Stdlib	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		
ArrayLabels.append : 'a array -> 'a array -> 'a array	CCArrayLabels.append : 'a array -> 'a array -> 'a array		Base.Array.append : 'a array -> 'a array -> 'a array
			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
ArrayLabels.blit : src:'a array -> src_pos:int -> dst:'a array -> dst_pos:int -> len:int -> unit	CCArrayLabels.blit : src:'a array -> src_pos:int -> dst:'a array -> dst_pos:int -> len:int -> unit	BatArray.Labels.blit : src:'a array -> src_pos:int -> dst:'a array -> dst_pos:int -> len:int -> unit	Base.Array0.blit : src:'a array ? src_pos:int -> dst:'a array -> dst_pos:int -> len:int -> unit
			Base.Array.blito : ('a array, 'a array) Base.Blit_intf.blito
	CCArrayLabels.bsearch : cmp:('a -> 'a -> int) -> key:'a -> 'a array -> [`All_bigger | `All_lower | `At of int | `Empty | `Just_after of int]		
		BatArray.cartesian_product : 'a array -> 'b array -> ('a * 'b) array	Base.Array.cartesian_product : 'a array -> 'b array -> ('a * 'b) array
	CCArrayLabels.compare : 'a CCArrayLabels.ord -> 'a array CCArrayLabels.ord		Base.Array.compare : ('a -> 'a -> int) -> 'a array -> 'a array -> int
ArrayLabels.concat : 'a array list -> 'a array	CCArrayLabels.concat : 'a array list -> 'a array		Base.Array.concat : 'a array list -> 'a array
			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
ArrayLabels.copy : 'a array -> 'a array	CCArrayLabels.copy : 'a array -> 'a array		Base.Array.copy : 'a array -> 'a array
		BatArray.Labels.count_matching : f:('a -> bool) -> 'a array -> int	Base.Array.count : 'a array -> f:('a -> bool) -> int
			Base.Array.counti : 'a array -> f:(int -> 'a -> bool) -> int
ArrayLabels.create : int -> 'a -> 'a array	CCArrayLabels.create : int -> 'a -> 'a array	BatArray.Labels.create : int -> init:'a -> 'a array	Base.Array.create : len:int -> 'a -> 'a array
ArrayLabels.create_float : int -> float array	CCArrayLabels.create_float : int -> float array		
ArrayLabels.create_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	CCArrayLabels.create_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	BatArray.Labels.create_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	
	CCArrayLabels.empty : 'a array = []		
	CCArrayLabels.equal : 'a CCArrayLabels.equal -> 'a array CCArrayLabels.equal		Base.Array.equal : ('a -> 'a -> bool) -> 'a array -> 'a array -> bool
	CCArrayLabels.except_idx : 'a array -> int -> 'a list		
ArrayLabels.exists : f:('a -> bool) -> 'a array -> bool	CCArrayLabels.exists : f:('a -> bool) -> 'a array -> bool	BatArray.Labels.exists : f:('a -> bool) -> 'a array -> bool	Base.Array.exists : 'a array -> f:('a -> bool) -> bool
ArrayLabels.exists2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArrayLabels.exists2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool		Base.Array.exists2_exn : 'a array -> 'b array -> f:('a -> 'b -> bool) -> bool
			Base.Array.existsi : 'a array -> f:(int -> 'a -> bool) -> bool
ArrayLabels.fast_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	CCArrayLabels.fast_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	BatArray.Labels.fast_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	
ArrayLabels.fill : 'a array -> pos:int -> len:int -> 'a -> unit	CCArrayLabels.fill : 'a array -> pos:int -> len:int -> 'a -> unit	BatArray.Labels.fill : 'a array -> pos:int -> len:int -> 'a -> unit	Base.Array.fill : 'a array -> pos:int -> len:int -> 'a -> unit

Stdlib	Containers	Batteries	Base
	CCArrayLabels.filter : f:('a -> bool) -> 'a array -> 'a array	BatArray.Labels.filter : f:('a -> bool) -> 'a array -> 'a array	Base.Array.filter : 'a array -> f:('a -> bool) -> 'a array
	CCArrayLabels.filter_map : f:('a -> 'b option) -> 'a array -> 'b array	BatArray.Labels.filter_map : f:('a -> 'b option) -> 'a array -> 'b array	Base.Array.filter_map : 'a array -> f:('a -> 'b option) -> 'b array
			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
		BatArray.Labels.find : f:('a -> bool) -> 'a array -> 'a	Base.Array.find_exn : 'a array -> f:('a -> bool) -> 'a
			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
		BatArray.Labels.findi : f:('a -> bool) -> 'a array -> int	
			Base.Array.findi : 'a array -> f:(int -> 'a -> bool) -> (int * 'a) option
			Base.Array.findi_exn : 'a array -> f:(int -> 'a -> bool) -> int * 'a
	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 : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	BatArray.Labels.fold : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	Base.Array.fold : 'a array -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
	CCArrayLabels.fold2 : f:('acc -> 'a -> 'b -> 'acc) -> init:'acc -> 'a array -> 'b array -> 'acc		Base.Array.fold2_exn : 'a array -> 'b array -> init:'c -> f:('c -> 'a -> 'b -> 'c) -> 'c
ArrayLabels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	CCArrayLabels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	BatArray.Labels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	
	CCArrayLabels.fold_map : f:('acc -> 'a -> 'acc * 'b) -> init:'acc -> 'a array -> 'acc * 'b array		Base.Array.fold_map : 'a array -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'b * 'c array
			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
ArrayLabels.fold_right : f:('b -> 'a -> 'a) -> 'b array -> init:'a -> 'a	CCArrayLabels.fold_right : f:('b -> 'a -> 'a) -> 'b array -> init:'a -> 'a	BatArray.Labels.fold_right : f:('b -> 'a -> 'a) -> 'b array -> init:'a -> 'a	Base.Array.fold_right : 'a array -> f:('a -> 'b -> 'b) -> init:'b -> 'b
			Base.Array.fold_until : 'a array -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
		BatArray.Labels.fold_while : p:('acc -> 'a -> bool) -> f:('acc -> 'a -> 'acc) -> init:'acc -> 'a array -> 'acc * int	
	CCArrayLabels.fold_while : f:('a -> 'b -> 'a * [`Continue | `Stop]) -> init:'a -> 'b array -> 'a		
	CCArrayLabels.foldi : f:('a -> int -> 'b -> 'a) -> init:'a -> 'b array -> 'a		Base.Array.foldi : 'a array -> init:'b -> f:(int -> 'b -> 'a -> 'b) -> 'b
			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
ArrayLabels.for_all : f:('a -> bool) -> 'a array -> bool	CCArrayLabels.for_all : f:('a -> bool) -> 'a array -> bool	BatArray.Labels.for_all : f:('a -> bool) -> 'a array -> bool	Base.Array.for_all : 'a array -> f:('a -> bool) -> bool
ArrayLabels.for_all2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArrayLabels.for_all2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool		Base.Array.for_all2_exn : 'a array -> 'b array -> f:('a -> 'b -> bool) -> bool

Stdlib	Containers	Batteries	Base
			Base.Array.for_alli : 'a array -> f:(int -> 'a -> bool) -> bool
ArrayLabels.get : 'a array -> int -> 'a	CCArrayLabels.get : 'a array -> int -> 'a		Base.Array.get : 'a array -> int -> 'a
	CCArrayLabels.get_safe : 'a array -> int -> 'a option		
ArrayLabels.init : int -> f:(int -> 'a) -> 'a array	CCArrayLabels.init : int -> f:(int -> 'a) -> 'a array	BatArray.Labels.init : int -> f:(int -> 'a) -> 'a array	Base.Array.init : int -> f:(int -> 'a) -> 'a array
			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
ArrayLabels.iter : f:('a -> unit) -> 'a array -> unit	CCArrayLabels.iter : f:('a -> unit) -> 'a array -> unit	BatArray.Labels.iter : f:('a -> unit) -> 'a array -> unit	Base.Array.iter : 'a array -> f:('a -> unit) -> unit
ArrayLabels.iter2 : f:('a -> 'b -> unit) -> 'a array -> 'b array -> unit	CCArrayLabels.iter2 : f:('a -> 'b -> unit) -> 'a array -> 'b array -> unit	BatArray.Labels.iter2 : f:('a -> 'b -> unit) -> 'a array -> 'b array -> unit	Base.Array.iter2_exn : 'a array -> 'b array -> f:('a -> 'b -> unit) -> unit
		BatArray.Labels.iter2i : f:(int -> 'a -> 'b -> unit) -> 'a array -> 'b array -> unit	
ArrayLabels.iteri : f:(int -> 'a -> unit) -> 'a array -> unit	CCArrayLabels.iteri : f:(int -> 'a -> unit) -> 'a array -> unit	BatArray.Labels.iteri : f:(int -> 'a -> unit) -> 'a array -> unit	Base.Array.iteri : 'a array -> f:(int -> 'a -> unit) -> unit
			Base.Array.last : 'a array -> 'a
ArrayLabels.length : 'a array -> int	CCArrayLabels.length : 'a array -> int		Base.Array.length : 'a array -> int
	CCArrayLabels.lookup : cmp:'a CCArrayLabels.ord -> key:'a -> 'a array -> int option		
	CCArrayLabels.lookup_exn : cmp:'a CCArrayLabels.ord -> key:'a -> 'a array -> int		
ArrayLabels.make : int -> 'a -> 'a array	CCArrayLabels.make : int -> 'a -> 'a array		
ArrayLabels.make_float : int -> float array	CCArrayLabels.make_float : int -> float array		
ArrayLabels.make_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	CCArrayLabels.make_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	BatArray.Labels.make_matrix : dimx:int -> dimy:int -> 'a -> 'a array array	Base.Array.make_matrix : dimx:int -> dimy:int -> 'a -> 'a array array
ArrayLabels.map : f:('a -> 'b) -> 'a array -> 'b array	CCArrayLabels.map : f:('a -> 'b) -> 'a array -> 'b array	BatArray.Labels.map : f:('a -> 'b) -> 'a array -> 'b array	Base.Array.map : 'a array -> f:('a -> 'b) -> 'b array
ArrayLabels.map2 : f:('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array	CCArrayLabels.map2 : f:('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array		Base.Array.map2_exn : 'a array -> 'b array -> f:('a -> 'b -> 'c) -> 'c array
			Base.Array.map_inplace : 'a array -> f:('a -> 'a) -> unit
ArrayLabels.mapi : f:(int -> 'a -> 'b) -> 'a array -> 'b array	CCArrayLabels.mapi : f:(int -> 'a -> 'b) -> 'a array -> 'b array	BatArray.Labels.mapi : f:(int -> 'a -> 'b) -> 'a array -> 'b array	Base.Array.mapi : 'a array -> f:(int -> 'a -> 'b) -> 'b array
			Base.Array.max_elt : 'a array -> compare:('a -> 'a -> int) -> 'a option
			Base.Array.max_length : int = 18014398509481983
ArrayLabels.mem : 'a -> set:'a array -> bool	CCArrayLabels.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a array -> bool		Base.Array.mem : 'a array -> 'a -> equal:('a -> 'a -> bool) -> bool
ArrayLabels.memq : 'a -> set:'a array -> bool	CCArrayLabels.memq : 'a -> set:'a array -> bool		
			Base.Array.min_elt : 'a array -> compare:('a -> 'a -> int) -> 'a option
		BatArray.Labels.modify : f:('a -> 'a) -> 'a array -> unit	
		BatArray.Labels.modifyi : f:(int -> 'a -> 'a) -> 'a array -> unit	
	CCArrayLabels.monoid_product : f:('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array		
ArrayLabels.of_list : 'a list -> 'a array	CCArrayLabels.of_list : 'a list -> 'a array		Base.Array.of_list : 'a list -> 'a array
			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

Stdlib	Containers	Batteries	Base
			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
ArrayLabels.of_seq : 'a Seq.t -> 'a array	CCArrayLabels.of_seq : 'a Seq.t -> 'a array		
		BatArray.partition : ('a -> bool) -> 'a array -> 'a array * 'a array	Base.Array.partition_tf : 'a array -> f:('a -> bool) -> 'a array * 'a array
			Base.Array.partitioni_tf : 'a array -> f:(int -> 'a -> bool) -> 'a array * 'a 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_choose : 'a array -> 'a CCArrayLabels.random_gen		Base.Array.random_element_exn : ?random_state:Base.Random.State.t -> 'a array -> 'a
			Base.Array.random_element : ?random_state:Base.Random.State.t -> 'a array -> 'a option
	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		
			Base.Array.reduce : 'a array -> f:('a -> 'a -> 'a) -> 'a option
		BatArray.reduce : ('a -> 'a -> 'a) -> 'a array -> 'a	Base.Array.reduce_exn : 'a array -> f:('a -> 'a -> 'a) -> 'a
	CCArrayLabels.rev : 'a array -> 'a array		
	CCArrayLabels.reverse_in_place : 'a array -> unit		Base.Array.rev_inplace : 'a array -> unit
	CCArrayLabels.scan_left : f:('acc -> 'a -> 'acc) -> init:'acc -> 'a array -> 'acc array		
ArrayLabels.set : 'a array -> int -> 'a -> unit	CCArrayLabels.set : 'a array -> int -> 'a -> unit		Base.Array.set : 'a array -> int -> 'a -> unit
			Base.Array.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> 'a array -> Sexplib0.Sexp.t
	CCArrayLabels.shuffle : 'a array -> unit		
	CCArrayLabels.shuffle_with : Random.State.t -> 'a array -> unit		Base.Array.permute : ?random_state:Base.Random.State.t -> 'a array -> unit
ArrayLabels.sort : cmp:('a -> 'a -> int) -> 'a array -> unit	CCArrayLabels.sort : cmp:('a -> 'a -> int) -> 'a array -> unit	BatArray.Labels.sort : cmp:('a -> 'a -> int) -> 'a array -> unit	Base.Array.sort : ?pos:int -> ?len:int -> 'a array -> compare:('a -> 'a -> int) -> 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.sorted : f:('a -> 'a -> int) -> 'a array -> 'a array		Base.Array.sorted_copy : 'a array -> compare:('a -> 'a -> int) -> 'a array
ArrayLabels.stable_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	CCArrayLabels.stable_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	BatArray.Labels.stable_sort : cmp:('a -> 'a -> int) -> 'a array -> unit	Base.Array.stable_sort : 'a array -> compare:('a -> 'a -> int) -> unit
ArrayLabels.sub : 'a array -> pos:int -> len:int -> 'a array	CCArrayLabels.sub : 'a array -> pos:int -> len:int -> 'a array	BatArray.Labels.sub : 'a array -> pos:int -> len:int -> 'a array	Base.Array0.sub : 'a array -> pos:int -> len:int -> 'a array
			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
	CCArrayLabels.swap : 'a array -> int -> int -> unit		Base.Array.swap : 'a array -> int -> int -> unit

Stdlib	Containers	Batteries	Base
			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
	CCArrayLabels.to_gen : 'a array -> 'a CCArrayLabels.gen		
	CCArrayLabels.to_iter : 'a array -> 'a CCArrayLabels.iter		
ArrayLabels.to_list : 'a array -> 'a list	CCArrayLabels.to_list : 'a array -> 'a list		Base.Array.to_list : 'a array -> 'a list
ArrayLabels.to_seq : 'a array -> 'a Seq.t	CCArrayLabels.to_seq : 'a array -> 'a Seq.t		
ArrayLabels.to_seqi : 'a array -> (int * 'a) Seq.t	CCArrayLabels.to_seqi : 'a array -> (int * 'a) Seq.t		
			Base.Array.to_sequence : 'a array -> 'a Base.Sequence.t
			Base.Array.to_sequence_mutable : 'a array -> 'a Base.Sequence.t
	CCArrayLabels.to_string : ?sep:string -> ('a -> string) -> 'a array -> string		
			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
ArrayLabels.unsafe_get : 'a array -> int -> 'a	CCArrayLabels.unsafe_get : 'a array -> int -> 'a		Base.Array.unsafe_get : 'a array -> int -> 'a
ArrayLabels.unsafe_set : 'a array -> int -> 'a -> unit	CCArrayLabels.unsafe_set : 'a array -> int -> 'a -> unit		Base.Array.unsafe_set : 'a array -> int -> 'a -> unit
		BatArray.split : ('a * 'b) array -> 'a array * 'b array	Base.Array.unzip : ('a * 'b) array -> 'a array * 'b array
			Base.Array.zip : 'a array -> 'b array -> ('a * 'b) array option
			Base.Array.zip_exn : 'a array -> 'b array -> ('a * 'b) array
	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		
Array.append : 'a array -> 'a array -> 'a array	CCArray.append : 'a array -> 'a array -> 'a array	BatArray.append : 'a array -> 'a array -> 'a array	
		BatArray.avg : int array -> float	
		BatArray.backwards : 'a array -> 'a BatEnum.t	
Array.blit : 'a array -> int -> 'a array -> int -> int -> unit	CCArray.blit : 'a array -> int -> 'a array -> int -> int -> unit	BatArray.blit : 'a array -> int -> 'a array -> int -> int -> unit	
	CCArray.bsearch : cmp:('a -> 'a -> int) -> 'a -> 'a array -> [`All_bigger | `All_lower | `At of int | `Empty | `Just_after of int]	BatArray.bsearch : 'a BatOrd.ord -> 'a array -> 'a -> [`All_bigger | `All_lower | `At of int | `Empty | `Just_after of int]	
		BatArray.cartesian_product : 'a array -> 'b array -> ('a * 'b) array	
	CCArray.compare : 'a CCArray.ord -> 'a array CCArray.ord	BatArray.compare : 'a BatOrd.comp -> 'a array BatOrd.comp	
Array.concat : 'a array list -> 'a array	CCArray.concat : 'a array list -> 'a array	BatArray.concat : 'a array list -> 'a array	
Array.copy : 'a array -> 'a array	CCArray.copy : 'a array -> 'a array	BatArray.copy : 'a array -> 'a array	
		BatArray.count_matching : ('a -> bool) -> 'a array -> int	

Stdlib	Containers	Batteries	Base
Array.create : int -> 'a -> 'a array	CCArray.create : int -> 'a -> 'a array	BatArray.create : int -> 'a -> 'a array	
Array.create_float : int -> float array	CCArray.create_float : int -> float array	BatArray.create_float : int -> float array	
Array.create_matrix : int -> int -> 'a -> 'a array array	CCArray.create_matrix : int -> int -> 'a -> 'a array array	BatArray.create_matrix : int -> int -> 'a -> 'a array array	
		BatArray.decorate_fast_sort : ('a -> 'b) -> 'a array -> 'a array	
		BatArray.decorate_stable_sort : ('a -> 'b) -> 'a array -> 'a array	
	CCArray.empty : 'a array = []		
		BatArray.enum : 'a array -> 'a BatEnum.t	
	CCArray.equal : 'a CCArray.equal -> 'a array CCArray.equal	BatArray.equal : 'a BatOrd.eq -> 'a array BatOrd.eq	
	CCArray.except_idx : 'a array -> int -> 'a list		
Array.exists : ('a -> bool) -> 'a array -> bool	CCArray.exists : ('a -> bool) -> 'a array -> bool	BatArray.exists : ('a -> bool) -> 'a array -> bool	
Array.exists2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArray.exists2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	BatArray.exists2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	
Array.fast_sort : ('a -> 'a -> int) -> 'a array -> unit	CCArray.fast_sort : ('a -> 'a -> int) -> 'a array -> unit	BatArray.fast_sort : ('a -> 'a -> int) -> 'a array -> unit	
		BatArray.favg : float array -> float	
Array.fill : 'a array -> int -> int -> 'a -> unit	CCArray.fill : 'a array -> int -> int -> 'a -> unit	BatArray.fill : 'a array -> int -> int -> 'a -> unit	
	CCArray.filter : ('a -> bool) -> 'a array -> 'a array	BatArray.filter : ('a -> bool) -> 'a array -> 'a array	
	CCArray.filter_map : ('a -> 'b option) -> 'a array -> 'b array	BatArray.filter_map : ('a -> 'b option) -> 'a array -> 'b array	
		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	
	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		
		BatArray.findi : ('a -> bool) -> 'a array -> int	
	CCArray.flat_map : ('a -> 'b array) -> 'a array -> 'b array		
	CCArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	BatArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	
	CCArray.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a array -> 'b array -> 'acc		
Array.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	CCArray.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	BatArray.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	
		BatArray.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b array -> 'a	
	CCArray.fold_map : ('acc -> 'a -> 'acc * 'b) -> 'acc -> 'a array -> 'acc * 'b array		
Array.fold_right : ('b -> 'a -> 'a) -> 'b array -> 'a -> 'a	CCArray.fold_right : ('b -> 'a -> 'a) -> 'b array -> 'a -> 'a	BatArray.fold_right : ('b -> 'a -> 'a) -> 'b array -> 'a -> 'a	
		BatArray.fold_righti : (int -> 'b -> 'a -> 'a) -> 'b array -> 'a -> 'a	
	CCArray.fold_while : ('a -> 'b -> 'a * [`Continue | `Stop]) -> 'a -> 'b array -> 'a		
		BatArray.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a array -> 'acc * int	
	CCArray.foldi : ('a -> int -> 'b -> 'a) -> 'a -> 'b array -> 'a		
Array.for_all : ('a -> bool) -> 'a array -> bool	CCArray.for_all : ('a -> bool) -> 'a array -> bool	BatArray.for_all : ('a -> bool) -> 'a array -> bool	
Array.for_all2 : ('a -> 'b -> bool) -> 'a array -> 'b	CCArray.for_all2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	BatArray.for_all2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	

Stdlib	Containers	Batteries	Base
array -> bool			
		BatArray.fsum : float array -> float	
Array.get : 'a array -> int -> 'a	CCArray.get : 'a array -> int -> 'a	BatArray.get : 'a array -> int -> 'a	
	CCArray.get_safe : 'a array -> int -> 'a option		
		BatArray.head : 'a array -> int -> 'a array	
Array.init : int -> (int -> 'a) -> 'a array	CCArray.init : int -> (int -> 'a) -> 'a array	BatArray.init : int -> (int -> 'a) -> 'a array	
		BatArray.insert : 'a array -> 'a -> int -> 'a array	
		BatArray.is_sorted_by : ('a -> 'b) -> 'a array -> bool	
Array.iter : ('a -> unit) -> 'a array -> unit	CCArray.iter : ('a -> unit) -> 'a array -> unit	BatArray.iter : ('a -> unit) -> 'a array -> unit	
Array.iter2 : ('a -> 'b -> unit) -> 'a array -> 'b array -> unit	CCArray.iter2 : ('a -> 'b -> unit) -> 'a array -> 'b array -> unit	BatArray.iter2 : ('a -> 'b -> unit) -> 'a array -> 'b array -> unit	
		BatArray.iter2i : (int -> 'a -> 'b -> unit) -> 'a array -> 'b array -> unit	
Array.iteri : (int -> 'a -> unit) -> 'a array -> unit	CCArray.iteri : (int -> 'a -> unit) -> 'a array -> unit	BatArray.iteri : (int -> 'a -> unit) -> 'a array -> unit	
		BatArray.kahan_sum : float array -> float	
		BatArray.left : 'a array -> int -> 'a array	
Array.length : 'a array -> int	CCArray.length : 'a array -> int	BatArray.length : 'a array -> int	
	CCArray.lookup : cmp:'a CCArray.ord -> 'a -> 'a array -> int option		
	CCArray.lookup_exn : cmp:'a CCArray.ord -> 'a -> 'a array -> int		
Array.make : int -> 'a -> 'a array	CCArray.make : int -> 'a -> 'a array	BatArray.make : int -> 'a -> 'a array	
Array.make_float : int -> float array	CCArray.make_float : int -> float array	BatArray.make_float : int -> float array	
Array.make_matrix : int -> int -> 'a -> 'a array array	CCArray.make_matrix : int -> int -> 'a -> 'a array array	BatArray.make_matrix : int -> int -> 'a -> 'a array array	
Array.map : ('a -> 'b) -> 'a array -> 'b array	CCArray.map : ('a -> 'b) -> 'a array -> 'b array	BatArray.map : ('a -> 'b) -> 'a array -> 'b array	
Array.map2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array	CCArray.map2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array	BatArray.map2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array	
Array.mapi : (int -> 'a -> 'b) -> 'a array -> 'b array	CCArray.mapi : (int -> 'a -> 'b) -> 'a array -> 'b array	BatArray.mapi : (int -> 'a -> 'b) -> 'a array -> 'b array	
		BatArray.max : 'a array -> 'a	
Array.mem : 'a -> 'a array -> bool	CCArray.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a array -> bool	BatArray.mem : 'a -> 'a array -> bool	
Array.memq : 'a -> 'a array -> bool	CCArray.memq : 'a -> 'a array -> bool	BatArray.memq : 'a -> 'a array -> bool	
		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	
	CCArray.monoid_product : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array		
		BatArray.of_backwards : 'a BatEnum.t -> 'a array	
		BatArray.of_enum : 'a BatEnum.t -> 'a array	
Array.of_list : 'a list -> 'a array	CCArray.of_list : 'a list -> 'a array	BatArray.of_list : 'a list -> 'a array	
Array.of_seq : 'a Seq.t -> 'a array	CCArray.of_seq : 'a Seq.t -> 'a array	BatArray.of_seq : 'a Seq.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	

Stdlib	Containers	Batteries	Base
	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		
		BatArray.print : ?first:string -> ?last:string -> ?sep:string -> ('a, 'b) BatIO.printer -> ('a array, 'b) BatIO.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		
		BatArray.range : 'a array -> int BatEnum.t	
		BatArray.reduce : ('a -> 'a -> 'a) -> 'a array -> 'a	
		BatArray.remove_at : int -> 'a array -> 'a array	
	CCArray.rev : 'a array -> 'a array	BatArray.rev : 'a array -> 'a array	
	CCArray.reverse_in_place : 'a array -> unit	BatArray.rev_in_place : 'a array -> unit	
		BatArray.right : 'a array -> int -> 'a array	
	CCArray.scan_left : ('acc -> 'a -> 'acc) -> 'acc -> 'a array -> 'acc array		
Array.set : 'a array -> int -> 'a -> unit	CCArray.set : 'a array -> int -> 'a -> unit	BatArray.set : 'a array -> int -> 'a -> unit	
	CCArray.shuffle : 'a array -> unit	BatArray.shuffle : ?state:Random.State.t -> 'a array -> unit	
	CCArray.shuffle_with : Random.State.t -> 'a array -> unit		
		BatArray.singleton : 'a -> 'a array	
Array.sort : ('a -> 'a -> int) -> 'a array -> unit	CCArray.sort : ('a -> 'a -> int) -> 'a array -> unit	BatArray.sort : ('a -> 'a -> int) -> '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		
		BatArray.split : ('a * 'b) array -> 'a array * 'b array	
Array.stable_sort : ('a -> 'a -> int) -> 'a array -> unit	CCArray.stable_sort : ('a -> 'a -> int) -> 'a array -> unit	BatArray.stable_sort : ('a -> 'a -> int) -> 'a array -> unit	
Array.sub : 'a array -> int -> int -> 'a array	CCArray.sub : 'a array -> int -> int -> 'a array	BatArray.sub : 'a array -> int -> int -> 'a array	
		BatArray.sum : int array -> int	
	CCArray.swap : 'a array -> int -> int -> unit		
		BatArray.tail : 'a array -> int -> 'a array	
	CCArray.to_gen : 'a array -> 'a CCArray.gen		
	CCArray.to_iter : 'a array -> 'a CCArray.iter		
Array.to_list : 'a array -> 'a list	CCArray.to_list : 'a array -> 'a list	BatArray.to_list : 'a array -> 'a list	
Array.to_seq : 'a array -> 'a Seq.t	CCArray.to_seq : 'a array -> 'a Seq.t	BatArray.to_seq : 'a array -> 'a Seq.t	
Array.to_seqi : 'a array -> (int * 'a) Seq.t	CCArray.to_seqi : 'a array -> (int * 'a) Seq.t	BatArray.to_seqi : 'a array -> (int * 'a) Seq.t	
	CCArray.to_string : ?sep:string -> ('a -> string) -> 'a array -> string		

Stdlib	Containers	Batteries	Base
Array.unsafe_get : 'a array -> int -> 'a	CCArray.unsafe_get : 'a array -> int -> 'a	BatArray.unsafe_get : 'a array -> int -> 'a	
Array.unsafe_set : 'a array -> int -> 'a -> unit	CCArray.unsafe_set : 'a array -> int -> 'a -> unit	BatArray.unsafe_set : 'a array -> int -> 'a -> unit	
Stdlib	Containers	Batteries	Base
	CCListLabels.(--) : int -> int -> int list		
	CCListLabels.(--^) : int -> int -> int list		
	CCListLabels.(<$>) : ('a -> 'b) -> 'a list -> 'b list		
	CCListLabels.(<*>) : ('a -> 'b) list -> 'a list -> 'b list		
	CCListLabels.(>>=) : 'a list -> ('a -> 'b list) -> 'b list		Base.List.(>>=) : 'a list -> ('a -> 'b list) -> 'b list
	CCListLabels.(>|=) : 'a list -> ('a -> 'b) -> 'b list		Base.List.(>>|) : 'a list -> ('a -> 'b) -> 'b list
	CCListLabels.(@) : 'a list -> 'a list -> 'a list	BatList.(@) : 'a list -> 'a list -> 'a list	
	CCListLabels.(and*) : 'a list -> 'b list -> ('a * 'b) list		
	CCListLabels.(and+) : 'a list -> 'b list -> ('a * 'b) list		
	CCListLabels.(let*) : 'a list -> ('a -> 'b list) -> 'b list		
	CCListLabels.(let+) : 'a list -> ('a -> 'b) -> 'b list		
			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
	CCListLabels.Assoc.get : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) CCListLabels.Assoc.t -> 'b option		
	CCListLabels.Assoc.get_exn : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) CCListLabels.Assoc.t -> '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
	CCListLabels.Assoc.mem : ?eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) CCListLabels.Assoc.t -> bool		Base.List.Assoc.mem : ('a, 'b) Base.List.Assoc.t -> equal:('a -> 'a -> bool) -> 'a -> bool
	CCListLabels.Assoc.remove : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) CCListLabels.Assoc.t -> ('a, 'b) CCListLabels.Assoc.t		Base.List.Assoc.remove : ('a, 'b) Base.List.Assoc.t -> equal:('a -> 'a -> bool) -> 'a -> ('a, 'b) Base.List.Assoc.t
	CCListLabels.Assoc.set : eq:('a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) CCListLabels.Assoc.t -> ('a, 'b) CCListLabels.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
			Base.List.Assoc.t_of_sexp : (Sexplib0.Sexp.t -> 'a) -> (Sexplib0.Sexp.t -> 'b) -> Sexplib0.Sexp.t -> ('a, 'b) Base.List.Assoc.t
	CCListLabels.Assoc.update : eq:('a -> 'a -> bool) -> f:('b option -> 'b option) -> 'a -> ('a, 'b) CCListLabels.Assoc.t -> ('a, 'b) CCListLabels.Assoc.t		
	CCListLabels.Infix.(and&) : 'a list -> 'b list -> ('a * 'b) list		
ListLabels.[] : 'a list = ListLabels.[]	CCListLabels.[] : 'a list = []		
	CCListLabels.add_nodup : eq:('a -> 'a -> bool) -> 'a -> 'a list -> 'a list		
	CCListLabels.all_ok : ('a, 'err) result list -> ('a CCListLabels.t, 'err) result		
	CCListLabels.all_some : 'a option list -> 'a list option		
			Base.List.all_unit : unit list list -> unit list
ListLabels.append : 'a list -> 'a list -> 'a list	CCListLabels.append : 'a list -> 'a list -> 'a list		Base.List.append : 'a list -> 'a list -> 'a list
ListLabels.assoc : 'a -> ('a * 'b) list -> 'b	CCListLabels.assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b		

Stdlib	Containers	Batteries	Base
ListLabels.assoc_opt : 'a -> ('a * 'b) list -> 'b option	CCListLabels.assoc_opt : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b option		
ListLabels.assq : 'a -> ('a * 'b) list -> 'b	CCListLabels.assq : 'a -> ('a * 'b) list -> 'b		
ListLabels.assq_opt : 'a -> ('a * 'b) list -> 'b option	CCListLabels.assq_opt : 'a -> ('a * 'b) list -> 'b option		
			Base.List.bind : 'a list -> f:('a -> 'b list) -> 'b list
	CCListLabels.cartesian_product : 'a list list -> 'a list list	BatList.n_cartesian_product : 'a list list -> 'a list list	Base.List.all : 'a list list -> 'a list list
		BatList.cartesian_product : 'a list -> 'b list -> ('a * 'b) list	Base.List.cartesian_product : 'a list -> 'b list -> ('a * 'b) list
	CCListLabels.chunks : int -> 'a list -> 'a list list		Base.List.chunks_of : 'a list -> length:int -> 'a list list
ListLabels.combine : 'a list -> 'b list -> ('a * 'b) list	CCListLabels.combine : 'a list -> 'b list -> ('a * 'b) list	BatList.combine : 'a list -> 'b list -> ('a * 'b) list	Base.List.zip_exn : 'a list -> 'b list -> ('a * 'b) list
	CCListLabels.combine_gen : 'a list -> 'b list -> ('a * 'b) CCListLabels.gen		
	CCListLabels.combine_shortest : 'a list -> 'b list -> ('a * 'b) list		
ListLabels.compare : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> int	CCListLabels.compare : ('a -> 'a -> int) -> 'a list -> 'a list -> int		Base.List.compare : ('a -> 'a -> int) -> 'a list -> 'a list -> int
ListLabels.compare_length_with : 'a list -> len:int -> int	CCListLabels.compare_length_with : 'a list -> int -> int		
ListLabels.compare_lengths : 'a list -> 'b list -> int	CCListLabels.compare_lengths : 'a list -> 'b list -> int		
ListLabels.concat : 'a list list -> 'a list	CCListLabels.concat : 'a list list -> 'a list		Base.List.concat : 'a list list -> 'a list
ListLabels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list	CCListLabels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list	BatList.Labels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list	Base.List.concat_map : '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
ListLabels.cons : 'a -> 'a list -> 'a list	CCListLabels.cons : 'a -> 'a list -> 'a list	BatList.cons : 'a -> 'a list -> 'a list	Base.List.cons : 'a -> 'a list -> 'a list
	CCListLabels.cons' : 'a list -> 'a -> 'a list		
	CCListLabels.cons_maybe : 'a option -> 'a list -> 'a list		
			Base.List.contains_dup : compare:('a -> 'a -> int) -> 'a list -> bool
	CCListLabels.count : f:('a -> bool) -> 'a list -> int	BatList.Labels.count_matching : f:('a -> bool) -> 'a list -> int	Base.List.count : 'a list -> f:('a -> bool) -> int
			Base.List.counti : 'a list -> f:(int -> 'a -> bool) -> int
	CCListLabels.count_true_false : f:('a -> bool) -> 'a list -> int * int		
			Base.List.dedup_and_sort : compare:('a -> 'a -> int) -> 'a list -> 'a list
	CCListLabels.diagonal : 'a list -> ('a * 'a) list		
	CCListLabels.drop : int -> 'a list -> 'a list	BatList.drop : int -> 'a list -> 'a list	Base.List.drop : 'a list -> int -> 'a list
			Base.List.drop_last : 'a list -> 'a list option
			Base.List.drop_last_exn : 'a list -> 'a list
	CCListLabels.drop_while : f:('a -> bool) -> 'a list -> 'a list	BatList.Labels.drop_while : f:('a -> bool) -> 'a list -> 'a list	Base.List.drop_while : 'a list -> f:('a -> bool) -> 'a list
	CCListLabels.empty : 'a list = []		
ListLabels.equal : eq:('a -> 'a -> bool) -> 'a list -> 'a list -> bool	CCListLabels.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool	BatList.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool	Base.List.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool
ListLabels.exists : f:('a -> bool) -> 'a list -> bool	CCListLabels.exists : f:('a -> bool) -> 'a list -> bool	BatList.Labels.exists : f:('a -> bool) -> 'a list -> bool	Base.List.exists : 'a list -> f:('a -> bool) -> bool
ListLabels.exists2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	CCListLabels.exists2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	BatList.Labels.exists2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	Base.List.exists2_exn : 'a list -> 'b list -> f:('a -> 'b -> bool) -> bool
			Base.List.exists2 : 'a list -> 'b list -> f:('a -> 'b -> bool) -> bool Base.List.Or_unequal_lengths.t

Stdlib	Containers	Batteries	Base
			Base.List.existsi : 'a list -> f:(int -> 'a -> bool) -> bool
ListLabels.fast_sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list	CCListLabels.fast_sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list	BatList.Labels.fast_sort : ?cmp:('a -> 'a -> int) -> 'a list -> 'a list	
ListLabels.filter : f:('a -> bool) -> 'a list -> 'a list	CCListLabels.filter : f:('a -> bool) -> 'a list -> 'a list	BatList.Labels.filter : f:('a -> bool) -> 'a list -> 'a list	Base.List.filter : 'a list -> f:('a -> bool) -> 'a list
ListLabels.filter_map : f:('a -> 'b option) -> 'a list -> 'b list	CCListLabels.filter_map : f:('a -> 'b option) -> 'a list -> 'b list	BatList.Labels.filter_map : f:('a -> 'b option) -> 'a list -> 'b list	Base.List.filter_map : 'a list -> f:('a -> 'b option) -> 'b list
			Base.List.filter_mapi : 'a list -> f:(int -> 'a -> 'b option) -> 'b list
			Base.List.filter_opt : 'a option list -> 'a list
ListLabels.filteri : f:(int -> 'a -> bool) -> 'a list -> 'a list	CCListLabels.filteri : f:(int -> 'a -> bool) -> 'a list -> 'a list		Base.List.filteri : 'a list -> f:(int -> 'a -> bool) -> 'a list
ListLabels.find : f:('a -> bool) -> 'a list -> 'a	CCListLabels.find : f:('a -> bool) -> 'a list -> 'a	BatList.Labels.find : f:('a -> bool) -> 'a list -> 'a	Base.List.find_exn : 'a list -> f:('a -> bool) -> 'a
ListLabels.find_all : f:('a -> bool) -> 'a list -> 'a list	CCListLabels.find_all : f:('a -> bool) -> 'a list -> 'a list	BatList.Labels.find_all : f:('a -> bool) -> 'a 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
		BatList.Labels.find_exn : f:('a -> bool) -> exn -> 'a list -> 'a	
	CCListLabels.find_idx : f:('a -> bool) -> 'a list -> (int * 'a) option		
ListLabels.find_map : f:('a -> 'b option) -> 'a list -> 'b option	CCListLabels.find_map : f:('a -> 'b option) -> 'a list -> 'b option	BatList.Labels.find_map_opt : f:('a -> 'b option) -> 'a list -> 'b option	Base.List.find_map : 'a list -> f:('a -> 'b option) -> 'b option
			Base.List.find_map_exn : 'a list -> f:('a -> 'b option) -> 'b
	CCListLabels.find_mapi : f:(int -> 'a -> 'b option) -> 'a list -> 'b option		Base.List.find_mapi : 'a list -> f:(int -> 'a -> 'b option) -> 'b option
			Base.List.find_mapi_exn : 'a list -> f:(int -> 'a -> 'b option) -> 'b
ListLabels.find_opt : f:('a -> bool) -> 'a list -> 'a option	CCListLabels.find_opt : f:('a -> bool) -> 'a list -> 'a option		Base.List.find : 'a list -> f:('a -> bool) -> 'a option
	CCListLabels.find_pred : f:('a -> bool) -> 'a list -> 'a option		
	CCListLabels.find_pred_exn : f:('a -> bool) -> 'a list -> 'a		
		BatList.Labels.findi : f:(int -> 'a -> bool) -> 'a list -> int * 'a	
			Base.List.findi : 'a list -> f:(int -> 'a -> bool) -> (int * 'a) option
	CCListLabels.flat_map : f:('a -> 'b list) -> 'a list -> 'b list		
	CCListLabels.flat_map_i : f:(int -> 'a -> 'b list) -> 'a list -> 'b list		
ListLabels.flatten : 'a list list -> 'a list	CCListLabels.flatten : 'a list list -> 'a list		
		BatList.Labels.fold : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a	Base.List.fold : 'a list -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
			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
	CCListLabels.fold_filter_map : f:('acc -> 'a -> 'acc * 'b option) -> init:'acc -> 'a list -> 'acc * 'b list		
	CCListLabels.fold_filter_map_i : f:('acc -> int -> 'a -> 'acc * 'b option) -> init:'acc -> 'a list -> 'acc * 'b list		
	CCListLabels.fold_flat_map : f:('acc -> 'a -> 'acc * 'b list) -> init:'acc -> 'a list -> 'acc * 'b list		
	CCListLabels.fold_flat_map_i : f:('acc -> int -> 'a -> 'acc * 'b list) -> init:'acc -> 'a list -> 'acc * 'b list		
ListLabels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a	CCListLabels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a	BatList.Labels.fold_left : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a	Base.List.fold_left : 'a list -> init:'b -> f:('b -> 'a -> 'b) -> 'b

Stdlib	**Containers**	**Batteries**	**Base**
ListLabels.fold_left2 : f:('a -> 'b -> 'c -> 'a) -> init:'a -> 'b list -> 'c list -> 'a	CCListLabels.fold_left2 : f:('a -> 'b -> 'c -> 'a) -> init:'a -> 'b list -> 'c list -> 'a	BatList.Labels.fold_left2 : f:('a -> 'b -> 'c -> 'a) -> init:'a -> 'b list -> 'c list -> 'a	
ListLabels.fold_left_map : f:('a -> 'b -> 'a * 'c) -> init:'a -> 'b list -> 'a * 'c list	CCListLabels.fold_left_map : f:('a -> 'b -> 'a * 'c) -> init:'a -> 'b list -> 'a * 'c list		
	CCListLabels.fold_map : f:('acc -> 'a -> 'acc * 'b) -> init:'acc -> 'a list -> 'acc * 'b list		Base.List.fold_map : 'a list -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'b * 'c list
	CCListLabels.fold_map2 : f:('acc -> 'a -> 'b -> 'acc * 'c) -> init:'acc -> 'a list -> 'b list -> 'acc * 'c list		
	CCListLabels.fold_map_i : f:('acc -> int -> 'a -> 'acc * 'b) -> init:'acc -> 'a list -> 'acc * 'b list		Base.List.fold_mapi : 'a list -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'b * 'c list
	CCListLabels.fold_on_map : f:('a -> 'b) -> reduce:('acc -> 'b -> 'acc) -> init:'acc -> 'a list -> 'acc		
	CCListLabels.fold_product : f:('c -> 'a -> 'b -> 'c) -> init:'c -> 'a list -> 'b list -> 'c		
			Base.List.fold_result : 'a list -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
ListLabels.fold_right : f:('a -> 'b -> 'b) -> 'a list -> init:'b -> 'b	CCListLabels.fold_right : f:('a -> 'b -> 'b) -> 'a list -> init:'b -> 'b	BatList.Labels.fold_right : f:('a -> 'b -> 'b) -> 'a list -> init:'b -> 'b	Base.List.fold_right : 'a list -> f:('a -> 'b -> 'b) -> init:'b -> 'b
ListLabels.fold_right2 : f:('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> init:'c -> 'c	CCListLabels.fold_right2 : f:('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> init:'c -> 'c	BatList.Labels.fold_right2 : f:('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> init:'c -> 'c	
			Base.List.fold_until : 'a list -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base.Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
	CCListLabels.fold_while : f:('a -> 'b -> 'a * [`Continue | `Stop]) -> init:'a -> 'b list -> 'a		
	CCListLabels.foldi : f:('b -> int -> 'a -> 'b) -> init:'b -> 'a list -> 'b		Base.List.foldi : 'a list -> init:'b -> f:(int -> 'b -> 'a -> 'b) -> 'b
	CCListLabels.foldi2 : f:('c -> int -> 'a -> 'b -> 'c) -> init:'c -> 'a list -> 'b list -> 'c		
			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
ListLabels.for_all : f:('a -> bool) -> 'a list -> bool	CCListLabels.for_all : f:('a -> bool) -> 'a list -> bool	BatList.Labels.for_all : f:('a -> bool) -> 'a list -> bool	Base.List.for_all : 'a list -> f:('a -> bool) -> bool
ListLabels.for_all2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	CCListLabels.for_all2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	BatList.Labels.for_all2 : f:('a -> 'b -> bool) -> 'a list -> 'b list -> bool	Base.List.for_all2_exn : 'a list -> 'b list -> f:('a -> 'b -> bool) -> bool
			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
	CCListLabels.get_at_idx : int -> 'a list -> 'a option		
	CCListLabels.get_at_idx_exn : int -> 'a list -> 'a		
			Base.List.groupi : 'a list -> break:(int -> 'a -> 'a -> bool) -> 'a list list
	CCListLabels.group_by : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a list -> 'a list list		Base.List.group : 'a list -> break:('a -> 'a -> bool) -> 'a list list
	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		
			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
ListLabels.hd : 'a list -> 'a	CCListLabels.hd : 'a list -> 'a	BatList.hd : 'a list -> 'a	Base.List.hd_exn : 'a list -> 'a
	CCListLabels.hd_tl : 'a list -> 'a * 'a list		
	CCListLabels.head_opt : 'a list -> 'a option		Base.List.hd : 'a list -> 'a option

Stdlib	Containers	Batteries	Base
			Base.List.ignore_m : 'a list -> unit list
ListLabels.init : len:int -> f:(int -> 'a) -> 'a list	CCListLabels.init : int -> f:(int -> 'a) -> 'a list	BatList.Labels.init : int -> f:(int -> 'a) -> 'a list	Base.List.init : int -> f:(int -> '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.intersperse : x:'a -> 'a list -> 'a list		Base.List.intersperse : 'a list -> sep:'a -> 'a list
			Base.List.invariant : 'a Base.Invariant_intf.inv -> 'a list Base.Invariant_intf.inv
	CCListLabels.is_empty : 'a list -> bool		Base.List.is_empty : 'a list -> bool
			Base.List.is_prefix : 'a list -> prefix:'a list -> equal:('a -> 'a -> bool) -> bool
	CCListLabels.is_sorted : cmp:('a -> 'a -> int) -> 'a list -> bool		Base.List.is_sorted : 'a list -> compare:('a -> 'a -> int) -> 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
ListLabels.iter : f:('a -> unit) -> 'a list -> unit	CCListLabels.iter : f:('a -> unit) -> 'a list -> unit	BatList.Labels.iter : f:('a -> unit) -> 'a list -> unit	Base.List.iter : 'a list -> f:('a -> unit) -> unit
ListLabels.iter2 : f:('a -> 'b -> unit) -> 'a list -> 'b list -> unit	CCListLabels.iter2 : f:('a -> 'b -> unit) -> 'a list -> 'b list -> unit	BatList.Labels.iter2 : f:('a -> 'b -> unit) -> 'a list -> 'b list -> unit	Base.List.iter2_exn : 'a list -> 'b list -> f:('a -> 'b -> unit) -> unit
			Base.List.iter2 : 'a list -> 'b list -> f:('a -> 'b -> unit) -> unit Base.List.Or_unequal_lengths.t
ListLabels.iteri : f:(int -> 'a -> unit) -> 'a list -> unit	CCListLabels.iteri : f:(int -> 'a -> unit) -> 'a list -> unit	BatList.Labels.iteri : f:(int -> 'a -> unit) -> 'a list -> unit	Base.List.iteri : 'a list -> f:(int -> 'a -> unit) -> unit
	CCListLabels.iteri2 : f:(int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit		
			Base.List.join : 'a list list -> 'a list
	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		
		BatList.last : 'a list -> 'a	Base.List.last_exn : 'a list -> 'a
	CCListLabels.last_opt : 'a list -> 'a option		Base.List.last : 'a list -> 'a option
ListLabels.length : 'a list -> int	CCListLabels.length : 'a list -> int		Base.List.length : 'a list -> int
ListLabels.map : f:('a -> 'b) -> 'a list -> 'b list	CCListLabels.map : f:('a -> 'b) -> 'a list -> 'b list	BatList.Labels.map : f:('a -> 'b) -> 'a list -> 'b list	Base.List.map : 'a list -> f:('a -> 'b) -> 'b list
ListLabels.map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	CCListLabels.map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	BatList.Labels.map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	Base.List.map2_exn : 'a list -> 'b list -> f:('a -> 'b -> 'c) -> 'c 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
			Base.List.map3_exn : 'a list -> 'b list -> 'c list -> f:('a -> 'b -> 'c -> 'd) -> 'd list
	CCListLabels.map_product_l : f:('a -> 'b list) -> 'a list -> 'b list list		
ListLabels.mapi : f:(int -> 'a -> 'b) -> 'a list -> 'b list	CCListLabels.mapi : f:(int -> 'a -> 'b) -> 'a list -> 'b list	BatList.Labels.mapi : f:(int -> 'a -> 'b) -> 'a list -> 'b list	Base.List.mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b list
			Base.List.max_elt : 'a list -> compare:('a -> 'a -> int) -> 'a option

Stdlib	Containers	Batteries	Base
ListLabels.mem : 'a -> set:'a list -> bool	CCListLabels.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a list -> bool		Base.List.mem : 'a list -> 'a -> equal:('a -> 'a -> bool) -> bool
ListLabels.mem_assoc : 'a -> map:('a * 'b) list -> bool	CCListLabels.mem_assoc : ?eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> bool		
ListLabels.mem_assq : 'a -> map:('a * 'b) list -> bool	CCListLabels.mem_assq : 'a -> map:('a * 'b) list -> bool		
ListLabels.memq : 'a -> set:'a list -> bool	CCListLabels.memq : 'a -> set:'a list -> bool		
ListLabels.merge : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	CCListLabels.merge : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	BatList.Labels.merge : ?cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	Base.List.merge : 'a list -> 'a list -> compare:('a -> 'a -> int) -> 'a list
	CCListLabels.mguard : bool -> unit list		
			Base.List.min_elt : 'a list -> compare:('a -> 'a -> int) -> 'a option
ListLabels.nth : 'a list -> int -> 'a	CCListLabels.nth : 'a list -> int -> 'a		Base.List.nth_exn : 'a list -> int -> 'a
ListLabels.nth_opt : 'a list -> int -> 'a option	CCListLabels.nth_opt : 'a list -> int -> 'a option		Base.List.nth : 'a list -> int -> 'a option
	CCListLabels.of_gen : 'a CCListLabels.gen -> 'a list		
	CCListLabels.of_iter : 'a CCListLabels.iter -> 'a list		
			Base.List.of_list : 'a list -> 'a list
ListLabels.of_seq : 'a Seq.t -> 'a list	CCListLabels.of_seq : 'a Seq.t -> 'a list		
	CCListLabels.of_seq_rev : 'a Seq.t -> 'a list		
ListLabels.partition : f:('a -> bool) -> 'a list -> 'a list * 'a list	CCListLabels.partition : f:('a -> bool) -> 'a list -> 'a list * 'a list	BatList.Labels.partition : f:('a -> bool) -> 'a list -> 'a list * 'a list	Base.List.partition_tf : 'a list -> f:('a -> bool) -> '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
	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		
ListLabels.partition_map : f:('a -> ('b, 'c) Either.t) -> 'a list -> 'b list * 'c list	CCListLabels.partition_map_either : f:('a -> ('b, 'c) CCEither.t) -> 'a list -> 'b list * 'c list	BatList.Labels.partition_map : f:('a -> ('b, 'c) BatEither.t) -> 'a list -> 'b list * 'c list	Base.List.partition_map : 'a list -> f:('a -> ('b, 'c) Base.Either0.t) -> 'b list * 'c 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
	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		
			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
	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		Base.List.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int list

Stdlib	Containers	Batteries	Base
	CCListLabels.range' : int -> int -> int list		
			Base.List.range' : compare:('a -> 'a -> int) -> stride:('a -> 'a) -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> 'a -> 'a -> 'a list
	CCListLabels.range_by : step:int -> int -> int -> int list		
	CCListLabels.reduce : f:('a -> 'a -> 'a) -> 'a list -> 'a option		Base.List.reduce : 'a list -> f:('a -> 'a -> 'a) -> 'a option
			Base.List.reduce_balanced : 'a list -> f:('a -> 'a -> 'a) -> 'a option
			Base.List.reduce_balanced_exn : 'a list -> f:('a -> 'a -> 'a) -> 'a
	CCListLabels.reduce_exn : f:('a -> 'a -> 'a) -> 'a list -> 'a		Base.List.reduce_exn : 'a list -> f:('a -> 'a -> 'a) -> 'a
	CCListLabels.remove : eq:('a -> 'a -> bool) -> key:'a -> 'a list -> 'a list		
ListLabels.remove_assoc : 'a -> ('a * 'b) list -> ('a * 'b) list	CCListLabels.remove_assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> ('a * 'b) list		
ListLabels.remove_assq : 'a -> ('a * 'b) list -> ('a * 'b) list	CCListLabels.remove_assq : 'a -> ('a * 'b) list -> ('a * 'b) list		
	CCListLabels.remove_at_idx : int -> 'a list -> 'a list		
			Base.List.remove_consecutive_duplicates : ?which_to_keep:[`First | `Last] -> 'a list -> equal:('a -> 'a -> bool) -> 'a list
		BatList.Labels.remove_if : f:('a -> bool) -> '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.return : 'a -> 'a list		Base.List.return : 'a -> 'a list
ListLabels.rev : 'a list -> 'a list	CCListLabels.rev : 'a list -> 'a list	BatList.rev : 'a list -> 'a list	Base.List.rev : 'a list -> 'a list
ListLabels.rev_append : 'a list -> 'a list -> 'a list	CCListLabels.rev_append : 'a list -> 'a list -> 'a list	BatList.rev_append : 'a list -> 'a list -> 'a list	Base.List.rev_append : 'a list -> 'a list -> '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
ListLabels.rev_map : f:('a -> 'b) -> 'a list -> 'b list	CCListLabels.rev_map : f:('a -> 'b) -> 'a list -> 'b list	BatList.Labels.rev_map : f:('a -> 'b) -> 'a list -> 'b list	Base.List.rev_map : 'a list -> f:('a -> 'b) -> 'b list
ListLabels.rev_map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	CCListLabels.rev_map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	BatList.Labels.rev_map2 : f:('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	Base.List.rev_map2_exn : 'a list -> 'b list -> f:('a -> 'b -> 'c) -> 'c 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
		BatList.Labels.rfind : f:('a -> bool) -> 'a list -> 'a	
	CCListLabels.scan_left : f:('acc -> 'a -> 'acc) -> init:'acc -> 'a list -> 'acc list		
	CCListLabels.set_at_idx : int -> 'a -> 'a list -> 'a list		
			Base.List.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> 'a list -> Sexplib0.Sexp.t
ListLabels.sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list	CCListLabels.sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list		Base.List.sort : 'a list -> compare:('a -> 'a -> int) -> 'a list
ListLabels.sort_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list	CCListLabels.sort_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list		

Stdlib	Containers	Batteries	Base
	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		
ListLabels.split : ('a * 'b) list -> 'a list * 'b list	CCListLabels.split : ('a * 'b) list -> 'a list * 'b list		
			Base.List.split_n : 'a list -> int -> 'a list * 'a list
			Base.List.split_while : 'a list -> f:('a -> bool) -> 'a list * 'a list
ListLabels.stable_sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list	CCListLabels.stable_sort : cmp:('a -> 'a -> int) -> 'a list -> 'a list	BatList.Labels.stable_sort : ?cmp:('a -> 'a -> int) -> 'a list -> 'a list	Base.List.stable_sort : 'a list -> compare:('a -> 'a -> int) -> 'a list
			Base.List.sub : 'a list -> pos:int -> len:int -> '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		
		BatList.Labels.subset : cmp:('a -> 'b -> int) -> 'a list -> 'b list -> bool	
			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...
	CCListLabels.tail_opt : 'a list -> 'a list option		
	CCListLabels.take : int -> 'a list -> 'a list	BatList.take : int -> 'a list -> 'a list	Base.List.take : 'a list -> int -> 'a list
	CCListLabels.take_drop : int -> 'a list -> 'a list * 'a list		
	CCListLabels.take_drop_while : f:('a -> bool) -> 'a list -> 'a list * 'a list		
	CCListLabels.take_while : f:('a -> bool) -> 'a list -> 'a list	BatList.Labels.take_while : f:('a -> bool) -> 'a list -> 'a list	Base.List.take_while : 'a list -> f:('a -> bool) -> 'a list
ListLabels.tl : 'a list -> 'a list	CCListLabels.tl : 'a list -> 'a list	BatList.tl : 'a list -> 'a list	Base.List.tl_exn : 'a list -> 'a list
			Base.List.tl : 'a list -> 'a list option
			Base.List.to_array : 'a list -> 'a array
	CCListLabels.to_gen : 'a list -> 'a CCListLabels.gen		
	CCListLabels.to_iter : 'a list -> 'a CCListLabels.iter		
			Base.List.to_list : 'a list -> 'a list
ListLabels.to_seq : 'a list -> 'a Seq.t	CCListLabels.to_seq : 'a list -> 'a Seq.t		
	CCListLabels.to_string : ?start:string -> ?stop:string -> ?sep:string -> ('a -> string) -> 'a list -> string		
			Base.List.transpose : 'a list list -> 'a list list option
		BatList.transpose : 'a list list -> 'a list list	Base.List.transpose_exn : 'a list list -> 'a list list
	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		
			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
	CCList.(--) : int -> int -> int list		
	CCList.(--^) : int -> int -> int list		

Stdlib	Containers	Batteries	Base
	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.(let*) : 'a list -> ('a -> 'b list) -> 'b list		
	CCList.(let+) : 'a list -> ('a -> 'b) -> 'b list		
List.[] : 'a list = List.[]	CCList.[] : 'a list = []	BatList.[] : 'a list = BatList.[]	
	CCList.Assoc.get : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) CCList.Assoc.t -> 'b option		
	CCList.Assoc.get_exn : 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		
	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		
List.append : 'a list -> 'a list -> 'a list	CCList.append : 'a list -> 'a list -> 'a list	BatList.append : 'a list -> 'a list -> 'a list	
List.assoc : 'a -> ('a * 'b) list -> 'b		BatList.assoc : 'a -> ('a * 'b) list -> 'b	
	CCList.assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b		
		BatList.assoc_inv : 'b -> ('a * 'b) list -> 'a	
List.assoc_opt : 'a -> ('a * 'b) list -> 'b option		BatList.assoc_opt : 'a -> ('a * 'b) list -> 'b option	
	CCList.assoc_opt : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> 'b option		
List.assq : 'a -> ('a * 'b) list -> 'b	CCList.assq : 'a -> ('a * 'b) list -> 'b	BatList.assq : 'a -> ('a * 'b) list -> 'b	
		BatList.assq_inv : 'b -> ('a * 'b) list -> 'a	
List.assq_opt : 'a -> ('a * 'b) list -> 'b option	CCList.assq_opt : 'a -> ('a * 'b) list -> 'b option	BatList.assq_opt : 'a -> ('a * 'b) list -> 'b option	
		BatList.at : 'a list -> int -> 'a	
		BatList.at_opt : 'a list -> int -> 'a option	
		BatList.backwards : 'a list -> 'a BatEnum.t	
	CCList.cartesian_product : 'a list list -> 'a list list	BatList.n_cartesian_product : 'a list list -> 'a list list	
		BatList.cartesian_product : 'a list -> 'b list -> ('a * 'b) list	
	CCList.chunks : int -> 'a list -> 'a list list		
List.combine : 'a list -> 'b list -> ('a * 'b) list	CCList.combine : 'a list -> 'b list -> ('a * 'b) list	BatList.combine : 'a list -> 'b list -> ('a * 'b) list	
	CCList.combine_gen : 'a list -> 'b list -> ('a * 'b) CCList.gen		
	CCList.combine_shortest : 'a list -> 'b list -> ('a * 'b) list		

Stdlib	Containers	Batteries	Base
List.compare : ('a -> 'a -> int) -> 'a list -> 'a list -> int	CCList.compare : ('a -> 'a -> int) -> 'a list -> 'a list -> int	BatList.compare : 'a BatOrd.comp -> 'a list BatOrd.comp	
List.compare_length_with : 'a list -> int -> int	CCList.compare_length_with : 'a list -> int -> int	BatList.compare_length_with : 'a list -> int -> int	
List.compare_lengths : 'a list -> 'b list -> int	CCList.compare_lengths : 'a list -> 'b list -> int	BatList.compare_lengths : 'a list -> 'b list -> int	
List.concat : 'a list list -> 'a list	CCList.concat : 'a list list -> 'a list	BatList.concat : 'a list list -> 'a list	
List.concat_map : ('a -> 'b list) -> 'a list -> 'b list	CCList.concat_map : ('a -> 'b list) -> 'a list -> 'b list	BatList.concat_map : ('a -> 'b list) -> 'a list -> 'b list	
List.cons : 'a -> 'a list -> 'a list	CCList.cons : 'a -> 'a list -> 'a list	BatList.cons : 'a -> 'a list -> 'a list	
	CCList.cons' : 'a list -> 'a -> 'a list		
	CCList.cons_maybe : 'a option -> 'a list -> 'a list		
	CCList.count : ('a -> bool) -> 'a list -> int	BatList.count_matching : ('a -> bool) -> 'a list -> int	
	CCList.count_true_false : ('a -> bool) -> 'a list -> int * int		
	CCList.diagonal : 'a list -> ('a * 'a) list		
	CCList.drop : int -> 'a list -> 'a list	BatList.drop : int -> 'a list -> 'a list	
	CCList.drop_while : ('a -> bool) -> 'a list -> 'a list	BatList.drop_while : ('a -> bool) -> 'a list -> 'a list	
		BatList.dropwhile : ('a -> bool) -> 'a list -> 'a list	
	CCList.empty : 'a list = []		
		BatList.enum : 'a list -> 'a BatEnum.t	
		BatList.eq : 'a BatOrd.eq -> 'a list BatOrd.eq	
List.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool	CCList.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool	BatList.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool	
List.exists : ('a -> bool) -> 'a list -> bool	CCList.exists : ('a -> bool) -> 'a list -> bool	BatList.exists : ('a -> bool) -> 'a list -> bool	
List.exists2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	CCList.exists2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	BatList.exists2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	
List.fast_sort : ('a -> 'a -> int) -> 'a list -> 'a list	CCList.fast_sort : ('a -> 'a -> int) -> 'a list -> 'a list	BatList.fast_sort : ('a -> 'a -> int) -> 'a list -> 'a list	
		BatList.favg : float list -> float	
List.filter : ('a -> bool) -> 'a list -> 'a list	CCList.filter : ('a -> bool) -> 'a list -> 'a list	BatList.filter : ('a -> bool) -> 'a list -> 'a list	
List.filter_map : ('a -> 'b option) -> 'a list -> 'b list	CCList.filter_map : ('a -> 'b option) -> 'a list -> 'b list	BatList.filter_map : ('a -> 'b option) -> 'a list -> 'b list	
List.filteri : (int -> 'a -> bool) -> 'a list -> 'a list	CCList.filteri : (int -> 'a -> bool) -> 'a list -> 'a list	BatList.filteri : (int -> 'a -> bool) -> 'a list -> 'a list	
		BatList.filteri_map : (int -> 'a -> 'b option) -> 'a list -> 'b list	
List.find : ('a -> bool) -> 'a list -> 'a	CCList.find : ('a -> bool) -> 'a list -> 'a	BatList.find : ('a -> bool) -> 'a list -> 'a	
List.find_all : ('a -> bool) -> 'a list -> 'a list	CCList.find_all : ('a -> bool) -> 'a list -> 'a list	BatList.find_all : ('a -> bool) -> 'a list -> 'a list	
		BatList.find_exn : ('a -> bool) -> exn -> 'a list -> 'a	
	CCList.find_idx : ('a -> bool) -> 'a list -> (int * 'a) option		
List.find_map : ('a -> 'b option) -> 'a list -> 'b option	CCList.find_map : ('a -> 'b option) -> 'a list -> 'b option	BatList.find_map_opt : ('a -> 'b option) -> 'a list -> 'b option	
		BatList.find_map : ('a -> 'b option) -> 'a list -> 'b	
	CCList.find_mapi : (int -> 'a -> 'b option) -> 'a list -> 'b option		
List.find_opt : ('a -> bool) -> 'a list -> 'a option	CCList.find_opt : ('a -> bool) -> 'a list -> 'a option	BatList.find_opt : ('a -> bool) -> 'a list -> 'a option	
	CCList.find_pred : ('a -> bool) -> 'a list -> 'a option		
	CCList.find_pred_exn : ('a -> bool) -> 'a list -> 'a		
		BatList.findi : (int -> 'a -> bool) -> 'a list -> int * 'a	
		BatList.first : 'a list -> 'a	

Stdlib	Containers	Batteries	Base
	CCList.flat_map : ('a -> 'b list) -> 'a list -> 'b list		
	CCList.flat_map_i : (int -> 'a -> 'b list) -> 'a list -> 'b list		
List.flatten : 'a list list -> 'a list	CCList.flatten : 'a list list -> 'a list	BatList.flatten : 'a list list -> 'a list	
		BatList.fold : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a	
	CCList.fold_filter_map : ('acc -> 'a -> 'acc * 'b option) -> 'acc -> 'a list -> 'acc * 'b list		
	CCList.fold_filter_map_i : ('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_map_i : ('acc -> int -> 'a -> 'acc * 'b list) -> 'acc -> 'a list -> 'acc * 'b list		
List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a	CCList.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a	BatList.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a	
List.fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a	CCList.fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a	BatList.fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a	
List.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b list -> 'a * 'c list	CCList.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b list -> 'a * 'c list	BatList.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b list -> 'a * 'c 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_map_i : ('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		
List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b	CCList.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b	BatList.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b	
List.fold_right2 : ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c	CCList.fold_right2 : ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c	BatList.fold_right2 : ('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c	
		BatList.fold_righti : (int -> 'b -> 'a -> 'a) -> 'b list -> 'a -> 'a	
	CCList.fold_while : ('a -> 'b -> 'a * [`Continue | `Stop]) -> 'a -> 'b list -> 'a		
		BatList.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc * 'a list	
	CCList.foldi : ('b -> int -> 'a -> 'b) -> 'b -> 'a list -> 'b	BatList.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b list -> 'a	
	CCList.foldi2 : ('c -> int -> 'a -> 'b -> 'c) -> 'c -> 'a list -> 'b list -> 'c		
List.for_all : ('a -> bool) -> 'a list -> bool	CCList.for_all : ('a -> bool) -> 'a list -> bool	BatList.for_all : ('a -> bool) -> 'a list -> bool	
List.for_all2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	CCList.for_all2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	BatList.for_all2 : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool	
		BatList.frange : float -> [< `Downto | `To] -> float -> int -> float list	
		BatList.fsum : float list -> float	
	CCList.get_at_idx : int -> 'a list -> 'a option		
	CCList.get_at_idx_exn : int -> 'a list -> 'a		
		BatList.group : ('a -> 'a -> int) -> 'a list -> 'a list list	
	CCList.group_by : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a list -> 'a list list		

Stdlib	Containers	Batteries	Base
		BatList.group_consecutive : ('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		
List.hd : 'a list -> 'a	CCList.hd : 'a list -> 'a	BatList.hd : 'a list -> 'a	
	CCList.hd_tl : 'a list -> 'a * 'a list		
	CCList.head_opt : 'a list -> 'a option		
		BatList.index_of : 'a -> 'a list -> int option	
		BatList.index_ofq : 'a -> 'a list -> int option	
List.init : int -> (int -> 'a) -> 'a list	CCList.init : int -> (int -> 'a) -> 'a list	BatList.init : int -> (int -> 'a) -> 'a list	
	CCList.insert_at_idx : int -> 'a -> 'a list -> 'a list		
	CCList.inter : eq:('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
	CCList.interleave : 'a list -> 'a list -> 'a list		
	CCList.intersperse : 'a -> 'a list -> 'a list	BatList.interleave : ?first:'a -> ?last:'a -> 'a -> 'a list -> 'a list	
	CCList.is_empty : 'a list -> bool	BatList.is_empty : 'a list -> bool	
	CCList.is_sorted : cmp:('a -> 'a -> int) -> 'a list -> bool		
List.iter : ('a -> unit) -> 'a list -> unit	CCList.iter : ('a -> unit) -> 'a list -> unit	BatList.iter : ('a -> unit) -> 'a list -> unit	
List.iter2 : ('a -> 'b -> unit) -> 'a list -> 'b list -> unit	CCList.iter2 : ('a -> 'b -> unit) -> 'a list -> 'b list -> unit	BatList.iter2 : ('a -> 'b -> unit) -> 'a list -> 'b list -> unit	
		BatList.iter2i : (int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit	
List.iteri : (int -> 'a -> unit) -> 'a list -> unit	CCList.iteri : (int -> 'a -> unit) -> 'a list -> unit	BatList.iteri : (int -> 'a -> unit) -> 'a list -> unit	
	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		
		BatList.kahan_sum : float list -> float	
	CCList.keep_ok : ('a, 'b) result list -> 'a list		
	CCList.keep_some : 'a option list -> 'a list		
	CCList.last : int -> 'a list -> 'a list		
		BatList.last : 'a list -> 'a	
	CCList.last_opt : 'a list -> 'a option		
List.length : 'a list -> int	CCList.length : 'a list -> int	BatList.length : 'a list -> int	
		BatList.make : int -> 'a -> 'a list	
List.map : ('a -> 'b) -> 'a list -> 'b list	CCList.map : ('a -> 'b) -> 'a list -> 'b list	BatList.map : ('a -> 'b) -> 'a list -> 'b list	
List.map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	CCList.map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	BatList.map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	
		BatList.map2i : (int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	
	CCList.map_product_l : ('a -> 'b list) -> 'a list -> 'b list list		
List.mapi : (int -> 'a -> 'b) -> 'a list -> 'b list	CCList.mapi : (int -> 'a -> 'b) -> 'a list -> 'b list	BatList.mapi : (int -> 'a -> 'b) -> 'a list -> 'b list	
		BatList.max : 'a list -> 'a	

Stdlib	Containers	Batteries	Base
List.mem : 'a -> 'a list -> bool	CCList.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a list -> bool	BatList.mem : 'a -> 'a list -> bool	
List.mem_assoc : 'a -> ('a * 'b) list -> bool	CCList.mem_assoc : ?eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> bool	BatList.mem_assoc : 'a -> ('a * 'b) list -> bool	
List.mem_assq : 'a -> ('a * 'b) list -> bool	CCList.mem_assq : 'a -> ('a * 'b) list -> bool	BatList.mem_assq : 'a -> ('a * 'b) list -> bool	
		BatList.mem_cmp : ('a -> 'a -> int) -> 'a -> 'a list -> bool	
List.memq : 'a -> 'a list -> bool	CCList.memq : 'a -> 'a list -> bool	BatList.memq : 'a -> 'a list -> bool	
List.merge : ('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	CCList.merge : ('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	BatList.merge : ('a -> 'a -> int) -> 'a list -> 'a list -> 'a list	
	CCList.mguard : bool -> unit list		
		BatList.min : 'a list -> 'a	
		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	
	CCList.cartesian_product : 'a list list -> 'a list list	BatList.n_cartesian_product : 'a list list -> 'a list list	
		BatList.nsplit : ('a -> bool) -> 'a list -> 'a list list	
		BatList.ntake : int -> 'a list -> 'a list list	
List.nth : 'a list -> int -> 'a	CCList.nth : 'a list -> int -> 'a	BatList.nth : 'a list -> int -> 'a	
List.nth_opt : 'a list -> int -> 'a option	CCList.nth_opt : 'a list -> int -> 'a option	BatList.nth_opt : 'a list -> int -> 'a option	
		BatList.of_backwards : 'a BatEnum.t -> 'a list	
		BatList.of_enum : 'a BatEnum.t -> 'a list	
	CCList.of_gen : 'a CCList.gen -> 'a list		
	CCList.of_iter : 'a CCList.iter -> 'a list		
List.of_seq : 'a Seq.t -> 'a list	CCList.of_seq : 'a Seq.t -> 'a list	BatList.of_seq : 'a Seq.t -> 'a list	
	CCList.of_seq_rev : 'a Seq.t -> 'a list		
		BatList.ord : 'a BatOrd.ord -> 'a list BatOrd.ord	
List.partition : ('a -> bool) -> 'a list -> 'a list * 'a list	CCList.partition : ('a -> bool) -> 'a list -> 'a list * 'a list	BatList.partition : ('a -> bool) -> 'a list -> 'a list * '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		
List.partition_map : ('a -> ('b, 'c) Either.t) -> 'a list -> 'b list * 'c list	CCList.partition_map_either : ('a -> ('b, 'c) CCEither.t) -> 'a list -> 'b list * 'c list	BatList.partition_map : ('a -> ('b, 'c) BatEither.t) -> '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		
		BatList.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b list -> unit	
	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		

Stdlib	Containers	Batteries	Base
	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		
		BatList.range : int -> [< `Downto | `To] -> int -> int list	
	CCList.range_by : step:int -> int -> int -> int list		
	CCList.reduce : ('a -> 'a -> 'a) -> 'a list -> 'a option		
	CCList.reduce_exn : ('a -> 'a -> 'a) -> 'a list -> 'a	BatList.reduce : ('a -> 'a -> 'a) -> 'a list -> 'a	
	CCList.remove : eq:('a -> 'a -> bool) -> key:'a -> 'a list -> 'a list		
		BatList.remove : 'a list -> 'a -> 'a list	
		BatList.remove_all : 'a list -> 'a -> 'a list	
List.remove_assoc : 'a -> ('a * 'b) list -> ('a * 'b) list	CCList.remove_assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) list -> ('a * 'b) list	BatList.remove_assoc : 'a -> ('a * 'b) list -> ('a * 'b) list	
List.remove_assq : 'a -> ('a * 'b) list -> ('a * 'b) list	CCList.remove_assq : 'a -> ('a * 'b) list -> ('a * 'b) list	BatList.remove_assq : 'a -> ('a * 'b) list -> ('a * 'b) list	
	CCList.remove_at_idx : int -> 'a list -> 'a list	BatList.remove_at : int -> 'a list -> 'a list	
		BatList.remove_if : ('a -> bool) -> '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		
List.rev : 'a list -> 'a list	CCList.rev : 'a list -> 'a list	BatList.rev : 'a list -> 'a list	
List.rev_append : 'a list -> 'a list -> 'a list	CCList.rev_append : 'a list -> 'a list -> 'a list	BatList.rev_append : 'a list -> 'a list -> 'a list	
List.rev_map : ('a -> 'b) -> 'a list -> 'b list	CCList.rev_map : ('a -> 'b) -> 'a list -> 'b list	BatList.rev_map : ('a -> 'b) -> 'a list -> 'b list	
List.rev_map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	CCList.rev_map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	BatList.rev_map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list	
		BatList.rfind : ('a -> bool) -> 'a list -> 'a	
		BatList.rindex_of : 'a -> 'a list -> int option	
		BatList.rindex_ofq : 'a -> 'a list -> int option	
	CCList.scan_left : ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc list		
	CCList.set_at_idx : int -> 'a -> 'a list -> 'a list		
		BatList.shuffle : ?state:Random.State.t -> 'a list -> 'a list	
		BatList.singleton : 'a -> 'a list	
List.sort : ('a -> 'a -> int) -> 'a list -> 'a list	CCList.sort : ('a -> 'a -> int) -> 'a list -> 'a list	BatList.sort : ('a -> 'a -> int) -> 'a list -> 'a list	
List.sort_uniq : ('a -> 'a -> int) -> 'a list -> 'a list	CCList.sort_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list	BatList.sort_uniq : ('a -> 'a -> int) -> 'a list -> 'a list	
		BatList.sort_unique : ('a -> 'a -> int) -> 'a list -> 'a list	
	CCList.sorted_insert : cmp:('a -> 'a -> int) -> ?uniq:bool -> 'a -> 'a list -> 'a list		
	CCList.sorted_merge : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list		

Stdlib	Containers	Batteries	Base
	CCList.sorted_merge_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list		
		BatList.span : ('a -> bool) -> 'a list -> 'a list * 'a list	
List.split : ('a * 'b) list -> 'a list * 'b list	CCList.split : ('a * 'b) list -> 'a list * 'b list	BatList.split : ('a * 'b) list -> 'a list * 'b list	
		BatList.split_at : int -> 'a list -> 'a list * 'a list	
		BatList.split_nth : int -> 'a list -> 'a list * 'a list	
List.stable_sort : ('a -> 'a -> int) -> 'a list -> 'a list	CCList.stable_sort : ('a -> 'a -> int) -> 'a list -> 'a list	BatList.stable_sort : ('a -> 'a -> int) -> 'a list -> 'a list	
	CCList.sublists_of_len : ?last:('a list -> 'a list option) -> ?offset:int -> int -> 'a list -> 'a list list		
	CCList.subset : eq:('a -> 'a -> bool) -> 'a list -> 'a list -> bool		
		BatList.subset : ('a -> 'b -> int) -> 'a list -> 'b list -> bool	
		BatList.sum : int list -> int	
	CCList.tail_opt : 'a list -> 'a list option		
	CCList.take : int -> 'a list -> 'a list	BatList.take : int -> 'a list -> 'a list	
	CCList.take_drop : int -> 'a list -> 'a list * 'a list		
	CCList.take_drop_while : ('a -> bool) -> 'a list -> 'a list * 'a list		
	CCList.take_while : ('a -> bool) -> 'a list -> 'a list	BatList.take_while : ('a -> bool) -> 'a list -> 'a list	
		BatList.takedrop : int -> 'a list -> 'a list * 'a list	
		BatList.takewhile : ('a -> bool) -> 'a list -> 'a list	
List.tl : 'a list -> 'a list	CCList.tl : 'a list -> 'a list	BatList.tl : 'a list -> 'a list	
	CCList.to_gen : 'a list -> 'a CCList.gen		
	CCList.to_iter : 'a list -> 'a CCList.iter		
List.to_seq : 'a list -> 'a Seq.t	CCList.to_seq : 'a list -> 'a Seq.t	BatList.to_seq : 'a list -> 'a Seq.t	
	CCList.to_string : ?start:string -> ?stop:string -> ?sep:string -> ('a -> string) -> 'a list -> string		
		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	
	CCList.union : eq:('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list		
	CCList.uniq : eq:('a -> 'a -> bool) -> 'a list -> 'a list	BatList.unique : ?eq:('a -> 'a -> bool) -> 'a list -> 'a list	
	CCList.uniq_succ : eq:('a -> 'a -> bool) -> 'a list -> 'a list		
		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	
Stdlib	Containers	Batteries	Base
		BatMap.(-->) : ('a, 'b) map -> 'a -> 'b	
		BatMap.(<--) : ('a, 'b) map -> 'a * 'b -> ('a, 'b) map	
Map.Make.add	CCMap.Make.add	BatMap.add : 'a -> 'b -> ('a, 'b) map -> ('a, 'b) map	Base.Map.add_exn : ('k, 'v, 'cmp) map -> key:'k -> data:'v -> ('k, 'v, 'cmp) map
		BatMap.add_carry : 'a -> 'b -> ('a, 'b) map -> ('a, 'b) map * 'b option	
			Base.Map.add : ('k, 'v, 'cmp) map -> key:'k -> data:'v -> ('k, 'v, 'cmp) map Base.Map.Or_duplicate.t
	CCMap.Make.add_iter		

Stdlib	Containers	Batteries	Base
	CCMap.Make.add_iter_with		
	CCMap.Make.add_list		
	CCMap.Make.add_list_with		
			Base.Map.add_multi : ('k, 'v list, 'cmp) map -> key:'k -> data:'v -> ('k, 'v list, 'cmp) map
Map.Make.add_seq	CCMap.Make.add_seq	BatMap.add_seq : ('key * 'a) BatSeq.t -> ('key, 'a) map -> ('key, 'a) map	
	CCMap.Make.add_seq_with		
		BatMap.any : ('key, 'a) map -> 'key * 'a	
			Base.Map.append : lower_part:('k, 'v, 'cmp) map -> upper_part:('k, 'v, 'cmp) map -> [`Ok of ('k, 'v, 'cmp) map | `Overlapping_key_ranges]
		BatMap.at_rank_exn : int -> ('key, 'a) map -> 'key * 'a	
		BatMap.backwards : ('a, 'b) map -> ('a * 'b) BatEnum.t	
			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
			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
Map.Make.bindings	CCMap.Make.bindings	BatMap.bindings : ('key, 'a) map -> ('key * 'a) list	
Map.Make.cardinal	CCMap.Make.cardinal	BatMap.cardinal : ('a, 'b) map -> int	
			Base.Map.change : ('k, 'v, 'cmp) map -> 'k -> f:('v option -> 'v option) -> ('k, 'v, 'cmp) map
Map.Make.choose	CCMap.Make.choose	BatMap.choose : ('key, 'a) map -> 'key * 'a	
Map.Make.choose_opt	CCMap.Make.choose_opt	BatMap.choose_opt : ('key, 'a) map -> ('key * 'a) option	
			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
Map.Make.compare	CCMap.Make.compare	BatMap.compare : ('b -> 'b -> int) -> ('a, 'b) map -> ('a, 'b) map -> int	Base.Map.compare_direct : ('v -> 'v -> int) -> ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> int
			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
		BatMap.diff : ('a, 'b) map -> ('a, 'b) map -> ('a, 'b) map	
Map.Make.empty	CCMap.Make.empty	BatMap.empty : ('a, 'b) map	Base.Map.empty : ('a, 'cmp) Base.Map.comparator -> ('a, 'b, 'cmp) map
		BatMap.enum : ('a, 'b) map -> ('a * 'b) BatEnum.t	
Map.Make.equal	CCMap.Make.equal	BatMap.equal : ('b -> 'b -> bool) -> ('a, 'b) map -> ('a, 'b) map -> bool	Base.Map.equal : ('v -> 'v -> bool) -> ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> bool
			Base.Map.equal_m__t : (module Base.Map.Equal_m) -> ('v -> 'v -> bool) -> ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> bool
Map.Make.exists	CCMap.Make.exists	BatMap.exists : ('a -> 'b -> bool) -> ('a, 'b) map -> bool	Base.Map.exists : ('k, 'v, 'a) map -> f:('v -> bool) -> bool
			Base.Map.existsi : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> bool) -> bool
		BatMap.extract : 'a -> ('a, 'b) map -> 'b * ('a, 'b) map	
Map.Make.filter	CCMap.Make.filter	BatMap.filter : ('key -> 'a -> bool) -> ('key, 'a) map -> ('key, 'a) map	Base.Map.filter : ('k, 'v, 'cmp) map -> f:('v -> bool) -> ('k, 'v, 'cmp) map

Stdlib	Containers	Batteries	Base
			Base.Map.filter_keys : ('k, 'v, 'cmp) map -> f:('k -> bool) -> ('k, 'v, 'cmp) map
Map.Make.filter_map	CCMap.Make.filter_map	BatMap.filter_map : ('key -> 'a -> 'b option) -> ('key, 'a) map -> ('key, 'b) map	Base.Map.filter_map : ('k, 'v1, 'cmp) map -> f:('v1 -> 'v2 option) -> ('k, 'v2, '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
		BatMap.filterv : ('a -> bool) -> ('key, 'a) map -> ('key, 'a) map	
Map.Make.find	CCMap.Make.find	BatMap.find : 'a -> ('a, 'b) map -> 'b	Base.Map.find_exn : ('k, 'v, 'cmp) map -> 'k -> 'v
		BatMap.find_default : 'b -> 'a -> ('a, 'b) map -> 'b	
			Base.Map.find : ('k, 'v, 'cmp) map -> 'k -> 'v option
Map.Make.find_first	CCMap.Make.find_first	BatMap.find_first : ('a -> bool) -> ('a, 'b) map -> 'a * 'b	
Map.Make.find_first_opt	CCMap.Make.find_first_opt	BatMap.find_first_opt : ('a -> bool) -> ('a, 'b) map -> ('a * 'b) option	
Map.Make.find_last	CCMap.Make.find_last	BatMap.find_last : ('a -> bool) -> ('a, 'b) map -> 'a * 'b	
Map.Make.find_last_opt	CCMap.Make.find_last_opt	BatMap.find_last_opt : ('a -> bool) -> ('a, 'b) map -> ('a * 'b) option	
			Base.Map.find_multi : ('k, 'v list, 'cmp) map -> 'k -> 'v list
Map.Make.find_opt	CCMap.Make.find_opt	BatMap.find_opt : 'a -> ('a, 'b) map -> 'b option	
Map.Make.fold	CCMap.Make.fold	BatMap.fold : ('b -> 'c -> 'c) -> ('a, 'b) map -> 'c -> 'c	Base.Map.fold : ('k, 'v, 'b) map -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a
			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
		BatMap.foldi : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) map -> 'c -> 'c	
			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
Map.Make.for_all	CCMap.Make.for_all	BatMap.for_all : ('a -> 'b -> bool) -> ('a, 'b) map -> bool	Base.Map.for_all : ('k, 'v, 'a) map -> f:('v -> bool) -> bool
			Base.Map.for_alli : ('k, 'v, 'a) map -> f:(key:'k -> data:'v -> bool) -> bool
	CCMap.Make.get		
	CCMap.Make.get_or		
			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
		BatMap.intersect : ('b -> 'c -> 'd) -> ('a, 'b) map -> ('a, 'c) map -> ('a, 'd) map	
			Base.Map.invariants : ('a, 'b, 'c) map -> bool
Map.Make.is_empty	CCMap.Make.is_empty	BatMap.is_empty : ('a, 'b) map -> bool	Base.Map.is_empty : ('a, 'b, 'c) map -> bool
Map.Make.iter	CCMap.Make.iter	BatMap.iter : ('a -> 'b -> unit) -> ('a, 'b) map -> unit	Base.Map.iter : ('a, 'v, 'b) map -> f:('v -> unit) -> unit
			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
	CCMap.Make.keys	BatMap.keys : ('a, 'b) map -> 'a BatEnum.t	Base.Map.keys : ('k, 'a, 'b) map -> 'k list

Stdlib	Containers	Batteries	Base
			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 ...)
Map.Make.map	CCMap.Make.map	BatMap.map : ('b -> 'c) -> ('a, 'b) map -> ('a, 'c) map	Base.Map.map : ('k, 'v1, 'cmp) map -> f:('v1 -> 'v2) -> ('k, 'v2, 'cmp) map
Map.Make.mapi	CCMap.Make.mapi	BatMap.mapi : ('a -> 'b -> 'c) -> ('a, 'b) map -> ('a, 'c) map	Base.Map.mapi : ('k, 'v1, 'cmp) map -> f:(key:'k -> data:'v1 -> 'v2) -> ('k, 'v2, 'cmp) map
Map.Make.max_binding	CCMap.Make.max_binding	BatMap.max_binding : ('key, 'a) map -> 'key * 'a	Base.Map.max_elt_exn : ('k, 'v, 'a) map -> 'k * 'v
Map.Make.max_binding_opt	CCMap.Make.max_binding_opt	BatMap.max_binding_opt : ('key, 'a) map -> ('key * 'a) option	Base.Map.max_elt : ('k, 'v, 'a) map -> ('k * 'v) option
Map.Make.mem	CCMap.Make.mem	BatMap.mem : 'a -> ('a, 'b) map -> bool	Base.Map.mem : ('k, 'a, 'cmp) map -> 'k -> bool
Map.Make.merge	CCMap.Make.merge	BatMap.merge : ('key -> 'a option -> 'b option -> 'c option) -> ('key, 'a) map -> ('key, 'b) map -> ('key, 'c) map	
			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
	CCMap.Make.merge_safe		
			Base.Map.merge_skewed : ('k, 'v, 'cmp) map -> ('k, 'v, 'cmp) map -> combine:(key:'k -> 'v -> 'v -> 'v) -> ('k, 'v, 'cmp) map
Map.Make.min_binding	CCMap.Make.min_binding	BatMap.min_binding : ('key, 'a) map -> 'key * 'a	Base.Map.min_elt_exn : ('k, 'v, 'a) map -> 'k * 'v
Map.Make.min_binding_opt	CCMap.Make.min_binding_opt	BatMap.min_binding_opt : ('key, 'a) map -> ('key * 'a) option	Base.Map.min_elt : ('k, 'v, 'a) map -> ('k * 'v) option
		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	
			Base.Map.nth : ('k, 'v, 'a) map -> int -> ('k * 'v) option
			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
		BatMap.of_enum : ('a * 'b) BatEnum.t -> ('a, 'b) 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
	CCMap.Make.of_iter		
	CCMap.Make.of_iter_with		
			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]
	CCMap.Make.of_list		
	CCMap.Make.of_list_with		
Map.Make.of_seq	CCMap.Make.of_seq	BatMap.of_seq : ('key * 'a) BatSeq.t -> ('key, 'a) map	

Stdlib	Containers	Batteries	Base
	CCMap.Make.of_seq_with		
			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
Map.Make.partition	CCMap.Make.partition	BatMap.partition : ('a -> 'b -> bool) -> ('a, 'b) map -> ('a, 'b) map * ('a, 'b) map	Base.Map.partition_tf : ('k, 'v, 'cmp) map -> f:('v -> bool) -> ('k, 'v, 'cmp) map * ('k, 'v, '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.partitioni_tf : ('k, 'v, 'cmp) map -> f:(key:'k -> data:'v -> bool) -> ('k, 'v, 'cmp) map * ('k, 'v, 'cmp) 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	
	CCMap.Make.pp		
		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	
			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
Map.Make.remove	CCMap.Make.remove	BatMap.remove : 'a -> ('a, 'b) map -> ('a, 'b) map	Base.Map.remove : ('k, 'v, 'cmp) map -> 'k -> ('k, 'v, 'cmp) map
		BatMap.remove_exn : 'a -> ('a, 'b) map -> ('a, 'b) map	
			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
Map.Make.singleton	CCMap.Make.singleton	BatMap.singleton : 'a -> 'b -> ('a, 'b) map	Base.Map.singleton : ('a, 'cmp) Base.Map.comparator -> 'a -> 'b -> ('a, 'b, 'cmp) map
Map.Make.split	CCMap.Make.split	BatMap.split : 'key -> ('key, 'a) map -> ('key, 'a) map * 'a option * ('key, 'a) map	Base.Map.split : ('k, 'v, 'cmp) map -> 'k -> ('k, 'v, 'cmp) map * ('k * 'v) option * ('k, 'v, 'cmp) map
			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

Stdlib	**Containers**	**Batteries**	**Base**
			Base.Map.to_alist : ?key_order:[`Decreasing | `Increasing] -> ('k, 'v, 'a) map -> ('k * 'v) list
	CCMap.Make.to_iter		
	CCMap.Make.to_list		
Map.Make.to_rev_seq	CCMap.Make.to_rev_seq	BatMap.to_rev_seq : ('key, 'a) map -> ('key * 'a) BatSeq.t	
Map.Make.to_seq	CCMap.Make.to_seq	BatMap.to_seq : ('key, 'a) map -> ('key * 'a) BatSeq.t	
Map.Make.to_seq_from	CCMap.Make.to_seq_from	BatMap.to_seq_from : 'key -> ('key, 'a) map -> ('key * 'a) BatSeq.t	
			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
Map.Make.union	CCMap.Make.union	BatMap.union : ('a, 'b) map -> ('a, 'b) map -> ('a, 'b) map	
		BatMap.union_stdlib : ('key -> 'a -> 'a -> 'a option) -> ('key, 'a) map -> ('key, 'a) map -> ('key, 'a) map	
Map.Make.update	CCMap.Make.update	BatMap.update : 'a -> 'a -> 'b -> ('a, 'b) map -> ('a, 'b) map	Base.Map.update : ('k, 'v, 'cmp) map -> 'k -> f:('v option -> 'v) -> ('k, 'v, 'cmp) map
		BatMap.update_stdlib : 'a -> ('b option -> 'b option) -> ('a, 'b) map -> ('a, 'b) map	
			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
	CCMap.Make.values	BatMap.values : ('a, 'b) map -> 'b BatEnum.t	
Stdlib	Containers	Batteries	Base
	CCOpt.(<$>) : ('a -> 'b) -> 'a option -> 'b option		
	CCOpt.(<*>) : ('a -> 'b) option -> 'a option -> 'b option		
	CCOpt.(<+>) : 'a option -> 'a option -> 'a option		
	CCOpt.(>>=) : 'a option -> ('a -> 'b option) -> 'b option		Base.Option.(>>=) : 'a option -> ('a -> 'b option) -> 'b option
	CCOpt.(>|=) : 'a option -> ('a -> 'b) -> 'b option		Base.Option.(>>|) : 'a option -> ('a -> 'b) -> 'b option
		BatOption.(|?) : 'a option -> 'a -> 'a	
	CCOpt.(and*) : 'a option -> 'b option -> ('a * 'b) option		
	CCOpt.(and+) : 'a option -> 'b option -> ('a * 'b) option		
	CCOpt.(let*) : 'a option -> ('a -> 'b option) -> 'b option		
	CCOpt.(let+) : 'a option -> ('a -> 'b) -> 'b option		
		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	
Option.None : 'a option = Option.None			Base.Option.None : 'a option = Base.Option.None
			Base.Option.all : 'a option list -> 'a list option
			Base.Option.all_unit : unit option list -> unit option
		BatOption.apply : ('a -> 'a) option -> 'a -> 'a	
Option.bind : 'a option -> ('a -> 'b option) -> 'b option	CCOpt.bind : 'a option -> ('a -> 'b option) -> 'b option	BatOption.bind : 'a option -> ('a -> 'b option) -> 'b option	Base.Option.bind : 'a option -> f:('a -> 'b option) -> 'b option
			Base.Option.both : 'a option -> 'b option -> ('a * 'b) option
			Base.Option.call : 'a -> f:('a -> unit) option -> unit
	CCOpt.choice : 'a option list -> 'a option		
	CCOpt.choice_iter : 'a option CCOpt.iter -> 'a option		

Stdlib	Containers	Batteries	Base
	CCOpt.choice_seq : 'a option Seq.t -> 'a option		
Option.compare : ('a -> 'a -> int) -> 'a option -> 'a option -> int	CCOpt.compare : ('a -> 'a -> int) -> 'a option -> 'a option -> int	BatOption.compare : ?cmp:('a -> 'a -> int) -> 'a option -> 'a option -> int	Base.Option.compare : ('a -> 'a -> int) -> 'a option -> 'a option -> int
			Base.Option.count : 'a option -> f:('a -> bool) -> int
		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	
Option.equal : ('a -> 'a -> bool) -> 'a option -> 'a option -> bool	CCOpt.equal : ('a -> 'a -> bool) -> 'a option -> 'a option -> bool		Base.Option.equal : 'a Base.Equal.equal -> 'a option Base.Equal.equal
	CCOpt.exists : ('a -> bool) -> 'a option -> bool		Base.Option.exists : 'a option -> f:('a -> bool) -> bool
	CCOpt.filter : ('a -> bool) -> 'a option -> 'a option	BatOption.filter : ('a -> bool) -> 'a option -> 'a option	Base.Option.filter : 'a option -> f:('a -> bool) -> 'a option
			Base.Option.find : 'a option -> f:('a -> bool) -> 'a option
	CCOpt.flat_map : ('a -> 'b option) -> 'a option -> 'b option		Base.Option.find_map : 'a option -> f:('a -> 'b option) -> 'b option
			Base.Option.first_some : 'a option -> 'a option -> 'a option
Option.fold : none:'a -> some:('b -> 'a) -> 'b option -> 'a	CCOpt.fold : ('a -> 'b -> 'a) -> 'a -> 'b option -> 'a		Base.Option.fold : 'a option -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
			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
	CCOpt.for_all : ('a -> bool) -> 'a option -> bool		Base.Option.for_all : 'a option -> f:('a -> bool) -> bool
Option.get : 'a option -> 'a		BatOption.get : 'a option -> 'a	
		BatOption.get_exn : 'a option -> exn -> 'a	
	CCOpt.get_exn_or : string -> 'a option -> 'a		
	CCOpt.get_lazy : (unit -> 'a) -> 'a option -> 'a		
	CCOpt.get_or : default:'a -> 'a option -> 'a		
			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
	CCOpt.if_ : ('a -> bool) -> 'a -> 'a option		
			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
Option.is_none : 'a option -> bool	CCOpt.is_none : 'a option -> bool	BatOption.is_none : 'a option -> bool	Base.Option.is_none : 'a option -> bool
Option.is_some : 'a option -> bool	CCOpt.is_some : 'a option -> bool	BatOption.is_some : 'a option -> bool	Base.Option.is_some : 'a option -> bool
Option.iter : ('a -> unit) -> 'a option -> unit	CCOpt.iter : ('a -> unit) -> 'a option -> unit		Base.Option.iter : 'a option -> f:('a -> unit) -> unit
Option.join : 'a option option -> 'a option	CCOpt.flatten : 'a option option -> 'a option		Base.Option.join : 'a option option -> 'a option
			Base.Option.length : 'a option -> int
Option.map : ('a -> 'b) -> 'a option -> 'b option	CCOpt.map : ('a -> 'b) -> 'a option -> 'b option	BatOption.map : ('a -> 'b) -> 'a option -> 'b option	Base.Option.map : 'a option -> f:('a -> 'b) -> 'b option
	CCOpt.map2 : ('a -> 'b -> 'c) -> 'a option -> 'b option -> 'c option		Base.Option.map2 : 'a option -> 'b option -> f:('a -> 'b -> 'c) -> 'c option
		BatOption.map_default : ('a -> 'b) -> 'b -> 'a option -> 'b	
		BatOption.map_default_delayed : ('a -> 'b) -> (unit -> 'b) -> 'a option -> 'b	

Stdlib	Containers	Batteries	Base
	CCOpt.map_lazy : (unit -> 'b) -> ('a -> 'b) -> 'a option -> 'b		
	CCOpt.map_or : default:'b -> ('a -> 'b) -> 'a option -> 'b		
			Base.Option.max_elt : 'a option -> compare:('a -> 'a -> int) -> 'a option
		BatOption.may : ('a -> unit) -> 'a option -> unit	
			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
		BatOption.of_enum : 'a BatEnum.t -> 'a option	
	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		
		BatOption.ord : 'a BatOrd.ord -> 'a option BatOrd.ord	
	CCOpt.pp : 'a CCOpt.printer -> 'a option CCOpt.printer		
		BatOption.print : ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b option -> unit	
	CCOpt.pure : 'a -> 'a option		
	CCOpt.random : 'a CCOpt.random_gen -> 'a option CCOpt.random_gen		
	CCOpt.return : 'a -> 'a option		Base.Option.return : 'a -> 'a option
	CCOpt.return_if : bool -> 'a -> 'a option		
	CCOpt.sequence_l : 'a option list -> 'a list option		
			Base.Option.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a option -> Sexplib0__.Sexp.t
		BatOption.some : 'a -> 'a option	Base.Option.some : 'a -> 'a option
			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
	CCOpt.to_gen : 'a option -> 'a CCOpt.gen		
	CCOpt.to_iter : 'a option -> 'a CCOpt.iter		
Option.to_list : 'a option -> 'a list	CCOpt.to_list : 'a option -> 'a list		Base.Option.to_list : 'a option -> 'a list
Option.to_result : none:'e -> 'a option -> ('a, 'e) result	CCOpt.to_result : 'e -> 'a option -> ('a, 'e) result		
	CCOpt.to_result_lazy : (unit -> 'e) -> 'a option -> ('a, 'e) result		
Option.to_seq : 'a option -> 'a Seq.t	CCOpt.to_seq : 'a option -> 'a Seq.t		
			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
Option.value : 'a option -> default:'a -> 'a	CCOpt.value : 'a option -> default:'a -> 'a		Base.Option.value : 'a option -> default:'a -> 'a
			Base.Option.value_exn : ?here:Base.Source_code_position0.t -> ?error:Base.Error.t -> ?message:string -> 'a option -> 'a

Stdlib	Containers	Batteries	Base
			Base.Option.value_map : 'a option -> default:'b -> f:('a -> 'b) -> 'b
	CCOpt.wrap : ?handler:(exn -> bool) -> ('a -> 'b) -> 'a -> 'b option		
	CCOpt.wrap2 : ?handler:(exn -> bool) -> ('a -> 'b -> 'c) -> 'a -> 'b -> 'c option		
			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
			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
Stdlib	(* Containers - empty *)	Batteries	Base
Printf.bprintf : Buffer.t -> ('a, Buffer.t, unit) format -> 'a		BatPrintf.bprintf : Buffer.t -> ('a, Buffer.t, unit) BatPrintf.t -> 'a	Base.Printf.bprintf : Base.Import0.Caml.Buffer.t -> ('r, Base.Import0.Caml.Buffer.t, unit) format -> 'r
		BatPrintf.bprintf2 : Buffer.t -> ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	
Printf.eprintf : ('a, out_channel, unit) format -> 'a		BatPrintf.eprintf : ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	(* Base.Printf.eprintf *)
			Base.Printf.failwithf : ('r, unit, string, unit -> 'a) format4 -> 'r
Printf.fprintf : out_channel -> ('a, out_channel, unit) format -> 'a		BatPrintf.fprintf : 'a BatInnerIO.output -> ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	(* Base.Printf.fprintf *)

Stdlib	**Containers**	**Batteries**	**Base**
Printf.ibprintf : Buffer.t -> ('a, Buffer.t, unit) format -> 'a			(* Base.Printf.ibprintf *)
Printf.ifprintf : 'b -> ('a, 'b, 'c, unit) format4 -> 'a		BatPrintf.ifprintf : 'c -> ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	Base.Printf.ifprintf : 'a -> ('r, 'a, 'c, unit) format4 -> 'r
Printf.ikbprintf : (Buffer.t -> 'd) -> Buffer.t -> ('a, Buffer.t, unit, 'd) format4 -> 'a			(* Base.Printf.ikbprintf *)
Printf.ikfprintf : ('b -> 'd) -> 'b -> ('a, 'b, 'c, 'd) format4 -> 'a			(* Base.Printf.ikfprintf *)
			Base.Printf.invalid_argf : ('r, unit, string, unit -> 'a) format4 -> 'r
Printf.kbprintf : (Buffer.t -> 'd) -> Buffer.t -> ('a, Buffer.t, unit, 'd) format4 -> 'a		BatPrintf.kbprintf : (Buffer.t -> 'a) -> Buffer.t -> ('b, Buffer.t, unit, 'a) format4 -> 'b	Base.Printf.kbprintf : (Base.Import0.Caml.Buffer.t -> 'a) -> Base.Import0.Caml.Buffer.t -> ('r, Base.Import0.Caml.Buffer.t, unit, 'a) format4 -> 'r
		BatPrintf.kbprintf2 : (Buffer.t -> 'b) -> Buffer.t -> ('c, 'a BatInnerIO.output, unit, 'b) format4 -> 'c	
Printf.kfprintf : (out_channel -> 'd) -> out_channel -> ('a, out_channel, unit, 'd) format4 -> 'a		BatPrintf.kfprintf : ('a BatInnerIO.output -> 'b) -> 'a BatInnerIO.output -> ('c, 'a BatInnerIO.output, unit, 'b) format4 -> 'c	(* Base.Printf.kfprintf *)
Printf.kprintf : (string -> 'b) -> ('a, unit, string, 'b) format4 -> 'a		BatPrintf.kprintf : (string -> 'a) -> ('b, unit, string, 'a) format4 -> 'b	(* Base.Printf.kprintf *)
Printf.ksprintf : (string -> 'd) -> ('a, unit, string, 'd) format4 -> 'a		BatPrintf.ksprintf : (string -> 'a) -> ('b, unit, string, 'a) format4 -> 'b	Base.Printf.ksprintf : (string -> 'a) -> ('r, unit, string, 'a) format4 -> 'r
		BatPrintf.ksprintf2 : (string -> 'b) -> ('c, 'a BatInnerIO.output, unit, 'b) format4 -> 'c	
Printf.printf : ('a, out_channel, unit) format -> 'a		BatPrintf.printf : ('b, 'a BatInnerIO.output, unit) BatPrintf.t -> 'b	(* Base.Printf.printf *)
Printf.sprintf : ('a, unit, string) format -> 'a		BatPrintf.sprintf : ('a, unit, string) BatPrintf.t -> 'a	Base.Printf.sprintf : ('r, unit, string) format -> 'r
		BatPrintf.sprintf2 : ('a, 'b BatInnerIO.output, unit, string) format4 -> 'a	
Stdlib	Containers	Batteries	Base
	CCResult.(<$>) : ('a -> 'b) -> ('a, 'err) result -> ('b, 'err) result		
	CCResult.(<*>) : ('a -> 'b, 'err) result -> ('a, 'err) result -> ('b, 'err) result		
	CCResult.(>>=) : ('a, 'err) result -> ('a -> ('b, 'err) result) -> ('b, 'err) result		Base.Result.(>>=) : ('a, 'e) result-> ('a -> ('b, 'e) result) -> ('b, 'e) result
	CCResult.(>|=) : ('a, 'err) result -> ('a -> 'b) -> ('b, 'err) result		Base.Result.(>>|) : ('a, 'e) result-> ('a -> 'b) -> ('b, 'e) 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		
			Base.Result.all : ('a, 'e) result list -> ('a list, 'e) result
			Base.Result.all_unit : (unit, 'e) result list -> (unit, 'e) result
Result.bind : ('a, 'e) result -> ('a -> ('b, 'e) result) -> ('b, 'e) result		BatResult.bind : ('a, 'e) result -> ('a -> ('b, 'e) result) -> ('b, 'e) result	Base.Result.bind : ('a, 'e) result-> f:('a -> ('b, 'e) result) -> ('b, 'e) result
	CCResult.both : ('a, 'err) result -> ('b, 'err) result -> ('a * 'b, 'err) result		
	CCResult.catch : ('a, 'err) result -> ok:('a -> 'b) -> err:('err -> 'b) -> 'b		
		BatResult.catch : ('a -> 'e) -> 'a -> ('e, exn) result	

Stdlib	Containers	Batteries	Base
		BatResult.catch2 : ('a -> 'b -> 'c) -> 'a -> 'b -> ('c, exn) result	
		BatResult.catch3 : ('a -> 'b -> 'c -> 'd) -> 'a -> 'b -> 'c -> ('d, exn) result	
	CCResult.choose : ('a, 'err) result list -> ('a, 'err list) 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
Result.compare : ok:('a -> 'a -> int) -> error:('e -> 'e -> int) -> ('a, 'e) result -> ('a, 'e) result -> int	CCResult.compare : err:'err CCResult.ord -> 'a CCResult.ord -> ('a, 'err) result CCResult.ord	BatResult.compare : ok:('a -> 'a -> int) -> error:('e -> 'e -> int) -> ('a, 'e) result -> ('a, 'e) result -> int	Base.Result.compare : ('ok -> 'ok -> int) -> ('err -> 'err -> int) -> ('ok, 'err) result-> ('ok, 'err) result-> int
		BatResult.default : 'a -> ('a, 'b) result -> 'a	
Result.equal : ok:('a -> 'a -> bool) -> error:('e -> 'e -> bool) -> ('a, 'e) result -> ('a, 'e) result -> bool	CCResult.equal : err:'err CCResult.equal -> 'a CCResult.equal -> ('a, 'err) result CCResult.equal	BatResult.equal : ok:('a -> 'a -> bool) -> error:('e -> 'e -> bool) -> ('a, 'e) result -> ('a, 'e) result -> bool	Base.Result.equal : ('ok -> 'ok -> bool) -> ('err -> 'err -> bool) -> ('ok, 'err) result-> ('ok, 'err) result-> bool
Result.error : 'e -> ('a, 'e) result	CCResult.fail : 'err -> ('a, 'err) result	BatResult.error : 'e -> ('a, 'e) result	Base.Result.fail : 'err -> ('a, 'err) result
			Base.Result.error : ('a, 'err) result-> 'err option
			Base.Result.failf : ('a, unit, string, ('b, string) result) format4 -> 'a
	CCResult.fail_fprintf : ('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		
Result.fold : ok:('a -> 'c) -> error:('e -> 'c) -> ('a, 'e) result -> 'c	CCResult.fold : ok:('a -> 'b) -> error:('err -> 'b) -> ('a, 'err) result -> 'b	BatResult.fold : ok:('a -> 'c) -> error:('e -> 'c) -> ('a, 'e) result -> 'c	
	CCResult.fold_iter : ('b -> 'a -> ('b, 'err) result) -> 'b -> 'a CCResult.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		
		BatResult.get : ('a, exn) result -> 'a	
Result.get_error : ('a, 'e) result -> 'e		BatResult.get_error : ('a, 'e) result -> 'e	
	CCResult.get_lazy : ('b -> 'a) -> ('a, 'b) result -> 'a		
Result.get_ok : ('a, 'e) result -> 'a	CCResult.get_exn : ('a, 'b) result -> 'a	BatResult.get_ok : ('a, 'e) 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		
	CCResult.guard_str : (unit -> 'a) -> ('a, string) result		
	CCResult.guard_str_trace : (unit -> 'a) -> ('a, string) result		
			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

Stdlib	Containers	Batteries	Base
		BatResult.is_bad : ('a, 'e) result -> bool	
Result.is_error : ('a, 'e) result -> bool	CCResult.is_error : ('a, 'err) result -> bool	BatResult.is_error : ('a, 'e) result -> bool	Base.Result.is_error : ('a, 'b) result-> bool
		BatResult.is_exn : exn -> ('a, exn) result -> bool	
Result.is_ok : ('a, 'e) result -> bool	CCResult.is_ok : ('a, 'err) result -> bool	BatResult.is_ok : ('a, 'e) result -> bool	Base.Result.is_ok : ('a, 'b) result-> bool
Result.iter : ('a -> unit) -> ('a, 'e) result -> unit	CCResult.iter : ('a -> unit) -> ('a, 'b) result -> unit	BatResult.iter : ('a -> unit) -> ('a, 'e) result -> unit	Base.Result.iter : ('ok, 'a) result-> f:('ok -> unit) -> unit
Result.iter_error : ('e -> unit) -> ('a, 'e) result -> unit	CCResult.iter_err : ('err -> unit) -> ('a, 'err) result -> unit	BatResult.iter_error : ('e -> unit) -> ('a, 'e) result -> unit	Base.Result.iter_error : ('a, 'err) result-> f:('err -> unit) -> unit
Result.join : (('a, 'e) result, 'e) result -> ('a, 'e) result	CCResult.join : (('a, 'err) CCResult.t, 'err) result -> ('a, 'err) result	BatResult.join : (('a, 'e) result, 'e) result -> ('a, 'e) result	Base.Result.join : (('a, 'e) result, 'e) result-> ('a, 'e) result
Result.map : ('a -> 'b) -> ('a, 'e) result -> ('b, 'e) result	CCResult.map : ('a -> 'b) -> ('a, 'err) result -> ('b, 'err) result	BatResult.map : ('a -> 'b) -> ('a, 'e) result -> ('b, 'e) result	Base.Result.map : ('ok, 'err) result-> f:('ok -> 'c) -> ('c, 'err) result
	CCResult.map2 : ('a -> 'b) -> ('err1 -> 'err2) -> ('a, 'err1) result -> ('b, 'err2) result		
		BatResult.map_both : ('a1 -> 'a2) -> ('b1 -> 'b2) -> ('a1, 'b1) result -> ('a2, 'b2) result	
		BatResult.map_default : 'b -> ('a -> 'b) -> ('a, 'c) result -> 'b	
Result.map_error : ('e -> 'f) -> ('a, 'e) result -> ('a, 'f) result	CCResult.map_err : ('err1 -> 'err2) -> ('a, 'err1) result -> ('a, 'err2) result	BatResult.map_error : ('e -> 'f) -> ('a, 'e) result -> ('a, 'f) result	Base.Result.map_error : ('ok, 'err) result-> f:('err -> 'c) -> ('ok, 'c) 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		
			Base.Result.of_either : ('ok, 'err) Base.Either0.t -> ('ok, 'err) result
	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		
		BatResult.of_option : 'a option -> ('a, unit) result	
			Base.Result.of_option : 'ok option -> error:'err -> ('ok, 'err) result
Result.ok : 'a -> ('a, 'e) result		BatResult.ok : 'a -> ('a, 'b) result	
			Base.Result.ok_exn : ('ok, exn) result-> 'ok
			Base.Result.ok : ('ok, 'a) result-> 'ok option
			Base.Result.ok_fst : ('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
	CCResult.pp : 'a CCResult.printer -> ('a, string) result CCResult.printer		
	CCResult.pp' : 'a CCResult.printer -> 'e CCResult.printer -> ('a, 'e) result CCResult.printer		
		BatResult.print : ('b BatInnerIO.output -> 'a -> unit) -> 'b BatInnerIO.output -> ('a, exn) result -> unit	
	CCResult.pure : 'a -> ('a, 'err) result		
	CCResult.retry : int -> (unit -> ('a, 'err) result) -> ('a, 'err list) result		
	CCResult.return : 'a -> ('a, 'err) result		Base.Result.return : 'a -> ('a, 'b) result
			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

Stdlib	**Containers**	**Batteries**	**Base**
			Base.Result.to_either : ('ok, 'err) result-> ('ok, 'err) Base.Either0.t
	CCResult.to_err : ('a, 'b) result -> ('a, 'b) CCResult.error		
	CCResult.to_iter : ('a, 'b) result -> 'a CCResult.iter		
Result.to_list : ('a, 'e) result -> 'a list		BatResult.to_list : ('a, 'e) result -> 'a list	
Result.to_option : ('a, 'e) result -> 'a option	CCResult.to_opt : ('a, 'b) result -> 'a option	BatResult.to_option : ('a, 'b) result -> 'a option	
Result.to_seq : ('a, 'e) result -> 'a Seq.t	CCResult.to_seq : ('a, 'b) result -> 'a Seq.t	BatResult.to_seq : ('a, 'e) result -> 'a BatSeq.t	
			Base.Result.try_with : (unit -> 'a) -> ('a, exn) result
Result.value : ('a, 'e) result -> default:'a -> 'a		BatResult.value : ('a, 'e) result -> default:'a -> 'a	
	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		
Stdlib	Containers	Batteries	Base
	CCSeq.(--) : int -> int -> int Seq.t	BatSeq.(--) : int -> int -> int Seq.t	
		BatSeq.(---) : int -> int -> int Seq.t	
		BatSeq.(--.) : float * float -> float -> float Seq.t	
	CCSeq.(--^) : int -> int -> int Seq.t	BatSeq.(--^) : int -> int -> int Seq.t	
		BatSeq.(--~) : char -> char -> char Seq.t	
	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.(>>=) : 'a Seq.t -> ('a -> 'b Seq.t) -> 'b Seq.t		Base.Sequence.(>>=) : 'a Base.Sequence.t -> ('a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
	CCSeq.(>|=) : 'a Seq.t -> ('a -> 'b) -> 'b Seq.t		Base.Sequence.(>>|) : 'a Base.Sequence.t -> ('a -> 'b) -> 'b Base.Sequence.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	
Seq.Nil : 'a Seq.node = Seq.Nil	CCSeq.Nil : 'a CCSeq.node = CCSeq.Nil	BatSeq.Nil : 'a BatSeq.node = BatSeq.Nil	
			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
Seq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	CCSeq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	BatSeq.append : 'a Seq.t -> 'a Seq.t -> 'a Seq.t	Base.Sequence.append : 'a Base.Sequence.t -> 'a Base.Sequence.t -> 'a Base.Sequence.t
		BatSeq.assoc : 'a -> ('a * 'b) Seq.t -> 'b option	
		BatSeq.at : 'a Seq.t -> int -> 'a	
			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
		BatSeq.combine : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t	
	CCSeq.compare : 'a CCSeq.ord -> 'a Seq.t CCSeq.ord		Base.Sequence.compare : ('a -> 'a -> int) -> 'a Base.Sequence.t -> 'a Base.Sequence.t -> int
		BatSeq.concat : 'a Seq.t Seq.t -> 'a Seq.t	Base.Sequence.concat : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
			Base.Sequence.concat_map : 'a Base.Sequence.t -> f:('a -> 'b Base.Sequence.t) -> 'b

Stdlib	Containers	Batteries	Base
			Base.Sequence.t
			Base.Sequence.concat_mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b Base.Sequence.t) -> 'b Base.Sequence.t
Seq.cons : 'a -> 'a Seq.t -> 'a Seq.t	CCSeq.cons : 'a -> 'a Seq.t -> 'a Seq.t	BatSeq.cons : 'a -> 'a Seq.t -> 'a Seq.t	
			Base.Sequence.count : 'a Base.Sequence.t -> f:('a -> bool) -> int
			Base.Sequence.counti : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> int
	CCSeq.cycle : 'a Seq.t -> 'a Seq.t		
			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
	CCSeq.drop : int -> 'a Seq.t -> 'a Seq.t	BatSeq.drop : int -> 'a Seq.t -> 'a Seq.t	Base.Sequence.drop : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
			Base.Sequence.drop_eagerly : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
	CCSeq.drop_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.drop_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.drop_while : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t
			Base.Sequence.drop_while_option : 'a Base.Sequence.t -> f:('a -> bool) -> ('a * 'a Base.Sequence.t) option
Seq.empty : 'a Seq.t	CCSeq.empty : 'a Seq.t	BatSeq.empty : 'a Seq.t	Base.Sequence.empty : 'a Base.Sequence.t
		BatSeq.enum : 'a Seq.t -> 'a BatEnum.t	
	CCSeq.equal : 'a CCSeq.equal -> 'a Seq.t CCSeq.equal	BatSeq.equal : ?eq:('a -> 'a -> bool) -> 'a Seq.t -> 'a Seq.t -> bool	Base.Sequence.equal : ('a -> 'a -> bool) -> 'a Base.Sequence.t -> 'a Base.Sequence.t -> bool
	CCSeq.exists : ('a -> bool) -> 'a Seq.t -> bool	BatSeq.exists : ('a -> bool) -> 'a Seq.t -> bool	Base.Sequence.exists : 'a Base.Sequence.t -> f:('a -> bool) -> bool
	CCSeq.exists2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
			Base.Sequence.existsi : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> 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		
Seq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	CCSeq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.filter : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.filter : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t
Seq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	CCSeq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	BatSeq.filter_map : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.filter_map : 'a Base.Sequence.t -> f:('a -> 'b option) -> 'b Base.Sequence.t
			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
		BatSeq.find : ('a -> bool) -> 'a Seq.t -> 'a option	Base.Sequence.find : 'a Base.Sequence.t -> f:('a -> bool) -> 'a option
			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
		BatSeq.find_map : ('a -> 'b option) -> 'a Seq.t -> 'b option	Base.Sequence.find_map : 'a Base.Sequence.t -> f:('a -> 'b option) -> 'b option
			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
		BatSeq.first : 'a Seq.t -> 'a	
Seq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	CCSeq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	BatSeq.flat_map : ('a -> 'b Seq.t) -> 'a Seq.t -> 'b Seq.t	
	CCSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	BatSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	
	CCSeq.fmap : ('a -> 'b option) -> 'a Seq.t -> 'b Seq.t		
	CCSeq.fold : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a		Base.Sequence.fold : 'a Base.Sequence.t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
	CCSeq.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a Seq.t -> 'b Seq.t -> 'acc		

Stdlib	Containers	Batteries	Base
Seq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	CCSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	BatSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b Seq.t -> 'a	
			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
		BatSeq.fold_right : ('a -> 'b -> 'b) -> 'a Seq.t -> 'b -> 'b	
			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
	CCSeq.for_all : ('a -> bool) -> 'a Seq.t -> bool	BatSeq.for_all : ('a -> bool) -> 'a Seq.t -> bool	Base.Sequence.for_all : 'a Base.Sequence.t -> f:('a -> bool) -> bool
			Base.Sequence.for_alli : 'a Base.Sequence.t -> f:(int -> 'a -> bool) -> bool
	CCSeq.for_all2 : ('a -> 'b -> bool) -> 'a Seq.t -> 'b Seq.t -> bool		
			Base.Sequence.force_eagerly : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.group : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t Seq.t		
			Base.Sequence.group : 'a Base.Sequence.t -> break:('a -> 'a -> bool) -> 'a list Base.Sequence.t
	CCSeq.head : 'a Seq.t -> 'a option		Base.Sequence.hd : 'a Base.Sequence.t -> 'a option
	CCSeq.head_exn : 'a Seq.t -> 'a	BatSeq.hd : 'a Seq.t -> 'a	Base.Sequence.hd_exn : 'a Base.Sequence.t -> 'a
			Base.Sequence.ignore_m : 'a Base.Sequence.t -> unit Base.Sequence.t
		BatSeq.init : int -> (int -> 'a) -> 'a Seq.t	Base.Sequence.init : int -> f:(int -> 'a) -> 'a Base.Sequence.t
	CCSeq.interleave : 'a Seq.t -> 'a Seq.t -> 'a Seq.t		Base.Sequence.interleave : 'a Base.Sequence.t Base.Sequence.t -> 'a 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
	CCSeq.is_empty : 'a Seq.t -> bool	BatSeq.is_empty : 'a Seq.t -> bool	Base.Sequence.is_empty : 'a Base.Sequence.t -> bool
Seq.iter : ('a -> unit) -> 'a Seq.t -> unit	CCSeq.iter : ('a -> unit) -> 'a Seq.t -> unit	BatSeq.iter : ('a -> unit) -> 'a Seq.t -> unit	Base.Sequence.iter : 'a Base.Sequence.t -> f:('a -> unit) -> unit
	CCSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	BatSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	
			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
	CCSeq.iteri : (int -> 'a -> unit) -> 'a Seq.t -> unit	BatSeq.iteri : (int -> 'a -> unit) -> 'a Seq.t -> unit	Base.Sequence.iteri : ('a Base.Sequence.t, 'a) Base.Indexed_container_intf.iteri
			Base.Sequence.join : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t
		BatSeq.last : 'a Seq.t -> 'a	
	CCSeq.length : 'a Seq.t -> int	BatSeq.length : 'a Seq.t -> int	Base.Sequence.length : 'a Base.Sequence.t -> int
			Base.Sequence.length_is_bounded_by : ?min:int -> ?max:int -> 'a Base.Sequence.t -> bool
		BatSeq.make : int -> 'a -> 'a Seq.t	
Seq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	CCSeq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	BatSeq.map : ('a -> 'b) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.map : 'a Base.Sequence.t -> f:('a -> 'b) -> 'b Base.Sequence.t
	CCSeq.map2 : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t	BatSeq.map2 : ('a -> 'b -> 'c) -> 'a Seq.t -> 'b Seq.t -> 'c Seq.t	
	CCSeq.mapi : (int -> 'a -> 'b) -> 'a Seq.t -> 'b Seq.t	BatSeq.mapi : (int -> 'a -> 'b) -> 'a Seq.t -> 'b Seq.t	Base.Sequence.mapi : 'a Base.Sequence.t -> f:(int -> 'a -> 'b) -> 'b Base.Sequence.t
		BatSeq.max : 'a Seq.t -> 'a	
			Base.Sequence.max_elt : 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a option
		BatSeq.mem : 'a -> 'a Seq.t -> bool	

Stdlib	Containers	Batteries	Base
			Base.Sequence.mem : 'a Base.Sequence.t -> 'a -> equal:('a -> 'a -> bool) -> bool
	CCSeq.memoize : 'a Seq.t -> 'a Seq.t		Base.Sequence.memoize : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.merge : 'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t -> 'a Seq.t		Base.Sequence.merge : 'a Base.Sequence.t -> 'a Base.Sequence.t -> compare:('a -> 'a -> int) -> 'a Base.Sequence.t
			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
		BatSeq.min : 'a Seq.t -> 'a	
			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
	CCSeq.nil : 'a Seq.t	BatSeq.nil : 'a Seq.t	
			Base.Sequence.nth : 'a Base.Sequence.t -> int -> 'a option
			Base.Sequence.nth_exn : 'a Base.Sequence.t -> int -> 'a
	CCSeq.of_array : 'a array -> 'a Seq.t		
	CCSeq.of_gen : 'a CCSeq.gen -> 'a Seq.t		
			Base.Sequence.of_lazy : 'a Base.Sequence.t Base.Lazy.t -> 'a Base.Sequence.t
	CCSeq.of_list : 'a list -> 'a Seq.t	BatSeq.of_list : 'a list -> 'a Seq.t	Base.Sequence.of_list : 'a list -> 'a Base.Sequence.t
			Base.Sequence.of_seq : 'a Base.Import.Caml.Seq.t -> 'a Base.Sequence.t
		BatSeq.of_string : ?first:string -> ?last:string -> ?sep:string -> (string -> 'a) -> string -> '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		
		BatSeq.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b Seq.t -> unit	
	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.range : int -> int -> int Seq.t		Base.Sequence.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int Base.Sequence.t
			Base.Sequence.reduce : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a option
		BatSeq.reduce : ('a -> 'a -> 'a) -> 'a Seq.t -> 'a	Base.Sequence.reduce_exn : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a
			Base.Sequence.remove_consecutive_duplicates : 'a Base.Sequence.t -> equal:('a -> 'a -> bool) -> 'a Base.Sequence.t
	CCSeq.repeat : ?n:int -> 'a -> 'a Seq.t		Base.Sequence.repeat : 'a -> 'a Base.Sequence.t
Seq.return : 'a -> 'a Seq.t	CCSeq.return : 'a -> 'a Seq.t	BatSeq.return : 'a -> 'a Seq.t	Base.Sequence.return : 'a -> '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
	CCSeq.singleton : 'a -> 'a Seq.t		Base.Sequence.singleton : 'a -> 'a Base.Sequence.t
		BatSeq.split : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t	
	CCSeq.sort : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.t		

Stdlib	Containers	Batteries	Base
	CCSeq.sort_uniq : cmp:'a CCSeq.ord -> 'a Seq.t -> 'a Seq.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
	CCSeq.tail : 'a Seq.t -> 'a Seq.t option		
	CCSeq.tail_exn : 'a Seq.t -> 'a Seq.t		
	CCSeq.take : int -> 'a Seq.t -> 'a Seq.t	BatSeq.take : int -> 'a Seq.t -> 'a Seq.t	Base.Sequence.take : 'a Base.Sequence.t -> int -> 'a Base.Sequence.t
	CCSeq.take_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	BatSeq.take_while : ('a -> bool) -> 'a Seq.t -> 'a Seq.t	Base.Sequence.take_while : 'a Base.Sequence.t -> f:('a -> bool) -> 'a Base.Sequence.t
			Base.Sequence.tl : 'a Base.Sequence.t -> 'a Base.Sequence.t option
		BatSeq.tl : 'a Seq.t -> 'a Seq.t	Base.Sequence.tl_eagerly_exn : 'a Base.Sequence.t -> 'a Base.Sequence.t
	CCSeq.to_array : 'a Seq.t -> 'a array		Base.Sequence.to_array : 'a Base.Sequence.t -> 'a array
		BatSeq.to_buffer : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> Buffer.t -> (unit -> 'a BatSeq.node) -> unit	
	CCSeq.to_gen : 'a Seq.t -> 'a CCSeq.gen		
	CCSeq.to_iter : 'a Seq.t -> 'a CCSeq.iter		
	CCSeq.to_list : 'a Seq.t -> 'a list		Base.Sequence.to_list : 'a Base.Sequence.t -> 'a list
	CCSeq.to_rev_list : 'a Seq.t -> 'a list		Base.Sequence.to_list_rev : 'a Base.Sequence.t -> 'a list
			Base.Sequence.to_seq : 'a Base.Sequence.t -> 'a Base.Import.Caml.Seq.t
		BatSeq.to_string : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> 'a Seq.t -> string	
Seq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	CCSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	BatSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a Seq.t	Base.Sequence.unfold : init:'s -> f:('s -> ('a * 's) option) -> 'a Base.Sequence.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
	CCSeq.uniq : 'a CCSeq.equal -> 'a Seq.t -> 'a Seq.t		
	CCSeq.unzip : ('a * 'b) Seq.t -> 'a Seq.t * 'b Seq.t		
	CCSeq.zip : 'a Seq.t -> 'b Seq.t -> ('a * 'b) Seq.t		Base.Sequence.zip : 'a Base.Sequence.t -> 'b Base.Sequence.t -> ('a * '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
Stdlib	Containers	Batteries	Base
Set.Make.add	CCSet.Make.add	BatSet.add : 'a -> 'a set -> 'a set	Base.Set.add : ('a, 'cmp) set -> 'a -> ('a, 'cmp) set
	CCSet.Make.add_iter		
	CCSet.Make.add_list		
Set.Make.add_seq	CCSet.Make.add_seq	BatSet.add_seq : 'a Seq.t -> 'a set -> 'a set	
		BatSet.any : 'a set -> 'a	
			Base.Set.are_disjoint : ('a, 'cmp) set -> ('a, 'cmp) set -> bool
		BatSet.at_rank_exn : int -> 'a set -> 'a	
		BatSet.backwards : 'a set -> 'a BatEnum.t	
			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 |

Stdlib	Containers	Batteries	Base
			`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
Set.Make.cardinal	CCSet.Make.cardinal	BatSet.cardinal : 'a set -> int	
		BatSet.cartesian_product : 'a set -> 'b set -> ('a * 'b) set	
Set.Make.choose	CCSet.Make.choose	BatSet.choose : 'a set -> 'a	Base.Set.choose_exn : ('a, 'b) set -> 'a
Set.Make.choose_opt	CCSet.Make.choose_opt	BatSet.choose_opt : 'a set -> 'a option	Base.Set.choose : ('a, 'b) set -> '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
Set.Make.compare	CCSet.Make.compare	BatSet.compare : 'a set -> 'a set -> int	Base.Set.compare_direct : ('a, 'cmp) set -> ('a, '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
Set.Make.diff	CCSet.Make.diff	BatSet.diff : 'a set -> 'a set -> 'a set	Base.Set.diff : ('a, 'cmp) set -> ('a, 'cmp) set -> ('a, 'cmp) set
Set.Make.disjoint	CCSet.Make.disjoint	BatSet.disjoint : 'a set -> 'a set -> bool	
Set.Make.elements	CCSet.Make.elements	BatSet.elements : 'a set -> 'a list	Base.Set.elements : ('a, 'b) set -> 'a list
Set.Make.empty	CCSet.Make.empty	BatSet.empty : 'a set	Base.Set.empty : ('a, 'cmp) Base.Set.comparator -> ('a, 'cmp) set
		BatSet.enum : 'a set -> 'a BatEnum.t	
Set.Make.equal	CCSet.Make.equal	BatSet.equal : 'a set -> 'a set -> bool	Base.Set.equal : ('a, 'cmp) set -> ('a, 'cmp) set -> bool
			Base.Set.equal_m__t : (module Base.Set.Equal_m) -> ('elt, 'cmp) set -> ('elt, 'cmp) set -> bool
Set.Make.exists	CCSet.Make.exists	BatSet.exists : ('a -> bool) -> 'a set -> bool	Base.Set.exists : ('a, 'b) set -> f:('a -> bool) -> bool
Set.Make.filter	CCSet.Make.filter	BatSet.filter : ('a -> bool) -> 'a set -> 'a set	Base.Set.filter : ('a, 'cmp) set -> f:('a -> bool) -> ('a, 'cmp) set
Set.Make.filter_map	CCSet.Make.filter_map	BatSet.filter_map : ('a -> 'b option) -> 'a set -> 'b set	Base.Set.filter_map : ('b, 'cmp) Base.Set.comparator -> ('a, 'c) set -> f:('a -> 'b option) -> ('b, 'cmp) set
		BatSet.filter_map_endo : ('a -> 'a option) -> 'a set -> 'a set	
Set.Make.find	CCSet.Make.find	BatSet.find : 'a -> 'a set -> 'a	
Set.Make.find_first	CCSet.Make.find_first	BatSet.find_first : ('a -> bool) -> 'a set -> 'a	Base.Set.find_exn : ('a, 'b) set -> f:('a -> bool) -> 'a
Set.Make.find_first_opt	CCSet.Make.find_first_opt	BatSet.find_first_opt : ('a -> bool) -> 'a set -> 'a option	Base.Set.find : ('a, 'b) set -> f:('a -> bool) -> 'a option
Set.Make.find_last	CCSet.Make.find_last	BatSet.find_last : ('a -> bool) -> 'a set -> 'a	
Set.Make.find_last_opt	CCSet.Make.find_last_opt	BatSet.find_last_opt : ('a -> bool) -> 'a set -> 'a option	
			Base.Set.find_map : ('a, 'c) set -> f:('a -> 'b option) -> 'b option
Set.Make.find_opt	CCSet.Make.find_opt	BatSet.find_opt : 'a -> 'a set -> 'a option	
Set.Make.fold	CCSet.Make.fold	BatSet.fold : ('a -> 'b -> 'b) -> 'a set -> 'b -> 'b	Base.Set.fold : ('a, 'b) set -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
			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
			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
Set.Make.for_all	CCSet.Make.for_all	BatSet.for_all : ('a -> bool) -> 'a set -> bool	Base.Set.for_all : ('a, 'b) set -> f:('a -> bool) -> bool
			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

Stdlib	Containers	Batteries	Base
Set.Make.inter	CCSet.Make.inter	BatSet.intersect : 'a set -> 'a set -> 'a set	Base.Set.inter : ('a, 'cmp) set -> ('a, 'cmp) set -> ('a, 'cmp) set
			Base.Set.invariants : ('a, 'b) set -> bool
Set.Make.is_empty	CCSet.Make.is_empty	BatSet.is_empty : 'a set -> bool	Base.Set.is_empty : ('a, 'b) set -> bool
			Base.Set.is_subset : ('a, 'cmp) set -> of_:('a, 'cmp) set -> bool
Set.Make.iter	CCSet.Make.iter	BatSet.iter : ('a -> unit) -> 'a set -> unit	Base.Set.iter : ('a, 'b) set -> f:('a -> unit) -> unit
			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
Set.Make.map	CCSet.Make.map	BatSet.map : ('a -> 'b) -> 'a set -> 'b set	Base.Set.map : ('b, 'cmp) Base.Set.comparator -> ('a, 'c) set -> f:('a -> 'b) -> ('b, 'cmp) set
		BatSet.map_endo : ('a -> 'a) -> 'a set -> 'a set	
Set.Make.max_elt	CCSet.Make.max_elt	BatSet.max_elt : 'a set -> 'a	Base.Set.max_elt_exn : ('a, 'b) set -> 'a
Set.Make.max_elt_opt	CCSet.Make.max_elt_opt	BatSet.max_elt_opt : 'a set -> 'a option	Base.Set.max_elt : ('a, 'b) set -> 'a option
Set.Make.mem	CCSet.Make.mem	BatSet.mem : 'a -> 'a set -> bool	Base.Set.mem : ('a, 'b) set -> 'a -> bool
			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
Set.Make.min_elt	CCSet.Make.min_elt	BatSet.min_elt : 'a set -> 'a	Base.Set.min_elt_exn : ('a, 'b) set -> 'a
Set.Make.min_elt_opt	CCSet.Make.min_elt_opt	BatSet.min_elt_opt : 'a set -> 'a option	Base.Set.min_elt : ('a, 'b) set -> 'a option
			Base.Set.nth : ('a, 'b) set -> int -> 'a option
		BatSet.of_array : 'a array -> 'a set	Base.Set.of_array : ('a, 'cmp) Base.Set.comparator -> 'a array -> ('a, 'cmp) set
		BatSet.of_enum : 'a BatEnum.t -> 'a set	
			Base.Set.of_increasing_iterator_unchecked : ('a, 'cmp) Base.Set.comparator -> len:int -> f:(int -> 'a) -> ('a, 'cmp) set
	CCSet.Make.of_iter		
Set.Make.of_list	CCSet.Make.of_list	BatSet.of_list : 'a list -> 'a set	Base.Set.of_list : ('a, 'cmp) Base.Set.comparator -> 'a list -> ('a, 'cmp) set
Set.Make.of_seq	CCSet.Make.of_seq	BatSet.of_seq : 'a Seq.t -> 'a 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
Set.Make.partition	CCSet.Make.partition	BatSet.partition : ('a -> bool) -> 'a set -> 'a set * 'a set	Base.Set.partition_tf : ('a, 'cmp) set -> f:('a -> bool) -> ('a, 'cmp) set * ('a, 'cmp) set
		BatSet.pop : 'a set -> 'a * 'a set	
		BatSet.pop_max : 'a set -> 'a * 'a set	
		BatSet.pop_min : 'a set -> 'a * 'a set	
	CCSet.Make.pp		
		BatSet.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'c -> unit) -> 'a BatInnerIO.output -> 'c set -> unit	
Set.Make.remove	CCSet.Make.remove	BatSet.remove : 'a -> 'a set -> 'a set	Base.Set.remove : ('a, 'cmp) set -> 'a -> ('a, 'cmp) set
		BatSet.remove_exn : 'a -> 'a set -> 'a 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
Set.Make.singleton	CCSet.Make.singleton	BatSet.singleton : 'a -> 'a set	Base.Set.singleton : ('a, 'cmp) Base.Set.comparator -> 'a -> ('a, 'cmp) set
Set.Make.split	CCSet.Make.split	BatSet.split : 'a -> 'a set -> 'a set * bool * 'a set	

Stdlib	**Containers**	**Batteries**	**Base**
		BatSet.split_le : 'a -> 'a set -> 'a set * 'a set	
		BatSet.split_lt : 'a -> 'a set -> 'a set * 'a set	
		BatSet.split_opt : 'a -> 'a set -> 'a set * 'a option * 'a set	Base.Set.split : ('a, 'cmp) set -> 'a -> ('a, 'cmp) set * 'a option * ('a, 'cmp) set
			Base.Set.stable_dedup_list : ('a, 'b) Base.Set.comparator -> 'a list -> 'a list
Set.Make.subset	CCSet.Make.subset	BatSet.subset : 'a set -> 'a set -> bool	
			Base.Set.sum : (module Base.Container.Summable with type t = 'sum) -> ('a, 'b) set -> f:('a -> 'sum) -> 'sum
		BatSet.sym_diff : 'a set -> 'a set -> 'a set	Base.Set.symmetric_diff : ('a, 'cmp) set -> ('a, 'cmp) set -> ('a, 'a) Base.Either.t Base.Sequence.t
		BatSet.to_array : 'a set -> 'a array	Base.Set.to_array : ('a, 'b) set -> 'a array
	CCSet.Make.to_iter		
	CCSet.Make.to_list	BatSet.to_list : 'a set -> 'a list	Base.Set.to_list : ('a, 'b) set -> 'a list
Set.Make.to_rev_seq	CCSet.Make.to_rev_seq	BatSet.to_rev_seq : 'a set -> 'a Seq.t	
Set.Make.to_seq	CCSet.Make.to_seq	BatSet.to_seq : 'a set -> 'a Seq.t	
Set.Make.to_seq_from	CCSet.Make.to_seq_from	BatSet.to_seq_from : 'a -> 'a set -> 'a Seq.t	
			Base.Set.to_sequence : ?order:[`Decreasing | `Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) set -> 'a Base.Sequence.t
	CCSet.Make.to_string		
Set.Make.union	CCSet.Make.union	BatSet.union : 'a set -> 'a set -> 'a set	Base.Set.union : ('a, 'cmp) set -> ('a, 'cmp) set -> ('a, 'cmp) set
			Base.Set.union_list : ('a, 'cmp) Base.Set.comparator -> ('a, 'cmp) set list -> ('a, 'cmp) set
		BatSet.update : 'a -> 'a -> 'a set -> 'a set	
Stdlib	Containers	Batteries	Base
	CCStringLabels.(<) : string -> string -> bool		Base.String.(<) : string -> string -> bool
	CCStringLabels.(<=) : string -> string -> bool		Base.String.(<=) : string -> string -> bool
	CCStringLabels.(<>) : string -> string -> bool		Base.String.(<>) : string -> string -> bool
	CCStringLabels.(=) : string -> string -> bool		Base.String.(=) : string -> string -> bool
	CCStringLabels.(>) : string -> string -> bool		Base.String.(>) : string -> string -> bool
	CCStringLabels.(>=) : string -> string -> bool		Base.String.(>=) : string -> string -> bool
			Base.String.(^) : string -> string -> string
			Base.String.ascending : string -> string -> int
		BatString.backwards : string -> char BatEnum.t	
			Base.String.between : string -> low:string -> high:string -> bool
StringLabels.blit : src:string -> src_pos:int -> dst:bytes -> dst_pos:int -> len:int -> unit	CCStringLabels.blit : src:string -> src_pos:int -> dst:bytes -> dst_pos:int -> len:int -> unit	BatString.blit : string -> int -> bytes -> int -> int -> unit	
StringLabels.capitalize : string -> string	CCStringLabels.capitalize : string -> string	BatString.capitalize : string -> string	Base.String.capitalize : string -> string
StringLabels.capitalize_ascii : string -> string	CCStringLabels.capitalize_ascii : string -> string	BatString.capitalize_ascii : string -> string	
		BatString.chop : ?l:int -> ?r:int -> string -> string	
	CCStringLabels.chop_prefix : pre:string -> string -> string option		Base.String.chop_prefix : string -> prefix:string -> string option
			Base.String.chop_prefix_exn : string -> prefix:string -> string
			Base.String.chop_prefix_if_exists : string -> prefix:string -> string
	CCStringLabels.chop_suffix : suf:string -> string -> string option		Base.String.chop_suffix : string -> suffix:string -> string option
			Base.String.chop_suffix_exn : string -> suffix:string -> string
			Base.String.chop_suffix_if_exists : string -> suffix:string -> string

Stdlib	Containers	Batteries	Base
			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}
StringLabels.compare : string -> string -> int	CCStringLabels.compare : string -> string -> int	BatString.compare : string -> string -> int	Base.String.compare : string -> string -> int
	CCStringLabels.compare_natural : string -> string -> int		
	CCStringLabels.compare_versions : string -> string -> int		
StringLabels.concat : sep:string -> string list -> string	CCStringLabels.concat : sep:string -> string list -> string	BatString.concat : string -> string list -> string	Base.String.concat : ?sep:string -> string list -> string
			Base.String.concat_array : ?sep:string -> string array -> string
	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		
StringLabels.contains : string -> char -> bool	CCStringLabels.contains : string -> char -> bool	BatString.contains : string -> char -> bool	Base.String.contains : ?pos:int -> ?len:int -> string -> char -> bool
StringLabels.contains_from : string -> int -> char -> bool	CCStringLabels.contains_from : string -> int -> char -> bool	BatString.contains_from : string -> int -> char -> bool	
StringLabels.copy : string -> string	CCStringLabels.copy : string -> string	BatString.copy : string -> string	Base.String.copy : string -> string
			Base.String.count : string -> f:(Base.String.elt -> bool) -> int
		BatString.count_char : string -> char -> int	
		BatString.count_string : string -> string -> int	
StringLabels.create : int -> bytes	CCStringLabels.create : int -> bytes	BatString.create : int -> bytes	
		BatString.cut_on_char : char -> int -> string -> string	
			Base.String.descending : string -> string -> int
	CCStringLabels.drop : int -> string -> string		
			Base.String.drop_prefix : string -> int -> string
			Base.String.drop_suffix : string -> int -> string
	CCStringLabels.drop_while : f:(char -> bool) -> string -> string		
	CCStringLabels.edit_distance : ?cutoff:int -> string -> string -> int	BatString.edit_distance : string -> string -> int	
		BatString.ends_with : string -> string -> bool	
		BatString.enum : string -> char BatEnum.t	
StringLabels.equal : string -> string -> bool	CCStringLabels.equal : string -> string -> bool	BatString.equal : string -> string -> bool	Base.String.equal : string -> string -> bool
	CCStringLabels.equal_caseless : string -> string -> bool		
StringLabels.escaped : string -> string	CCStringLabels.escaped : string -> string	BatString.escaped : string -> string	Base.String.escaped : string -> string
	CCStringLabels.exists : f:(char -> bool) -> string -> bool		Base.String.exists : string -> f:(Base.String.elt -> bool) -> bool
		BatString.exists : string -> string -> bool	
	CCStringLabels.exists2 : f:(char -> char -> bool) -> string -> string -> bool		
		BatString.explode : string -> char list	
StringLabels.fill : bytes -> pos:int -> len:int -> char -> unit	CCStringLabels.fill : bytes -> pos:int -> len:int -> char -> unit	BatString.fill : bytes -> int -> int -> char -> unit	
	CCStringLabels.filter : f:(char -> bool) -> string -> string	BatString.filter : (char -> bool) -> string -> string	Base.String.filter : string -> f:(char -> bool) -> string
	CCStringLabels.filter_map : f:(char -> char option) -> string -> string	BatString.filter_map : (char -> char option) -> string -> string	

Stdlib	Containers	Batteries	Base
	CCStringLabels.find : ?start:int -> sub:string -> string -> int	BatString.find : string -> string -> int	
			Base.String.find : string -> f:(Base.String.elt -> bool) -> Base.String.elt option
	CCStringLabels.find_all : ?start:int -> sub:string -> string -> int CCStringLabels.gen	BatString.find_all : string -> string -> int BatEnum.t	
	CCStringLabels.find_all_l : ?start:int -> sub:string -> string -> int list		
		BatString.find_from : string -> int -> string -> int	
			Base.String.find_map : string -> f:(Base.String.elt -> 'a option) -> 'a option
	CCStringLabels.flat_map : ?sep:string -> f:(char -> string) -> string -> string		Base.String.concat_map : ?sep:string -> string -> f:(char -> string) -> string
	CCStringLabels.fold : f:('a -> char -> 'a) -> init:'a -> string -> 'a	BatString.fold_left : ('a -> char -> 'a) -> 'a -> string -> 'a	Base.String.fold : string -> init:'accum -> f:('accum -> Base.String.elt -> 'accum) -> 'accum
	CCStringLabels.fold2 : f:('a -> char -> char -> 'a) -> init:'a -> string -> string -> 'a		
		BatString.fold_lefti : ('a -> int -> char -> 'a) -> 'a -> string -> 'a	
			Base.String.fold_result : string -> init:'accum -> f:('accum -> Base.String.elt -> ('accum, 'e) Base.Result.t) -> ('accum, 'e) Base.Result.t
		BatString.fold_right : (char -> 'a -> 'a) -> string -> 'a -> 'a	
		BatString.fold_righti : (int -> char -> 'a -> 'a) -> string -> 'a -> 'a	
			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
	CCStringLabels.foldi : f:('a -> int -> char -> 'a) -> 'a -> string -> 'a		Base.String.foldi : string -> init:'a -> f:(int -> 'a -> char -> 'a) -> 'a
	CCStringLabels.for_all : f:(char -> bool) -> string -> bool		Base.String.for_all : string -> f:(Base.String.elt -> bool) -> bool
	CCStringLabels.for_all2 : f:(char -> char -> bool) -> string -> string -> bool		
StringLabels.get : string -> int -> char	CCStringLabels.get : string -> int -> char	BatString.get : string -> int -> char	Base.String.get : string -> int -> char
	CCStringLabels.hash : string -> int		Base.String.hash : string -> int
			Base.String.hash_fold_t : Base.Ppx_hash_lib.Std.Hash.state -> string -> Base.Ppx_hash_lib.Std.Hash.state
		BatString.head : string -> int -> string	
		BatString.icompare : string -> string -> int	
		BatString.implode : char list -> string	
		BatString.in_place_mirror : bytes -> unit	
StringLabels.index : string -> char -> int	CCStringLabels.index : string -> char -> int	BatString.index : string -> char -> int	Base.String.index_exn : string -> char -> int
			Base.String.index : string -> char -> int option
		BatString.index_after_n : char -> int -> string -> int	
StringLabels.index_from : string -> int -> char -> int	CCStringLabels.index_from : string -> int -> char -> int	BatString.index_from : string -> int -> char -> int	Base.String.index_from_exn : string -> int -> char -> int
			Base.String.index_from : string -> int -> char -> int option
StringLabels.index_from_opt : string -> int -> char -> int option	CCStringLabels.index_from_opt : string -> int -> char -> int option	BatString.index_from_opt : string -> int -> char -> int option	
StringLabels.index_opt : string -> char -> int option	CCStringLabels.index_opt : string -> char -> int option	BatString.index_opt : string -> char -> int option	
StringLabels.init : int -> f:(int -> char) -> string	CCStringLabels.init : int -> f:(int -> char) -> string	BatString.init : int -> (int -> char) -> string	Base.String.init : int -> f:(int -> char) -> string
			Base.String.invariant : string Base.Invariant_intf.inv
	CCStringLabels.is_empty : string -> bool	BatString.is_empty : string -> bool	Base.String.is_empty : string -> bool
			Base.String.is_prefix : string -> prefix:string -> bool

Stdlib	Containers	Batteries	Base
	CCStringLabels.is_sub : sub:string -> sub_pos:int -> string -> pos:int -> sub_len:int -> 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
StringLabels.iter : f:(char -> unit) -> string -> unit	CCStringLabels.iter : f:(char -> unit) -> string -> unit	BatString.iter : (char -> unit) -> string -> unit	Base.String.iter : string -> f:(Base.String.elt -> unit) -> unit
	CCStringLabels.iter2 : f:(char -> char -> unit) -> string -> string -> unit		
StringLabels.iteri : f:(int -> char -> unit) -> string -> unit	CCStringLabels.iteri : f:(int -> char -> unit) -> string -> unit	BatString.iteri : (int -> char -> unit) -> string -> unit	
	CCStringLabels.iteri2 : f:(int -> char -> char -> unit) -> string -> string -> unit		
		BatString.join : string -> string list -> string	
		BatString.lchop : ?n:int -> string -> string	
		BatString.left : string -> int -> string	
StringLabels.length : string -> int	CCStringLabels.length : string -> int	BatString.length : string -> int	Base.String.length : string -> int
			Base.String.lfindi : ?pos:int -> string -> f:(int -> char -> bool) -> int option
	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		
StringLabels.lowercase : string -> string	CCStringLabels.lowercase : string -> string	BatString.lowercase : string -> string	Base.String.lowercase : string -> string
StringLabels.lowercase_ascii : string -> string	CCStringLabels.lowercase_ascii : string -> string	BatString.lowercase_ascii : string -> string	
			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
	CCStringLabels.ltrim : string -> string		
StringLabels.make : int -> char -> string	CCStringLabels.make : int -> char -> string	BatString.make : int -> char -> string	Base.String.make : int -> char -> string
StringLabels.map : f:(char -> char) -> string -> string	CCStringLabels.map : f:(char -> char) -> string -> string	BatString.map : (char -> char) -> string -> string	Base.String.map : string -> f:(char -> char) -> string
	CCStringLabels.map2 : f:(char -> char -> char) -> string -> string -> string		
StringLabels.mapi : f:(int -> char -> char) -> string -> string	CCStringLabels.mapi : f:(int -> char -> char) -> string -> string	BatString.mapi : (int -> char -> char) -> string -> string	Base.String.mapi : string -> f:(int -> char -> char) -> 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
	CCStringLabels.mem : ?start:int -> sub:string -> string -> bool		Base.String.mem : string -> Base.String.elt -> bool
			Base.String.min : string -> string -> string
			Base.String.min_elt : string -> compare:(Base.String.elt -> Base.String.elt -> int) -> Base.String.elt option
		BatString.nreplace : str:string -> sub:string -> by:string -> string	
		BatString.nsplit : string -> by:string -> string list	

Stdlib	Containers	Batteries	Base
		BatString.numeric_compare : string -> string -> int	
	CCStringLabels.of_array : char array -> string		
		BatString.of_backwards : char BatEnum.t -> string	
	CCStringLabels.of_char : char -> string	BatString.of_char : char -> string	Base.String.of_char : char -> string
			Base.String.of_char_list : char list -> string
		BatString.of_enum : char BatEnum.t -> string	
		BatString.of_float : float -> string	
	CCStringLabels.of_gen : char CCStringLabels.gen -> string		
		BatString.of_int : int -> string	
	CCStringLabels.of_iter : char CCStringLabels.iter -> string		
	CCStringLabels.of_list : char list -> string	BatString.of_list : char list -> string	
StringLabels.of_seq : char Seq.t -> string	CCStringLabels.of_seq : char Seq.t -> string	BatString.of_seq : char Seq.t -> string	
			Base.String.of_string : string -> string
		BatString.ord : string -> string -> BatOrd.order	
	CCStringLabels.pad : ?side:[`Left | `Right] -> ?c:char -> int -> string -> string		
	CCStringLabels.pp : Format.formatter -> string -> unit		Base.String.pp : Base.Formatter.t -> string -> unit
	CCStringLabels.pp_buf : Buffer.t -> string -> unit		
	CCStringLabels.prefix : pre:string -> string -> bool		
			Base.String.prefix : string -> int -> string
		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	
StringLabels.rcontains_from : string -> int -> char -> bool	CCStringLabels.rcontains_from : string -> int -> char -> bool	BatString.rcontains_from : string -> int -> char -> bool	
	CCStringLabels.rdrop_while : f:(char -> bool) -> string -> string		
	CCStringLabels.repeat : string -> int -> string	BatString.repeat : string -> int -> string	
	CCStringLabels.replace : ?which:[`All | `Left | `Right] -> sub:string -> by:string -> string -> string		
		BatString.replace : str:string -> sub:string -> by:string -> bool * string	
		BatString.replace_chars : (char -> string) -> string -> string	
	CCStringLabels.rev : string -> string	BatString.rev : string -> string	Base.String.rev : string -> string
		BatString.rev_in_place : bytes -> unit	
	CCStringLabels.rfind : sub:string -> string -> int	BatString.rfind : string -> string -> int	
			Base.String.rfindi : ?pos:int -> string -> f:(int -> char -> bool) -> int option
		BatString.rfind_from : string -> int -> string -> int	
		BatString.right : string -> int -> string	
StringLabels.rindex : string -> char -> int	CCStringLabels.rindex : string -> char -> int	BatString.rindex : string -> char -> int	Base.String.rindex_exn : string -> char -> int
			Base.String.rindex : string -> char -> int option
StringLabels.rindex_from : string -> int ->	CCStringLabels.rindex_from : string -> int -> char -> int	BatString.rindex_from : string -> int -> char -> int	Base.String.rindex_from_exn : string -> int -> char -> int

Stdlib	Containers	Batteries	Base
char -> int			
			Base.String.rindex_from : string -> int -> char -> int option
StringLabels.rindex_from_opt : string -> int -> char -> int option	CCStringLabels.rindex_from_opt : string -> int -> char -> int option	BatString.rindex_from_opt : string -> int -> char -> int option	
StringLabels.rindex_opt : string -> char -> int option	CCStringLabels.rindex_opt : string -> char -> int option	BatString.rindex_opt : string -> char -> int option	
		BatString.rsplit : string -> by:string -> string * string	
			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
	CCStringLabels.rtrim : string -> string		
StringLabels.set : bytes -> int -> char -> unit	CCStringLabels.set : string -> int -> char -> string	BatString.set : bytes -> int -> char -> unit	
			Base.String.sexp_of_t : string -> Sexplib0__.Sexp.t
		BatString.slice : ?first:int -> ?last:int -> string -> string	
		BatString.splice : string -> int -> int -> string -> string	
	CCStringLabels.split : by:string -> string -> string list		
		BatString.split : string -> by:string -> string * string	
StringLabels.split_on_char : sep:char -> string -> string list	CCStringLabels.split_on_char : by:char -> string -> string list	BatString.split_on_char : char -> string -> string list	Base.String.split : string -> on:char -> string list
			Base.String.split_lines : string -> string list
			Base.String.split_on_chars : string -> on:char list -> string list
		BatString.split_on_string : by:string -> string -> string list	
		BatString.starts_with : string -> string -> bool	
		BatString.strip : ?chars:string -> string -> string	Base.String.strip : ?drop:(char -> bool) -> string -> string
StringLabels.sub : string -> pos:int -> len:int -> string	CCStringLabels.sub : string -> pos:int -> len:int -> string	BatString.sub : string -> int -> int -> string	Base.String.sub : (string, string) Base.Blit.sub
			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
	CCStringLabels.suffix : suf:string -> string -> bool		
			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
			Base.String.t_of_sexp : Sexplib0__.Sexp.t -> string
			Base.String.t_sexp_grammar : Base.Ppx_sexp_conv_lib.Sexp.Private.Raw_grammar.t...
		BatString.tail : string -> int -> string	
	CCStringLabels.take : int -> string -> string		
	CCStringLabels.take_drop : int -> string -> string * string		
	CCStringLabels.to_array : string -> char array		Base.String.to_array : string -> Base.String.elt array
		BatString.to_float : string -> float	

Stdlib	Containers	Batteries	Base
	CCStringLabels.to_gen : string -> char CCStringLabels.gen		
		BatString.to_int : string -> int	
	CCStringLabels.to_iter : string -> char CCStringLabels.iter		
	CCStringLabels.to_list : string -> char list	BatString.to_list : string -> char list	Base.String.to_list : string -> Base.String.elt list
			Base.String.to_list_rev : string -> char list
StringLabels.to_seq : string -> char Seq.t	CCStringLabels.to_seq : string -> char Seq.t	BatString.to_seq : string -> char Seq.t	
StringLabels.to_seqi : string -> (int * char) Seq.t	CCStringLabels.to_seqi : string -> (int * char) Seq.t	BatString.to_seqi : string -> (int * char) Seq.t	
			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
StringLabels.trim : string -> string	CCStringLabels.trim : string -> string	BatString.trim : string -> string	
StringLabels.uncapitalize : string -> string	CCStringLabels.uncapitalize : string -> string	BatString.uncapitalize : string -> string	Base.String.uncapitalize : string -> string
StringLabels.uncapitalize_ascii : string -> string	CCStringLabels.uncapitalize_ascii : string -> string	BatString.uncapitalize_ascii : string -> string	
	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		
StringLabels.unsafe_blit : src:string -> src_pos:int -> dst:bytes -> dst_pos:int -> len:int -> unit	CCStringLabels.unsafe_blit : src:string -> src_pos:int -> dst:bytes -> dst_pos:int -> len:int -> unit	BatString.unsafe_blit : string -> int -> bytes -> int -> int -> unit	
StringLabels.unsafe_fill : bytes -> pos:int -> len:int -> char -> unit	CCStringLabels.unsafe_fill : bytes -> pos:int -> len:int -> char -> unit	BatString.unsafe_fill : bytes -> int -> int -> char -> unit	
StringLabels.unsafe_get : string -> int -> char	CCStringLabels.unsafe_get : string -> int -> char	BatString.unsafe_get : string -> int -> char	Base.String.unsafe_get : string -> int -> char
StringLabels.unsafe_set : bytes -> int -> char -> unit	CCStringLabels.unsafe_set : bytes -> int -> char -> unit	BatString.unsafe_set : bytes -> int -> char -> unit	
StringLabels.uppercase : string -> string	CCStringLabels.uppercase : string -> string	BatString.uppercase : string -> string	Base.String.uppercase : string -> string
StringLabels.uppercase_ascii : string -> string	CCStringLabels.uppercase_ascii : string -> string	BatString.uppercase_ascii : string -> string	
			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
	CCString.(<) : string -> string -> bool		
	CCString.(<=) : string -> string -> bool		
	CCString.(<>) : string -> string -> bool		
	CCString.(=) : string -> string -> bool		
	CCString.(>) : string -> string -> bool		
	CCString.(>=) : string -> string -> bool		
String.blit : string -> int -> bytes -> int -> int -> unit	CCString.blit : string -> int -> bytes -> int -> int -> unit		
String.capitalize : string -> string	CCString.capitalize : string -> string		
String.capitalize_ascii : string -> string	CCString.capitalize_ascii : string -> string		
	CCString.chop_prefix : pre:string -> string -> string option		

Stdlib	Containers	Batteries	Base
	CCString.chop_suffix : suf:string -> string -> string option		
String.compare : String.t -> String.t -> int	CCString.compare : string -> string -> int		
	CCString.compare_natural : string -> string -> int		
	CCString.compare_versions : string -> string -> int		
String.concat : string -> string list -> string	CCString.concat : string -> string list -> string		
	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		
String.contains : string -> char -> bool	CCString.contains : string -> char -> bool		
String.contains_from : string -> int -> char -> bool	CCString.contains_from : string -> int -> char -> bool		
String.copy : string -> string	CCString.copy : string -> string		
String.create : int -> bytes	CCString.create : int -> bytes		
	CCString.drop : int -> string -> string		
	CCString.drop_while : (char -> bool) -> string -> string		
	CCString.edit_distance : ?cutoff:int -> string -> string -> int		
String.equal : String.t -> String.t -> bool	CCString.equal : string -> string -> bool		
	CCString.equal_caseless : string -> string -> bool		
String.escaped : string -> string	CCString.escaped : string -> string		
	CCString.exists : (char -> bool) -> string -> bool		
	CCString.exists2 : (char -> char -> bool) -> string -> string -> bool		
String.fill : bytes -> int -> int -> char -> unit	CCString.fill : bytes -> int -> int -> char -> unit		
	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.foldi : ('a -> int -> char -> 'a) -> 'a -> string -> 'a		
	CCString.for_all : (char -> bool) -> string -> bool		
	CCString.for_all2 : (char -> char -> bool) -> string -> string -> bool		
String.get : string -> int -> char	CCString.get : string -> int -> char		
	CCString.hash : string -> int		
String.index : string -> char -> int	CCString.index : string -> char -> int		
String.index_from : string -> int -> char -> int	CCString.index_from : string -> int -> char -> int		
String.index_from_opt : string -> int -> char -> int option	CCString.index_from_opt : string -> int -> char -> int option		
String.index_opt : string -> char -> int option	CCString.index_opt : string -> char -> int option		
String.init : int -> (int -> char) -> string	CCString.init : int -> (int -> char) -> string		

Stdlib	Containers	Batteries	Base
	CCString.is_empty : string -> bool		
	CCString.is_sub : sub:string -> int -> string -> int -> sub_len:int -> bool		
String.iter : (char -> unit) -> string -> unit	CCString.iter : (char -> unit) -> string -> unit		
	CCString.iter2 : (char -> char -> unit) -> string -> string -> unit		
String.iteri : (int -> char -> unit) -> string -> unit	CCString.iteri : (int -> char -> unit) -> string -> unit		
	CCString.iteri2 : (int -> char -> char -> unit) -> string -> string -> unit		
String.length : string -> int	CCString.length : string -> int		
	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		
String.lowercase : string -> string	CCString.lowercase : string -> string		
String.lowercase_ascii : string -> string	CCString.lowercase_ascii : string -> string		
	CCString.ltrim : string -> string		
String.make : int -> char -> string	CCString.make : int -> char -> string		
String.map : (char -> char) -> string -> string	CCString.map : (char -> char) -> string -> string		
	CCString.map2 : (char -> char -> char) -> string -> string -> string		
String.mapi : (int -> char -> char) -> string -> string	CCString.mapi : (int -> char -> char) -> 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		
String.of_seq : char Seq.t -> String.t	CCString.of_seq : char Seq.t -> 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		
String.rcontains_from : string -> int -> char -> bool	CCString.rcontains_from : string -> int -> char -> 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		
String.rindex : string -> char -> int	CCString.rindex : string -> char -> int		
String.rindex_from : string -> int -> char -> int	CCString.rindex_from : string -> int -> char -> int		
String.rindex_from_opt : string -> int -> char -	CCString.rindex_from_opt : string -> int -> char -> int option		

Stdlib	Containers	Batteries	Base
> int option			
String.rindex_opt : string -> char -> int option	CCString.rindex_opt : string -> char -> int option		
	CCString.rtrim : string -> string		
String.set : bytes -> int -> char -> unit	CCString.set : string -> int -> char -> string		
	CCString.split : by:string -> string -> string list		
String.split_on_char : char -> string -> string list	CCString.split_on_char : char -> string -> string list		
String.sub : string -> int -> int -> string	CCString.sub : string -> int -> int -> string		
	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		
String.to_seq : String.t -> char Seq.t	CCString.to_seq : string -> char Seq.t		
String.to_seqi : String.t -> (int * char) Seq.t	CCString.to_seqi : string -> (int * char) Seq.t		
String.trim : string -> string	CCString.trim : string -> string		
String.uncapitalize : string -> string	CCString.uncapitalize : string -> string		
String.uncapitalize_ascii : string -> string	CCString.uncapitalize_ascii : string -> string		
	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		
String.unsafe_blit : string -> int -> bytes -> int -> int -> unit	CCString.unsafe_blit : string -> int -> bytes -> int -> int -> unit		
String.unsafe_fill : bytes -> int -> int -> char -> unit	CCString.unsafe_fill : bytes -> int -> int -> char -> unit		
String.unsafe_get : string -> int -> char	CCString.unsafe_get : string -> int -> char		
String.unsafe_set : bytes -> int -> char -> unit	CCString.unsafe_set : bytes -> int -> char -> unit		
String.uppercase : string -> string	CCString.uppercase : string -> string		
String.uppercase_ascii : string -> string	CCString.uppercase_ascii : string -> string		