

Objective Caml version 3.06

```
# let rec exp x n = if n = 0 then 1 else x * exp x (n-1);;
val exp : int -> int -> int = <fun>

# exp 3 10;;
- : int = 59049

# exp 3 1000;;
- : int = -742892767

# exp 3 100000;;
Stack overflow during evaluation (looping recursion?).

# let rec tail_exp x n y = if n = 0 then y else tail_exp x (n - 1) (x * y);;
val tail_exp : int -> int -> int -> int = <fun>

# let exp x n = tail_exp x n 1;;
val exp : int -> int -> int = <fun>

# let exp x n =
  let rec tail_exp x n y = if n = 0 then y else tail_exp x (n - 1) (x * y)
  in tail_exp x n 1;;
  val exp : int -> int -> int = <fun>

# let exp x n =
  let rec tail_exp_x n y = if n = 0 then y else tail_exp_x (n - 1) (x * y)
  in tail_exp_x n 1;;
  val exp : int -> int -> int = <fun>

# exp 3 10;;
- : int = 59049

# exp 3 1000;;
- : int = -742892767

# exp 3 100000;;
- : int = -863145855

# exp 3 10000000;;
- : int = -902467071

# let rec fast_exp x n =
  if n = 0
  then 1
  else if n mod 2 = 0 then fast_exp (x * x) (n/2) else x * fast_exp (x * x) (n/2);;
  val fast_exp : int -> int -> int = <fun>

# let rec fast_exp x n =
  if n = 0 then 1 else let a = fast_exp (x * x) (n/2) in if n mod 2 = 0 then a else x * a;;
  val fast_exp : int -> int -> int = <fun>

# let rec fast_exp x n =
  if n = 0 then 1 else (if n mod 2 = 0 then 1 else x) * fast_exp (x * x) (n/2);;
  val fast_exp : int -> int -> int = <fun>

# fast_exp 3 10;;
- : int = 59049

# fast_exp 3 1000;;
- : int = -742892767

# fast_exp 3 100000;;
- : int = -863145855

# fast_exp 3 10000000;;
- : int = -902467071

# let rec fast_exp x n =
  let rec tail_fast_exp x n y =
    if n = 0 then y else tail_fast_exp (x * x) (n/2) (if n mod 2 = 0 then y else x * y)
  in tail_fast_exp x n 1;;
  val fast_exp : int -> int -> int = <fun>

# fast_exp 3 10;;
- : int = 59049

# fast_exp 3 1000;;
```



```

# exists 0 100 (fun x -> x = x + 1);;
- : bool = false

# exists 0 100 (fun x -> x mod 15 = 7 && x mod 14 = 9);;
- : bool = true

# let rec forall m n p = m > n || (p m && forall (m + 1) n p);;
val forall : int -> int -> (int -> bool) -> bool = <fun>

# forall 0 100 (fun x -> x < x + 1);;
- : bool = true

# forall 0 100 (fun x -> x * x < 5000);;
- : bool = false

# exception Empty;;
exception Empty

# let rec min m n p = if m > n then raise Empty else if p m then m else min (m + 1) n p;;
val min : int -> int -> (int -> bool) -> int = <fun>

# min 0 100 (fun x -> x mod 15 = 7 && x mod 14 = 9);;
- : int = 37

# min 0 100 (fun x -> not (x * x < 5000));;
- : int = 71

# let ferat_test p = forall 1 (p - 1) (fun x -> ferat x p = x);;
val ferat_test : int -> bool = <fun>

# ferat_test 101;;
- : bool = true

# ferat_test 100;;
- : bool = false

# let ferat_witness p = min 1 (p - 1) (fun x -> ferat x p <> x);;
val ferat_witness : int -> int = <fun>

# ferat_witness 101;;
Exception: Empty.

# ferat_witness 100;;
- : int = 2

# ferat_witness 341;;
- : int = 3

# ferat 2 341;;
- : int = 2

# ferat 3 341;;
- : int = 168

# let verbose_ferat_test p =
  try (let x = ferat_witness p
        in print_string (string_of_int p ^ " ist keine Primzahl. " ^
                        "Zeuge: exp " ^ string_of_int x ^ " " ^ string_of_int p ^ " mod " ^
                        string_of_int p ^ " = " ^ string_of_int (ferat x p) ^ "\n"))
  with Empty -> (print_string (string_of_int p ^
                              " hat den Fermat-Test bestanden, koennte also eine Primzahl sei
n.\n"));;
  val verbose_ferat_test : int -> unit = <fun>

# verbose_ferat_test 100;;
100 ist keine Primzahl. Zeuge: exp 2 100 mod 100 = 76
- : unit = ()

# verbose_ferat_test 341;;
341 ist keine Primzahl. Zeuge: exp 3 341 mod 341 = 168
- : unit = ()

# verbose_ferat_test 101;;
101 hat den Fermat-Test bestanden, koennte also eine Primzahl sein.
- : unit = ()

#

```