CS3110 OCaml Cheat Sheet

(); unit
3; int
3.0; float
'A'; char
"xyz"; string
false; bool
3 < 5 && true; bool
Some 3; int option
None; 'a option
ref 3; int ref
[3; 4]; int list
[]; 'a list
(2, "xyz", 3.0); int * string * float
fun x -> x + 1; int -> int
fun x y -> x + y; int -> int -> int
fun () -> 4; unit -> int
Not_found; exn
if x < 0 || x > 0
then "nonzero"
else "zero"
match x with
| 0 -> "zero"
| 1 -> "one"
| _ -> "more than one"
Char.code 'a'; 97
Char.code 'A'; 65
Char.code '0'; 48
Char.chr 97; 'a'
(fun x -> x + 1); 3 4
"x" ^ "y" ^ "z"; "xyz"
-5 + 7; 2
let compose f g x = f (g x) in
let f x = x * x in
let ff = compose f f in
let fff = compose f ff in
(f 2, ff 2, fff 2); (4, 16, 256)
List.hd [3; 4]; 3
List.tl [3; 4]; [4]
List.tl [4]; []
3 :: [4; 5]; [3; 4; 5]
[1;2;3] @ [4;5;6]; [1;2;3;4;5;6]
fst (2, "abc"); 2
snd (2, "abc"); "abc"
type 'a option = Some of 'a | None
let (x, y) = (Some 111, 2999) in
match (x, y) with
| (Some z, _) -> z + y
| (None, _) -> y; 3110
let e = exp 1. in
let pi = 2. *. asin 1. in
(e, pi); (2.7182818284590451, 3.1415926535897931)
let uncurried (x, y) = x + y in
let curried x y = x + y in
(uncurried (1, 2), curried 1 2); (3, 3)
let rec sum (x : int list) : int =
match x with
[] -> 0
| u :: t -> u + sum t
List.fold_left : ('a->'b->'a) -> 'a -> 'b list -> 'a
List.fold_left ('"x" "a";"b";"c"); "xabc"
List.find : ('a -> bool) -> 'a list -> 'a
List.find (fun x -> x > 10); [1;5;10;13;19]; 13
List.find (fun x -> x > 10); [1;5;10]; raises Not_found

String.length "hello"; 5
List.length [8; 9; 10]; 3
List.rev [8; 9; 10]; [10; 9; 8]
List.nth [8; 9; 10]; 2 10
List.nth [8; 9; 10]; 3; raises Failure "nth"
let (x, y) = (Some 111, 2999) in
match (x, y) with
| (Some z, _); z + y;
| (None, _); y
let e = exp 1. in
let pi = 2. *. asin 1. in
(e, pi); (2.7182818284590451, 3.1415926535897931)
let uncurried (x, y) = x + y in
let curried x y = x + y in
(uncurried (1, 2), curried 1 2); (3, 3)
let rec sum (x : int list) : int =
match x with
[] -> 0
| u :: t -> u + sum t
module type STACK = sig
  type 'a stack
  exception Empty of string
  val make : unit -> 'a stack
  val push : 'a stack * 'a -> 'a stack
  val pop : 'a stack -> 'a * 'a stack
  val isEmpty : 'a stack -> bool
end
module Stack : STACK = struct
  type 'a stack = 'a list
  exception Empty of string
  let make () = []
  let push (s, x) = x :: s
  let pop s =
    match s with
    x :: t -> (x, t)
    | _ -> raise (Empty "empty")
  let isEmpty = fun x -> x = []
end
let xr : int ref = ref 2999 in
xr := !xr + 111; ();
sets xr to 3110 as a side effect
(print_endline "hello"; 3110); 3110
prints "hello" as a side effect
try Some (List.find (fun x -> x > 10); [1;5;10])
with Not_found -> None
raise Not_found
raise (Failure "error"
failwith "error"