```
Last login: Fri Apr  6 15:35:27 on ttys008
carbon:$ utop
──────────────────────────────────────────────────────────────────────────────
come to utop version 2.0.2 (using OCaml version 4.06.
──────────────────────────────────────────────────────────────────────────────

Type #utop_help for help about using utop.

─( 15:46:50 )─< command 0 >──────────{ counter: 0 }─
utop # #quit;;
carbon:$ cd Search/
carbon:$ utop
──────────────────────────────────────────────────────────────────────────────
come to utop version 2.0.2 (using OCaml version 4.06.
──────────────────────────────────────────────────────────────────────────────

Type #utop_help for help about using utop.

─( 15:47:01 )─< 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>
File "wolf.ml", line 83, characters 6-270:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 104, characters 6-360:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
File "wolf.ml", line 114, characters 9-17:
Error: This function has type
         state -> state list -> unit
       It is applied to too many arguments;
       maybe you forgot a `;'.
─( 15:47:01 )─< 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>
File "wolf.ml", line 83, characters 6-270:
```

```
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 104, characters 6-360:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
val crossing_v2 :
  unit -> (loc * loc * loc * loc) list option =
  <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>
-( 15:47:05 )-< command 2 >----------{ counter: 0 }-
utop # crossing_v2 () ;;
- : (loc * loc * loc * loc) 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)]
-( 15:47:17 )-< command 3 >----------{ 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>
File "wolf.ml", line 83, characters 6-270:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 104, characters 6-360:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
```

```
val crossing_v2 :
  unit -> (loc * loc * loc * loc) list option =
  <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 : ('a * 'b * 'c * 'd) list -> string =
  <fun>
-( 15:47:23 )-< command 4 >-------------{ counter: 0 }-
utop # crossing_v2 () ;;
- : (loc * loc * loc * loc) 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)]
-( 15:49:02 )-< command 5 >-------------{ counter: 0 }-
utop # List.fold_left ;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
-( 15:50:15 )-< command 6 >-------------{ counter: 0 }-
utop # #use "wolf.ml";;
File "wolf.ml", line 133, characters 21-22:
Error: Syntax error
-( 15:53:19 )-< 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>
File "wolf.ml", line 83, characters 6-270:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_
val crossing_v1 : unit -> state list option = <fun>
exception FoundPath of (loc * loc * loc * loc) list
File "wolf.ml", line 104, characters 6-360:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
_::_::_::_::_
val crossing_v2 :
  unit -> (loc * loc * loc * loc) list option =
  <fun>
```

```
val crossing_many_possible_moves :
  unit -> (loc * loc * loc * loc) list option =
  <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 : ('a * 'b * 'c * 'd) list -> string =
  <fun>
─( 15:57:30 )─< command 8 >──────────{ counter: 0 }─
utop # crossing_many_possible_moves () ;;
- : (loc * loc * loc * loc) 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)]
─( 15:57:43 )─< command 9 >──────────{ counter: 0 }─
utop # List.iter ;;
- : ('a -> unit) -> 'a list -> unit = <fun>
─( 15:57:52 )─< command 10 >──────( ─( 1──(──(─────(─────────(─( ─( 15─( 15:─( 1
──( 15─( ──(────( 15:58:43 )─< command 10 >──────────────────────────────────
─────────────{ counter: 0 ─( 15:58:43 )─< command 10 >──────────────────────
──────────────────────────{ counter: 0─( 15:58:43 )─< command 10 >──────────
──────────────────────────────────────{ counter: ─( 15:58:43 )─< command 10 >──
──────────────────────────────────────────{ counter:─( 15:58:43 )─< command
10 >──────────────────────────────────────{ counter─( 15:58:43 )─< com
mand 10 >─────────────────────────────────{ counte─( 15:58:43 )─<
command 10 >──────────────────────────────{ count─( 15:58:43 )─<
 command 10 >─────────────────────────────{ coun─( 15:58:43 )─<
command 10 >──────────────────────────────{ cou─( 15:58:43 )─< com
mand 10 >─────────────────────────────────{ co─( 15:58:43 )─< command
10 >──────────────────────────────────────{ c─( 15:58:43 )─< command 10 >──
──────────────────────────────────────{ ─( 15:58:43 )─< command 10 >──────────
──────────────────────{─( 15:58:43 )─< command 10 >──────────────────────
─────────────( 15:58:43 )─< command 10 >──────────────────────────────────
─( 15:58:43 )─< command 10 >──────────────────────────( 15:58:43 )─< c
─( 15:58:43 )─< command 10 >──────────────────────────{ counter: 0 }─
utop #
```

Arg	Array	ArrayLabels	Assert_failure	Bigarray	Buffer	Bytes	BytesLabels	Callba