

Containers	Batteries	Base
CCArrayLabels.(--) : int -> int -> int array		
CCArrayLabels.(--^) : int -> int -> int array		
CCArrayLabels.(>=) : 'a array -> ('a -> 'b array) -> 'b array		
CCArrayLabels.(>) : 'a array -> ('a -> 'b) -> 'b array		
CCArrayLabels.(> =) : 'a array -> ('a -> 'b) -> 'b array		
CCArrayLabels.(and*) : 'a array -> 'b array -> ('a * 'b) array		
CCArrayLabels.(and+) : 'a array -> 'b array -> ('a * 'b) array		
CCArrayLabels.(let*) : 'a array -> ('a -> 'b array) -> 'b array		
CCArrayLabels.(let+) : 'a array -> ('a -> 'b) -> 'b array		
CCArrayLabels.bsearch : cmp:('a -> 'a -> int) -> key:'a -> 'a array -> [`All_bigger `All_lower `At of int `Empty `Just_after of int]		
CCArrayLabels.empty : 'a array = []		
CCArrayLabels.except_idx : 'a array -> int -> 'a list		
CCArrayLabels.find_idx : f:('a -> bool) -> 'a array -> (int * 'a) option		
CCArrayLabels.find_map : f:('a -> 'b option) -> 'a array -> 'b option		
CCArrayLabels.find_map_i : f:(int -> 'a -> 'b option) -> 'a array -> 'b option		
CCArrayLabels.flat_map : f:('a -> 'b array) -> 'a array -> 'b array		
CCArrayLabels.fold_while : f:('a -> 'b -> 'a * [`Continue `Stop]) -> init:'a -> 'b array -> 'a		
CCArrayLabels.get_safe : 'a array -> int -> 'a option		
CCArrayLabels.lookup : cmp:'a CCArrayLabels.ord -> key:'a -> 'a array -> int option		
CCArrayLabels.lookup_exn : cmp:'a CCArrayLabels.ord -> key:'a -> 'a array -> int		
CCArrayLabels.monoid_product : f:('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array		
CCArrayLabels.pp : ?pp_start:unit CCArrayLabels.printer -> ?pp_stop:unit CCArrayLabels.printer -> ?pp_sep:unit CCArrayLabels.printer -> 'a CCArrayLabels.printer -> 'a array CCArrayLabels.printer		
CCArrayLabels.pp_i : ?pp_start:unit CCArrayLabels.printer -> ?pp_stop:unit CCArrayLabels.printer -> ?pp_sep:unit CCArrayLabels.printer -> (int -> 'a CCArrayLabels.printer) -> 'a array CCArrayLabels.printer		
CCArrayLabels.random : 'a CCArrayLabels.random_gen -> 'a array CCArrayLabels.random_gen		
CCArrayLabels.random_len : int -> 'a CCArrayLabels.random_gen -> 'a array CCArrayLabels.random_gen		
CCArrayLabels.random_non_empty : 'a CCArrayLabels.random_gen -> 'a array CCArrayLabels.random_gen		
CCArrayLabels.rev : 'a array -> 'a array		
CCArrayLabels.scan_left : f:(acc -> 'a -> 'acc) -> init:'acc -> 'a array -> 'acc array		
CCArrayLabels.shuffle : 'a array -> unit		
CCArrayLabels.sort_generic : (module CCArrayLabels.MONO_ARRAY with type elt = 'elt and type t = 'arr) -> cmp:('elt -> 'elt -> int) -> 'arr -> unit		
CCArrayLabels.sort_indices : f:('a -> 'a -> int) -> 'a array -> int array		
CCArrayLabels.sort_ranking : f:('a -> 'a -> int) -> 'a array -> int array		
CCArrayLabels.to_gen : 'a array -> 'a CCArrayLabels.gen		
CCArrayLabels.to_iter : 'a array -> 'a CCArrayLabels.iter		
CCArrayLabels.to_string : ?sep:string -> ('a -> string) -> 'a array -> string		

Containers	Batteries	Base
CCArray.(-) : int -> int -> int array		
CCArray.(~^) : int -> int -> int array		
CCArray.(>=>) : 'a array -> ('a -> 'b array) -> 'b array		
CCArray.(>>) : 'a array -> ('a -> 'b) -> 'b array		
CCArray.(>) : 'a array -> ('a -> 'b) -> 'b array		
CCArray.(and*) : 'a array -> 'b array -> ('a * 'b) array		
CCArray.(and+) : 'a array -> 'b array -> ('a * 'b) array		
CCArray.(let*) : 'a array -> ('a -> 'b array) -> 'b array		
CCArray.(let+) : 'a array -> ('a -> 'b) -> 'b array		
CCArray.empty : 'a array = []		
CCArray.except_idx : 'a array -> int -> 'a list		
CCArray.find_idx : ('a -> bool) -> 'a array -> (int * 'a) option		
CCArray.find_map : ('a -> 'b option) -> 'a array -> 'b option		
CCArray.find_map_i : (int -> 'a -> 'b option) -> 'a array -> 'b option		
CCArray.flat_map : ('a -> 'b array) -> 'a array -> 'b array		
CCArray.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a array -> 'b array -> 'acc		
CCArray.fold_map : ('acc -> 'a -> 'acc * 'b) -> 'acc -> 'a array -> 'acc * 'b array		
CCArray.fold_while : ('a -> 'b -> 'a * [`Continue `Stop]) -> 'a -> 'b array -> 'a		
CCArray.foldi : ('a -> int -> 'b -> 'a) -> 'a -> 'b array -> 'a		
CCArray.get_safe : 'a array -> int -> 'a option		
CCArray.lookup : cmp:'a CCArray.ord -> 'a -> 'a array -> int option		
CCArray.lookup_exn : cmp:'a CCArray.ord -> 'a -> 'a array -> int		
CCArray.monoid_product : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array		
CCArray.pp : ?pp_start:unit CCArray.printer -> ?pp_stop:unit CCArray.printer -> ? pp_sep:unit CCArray.printer -> 'a CCArray.printer -> 'a array CCArray.printer		
CCArray.pp_i : ?pp_start:unit CCArray.printer -> ?pp_stop:unit CCArray.printer -> ? pp_sep:unit CCArray.printer -> (int -> 'a CCArray.printer) -> 'a array CCArray.printer		
CCArray.random : 'a CCArray.random_gen -> 'a array CCArray.random_gen		
CCArray.random_choose : 'a array -> 'a CCArray.random_gen		
CCArray.random_len : int -> 'a CCArray.random_gen -> 'a array CCArray.random_gen		
CCArray.random_non_empty : 'a CCArray.random_gen -> 'a array CCArray.random_gen		
CCArray.scan_left : ('acc -> 'a -> 'acc) -> 'acc -> 'a array -> 'acc array		
CCArray.shuffle_with : Random.State.t -> 'a array -> unit		
CCArray.sort_generic : (module CCArray.MONO_ARRAY with type elt = 'elt and type t = 'arr) -> cmp:(elt -> 'elt -> int) -> 'arr -> unit		
CCArray.sort_indices : ('a -> 'a -> int) -> 'a array -> int array		
CCArray.sort_ranking : ('a -> 'a -> int) -> 'a array -> int array		
CCArray.sorted : ('a -> 'a -> int) -> 'a array -> 'a array		
CCArray.swap : 'a array -> int -> int -> unit		
CCArray.to_gen : 'a array -> 'a CCArray.gen		
CCArray.to_iter : 'a array -> 'a CCArray.iter		

Containers	Batteries	Base
CArray.to_string : ?sep:string -> ('a -> string) -> 'a array -> string		
CListLabels.(-) : int -> int -> int list		
CListLabels.(-^) : int -> int -> int list		
CListLabels.(<\$>) : ('a -> 'b) -> 'a list -> 'b list		
CListLabels.(<*>) : ('a -> 'b) list -> 'a list -> 'b list		
CListLabels.(and*) : 'a list -> 'b list -> ('a * 'b) list		
CListLabels.(and+) : 'a list -> 'b list -> ('a * 'b) list		
CListLabels.(let*) : 'a list -> ('a -> 'b list) -> 'b list		
CListLabels.(let+) : 'a list -> ('a -> 'b) -> 'b list		
CListLabels.Assoc.get : eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CListLabels.Assoc.t -> 'b option		
CListLabels.Assoc.get_exn : eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CListLabels.Assoc.t -> 'b		
CListLabels.Assoc.set : eq:(a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) CListLabels.Assoc.t -> ('a, 'b) CListLabels.Assoc.t		
CListLabels.Assoc.update : eq:(a -> 'a -> bool) -> f:(b option -> 'b option) -> 'a -> ('a, 'b) CListLabels.Assoc.t -> ('a, 'b) CListLabels.Assoc.t		
CListLabels.Infix.(and&) : 'a list -> 'b list -> ('a * 'b) list		
CListLabels.add_nodup : eq:(a -> 'a -> bool) -> 'a -> 'a list -> 'a list		
CListLabels.all_ok : ('a, 'err) result list -> ('a CListLabels.t, 'err) result		
CListLabels.all_some : 'a option list -> 'a list option		
CListLabels.combine_gen : 'a list -> 'b list -> ('a * 'b) CListLabels.gen		
CListLabels.combine_shortest : 'a list -> 'b list -> ('a * 'b) list		
CListLabels.cons' : 'a list -> 'a -> 'a list		
CListLabels.cons_maybe : 'a option -> 'a list -> 'a list		
CListLabels.count_true_false : f:(a -> bool) -> 'a list -> int * int		
CListLabels.diagonal : 'a list -> ('a * 'a) list		
CListLabels.empty : 'a list = []		
CListLabels.find_idx : f:(a -> bool) -> 'a list -> (int * 'a) option		
CListLabels.find_pred : f:(a -> bool) -> 'a list -> 'a option		
CListLabels.find_pred_exn : f:(a -> bool) -> 'a list -> 'a		
CListLabels.flat_map : f:(a -> 'b list) -> 'a list -> 'b list		
CListLabels.flat_map_i : f:(int -> 'a -> 'b list) -> 'a list -> 'b list		
CListLabels.fold_filter_map : f:(acc -> 'a -> 'acc * 'b option) -> init:'acc -> 'a list -> 'acc * 'b list		
CListLabels.fold_filter_map_i : f:(acc -> int -> 'a -> 'acc * 'b option) -> init:'acc -> 'a list -> 'acc * 'b list		
CListLabels.fold_flat_map : f:(acc -> 'a -> 'acc * 'b list) -> init:'acc -> 'a list -> 'acc * 'b list		
CListLabels.fold_flat_map_i : f:(acc -> int -> 'a -> 'acc * 'b list) -> init:'acc -> 'a list -> 'acc * 'b list		
CListLabels.fold_map2 : f:(acc -> 'a -> 'b -> 'acc * 'c) -> init:'acc -> 'a list -> 'b list -> 'acc * 'c list		
CListLabels.fold_on_map : f:(a -> 'b) -> reduce:(acc -> 'b -> 'acc) -> init:'acc -> 'a list -> 'acc		

Containers	Batteries	Base
CCListLabels.fold_product : f:(c -> 'a -> 'b -> 'c) -> init:'c -> 'a list -> 'b list -> 'c		
CCListLabels.fold_while : f:(a -> 'b -> 'a * [`Continue `Stop]) -> init:'a -> 'b list -> 'a		
CCListLabels.foldi2 : f:(c -> int -> 'a -> 'b -> 'c) -> init:'c -> 'a list -> 'b list -> 'c		
CCListLabels.get_at_idx : int -> 'a list -> 'a option		
CCListLabels.get_at_idx_exn : int -> 'a list -> 'a		
CCListLabels.group_join_by : ?eq:(a -> 'a -> bool) -> ?hash:(a -> int) -> ('b -> 'a) -> 'a list -> 'b list -> ('a * 'b list) list		
CCListLabels.group_succ : eq:(a -> 'a -> bool) -> 'a list -> 'a list list		
CCListLabels.hd_tl : 'a list -> 'a * 'a list		
CCListLabels.insert_at_idx : int -> 'a -> 'a list -> 'a list		
CCListLabels.inter : eq:(a -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
CCListLabels.interleave : 'a list -> 'a list -> 'a list		
CCListLabels.iteri2 : f:(int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit		
CCListLabels.join : join_row:(a -> 'b -> 'c option) -> 'a list -> 'b list -> 'c list		
CCListLabels.join_all_by : ?eq:(key -> 'key -> bool) -> ?hash:(key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:(key -> 'a list -> 'b list -> 'c option) -> 'a list -> 'b list -> 'c list		
CCListLabels.join_by : ?eq:(key -> 'key -> bool) -> ?hash:(key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:(key -> 'a -> 'b -> 'c option) -> 'a list -> 'b list -> 'c list		
CCListLabels.keep_ok : ('a, 'b) result list -> 'a list		
CCListLabels.keep_some : 'a option list -> 'a list		
CCListLabels.last : int -> 'a list -> 'a list		
CCListLabels.map_product_l : f:(a -> 'b list) -> 'a list -> 'b list list		
CCListLabels.mguard : bool -> unit list		
CCListLabels.of_gen : 'a CCListLabels.gen -> 'a list		
CCListLabels.of_iter : 'a CCListLabels.iter -> 'a list		
CCListLabels.of_seq_rev : 'a Seq.t -> 'a list		
CCListLabels.partition_filter_map : f:(a -> [< `Drop `Left of 'b `Right of 'c]) -> 'a list -> 'b list * 'c list		
CCListLabels.partition_map : f:(a -> [< `Drop `Left of 'b `Right of 'c]) -> 'a list -> 'b list * 'c list		
CCListLabels.pp : ?pp_start:unit CCListLabels.printer -> ?pp_stop:unit CCListLabels.printer -> ?pp_sep:unit CCListLabels.printer -> 'a CCListLabels.printer -> 'a list CCListLabels.printer		
CCListLabels.product : f:(a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list		
CCListLabels.pure : 'a -> 'a list		
CCListLabels.random : 'a CCListLabels.random_gen -> 'a list CCListLabels.random_gen		
CCListLabels.random_choose : 'a list -> 'a CCListLabels.random_gen		
CCListLabels.random_len : int -> 'a CCListLabels.random_gen -> 'a list CCListLabels.random_gen		
CCListLabels.random_non_empty : 'a CCListLabels.random_gen -> 'a list CCListLabels.random_gen		
CCListLabels.random_sequence : 'a CCListLabels.random_gen list -> 'a list CCListLabels.random_gen		
CCListLabels.range' : int -> int -> int list		
CCListLabels.range_by : step:int -> int -> int -> int list		

Containers	Batteries	Base
CCListLabels.remove : eq:(a -> 'a -> bool) -> key:'a -> 'a list -> 'a list		
CCListLabels.remove_at_idx : int -> 'a list -> 'a list		
CCListLabels.remove_one : eq:(a -> 'a -> bool) -> 'a -> 'a list -> 'a list		
CCListLabels.repeat : int -> 'a list -> 'a list		
CCListLabels.replicate : int -> 'a -> 'a list		
CCListLabels.scan_left : f:(acc -> 'a -> 'acc) -> init:'acc -> 'a list -> 'acc list		
CCListLabels.set_at_idx : int -> 'a -> 'a list -> 'a list		
CCListLabels.sorted_insert : cmp:(a -> 'a -> int) -> ?uniq:bool -> 'a -> 'a list -> 'a list		
CCListLabels.sorted_merge : cmp:(a -> 'a -> int) -> 'a list -> 'a list -> 'a list		
CCListLabels.sorted_merge_uniq : cmp:(a -> 'a -> int) -> 'a list -> 'a list -> 'a list		
CCListLabels.sublists_of_len : ?last:(a list -> 'a list option) -> ?offset:int -> len:int -> 'a list -> 'a list list		
CCListLabels.subset : eq:(a -> 'a -> bool) -> 'a list -> 'a list -> bool		
CCListLabels.tail_opt : 'a list -> 'a list option		
CCListLabels.take_drop : int -> 'a list -> 'a list * 'a list		
CCListLabels.take_drop_while : f:(a -> bool) -> 'a list -> 'a list * 'a list		
CCListLabels.to_gen : 'a list -> 'a CCListLabels.gen		
CCListLabels.to_iter : 'a list -> 'a CCListLabels.iter		
CCListLabels.to_string : ?start:string -> ?stop:string -> ?sep:string -> (a -> string) -> 'a list -> string		
CCListLabels.union : eq:(a -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
CCListLabels.uniq : eq:(a -> 'a -> bool) -> 'a list -> 'a list		
CCListLabels.uniq_succ : eq:(a -> 'a -> bool) -> 'a list -> 'a list		
CCList.(-) : int -> int -> int list		
CCList.(-^) : int -> int -> int list		
CCList.(<\$>) : (a -> 'b) -> 'a list -> 'b list		
CCList.(<*>) : (a -> 'b) list -> 'a list -> 'b list		
CCList.(>=>) : 'a list -> (a -> 'b list) -> 'b list		
CCList.(> =) : 'a list -> (a -> 'b) -> 'b list		
CCList.(@) : 'a list -> 'a list -> 'a list		
CCList.(and*) : 'a list -> 'b list -> (a * 'b) list		
CCList.(and+) : 'a list -> 'b list -> (a * 'b) list		
CCList.(!et*) : 'a list -> (a -> 'b list) -> 'b list		
CCList.(!et+) : 'a list -> (a -> 'b) -> 'b list		
CCList.Assoc.get : eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CCList.Assoc.t -> 'b option		
CCList.Assoc.get_lexn : eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CCList.Assoc.t -> 'b		
CCList.Assoc.mem : ?eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CCList.Assoc.t -> bool		
CCList.Assoc.remove : eq:(a -> 'a -> bool) -> 'a -> ('a, 'b) CCList.Assoc.t -> ('a, 'b) CCList.Assoc.t		
CCList.Assoc.set : eq:(a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) CCList.Assoc.t -> ('a, 'b) CCList.Assoc.t		
CCList.Assoc.update : eq:(a -> 'a -> bool) -> f:(b option -> 'b option) -> 'a -> ('a, 'b) CCList.Assoc.t -> ('a, 'b) CCList.Assoc.t		

Containers	Batteries	Base
CCList.Infix.(and&) : 'a list -> 'b list -> ('a * 'b) list		
CCList.add_nodup : eq:('a -> 'a -> bool) -> 'a -> 'a list -> 'a list		
CCList.all_ok : ('a, 'err) result list -> ('a CCList.t, 'err) result		
CCList.all_some : 'a option list -> 'a list option		
CCList.assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b		
CCList.assoc_opt : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b option		
CCList.chunks : int -> 'a list -> 'a list list		
CCList.combine_gen : 'a list -> 'b list -> ('a * 'b) CCList.gen		
CCList.combine_shortest : 'a list -> 'b list -> ('a * 'b) list		
CCList.cons' : 'a list -> 'a -> 'a list		
CCList.cons_maybe : 'a option -> 'a list -> 'a list		
CCList.count_true_false : ('a -> bool) -> 'a list -> int * int		
CCList.diagonal : 'a list -> ('a * 'a) list		
CCList.empty : 'a list = []		
CCList.find_idx : ('a -> bool) -> 'a list -> (int * 'a) option		
CCList.find_map1 : (int -> 'a -> 'b option) -> 'a list -> 'b option		
CCList.find_pred : ('a -> bool) -> 'a list -> 'a option		
CCList.find_pred_exn : ('a -> bool) -> 'a list -> 'a		
CCList.flat_map : ('a -> 'b list) -> 'a list -> 'b list		
CCList.flat_map1 : (int -> 'a -> 'b list) -> 'a list -> 'b list		
CCList.fold_filter_map : ('acc -> 'a -> 'acc * 'b option) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_filter_map1 : ('acc -> int -> 'a -> 'acc * 'b option) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_flat_map : ('acc -> 'a -> 'acc * 'b list) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_flat_map1 : ('acc -> int -> 'a -> 'acc * 'b list) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_map : ('acc -> 'a -> 'acc * 'b) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_map2 : ('acc -> 'a -> 'b -> 'acc * 'c) -> 'acc -> 'a list -> 'b list -> 'acc * 'c list		
CCList.fold_map1 : ('acc -> int -> 'a -> 'acc * 'b) -> 'acc -> 'a list -> 'acc * 'b list		
CCList.fold_on_map : f:('a -> 'b) -> reduce:('acc -> 'b -> 'acc) -> 'acc -> 'a list -> 'acc		
CCList.fold_product : ('c -> 'a -> 'b -> 'c) -> 'c -> 'a list -> 'b list -> 'c		
CCList.fold_while : ('a -> 'b -> 'a * [`Continue `Stop]) -> 'a -> 'b list -> 'a		
CCList.foldi2 : ('c -> int -> 'a -> 'b -> 'c) -> 'c -> 'a list -> 'b list -> 'c		
CCList.get_at_idx : int -> 'a list -> 'a option		
CCList.get_at_idx_exn : int -> 'a list -> 'a		
CCList.group_by : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a list -> 'a list list		
CCList.group_join_by : ?eq:('a -> 'a -> bool) -> ?hash:('a -> int) -> ('b -> 'a) -> 'a list -> 'b list -> ('a * 'b list) list		
CCList.group_succ : eq:('a -> 'a -> bool) -> 'a list -> 'a list list		
CCList.hd_tl : 'a list -> 'a * 'a list		
CCList.head_opt : 'a list -> 'a option		
CCList.insert_at_idx : int -> 'a -> 'a list -> 'a list		

Containers	Batteries	Base
CCList.inter : eq:(a -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
CCList.interleave : 'a list -> 'a list -> 'a list		
CCList.is_sorted : cmp:(a -> 'a -> int) -> 'a list -> bool		
CCList.iteri2 : (int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit		
CCList.join : join_row:(a -> 'b -> 'c option) -> 'a list -> 'b list -> 'c list		
CCList.join_all_by : ?eq:(key -> 'key -> bool) -> ?hash:(key -> int) -> (a -> 'key) -> (b -> 'key) -> merge:(key -> 'a list -> 'b list -> 'c option) -> 'a list -> 'b list -> 'c list		
CCList.join_by : ?eq:(key -> 'key -> bool) -> ?hash:(key -> int) -> (a -> 'key) -> (b -> 'key) -> merge:(key -> 'a -> 'b -> 'c option) -> 'a list -> 'b list -> 'c list		
CCList.keep_ok : (a, b) result list -> 'a list		
CCList.keep_some : 'a option list -> 'a list		
CCList.last : int -> 'a list -> 'a list		
CCList.last_opt : 'a list -> 'a option		
CCList.map_product_l : (a -> 'b list) -> 'a list -> 'b list list		
CCList.mguard : bool -> unit list		
CCList.of_gen : 'a CCList.gen -> 'a list		
CCList.of_iter : 'a CCList.iter -> 'a list		
CCList.of_seq_rev : 'a Seq.t -> 'a list		
CCList.partition_filter_map : (a -> [< `Drop `Left of 'b `Right of 'c]) -> 'a list -> 'b list * 'c list		
CCList.partition_map : (a -> [< `Drop `Left of 'b `Right of 'c]) -> 'a list -> 'b list * 'c list		
CCList.pp : ?pp_start:unit CCList.printer -> ?pp_stop:unit CCList.printer -> ?pp_sep:unit CCList.printer -> 'a CCList.printer -> 'a list CCList.printer		
CCList.product : (a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list		
CCList.pure : 'a -> 'a list		
CCList.random : 'a CCList.random_gen -> 'a list CCList.random_gen		
CCList.random_choose : 'a list -> 'a CCList.random_gen		
CCList.random_len : int -> 'a CCList.random_gen -> 'a list CCList.random_gen		
CCList.random_non_empty : 'a CCList.random_gen -> 'a list CCList.random_gen		
CCList.random_sequence : 'a CCList.random_gen list -> 'a list CCList.random_gen		
CCList.range : int -> int -> int list		
CCList.range' : int -> int -> int list		
CCList.range_by : step:int -> int -> int -> int list		
CCList.reduce : (a -> 'a -> 'a) -> 'a list -> 'a option		
CCList.remove : eq:(a -> 'a -> bool) -> key:'a -> 'a list -> 'a list		
CCList.remove_one : eq:(a -> 'a -> bool) -> 'a -> 'a list -> 'a list		
CCList.repeat : int -> 'a list -> 'a list		
CCList.replicate : int -> 'a -> 'a list		
CCList.return : 'a -> 'a list		
CCList.scan_left : (acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc list		
CCList.set_at_idx : int -> 'a -> 'a list -> 'a list		
CCList.sorted_insert : cmp:(a -> 'a -> int) -> ?uniq:bool -> 'a -> 'a list -> 'a list		

Containers	Batteries	Base
CCList.sorted_merge : cmp:(<i>a</i> -> <i>a</i> -> int) -> 'a list -> 'a list -> 'a list		
CCList.sorted_merge_uniq : cmp:(<i>a</i> -> <i>a</i> -> int) -> 'a list -> 'a list -> 'a list		
CCList.sublists_of_len : ?last:(<i>a</i> list -> 'a list option) -> ?offset:int -> int -> 'a list -> 'a list list		
CCList.subset : eq:(<i>a</i> -> 'a -> bool) -> 'a list -> 'a list -> bool		
CCList.tail_opt : 'a list -> 'a list option		
CCList.take_drop : int -> 'a list -> 'a list * 'a list		
CCList.take_drop_while : (<i>a</i> -> bool) -> 'a list -> 'a list * 'a list		
CCList.to_gen : 'a list -> 'a CCList.gen		
CCList.to_iter : 'a list -> 'a CCList.iter		
CCList.to_string : ?start:string -> ?stop:string -> ?sep:string -> (<i>a</i> -> string) -> 'a list -> string		
CCList.union : eq:(<i>a</i> -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
CCList.uniq_succ : eq:(<i>a</i> -> 'a -> bool) -> 'a list -> 'a list		
CCMap.Make.add_iter		
CCMap.Make.add_iter_with		
CCMap.Make.add_list		
CCMap.Make.add_list_with		
CCMap.Make.add_seq_with		
CCMap.Make.get		
CCMap.Make.get_or		
CCMap.Make.merge_safe		
CCMap.Make.of_iter		
CCMap.Make.of_iter_with		
CCMap.Make.of_list		
CCMap.Make.of_list_with		
CCMap.Make.of_seq_with		
CCMap.Make.pp		
CCMap.Make.to_iter		
CCMap.Make.to_list		
CCOpt.(<\$>) : (<i>a</i> -> 'b) -> 'a option -> 'b option		
CCOpt.(<*>) : (<i>a</i> -> 'b) option -> 'a option -> 'b option		
CCOpt.(<+>) : 'a option -> 'a option -> 'a option		
CCOpt.(and*) : 'a option -> 'b option -> (<i>a</i> * 'b) option		
CCOpt.(and+) : 'a option -> 'b option -> (<i>a</i> * 'b) option		
CCOpt.(let*) : 'a option -> (<i>a</i> -> 'b option) -> 'b option		
CCOpt.(let+) : 'a option -> (<i>a</i> -> 'b) -> 'b option		
CCOpt.choice : 'a option list -> 'a option		
CCOpt.choice_iter : 'a option CCOpt.iter -> 'a option		
CCOpt.choice_seq : 'a option Seq.t -> 'a option		
CCOpt.get_exn_or : string -> 'a option -> 'a		
CCOpt.get_lazy : (unit -> 'a) -> 'a option -> 'a		

Containers	Batteries	Base
CCOpt.get_or : default:'a -> 'a option -> 'a		
CCOpt.if_ : ('a -> bool) -> 'a -> 'a option		
CCOpt.map_lazy : (unit -> 'b) -> ('a -> 'b) -> 'a option -> 'b		
CCOpt.map_or : default:'b -> ('a -> 'b) -> 'a option -> 'b		
CCOpt.of_list : 'a list -> 'a option		
CCOpt.of_result : ('a, 'b) result -> 'a option		
CCOpt.or_ : else_:'a option -> 'a option -> 'a option		
CCOpt.or_lazy : else_(unit -> 'a option) -> 'a option -> 'a option		
CCOpt.pp : 'a CCOpt.printer -> 'a option CCOpt.printer		
CCOpt.pure : 'a -> 'a option		
CCOpt.random : 'a CCOpt.random_gen -> 'a option CCOpt.random_gen		
CCOpt.return_if : bool -> 'a -> 'a option		
CCOpt.sequence_l : 'a option list -> 'a list option		
CCOpt.to_gen : 'a option -> 'a CCOpt.gen		
CCOpt.to_iter : 'a option -> 'a CCOpt.iter		
CCOpt.to_result_lazy : (unit -> 'e) -> 'a option -> ('a, 'e) result		
CCOpt.wrap : ?handler:(exn -> bool) -> ('a -> 'b) -> 'a -> 'b option		
CCOpt.wrap2 : ?handler:(exn -> bool) -> ('a -> 'b -> 'c) -> 'a -> 'b -> 'c option		
CCResult(<\$>) : ('a -> 'b) -> ('a, 'err) result -> ('b, 'err) result		
CCResult(<*>) : ('a -> 'b, 'err) result -> ('a, 'err) result -> ('b, 'err) result		
CCResult(and*) : ('a, 'e) result -> ('b, 'e) result -> ('a * 'b, 'e) result		
CCResult(and+) : ('a, 'e) result -> ('b, 'e) result -> ('a * 'b, 'e) result		
CCResult(let*) : ('a, 'e) result -> ('a -> ('b, 'e) result) -> ('b, 'e) result		
CCResult(let+) : ('a, 'e) result -> ('a -> 'b) -> ('b, 'e) result		
CCResult.add_ctx : string -> ('a, string) result -> ('a, string) result		
CCResult.add_ctxf : ('a, Format.formatter, unit, ('b, string) result) -> ('b, string) result format4 -> 'a		
CCResult.both : ('a, 'err) result -> ('b, 'err) result -> ('a * 'b, 'err) result		
CCResult.catch : ('a, 'err) result -> ok:('a -> 'b) -> err:('err -> 'b) -> 'b		
CCResult.choose : ('a, 'err) result list -> ('a, 'err list) result		
CCResult.fail_printf : ('a, Format.formatter, unit, ('b, string) result) format4 -> 'a		
CCResult.fail_printf : ('a, Buffer.t, unit, ('b, string) result) format4 -> 'a		
CCResult.flat_map : ('a -> ('b, 'err) result) -> ('a, 'err) result -> ('b, 'err) result		
CCResult.flatten_l : ('a, 'err) result list -> ('a list, 'err) result		
CCResult.fold_iter : ('b -> 'a -> ('b, 'err) result) -> 'b -> 'a CCRResult.iter -> ('b, 'err) result		
CCResult.fold_l : ('b -> 'a -> ('b, 'err) result) -> 'b -> 'a list -> ('b, 'err) result		
CCResult.fold_ok : ('a -> 'b -> 'a) -> 'a -> ('b, 'c) result -> 'a		
CCResult.get_lazy : ('b -> 'a) -> ('a, 'b) result -> 'a		
CCResult.get_or : ('a, 'b) result -> default:'a -> 'a		
CCResult.get_or_failwith : ('a, string) result -> 'a		
CCResult.guard : (unit -> 'a) -> ('a, exn) result		

Containers	Batteries	Base
CCResult.guard_str : (unit -> 'a) -> ('a, string) result		
CCResult.guard_str_trace : (unit -> 'a) -> ('a, string) result		
CCResult.map2 : ('a -> 'b) -> ('err1 -> 'err2) -> ('a, 'err1) result -> ('b, 'err2) result		
CCResult.map_L : ('a -> ('b, 'err) result) -> 'a list -> ('b list, 'err) result		
CCResult.map_or : ('a -> 'b) -> ('a, 'c) result -> default:'b -> 'b		
CCResult.of_err : ('a, 'b) CCResult.error -> ('a, 'b) result		
CCResult.of_exn : exn -> ('a, string) result		
CCResult.of_exn_trace : exn -> ('a, string) result		
CCResult.of_opt : 'a option -> ('a, string) result		
CCResult.pp : 'a CCResult.printer -> ('a, string) result CCResult.printer		
CCResult.pp' : 'a CCResult.printer -> 'e CCResult.printer -> ('a, 'e) result CCResult.printer		
CCResult.pure : 'a -> ('a, 'err) result		
CCResult.retry : int -> (unit -> ('a, 'err) result) -> ('a, 'err list) result		
CCResult.to_err : ('a, 'b) result -> ('a, 'b) CCResult.error		
CCResult.to_iter : ('a, 'b) result -> 'a CCResult.iter		
CCResult.wrap1 : ('a -> 'b) -> 'a -> ('b, exn) result		
CCResult.wrap2 : ('a -> 'b -> 'c) -> 'a -> 'b -> ('c, exn) result		
CCResult.wrap3 : ('a -> 'b -> 'c -> 'd) -> 'a -> 'b -> 'c -> ('d, exn) result		
CCSeq.(<*>) : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
CCSeq.(<.>) : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
CCSeq.(>>) : 'a Seq.t -> ('a -> 'b Seq.t) -> 'b Seq.t		
CCSeq.cycle : 'a Seq.t -> 'a Seq.t		
CCSeq.exists2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
CCSeq.fair_app : ('a -> 'b) Seq.t -> 'a Seq.t -> 'b Seq.t		
CCSeq.fair_flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t		
CCSeq.fmap : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t		
CCSeq.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a Seq.t -> 'b Seq.t -> 'acc		
CCSeq.for_all2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
CCSeq.group : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t Seq.t		
CCSeq.of_array : 'a array -> 'a Seq.t		
CCSeq.of_gen : 'a CCSeq.gen -> 'a Seq.t		
CCSeq.pp : ?pp_start:unit CCSeq.printer -> ?pp_stop:unit CCSeq.printer -> ? pp_sep:unit CCSeq.printer -> 'a CCSeq.printer -> 'a Seq.t CCSeq.printer		
CCSeq.product : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t		
CCSeq.product_with : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t		
CCSeq.pure : 'a -> 'a Seq.t		
CCSeq.sort : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t		
CCSeq.sort_uniq : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t		
CCSeq.tail : 'a Seq.t -> 'a Seq.t option		
CCSeq.tail_exn : 'a Seq.t -> 'a Seq.t		
CCSeq.to_gen : 'a Seq.t -> 'a CCSeq.gen		

Containers	Batteries	Base
CCSeq.to_iter : 'a Seq.t -> 'a CCSeq.iter		
CCSeq.uniq : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t		
CCSeq.unzip : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t		
CCSet.Make.add_iter		
CCSet.Make.add_list		
CCSet.Make.of_iter		
CCSet.Make.pp		
CCSet.Make.to_iter		
CCSet.Make.to_string		
CCStringLabels.compare_natural : string -> string -> int		
CCStringLabels.compare_versions : string -> string -> int		
CCStringLabels.concat_gen : sep:string -> string CCStringLabels.gen -> string		
CCStringLabels.concat_iter : sep:string -> string CCStringLabels.iter -> string		
CCStringLabels.concat_seq : sep:string -> string Seq.t -> string		
CCStringLabels.drop : int -> string -> string		
CCStringLabels.drop_while : f:(char -> bool) -> string -> string		
CCStringLabels.equal_caseless : string -> string -> bool		
CCStringLabels.exists2 : f:(char -> char -> bool) -> string -> string -> bool		
CCStringLabels.find_all_l : ?start:int -> sub:string -> string -> int list		
CCStringLabels.fold2 : f:(a -> char -> char -> 'a) -> init:'a -> string -> string -> 'a		
CCStringLabels.for_all2 : f:(char -> char -> bool) -> string -> string -> bool		
CCStringLabels.is_sub : sub:string -> sub_pos:int -> string -> pos:int -> sub_len:int -> bool		
CCStringLabels.iter2 : f:(char -> char -> unit) -> string -> string -> unit		
CCStringLabels.iteri2 : f:(int -> char -> char -> unit) -> string -> string -> unit		
CCStringLabels.lines : string -> string list		
CCStringLabels.lines_gen : string -> string CCStringLabels.gen		
CCStringLabels.lines_iter : string -> string CCStringLabels.iter		
CCStringLabels.lines_seq : string -> string Seq.t		
CCStringLabels.ltrim : string -> string		
CCStringLabels.map2 : f:(char -> char -> char) -> string -> string -> string		
CCStringLabels.of_array : char array -> string		
CCStringLabels.of_gen : char CCStringLabels.gen -> string		
CCStringLabels.of_iter : char CCStringLabels.iter -> string		
CCStringLabels.pad : ?side:['Left' 'Right'] -> ?c:char -> int -> string -> string		
CCStringLabels.pp_buf : Buffer.t -> string -> unit		
CCStringLabels.prefix : pre:string -> string -> bool		
CCStringLabels.rdrop_while : f:(char -> bool) -> string -> string		
CCStringLabels.replace : ?which:['All' 'Left' 'Right'] -> sub:string -> by:string -> string -> string		
CCStringLabels.rtrim : string -> string		
CCStringLabels.split : by:string -> string -> string list		

Containers	Batteries	Base
CCStringLabels.suffix : suf:string -> string -> bool		
CCStringLabels.take : int -> string -> string		
CCStringLabels.take_drop : int -> string -> string * string		
CCStringLabels.to_gen : string -> char CCStringLabels.gen		
CCStringLabels.to_iter : string -> char CCStringLabels.iter		
CCStringLabels.unlines : string list -> string		
CCStringLabels.unlines_gen : string CCStringLabels.gen -> string		
CCStringLabels.unlines_iter : string CCStringLabels.iter -> string		
CCStringLabels.unlines_seq : string Seq.t -> string		
CCString.(<) : string -> string -> bool		
CCString.(<=) : string -> string -> bool		
CCString.(<>) : string -> string -> bool		
CCString.(=) : string -> string -> bool		
CCString.(>) : string -> string -> bool		
CCString.(>=) : string -> string -> bool		
CCString.chop_prefix : pre:string -> string -> string option		
CCString.chop_suffix : suf:string -> string -> string option		
CCString.compare_natural : string -> string -> int		
CCString.compare_versions : string -> string -> int		
CCString.concat_gen : sep:string -> string CCString.gen -> string		
CCString.concat_iter : sep:string -> string CCString.iter -> string		
CCString.concat_seq : sep:string -> string Seq.t -> string		
CCString.drop : int -> string -> string		
CCString.drop_while : (char -> bool) -> string -> string		
CCString.edit_distance : ?cutoff:int -> string -> string -> int		
CCString.equal_caseless : string -> string -> bool		
CCString.exists : (char -> bool) -> string -> bool		
CCString.exists2 : (char -> char -> bool) -> string -> string -> bool		
CCString.filter : (char -> bool) -> string -> string		
CCString.filter_map : (char -> char option) -> string -> string		
CCString.find : ?start:int -> sub:string -> string -> int		
CCString.find_all : ?start:int -> sub:string -> string -> int CCString.gen		
CCString.find_all_l : ?start:int -> sub:string -> string -> int list		
CCString.flat_map : ?sep:string -> (char -> string) -> string -> string		
CCString.fold : ('a -> char -> 'a) -> 'a -> string -> 'a		
CCString.fold2 : ('a -> char -> char -> 'a) -> 'a -> string -> string -> 'a		
CCString.foldl : ('a -> int -> char -> 'a) -> 'a -> string -> 'a		
CCString.for_all : (char -> bool) -> string -> bool		
CCString.for_all2 : (char -> char -> bool) -> string -> string -> bool		
CCString.hash : string -> int		
CCString.is_empty : string -> bool		

Containers	Batteries	Base
CCString.is_sub : sub:string -> int -> string -> int -> sub_len:int -> bool		
CCString.iter2 : (char -> char -> unit) -> string -> string -> unit		
CCString.iteri2 : (int -> char -> char -> unit) -> string -> string -> unit		
CCString.lines : string -> string list		
CCString.lines_gen : string -> string CCString.gen		
CCString.lines_iter : string -> string CCString.iter		
CCString.lines_seq : string -> string Seq.t		
CCString.ltrim : string -> string		
CCString.map2 : (char -> char -> char) -> string -> string -> string		
CCString.mem : ?start:int -> sub:string -> string -> bool		
CCString.of_array : char array -> string		
CCString.of_char : char -> string		
CCString.of_gen : char CCString.gen -> string		
CCString.of_iter : char CCString.iter -> string		
CCString.of_list : char list -> string		
CCString.pad : ?side:[`Left `Right] -> ?c:char -> int -> string -> string		
CCString.pp : Format.formatter -> string -> unit		
CCString.pp_buf : Buffer.t -> string -> unit		
CCString.prefix : pre:string -> string -> bool		
CCString.rdrop_while : (char -> bool) -> string -> string		
CCString.repeat : string -> int -> string		
CCString.replace : ?which:[`All `Left `Right] -> sub:string -> by:string -> string -> string		
CCString.rev : string -> string		
CCString.rfind : sub:string -> string -> int		
CCString.rtrim : string -> string		
CCString.split : by:string -> string -> string list		
CCString.suffix : suf:string -> string -> bool		
CCString.take : int -> string -> string		
CCString.take_drop : int -> string -> string * string		
CCString.to_array : string -> char array		
CCString.to_gen : string -> char CCString.gen		
CCString.to_iter : string -> char CCString.iter		
CCString.to_list : string -> char list		
CCString.uniq : (char -> char -> bool) -> string -> string		
CCString.unlines : string list -> string		
CCString.unlines_gen : string CCString.gen -> string		
CCString.unlines_iter : string CCString.iter -> string		
CCString.unlines_seq : string Seq.t -> string		
	BatArray.Labels.findi : f:(a -> bool) -> 'a array -> int	
	BatArray.Labels.fold_while : p:(acc -> 'a -> bool) -> f:(acc -> 'a -> 'acc) -> init:'acc -> 'a array -> 'acc * int	

Containers	Batteries	Base
	BatArray.Labels.iter2i : f:(int -> 'a -> 'b -> unit) -> 'a array -> 'b array -> unit	
	BatArray.Labels.modify : f:(a -> 'a) -> 'a array -> unit	
	BatArray.Labels.modifyi : f:(int -> 'a -> 'a) -> 'a array -> unit	
	BatArray.avg : int array -> float	
	BatArray.backwards : 'a array -> 'a BatEnum.t	
	BatArray.cartesian_product : 'a array -> 'b array -> ('a * 'b) array	
	BatArray.count_matching : ('a -> bool) -> 'a array -> int	
	BatArray.decorate_fast_sort : ('a -> 'b) -> 'a array -> 'a array	
	BatArray.decorate_stable_sort : ('a -> 'b) -> 'a array -> 'a array	
	BatArray.enum : 'a array -> 'a BatEnum.t	
	BatArray.favg : float array -> float	
	BatArray.filteri : (int -> 'a -> bool) -> 'a array -> 'a array	
	BatArray.find : ('a -> bool) -> 'a array -> 'a	
	BatArray.find_all : ('a -> bool) -> 'a array -> 'a array	
	BatArray.findi : ('a -> bool) -> 'a array -> int	
	BatArray.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b array -> 'a	
	BatArray.fold_righti : (int -> 'b -> 'a -> 'a) -> 'b array -> 'a -> 'a	
	BatArray.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a array -> 'acc * int	
	BatArray.fsum : float array -> float	
	BatArray.head : 'a array -> int -> 'a array	
	BatArray.insert : 'a array -> 'a -> int -> 'a array	
	BatArray.is_sorted_by : ('a -> 'b) -> 'a array -> bool	
	BatArray.iter2i : (int -> 'a -> 'b -> unit) -> 'a array -> 'b array -> unit	
	BatArray.kahan_sum : float array -> float	
	BatArray.left : 'a array -> int -> 'a array	
	BatArray.max : 'a array -> 'a	
	BatArray.min : 'a array -> 'a	
	BatArray.min_max : 'a array -> 'a * 'a	
	BatArray.modify : ('a -> 'a) -> 'a array -> unit	
	BatArray.modifyi : (int -> 'a -> 'a) -> 'a array -> unit	
	BatArray.of_backwards : 'a BatEnum.t -> 'a array	
	BatArray.of_enum : 'a BatEnum.t -> 'a array	
	BatArray.ord : 'a BatOrd.ord -> 'a array BatOrd.ord	
	BatArray.partition : ('a -> bool) -> 'a array -> 'a array * 'a array	
	BatArray.pivot_split : 'a BatOrd.ord -> 'a array -> 'a -> int * int	
	BatArray.print : ?first:string -> ?last:string -> ?sep:string -> ('a, 'b) BatIO.printer -> ('a array, 'b) BatIO.printer	
	BatArray.range : 'a array -> int BatEnum.t	
	BatArray.reduce : ('a -> 'a -> 'a) -> 'a array -> 'a	
	BatArray.remove_at : int -> 'a array -> 'a array	
	BatArray.right : 'a array -> int -> 'a array	

Containers	Batteries	Base
	BatArray.singleton : 'a -> 'a array	
	BatArray.split : ('a * 'b) array -> 'a array * 'b array	
	BatArray.sum : int array -> int	
	BatArray.tail : 'a array -> int -> 'a array	
	BatList.Labels.find_exn : f:('a -> bool) -> exn -> 'a list -> 'a	
	BatList.Labels.findi : f:(int -> 'a -> bool) -> 'a list -> int * 'a	
	BatList.Labels.remove_if : f:('a -> bool) -> 'a list -> 'a list	
	BatList.Labels.rfind : f:('a -> bool) -> 'a list -> 'a	
	BatList.Labels.subset : cmp:('a -> 'b -> int) -> 'a list -> 'b list -> bool	
	BatList.assoc_inv : 'b -> ('a * 'b) list -> 'a	
	BatList.assq_inv : 'b -> ('a * 'b) list -> 'a	
	BatList.at : 'a list -> int -> 'a	
	BatList.at_opt : 'a list -> int -> 'a option	
	BatList.backwards : 'a list -> 'a BatEnum.t	
	BatList.cartesian_product : 'a list -> 'b list -> ('a * 'b) list	
	BatList.dropwhile : ('a -> bool) -> 'a list -> 'a list	
	BatList.enum : 'a list -> 'a BatEnum.t	
	BatList.eq : 'a BatOrd.eq -> 'a list BatOrd.eq	
	BatList.favg : float list -> float	
	BatList.filter_map : (int -> 'a -> 'b option) -> 'a list -> 'b list	
	BatList.find_exn : ('a -> bool) -> exn -> 'a list -> 'a	
	BatList.find_map : ('a -> 'b option) -> 'a list -> 'b	
	BatList.findi : (int -> 'a -> bool) -> 'a list -> int * 'a	
	BatList.first : 'a list -> 'a	
	BatList.fold : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a	
	BatList.fold_righti : (int -> 'b -> 'a -> 'a) -> 'b list -> 'a -> 'a	
	BatList.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc * 'a list	
	BatList.frange : float -> [< 'Downto `To] -> float -> int -> float list	
	BatList.fsum : float list -> float	
	BatList.group : ('a -> 'a -> int) -> 'a list -> 'a list list	
	BatList.group_consecutive : ('a -> 'a -> bool) -> 'a list -> 'a list list	
	BatList.index_of : 'a -> 'a list -> int option	
	BatList.index_ofq : 'a -> 'a list -> int option	
	BatList.iter2i : (int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit	
	BatList.kahan_sum : float list -> float	
	BatList.last : 'a list -> 'a	
	BatList.make : int -> 'a -> 'a list	
	BatList.map2i : (int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	
	BatList.max : 'a list -> 'a	
	BatList.mem_cmp : ('a -> 'a -> int) -> 'a -> 'a list -> bool	
	BatList.min : 'a list -> 'a	

Containers	Batteries	Base
	BatList.min_max : ?cmp:(a -> 'a -> int) -> 'a list -> 'a * 'a	
	BatList.modify : 'a -> ('b -> 'b) -> ('a * 'b) list -> ('a * 'b) list	
	BatList.modify_at : int -> ('a -> 'a) -> 'a list -> 'a list	
	BatList.modify_def : 'b -> 'a -> ('b -> 'b) -> ('a * 'b) list -> ('a * 'b) list	
	BatList.modify_opt : 'a -> ('b option -> 'b option) -> ('a * 'b) list -> ('a * 'b) list	
	BatList.modify_opt_at : int -> ('a -> 'a option) -> 'a list -> 'a list	
	BatList.nsplit : ('a -> bool) -> 'a list -> 'a list list	
	BatList.ntake : int -> 'a list -> 'a list list	
	BatList.of_backwards : 'a BatEnum.t -> 'a list	
	BatList.of_enum : 'a BatEnum.t -> 'a list	
	BatList.ord : 'a BatOrd.ord -> 'a list BatOrd.ord	
	BatList.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b list -> unit	
	BatList.range : int -> [< `Downto `To] -> int -> int list	
	BatList.remove : 'a list -> 'a -> 'a list	
	BatList.remove_all : 'a list -> 'a -> 'a list	
	BatList.remove_if : ('a -> bool) -> 'a list -> 'a list	
	BatList.rfind : ('a -> bool) -> 'a list -> 'a	
	BatList.rindex_of : 'a -> 'a list -> int option	
	BatList.rindex_ofq : 'a -> 'a list -> int option	
	BatList.shuffle : ?state:Random.State.t -> 'a list -> 'a list	
	BatList.singleton : 'a -> 'a list	
	BatList.sort_unique : ('a -> 'a -> int) -> 'a list -> 'a list	
	BatList.span : ('a -> bool) -> 'a list -> 'a list * 'a list	
	BatList.split_at : int -> 'a list -> 'a list * 'a list	
	BatList.split_nth : int -> 'a list -> 'a list * 'a list	
	BatList.subset : ('a -> 'b -> int) -> 'a list -> 'b list -> bool	
	BatList.sum : int list -> int	
	BatList.takedown : int -> 'a list -> 'a list * 'a list	
	BatList.takewhile : ('a -> bool) -> 'a list -> 'a list	
	BatList.transpose : 'a list list -> 'a list list	
	BatList.unfold : 'b -> ('b -> ('a * 'b) option) -> 'a list	
	BatList.unfold_exc : (unit -> 'a) -> 'a list * exn	
	BatList.unfold_exn : (unit -> 'a) -> 'a list * exn	
	BatList.unique_cmp : ?cmp:(a -> 'a -> int) -> 'a list -> 'a list	
	BatList.unique_hash : ?hash:(a -> int) -> ?eq:(a -> 'a -> bool) -> 'a list -> 'a list	
	BatMap.(~>) : ('a, 'b) map -> 'a -> 'b	
	BatMap.(<~) : ('a, 'b) map -> 'a * 'b -> ('a, 'b) map	
	BatMap.add_carry : 'a -> 'b -> ('a, 'b) map -> ('a, 'b) map * 'b option	
	BatMap.any : ('key, 'a) map -> 'key * 'a	
	BatMap.at_rank_exn : int -> ('key, 'a) map -> 'key * 'a	
	BatMap.backwards : ('a, 'b) map -> ('a * 'b) BatEnum.t	

Containers	Batteries	Base
	BatMap.diff : ('a, 'b) map -> ('a, 'b) map -> ('a, 'b) map	
	BatMap.enum : ('a, 'b) map -> ('a * 'b) BatEnum.t	
	BatMap.extract : 'a -> ('a, 'b) map -> 'b * ('a, 'b) map	
	BatMap.filterv : ('a -> bool) -> ('key, 'a) map -> ('key, 'a) map	
	BatMap.find_default : 'b -> 'a -> ('a, 'b) map -> 'b	
	BatMap.foldi : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) map -> 'c -> 'c	
	BatMap.intersect : ('b -> 'c -> 'd) -> ('a, 'b) map -> ('a, 'c) map -> ('a, 'd) map	
	BatMap.modify : 'a -> ('b -> 'b) -> ('a, 'b) map -> ('a, 'b) map	
	BatMap.modify_def : 'b -> 'a -> ('b -> 'b) -> ('a, 'b) map -> ('a, 'b) map	
	BatMap.modify_opt : 'a -> ('b option -> 'b option) -> ('a, 'b) map -> ('a, 'b) map	
	BatMap.of_enum : ('a * 'b) BatEnum.t -> ('a, 'b) map	
	BatMap.pop : ('a, 'b) map -> ('a * 'b) * ('a, 'b) map	
	BatMap.pop_max_binding : ('key, 'a) map -> ('key * 'a) * ('key, 'a) map	
	BatMap.pop_min_binding : ('key, 'a) map -> ('key * 'a) * ('key, 'a) map	
	BatMap.print : ?first:string -> ?last:string -> ?sep:string -> ?kvsep:string -> ('a BatInnerIO.output -> 'b -> unit) -> ('a BatInnerIO.output -> 'c -> unit) -> 'a BatInnerIO.output -> ('b, 'c) map -> unit	
	BatMap.remove_exn : 'a -> ('a, 'b) map -> ('a, 'b) map	
	BatMap.union_stdlib : ('key -> 'a -> 'a -> 'a option) -> ('key, 'a) map -> ('key, 'a) map -> ('key, 'a) map	
	BatMap.update_stdlib : 'a -> ('b option -> 'b option) -> ('a, 'b) map -> ('a, 'b) map	
	BatOption.(!?) : 'a option -> 'a -> 'a	
	BatOption.Labels.map : f:(a -> 'b) -> 'a option -> 'b option	
	BatOption.Labels.map_default : f:(a -> 'b) -> 'b -> 'a option -> 'b	
	BatOption.Labels.may : f:(a -> unit) -> 'a option -> unit	
	BatOption.apply : ('a -> 'a) option -> 'a -> 'a	
	BatOption.default : 'a -> 'a option -> 'a	
	BatOption.default_delayed : (unit -> 'a) -> 'a option -> 'a	
	BatOption.enum : 'a option -> 'a BatEnum.t	
	BatOption.eq : ?eq:(a -> 'a -> bool) -> 'a option -> 'a option -> bool	
	BatOption.get_exn : 'a option -> exn -> 'a	
	BatOption.map_default : ('a -> 'b) -> 'b -> 'a option -> 'b	
	BatOption.map_default_delayed : ('a -> 'b) -> (unit -> 'b) -> 'a option -> 'b	
	BatOption.may : ('a -> unit) -> 'a option -> unit	
	BatOption.of_enum : 'a BatEnum.t -> 'a option	
	BatOption.ord : 'a BatOrd.ord -> 'a option BatOrd.ord	
	BatOption.print : ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b option -> unit	
	BatPrintf.bprintf2 : Buffer.t -> ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	
	BatPrintf.kbprintf2 : (Buffer.t -> 'b) -> Buffer.t -> ('c, 'a BatInnerIO.output, unit, 'b) format4 -> 'c	
	BatPrintf.ksprintf2 : (string -> 'b) -> ('c, 'a BatInnerIO.output, unit, 'b) format4 -> 'c	

Containers	Batteries	Base
	BatPrintf.sprintf2 : ('a, 'b BatInnerIO.output, unit, string) format4 -> 'a	
	BatResult.catch : ('a -> 'e) -> 'a -> ('e, exn) result	
	BatResult.catch2 : ('a -> 'b -> 'c) -> 'a -> 'b -> ('c, exn) result	
	BatResult.catch3 : ('a -> 'b -> 'c -> 'd) -> 'a -> 'b -> 'c -> ('d, exn) result	
	BatResult.default : 'a -> ('a, 'b) result -> 'a	
	BatResult.get : ('a, exn) result -> 'a	
	BatResult.is_bad : ('a, 'e) result -> bool	
	BatResult.is_exn : exn -> ('a, exn) result -> bool	
	BatResult.map_both : ('a1 -> 'a2) -> ('b1 -> 'b2) -> ('a1, 'b1) result -> ('a2, 'b2) result	
	BatResult.map_default : 'b -> ('a -> 'b) -> ('a, 'c) result -> 'b	
	BatResult.of_option : 'a option -> ('a, unit) result	
	BatResult.print : ('b BatInnerIO.output -> 'a -> unit) -> 'b BatInnerIO.output -> ('a, exn) result -> unit	
	BatSeq.(--): int -> int -> int Seq.t	
	BatSeq.(--.): float * float -> float -> float Seq.t	
	BatSeq.(--~): char -> char -> char Seq.t	
	BatSeq.(//): 'a Seq.t -> ('a -> bool) -> 'a Seq.t	
	BatSeq.(//@): 'a Seq.t -> ('a -> 'b option) -> 'b Seq.t	
	BatSeq.(/@): 'a Seq.t -> ('a -> 'b) -> 'b Seq.t	
	BatSeq.(@/): ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	
	BatSeq.(@//): ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	
	BatSeq.assoc : 'a -> ('a * 'b) Seq.t -> 'b option	
	BatSeq.at : 'a Seq.t -> int -> 'a	
	BatSeq.combine : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t	
	BatSeq.enum : 'a Seq.t -> 'a BatEnum.t	
	BatSeq.first : 'a Seq.t -> 'a	
	BatSeq.fold_right : ('a -> 'b -> 'b) -> 'a Seq.t -> 'b -> 'b	
	BatSeq.last : 'a Seq.t -> 'a	
	BatSeq.make : int -> 'a -> 'a Seq.t	
	BatSeq.max : 'a Seq.t -> 'a	
	BatSeq.mem : 'a -> 'a Seq.t -> bool	
	BatSeq.min : 'a Seq.t -> 'a	
	BatSeq.of_string : ?first:string -> ?last:string -> ?sep:string -> (string -> 'a) -> string -> 'a Seq.t	
	BatSeq.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b Seq.t -> unit	
	BatSeq.split : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t	
	BatSeq.to_buffer : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> Buffer.t -> (unit -> 'a BatSeq.node) -> unit	
	BatSeq.to_string : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> 'a Seq.t -> string	
	BatSet.any : 'a set -> 'a	
	BatSet.at_rank_exn : int -> 'a set -> 'a	

Containers	Batteries	Base
	BatSet.backwards : 'a set -> 'a BatEnum.t	
	BatSet.cartesian_product : 'a set -> 'b set -> ('a * 'b) set	
	BatSet.enum : 'a set -> 'a BatEnum.t	
	BatSet.filter_map_endo : ('a -> 'a option) -> 'a set -> 'a set	
	BatSet.map_endo : ('a -> 'a) -> 'a set -> 'a set	
	BatSet.of_enum : 'a BatEnum.t -> 'a set	
	BatSet.pop : 'a set -> 'a * 'a set	
	BatSet.pop_max : 'a set -> 'a * 'a set	
	BatSet.pop_min : 'a set -> 'a * 'a set	
	BatSet.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'c -> unit) -> 'a BatInnerIO.output -> 'c set -> unit	
	BatSet.remove_exn : 'a -> 'a set -> 'a set	
	BatSet.split_le : 'a -> 'a set -> 'a set * 'a set	
	BatSet.split_lt : 'a -> 'a set -> 'a set * 'a set	
	BatSet.update : 'a -> 'a -> 'a set -> 'a set	
	BatString.backwards : string -> char BatEnum.t	
	BatString.chop : ?l:int -> ?r:int -> string -> string	
	BatString.count_char : string -> char -> int	
	BatString.count_string : string -> string -> int	
	BatString.cut_on_char : char -> int -> string -> string	
	BatString.ends_with : string -> string -> bool	
	BatString.enum : string -> char BatEnum.t	
	BatString.exists : string -> string -> bool	
	BatString.explode : string -> char list	
	BatString.find_from : string -> int -> string -> int	
	BatString.fold_lefti : ('a -> int -> char -> 'a) -> 'a -> string -> 'a	
	BatString.fold_right : (char -> 'a -> 'a) -> string -> 'a -> 'a	
	BatString.fold_righti : (int -> char -> 'a -> 'a) -> string -> 'a -> 'a	
	BatString.head : string -> int -> string	
	BatString.icompare : string -> string -> int	
	BatString.implode : char list -> string	
	BatString.in_place_mirror : bytes -> unit	
	BatString.index_after_n : char -> int -> string -> int	
	BatString.join : string -> string list -> string	
	BatString.lchop : ?n:int -> string -> string	
	BatString.left : string -> int -> string	
	BatString.nreplace : str:string -> sub:string -> by:string -> string	
	BatString.nsplit : string -> by:string -> string list	
	BatString.numeric_compare : string -> string -> int	
	BatString.of_backwards : char BatEnum.t -> string	
	BatString.of_enum : char BatEnum.t -> string	
	BatString.of_float : float -> string	

Containers	Batteries	Base
	BatString.of_int : int -> string	
	BatString.ord : string -> string -> BatOrd.order	
	BatString.print : 'a BatInnerIO.output -> string -> unit	
	BatString.print_quoted : 'a BatInnerIO.output -> string -> unit	
	BatString.println : 'a BatInnerIO.output -> string -> unit	
	BatString.quote : string -> string	
	BatString.rchop : ?n:int -> string -> string	
	BatString.replace : str:string -> sub:string -> by:string -> bool * string	
	BatString.replace_chars : (char -> string) -> string -> string	
	BatString.rev_in_place : bytes -> unit	
	BatString.rfind_from : string -> int -> string -> int	
	BatString.right : string -> int -> string	
	BatString.rsplit : string -> by:string -> string * string	
	BatString.slice : ?first:int -> ?last:int -> string -> string	
	BatString.splice : string -> int -> int -> string -> string	
	BatString.split : string -> by:string -> string * string	
	BatString.split_on_string : by:string -> string -> string list	
	BatString.starts_with : string -> string -> bool	
	BatString.tail : string -> int -> string	
	BatString.to_float : string -> float	
	BatString.to_int : string -> int	
		Base.Array.binary_search : ('a array, 'a, 'key) Base.Binary_searchable_intf.binary_search
		Base.Array.binary_search_segmented : ('a array, 'a) Base.Binary_searchable_intf.binary_search_segmented
		Base.Array.blit : ('a array, 'a array) Base.Blit_intf.blit
		Base.Array.concat_map : 'a array -> f:('a -> 'b array) -> 'b array
		Base.Array.concat_mapi : 'a array -> f:(int -> 'a -> 'b array) -> 'b array
		Base.Array.counti : 'a array -> f:(int -> 'a -> bool) -> int
		Base.Array.existsi : 'a array -> f:(int -> 'a -> bool) -> bool
		Base.Array.filter_mapi : 'a array -> f:(int -> 'a -> 'b option) -> 'b array
		Base.Array.filter_opt : 'a option array -> 'a array
		Base.Array.filteri : 'a array -> f:(int -> 'a -> bool) -> 'a array
		Base.Array.find : 'a array -> f:('a -> bool) -> 'a option
		Base.Array.find_consecutive_duplicate : 'a array -> equal:('a -> 'a -> bool) -> ('a * 'a) option
		Base.Array.find_map : 'a array -> f:('a -> 'b option) -> 'b option
		Base.Array.find_map_exn : 'a array -> f:('a -> 'b option) -> 'b
		Base.Array.find_mapi : 'a array -> f:(int -> 'a -> 'b option) -> 'b option
		Base.Array.find_mapi_exn : 'a array -> f:(int -> 'a -> 'b option) -> 'b
		Base.Array.findi : 'a array -> f:(int -> 'a -> bool) -> (int * 'a) option
		Base.Array.findi_exn : 'a array -> f:(int -> 'a -> bool) -> int * 'a
		Base.Array.fold_mapi : 'a array -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'b * 'c array
		Base.Array.fold_result : 'a array -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t

Containers	Batteries	Base
		Base.Array.fold_until : 'a array -> init:'accum -> f:(('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:(('accum -> 'final) -> 'final)
		Base.Array.folding_map : 'a array -> init:'b -> f:(('b -> 'a -> 'b * 'c) -> 'c array
		Base.Array.folding_mapi : 'a array -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c array
		Base.Array.for_alli : 'a array -> f:(int -> 'a -> bool) -> bool
		Base.Array.invariant : 'a Base.Invariant_intf.inv -> 'a array Base.Invariant_intf.inv
		Base.Array.is_empty : 'a array -> bool
		Base.Array.is_sorted : 'a array -> compare:(('a -> 'a -> int) -> bool
		Base.Array.is_sorted_strictly : 'a array -> compare:(('a -> 'a -> int) -> bool
		Base.Array.last : 'a array -> 'a
		Base.Array.map_inplace : 'a array -> f:(('a -> 'a) -> unit
		Base.Array.max_elt : 'a array -> compare:(('a -> 'a -> int) -> 'a option
		Base.Array.max_length : int = 18014398509481983
		Base.Array.min_elt : 'a array -> compare:(('a -> 'a -> int) -> 'a option
		Base.Array.of_list_map : 'a list -> f:(('a -> 'b) -> 'b array
		Base.Array.of_list_mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b array
		Base.Array.of_list_rev : 'a list -> 'a array
		Base.Array.of_list_rev_map : 'a list -> f:(('a -> 'b) -> 'b array
		Base.Array.of_list_rev_mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b array
		Base.Array.partition_tf : 'a array -> f:(int -> 'a -> bool) -> 'a array * 'a array
		Base.Array.random_element : ?random_state:Base.Random.State.t -> 'a array -> 'a option
		Base.Array.reduce : 'a array -> f:(('a -> 'a -> 'a) -> 'a option
		Base.Array.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> 'a array -> Sexplib0.Sexp.t
		Base.Array.subo : ('a array, 'a array) Base.Blit_intf.subo
		Base.Array.sum : (module Base.Container_intf.Summable with type t = 'sum) -> 'a array -> f:(('a -> 'sum) -> 'sum
		Base.Array.t_of_sexp : (Sexplib0.Sexp.t -> 'a) -> Sexplib0.Sexp.t -> 'a array
		Base.Array.t_sexp_grammar : Base.Ppx_sexp_conv_lib.Sexp.Private.Raw_grammar.t...
		Base.Array.to_array : 'a array -> 'a array
		Base.Array.to_sequence : 'a array -> 'a Base.Sequence.t
		Base.Array.to_sequence_mutable : 'a array -> 'a Base.Sequence.t
		Base.Array.transpose : 'a array array -> 'a array array option
		Base.Array.transpose_exn : 'a array array -> 'a array array
		Base.Array.unsafe_blit : ('a array, 'a array) Base.Blit.blit
		Base.Array.zip : 'a array -> 'b array -> ('a * 'b) array option
		Base.Array.zip_exn : 'a array -> 'b array -> ('a * 'b) array
		Base.List.Assoc.add : ('a, 'b) Base.List.Assoc.t -> equal:(('a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) Base.List.Assoc.t
		Base.List.Assoc.find : ('a, 'b) Base.List.Assoc.t -> equal:(('a -> 'a -> bool) -> 'a -> 'b option
		Base.List.Assoc.find_exn : ('a, 'b) Base.List.Assoc.t -> equal:(('a -> 'a -> bool) -> 'a -> 'b
		Base.List.Assoc.inverse : ('a, 'b) Base.List.Assoc.t -> ('b, 'a) Base.List.Assoc.t
		Base.List.Assoc.map : ('a, 'b) Base.List.Assoc.t -> f:(('b -> 'c) -> ('a, 'c) Base.List.Assoc.t
		Base.List.Assoc.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> ('b -> Sexplib0.Sexp.t) -> ('a, 'b) Base.List.Assoc.t -> Sexplib0.Sexp.t

Containers	Batteries	Base
		Base.List.Assoc.t_of_sexp : (Sexplib0.Sexp.t -> 'a) -> (Sexplib0.Sexp.t -> 'b) -> Sexplib0.Sexp.t -> ('a, 'b) Base.List.Assoc.t
		Base.List.all_unit : unit list list -> unit list
		Base.List.bind : 'a list -> f:('a -> 'b list) -> 'b list
		Base.List.concat_mapi : 'a list -> f:(int -> 'a -> 'b list) -> 'b list
		Base.List.concat_no_order : 'a list list -> 'a list
		Base.List.contains_dup : compare:('a -> 'a -> int) -> 'a list -> bool
		Base.List.counti : 'a list -> f:(int -> 'a -> bool) -> int
		Base.List.dedup_and_sort : compare:('a -> 'a -> int) -> 'a list -> 'a list
		Base.List.drop_last : 'a list -> 'a list option
		Base.List.drop_last_exn : 'a list -> 'a list
		Base.List.exists2 : 'a list -> 'b list -> f:('a -> 'b -> bool) -> bool Base.List.Or_unequal_lengths.t
		Base.List.existsi : 'a list -> f:(int -> 'a -> bool) -> bool
		Base.List.filter_mapi : 'a list -> f:(int -> 'a -> 'b option) -> 'b list
		Base.List.filter_opt : 'a option list -> 'a list
		Base.List.find_a_dup : compare:('a -> 'a -> int) -> 'a list -> 'a option
		Base.List.find_all_dups : compare:('a -> 'a -> int) -> 'a list -> 'a list
		Base.List.find_consecutive_duplicate : 'a list -> equal:('a -> 'a -> bool) -> ('a * 'a) option
		Base.List.find_map_exn : 'a list -> f:('a -> 'b option) -> 'b
		Base.List.find_mapi_exn : 'a list -> f:(int -> 'a -> 'b option) -> 'b
		Base.List.findi : 'a list -> f:(int -> 'a -> bool) -> (int * 'a) option
		Base.List.fold2 : 'a list -> 'b list -> init:'c -> f:('c -> 'a -> 'b -> 'c) -> 'c Base.List.Or_unequal_lengths.t
		Base.List.fold2_exn : 'a list -> 'b list -> init:'c -> f:('c -> 'a -> 'b -> 'c) -> 'c
		Base.List.fold_result : 'a list -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		Base.List.fold_until : 'a list -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
		Base.List.folding_map : 'a list -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'c list
		Base.List.folding_mapi : 'a list -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c list
		Base.List.for_all2 : 'a list -> 'b list -> f:('a -> 'b -> bool) -> bool Base.List.Or_unequal_lengths.t
		Base.List.for_alli : 'a list -> f:(int -> 'a -> bool) -> bool
		Base.List.groupi : 'a list -> break:(int -> 'a -> 'a -> bool) -> 'a list list
		Base.List.hash_fold_t : (Base.Ppx_hash_lib.Std.Hash.state -> 'a -> Base.Ppx_hash_lib.Std.Hash.state) -> Base.Ppx_hash_lib.Std.Hash.state -> 'a list -> Base.Ppx_hash_lib.Std.Hash.state
		Base.List.ignore_m : 'a list -> unit list
		Base.List.invariant : 'a Base.Invariant_intf.inv -> 'a list Base.Invariant_intf.inv
		Base.List.is_prefix : 'a list -> prefix:'a list -> equal:('a -> 'a -> bool) -> bool
		Base.List.is_sorted_strictly : 'a list -> compare:('a -> 'a -> int) -> bool
		Base.List.is_suffix : 'a list -> suffix:'a list -> equal:('a -> 'a -> bool) -> bool
		Base.List.iter2 : 'a list -> 'b list -> f:('a -> 'b -> unit) -> unit Base.List.Or_unequal_lengths.t
		Base.List.join : 'a list list -> 'a list
		Base.List.map2 : 'a list -> 'b list -> f:('a -> 'b -> 'c) -> 'c list Base.List.Or_unequal_lengths.t
		Base.List.map3 : 'a list -> 'b list -> 'c list -> f:('a -> 'b -> 'c -> 'd) -> 'd list Base.List.Or_unequal_lengths.t

Containers	Batteries	Base
		Base.List.map3_exn : 'a list -> 'b list -> 'c list -> f:('a -> 'b -> 'c -> 'd) -> 'd list
		Base.List.max_elt : 'a list -> compare:('a -> 'a -> int) -> 'a option
		Base.List.min_elt : 'a list -> compare:('a -> 'a -> int) -> 'a option
		Base.List.of_list : 'a list -> 'a list
		Base.List.partition3_map : 'a list -> f:('a -> ['Fst of 'b 'Snd of 'c 'Trd of 'd]) -> 'b list * 'c list * 'd list
		Base.List.partition_result : ('ok, 'error) Base.Result.t list -> 'ok list * 'error list
		Base.List.permute : ?random_state:Base.Random.State.t -> 'a list -> 'a list
		Base.List.random_element : ?random_state:Base.Random.State.t -> 'a list -> 'a option
		Base.List.random_element_exn : ?random_state:Base.Random.State.t -> 'a list -> 'a
		Base.List.range : compare:('a -> 'a -> int) -> stride:('a -> 'a) -> start:['exclusive 'inclusive] -> stop:['exclusive 'inclusive] -> 'a -> 'a -> 'a list
		Base.List.reduce_balanced : 'a list -> f:('a -> 'a -> 'a) -> 'a option
		Base.List.reduce_balanced_exn : 'a list -> f:('a -> 'a -> 'a) -> 'a
		Base.List.remove_consecutive_duplicates : ?which_to_keep:['First 'Last] -> 'a list -> equal:('a -> 'a -> bool) -> 'a list
		Base.List.rev_filter : 'a list -> f:('a -> bool) -> 'a list
		Base.List.rev_filter_map : 'a list -> f:('a -> 'b option) -> 'b list
		Base.List.rev_filter_mapi : 'a list -> f:(int -> 'a -> 'b option) -> 'b list
		Base.List.rev_map2 : 'a list -> 'b list -> f:('a -> 'b -> 'c) -> 'c list Base.List.Or_unequal_lengths.t
		Base.List.rev_map3 : 'a list -> 'b list -> 'c list -> f:('a -> 'b -> 'c -> 'd) -> 'd list Base.List.Or_unequal_lengths.t
		Base.List.rev_map3_exn : 'a list -> 'b list -> 'c list -> f:('a -> 'b -> 'c -> 'd) -> 'd list
		Base.List.rev_map_append : 'a list -> 'b list -> f:('a -> 'b) -> 'b list
		Base.List.rev_mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b list
		Base.List.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> 'a list -> Sexplib0.Sexp.t
		Base.List.split_n : 'a list -> int -> 'a list * 'a list
		Base.List.split_while : 'a list -> f:('a -> bool) -> 'a list * 'a list
		Base.List.sub : 'a list -> pos:int -> len:int -> 'a list
		Base.List.sum : (module Base.Container_intf.Summable with type t = 'sum) -> 'a list -> f:('a -> 'sum) -> 'sum
		Base.List.t_of_sexp : (Sexplib0.Sexp.t -> 'a) -> Sexplib0.Sexp.t -> 'a list
		Base.List.t_sexp_grammar : Base.Ppx_sexp_conv_lib.Sexp.Private.Raw_grammar.t...
		Base.List.tl : 'a list -> 'a list option
		Base.List.to_array : 'a list -> 'a array
		Base.List.to_list : 'a list -> 'a list
		Base.List.transpose : 'a list list -> 'a list list option
		Base.List.unordered_append : 'a list -> 'a list -> 'a list
		Base.List.unzip3 : ('a * 'b * 'c) list -> 'a list * 'b list * 'c list
		Base.List.zip : 'a list -> 'b list -> ('a * 'b) list Base.List.Or_unequal_lengths.t
		Base.Map.add : ('k, 'v, 'cmp) map -> key: 'k -> data: 'v -> ('k, 'v, 'cmp) map Base.Map.Or_duplicate.t
		Base.Map.add_multi : ('k, 'v list, 'cmp) map -> key: 'k -> data: 'v -> ('k, 'v list, 'cmp) map
		Base.Map.append : lower_part:('k, 'v, 'cmp) map -> upper_part:('k, 'v, 'cmp) map -> ['Ok of ('k, 'v, 'cmp) map 'Overlapping_key_ranges]
		Base.Map.binary_search : ('k, 'v, 'cmp) map -> compare:(key: 'k -> data: 'v -> 'key -> int) -> ['First_equal_to 'First_greater_than_or_equal_to 'First_strictly_greater_than 'Last_equal_to 'Last_less_than_or_equal_to 'Last_strictly_less_than] -> 'key -> ('k * 'v) option

Containers	Batteries	Base
		Base.Map.binary_search_segmented : ('k, 'v, 'cmp) map -> segment_of:(key:'k -> data:'v -> ['Left 'Right]) -> ['First_on_right 'Last_on_left] -> ('k * 'v) option
		Base.Map.change : ('k, 'v, 'cmp) map -> 'k -> f:(v option -> 'v option) -> ('k, 'v, 'cmp) map
		Base.Map.closest_key : ('k, 'v, 'cmp) map -> ['Greater_or_equal_to 'Greater_than 'Less_or_equal_to 'Less_than] -> 'k -> ('k * 'v) option
		Base.Map.combine_errors : ('k, 'v Base.Or_error.t, 'cmp) map -> ('k, 'v, 'cmp) map Base.Or_error.t
		Base.Map.comparator : ('a, 'b, 'cmp) map -> ('a, 'cmp) Base.Comparator.t
		Base.Map.comparator_s : ('a, 'b, 'cmp) map -> ('a, 'cmp) Base.Map.comparator
		Base.Map.compare_m__t : (module Base.Map.Compare_m) -> ('v -> 'v -> int) -> ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> int
		Base.Map.count : ('k, 'v, 'a) map -> f:(v -> bool) -> int
		Base.Map.counti : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> bool) -> int
		Base.Map.data : ('a, 'v, 'b) map -> 'v list
		Base.Map.equal_m__t : (module Base.Map.Equal_m) -> ('v -> 'v -> bool) -> ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> bool
		Base.Map.existsi : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> bool) -> bool
		Base.Map.filter_keys : ('k, 'v, 'cmp) map -> f:(k -> bool) -> ('k, 'v, 'cmp) map
		Base.Map.filter_mapi : ('k, 'v1, 'cmp) map -> f:(key:'k -> data:'v1 -> 'v2 option) -> ('k, 'v2, 'cmp) map
		Base.Map.filteri : ('k, 'v, 'cmp) map -> f:(key:'k -> data:'v -> bool) -> ('k, 'v, 'cmp) map
		Base.Map.find : ('k, 'v, 'cmp) map -> 'k -> 'v option
		Base.Map.find_multi : ('k, 'v list, 'cmp) map -> 'k -> 'v list
		Base.Map.fold2 : ('k, 'v1, 'cmp) map -> ('k, 'v2, 'cmp) map -> init:'a -> f:(key:'k -> data:['Both of 'v1 * 'v2 'Left of 'v1 'Right of 'v2] -> 'a -> 'a) -> 'a
		Base.Map.fold_range_inclusive : ('k, 'v, 'cmp) map -> min:'k -> max:'k -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a
		Base.Map.fold_right : ('k, 'v, 'b) map -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a
		Base.Map.fold_symmetric_diff : ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> data_equal:(v -> v -> bool) -> init:'a -> f:(a -> ('k, 'v) Base.Map.Symmetric_diff_element.t -> 'a) -> 'a
		Base.Map.for_alli : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> bool) -> bool
		Base.Map.hash_fold_direct : 'k Base.Hash.folder -> 'v Base.Hash.folder -> ('k, 'v, 'cmp) map Base.Hash.folder
		Base.Map.hash_fold_m__t : (module Base.Map.Hash_fold_m with type t = 'k) -> (Base.Hash.state -> 'v -> Base.Hash.state) -> Base.Hash.state -> ('k, 'v, 'a) map -> Base.Hash.state
		Base.Map.invariants : ('a, 'b, 'c) map -> bool
		Base.Map.iter2 : ('k, 'v1, 'cmp) map -> ('k, 'v2, 'cmp) map -> f:(key:'k -> data:['Both of 'v1 * 'v2 'Left of 'v1 'Right of 'v2] -> unit) -> unit
		Base.Map.iter_keys : ('k, 'a, 'b) map -> f:(k -> unit) -> unit
		Base.Map.iteri : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> unit) -> unit
		Base.Map.iteri_until : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> Base.Map.Continue_or_stop.t) -> Base.Map.Finished_or_unfinished.t
		Base.Map.length : ('a, 'b, 'c) map -> int
		Base.Map.m__t_of_sexp : (module Base.Map.M_of_sexp with type comparator_witness = 'cmp and type t = 'k) -> (Base.Sexp.t -> 'v) -> Base.Sexp.t -> ('k, 'v, 'cmp) map
		Base.Map.m__t_sexp_grammar : Base.Ppx_sexp_conv_lib.Sexp.Private.Raw_grammar.t ...)
		Base.Map.merge : ('k, 'v1, 'cmp) map -> ('k, 'v2, 'cmp) map -> f:(key:'k -> ['Both of 'v1 * 'v2 'Left of 'v1 'Right of 'v2] -> 'v3 option) -> ('k, 'v3, 'cmp) map
		Base.Map.merge_skewed : ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> combine:(key:'k -> 'v -> 'v -> 'v) -> ('k, 'v, 'cmp) map
		Base.Map.nth : ('k, 'v, 'a) map -> int -> ('k * 'v) option

Containers	Batteries	Base
		Base.Map.nth_exn : ('k, 'v, 'a) map -> int -> 'k * 'v
		Base.Map.of_alist : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> [`Duplicate_key of 'a `Ok of ('a, 'b, 'cmp) map]
		Base.Map.of_alist_exn : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> ('a, 'b, 'cmp) map
		Base.Map.of_alist_fold : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> init:'c -> f:(c -> 'b -> 'c) -> ('a, 'c, 'cmp) map
		Base.Map.of_alist_multi : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> ('a, 'b list, 'cmp) map
		Base.Map.of_alist_or_error : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> ('a, 'b, 'cmp) map Base.Or_error.t
		Base.Map.of_alist_reduce : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) list -> f:(b -> 'b -> 'b) -> ('a, 'b, 'cmp) map
		Base.Map.of_increasing_iterator_unchecked : ('a, 'cmp) Base.Map.comparator -> len:int -> f:(int -> 'a * 'b) -> ('a, 'b, 'cmp) map
		Base.Map.of_increasing_sequence : ('k, 'cmp) Base.Map.comparator -> ('k * 'v) Base.Sequence.t -> ('k, 'v, 'cmp) map Base.Or_error.t
		Base.Map.of_iteri : ('a, 'cmp) Base.Map.comparator -> iteri:(f:(key:'a -> data:'b -> unit) -> unit) -> [`Duplicate_key of 'a `Ok of ('a, 'b, 'cmp) map]
		Base.Map.of_sequence : ('k, 'cmp) Base.Map.comparator -> ('k * 'v) Base.Sequence.t -> [`Duplicate_key of 'k `Ok of ('k, 'v, 'cmp) map]
		Base.Map.of_sequence_exn : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) Base.Sequence.t -> ('a, 'b, 'cmp) map
		Base.Map.of_sequence_fold : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) Base.Sequence.t -> init:'c -> f:(c -> 'b -> 'c) -> ('a, 'c, 'cmp) map
		Base.Map.of_sequence_multi : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) Base.Sequence.t -> ('a, 'b list, 'cmp) map
		Base.Map.of_sequence_or_error : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) Base.Sequence.t -> ('a, 'b, 'cmp) map Base.Or_error.t
		Base.Map.of_sequence_reduce : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) Base.Sequence.t -> f:(b -> 'b -> 'b) -> ('a, 'b, 'cmp) map
		Base.Map.of_sorted_array : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) array -> ('a, 'b, 'cmp) map Base.Or_error.t
		Base.Map.of_sorted_array_unchecked : ('a, 'cmp) Base.Map.comparator -> ('a * 'b) array -> ('a, 'b, 'cmp) map
		Base.Map.partition_map : ('k, 'v1, 'cmp) map -> f:(v1 -> ('v2, 'v3) Base.Either.t) -> ('k, 'v2, 'cmp) map * ('k, 'v3, 'cmp) map
		Base.Map.partition_mapi : ('k, 'v1, 'cmp) map -> f:(key:'k -> data:'v1 -> ('v2, 'v3) Base.Either.t) -> ('k, 'v2, 'cmp) map * ('k, 'v3, 'cmp) map
		Base.Map.partition_tf : ('k, 'v, 'cmp) map -> f:(key:'k -> data:'v -> bool) -> ('k, 'v, 'cmp) map * ('k, 'v, 'cmp) map
		Base.Map.range_to_alist : ('k, 'v, 'cmp) map -> min:'k -> max:'k -> ('k * 'v) list
		Base.Map.rank : ('k, 'v, 'cmp) map -> 'k -> int option
		Base.Map.remove_multi : ('k, 'v list, 'cmp) map -> 'k -> ('k, 'v list, 'cmp) map
		Base.Map.set : ('k, 'v, 'cmp) map -> key:'k -> data:'v -> ('k, 'v, 'cmp) map
		Base.Map.sexp_of_m__t : (module Base.Map.Sexp_of_m with type t = 'k) -> ('v -> Base.Sexp.t) -> ('k, 'v, 'cmp) map -> Base.Sexp.t
		Base.Map.subrange : ('k, 'v, 'cmp) map -> lower_bound:'k Base.Maybe_bound.t -> upper_bound:'k Base.Maybe_bound.t -> ('k, 'v, 'cmp) map
		Base.Map.symmetric_diff : ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> data_equal:(v -> 'v -> bool) -> ('k, 'v) Base.Map.Symmetric_diff_element.t Base.Sequence.t
		Base.Map.to_alist : ?key_order:[`Decreasing `Increasing] -> ('k, 'v, 'a) map -> ('k * 'v) list
		Base.Map.to_sequence : ?order:[`Decreasing_key `Increasing_key] -> ?keys_greater_or_equal_to:'k -> ?keys_less_or_equal_to:'k -> ('k, 'v, 'cmp) map -> ('k * 'v) Base.Sequence.t
		Base.Map.validate : name:(k -> string) -> 'v Base.Validate.check -> ('k, 'v, 'a) map Base.Validate.check
		Base.Map.validatei : name:(k -> string) -> ('k * 'v) Base.Validate.check -> ('k, 'v, 'a) map Base.Validate.check
		Base.Option.all : 'a option list -> 'a list option
		Base.Option.all_unit : unit option list -> unit option

Containers	Batteries	Base
		Base.Option.both : 'a option -> 'b option -> ('a * 'b) option
		Base.Option.call : 'a -> f:('a -> unit) option -> unit
		Base.Option.count : 'a option -> f:('a -> bool) -> int
		Base.Option.find : 'a option -> f:('a -> bool) -> 'a option
		Base.Option.first_some : 'a option -> 'a option -> 'a option
		Base.Option.fold_result : 'a option -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		Base.Option.fold_until : 'a option -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
		Base.Option.hash_fold_t : (Base.Ppx_hash_lib.Std.Hash.state -> 'a -> Base.Ppx_hash_lib.Std.Hash.state) -> Base.Ppx_hash_lib.Std.Hash.state -> 'a option -> Base.Ppx_hash_lib.Std.Hash.state
		Base.Option.ignore_m : 'a option -> unit option
		Base.Option.invariant : 'a Base.Invariant_intf.inv -> 'a option Base.Invariant_intf.inv
		Base.Option.is_empty : 'a option -> bool
		Base.Option.length : 'a option -> int
		Base.Option.max_elt : 'a option -> compare:('a -> 'a -> int) -> 'a option
		Base.Option.mem : 'a option -> 'a -> equal:('a -> 'a -> bool) -> bool
		Base.Option.merge : 'a option -> 'a option -> f:('a -> 'a -> 'a) -> 'a option
		Base.Option.min_elt : 'a option -> compare:('a -> 'a -> int) -> 'a option
		Base.Option.sexp_of_t : ('a -> Sexplib0__Sexp.t) -> 'a option -> Sexplib0__Sexp.t
		Base.Option.some_if : bool -> 'a -> 'a option
		Base.Option.sum : (module Base.Container_intf.Summable with type t = 'sum) -> 'a option -> f:('a -> 'sum) -> 'sum
		Base.Option.t_of_sexp : (Sexplib0__Sexp.t -> 'a) -> Sexplib0__Sexp.t -> 'a option
		Base.Option.t_sexp_grammar : Base.Ppx_sexp_conv_lib...
		Base.Option.to_array : 'a option -> 'a array
		Base.Option.try_with : (unit -> 'a) -> 'a option
		Base.Option.try_with_join : (unit -> 'a option) -> 'a option
		Base.Option.validate : none:unit Base.Validate.check -> some:'a Base.Validate.check -> 'a option Base.Validate.check
		Base.Option.value_exn : ?here:Base.Source_code_position0.t -> ?error:Base.Error.t -> ?message:string -> 'a option -> 'a
		Base.Option.value_map : 'a option -> default:'b -> f:('a -> 'b) -> 'b
		Base.Option_array.blit : ('a Base.Option_array.t, 'a Base.Option_array.t) Base.Blit_intf.blit
		Base.Option_array.blito : ('a Base.Option_array.t, 'a Base.Option_array.t) Base.Blit_intf.blito
		Base.Option_array.clear : 'a Base.Option_array.t -> unit
		Base.Option_array.copy : 'a Base.Option_array.t -> 'a Base.Option_array.t
		Base.Option_array.create : len:int -> 'a Base.Option_array.t
		Base.Option_array.empty : 'a Base.Option_array.t
		Base.Option_array.get : 'a Base.Option_array.t -> int -> 'a option
		Base.Option_array.get_some_exn : 'a Base.Option_array.t -> int -> 'a
		Base.Option_array.init : int -> f:(int -> 'a option) -> 'a Base.Option_array.t
		Base.Option_array.init_some : int -> f:(int -> 'a) -> 'a Base.Option_array.t
		Base.Option_array.is_none : 'a Base.Option_array.t -> int -> bool
		Base.Option_array.is_some : 'a Base.Option_array.t -> int -> bool

Containers	Batteries	Base
		Base.Option_array.length : 'a Base.Option_array.t -> int
		Base.Option_array.set : 'a Base.Option_array.t -> int -> 'a option -> unit
		Base.Option_array.set_none : 'a Base.Option_array.t -> int -> unit
		Base.Option_array.set_some : 'a Base.Option_array.t -> int -> 'a -> unit
		Base.Option_array.sexp_of_t : ('a -> Sexplib0__Sexp.t) -> 'a Base.Option_array.t -> Sexplib0__Sexp.t
		Base.Option_array.sub : ('a Base.Option_array.t, 'a Base.Option_array.t) Base.Blit_intf.sub
		Base.Option_array.subo : ('a Base.Option_array.t, 'a Base.Option_array.t) Base.Blit_intf.subo
		Base.Option_array.swap : 'a Base.Option_array.t -> int -> int -> unit
		Base.Option_array.t_of_sexp : (Sexplib0__Sexp.t -> 'a) -> Sexplib0__Sexp.t -> 'a Base.Option_array.t
		Base.Option_array.unsafe_blit : ('a Base.Option_array.t, 'a Base.Option_array.t) Base.Blit_intf.blit
		Base.Option_array.unsafe_get : 'a Base.Option_array.t -> int -> 'a option
		Base.Option_array.unsafe_get_some_assuming_some : 'a Base.Option_array.t -> int -> 'a
		Base.Option_array.unsafe_get_some_exn : 'a Base.Option_array.t -> int -> 'a
		Base.Option_array.unsafe_is_some : 'a Base.Option_array.t -> int -> bool
		Base.Option_array.unsafe_set : 'a Base.Option_array.t -> int -> 'a option -> unit
		Base.Option_array.unsafe_set_none : 'a Base.Option_array.t -> int -> unit
		Base.Option_array.unsafe_set_some : 'a Base.Option_array.t -> int -> 'a -> unit
		Base.Printf.failwithf : ('r, unit, string, unit -> 'a) format4 -> 'r
		Base.Printf.invalid_argf : ('r, unit, string, unit -> 'a) format4 -> 'r
		Base.Result.all : ('a, 'e) result list -> ('a list, 'e) result
		Base.Result.all_unit : (unit, 'e) result list -> (unit, 'e) result
		Base.Result.combine : ('ok1, 'err) result -> ('ok2, 'err) result -> ok:('ok1 -> 'ok2 -> 'ok3) -> err:('err -> 'err -> 'err) -> ('ok3, 'err) result
		Base.Result.combine_errors : ('ok, 'err) result list -> ('ok list, 'err list) result
		Base.Result.combine_errors_unit : (unit, 'err) result list -> (unit, 'err list) result
		Base.Result.error : ('a, 'err) result -> 'err option
		Base.Result.failf : ('a, unit, string, ('b, string) result) format4 -> 'a
		Base.Result.hash_fold_t : (Base.Ppx_hash_lib.Std.Hash.state -> 'ok -> Base.Ppx_hash_lib.Std.Hash.state) -> (Base.Ppx_hash_lib.Std.Hash.state -> 'err -> Base.Ppx_hash_lib.Std.Hash.state) -> Base.Ppx_hash_lib.Std.Hash.state -> ('ok, 'err) result -> Base.Ppx_hash_lib.Std.Hash.state
		Base.Result.ignore_m : ('a, 'e) result -> (unit, 'e) result
		Base.Result.invariant : 'a Base.Invariant_intf.inv -> 'b Base.Invariant_intf.inv -> ('a, 'b) resultBase.Invariant_intf.inv
		Base.Result.of_either : ('ok, 'err) Base.Either0.t -> ('ok, 'err) result
		Base.Result.of_option : 'ok option -> error:'err -> ('ok, 'err) result
		Base.Result.ok_exn : ('ok, exn) result -> 'ok
		Base.Result.ok : ('ok, 'a) result -> 'ok option
		Base.Result.okfst : ('ok, 'err) result -> ('ok, 'err) Base.Either0.t
		Base.Result.ok_if_true : bool -> error:'err -> (unit, 'err) result
		Base.Result.ok_or_failwith : ('ok, string) result -> 'ok
		Base.Result.sexp_of_t : ('a -> Sexplib0__Sexp.t) -> ('b -> Sexplib0__Sexp.t) -> ('a, 'b) result -> Sexplib0__Sexp.t
		Base.Result.t_of_sexp : (Sexplib0__Sexp.t -> 'a) -> (Sexplib0__Sexp.t -> 'b) -> Sexplib0__Sexp.t -> ('a, 'b) result
		Base.Result.to_either : ('ok, 'err) result -> ('ok, 'err) Base.Either0.t

Containers	Batteries	Base
		Base.Result.try_with : (unit -> 'a) -> ('a, exn) result
		Base.Sequence.all : 'a Base.Sequence.t list -> 'a list Base.Sequence.t
		Base.Sequence.all_unit : unit Base.Sequence.t list -> unit Base.Sequence.t
		Base.Sequence.bind : 'a Base.Sequence.t -> f:(a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
		Base.Sequence.bounded_length : 'a Base.Sequence.t -> at_most:int -> ['Greater 'Is of int]
		Base.Sequence.cartesian_product : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * 'b) Base.Sequence.t
		Base.Sequence.chunks_exn : 'a Base.Sequence.t -> int -> 'a list Base.Sequence.t
		Base.Sequence.concat_map : 'a Base.Sequence.t -> f:(a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
		Base.Sequence.concat_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
		Base.Sequence.count : 'a Base.Sequence.t -> f:(a -> bool) -> int
		Base.Sequence.counti : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> int
		Base.Sequence.cycle_list_exn : 'a list -> 'a Base.Sequence.t
		Base.Sequence.delayed_fold : 'a Base.Sequence.t -> init:'s -> f:(s -> 'a -> k:(s -> 'r) -> 'r) -> finish:(s -> 'r) -> 'r
		Base.Sequence.drop_eagerly : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
		Base.Sequence.drop_while_option : 'a Base.Sequence.t -> f:(a -> bool) -> ('a * 'a Base.Sequence.t) option
		Base.Sequence.existsi : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> bool
		Base.Sequence.filter_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b option) -> 'b Base.Sequence.t
		Base.Sequence.filter_opt : 'a option Base.Sequence.t -> 'a Base.Sequence.t
		Base.Sequence.filteri : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> 'a Base.Sequence.t
		Base.Sequence.find_consecutive_duplicate : 'a Base.Sequence.t -> equal:(a -> 'a -> bool) -> ('a * 'a) option
		Base.Sequence.find_exn : 'a Base.Sequence.t -> f:(a -> bool) -> 'a
		Base.Sequence.find_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b option) -> 'b option
		Base.Sequence.findi : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> (int * 'a) option
		Base.Sequence.fold_m : bind:(acc_m -> f:(acc -> 'acc_m) -> 'acc_m) -> return:(acc -> 'acc_m) -> 'elt Base.Sequence.t -> init:'acc -> f:(acc -> 'elt -> 'acc_m) -> 'acc_m
		Base.Sequence.fold_result : 'a Base.Sequence.t -> init:'accum -> f:(accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		Base.Sequence.fold_until : 'a Base.Sequence.t -> init:'accum -> f:(accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:(accum -> 'final) -> 'final
		Base.Sequence.foldi : ('a Base.Sequence.t, 'a, 'b) Base.Indexed_container_intf.foldi
		Base.Sequence.folding_map : 'a Base.Sequence.t -> init:'b -> f:(b -> 'a -> 'b * 'c) -> 'c Base.Sequence.t
		Base.Sequence.folding_mapi : 'a Base.Sequence.t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c Base.Sequence.t
		Base.Sequence.for_alli : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> bool
		Base.Sequence.force_eagerly : 'a Base.Sequence.t -> 'a Base.Sequence.t
		Base.Sequence.group : 'a Base.Sequence.t -> break:(a -> 'a -> bool) -> 'a list Base.Sequence.t
		Base.Sequence.ignore_m : 'a Base.Sequence.t -> unit Base.Sequence.t
		Base.Sequence.interleaved_cartesian_product : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * 'b) Base.Sequence.t
		Base.Sequence.intersperse : 'a Base.Sequence.t -> sep:'a -> 'a Base.Sequence.t
		Base.Sequence.iter_m : bind:(unit_m -> f:(unit -> 'unit_m) -> 'unit_m) -> return:(unit -> 'unit_m) -> 'elt Base.Sequence.t -> f:(elt -> 'unit_m) -> 'unit_m
		Base.Sequence.join : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
		Base.Sequence.length_is_bounded_by : ?min:int -> ?max:int -> 'a Base.Sequence.t -> bool
		Base.Sequence.max_elt : 'a Base.Sequence.t -> compare:(a -> 'a -> int) -> 'a option

Containers	Batteries	Base
		Base.Sequence.mem : 'a Base.Sequence.t -> 'a -> equal:('a -> 'a -> bool) -> bool
		Base.Sequence.merge_with_duplicates : 'a Base.Sequence.t -> 'b Base.Sequence.t -> compare:('a -> 'b -> int) -> ('a, 'b) Base.Sequence.Merge_with_duplicates_element.t Base.Sequence.t
		Base.Sequence.min_elt : 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a option
		Base.Sequence.next : 'a Base.Sequence.t -> ('a * 'a Base.Sequence.t) option
		Base.Sequence.nth : 'a Base.Sequence.t -> int -> 'a option
		Base.Sequence.nth_exn : 'a Base.Sequence.t -> int -> 'a
		Base.Sequence.of_lazy : 'a Base.Sequence.t Base.Lazy.t -> 'a Base.Sequence.t
		Base.Sequence.of_seq : 'a Base.Import.Caml.Seq.t -> 'a Base.Sequence.t
		Base.Sequence.reduce : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a option
		Base.Sequence.remove_consecutive_duplicates : 'a Base.Sequence.t -> equal:('a -> 'a -> bool) -> 'a Base.Sequence.t
		Base.Sequence.round_robin : 'a Base.Sequence.t list -> 'a Base.Sequence.t
		Base.Sequence.sexp_of_t : ('a -> Base.Ppx_sexp_conv_lib.Sexp.t) -> 'a Base.Sequence.t -> Base.Ppx_sexp_conv_lib.Sexp.t
		Base.Sequence.shift_left : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
		Base.Sequence.shift_right : 'a Base.Sequence.t -> 'a -> 'a Base.Sequence.t
		Base.Sequence.shift_right_with_list : 'a Base.Sequence.t -> 'a list -> 'a Base.Sequence.t
		Base.Sequence.split_n : 'a Base.Sequence.t -> int -> 'a list * 'a Base.Sequence.t
		Base.Sequence.sub : 'a Base.Sequence.t -> pos:int -> len:int -> 'a Base.Sequence.t
		Base.Sequence.sum : (module Base.Container_intf.Summable with type t = 'sum) -> 'a Base.Sequence.t -> f:('a -> 'sum) -> 'sum
		Base.Sequence.tl : 'a Base.Sequence.t -> 'a Base.Sequence.t option
		Base.Sequence.to_seq : 'a Base.Sequence.t -> 'a Base.Import.Caml.Seq.t
		Base.Sequence.unfold_step : init:'s -> f:('s -> ('a, 's) Base.Sequence.Step.t) -> 'a Base.Sequence.t
		Base.Sequence.unfold_with : 'a Base.Sequence.t -> init:'s -> f:('s -> 'a -> ('b, 's) Base.Sequence.Step.t) -> 'b Base.Sequence.t
		Base.Sequence.unfold_with_and_finish : 'a Base.Sequence.t -> init:'s_a -> running_step:('s_a -> 'a -> ('b, 's_a) Base.Sequence.Step.t) -> inner_finished:('s_a -> 's_b) -> finishing_step:('s_b -> ('b, 's_b) Base.Sequence.Step.t) -> 'b Base.Sequence.t
		Base.Sequence.zip_full : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ['Both of 'a * 'b 'Left of 'a 'Right of 'b] Base.Sequence.t
		Base.Set.are_disjoint : ('a, 'cmp) set -> ('a, 'cmp) set -> bool
		Base.Set.binary_search : ('a, 'cmp) set -> compare:('a -> 'key -> int) -> ['First_equal_to 'First_greater_than_or_equal_to 'First_strictly_greater_than 'Last_equal_to 'Last_less_than_or_equal_to 'Last_strictly_less_than] -> 'key -> 'a option
		Base.Set.binary_search_segmented : ('a, 'cmp) set -> segment_of:('a -> ['Left 'Right]) -> ['First_on_right 'Last_on_left] -> 'a option
		Base.Set.comparator : ('a, 'cmp) set -> ('a, 'cmp) Base.Comparator.t
		Base.Set.comparator_s : ('a, 'cmp) set -> ('a, 'cmp) Base.Set.comparator
		Base.Set.compare : ('elt -> 'elt -> int) -> ('cmp -> 'cmp -> int) -> ('elt, 'cmp) set -> ('elt, 'cmp) set -> int
		Base.Set.compare_m__t : (module Base.Set.Compare_m) -> ('elt, 'cmp) set -> ('elt, 'cmp) set -> int
		Base.Set.count : ('a, 'b) set -> f:('a -> bool) -> int
		Base.Set.equal_m__t : (module Base.Set.Equal_m) -> ('elt, 'cmp) set -> ('elt, 'cmp) set -> bool
		Base.Set.find_map : ('a, 'c) set -> f:('a -> 'b option) -> 'b option
		Base.Set.fold_result : ('a, 'b) set -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		Base.Set.fold_right : ('a, 'b) set -> init:'accum -> f:('a -> 'accum -> 'accum) -> 'accum

Containers	Batteries	Base
		Base.Set.fold_until : ('a, 'b) set -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Set_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
		Base.Set.group_by : ('a, 'cmp) set -> equiv:('a -> 'a -> bool) -> ('a, 'cmp) set list
		Base.Set.hash_fold_direct : 'a Base.Hash.folder -> ('a, 'cmp) set Base.Hash.folder
		Base.Set.hash_fold_m__t : (module Base.Set.Hash_fold_m with type t = 'elt) -> Base.Hash.state -> ('elt, 'a) set -> Base.Hash.state
		Base.Set.hash_m__t : (module Base.Set.Hash_fold_m with type t = 'elt) -> ('elt, 'a) set -> int
		Base.Set.invariants : ('a, 'b) set -> bool
		Base.Set.is_subset : ('a, 'cmp) set -> of_:('a, 'cmp) set -> bool
		Base.Set.iter2 : ('a, 'cmp) set -> ('a, 'cmp) set -> f:(['Both of 'a * 'a 'Left of 'a 'Right of 'a] -> unit) -> unit
		Base.Set.length : ('a, 'b) set -> int
		Base.Set.m__t_of_sexp : (module Base.Set.M_of_sexp with type comparator_witness = 'cmp and type t = 'elt) -> Base.Sexp.t -> ('elt, 'cmp) set
		Base.Set.merge_to_sequence : ?order:['Decreasing 'Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) set -> ('a, 'cmp) set -> ('a, 'a) Base.Set.Merge_to_sequence_element.t Base.Sequence.t
		Base.Set.nth : ('a, 'b) set -> int -> 'a option
		Base.Set.of_increasing_iterator_unchecked : ('a, 'cmp) Base.Set.comparator -> len:int -> f:(int -> 'a) -> ('a, 'cmp) set
		Base.Set.of_sorted_array : ('a, 'cmp) Base.Set.comparator -> 'a array -> ('a, 'cmp) set Base.Or_error.t
		Base.Set.of_sorted_array_unchecked : ('a, 'cmp) Base.Set.comparator -> 'a array -> ('a, 'cmp) set
		Base.Set.remove_index : ('a, 'cmp) set -> int -> ('a, 'cmp) set
		Base.Set.sexp_of_m__t : (module Base.Set.Sexp_of_m with type t = 'elt) -> ('elt, 'cmp) set -> Base.Sexp.t
		Base.Set.stable_dedup_list : ('a, 'b) Base.Set.comparator -> 'a list -> 'a list
		Base.Set.sum : (module Base.Container.Summable with type t = 'sum) -> ('a, 'b) set -> f:('a -> 'sum) -> 'sum
		Base.Set.to_sequence : ?order:['Decreasing 'Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) set -> 'a Base.Sequence.t
		Base.Set.union_list : ('a, 'cmp) Base.Set.comparator -> ('a, 'cmp) set list -> ('a, 'cmp) set
		Base.String.(^) : string -> string -> string
		Base.String.ascending : string -> string -> int
		Base.String.between : string -> low:string -> high:string -> bool
		Base.String.chop_prefix_exn : string -> prefix:string -> string
		Base.String.chop_prefix_if_exists : string -> prefix:string -> string
		Base.String.chop_suffix_exn : string -> suffix:string -> string
		Base.String.chop_suffix_if_exists : string -> suffix:string -> string
		Base.String.clamp : string -> min:string -> max:string -> string Base.Or_error.t
		Base.String.clamp_exn : string -> min:string -> max:string -> string
		Base.String.comparator : (string, Base.String.comparator_witness) Base.Comparator.comparator = {Base.Comparator.compare; sexp_of_t}
		Base.String.concat_array : ?sep:string -> string array -> string
		Base.String.count : string -> f:(Base.String.elt -> bool) -> int
		Base.String.descending : string -> string -> int
		Base.String.drop_prefix : string -> int -> string
		Base.String.drop_suffix : string -> int -> string
		Base.String.find : string -> f:(Base.String.elt -> bool) -> Base.String.elt option
		Base.String.find_map : string -> f:(Base.String.elt -> 'a option) -> 'a option

Containers	Batteries	Base
		Base.String.fold_result : string -> init:'accum -> f:('accum -> Base.String.elt -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		Base.String.fold_until : string -> init:'accum -> f:('accum -> Base.String.elt -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
		Base.String.hash_fold_t : Base.Ppx_hash_lib.Std.Hash.state -> string -> Base.Ppx_hash_lib.Std.Hash.state
		Base.String.index : string -> char -> int option
		Base.String.index_from : string -> int -> char -> int option
		Base.String.invariant : string Base.Invariant_intf.inv
		Base.String.is_prefix : string -> prefix:string -> bool
		Base.String.is_substring : string -> substring:string -> bool
		Base.String.is_substring_at : string -> pos:int -> substring:string -> bool
		Base.String.is_suffix : string -> suffix:string -> bool
		Base.String.lfindi : ?pos:int -> string -> f:(int -> char -> bool) -> int option
		Base.String.lsplit2 : string -> on:char -> (string * string) option
		Base.String.lsplit2_exn : string -> on:char -> string * string
		Base.String.lstrip : ?drop:(char -> bool) -> string -> string
		Base.String.max : string -> string -> string
		Base.String.max_elt : string -> compare:(Base.String.elt -> Base.String.elt -> int) -> Base.String.elt option
		Base.String.max_length : int = 144115188075855863
		Base.String.min : string -> string -> string
		Base.String.min_elt : string -> compare:(Base.String.elt -> Base.String.elt -> int) -> Base.String.elt option
		Base.String.of_char_list : char list -> string
		Base.String.of_string : string -> string
		Base.String.prefix : string -> int -> string
		Base.String.rfindi : ?pos:int -> string -> f:(int -> char -> bool) -> int option
		Base.String.rindex : string -> char -> int option
		Base.String.rindex_from : string -> int -> char -> int option
		Base.String.rsplit2 : string -> on:char -> (string * string) option
		Base.String.rsplit2_exn : string -> on:char -> string * string
		Base.String.rstrip : ?drop:(char -> bool) -> string -> string
		Base.String.sexp_of_t : string -> Sexplib0__Sexp.t
		Base.String.split_lines : string -> string list
		Base.String.split_on_chars : string -> on:char list -> string list
		Base.String.subo : (string, string) Base.Blit.subo
		Base.String.substr_index : ?pos:int -> string -> pattern:string -> int option
		Base.String.substr_index_all : string -> may_overlap:bool -> pattern:string -> int list
		Base.String.substr_index_exn : ?pos:int -> string -> pattern:string -> int
		Base.String.substr_replace_all : string -> pattern:string -> with_:string -> string
		Base.String.substr_replace_first : ?pos:int -> string -> pattern:string -> with_:string -> string
		Base.String.suffix : string -> int -> string
		Base.String.sum : (module Base.Container_intf.Summable with type t = 'sum) -> string -> f:(Base.String.elt -> 'sum) -> 'sum

Containers	Batteries	Base
		Base.String.t_of_sexp : Sexplib0__.Sexp.t -> string
		Base.String.t_sexp_grammar : Base.Ppx_sexp_conv_lib.Sexp.Private.Raw_grammar.t...
		Base.String.to_list_rev : string -> char list
		Base.String.to_string : string -> string
		Base.String.tr : target:char -> replacement:char -> string -> string
		Base.String.tr_multi : target:string -> replacement:string -> (string -> string) Base.Staged.t
		Base.String.validate_bound : min:string Base.Maybe_bound.t -> max:string Base.Maybe_bound.t -> string Base.Validate.check
		Base.String.validate_lbound : min:string Base.Maybe_bound.t -> string Base.Validate.check
		Base.String.validate_ubound : max:string Base.Maybe_bound.t -> string Base.Validate.check