;;
```

Matching a List

```
# <head_element> :: <list_of_other_elements>
# <elem1> :: <elem2> :: ... <list_of_other_elements>
```

Notes

Patterns can be nested.

Example: Dropping Specific Value

```
# let rec drop_value l to_drop = match l with
    | [] -> []
    | hd :: tl ->
      let new_tl = drop_value tl to_drop in
      if hd = to_drop then new_tl else hd :: new_tl
;;
```

Practice Problem #1

Practice Problem #1: A Power Function for Integer

In OCaml you can easily use <float1> ** <float2> to compute the power of <float1>. But its int version is not included in standard library. Can you implement the int version of the power function?

Solution #0

```
# let power base exponent =
  int_of_float (float_of_int base ** float_of_int exponent)
;;
```

Solution #1

```
# let rec power base exponent =
  if exponent = 0 then 1
  else base * (power base (exponent-1))
;;
```

Practice Problem #2

Practice Problem #2: String to List of Char

In order to count the number of appearances of different characters, your friend decided to first of all write a helping function that converts a string to a list of chars. Can you help your friend implement the helping function?

Solution #2: The explode Function

```
# let explode s =
    let rec exp i l =
      if i < 0 then l else exp (i - 1) (s.[i] :: l) in
    exp (String.length s - 1) []
;;
```

Solution #2: An Easier-To-Understand Version

```
# let rec s2lc str = match str with
    | "" -> []
    | str -> str.[0] :: (s2lc (String.sub str 1 ((String.length str)-1)))
;;
```

Practice Problem #3

Practice Problem #3: Count the Specific Character in a String

In fact, you realized that you don't even need a helping function when counting the number of specific character in a string. So, can you write that function down?

Solution #3

```
# let rec count_ch str ch = match str with
    ...
```

Solution #3

```
# let rec count_ch str ch = match str with
    | "" -> 0
    | str -> if str.[0]=ch then 1 + ( count_ch (String.sub str 1 ( (String.length str)-1 ) ) ch )
             else 0 + ( count_ch (String.sub str 1 ( (String.length str)-1 ) ) ch )
;;
```

Practice Problem #4

Practice Problem #4: Computing the Moving Average

Given a list of floats, compute the averaged value of every consecutive 3 floats. If the length of the list is smaller than 3, just return 0.

For example:

- **input [1.;2.;3.;4.;5.] you get [2.;3.;4.]**
- **input [3.;12.;9.;6.;81.] you get [8.; 9.; 32.]**

Solution #4

```
# let rec moving_average l = match l with
    ....
```

Solution #4

```
# let rec moving_average l = match l with
      | h0::h1::h2::t -> [ (h0+.h1+.h2) /. 3.] @ moving_average (h1::h2::t)
      | _ -> []
;;
```

What I wish I knew when learning OCaml
https://baturin.org/docs/ocaml-faq/