CS 131 - Week 1

TA : Tanmay Sardesai

How to find these slides

Piazza -> CS 131 -> Resources -> Discussion 1B

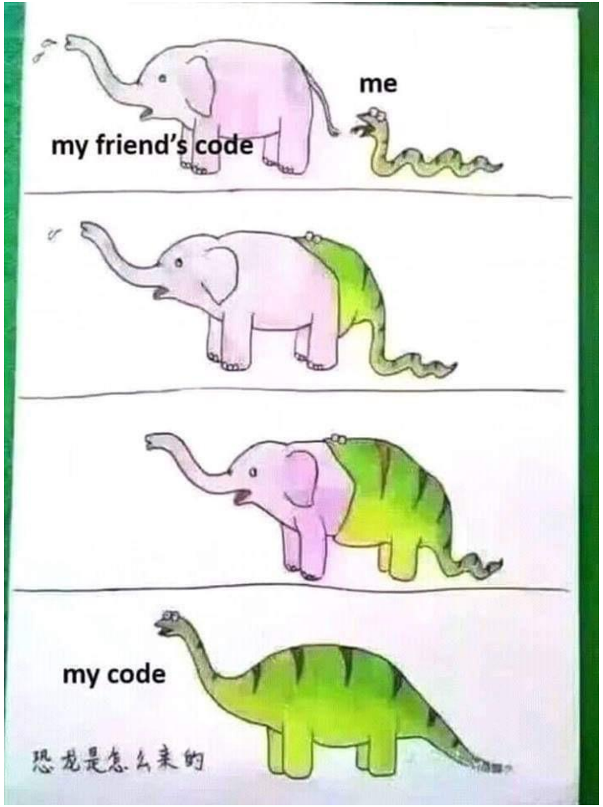
https://piazza.com/class/jqr2kb45gw2jt

Intro

- Email (tanmays@cs.ucla.edu)
- Office Hours (Thursday 1:30 pm - 3:30 pm)
- Bolter Hall 3256S-A

Homework Announcements

- HW1 due 01/16 11:55 pm
- HW2 due 01/29 11:55 pm
- All homeworks will be submitted to ccle
- Some homeworks will have automated grading scripts
    - Make sure code compiles
    - Make sure that you follow the function signatures
    - Follow all the instructions
- HW done independently
- Use piazza for all questions
- HW is checked for plagiarism

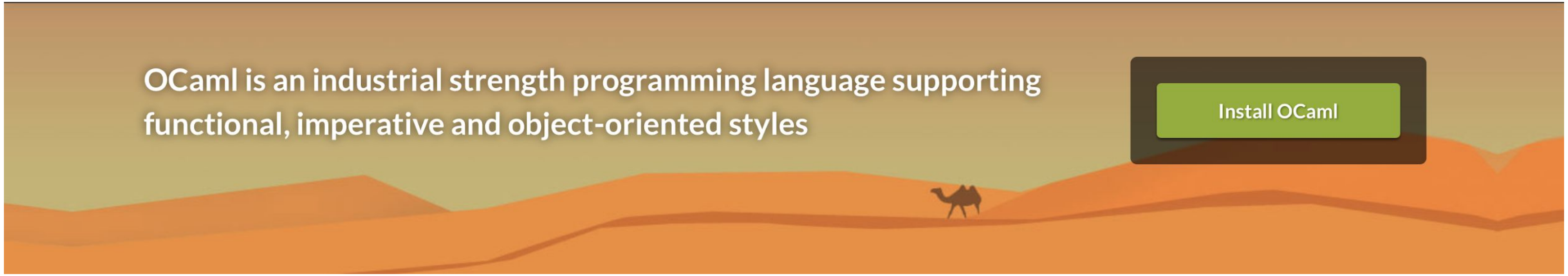


Topics covered today

- Review of Last Class
- High Order Functions and more
- Grammer
- Ocaml Hands on if hdmi works..

Ocaml

https://ocaml.org



OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles

Ocaml

- Functional programming language
- Immutable “variable”
- No loops without side effects
- Statically typed

Ocaml

- Functional programming language
- Immutable “variable”
- No loops without side effects
- Statically typed

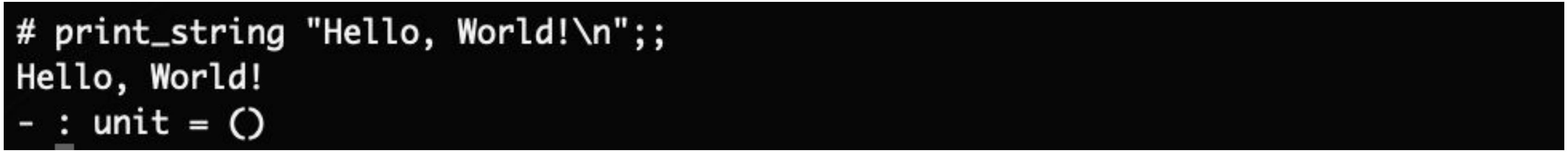
```
for i = 1 to n_jobs () do
  do_next_job ()
done
```

```
let i = 1 in
do_next_job ();
let i = 2 in
do_next_job ();
let i = 3 in
do_next_job ();
  ...
let i = n_jobs () in
do_next_job ();
()
```

Installing Ocaml

- Installation instructions: https://ocaml.org/docs/install.html
    - Make sure you are using version 4.07.0
- You can connect to SEASnet servers
    - lnxsrv06.seas.ucla.edu, lnxsrv07.seas.ucla.edu, lnxsrv09.seas.ucla.edu, and lnxsrv10.seas.ucla.edu
    - If you don’t have a SEASnet account, apply for one ASAP: https://www.seas.ucla.edu/acctapp/
    - Make sure that the OCaml version is correct (ocaml --version should show 4.07.0)
        - If not, check that /usr/local/cs/bin is in your path (which ocaml)
        - Instructions for this are at the course website under homework #1

First program



```
# print_string "Hello, World!\n";;
Hello, World!
- : unit = ()
```

- print_string is a function, “Hello, World!\n” is the parameter to the function.
- Statement ends with ;; (necessary in interactive mode, not in compiled)
- 2nd line is printed by the function
- Last line is unit() which is the return value of our function call. (not very informative for this program)
- #use “filename.ml” to load a file in the interpreter

Comments

- (* This is a comment *)
- (* This
   * is
   * also
   * a
   * comment *)
- (* Nested (* comments *) are allowed too *)

“Variables”

- These are not really variables as we cannot change the value

```
(* Global variable declaration *)
let my_val = 42;;


(* Local variable declaration *)
let fortytwo =
        let six = 6
        and seven = 7
        in six * seven
```

Lists

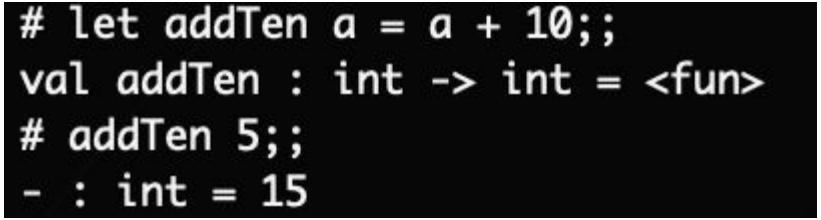
- Defined using square brackets, values are separated by semi-colons
- let numbers_list = [1 ; 2 ; 3 ; 4 ; 5]
- Lists are immutable singly-linked lists under the hood
  - Random access is slow. Iteration is fast

Lists

- Lists consists of head and tail. Use List.hd and List.tl to access them
- Adding new element to the beginning of a list
  - 0 :: [1 ; 2; 3] gives [0 ; 1 ; 2 ; 3]
  - 0 :: 1 :: 2 :: [3] gives the same thing
  - 0 :: 1 :: 2 :: 3 gives an error … WHY?
- Lists are immutable too so need to create a new list if you want to change value

Functions

- Functions

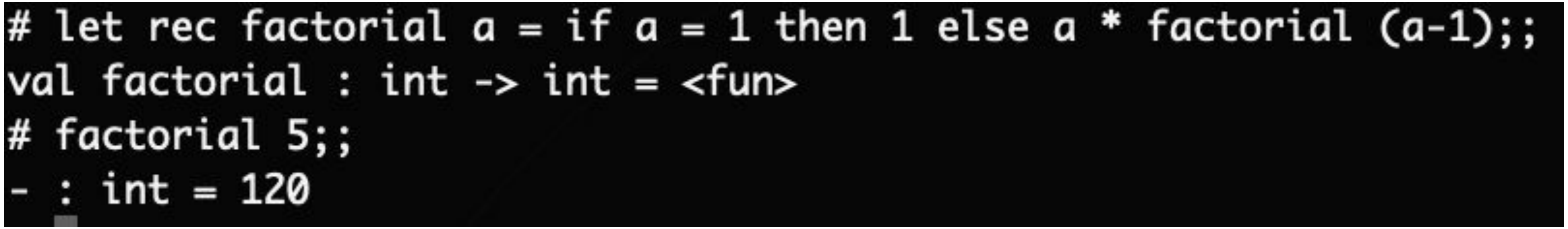


```
# let addTen a = a + 10;;
val addTen : int -> int = <fun>
# addTen 5;;
- : int = 15
```

- Not input and output are inferred to be integers
- let addTenFloat a = a +. 10.0;; for adding 10 to floats
- No need to use parentheses to call a function

Recursive Functions

- We have to specify they are recursive by stating `let rec …`



```
# let rec factorial a = if a = 1 then 1 else a * factorial (a-1);;
val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
```

Anonymous functions

- Functions with no name
- Usually provided to other functions in line. For example List.map or List.filter

Anonymous functions

- Functions with no name
- Usually provided to other functions in line. For example List.map or List.filter
- Example:
  - fun x -> x + 10;;
    - - : int -> int = <fun>
  - List.map(fun x -> x * 10) [1;2;3;4];;
    - - : int list = [10; 20; 30; 40]
- let addTen = fun x -> x + 10;; and let addTen x = x + 10;; are same

Higher Order Functions

- Higher-order Functions either take functions as parameters, return functions or both.

Higher Order Functions

- Higher-order Functions either take functions as parameters, return functions or both.
- Example of function as a parameter

  ```
  let twice f x = f (f x);;
  twice (fun i -> i + 10) 5;;        (return 25)
  ```

Higher Order Functions

- Higher-order Functions either take functions as parameters, return functions or both.
- Example of function as a parameter

  ```
  let twice f x = f (f x);;
  twice (fun i -> i + 10) 5;;        (return 25)
  ```

- Example of function as a result

  ```
  let makeMultiplier x = fun a -> a*x;;
  let mul5 = makeMultiplier 5;;
  List.map mul5 [1;2;3;4;5];;     (return [5; 10; 15; 20; 25])
  ```

User defined types in Ocaml

```
type foo =
    | Nothing
    | Int of int
    | Pair of int * int
    | String of string;;
```

```
Nothing
Int 3
Pair (4, 5)
String "hello"
...
```

C:

```
enum sign { positive, zero,
negative };
```

Ocaml:

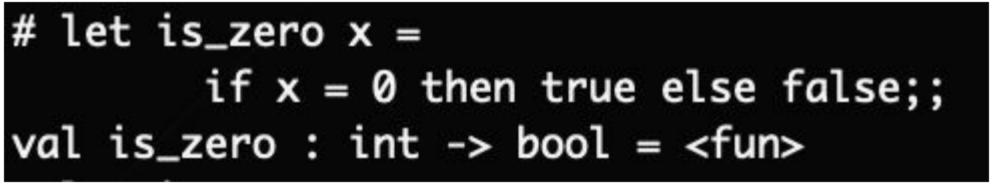
```
type sign = Positive | Zero |
Negative
```

Recursive Types

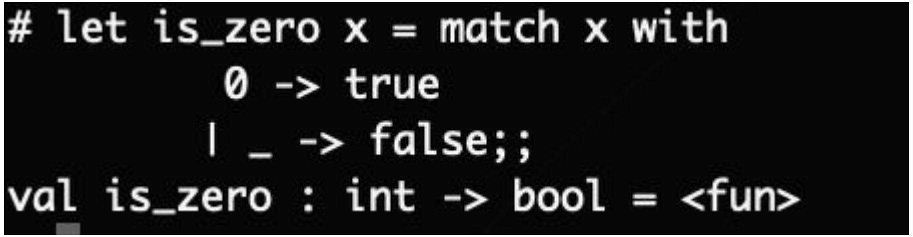
```
# type binary_tree =
    | Leaf of int
    | Tree of binary_tree * binary_tree;;
```

Pattern Matching

Some problems require use of conditionals or pattern matching



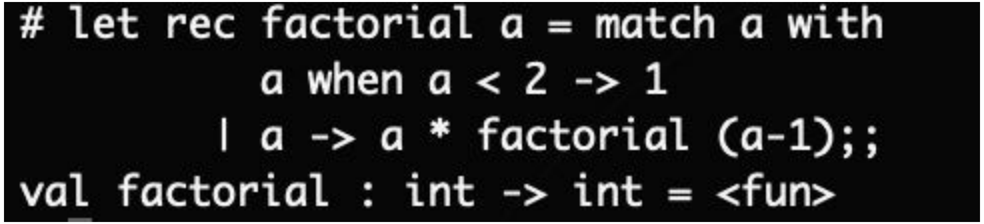
```
# let is_zero x =
        if x = 0 then true else false;;
val is_zero : int -> bool = <fun>
```



```
# let is_zero x = match x with
          0 -> true
         | _ -> false;;
val is_zero : int -> bool = <fun>
```

Pattern Matching

Pattern matching can include conditionals using when keyword



```
# let rec factorial a = match a with
          a when a < 2 -> 1
        | a -> a * factorial (a-1);;
val factorial : int -> int = <fun>
```

List module

List.rev - reverses a list

List.flatten or List.concat - Concatenate the list

List.map - Applies a function to each element of the list

List.filter - Applies a boolean function to each elements of the list. Returns only once that returns true

List.mapi - Applies a function that takes 2 arguments, index and value

List.hd - Return 1st element of the list

List.tl - Return eveything but 1st element of the list

List module

List.forall - Checks if each element of the list is valid for a boolean function

List.exists - Checks if at least 1 element of the list is valid for a boolean function

List.mem - Checks if an element is equal to something in the list

Pervasives module

Comparisons: =, <>, >=,<=,>,<, ==, !=, min, max

Boolean: not, &&, ||

Integer Arithmetic: +, -, *, /, mod, abs, succ, pred

Floating Arithmetic: +., -., *., /., **, sqrt, log, sin, cos, ……

Conversions: int_of_float, string_of_bool, string_of_int, ……

Pair operations: fst, snd

Context Free Grammar

- Review
  - 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

Example of Grammar

Symbols: E, T, F, *, N,0,1,2,3,4,5,6,7,8,9, (,)

Non-Terminals: E, T, F, N

Terminals: *,0,1,2,3,4,5,6,7,8,9,(,)

Starting Symbol: E

Rules:

E -> E + T
E -> T
T -> T*F
T -> F
F -> (E)|N
N -> 0
N -> 1
….
N -> 9

Rules Abbr:

E -> E+T | T
T -> T*F | F
F -> (E)|N
N -> 0 | 1 | 2 | … | 9

Example Derivates

3 + 4

Rule	After it is applied
start	E
E	E + T
E + T	T + T
T + T	F + T
F + T	N + T
N + T	3 + T
3 + T	3 + F
3 + F	3 + N
3 + N	3 + 4

3 + (4 * 8)

start	E
E	E + T
E + T	T + T
T + T	F + T
F + T	N + T
N + T	3 + T
3 + T	3 + F
3 + F	3 + (E)
3 + (E)	3 + (T)
3 + (T)	3 + (T * F)
3 + (T * F)	3 + (F * F)
3 + (F * F)	3 + (N * F)
3 + (N * F)	3 + (4 * F)
3 + (4 * F)	3 + (4* N)
3 + (4 * N)	3 + (4 * 8)

HW 1

1. Write a function to determine if one list is a subset of another list - i.e. is every element of list *a* also in list *b*
2. Write a function to determine if two sets are equal - Both should contain the same elements
3. Write a function that returns the union of two sets - A set that has every element that exists in either set or in both of them
4. Write a function that returns the intersection of two sets - A set that contains every element that is in both of the given sets
5. Write a function that returns the difference of two sets - All elements that belong to the first set but do not belong to the second set
6. Write a function that returns the fixed point of a function - Value x where f(x) = x

HW 1

7. Write a function that takes a grammar as its input and returns a grammar where all the unreachable rules have been removed

Submission

- 3 files
  - hw1.ml - Your code
  - hw1test.ml - Your tests
  - hw1.txt - Your assessments

https://medium.com/@cscalfani/so-you-want-to-be-a-functional-programmer-part-1-1f15e387e536

https://ocaml.org/learn/tutorials/functional_programming.html

https://ocaml.org/learn/tutorials/basics.html

https://ocaml.org/learn/tutorials/data_types_and_matching.html

https://ocaml.org/learn/tutorials/if_statements_loops_and_recursion.html

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

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

https://ocaml.org/learn/tutorials/99problems.html

Hands On With Questions