STANDARD ML

REFS

REF CELLS

use `ref` to create a single **mutable** cell

```
val x = ref 4;
```

`ref` is a constructor and as such is also a function

```
ref;
```

use `:=` to replace a cell's contents

```
val x = ref 0;
x := 15;
x;
```

note that `:=` returns `()`

```
val := = fn : 'a ref * 'a -> unit
```

use `!` to get the the cell's contents

```
val x = ref 8;
!x;
```

```
val ! = fn : 'a ref -> 'a
```

SEQUENCING WITH `;`

`;` is used to sequence expressions (with side-effects)

```
fun swap x y =
    (x := !x + !y ; y := !x - !y ; x := !x - !y);
```

an expression created by `;` evaluates to the value of the last expression

```
val x = ref 42;
(x := !x * !x; !x);
```

MEMOIZATION

```
type (''a, 'b) memoizer = {max_size: int, memory: (''a * 'b) list ref};

fun memoizer_put (memo: (''a, 'b) memoizer) x y =
    #memory(memo) :=
        (if length (!(#memory memo)) < #max_size memo
        then !(#memory memo)
        else tl (!(#memory memo)))
        @ [(x, y)];
```

```
fun memoize (memo: (''a, 'b) memoizer) f x =
    case (List.find (fn t => x = #1 t) (!(#memory memo))) of
      SOME (_, y) => y
    | NONE => (
        let val y = f x in
            memoizer_put memo x y;
            y
        end
    );
```

```
local
    val memo = {max_size=10, memory=ref []}
in
fun fib 0 = 0
  | fib 1 = 1
  | fib n = let
        val aux = memoize memo fib
    in
        (aux (n - 1)) + (aux (n - 2))
    end
end;
```

let's compare

```
fib 43;
```

```
fun fib_exp 0 = 0
  | fib_exp 1 = 1
  | fib_exp n = (fib_exp (n-1)) + (fib_exp (n-2));
fib_exp 43;
```