```
Last login: Wed Feb 14 13:23:25 on ttys004
carbon:$ utop
─────────────────────────────────────────────────────────────
Welcome to utop version 2.0.2 (using OCaml version 4.06.0)
─────────────────────────────────────────────────────────────

Type #utop_help for help about using utop.

─( 15:34:56 )─< command 0 >────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
─( 15:34:56 )─< command 1 >────────────────{ counter: 0 }─
utop # Nothing ;;
- : 'a maybe = Nothing
─( 15:44:31 )─< command 2 >────────────────{ counter: 0 }─
utop # Just 6 ;;
- : int maybe = Just 6
─( 15:44:36 )─< command 3 >────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
-( 15:44:51 )-< command 4 >────────────────{ counter: 0 }-
utop # divide 5 2 ;;
- : int maybe = Just 2
-( 15:46:12 )-< command 5 >────────────────{ counter: 0 }-
utop # divide 56 2 ;;
- : int maybe = Just 28
-( 15:46:17 )-< command 6 >────────────────{ counter: 0 }-
utop # divide 56 0 ;;
- : int maybe = Nothing
-( 15:46:23 )-< command 7 >────────────────{ counter: 0 }-
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
─( 15:46:26 )─< command 8 >────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
```

```
─( 15:50:10 )─< command 9 >──────────────────{ counter: 0 }─
utop # (3, Nil) ;;
- : int * 'a myList = (3, Nil)
─( 15:54:03 )─< command 10 >─────────────────{ counter: 0 }─
utop # [] ;;
- : 'a list = []
─( 15:54:41 )─< command 11 >─────────────────{ counter: 0 }─
utop # Cons (3, Nil) ;;
- : int myList = Cons (3, Nil)
─( 15:54:53 )─< command 12 >─────────────────{ counter: 0 }─
utop # 1::[];;
- : int list = [1]
─( 15:55:04 )─< command 13 >─────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
```

```
─( 15:56:37 )─< command 14 >────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
─( 15:59:05 )─< command 15 >────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
-( 16:00:25 )-< command 16 >───────────────{ counter: 0 }-
utop # sumList l2 ;;
- : int = 6
-( 16:03:59 )-< command 17 >───────────────{ counter: 0 }-
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
```

```
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Noe of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Noe (1, Noe (2, Empty, Empty), Noe (3, Empty, Empty))
─( 16:04:05 )─< command 18 >───────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
```

```
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Node (1, Node (2, Empty, Empty),
   Node (3, Empty, Empty))
val sumTree : int btree -> int = <fun>
─( 16:08:13 )─< command 19 >───────────────{ counter: 0 }─
utop # sumTree treeAmin ;;
- : int = 6
─( 16:11:37 )─< command 20 >───────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Node (1, Node (2, Empty, Empty),
   Node (3, Empty, Empty))
val sumTree : int btree -> int = <fun>
val tstr : string btree =
  Node ("a", Node ("Hello", Empty, Empty),
   Node ("Why?", Empty, Empty))
val concatTree : string btree -> string = <fun>
─( 16:11:44 )─< command 21 >───────────────{ counter: 0 }─
utop # concatTree tstr ;;
- : string = "HelloaWhy?"
─( 16:15:22 )─< command 22 >───────────────{ counter: 0 }─
utop # List.map string_of_int [1;2;3]  ;;
- : string list = ["1"; "2"; "3"]
─( 16:15:27 )─< command 23 >───────────────{ counter: 0 }─
utop # treeAmin
;;
- : int btree =
Node (1, Node (2, Empty, Empty), Node (3, Empty, Empty))
─( 16:16:27 )─< command 24 >───────────────{ counter: 0 }─
utop # treeMap string_of_int treeAmin ;;
Error: Unbound value treeMap
─( 16:18:31 )─< command 25 >───────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Node (1, Node (2, Empty, Empty),
   Node (3, Empty, Empty))
val sumTree : int btree -> int = <fun>
val tstr : string btree =
  Node ("a", Node ("Hello", Empty, Empty),
   Node ("Why?", Empty, Empty))
val concatTree : string btree -> string = <fun>
val treeMap : ('a -> 'b) -> 'a btree -> 'b btree = <fun>
─( 16:18:43 )─< command 26 >───────────────{ counter: 0 }─
utop # treeMap string_of_int treeAmin ;;
- : string btree =
Node ("1", Node ("2", Empty, Empty),
 Node ("3", Empty, Empty))
─( 16:18:48 )─< command 27 >───────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Node (1, Node (2, Empty, Empty),
   Node (3, Empty, Empty))
val sumTree : int btree -> int = <fun>
val tstr : string btree =
  Node ("a", Node ("Hello", Empty, Empty),
   Node ("Why?", Empty, Empty))
val concatTree : string btree -> string = <fun>
val treeMap : ('a -> 'b) -> 'a btree -> 'b btree = <fun>
val treeFold :
  ('a ->
   ('a btree -> 'a btree) ->
   ('a btree -> 'a btree) -> 'a btree) ->
  'a btree -> 'a btree -> 'a btree = <fun>
─( 16:18:50 )─< command 28 >────────────────{ counter: 0 }─
utop # treeFold (fun a b c -> a + b + c) 0 treeAmin ;;
Error: This expression has type int btree -> int btree
       but an expression was expected of type int
```

```
─( 16:25:06 )─< command 29 >──────────────────{ counter: 0 }─
utop # #use "inductive.ml";;
type color = Red | Green | Blue
type weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
type boolean = True | False
val isRed : color -> bool = <fun>
val isWorkday : weekday -> bool = <fun>
type intorstr = Int of int | Str of string
type coord = float * float
type circ_desc = coord * float
type tri_desc = coord * coord * coord
type sqr_desc = coord * coord * coord * coord
type shape =
    Circ of circ_desc
  | Tri of tri_desc
  | Sqr of sqr_desc
val isRect : shape -> bool = <fun>
File "inductive.ml", line 35, characters 2-74:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Sqr _
val area : shape -> float = <fun>
type 'a maybe = Nothing | Just of 'a
val divide : int -> int -> int maybe = <fun>
val listHd : 'a list -> 'a option = <fun>
type 'a myList = Nil | Cons of 'a * 'a myList
type intList = IntNil | IntCons of int * intList
type ('a, 'b) dictionary = ('a * 'b) list
val l1 : 'a myList = Nil
val l2 : int myList = Cons (1, Cons (2, Cons (3, Nil)))
val sumList : int myList -> int = <fun>
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
val treeAmin : int btree =
  Node (1, Node (2, Empty, Empty),
   Node (3, Empty, Empty))
val sumTree : int btree -> int = <fun>
val tstr : string btree =
  Node ("a", Node ("Hello", Empty, Empty),
   Node ("Why?", Empty, Empty))
val concatTree : string btree -> string = <fun>
val treeMap : ('a -> 'b) -> 'a btree -> 'b btree = <fun>
val treeFold :
  ('a -> 'b -> 'b -> 'b) -> 'b -> 'a btree -> 'b = <fun>
─( 16:25:34 )─< command 30 >───────────────{ counter: 0 }─
utop # treeFold (fun a b c -> a + b + c) 0 treeAmin ;;
- : int = 6
─( 16:25:51 )─< command 31 >───────────────{ counter: 0 }─
utop # treeAmin
;;
- : int btree =
Node (1, Node (2, Empty, Empty), Node (3, Empty, Empty))
─( 16:25:53 )─< command 32 >───────────────{ counter: 0 }─
utop #

 │Arg│Array│ArrayLabels│Assert_failure│Bigarray│Blue│Buffe│
```