```
(*
                            CS51 Lab 15
                  Computing Pi Using Taylor Expansion

A Taylor series is a representation of a function as an infinite sum
of individual terms. For instance, the arctangent function can be
characterized as the following infinite sum:

    arctan x = x - x^3/3 + x^5/5 - x^7/7 + ...

Setting x to 1, we have

    arctan 1 = pi/4 = 1 - 1/3 + 1/5 - 1/7 + ...

so

    pi = 4 - 4/3 + 4/5 - 4/7 + ...

This gives us a simple way to approximate pi. Below, we generate a
stream for the terms of this formula and then compute sums of the
terms to approximate pi.

      ************************************************************
      YOU SHOULD NOT NEED TO CHANGE ANYTHING IN THIS FILE.

      NOTE THAT THIS FILE WILL NOT COMPILE UNTIL YOU HAVE COMPLETED
      THE `NativeLazyStreams` MODULE IN `nativeLazyStreams.ml`.
      ************************************************************
 *)

open NativeLazyStreams ;;

(* nats -- The natural numbers *)
let rec nats : int stream =
  lazy (Cons (0, smap ((+) 1) nats)) ;;

(* to_float s -- Converts a stream `s` of `int`s to the corresponding
   `float`s. *)
let to_float : int stream -> float stream =
  smap float_of_int ;;

(* odds -- A stream of odd numbers: 1, 3, 5, 7, ... *)
let odds : int stream =
  smap (fun x -> x * 2 + 1) nats ;;

(* neg_evens s -- Negates every other value in a stream `s`. Example:
      # first 5 (neg_evens nats) ;;
      - : int list = [0; 1; -2; 3; -4]
 *)
let neg_evens : int stream -> int stream =
  smap (fun x -> if x mod 2 = 0 then -x else x) ;;

(* alt_signs -- A stream of alternating 1s and -1s: 1, -1, 1, -1,
   ... *)
let alt_signs : int stream =
  smap2 ( / ) (neg_evens (tail nats)) (tail nats) ;;

(* pi_stream -- A stream of the terms of the Taylor expansion for
   pi: 4, -4/3, 4/5, -4/7, ... *)
let pi_stream : float stream =
  smap2 ( /. )
        (to_float (smap (( * ) 4) alt_signs))
        (to_float odds) ;;

(* pi_approx n -- The sum of the first `n` elements in the Taylor
```

```
   expansion for pi. *)
let pi_approx (n : int) : float =
  List.fold_left ( +. ) 0.0 (first n pi_stream) ;;

(* sums s -- A stream of the partial sums of the stream `s`. *)
let rec sums (s : float stream) : float stream =
  smap2 ( +. ) s (lazy (Cons (0.0, sums s))) ;;

(* pi_sums -- A stream of better and better approximations of pi. *)
let pi_sums : float stream =
  sums pi_stream ;;

(* within epsilon s -- Returns the index and the value of the first
   element in the stream `s` to be within `epsilon` of its following
   element. *)
let within (epsilon : float) (s : float stream) : int * float =
  let rec within' steps s =
    let h, t = head s, tail s in
    if abs_float (h -. (head t)) < epsilon then steps, h
    else within' (steps + 1) t in
  within' 0 s ;;

(* Here, we compute pi to three decimal places and print out the
   result. After uncommenting the last expression in this file, you
   can try it yourself with:

      % ocamlbuild pi.byte
      Finished, 5 targets (0 cached) in 00:00:00.
      % ./pi.byte
      Pi within 0.001000 is 3.141093 after 1999 steps

   Any more digits than this will take a long time. See the rest of
   the lab for speeding up the process. *)

let test () =
  let tolerance = 0.001 in
  let steps, approx = within tolerance pi_sums in
  Printf.printf "Pi within %f is %f after %d steps\n"
                tolerance approx steps ;;
(*
let _ = test () ;;
 *)
```