```
(*
                              CS51 Lab 7
                   Modules and Abstract Data Types
 *)

(* Objective: This lab practices concepts of modules, including files
as modules, signatures, and polymorphic abstract data types.

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

-> lab7_part1.ml -- Part 1: Implementing modules (this file)
   lab7_part2.ml -- Part 2: Files as modules
   lab7_part3.ml -- Part 3: Polymorphic abstract types and
                            Interfaces as abstraction barriers
 *)

(*======================================================================
Part 1: Implementing Modules

*Modules* are a way to package together and encapsulate types and values
(including functions) into a single discrete unit.

By applying a *signature* to a module, we guarantee that the module
implements at least the values and functions defined within it. The
module may also implement more as well, for internal use, but only those
specified in the signature will be exposed and available outside the
module definition. This form of abstraction, information hiding,
implements the edict of compartmentalization.

In this part, you'll revisit the "weather" example from lab 6 part
1. Recall that in that lab, you defined some algebraic data types for
representing aspects of the season and weather, along with some
functions to extract the precipitation amount and to generate a string
description of the weather. We can define a module signature for the
types and functions from that lab as follows:
 *)

module type WEATHER = sig
  type season = Spring | Summer | Autumn | Winter

  type condition =
    | Sunny
    | Rainy of int  (* precipitation in mm *)
    | Snowy of int  (* precipitation in mm *)

  type weather_status = {season : season; condition : condition}

  val describe_weather : weather_status -> string
  val precipitation_amount : condition -> int
end ;;

(* Notice that we've left out the function `season_to_string`, since
it was really just a helper function for `describe_weather`. There's
no reason that users of the module should need to use this
function. *)

(*......................................................................
Exercise 1A: Complete the implementation of a module called `Weather`
that satisfies the signature above. Feel free to make use of your
solution or the staff solution for lab 6 part 1. *)

(* (You may wonder, what's that `nan` in our dummy definition below? The
value `nan` stands for "not a number" and is an actual value of the
`float` type, as dictated by the IEEE Floating Point standard described
at <https://en.wikipedia.org/wiki/IEEE_754>. We're using it here as a
temporary value pending your putting in appropriate ones.) *)
(*......................................................................*)

module Weather : WEATHER = struct
  type season = Spring | Summer | Autumn | Winter

  type condition =
    | Sunny
    | Rainy of int
    | Snowy of int

  type weather_status = {season : season; condition : condition}

  let describe_weather (status : weather_status) : string =
    failwith "describe_weather not implemented"
  let precipitation_amount (condition : condition) : int =
    failwith "precipitation_amount not implemented"
end ;;

(*.......................................................................
Exercise 1B: Now that you've implemented the `Weather` module, use it
to generate a string description of a rainy winter day with 20 mm of
rain. That is, define a value `example : string` that uses
the `Weather` module to generate its string value

    "It's raining in winter. Precipitation: 20 mm."

(Use explicit module prefixes for this exercise, not global or local
opens.)
......................................................................*)

let example =
  "replace this string with an appropriate computation"  ;;

(*.......................................................................
Exercise 1C: Reimplement `example` from 1B above, now as
`example_local_open`, but using a "local open" to write your
computation in a more succinct manner.
......................................................................*)

let example_local_open () =
  "replace this string with an appropriate computation"  ;;
```