```
(* Moduli e Tipi astratti *)

let rec gcd x y = if x=y then x else if x>y then gcd (x-y) y else gcd x (y-x);;
let decToString x = match x with
    0 ->"0"
    |1-> "1"
    |2-> "2"
    |3-> "3"
    |4-> "4"
    |5-> "5"
    |6-> "6"
    |7-> "7"
    |8-> "8"
    |9-> "9";;

let rec intToString x=if (x<10) then (decToString x) else
let r=x mod 10 in let v=x/10 in (intToString v)^(decToString r);;

module type RAZIONALE=
 sig type razionale
 val raz: int-> int-> razionale
 val num : razionale -> int
 val den : razionale -> int
 val reduce : razionale -> razionale
 val sum : razionale -> razionale -> razionale
 val equals : razionale -> razionale -> bool
 val toString : razionale -> string
 end;;

module Razionale =
(  struct
  type razionale= Razio of int * int
  let raz x y = Razio (x,y)
  let num (Razio (n,d))=n
  let den (Razio(n,d))= d
  let reduce (Razio (n,d))= let g=if n<0 then gcd (-n) d else gcd n d
                    in Razio(n/g,d/g)
  let sum (Razio(n1,d1)) (Razio(n2,d2))= reduce (Razio(n1*d2+n2*d1,d1*d2))
  let equals r1 r2= (reduce r1=reduce r2)
  let toString (Razio (n, d )) = (intToString n)^"/"^ (intToString d)
end : RAZIONALE) ;;

let v =Razionale.raz 3 4 in
let t =Razionale.raz 4 5 in Razionale.sum v t;;
```