

Objective Caml version 3.06

```
# type 'a stream =
  Nil
  | Cons of 'a * (unit -> 'a stream);;
  type 'a stream = Nil | Cons of 'a * (unit -> 'a stream)

# let head (Cons (x, _) = x);;
Characters 9-26:
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
Nil
  let head (Cons (x, _) = x);;
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val head : 'a stream -> 'a = <fun>

# let tail (Cons (_, s) = s ());;
Characters 9-29:
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
Nil
  let tail (Cons (_, s) = s ());;
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
val tail : 'a stream -> 'a stream = <fun>

# let cons (x, s) = Cons (x, fun () -> s);;
val cons : 'a * 'a stream -> 'a stream = <fun>

# let rec nats_from (k: int) =
  cons (k, nats_from (k + 1));;
  val nats_from : int -> int stream = <fun>

# let nats = nats_from 0;;
Stack overflow during evaluation (looping recursion?).

# let rec nats_from (k: int) =
  Cons (k, fun () -> nats_from (k + 1));;
  val nats_from : int -> int stream = <fun>

# let nats = nats_from 0;;
val nats : int stream = Cons (0, <fun>)

# head nats;;
- : int = 0

# tail nats;;
- : int stream = Cons (1, <fun>)

# head (tail (tail nats));;
- : int = 2

# let rec take =
  function
    (0, _) -> []
  | (_, Nil) -> []
  | (n, Cons (x, s)) -> x :: take (n - 1, s ());;
  val take : int * 'a stream -> 'a list = <fun>

# take (20, nats);;
- : int list =
[0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19]

# let rec square_stream =
  function
    Nil -> Nil
  | Cons (x, s) -> Cons (x * x, square_stream s);;
  Characters 101-102:
  | Cons (x, s) -> Cons (x * x, square_stream s);;
      ^
This expression has type unit -> int stream but is here used with type
int stream

# let rec square_stream =
  function
    Nil -> Nil
  | Cons (x, s) -> Cons (x * x, function () -> square_stream (s ()));;
  val square_stream : int stream -> int stream = <fun>
```

```

# let squares = square_stream nats;;
val squares : int stream = Cons (0, <fun>)

# take (20, squares);;
- : int list =
[0; 1; 4; 9; 16; 25; 36; 49; 64; 81; 100; 121; 144; 169; 196; 225; 256; 289;
 324; 361]

# let rec add_streams =
  function
    (Cons (x, xs), Cons (y, ys)) -> Cons (x + y, function () -> add_streams (xs (), ys ()))
  | _ -> Nil;;
  val add_streams : int stream * int stream -> int stream = <fun>

# let s = add_streams (nats, squares);;
val s : int stream = Cons (0, <fun>)

# take (20, s);;
- : int list =
[0; 2; 6; 12; 20; 30; 42; 56; 72; 90; 110; 132; 156; 182; 210; 240; 272; 306;
 342; 380]

# let rec append_streams =
  function
    (Nil, s) -> s
  | (Cons (x, xs), s) -> Cons (x, function () -> append_streams (xs (), s));;
  val append_streams : 'a stream * 'a stream -> 'a stream = <fun>

# let s = append_streams (nats, squares);;
val s : int stream = Cons (0, <fun>)

# take (20, s);;
- : int list =
[0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17; 18; 19]

# let finite_stream = cons (5, cons (3, Nil));;
val finite_stream : int stream = Cons (5, <fun>)

# let s = append_streams (finite_stream, nats);;
val s : int stream = Cons (5, <fun>)

# take (20, s);;
- : int list =
[5; 3; 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17]

# let rec map_stream f s =
  match s with
  Nil -> Nil
  | Cons (x, xs) -> Cons (f x, fun () -> map_stream f (xs ()));;
  val map_stream : ('a -> 'b) -> 'a stream -> 'b stream = <fun>

# let squares = map_stream (fun x -> x * x) nats;;
val squares : int stream = Cons (0, <fun>)

# take (20, squares);;
- : int list =
[0; 1; 4; 9; 16; 25; 36; 49; 64; 81; 100; 121; 144; 169; 196; 225; 256; 289;
 324; 361]

# let rec filter_stream p s =
  match s with
  Nil -> Nil
  | Cons (x, xs) ->
    if p x
    then Cons (x, fun () -> filter_stream p (xs ()))
    else filter_stream p (xs ());;
  val filter_stream : ('a -> bool) -> 'a stream -> 'a stream = <fun>

# let s = filter_stream (fun x -> x mod 2 = 1) squares;;
val s : int stream = Cons (1, <fun>)

# take (20, s);;
- : int list =
[1; 9; 25; 49; 81; 121; 169; 225; 289; 361; 441; 529; 625; 729; 841; 961;
 1089; 1225; 1369; 1521]

# let sieve_by m = filter_stream (fun n -> n mod m <> 0);;
val sieve_by : int -> int stream -> int stream = <fun>

```

```
# let rec sieve_by_all =
function
  Cons (m, s) -> Cons (m, fun () -> sieve_by_all (sieve_by m (s ())));;
  Characters 25-108:
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
Nil
  ..function
    Cons (m, s) -> Cons (m, fun () -> sieve_by_all (sieve_by m (s ())))..
val sieve_by_all : int stream -> int stream = <fun>

# let primes = sieve_by_all (nats_from 2);;
val primes : int stream = Cons (2, <fun>)

# take (100, primes);;
- : int list =
[2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43; 47; 53; 59; 61; 67; 71;
 73; 79; 83; 89; 97; 101; 103; 107; 109; 113; 127; 131; 137; 139; 149; 151;
 157; 163; 167; 173; 179; 181; 191; 193; 197; 199; 211; 223; 227; 229; 233;
 239; 241; 251; 257; 263; 269; 271; 277; 281; 283; 293; 307; 311; 313; 317;
 331; 337; 347; 349; 353; 359; 367; 373; 379; 383; 389; 397; 401; 409; 419;
 421; 431; 433; 439; 443; 449; 457; 461; 463; 467; 479; 487; 491; 499; 503;
 509; 521; 523; 541]

#
```