```
(*
                              CS51 Lab 12
                  Imperative Programming and References
 *)

(*
Objective:

This lab provides practice with reference types and their use in
building mutable data structures and in imperative programming more
generally. It also gives further practice in using modules to abstract
data types.

There are 4 total parts to this lab. Please refer to the following
files to complete all exercises:

-> lab12_part1.ml -- Part 1: Fun with references (this file)
   lab12_part2.ml -- Part 2: Gensym
   lab12_part3.ml -- Part 3: Appending mutable lists
   lab12_part4.ml -- Part 4: Adding serialization to imperative queues

*****************************************************************
                                IMPORTANT

As usual, when implementing functions in this lab, you shouldn't feel
beholden to how the definition is introduced in the skeleton code
below. For instance, you might want to add a `rec`, or use a different
argument list, or no argument list at all but binding to an anonymous
function instead. THIS IS ESPECIALLY PERTINENT TO CERTAIN PROBLEMS IN
THIS LAB, WHERE THE NORMAL COMPACT NOTATION FOR INTRODUCING FUNCTIONS
MAY NOT BE APPROPRIATE.
*****************************************************************)

(*================================================================
Part 1: Fun with references

................................................................
Exercise 1: Consider a function `incr` that takes an `int ref`
argument and has the side effect of incrementing the integer stored in
the `ref`. What is an appropriate type for the return value? What
should the type for the function as a whole be?
................................................................*)

(* Know the answer? Call over a staff member to check. If you're
working on this lab outside of lab times, check
<https://url.cs51.io/lab12-1> for our answer, which you should keep to
in the next exercise. *)

(*................................................................
Exercise 2: Now implement the `incr` function.
................................................................*)

let incr _ =
  failwith "incr not implemented" ;;

(*................................................................
Exercise 3: Write a function `remember` that returns the last string
that it was called with. The first time it is called, it should return
the empty string.

    # remember "don't forget" ;;
    - : string = ""
    # remember "what was that again?" ;;
    - : string = "don't forget"
    # remember "etaoin shrdlu" ;;
```

```
    - : string = "what was that again?"

(This is probably the least "functional" function ever written!)

As usual, you shouldn't feel beholden to how the definition is
introduced in the skeleton code below.
.....................................................................*)

let remember _ =
  failwith "remember not implemented" ;;
```