```
Last login: Fri Mar 31 13:11:18 on ttys015
carbon:SamplePrograms$ cd Sec_01_1\:25pm/
carbon:Sec_01_1:25pm$ utop
─────────────┬──────────────────────────────────────────────────────────┬─────────────
             │ Welcome to utop version 1.14 (using OCaml version 4.01.0)! │
             └──────────────────────────────────────────────────────────┘

Type #utop_help for help about using utop.

─( 18:00:00 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool                                          t
ype expr =
  Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
─( 13:46:29 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # eval ;;
- : expr -> environment -> value = <fun>                                        ─
( 13:46:37 )─< command 2 >──────────────────────────────────────{ counter: 0 }─ut
op # eval (Add (Value (Int 1), Value (Int 3))) ;;
- : environment -> value = <fun>                                                ─
( 13:46:45 )─< command 3 >──────────────────────────────────────{ counter: 0 }─ut
op # eval (Add (Value (Int 1), Value (Int 3)))  []  ;;
- : value = Int 4                                                               ─
( 13:47:03 )─< command 4 >──────────────────────────────────────{ counter: 0 }─ut
op # eval (Add (Value (Int 1), Var "x"))  [("x", Int 4)]  ;;
- : value = Int 5                                                               ─
( 13:47:17 )─< command 5 >──────────────────────────────────────{ counter: 0 }─ut
op # eval (Add (Value (Int 1), Var "x")) ;;
- : environment -> value = <fun>                                                ─
( 13:47:33 )─< command 6 >──────────────────────────────────────{ counter: 0 }─ut
op # #use "interpreter.ml";;
type value = Int of int | Bool of bool                                          t
ype expr =
  Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
type environment = (string * value) list
```

```
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | While of expr * stmt
  | IfThen of expr * stmt
  | Seq of stmt * stmt
  | WriteNum of expr
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 1)),
   Seq (Assign ("y", Add (Var "x", Value (Int 2))),
    Seq (Assign ("z", Add (Var "y", Value (Int 3))), WriteNum (Var "z"))))
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
─( 13:47:42 )─< command 7 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool                                          t
ype expr =
  Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | While of expr * stmt
  | IfThen of expr * stmt
  | Seq of stmt * stmt
  | WriteNum of expr
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 1)),
   Seq (Assign ("y", Add (Var "x", Value (Int 2))),
    Seq (Assign ("z", Add (Var "y", Value (Int 3))), WriteNum (Var "z"))))
File "interpreter.ml", line 93, characters 36-100:
Error: This expression has type 'a * 'b but an expression was expected of type
         state = (string * value) list
─( 14:01:06 )─< command 8 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool                                          t
ype expr =
  Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | While of expr * stmt
  | IfThen of expr * stmt
  | Seq of stmt * stmt
  | WriteNum of expr
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 1)),
   Seq (Assign ("y", Add (Var "x", Value (Int 2))),
    Seq (Assign ("z", Add (Var "y", Value (Int 3))), WriteNum (Var "z"))))
File "interpreter.ml", line 94, characters 2-61:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
(While (_, _)|IfThen (_, _)|Seq (_, _)|WriteNum _)
val exec : stmt -> state -> state = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
─( 14:05:05 )─< command 9 >──────────────────────────────────{ counter: 0 }─
utop # exec (Assign ("x", Add (Value (Int 4), Var "y"))) [ ("y", Int 5) ] ;;
- : state = [("x", Int 9); ("y", Int 5)]                                    ─
( 14:05:22 )─< command 10 >─────────────────────────────────{ counter: 0 }─ut
op # exec (Assign ("x", Add (Value (Int 4), Var "y"))) [ ("x", Int 9) ; ("y", Int
 5) ] ;;
- : state = [("x", Int 9); ("x", Int 9); ("y", Int 5)]                      ─
( 14:06:36 )─< command 11 >─────────────────────────────────{ counter: 0 }─ut
op # exec (Assign ("x", Add (Value (Int 4), Var "x"))) [ ("x", Int 9) ; ("y", Int
 5) ] ;;
- : state = [("x", Int 13); ("x", Int 9); ("y", Int 5)]                     ─
( 14:06:59 )─< command 12 >─────────────────────────────────{ counter: 0 }─ut
op # lookup "x" (exec (Assign ("x", Add (Value (Int 4), Var "x"))) [ ("x", Int 9)
 ; ("y", Int 5) ])  ;;
- : value = Int 13                                                          ─
( 14:07:13 )─< command 13 >─────────────────────────────────{ counter: 0 }─ut
op # #use "interpreter.ml";;
type value = Int of int | Bool of bool                                      t
ype expr =
  Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | While of expr * stmt
  | IfThen of expr * stmt
  | Seq of stmt * stmt
  | WriteNum of expr
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 1)),
   Seq (Assign ("y", Add (Var "x", Value (Int 2))),
    Seq (Assign ("z", Add (Var "y", Value (Int 3))), WriteNum (Var "z"))))
File "interpreter.ml", line 94, characters 2-104:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
(While (_, _)|IfThen (_, _)|WriteNum _)
val exec : stmt -> state -> state = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
-( 14:07:33 )-< command 14 >-------------------------------------{ counter: 0 }-
utop # exec ( Seq (Assign ("x", Value (Int 4)), Assign ("x", Add (Var "x", Value
(Int 5))))) [] ;;
-( 14:11:43 )-< command 15 >--------------------------------------{ counter: 0 }-
utop #
┌───┬───┬───┬────────────┬─────┬───────────┬──────────────┬──────┬───────┬──────┐
│Add│And│Arg│Arith_status│Array│ArrayLabels│Assert_failure│Assign│Big_int│Bigarr│
└───┴───┴───┴────────────┴─────┴───────────┴──────────────┴──────┴───────┴──────┘
└───┴───┴───┴────────────┴─────┴───────────┴──────────────┴──────┴───────┴─────┘
```