

Objective Caml version 3.06

```
# type bintree =
  Leaf of int
  | Node of bintree * bintree;;
type bintree = Leaf of int | Node of bintree * bintree

# let t: bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3);;
val t : bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3)

# type 'a bintree =
  Leaf of 'a
  | Node of 'a bintree * 'a bintree;;
type 'a bintree = Leaf of 'a | Node of 'a bintree * 'a bintree

# let t: bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3);;
Characters 7-14:
  let t: bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3);;
      ^^^^^^^
The type constructor bintree expects 1 argument(s),
but is here applied to 0 argument(s)

# let t: int bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3);;
val t : int bintree = Node (Node (Leaf 1, Leaf 2), Leaf 3)

# let rec leaves t =
  match t with
  | Leaf x -> [x]
  | Node (t1, t2) -> leaves t1 @ leaves t2;;
  val leaves : 'a bintree -> 'a list = <fun>

# leaves t;;
- : int list = [1; 2; 3]

# type ('a,'b) bintree =
  Leaf of 'a
  | Node of 'b * ('a,'b) bintree * ('a,'b) bintree;;
type ('a,'b) bintree =
  Leaf of 'a
  | Node of 'b * ('a,'b) bintree * ('a,'b) bintree

# let t = Node ((-), Node ((+), Leaf 1, Leaf 2), (Node ((-), Leaf 3, Leaf 4)));;
val t : (int, int -> int -> int) bintree =
  Node (<fun>, Node (<fun>, Leaf 1, Leaf 2), Node (<fun>, Leaf 3, Leaf 4))

# let rec eval t =
  match t with
  | Leaf x -> x
  | Node (f, t1, t2) -> f (eval t1) (eval t2);;
  val eval : ('a, 'a -> 'a -> 'a) bintree -> 'a = <fun>

# eval t;;
- : int = 4

# type 'a tree =
  Leaf of 'a
  | Node of 'a tree list;;
type 'a tree = Leaf of 'a | Node of 'a tree list

# let t = Node [Node [Leaf 1; Leaf 2; Leaf 3]; Leaf 4];;
val t : int tree = Node [Node [Leaf 1; Leaf 2; Leaf 3]; Leaf 4]

# let rec leaves t =
  match t with
  | Leaf x -> [x]
  | Node l -> List.concat (List.map leaves l);;
  val leaves : 'a tree -> 'a list = <fun>

# leaves t;;
- : int list = [1; 2; 3; 4]

# type ('a,'b) tree =
  Leaf of 'a
  | Node of 'b * ('a,'b) tree list;;
type ('a,'b) tree = Leaf of 'a | Node of 'b * ('a,'b) tree list
```

```
# let add [x;y] = x + y;;
Characters 8-21:
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
  let add [x;y] = x + y;;
          ^^^^^^^^^^^^^^^
val add : int list -> int = <fun>

# let uminus [x] = -x;;
Characters 11-19:
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
  let uminus [x] = -x;;
             ^^^^^^^
val uminus : int list -> int = <fun>

# let t = Node (add, [Node (uminus, [Leaf 5]); Node (uminus, [Leaf 5])]);;
val t : (int, int list -> int) tree =
  Node (<fun>, [Node (<fun>, [Leaf 5]); Node (<fun>, [Leaf 5])])

# let rec eval t =
  match t with
  | Leaf x -> x
  | Node (f, l) -> f (List.map eval l);;
  val eval : ('a, 'a list -> 'a) tree -> 'a = <fun>

# eval t;;
- : int = -10

#
```