Containers	Batteries	Base
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]	
CCArray.compare : 'a CCArray.ord -> 'a array CCArray.ord	BatArray.compare : 'a BatOrd.comp -> 'a array BatOrd.comp	
CCArray.equal : 'a CCArray.equal -> 'a array CCArray.equal	BatArray.equal : 'a BatOrd.eq -> 'a array BatOrd.eq	
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	
CCArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	BatArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a	
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	
CCArray.shuffle : 'a array -> unit	BatArray.shuffle : ?state:Random.State.t -> 'a array -> unit	
CCListLabels.(@) : 'a list -> 'a list -> 'a list	BatList.(@) : 'a list -> 'a list -> 'a list	
CCList.cartesian_product : 'a list list -> 'a list list	BatList.n_cartesian_product : 'a list list -> 'a list list	
CCList.count : ('a -> bool) -> 'a list -> int	BatList.count_matching : ('a -> bool) -> 'a list -> int	
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	
CCList.foldi : ('b -> int -> 'a -> 'b) -> 'b -> 'a list -> 'b	BatList.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b list -> 'a	
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.cartesian_product : 'a list list -> 'a list list	BatList.n_cartesian_product : 'a list list -> 'a list list	
CCList.reduce_exn : ('a -> 'a -> 'a) -> 'a list -> 'a	BatList.reduce : ('a -> 'a -> 'a) -> 'a list -> 'a	
CCList.remove_at_idx : int -> 'a list -> 'a list	BatList.remove_at : int -> 'a list -> 'a list	
CCList.take : int -> 'a list -> 'a list	BatList.take : int -> 'a list -> 'a list	
CCList.take_while : ('a -> bool) -> 'a list -> 'a list	BatList.take_while : ('a -> bool) -> 'a list -> 'a list	
CCList.uniq : eq:('a -> 'a -> bool) -> 'a list -> 'a list	BatList.unique : ?eq:('a -> 'a -> bool) -> 'a list -> 'a list	
CCMap.Make.of_seq	BatMap.of_seq : ('key * 'a) BatSeq.t -> ('key, 'a) map	
CCMap.Make.values	BatMap.values : ('a, 'b) map -> 'b BatEnum.t	
CCSeq.(--) : int -> int -> int Seq.t	BatSeq.(--) : int -> int -> int Seq.t	
CCSeq.(--^) : int -> int -> int Seq.t	BatSeq.(--^) : int -> int -> int Seq.t	
CCSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	BatSeq.flatten : 'a Seq.t Seq.t -> 'a Seq.t	
CCSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	BatSeq.iter2 : ('a -> 'b -> unit) -> 'a Seq.t -> 'b Seq.t -> unit	
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.nil : 'a Seq.t	BatSeq.nil : 'a Seq.t	
CCStringLabels.edit_distance : ?cutoff:int -> string -> string -> int	BatString.edit_distance : string -> string -> int	
CCStringLabels.filter_map : f:(char -> char option) -> string -> string	BatString.filter_map : (char -> char option) -> string -> string	
CCStringLabels.find : ?start:int -> sub:string -> string -> int	BatString.find : string -> string -> int	
CCStringLabels.find_all : ?start:int -> sub:string -> string -> int CCStringLabels.gen	BatString.find_all : string -> string -> int BatEnum.t	
CCStringLabels.of_list : char list -> string	BatString.of_list : char list -> string	
CCStringLabels.repeat : string -> int -> string	BatString.repeat : string -> int -> string	
CCStringLabels.rfind : sub:string -> string -> int	BatString.rfind : string -> string -> int	
CCArrayLabels.compare : 'a CCArrayLabels.ord -> 'a array CCArrayLabels.ord		Base.Array.compare : ('a -> 'a -> int) -> 'a array -> 'a array -> int
CCArrayLabels.equal : 'a CCArrayLabels.equal -> 'a array CCArrayLabels.equal		Base.Array.equal : ('a -> 'a -> bool) -> 'a array -> 'a array -> bool

Containers	Batteries	Base
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
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
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
CCArrayLabels.random_choose : 'a array -> 'a CCArrayLabels.random_gen		Base.Array.random_element_exn : ?random_state:Base.Random.State.t -> 'a array -> 'a
CCArrayLabels.reverse_in_place : 'a array -> unit		Base.Array.rev_inplace : 'a array -> unit
CCArrayLabels.shuffle_with : Random.State.t -> 'a array -> unit		Base.Array.permute : ?random_state:Base.Random.State.t -> 'a array -> unit
CCArrayLabels.sorted : f:('a -> 'a -> int) -> 'a array -> 'a array		Base.Array.sorted_copy : 'a array -> compare:('a -> 'a -> int) -> 'a array
CCArrayLabels.swap : 'a array -> int -> int -> unit		Base.Array.swap : 'a array -> int -> int -> unit
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.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.chunks : int -> 'a list -> 'a list list		Base.List.chunks_of : 'a list -> length:int -> 'a list list
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
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_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.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.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.head_opt : 'a list -> 'a option		Base.List.hd : 'a list -> 'a option
CCListLabels.intersperse : x:'a -> 'a list -> 'a list		Base.List.intersperse : 'a list -> sep:'a -> 'a list
CCListLabels.is_empty : 'a list -> bool		Base.List.is_empty : 'a list -> bool
CCListLabels.is_sorted : cmp:('a -> 'a -> int) -> 'a list -> bool		Base.List.is_sorted : 'a list -> compare:('a -> 'a -> int) -> bool
CCListLabels.last_opt : 'a list -> 'a option		Base.List.last : 'a list -> 'a option
CCListLabels.range : int -> int -> int list		Base.List.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> 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
CCListLabels.reduce_exn : f:('a -> 'a -> 'a) -> 'a list -> 'a		Base.List.reduce_exn : 'a list -> f:('a -> 'a -> 'a) -> 'a
CCListLabels.return : 'a -> 'a list		Base.List.return : 'a -> 'a list
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
CCOpt.exists : ('a -> bool) -> 'a option -> bool		Base.Option.exists : 'a option -> f:('a -> bool) -> bool
CCOpt.flat_map : ('a -> 'b option) -> 'a option -> 'b option		Base.Option.find_map : 'a option -> f:('a -> 'b option) -> 'b option
CCOpt.for_all : ('a -> bool) -> 'a option -> bool		Base.Option.for_all : 'a option -> f:('a -> bool) -> bool
CCOpt.map2 : ('a -> 'b -> 'c) -> 'a option -> 'b option -> 'c option		Base.Option.map2 : 'a option -> 'b option -> f:('a -> 'b -> 'c) -> 'c option
CCOpt.return : 'a -> 'a option		Base.Option.return : 'a -> 'a option
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.return : 'a -> ('a, 'err) result		Base.Result.return : 'a -> ('a, 'b) result
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
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

Containers	Batteries	Base
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.head : 'a Seq.t -> 'a option		Base.Sequence.hd : 'a Base.Sequence.t -> 'a option
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
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
CCSeq.range : int -> int -> int Seq.t		Base.Sequence.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int Base.Sequence.t
CCSeq.repeat : ?n:int -> 'a -> 'a Seq.t		Base.Sequence.repeat : 'a -> 'a Base.Sequence.t
CCSeq.singleton : 'a -> 'a Seq.t		Base.Sequence.singleton : 'a -> 'a Base.Sequence.t
CCSeq.to_array : 'a Seq.t -> 'a array		Base.Sequence.to_array : 'a Base.Sequence.t -> 'a array
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
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
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
CCStringLabels.chop_prefix : pre:string -> string -> string option		Base.String.chop_prefix : string -> prefix:string -> string option
CCStringLabels.chop_suffix : suf:string -> string -> string option		Base.String.chop_suffix : string -> suffix:string -> string option
CCStringLabels.exists : f:(char -> bool) -> string -> bool		Base.String.exists : string -> f:(Base.String.elt -> bool) -> bool
CCStringLabels.flat_map : ?sep:string -> f:(char -> string) -> string -> string		Base.String.concat_map : ?sep:string -> string -> f:(char -> string) -> string
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.hash : string -> int		Base.String.hash : string -> int
CCStringLabels.mem : ?start:int -> sub:string -> string -> bool		Base.String.mem : string -> Base.String.elt -> bool
CCStringLabels.pp : Format.formatter -> string -> unit		Base.String.pp : Base.Formatter.t -> string -> unit
CCStringLabels.to_array : string -> char array		Base.String.to_array : string -> Base.String.elt array
	BatArray.cartesian_product : 'a array -> 'b array -> ('a * 'b) array	Base.Array.cartesian_product : 'a array -> 'b array -> ('a * 'b) array
	BatArray.Labels.count_matching : f:('a -> bool) -> 'a array -> int	Base.Array.count : 'a array -> f:('a -> bool) -> int
	BatArray.Labels.find : f:('a -> bool) -> 'a array -> 'a	Base.Array.find_exn : 'a array -> f:('a -> bool) -> 'a
	BatArray.partition : ('a -> bool) -> 'a array -> 'a array * 'a array	Base.Array.partition_tf : 'a array -> f:('a -> bool) -> 'a array * 'a array
	BatArray.reduce : ('a -> 'a -> 'a) -> 'a array -> 'a	Base.Array.reduce_exn : 'a array -> f:('a -> 'a -> 'a) -> 'a
	BatArray.split : ('a * 'b) array -> 'a array * 'b array	Base.Array.unzip : ('a * 'b) array -> 'a array * 'b array
	BatList.cartesian_product : 'a list -> 'b list -> ('a * 'b) list	Base.List.cartesian_product : 'a list -> 'b list -> ('a * 'b) 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
	BatList.last : 'a list -> 'a	Base.List.last_exn : 'a list -> 'a
	BatList.transpose : 'a list list -> 'a list list	Base.List.transpose_exn : 'a list list -> 'a list list
	BatOption.some : 'a -> 'a option	Base.Option.some : 'a -> 'a option
	BatSeq.concat : 'a Seq.t Seq.t -> 'a Seq.t	Base.Sequence.concat : 'a Base.Sequence.t Base.Sequence.t -> 'a Base.Sequence.t

Containers	Batteries	Base
	BatSeq.find : ('a -> bool) -> 'a Seq.t -> 'a option	Base.Sequence.find : 'a Base.Sequence.t -> f:('a -> bool) -> 'a option
	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
	BatSeq.init : int -> (int -> 'a) -> 'a Seq.t	Base.Sequence.init : int -> f:(int -> 'a) -> 'a Base.Sequence.t
	BatSeq.reduce : ('a -> 'a -> 'a) -> 'a Seq.t -> 'a	Base.Sequence.reduce_exn : 'a Base.Sequence.t -> f:('a -> 'a -> 'a) -> 'a
	BatSeq.tl : 'a Seq.t -> 'a Seq.t	Base.Sequence.tl_eagerly_exn : 'a Base.Sequence.t -> 'a Base.Sequence.t
	BatSet.of_array : 'a array -> 'a set	Base.Set.of_array : ('a, 'cmp) Base.Set.comparator -> 'a array -> ('a, 'cmp) 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
	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
	BatString.strip : ?chars:string -> string -> string	Base.String.strip : ?drop:(char -> bool) -> string -> string