```
(* $Id: exponent.ml,v 330.4 2003-02-03 10:42:46-08 - - $ *)

(*
* Power function.
* power n a = a ** n
* Computed recursively and also tail-recursively.
* Runs in O(log2 n) time.  O(n) time is not acceptable.
* Note:  we put the power first so that it can be curried.
*)

(*
* Utility fns.
*)
let compose f g x = f (g x);;
let odd n         = n mod 2 <> 0;;
let even          = compose not odd;;
let swap fn x y   = fn y x;;

(*
* powerr - recursive version
*)
let rec powerr a n = match n with
    | 0            -> 1.
    | n when n < 0 -> powerr (1. /. a) (- n)
    | n when odd n -> a *. powerr a (n - 1)
    | n            -> powerr (a *. a) (n / 2)
    ;;

(*
* powert - more efficient tail recursive version
*)
let powert a n =
    let rec powert' a n result = match n with
        | 0            -> result
        | n when odd n -> powert' a (n - 1) (result *. a)
        | n            -> powert' (a *. a) (n / 2) result
    in  if n < 0 then powert' (1. /. a) (- n) 1.
                 else powert' a n 1.
    ;;

(*
* Make use of some of these functions by currying.
*)
let square = swap powert 2;;
let cube   = swap powert 3;;
let iiito  = powert 3.;;
let ivto   = powert 4.;;

let e      = 2.718281828459045235360287471352662497757247093;;
let eto    = powert e;;

```

```
bash-1$ ocaml
        OCaml version 4.02.1

# #use "exponent.ml";;
val compose : (’a -> ’b) -> (’c -> ’a) -> ’c -> ’b = <fun>
val odd : int -> bool = <fun>
val even : int -> bool = <fun>
val swap : (’a -> ’b -> ’c) -> ’b -> ’a -> ’c = <fun>
val powerr : float -> int -> float = <fun>
val powert : float -> int -> float = <fun>
val square : float -> float = <fun>
val cube : float -> float = <fun>
val iiito : int -> float = <fun>
val ivto : int -> float = <fun>
val e : float = 2.71828182845904509
val eto : int -> float = <fun>
# exit;;
- : int -> ’a = <fun>
# exit 0;;
```