```
Last login: Fri Apr  6 13:17:39 on ttys008
carbon:$ utop
────────┬─────────────────────────────────────────────────────────────┬────
        │ Welcome to utop version 2.0.2 (using OCaml version 4.06.0)! │
        └─────────────────────────────────────────────────────────────┘

Type #utop_help for help about using utop.

─( 13:27:44 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop # #quit;;
carbon:$ pwd
/project/evw/Teaching/18_Spring_2041/carbon-repos
carbon:$ cd public-class-repo/Sample\ Programs/Sec_01_1-25pm/Search/
carbon:$ utop
────────┬─────────────────────────────────────────────────────────────┬────
        │ Welcome to utop version 2.0.2 (using OCaml version 4.06.0)! │
        └─────────────────────────────────────────────────────────────┘

Type #utop_help for help about using utop.

─( 13:28:07 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> unit = <fun>
exception FoundPath of (loc * loc * loc * loc) list
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak1 * '_weak2 * '_weak3 * '_weak4) list -> string = <fun>
File "wolf.ml", line 157, characters 22-26:
Error: This variant expression is expected to have type unit
       The constructor None does not belong to type unit
─( 13:28:07 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> unit = <fun>
exception FoundPath of (loc * loc * loc * loc) list
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak5 * '_weak6 * '_weak7 * '_weak8) list -> string = <fun>
─( 13:28:10 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # moves (L,L,L,L) ;;
- : state list = [(R, L, R, L)]
─( 13:28:41 )─< command 3 >──────────────────────────────────────{ counter: 0 }─
utop # moves (R,L,R,L) ;;
- : state list = [(L, L, R, L); (L, L, L, L)]
─( 13:28:52 )─< command 4 >──────────────────────────────────────{ counter: 0 }─
utop # List.mem ;;
- : 'a -> 'a list -> bool = <fun>
─( 13:29:10 )─< command 5 >──────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak9 * '_weak10 * '_weak11 * '_weak12) list -> string =
  <fun>
─( 13:34:37 )─< command 6 >──────────────────────────────────────{ counter: 0 }─
utop # crossing_v1 () ;;
- : state list option =
Some
 [(L, L, L, L); (R, L, R, L); (L, L, R, L); (R, R, R, L); (L, R, L, L);
  (R, R, L, R); (L, R, L, R); (R, R, R, R)]
─( 13:38:51 )─< command 7 >──────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
```

```
File "wolf.ml", line 105, characters 6-346:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak13 * '_weak14 * '_weak15 * '_weak16) list -> string =
  <fun>
─( 13:39:00 )─< command 8 >──────────────────────────────────────{ counter: 0 }─
utop # crossing_v2 () ;;
Exception:
FoundPath
 [(L, L, L, L); (R, L, R, L); (L, L, R, L); (R, R, R, L); (L, R, L, L);
  (R, R, L, R); (L, R, L, R); (R, R, R, R)].
─( 13:44:26 )─< command 9 >──────────────────────────────────────{ counter: 0 }─
utop # List.fold_left ;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
─( 13:44:30 )─< command 10 >─────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 105, characters 6-346:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak17 * '_weak18 * '_weak19 * '_weak20) list -> string =
  <fun>
─( 13:49:01 )─< command 11 >─────────────────────────────────────{ counter: 0 }─
utop # crossing_many_possible_moves () ;;
- : unit = ()
─( 13:52:10 )─< command 12 >─────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
File "wolf.ml", line 149, characters 0-3:
```

```
Error: Syntax error
─( 13:52:25 )─< command 13 >────────────────────────────────────{ counter: 0 }─
utop # #use "wolf.ml";;
val is_not_elem : 'a list -> 'a -> bool = <fun>
type loc = L | R
type state = loc * loc * loc * loc
val ok_state : state -> bool = <fun>
val final : loc * loc * loc * loc -> bool = <fun>
val other_side : loc -> loc = <fun>
val moves : state -> state list = <fun>
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 105, characters 6-346:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
val crossing_v2 : unit -> unit = <fun>
val crossing_many_possible_moves : unit -> unit = <fun>
val crossing_many_possible_moves' : unit -> unit = <fun>
exception KeepLooking
val process_solution_exn : ('a -> string) -> 'a -> 'a option = <fun>
val show_list : ('a -> string) -> 'a list -> string = <fun>
val show_loc : 'a -> string = <fun>
val show_state : 'a * 'b * 'c * 'd -> string = <fun>
val show_path : ('_weak21 * '_weak22 * '_weak23 * '_weak24) list -> string =
  <fun>
─( 13:54:16 )─< command 14 >────────────────────────────────────{ counter: 0 }─
utop # crossing_many_possible_moves () ;;
Exception:
FoundPath
 [(L, L, L, L); (R, L, R, L); (L, L, R, L); (R, R, R, L); (L, R, L, L);
  (R, R, L, R); (L, R, L, R); (R, R, R, R)].
─( 13:54:27 )─< command 15 >────────────────────────────────────{ counter: 0 }─
utop # #quit ;;
carbon:$
```