```
Last login: Fri Feb 16 13:23:43 on ttys003
carbon:$ utop

lcome to utop version 2.0.2 (using OCaml version 4.06.


Type #utop_help for help about using utop.

─( 15:55:44 )─< 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
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>
val add3 : int -> int -> int -> int = <fun>
val a_sum : int = 6
val treeFoldLeft :
  ('a -> 'b -> 'b) -> 'b -> 'a btree -> 'b = <fun>
─( 15:55:44 )─< command 1 >────────────{ 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>
val add3 : int -> int -> int -> int = <fun>
val a_sum : int = 6
val treeFoldLeft :
  ('a -> 'b -> 'b) -> 'b -> 'a btree -> 'b = <fun>
val sumLeft : int btree -> int = <fun>
─( 15:55:49 )─< command 2 >────────────{ counter: 0 }─
utop # sumLeft treeAmin ;;
- : int = 6
─( 15:57:50 )─< 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>
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>
val add3 : int -> int -> int -> int = <fun>
val a_sum : int = 6
val treeFoldLeft :
  ('a -> 'b -> 'b) -> 'b -> 'a btree -> 'b = <fun>
val sumLeft : int btree -> int = <fun>
val toInorderList : 'a btree -> 'a list = <fun>
─( 15:57:56 )─< command 4 >──────────────{ counter: 0 }─
utop # toInorderList treeAmin ;;
- : int list = [3; 2; 1]
─( 16:00:01 )─< command 5 >──────────────{ 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>
val add3 : int -> int -> int -> int = <fun>
val a_sum : int = 6
val treeFoldLeft :
  ('a -> 'b -> 'b) -> 'b -> 'a btree -> 'b = <fun>
val sumLeft : int btree -> int = <fun>
val toInorderList : 'a btree -> 'a list = <fun>
─( 16:00:23 )─< command 6 >────────────{ counter: 0 }─
utop # toInorderList treeAmin ;;
- : int list = [1; 2; 3]
─( 16:00:53 )─< command 7 >────────────{ counter: 0 }─
utop #
┌───┬─────┬───────────┬──────────────┬────────┬────┬─┐
│Arg│Array│ArrayLabels│Assert_failure│Bigarray│Blue│B│
└───┴─────┴───────────┴──────────────┴────────┴────┴─┘
```