```
(* The print function only takes strings as parameters *)
- print "hello\n";
hello
val it = () : unit


(* sequencing using ";" *)
- fun foo x = ( print "x is "; x)
= ;
val foo = fn : 'a -> 'a
- foo 7;
x is val it = 7 : int

(* converting an int to a string *)
- Int.toString 7;
val it = "7" : string

-fun foo x = (print "x is "; print (Int.toString x); print "\n");
val foo = fn : int -> unit
- foo 7;
x is 7
val it = () : unit

- (* nested defintions using let *)
- fun bar x =
=   let val z = x*2
=       fun g w = w + 2
=   in  g z
=   end;
val bar = fn : int -> int

(* If you want to return multiple values from
   a function, return a tuple of those values *)
- fun f x y = (x+1, y-1);
val f = fn : int -> int -> int * int
- f 5 8;
val it = (6,7) : int * int

(* Use tuple patterns to bind variables to the
   components of a tuple *)
- let val (a,b) = f 5 8
= in a+b
= end
= ;
val it = 13 : int

(* Declaring == as an infix operator *)
- infix ==
= ;
infix ==

(* Defining a function named ==, using infix
   notation *)
- fun [] == [] = true
=   |  (x::xs) == (y::ys) = x = y andalso xs == ys
=   |  _ == _ = false
= ;
stdIn:26.28 Warning: calling polyEqual
val == = fn : ''a list * ''a list -> bool
```

```
(* Ignore the above warning about polyEqual. *)

(* Calling == using infix notation *)
- [1,2,3] == [1,2,3];
val it = true : bool
- [1,2,3] == [1,2,4];
val it = false : bool
- [[1,2],[3,4]] == [[1,2],[3,4]];
val it = true : bool

(* Passing an infix operator as a parameter, using
   (op ...) *)
- fun f compare x y = compare (x,y);
val f = fn : ('a * 'b -> 'c) -> 'a -> 'b -> 'c
- f (op ==) [1,2] [1,2]
= ;
val it = true : bool

(* When declaring a parameter that you want to use
   as an infix operator, also use (op ...) *)
- fun q (op <) x y = if x < y then "yes" else "no";
val q = fn : ('a * 'b -> bool) -> 'a -> 'b -> string
- q (fn (a,b) => length a < length b) [1,2,3] [4,5]
= ;
val it = "no" : string


(* Declaring an exception *)
- exception e
= ;
exception e

(* Raising an exception *)
- fun head (x::xs) = x
=   |  head _ = raise e
= ;
val head = fn : 'a list -> 'a

(* Handling an exception *)
- fun foo L = (if head L = 7 then 3 else 0) handle e => 25;
val foo = fn : int list -> int
- foo [4,5,6];
val it = 0 : int
- foo [];
val it = 25 : int

(* Carrying a value along with an exception *)
- exception myexception of int;
exception myexception of int

(* Raising that exception *)
- (if 0 = 0 then raise (myexception 20) else 4) handle (myexception n) => n+1;
```