



**OCaml-Hausaufgaben die nicht kompilieren werden nicht gewertet!** Sollten Ihr einmal in die Situation kommen, dass sich ein Fehler in einen Teil Eurer Abgabe eingeschlichen hat, der zur Folge hat, dass sich die gesamte Datei nicht mehr kompilieren lässt, dann kommentiert diesen Teil aus und ersetzt ihn durch den Standardwert aus der Angabe (z.B. `let f x = todo ()`). Andernfalls wird die gesamte OCaml-Hausaufgabe mit Null Punkten gewertet.

### Aufgabe 8.1 OCaml-Hausaufgabe

Zur Abgabe melden Sie sich mit Ihrer TUM-Kennung auf <https://vmnipkow3.in.tum.de> an und laden ihre `ha4.ml`-Datei (nicht umbenennen!) hoch. Die Ergebnisse der Tests sind sichtbar sobald die Abgabe abgearbeitet wurde. Die endgültige Bewertung erfolgt aber erst nach der Frist.

Die Datei `batteries.ml` enthält die bisher implementierten Funktionen und ist auch in der Testumgebung verfügbar (muss nicht hochgeladen werden).

In der Datei gibt es zwei Module mit Funktionen die zu implementieren sind:

- a) `Map`: Mehr Funktionen zu Maps als Binärbäume.
- b) `Json`: Funktionen zum Arbeiten mit JSON-Dokumenten.

### Aufgabe 8.2 OCaml-Tutoraufgaben

- a) Im folgenden definieren wir einige Hilfsfunktion zum Umgang mit `option`. Machen Sie sich klar wie Sie mit `>>=` Ihren Code verständlicher machen können.

```
module Option = struct
  let some x = Some x
  let is_some x = x <> None
  let is_none x = x = None

  let map f = function
    | Some x -> Some (f x)
    | None -> None

  let bind o f = match o with
    | Some x -> f x
    | None -> None
  let (>>=) = bind

  let filter p = function
    | Some x when p x -> Some x
```

```

| _ -> None

let default x = function
| Some y -> y
| None -> x
(* Some 1 |? 0 = 1, None |? 0 = 0, None |? Some 1 |? 0 = 1*)
let (|?) x y = default y x

let try_some f x = try Some (f x) with _ -> None
end

```

- b) Wir nutzen im Folgenden Binärbäume um eine Abbildung von Schlüsseln auf Werte darzustellen. Verstehen Sie den Typ und implementieren Sie folgende Grundfunktionen:

```

type ('k, 'v) t = Empty | Node of 'k * 'v * ('k, 'v) t * ('k, 'v) t
val empty : ('k, 'v) t
(* { }, { k1 -> v1 }, { k1 -> v1, k2 -> v2 } *)
val show : ('k -> string) -> ('v -> string) -> ('k, 'v) t -> string
val add : 'k -> 'v -> ('k, 'v) t -> ('k, 'v) t
val find : 'k -> ('k, 'v) t -> 'v option

```