Stdlib	Containers	Batteries	Base	F#
ArrayLabels.append : 'a array -> 'a array -> 'a array	CCArrayLabels.append : 'a array -> 'a array -> 'a array		Base.Array.append : 'a t -> 'a t -> 'a t	
			Base.Array.binary_search : ('a t, 'a, 'key) Base__Binary_searchable_intf.binary_search	
			Base.Array.binary_search_segmented : ('a t, '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.Array.blit : ('a t, 'a t) Base__Blit_intf.blit	
			Base.Array.blito : ('a t, 'a t) Base__Blit_intf.blito	
			Base.Array.cartesian_product : 'a t -> 'b t -> ('a * 'b) t	
	CCArrayLabels.bsearch : cmp:('a -> 'a -> int) -> key:'a -> 'a t -> [`All_bigger | `All_lower | `At of int | `Empty | `Just_after of int]			
ArrayLabels.combine : 'a array -> 'b array -> ('a * 'b) array	CCArrayLabels.combine : 'a array -> 'b array -> ('a * 'b) array			
	CCArrayLabels.compare : 'a ord -> 'a t ord		Base.Array.compare : 'a Base__Ppx_compare_lib.compare -> 'a t Base__Ppx_compare_lib.compare	
ArrayLabels.concat : 'a array list -> 'a array	CCArrayLabels.concat : 'a array list -> 'a array		Base.Array.concat : 'a t list -> 'a t	
			Base.Array.concat_map : 'a t -> f:('a -> 'b array) -> 'b array	
			Base.Array.concat_mapi : 'a t -> f:(int -> 'a -> 'b array) -> 'b array	
ArrayLabels.copy : 'a array -> 'a array	CCArrayLabels.copy : 'a array -> 'a array		Base.Array.copy : 'a t -> 'a t	
			Base.Array.copy_matrix : 'a t t -> 'a t t	
			Base.Array.counti : 'a t -> f:(int -> 'a -> bool) -> int	
		BatArray.Labels.count_matching : f:('a -> bool) -> 'a t -> int	Base.Array.count : 'a t -> f:('a -> bool) -> int	
		BatArray.Labels.create : int -> init:'a -> 'a array	Base.Array.create : len:int -> 'a -> 'a t	
			Base.Array.create_float_uninitialized : len:int -> float t	
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 t			
	CCArrayLabels.equal : 'a equal -> 'a t equal		Base.Array.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool	
	CCArrayLabels.except_idx : 'a t -> 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 t -> bool	Base.Array.exists : 'a t -> f:('a -> bool) -> bool	
ArrayLabels.exists2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArrayLabels.exists2 : f:('a -> 'b -> bool) -> 'a t -> 'b t -> bool		Base.Array.exists2_exn : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool	
			Base.Array.existsi : 'a t -> 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 t -> pos:int -> len:int -> 'a -> unit	
	CCArrayLabels.filter : f:('a -> bool) -> 'a t -> 'a t	BatArray.Labels.filter : f:('a -> bool) -> 'a t -> 'a t	Base.Array.filter : 'a t -> f:('a -> bool) -> 'a t	
	CCArrayLabels.filter_map : f:('a -> 'b option) -> 'a t -> 'b t	BatArray.Labels.filter_map : f:('a -> 'b option) -> 'a t -> 'b t	Base.Array.filter_map : 'a t -> f:('a -> 'b option) -> 'b t	
			Base.Array.filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t	
			Base.Array.filter_opt : 'a option t -> 'a t	

Stdlib	Containers	Batteries	Base	F#
			Base.Array.filteri : 'a t -> f:(int -> 'a -> bool) -> 'a t	
		BatArray.Labels.find : f:('a -> bool) -> 'a t -> 'a	Base.Array.find_exn : 'a t -> f:('a -> bool) -> 'a	
			Base.Array.find : 'a t -> f:('a -> bool) -> 'a option	
			Base.Array.find_consecutive_duplicate : 'a t -> equal:('a -> 'a -> bool) -> ('a * 'a) option	
	CCArrayLabels.find_idx : f:('a -> bool) -> 'a t -> (int * 'a) option			
ArrayLabels.find_map : f:('a -> 'b option) -> 'a array -> 'b option	CCArrayLabels.find_map : f:('a -> 'b option) -> 'a t -> 'b option	BatArray.Labels.find_map : f:('a -> 'b option) -> 'a array -> 'b option	Base.Array.find_map : 'a t -> f:('a -> 'b option) -> 'b option	
			Base.Array.find_map_exn : 'a t -> f:('a -> 'b option) -> 'b	
	CCArrayLabels.find_map_i : f:(int -> 'a -> 'b option) -> 'a t -> 'b option		Base.Array.find_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b option	
			Base.Array.find_mapi_exn : 'a t -> f:(int -> 'a -> 'b option) -> 'b	
ArrayLabels.find_opt : f:('a -> bool) -> 'a array -> 'a option	CCArrayLabels.find_opt : f:('a -> bool) -> 'a array -> 'a option	BatArray.Labels.find_opt : f:('a -> bool) -> 'a t -> 'a option		
		BatArray.Labels.findi : f:('a -> bool) -> 'a t -> int		
			Base.Array.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option	
			Base.Array.findi_exn : 'a t -> f:(int -> 'a -> bool) -> int * 'a	
	CCArrayLabels.flat_map : f:('a -> 'b t) -> 'a t -> 'b array			
	CCArrayLabels.fold : f:('a -> 'b -> 'a) -> init:'a -> 'b t -> 'a	BatArray.Labels.fold : f:('a -> 'b -> 'a) -> init:'a -> 'b array -> 'a	Base.Array.fold : 'a t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum	
	CCArrayLabels.fold2 : f:('acc -> 'a -> 'b -> 'acc) -> init:'acc -> 'a t -> 'b t -> 'acc			
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		
ArrayLabels.fold_left_map : f:('a -> 'b -> 'a * 'c) -> init:'a -> 'b array -> 'a * 'c array	CCArrayLabels.fold_left_map : f:('a -> 'b -> 'a * 'c) -> init:'a -> 'b array -> 'a * 'c array			
			Base.Array.fold_mapi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'b * 'c t	
			Base.Array.fold_result : 'a t -> 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		BatArray.Labels.fold_right : f:('b -> 'a -> 'a) -> 'b array -> init:'a -> 'a	Base.Array.fold_right : 'a t -> f:('a -> 'b -> 'b) -> init:'b -> 'b	
			Base.Array.fold_until : 'a t -> 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_map : f:('acc -> 'a -> 'acc * 'b) -> init:'acc -> 'a t -> 'acc * 'b t			
	CCArrayLabels.foldi : f:('a -> int -> 'b -> 'a) -> init:'a -> 'b t -> 'a		Base.Array.foldi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b) -> 'b	
			Base.Array.folding_map : 'a t -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'c t	
			Base.Array.folding_mapi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c t	
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 t -> bool	Base.Array.for_all : 'a t -> f:('a -> bool) -> bool	
ArrayLabels.for_all2 : f:('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArrayLabels.for_all2 : f:('a -> 'b -> bool) -> 'a t -> 'b t -> bool		Base.Array.for_all2_exn : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool	
			Base.Array.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool	
	CCArrayLabels.get_safe : 'a t -> int -> 'a option			

Stdlib	Containers	Batteries	Base	F#
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 t	
			Base.Array.invariant : 'a Base__Invariant_intf.inv -> 'a t Base__Invariant_intf.inv	
			Base.Array.is_empty : 'a t -> bool	
			Base.Array.is_sorted : 'a t -> compare:('a -> 'a -> int) -> bool	
			Base.Array.is_sorted_strictly : 'a t -> 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 t -> f:('a -> unit) -> unit	
ArrayLabels.iter2 : f:('a -> 'b -> unit) -> 'a array -> 'b array -> unit	CCArrayLabels.iter2 : f:('a -> 'b -> unit) -> 'a t -> 'b t -> unit	BatArray.Labels.iter2 : f:('a -> 'b -> unit) -> 'a t -> 'b t -> unit	Base.Array.iter2_exn : 'a t -> 'b t -> f:('a -> 'b -> unit) -> unit	
		BatArray.Labels.iter2i : f:(int -> 'a -> 'b -> unit) -> 'a t -> 'b t -> 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 t -> f:(int -> 'a -> unit) -> unit	
			Base.Array.last : 'a t -> 'a	
	CCArrayLabels.lookup : cmp:'a ord -> key:'a -> 'a t -> int option			
	CCArrayLabels.lookup_exn : cmp:'a ord -> key:'a -> 'a t -> int			
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 t t	
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 t -> 'b t	Base.Array.map : 'a t -> f:('a -> 'b) -> 'b t	
ArrayLabels.map2 : f:('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array	CCArrayLabels.map2 : f:('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t		Base.Array.map2_exn : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t	
	CCArrayLabels.map_inplace : f:('a -> 'a) -> 'a t -> unit		Base.Array.map_inplace : 'a t -> 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 t -> 'b t	Base.Array.mapi : 'a t -> f:(int -> 'a -> 'b) -> 'b t	
			Base.Array.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
			Base.Array.max_length : int	
ArrayLabels.mem : 'a -> set:'a array -> bool	CCArrayLabels.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a t -> bool	BatArray.Labels.modify : f:('a -> 'a) -> 'a array -> unit	Base.Array.mem : 'a t -> 'a -> equal:('a -> 'a -> bool) -> bool	
ArrayLabels.memq : 'a -> set:'a array -> bool	CCArrayLabels.memq : 'a -> set:'a array -> bool			
			Base.Array.merge : 'a t -> 'a t -> compare:('a -> 'a -> int) -> 'a t	
			Base.Array.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
		BatArray.Labels.modifyi : f:(int -> 'a -> 'a) -> 'a array -> unit		
	CCArrayLabels.monoid_product : f:('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t			
ArrayLabels.of_list : 'a list -> 'a array	CCArrayLabels.of_list : 'a list -> 'a array		Base.Array.of_list : 'a list -> 'a t	
			Base.Array.of_list_map : 'a list -> f:('a -> 'b) -> 'b t	
			Base.Array.of_list_mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b t	
			Base.Array.of_list_rev : 'a list -> 'a t	
			Base.Array.of_list_rev_map : 'a list -> f:('a -> 'b) -> 'b t	
			Base.Array.of_list_rev_mapi : 'a list -> f:(int -> 'a -> 'b) -> 'b t	

Stdlib	Containers	Batteries	Base	F#
ArrayLabels.of_seq : 'a Seq.t -> 'a array	CCArrayLabels.of_seq : 'a Seq.t -> 'a array			
			Base.Array.partition_tf : 'a t -> f:('a -> bool) -> 'a t * 'a t	
			Base.Array.partitioni_tf : 'a t -> f:(int -> 'a -> bool) -> 'a t * 'a t	
	CCArrayLabels.shuffle_with : Random.State.t -> 'a t -> unit		Base.Array.permute : ?random_state:Base__.Random.State.t -> ?pos:int -> ?len:int -> 'a t -> unit	
	CCArrayLabels.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> 'a printer -> 'a t printer			
	CCArrayLabels.pp_i : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> (int -> 'a printer) -> 'a t printer			
	CCArrayLabels.random : 'a random_gen -> 'a t random_gen			
	CCArrayLabels.random_choose : 'a t -> 'a random_gen			
			Base.Array.random_element : ?random_state:Base__.Random.State.t -> 'a t -> 'a option	
			Base.Array.random_element_exn : ?random_state:Base__.Random.State.t -> 'a t -> 'a	
	CCArrayLabels.random_len : int -> 'a random_gen -> 'a t random_gen			
	CCArrayLabels.random_non_empty : 'a random_gen -> 'a t random_gen			
			Base.Array.reduce : 'a t -> f:('a -> 'a -> 'a) -> 'a option	
		BatArray.reduce : ('a -> 'a -> 'a) -> 'a array -> 'a	Base.Array.reduce_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a	
	CCArrayLabels.rev : 'a t -> 'a t		Base.Array.rev : 'a t -> 'a t	
	CCArrayLabels.reverse_in_place : 'a t -> unit		Base.Array.rev_inplace : 'a t -> unit	
	CCArrayLabels.scan_left : f:('acc -> 'a -> 'acc) -> init:'acc -> 'a t -> 'acc t			
			Base.Array.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> Sexplib0__.Sexp.t	
	CCArrayLabels.shuffle : 'a t -> 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 t -> compare:('a -> 'a -> int) -> unit	
	CCArrayLabels.sort_generic : (module MONO_ARRAY with type elt = 'elt and type t = 'arr) -> cmp:('elt -> 'elt -> int) -> 'arr -> unit			
	CCArrayLabels.sort_indices : f:('a -> 'a -> int) -> 'a t -> int array			
	CCArrayLabels.sort_ranking : f:('a -> 'a -> int) -> 'a t -> int array			
	CCArrayLabels.sorted : f:('a -> 'a -> int) -> 'a t -> 'a array		Base.Array.sorted_copy : 'a t -> compare:('a -> 'a -> int) -> 'a t	
ArrayLabels.split : ('a * 'b) array -> 'a array * 'b array	CCArrayLabels.split : ('a * 'b) array -> 'a array * 'b 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 t -> 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.Array.sub : ('a t, 'a t) Base__Blit_intf.sub	
			Base.Array.subo : ('a t, 'a t) Base__Blit_intf.subo	
			Base.Array.sum : (module Base__Container_intf.Summable with type t = 'sum) -> 'a t -> f:('a -> 'sum) -> 'sum	

Stdlib	Containers	Batteries	Base	F#
	CCArrayLabels.swap : 'a t -> int -> int -> unit		Base.Array.swap : 'a t -> int -> int -> unit	
			Base.Array.t_of_sexp : (Sexplib0__.Sexp.t -> 'a) -> Sexplib0__.Sexp.t -> 'a t	
			Base.Array.t_sexp_grammar : 'a Sexplib0.Sexp_grammar.t -> 'a t Sexplib0.Sexp_grammar.t	
			Base.Array.to_array : 'a t -> 'a array	
	CCArrayLabels.to_gen : 'a t -> 'a gen			
	CCArrayLabels.to_iter : 'a t -> 'a iter			
ArrayLabels.to_list : 'a array -> 'a list	CCArrayLabels.to_list : 'a array -> 'a list		Base.Array.to_list : 'a t -> 'a list	
ArrayLabels.to_seq : 'a array -> 'a Seq.t	CCArrayLabels.to_seq : 'a t -> 'a Seq.t		Base.Array.to_sequence : 'a t -> 'a Base__.Sequence.t	
			Base.Array.to_sequence_mutable : 'a t -> 'a Base__.Sequence.t	
ArrayLabels.to_seqi : 'a array -> (int * 'a) Seq.t	CCArrayLabels.to_seqi : 'a array -> (int * 'a) Seq.t			
	CCArrayLabels.to_string : ?sep:string -> ('a -> string) -> 'a array -> string			
			Base.Array.transpose : 'a t t -> 'a t t option	
			Base.Array.transpose_exn : 'a t t -> 'a t t	
			Base.Array.unsafe_blit : ('a t, 'a t) Base__Blit_intf.blit	
Array.split : ('a * 'b) array -> 'a array * 'b array	CCArray.split : ('a * 'b) array -> 'a array * 'b array	BatArray.split : ('a * 'b) array -> 'a array * 'b array	Base.Array.unzip : ('a * 'b) t -> 'a t * 'b t	
			Base.Array.zip : 'a t -> 'b t -> ('a * 'b) t option	
			Base.Array.zip_exn : 'a t -> 'b t -> ('a * 'b) t	
	CCArrayLabels.(--) : int -> int -> int t			
	CCArrayLabels.(--^) : int -> int -> int t			
	CCArrayLabels.(>>=) : 'a t -> ('a -> 'b t) -> 'b t			
	CCArrayLabels.(>>|) : 'a t -> ('a -> 'b) -> 'b t			
	CCArrayLabels.(>|=) : 'a t -> ('a -> 'b) -> 'b t			
	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			Array.allPairs : ('a [] -> 'b [] -> ('a * 'b) [])
Array.append : 'a array -> 'a array -> 'a array	CCArray.append : 'a array -> 'a array -> 'a array	BatArray.append : 'a array -> 'a array -> 'a array		Array.append : ('a [] -> 'a [] -> 'a [])
		BatArray.avg : int array -> float		
				Array.average : ???
				Array.averageBy : ???
		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		Array.blit : ('a [] -> int -> 'a [] -> int -> int -> unit)
	CCArray.bsearch : cmp:('a -> 'a -> int) -> 'a -> 'a t -> [`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		

Stdlib	Containers	Batteries	Base	F#
				Array.choose : (('a -> 'b option) -> 'a [] -> 'b [])
				Array.chunkBySize : (int -> 'a [] -> 'a [] [])
				Array.collect : (('a -> 'b []) -> 'a [] -> 'b [])
Array.combine : 'a array -> 'b array -> ('a * 'b) array	CCArray.combine : 'a array -> 'b array -> ('a * 'b) array	BatArray.combine : 'a array -> 'b array -> ('a * 'b) array		
	CCArray.compare : 'a ord -> 'a t ord	BatArray.compare : 'a BatOrd.comp -> 'a array BatOrd.comp		Array.compareWith : (('a -> 'a -> int) -> 'a [] -> 'a [] -> int)
Array.concat : 'a array list -> 'a array	CCArray.concat : 'a array list -> 'a array	BatArray.concat : 'a array list -> 'a array		Array.concat : (seq<'a []> -> 'a [])
				Array.contains : ('a -> 'a [] -> bool) when 'a : equality
Array.copy : 'a array -> 'a array	CCArray.copy : 'a array -> 'a array	BatArray.copy : 'a array -> 'a array		Array.copy : ('a [] -> 'a [])
		BatArray.count_matching : ('a -> bool) -> 'a array -> int		
				Array.countBy : (('a -> 'b) -> 'a [] -> ('b * int) []) when 'b : equality
				Array.create : (int -> 'a -> 'a [])
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		
				Array.distinct : ('a [] -> 'a []) when 'a : equality
				Array.distinctBy : (('a -> 'b) -> 'a [] -> 'a []) when 'b : equality
	CCArray.empty : 'a t			Array.empty : 'a []
		BatArray.enum : 'a array -> 'a BatEnum.t		
	CCArray.equal : 'a equal -> 'a t equal	BatArray.equal : 'a BatOrd.eq -> 'a array BatOrd.eq		
				Array.exactlyOne : ('a [] -> 'a)
	CCArray.except_idx : 'a t -> int -> 'a list			
				Array.except : (seq<'a> -> 'a [] -> 'a []) when 'a : equality
Array.exists : ('a -> bool) -> 'a array -> bool	CCArray.exists : ('a -> bool) -> 'a array -> bool	BatArray.exists : ('a -> bool) -> 'a array -> bool		Array.exists : (('a -> bool) -> 'a [] -> bool)
Array.exists2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArray.exists2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	BatArray.exists2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool		Array.exists2 : (('a -> 'b -> bool) -> 'a [] -> 'b [] -> 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		Array.fill : ('a [] -> int -> int -> 'a -> unit)
	CCArray.filter : ('a -> bool) -> 'a t -> 'a t	BatArray.filter : ('a -> bool) -> 'a array -> 'a array		Array.filter : (('a -> bool) -> 'a [] -> 'a [])
	CCArray.filter_map : ('a -> 'b option) -> 'a t -> 'b t	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		Array.find : (('a -> bool) -> 'a [] -> 'a)

Stdlib	Containers	Batteries	Base	F#
		BatArray.find_all : ('a -> bool) -> 'a array -> 'a array		
	CCArray.find_idx : ('a -> bool) -> 'a t -> (int * 'a) option			
Array.find_map : ('a -> 'b option) -> 'a array -> 'b option	CCArray.find_map : ('a -> 'b option) -> 'a t -> 'b option	BatArray.find_map : ('a -> 'b option) -> 'a array -> 'b option		
	CCArray.find_map_i : (int -> 'a -> 'b option) -> 'a t -> 'b option			
Array.find_opt : ('a -> bool) -> 'a array -> 'a option	CCArray.find_opt : ('a -> bool) -> 'a array -> 'a option	BatArray.find_opt : ('a -> bool) -> 'a array -> 'a option		
				Array.findBack : (('a -> bool) -> 'a [] -> 'a)
		BatArray.findi : ('a -> bool) -> 'a array -> int		Array.findIndex : (('a -> bool) -> 'a [] -> int)
				Array.findIndexBack : (('a -> bool) -> 'a [] -> int)
	CCArray.flat_map : ('a -> 'b t) -> 'a t -> 'b array			
	CCArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a	BatArray.fold : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a		Array.fold : (('a -> 'b -> 'a) -> 'a -> 'b [] -> 'a)
	CCArray.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a t -> 'b t -> 'acc			Array.fold2 : (('a -> 'b -> 'c -> 'a) -> 'a -> 'b [] -> 'c [] -> 'a)
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		
Array.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b array -> 'a * 'c array	CCArray.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b array -> 'a * 'c array	BatArray.fold_left_map : ('a -> 'b -> 'a * 'c) -> 'a -> 'b array -> 'a * 'c array		
		BatArray.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b array -> 'a		
	CCArray.fold_map : ('acc -> 'a -> 'acc * 'b) -> 'acc -> 'a t -> 'acc * 'b t			
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		Array.foldBack : (('a -> 'b -> 'b) -> 'a [] -> 'b -> 'b)
				Array.foldBack2 : (('a -> 'b -> 'c -> 'c) -> 'a [] -> 'b [] -> 'c -> 'c)
		BatArray.fold_righti : (int -> 'b -> 'a -> 'a) -> 'b array -> 'a -> 'a		
	CCArray.fold_while : ('a -> 'b -> 'a * [`Continue | `Stop]) -> 'a -> 'b t -> 'a	BatArray.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a array -> 'acc * int		
	CCArray.foldi : ('a -> int -> 'b -> 'a) -> 'a -> 'b t -> '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.forall : (('a -> bool) -> 'a [] -> bool)
Array.for_all2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool	CCArray.for_all2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	BatArray.for_all2 : ('a -> 'b -> bool) -> 'a array -> 'b array -> bool		Array.forall2 : (('a -> 'b -> bool) -> 'a [] -> 'b [] -> bool)
		BatArray.fsum : float array -> float		
	CCArray.get_safe : 'a t -> int -> 'a option			
				Array.get : ('a [] -> int -> 'a)
				Array.groupBy : (('a -> 'b) -> 'a [] -> ('b * 'a []) []) when 'b : equality
		BatArray.head : 'a array -> int -> 'a array		
				Array.head : ('a [] -> 'a)
				Array.indexed : ('a [] -> (int * 'a) [])
Array.init : int -> (int -> 'a) -> 'a array	CCArray.init : int -> (int -> 'a) -> 'a array	BatArray.init : int -> (int -> 'a) -> 'a array		Array.init : (int -> (int -> 'a) -> 'a [])
		BatArray.insert : 'a array -> 'a -> int -> 'a array		
				Array.insertAt : ???

Stdlib	Containers	Batteries	Base	F#
				Array.insertManyAt : ???
				Array.isEmpty : ('a [] -> bool)
		BatArray.is_sorted_by : ('a -> 'b) -> 'a array -> bool		
				Array.item : (int -> 'a [] -> 'a)
Array.iter : ('a -> unit) -> 'a array -> unit	CCArray.iter : ('a -> unit) -> 'a array -> unit	BatArray.iter : ('a -> unit) -> 'a array -> unit		Array.iter : (('a -> unit) -> 'a [] -> 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		Array.iter2 : (('a -> 'b -> unit) -> 'a [] -> 'b [] -> unit)
		BatArray.iter2i : (int -> 'a -> 'b -> unit) -> 'a array -> 'b array -> unit		Array.iteri2 : ((int -> 'a -> 'b -> unit) -> 'a [] -> 'b [] -> 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		Array.iteri : ((int -> 'a -> unit) -> 'a [] -> unit)
		BatArray.kahan_sum : float array -> float		
				Array.last : ('a [] -> 'a)
		BatArray.left : 'a array -> int -> 'a array		
				Array.length : ('a [] -> int)
	CCArray.lookup : cmp:'a ord -> 'a -> 'a t -> int option			
	CCArray.lookup_exn : cmp:'a ord -> 'a -> 'a t -> int			
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.map : (('a -> 'b) -> 'a [] -> 'b [])
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.map2 : (('a -> 'b -> 'c) -> 'a [] -> 'b [] -> 'c [])
				Array.map3 : (('a -> 'b -> 'c -> 'd) -> 'a [] -> 'b [] -> 'c [] -> 'd [])
				Array.mapFold : (('a -> 'b -> 'c * 'a) -> 'a -> 'b [] -> 'c [] * 'a)
				Array.mapFoldBack : (('a -> 'b -> 'c * 'b) -> 'a [] -> 'b -> 'c [] * 'b)
	CCArray.map_inplace : ('a -> 'a) -> 'a array -> unit			
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		Array.mapi : ((int -> 'a -> 'b) -> 'a [] -> 'b [])
				Array.mapi2 : ((int -> 'a -> 'b -> 'c) -> 'a [] -> 'b [] -> 'c [])
		BatArray.max : 'a array -> 'a		Array.max : ('a [] -> 'a) when 'a : comparison
				Array.maxBy : (('a -> 'b) -> 'a [] -> 'a) when 'b : comparison
Array.mem : 'a -> 'a array -> bool	CCArray.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a t -> 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		Array.min : ('a [] -> 'a) when 'a : comparison
				Array.minBy : (('a -> 'b) -> 'a [] -> 'a) when 'b : comparison
		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 t -> 'b t -> 'c t			

Stdlib	Containers	Batteries	Base	F#
		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.ofList : ('a list -> 'a [])
Array.of_seq : 'a Seq.t -> 'a array	CCArray.of_seq : 'a Seq.t -> 'a array	BatArray.of_seq : 'a Seq.t -> 'a array		Array.ofSeq : (seq<'a> -> 'a [])
		BatArray.ord : 'a BatOrd.ord -> 'a array BatOrd.ord		
				Array.pairwise : ('a [] -> ('a * 'a) [])
		BatArray.partition : ('a -> bool) -> 'a array -> 'a array * 'a array		Array.partition : (('a -> bool) -> 'a [] -> 'a [] * 'a [])
				Array.permute : ((int -> int) -> 'a [] -> 'a [])
				Array.pick : (('a -> 'b option) -> 'a [] -> 'b)
		BatArray.pivot_split : 'a BatOrd.ord -> 'a array -> 'a -> int * int		
	CCArray.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> 'a printer -> 'a t printer			
	CCArray.pp_i : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> (int -> 'a printer) -> 'a t printer			
		BatArray.print : ?first:string -> ?last:string -> ?sep:string -> ('a, 'b) BatIO.printer -> ('a t, 'b) BatIO.printer		
	CCArray.random : 'a random_gen -> 'a t random_gen			
	CCArray.random_choose : 'a t -> 'a random_gen			
	CCArray.random_len : int -> 'a random_gen -> 'a t random_gen			
	CCArray.random_non_empty : 'a random_gen -> 'a t random_gen			
		BatArray.range : 'a array -> int BatEnum.t		
				Array.reduce : (('a -> 'a -> 'a) -> 'a [] -> 'a)
				Array.reduceBack : (('a -> 'a -> 'a) -> 'a [] -> 'a)
		BatArray.remove_at : int -> 'a array -> 'a array		Array.removeAt : ???
				Array.removeManyAt : ???
				Array.replicate : (int -> 'a -> 'a [])
	CCArray.rev : 'a t -> 'a t	BatArray.rev : 'a array -> 'a array		Array.rev : ('a [] -> 'a [])
	CCArray.reverse_in_place : 'a t -> unit	BatArray.rev_in_place : 'a array -> unit		
		BatArray.right : 'a array -> int -> 'a array		
	CCArray.scan_left : ('acc -> 'a -> 'acc) -> 'acc -> 'a t -> 'acc t			Array.scan : (('a -> 'b -> 'a) -> 'a -> 'b [] -> 'a [])
				Array.scanBack : (('a -> 'b -> 'b) -> 'a [] -> 'b -> 'b [])
				Array.set : ('a [] -> int -> 'a -> unit)
	CCArray.shuffle : 'a t -> unit	BatArray.shuffle : ?state:Random.State.t -> 'a array -> unit		
	CCArray.shuffle_with : Random.State.t -> 'a t -> unit			
		BatArray.singleton : 'a -> 'a array		Array.singleton : ('a -> 'a [])
				Array.skip : (int -> 'a [] -> 'a [])
				Array.skipWhile : (('a -> bool) -> 'a [] -> 'a [])

Stdlib	Containers	Batteries	Base	F#
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 MONO_ARRAY with type elt = 'elt and type t = 'arr) -> cmp:('elt -> 'elt -> int) -> 'arr -> unit			
	CCArray.sort_indices : ('a -> 'a -> int) -> 'a t -> int array			
	CCArray.sort_ranking : ('a -> 'a -> int) -> 'a t -> int array			
	CCArray.sorted : ('a -> 'a -> int) -> 'a t -> 'a array			
				Array.sort : ('a [] -> 'a []) when 'a : comparison
				Array.sortBy : (('a -> 'b) -> 'a [] -> 'a []) when 'b : comparison
				Array.sortByDescending : (('a -> 'b) -> 'a [] -> 'a []) when 'b : comparison
				Array.sortDescending : ('a [] -> 'a []) when 'a : comparison
				Array.sortInPlace : ('a [] -> unit) when 'a : comparison
				Array.sortInPlaceBy : (('a -> 'b) -> 'a [] -> unit) when 'b : comparison
				Array.sortInPlaceWith : (('a -> 'a -> int) -> 'a [] -> unit)
				Array.sortWith : (('a -> 'a -> int) -> 'a [] -> 'a [])
Array.split : ('a * 'b) array -> 'a array * 'b array	CCArray.split : ('a * 'b) array -> 'a array * 'b array	BatArray.split : ('a * 'b) array -> 'a array * 'b array		
				Array.splitAt : (int -> 'a [] -> 'a [] * 'a [])
				Array.splitInto : (int -> 'a [] -> 'a [] [])
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		Array.sub : ('a [] -> int -> int -> 'a [])
		BatArray.sum : int array -> int		
				Array.sum : ???
				Array.sumBy : ???
	CCArray.swap : 'a t -> int -> int -> unit			
		BatArray.tail : 'a array -> int -> 'a array		
				Array.tail : ('a [] -> 'a [])
				Array.take : (int -> 'a [] -> 'a [])
				Array.takeWhile : (('a -> bool) -> 'a [] -> 'a [])
	CCArray.to_gen : 'a t -> 'a gen			
	CCArray.to_iter : 'a t -> 'a iter			
Array.to_list : 'a array -> 'a list	CCArray.to_list : 'a array -> 'a list	BatArray.to_list : 'a array -> 'a list		Array.toList : ('a [] -> 'a list)
Array.to_seq : 'a array -> 'a Seq.t	CCArray.to_seq : 'a t -> 'a Seq.t	BatArray.to_seq : 'a array -> 'a Seq.t		Array.toSeq : ('a [] -> seq<'a>)
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			
				Array.transpose : (seq<'a []> -> 'a [] [])
				Array.truncate : (int -> 'a [] -> 'a [])

Stdlib	Containers	Batteries	Base	F#
				Array.tryExactlyOne : ('a [] -> 'a option)
				Array.tryFind : (('a -> bool) -> 'a [] -> 'a option)
				Array.tryFindBack : (('a -> bool) -> 'a [] -> 'a option)
				Array.tryFindIndex : (('a -> bool) -> 'a [] -> int option)
				Array.tryFindIndexBack : (('a -> bool) -> 'a [] -> int option)
				Array.tryHead : ('a [] -> 'a option)
				Array.tryItem : (int -> 'a [] -> 'a option)
				Array.tryLast : ('a [] -> 'a option)
				Array.tryPick : (('a -> 'b option) -> 'a [] -> 'b option)
				Array.unfold : (('a -> ('b * 'a) option) -> 'a -> 'b [])
				Array.unzip : (('a * 'b) [] -> 'a [] * 'b [])
				Array.unzip3 : (('a * 'b * 'c) [] -> 'a [] * 'b [] * 'c [])
				Array.updateAt : ???
				Array.where : (('a -> bool) -> 'a [] -> 'a [])
				Array.windowed : (int -> 'a [] -> 'a [] [])
				Array.zeroCreate : (int -> 'a [])
				Array.zip : ('a [] -> 'b [] -> ('a * 'b) [])
				Array.zip3 : ('a [] -> 'b [] -> 'c [] -> ('a * 'b * 'c) [])
	CCArray.(--) : int -> int -> int t			
	CCArray.(--^) : int -> int -> int t			
	CCArray.(>>=) : 'a t -> ('a -> 'b t) -> 'b t			
	CCArray.(>>|) : 'a t -> ('a -> 'b) -> 'b t			
	CCArray.(>|=) : 'a t -> ('a -> 'b) -> 'b t			
	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.Parallel.choose : (('a -> 'b option) -> 'a [] -> 'b [])
				Array.Parallel.collect : (('a -> 'b []) -> 'a [] -> 'b [])
				Array.Parallel.init : (int -> (int -> 'a) -> 'a [])
				Array.Parallel.iter : (('a -> unit) -> 'a [] -> unit)
				Array.Parallel.iteri : ((int -> 'a -> unit) -> 'a [] -> unit)
				Array.Parallel.map : (('a -> 'b) -> 'a [] -> 'b [])
				Array.Parallel.mapi : ((int -> 'a -> 'b) -> 'a [] -> 'b [])

Stdlib	Containers	Batteries	Base	F#
				Array.Parallel.partition : (('a -> bool) -> 'a [] -> 'a [] * 'a [])
				Array2D.base1 : ('a [,] -> int)
				Array2D.base2 : ('a [,] -> int)
				Array2D.blit : ('a [,] -> int -> int -> 'a [,] -> int -> int -> int -> int -> unit)
				Array2D.copy : ('a [,] -> 'a [,])
				Array2D.create : (int -> int -> 'a -> 'a [,])
				Array2D.createBased : (int -> int -> int -> int -> 'a -> 'a [,])
				Array2D.get : ('a [,] -> int -> int -> 'a)
				Array2D.init : (int -> int -> (int -> int -> 'a) -> 'a [,])
				Array2D.initBased : (int -> int -> int -> int -> (int -> int -> 'a) -> 'a [,])
				Array2D.iter : (('a -> unit) -> 'a [,] -> unit)
				Array2D.iteri : ((int -> int -> 'a -> unit) -> 'a [,] -> unit)
				Array2D.length1 : ('a [,] -> int)
				Array2D.length2 : ('a [,] -> int)
				Array2D.map : (('a -> 'b) -> 'a [,] -> 'b [,])
				Array2D.mapi : ((int -> int -> 'a -> 'b) -> 'a [,] -> 'b [,])
				Array2D.rebase : ('a [,] -> 'a [,])
				Array2D.set : ('a [,] -> int -> int -> 'a -> unit)
				Array2D.zeroCreate : (int -> int -> 'a [,])
				Array2D.zeroCreateBased : (int -> int -> int -> int -> 'a [,])
				Array3D.create : (int -> int -> int -> 'a -> 'a [,,])
				Array3D.get : ('a [,,] -> int -> int -> int -> 'a)
				Array3D.init : (int -> int -> int -> (int -> int -> int -> 'a) -> 'a [,,])
				Array3D.iter : (('a -> unit) -> 'a [,,] -> unit)
				Array3D.iteri : ((int -> int -> int -> 'a -> unit) -> 'a [,,] -> unit)
				Array3D.length1 : ('a [,,] -> int)
				Array3D.length2 : ('a [,,] -> int)
				Array3D.length3 : ('a [,,] -> int)
				Array3D.map : (('a -> 'b) -> 'a [,,] -> 'b [,,])
				Array3D.mapi : ((int -> int -> int -> 'a -> 'b) -> 'a [,,] -> 'b [,,])
				Array3D.set : ('a [,,] -> int -> int -> int -> 'a -> unit)
				Array3D.zeroCreate : (int -> int -> int -> 'a [,,])
				Array4D.create : (int -> int -> int -> int -> 'a -> 'a [,,,])
				Array4D.get : ('a [,,,] -> int -> int -> int -> int ->

Stdlib	Containers	Batteries	Base	F#
				'a)
				Array4D.init : (int -> int -> int -> int -> (int -> int -> int -> int -> 'a) -> 'a [,,,])
				Array4D.length1 : ('a [,,,] -> int)
				Array4D.length2 : ('a [,,,] -> int)
				Array4D.length3 : ('a [,,,] -> int)
				Array4D.length4 : ('a [,,,] -> int)
				Array4D.set : ('a [,,,] -> int -> int -> int -> int -> 'a -> unit)
				Array4D.zeroCreate : (int -> int -> int -> int -> 'a [,,,])
	CCListLabels.add_nodup : eq:('a -> 'a -> bool) -> 'a -> 'a t -> 'a t			
			Base.List.all : 'a t list -> 'a list t	
			Base.List.all_equal : 'a t -> equal:('a -> 'a -> bool) -> 'a option	
	CCListLabels.all_ok : ('a, 'err) result t -> ('a t, 'err) result			
	CCListLabels.all_some : 'a option t -> 'a t option			
			Base.List.all_unit : unit t list -> unit t	
ListLabels.append : 'a list -> 'a list -> 'a list	CCListLabels.append : 'a t -> 'a t -> 'a t		Base.List.append : 'a t -> 'a t -> 'a t	
ListLabels.assoc : 'a -> ('a * 'b) list -> 'b	CCListLabels.assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) t -> 'b			
ListLabels.assoc_opt : 'a -> ('a * 'b) list -> 'b option	CCListLabels.assoc_opt : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) t -> '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) t -> 'b option			
			Base.List.bind : 'a t -> f:('a -> 'b t) -> 'b t	
	CCListLabels.cartesian_product : 'a t t -> 'a t t			
			Base.List.cartesian_product : 'a t -> 'b t -> ('a * 'b) t	
	CCListLabels.chunks : int -> 'a list -> 'a list list		Base.List.chunks_of : 'a t -> length:int -> 'a t t	
ListLabels.combine : 'a list -> 'b list -> ('a * 'b) list	CCListLabels.combine : 'a list -> 'b list -> ('a * 'b) list		Base.List.zip_exn : 'a t -> 'b t -> ('a * 'b) t	
	CCListLabels.combine_gen : 'a list -> 'b list -> ('a * 'b) 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 t -> 'a t -> int		Base.List.compare : 'a Base__Ppx_compare_lib.compare -> 'a t Base__Ppx_compare_lib.compare	
ListLabels.compare_length_with : 'a list -> len:int -> int	CCListLabels.compare_length_with : 'a t -> int -> int			
ListLabels.compare_lengths : 'a list -> 'b list -> int	CCListLabels.compare_lengths : 'a t -> 'b t -> int			
ListLabels.concat : 'a list list -> 'a list	CCListLabels.concat : 'a list list -> 'a list	BatList.Labels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list	Base.List.concat : 'a t t -> 'a t	
ListLabels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list	CCListLabels.concat_map : f:('a -> 'b list) -> 'a list -> 'b list		Base.List.concat_map : 'a t -> f:('a -> 'b t) -> 'b t	

Stdlib	Containers	Batteries	Base	F#
			Base.List.concat_mapi : 'a t -> f:(int -> 'a -> 'b t) -> 'b t	
			Base.List.concat_no_order : 'a t t -> 'a t	
ListLabels.cons : 'a -> 'a list -> 'a list	CCListLabels.cons : 'a -> 'a t -> 'a t		Base.List.cons : 'a -> 'a t -> 'a t	
			Base.List.contains_dup : 'a t -> compare:('a -> 'a -> int) -> bool	
	CCListLabels.cons' : 'a t -> 'a -> 'a t			
	CCListLabels.cons_maybe : 'a option -> 'a t -> 'a t			
	CCListLabels.count : f:('a -> bool) -> 'a list -> int		Base.List.count : 'a t -> f:('a -> bool) -> int	
			Base.List.counti : 'a t -> f:(int -> 'a -> bool) -> int	
		BatList.Labels.count_matching : f:('a -> bool) -> 'a list -> int		
	CCListLabels.count_true_false : f:('a -> bool) -> 'a list -> int * int			
			Base.List.dedup_and_sort : 'a t -> compare:('a -> 'a -> int) -> 'a t	
	CCListLabels.diagonal : 'a t -> ('a * 'a) t			
	CCListLabels.drop : int -> 'a t -> 'a t		Base.List.drop : 'a t -> int -> 'a t	
			Base.List.drop_last : 'a t -> 'a t option	
			Base.List.drop_last_exn : 'a t -> 'a t	
	CCListLabels.drop_while : f:('a -> bool) -> 'a t -> 'a t	BatList.Labels.drop_while : f:('a -> bool) -> 'a list -> 'a list	Base.List.drop_while : 'a t -> f:('a -> bool) -> 'a t	
	CCListLabels.empty : 'a t			
ListLabels.equal : eq:('a -> 'a -> bool) -> 'a list -> 'a list -> bool	CCListLabels.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool		Base.List.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> 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 t -> f:('a -> bool) -> bool	
			Base.List.exists2 : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool Or_unequal_lengths.t	
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 t -> 'b t -> f:('a -> 'b -> bool) -> bool	
			Base.List.existsi : 'a t -> 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 t -> 'a t	BatList.Labels.filter : f:('a -> bool) -> 'a list -> 'a list	Base.List.filter : 'a t -> f:('a -> bool) -> 'a t	
ListLabels.filter_map : f:('a -> 'b option) -> 'a list -> 'b list	CCListLabels.filter_map : f:('a -> 'b option) -> 'a t -> 'b t	BatList.Labels.filter_map : f:('a -> 'b option) -> 'a list -> 'b list	Base.List.filter_map : 'a t -> f:('a -> 'b option) -> 'b t	
			Base.List.filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t	
			Base.List.filter_opt : 'a option t -> 'a t	
ListLabels.filteri : f:(int -> 'a -> bool) -> 'a list -> 'a list	CCListLabels.filteri : f:(int -> 'a -> bool) -> 'a list -> 'a list		Base.List.filteri : 'a t -> f:(int -> 'a -> bool) -> 'a t	
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 t -> f:('a -> bool) -> 'a	
			Base.List.find_a_dup : 'a t -> compare:('a -> 'a -> int) -> 'a option	
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_all_dups : 'a t -> compare:('a -> 'a -> int) -> 'a list	
			Base.List.find_consecutive_duplicate : 'a t -> equal:('a -> 'a -> bool) -> ('a * 'a) option	
		BatList.Labels.find_exn : f:('a -> bool) -> exn -> 'a list -> 'a		

Stdlib	Containers	Batteries	Base	F#
	CCListLabels.find_idx : f:('a -> bool) -> 'a t -> (int * 'a) option			
ListLabels.find_map : f:('a -> 'b option) -> 'a list -> 'b option	CCListLabels.find_map : f:('a -> 'b option) -> 'a t -> 'b option		Base.List.find_map : 'a t -> f:('a -> 'b option) -> 'b option	
			Base.List.find_map_exn : 'a t -> f:('a -> 'b option) -> 'b	
		BatList.Labels.find_map_opt : f:('a -> 'b option) -> 'a list -> 'b option		
	CCListLabels.find_mapi : f:(int -> 'a -> 'b option) -> 'a t -> 'b option		Base.List.find_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b option	
			Base.List.find_mapi_exn : 'a t -> f:(int -> 'a -> 'b option) -> 'b	
ListLabels.find_opt : f:('a -> bool) -> 'a list -> 'a option	CCListLabels.find_opt : f:('a -> bool) -> 'a t -> 'a option		Base.List.find : 'a t -> f:('a -> bool) -> 'a option	
	CCListLabels.find_pred : f:('a -> bool) -> 'a t -> 'a option			
	CCListLabels.find_pred_exn : f:('a -> bool) -> 'a t -> 'a			
			Base.List.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option	
		BatList.Labels.findi : f:(int -> 'a -> bool) -> 'a list -> int * 'a	Base.List.findi_exn : 'a t -> f:(int -> 'a -> bool) -> int * 'a	
	CCListLabels.flat_map : f:('a -> 'b t) -> 'a t -> 'b t			
	CCListLabels.flat_map_i : f:(int -> 'a -> 'b t) -> 'a t -> 'b t			
ListLabels.flatten : 'a list list -> 'a list	CCListLabels.flatten : 'a t t -> 'a t			
		BatList.Labels.fold : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a	Base.List.fold : 'a t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum	
			Base.List.fold2 : 'a t -> 'b t -> init:'c -> f:('c -> 'a -> 'b -> 'c) -> 'c Or_unequal_lengths.t	
			Base.List.fold2_exn : 'a t -> 'b t -> 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 t -> init:'b -> f:('b -> 'a -> 'b) -> 'b	
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 t -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'b * 'c t	
	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 t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'b * 'c t	
	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 t -> 'b t -> 'c			
			Base.List.fold_result : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e)	

Stdlib	Containers	Batteries	Base	F#
			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 t -> init:'b -> 'b	BatList.Labels.fold_right : f:('a -> 'b -> 'b) -> 'a list -> init:'b -> 'b	Base.List.fold_right : 'a t -> 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 t -> 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 t -> 'a			
	CCListLabels.foldi : f:('b -> int -> 'a -> 'b) -> init:'b -> 'a t -> 'b		Base.List.foldi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b) -> 'b	
	CCListLabels.foldi2 : f:('c -> int -> 'a -> 'b -> 'c) -> init:'c -> 'a t -> 'b t -> 'c			
			Base.List.folding_map : 'a t -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'c t	
			Base.List.folding_mapi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c t	
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 t -> f:('a -> bool) -> bool	
			Base.List.for_all2 : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool Or_unequal_lengths.t	
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 t -> 'b t -> f:('a -> 'b -> bool) -> bool	
			Base.List.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool	
	CCListLabels.get_at_idx : int -> 'a t -> 'a option			
	CCListLabels.get_at_idx_exn : int -> 'a t -> 'a			
	CCListLabels.group_by : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a t -> 'a list t		Base.List.group : 'a t -> break:('a -> 'a -> bool) -> 'a t t	
			Base.List.groupi : 'a t -> break:(int -> 'a -> 'a -> bool) -> 'a t t	
	CCListLabels.group_join_by : ?eq:('a -> 'a -> bool) -> ?hash:('a -> int) -> ('b -> 'a) -> 'a t -> 'b t -> ('a * 'b list) t			
	CCListLabels.group_succ : eq:('a -> 'a -> bool) -> 'a list -> 'a list list			
			Base.List.hash_fold_t : 'a Base__Ppx_hash_lib.hash_fold -> 'a t Base__Ppx_hash_lib.hash_fold	
			Base.List.hd : 'a t -> 'a option	
ListLabels.hd : 'a list -> 'a	CCListLabels.hd : 'a list -> 'a		Base.List.hd_exn : 'a t -> 'a	
	CCListLabels.hd_tl : 'a t -> 'a * 'a t			
	CCListLabels.head_opt : 'a t -> 'a option			
			Base.List.ignore_m : 'a t -> unit t	
ListLabels.init : len:int -> f:(int -> 'a) -> 'a list	CCListLabels.init : int -> f:(int -> 'a) -> 'a t	BatList.Labels.init : int -> f:(int -> 'a) -> 'a list	Base.List.init : int -> f:(int -> 'a) -> 'a t	
	CCListLabels.insert_at_idx : int -> 'a -> 'a t -> 'a t			
	CCListLabels.inter : eq:('a -> 'a -> bool) -> 'a t -> 'a t -> 'a t			
	CCListLabels.interleave : 'a list -> 'a list -> 'a list			
	CCListLabels.intersperse : x:'a -> 'a list -> 'a list		Base.List.intersperse : 'a t -> sep:'a -> 'a t	
			Base.List.invariant : 'a Base__Invariant_intf.inv -> 'a t Base__Invariant_intf.inv	
	CCListLabels.is_empty : 'a t -> bool		Base.List.is_empty : 'a t -> bool	
			Base.List.is_prefix : 'a t -> prefix:'a t -> equal:('a -> 'a -> bool) -> bool	
	CCListLabels.is_sorted : cmp:('a -> 'a -> int) -> 'a list ->		Base.List.is_sorted : 'a t -> compare:('a -> 'a -> int) -> bool	

Stdlib	Containers	Batteries	Base	F#
	bool			
			Base.List.is_sorted_strictly : 'a t -> compare:('a -> 'a -> int) -> bool	
			Base.List.is_suffix : 'a t -> suffix:'a t -> 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 t -> f:('a -> unit) -> unit	
			Base.List.iter2 : 'a t -> 'b t -> f:('a -> 'b -> unit) -> unit Or_unequal_lengths.t	
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 t -> 'b t -> f:('a -> 'b -> unit) -> unit	
ListLabels.iteri : f:(int -> 'a -> unit) -> 'a list -> unit	CCListLabels.iteri : f:(int -> 'a -> unit) -> 'a t -> unit	BatList.Labels.iteri : f:(int -> 'a -> unit) -> 'a list -> unit	Base.List.iteri : 'a t -> f:(int -> 'a -> unit) -> unit	
	CCListLabels.iteri2 : f:(int -> 'a -> 'b -> unit) -> 'a t -> 'b t -> unit			
			Base.List.join : 'a t t -> 'a t	
	CCListLabels.join : join_row:('a -> 'b -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCListLabels.join_all_by : ?eq:('key -> 'key -> bool) -> ?hash:('key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:('key -> 'a list -> 'b list -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCListLabels.join_by : ?eq:('key -> 'key -> bool) -> ?hash:('key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:('key -> 'a -> 'b -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCListLabels.keep_ok : ('a, 'b) result t -> 'a t			
	CCListLabels.keep_some : 'a option t -> 'a t			
	CCListLabels.last : int -> 'a t -> 'a t		Base.List.last_exn : 'a t -> 'a	
	CCListLabels.last_opt : 'a t -> 'a option		Base.List.last : 'a t -> 'a option	
ListLabels.length : 'a list -> int	CCListLabels.length : 'a list -> int		Base.List.length : 'a t -> int	
ListLabels.map : f:('a -> 'b) -> 'a list -> 'b list	CCListLabels.map : f:('a -> 'b) -> 'a t -> 'b t	BatList.Labels.map : f:('a -> 'b) -> 'a list -> 'b list	Base.List.map : 'a t -> f:('a -> 'b) -> 'b t	
			Base.List.map2 : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t Or_unequal_lengths.t	
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 t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t	
			Base.List.map3 : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t Or_unequal_lengths.t	
			Base.List.map3_exn : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t	
	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 t -> 'b t	BatList.Labels.mapi : f:(int -> 'a -> 'b) -> 'a list -> 'b list	Base.List.mapi : 'a t -> f:(int -> 'a -> 'b) -> 'b t	
			Base.List.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
ListLabels.mem : 'a -> set:'a list -> bool	CCListLabels.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a t -> bool		Base.List.mem : 'a t -> '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) t -> 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 t -> 'a t -> compare:('a -> 'a -> int) -> 'a t	
	CCListLabels.mguard : bool -> unit t			

Stdlib	Containers	Batteries	Base	F#
			Base.List.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
ListLabels.nth : 'a list -> int -> 'a	CCListLabels.nth : 'a list -> int -> 'a		Base.List.nth_exn : 'a t -> int -> 'a	
ListLabels.nth_opt : 'a list -> int -> 'a option	CCListLabels.nth_opt : 'a t -> int -> 'a option		Base.List.nth : 'a t -> int -> 'a option	
	CCListLabels.of_gen : 'a gen -> 'a t			
	CCListLabels.of_iter : 'a iter -> 'a t			
ListLabels.of_seq : 'a Seq.t -> 'a list	CCListLabels.of_seq : 'a Seq.t -> 'a t		Base.List.of_list : 'a t -> 'a t	
	CCListLabels.of_seq_rev : 'a Seq.t -> 'a t			
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 t -> f:('a -> bool) -> 'a t * 'a t	
			Base.List.partition3_map : 'a t -> f:('a -> [`Fst of 'b | `Snd of 'c | `Trd of 'd]) -> 'b t * 'c t * 'd t	
	CCListLabels.partition_filter_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 : f:('a -> [< `Drop | `Left of 'b | `Right of 'c]) -> '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 t -> f:('a -> ('b, 'c) Base__.Either0.t) -> 'b t * 'c t	
	CCListLabels.partition_map_either : f:('a -> ('b, 'c) CCEither.t) -> 'a list -> 'b list * 'c list			
			Base.List.partition_result : ('ok, 'error) Base__.Result.t t -> 'ok t * 'error t	
			Base.List.permute : ?random_state:Base__.Random.State.t -> 'a t -> 'a t	
	CCListLabels.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> 'a printer -> 'a t printer			
	CCListLabels.product : f:('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t			
	CCListLabels.pure : 'a -> 'a t			
	CCListLabels.random : 'a random_gen -> 'a t random_gen			
	CCListLabels.random_choose : 'a t -> 'a random_gen			
			Base.List.random_element : ?random_state:Base__.Random.State.t -> 'a t -> 'a option	
			Base.List.random_element_exn : ?random_state:Base__.Random.State.t -> 'a t -> 'a	
	CCListLabels.random_len : int -> 'a random_gen -> 'a t random_gen			
	CCListLabels.random_non_empty : 'a random_gen -> 'a t random_gen			
	CCListLabels.random_sequence : 'a random_gen t -> 'a t random_gen			
	CCListLabels.range : int -> int -> int t		Base.List.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int t	
	CCListLabels.range' : int -> int -> int t		Base.List.range' : compare:('a -> 'a -> int) -> stride:('a -> 'a) -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> 'a -> 'a -> 'a t	
	CCListLabels.range_by : step:int -> int -> int -> int t			
	CCListLabels.reduce : f:('a -> 'a -> 'a) -> 'a list -> 'a option		Base.List.reduce : 'a t -> f:('a -> 'a -> 'a) -> 'a option	
			Base.List.reduce_balanced : 'a t -> f:('a -> 'a -> 'a) -> 'a option	
			Base.List.reduce_balanced_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a	
	CCListLabels.reduce_exn : f:('a -> 'a -> 'a) -> 'a list -> 'a		Base.List.reduce_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a	
	CCListLabels.remove : eq:('a -> 'a -> bool) -> key:'a -> 'a t -> 'a t			

Stdlib	Containers	Batteries	Base	F#
ListLabels.remove_assoc : 'a -> ('a * 'b) list -> ('a * 'b) list	CCListLabels.remove_assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) t -> ('a * 'b) t			
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 t -> 'a t			
			Base.List.remove_consecutive_duplicates : ?which_to_keep:[`First | `Last] -> 'a t -> equal:('a -> 'a -> bool) -> 'a t	
		BatList.Labels.remove_if : f:('a -> bool) -> 'a list -> 'a list		
	CCListLabels.remove_one : eq:('a -> 'a -> bool) -> 'a -> 'a t -> 'a t			
	CCListLabels.repeat : int -> 'a t -> 'a t			
	CCListLabels.replicate : int -> 'a -> 'a t			
	CCListLabels.return : 'a -> 'a t		Base.List.return : 'a -> 'a t	
ListLabels.rev : 'a list -> 'a list	CCListLabels.rev : 'a list -> 'a list		Base.List.rev : 'a t -> 'a t	
ListLabels.rev_append : 'a list -> 'a list -> 'a list	CCListLabels.rev_append : 'a list -> 'a list -> 'a list		Base.List.rev_append : 'a t -> 'a t -> 'a t	
			Base.List.rev_filter : 'a t -> f:('a -> bool) -> 'a t	
			Base.List.rev_filter_map : 'a t -> f:('a -> 'b option) -> 'b t	
			Base.List.rev_filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t	
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 t -> f:('a -> 'b) -> 'b t	
			Base.List.rev_map2 : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t Or_unequal_lengths.t	
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 t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t	
			Base.List.rev_map3 : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t Or_unequal_lengths.t	
			Base.List.rev_map3_exn : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t	
			Base.List.rev_map_append : 'a t -> 'b t -> f:('a -> 'b) -> 'b t	
			Base.List.rev_mapi : 'a t -> f:(int -> 'a -> 'b) -> 'b t	
		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 t -> 'a t			
			Base.List.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> 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 t -> compare:('a -> 'a -> int) -> 'a t	
			Base.List.sort_and_group : 'a t -> compare:('a -> 'a -> int) -> 'a t t	
ListLabels.sort_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list	CCListLabels.sort_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list			
	CCListLabels.sorted_diff : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCListLabels.sorted_diff_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCListLabels.sorted_insert : cmp:('a -> 'a -> int) -> ?uniq:bool -> 'a -> 'a list -> 'a list			
	CCListLabels.sorted_mem : cmp:('a -> 'a -> int) -> 'a -> 'a list -> bool			

Stdlib	Containers	Batteries	Base	F#
	CCListLabels.sorted_merge : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCListLabels.sorted_merge_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCListLabels.sorted_remove : cmp:('a -> 'a -> int) -> ?all:bool -> 'a -> 'a list -> 'a list			
ListLabels.split : ('a * 'b) list -> 'a list * 'b list	CCListLabels.split : ('a * 'b) t -> 'a t * 'b t			
			Base.List.split_n : 'a t -> int -> 'a t * 'a t	
			Base.List.split_while : 'a t -> f:('a -> bool) -> 'a t * 'a t	
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 t -> compare:('a -> 'a -> int) -> 'a t	
			Base.List.sub : 'a t -> pos:int -> len:int -> 'a t	
	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 t -> 'a t -> 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 t -> f:('a -> 'sum) -> 'sum	
			Base.List.t_of_sexp : (Sexplib0__.Sexp.t -> 'a) -> Sexplib0__.Sexp.t -> 'a t	
			Base.List.t_sexp_grammar : 'a Sexplib0.Sexp_grammar.t -> 'a t Sexplib0.Sexp_grammar.t	
	CCListLabels.tail_opt : 'a t -> 'a t option			
	CCListLabels.take : int -> 'a t -> 'a t		Base.List.take : 'a t -> int -> 'a t	
	CCListLabels.take_drop : int -> 'a t -> 'a t * 'a t			
	CCListLabels.take_drop_while : f:('a -> bool) -> 'a t -> 'a t * 'a t			
	CCListLabels.take_while : f:('a -> bool) -> 'a t -> 'a t	BatList.Labels.take_while : f:('a -> bool) -> 'a list -> 'a list	Base.List.take_while : 'a t -> f:('a -> bool) -> 'a t	
			Base.List.tl : 'a t -> 'a t option	
ListLabels.tl : 'a list -> 'a list	CCListLabels.tl : 'a list -> 'a list		Base.List.tl_exn : 'a t -> 'a t	
			Base.List.to_array : 'a t -> 'a array	
	CCListLabels.to_gen : 'a t -> 'a gen			
	CCListLabels.to_iter : 'a t -> 'a iter			
			Base.List.to_list : 'a t -> 'a list	
ListLabels.to_seq : 'a list -> 'a Seq.t	CCListLabels.to_seq : 'a t -> 'a Seq.t			
	CCListLabels.to_string : ?start:string -> ?stop:string -> ?sep:string -> ('a -> string) -> 'a t -> string			
			Base.List.transpose : 'a t t -> 'a t t option	
			Base.List.transpose_exn : 'a t t -> 'a t t	
	CCListLabels.union : eq:('a -> 'a -> bool) -> 'a t -> 'a t -> 'a t			
	CCListLabels.uniq : eq:('a -> 'a -> bool) -> 'a t -> 'a t			
	CCListLabels.uniq_succ : eq:('a -> 'a -> bool) -> 'a list -> 'a list			
			Base.List.unordered_append : 'a t -> 'a t -> 'a t	
			Base.List.unzip : ('a * 'b) t -> 'a t * 'b t	

Stdlib	Containers	Batteries	Base	F#
			Base.List.unzip3 : ('a * 'b * 'c) t -> 'a t * 'b t * 'c t	
			Base.List.zip : 'a t -> 'b t -> ('a * 'b) t Or_unequal_lengths.t	
	CCListLabels.(--) : int -> int -> int CCList.t			
	CCListLabels.(--^) : int -> int -> int CCList.t			
	CCListLabels.(<$>) : ('a -> 'b) -> 'a CCList.t -> 'b CCList.t			
	CCListLabels.(<*>) : ('a -> 'b) CCList.t -> 'a CCList.t -> 'b CCList.t			
	CCListLabels.(>>=) : 'a CCList.t -> ('a -> 'b CCList.t) -> 'b CCList.t		Base.List.(>>=) : 'a t -> ('a -> 'b t) -> 'b t	
	CCListLabels.(>|=) : 'a CCList.t -> ('a -> 'b) -> 'b CCList.t		Base.List.(>>|) : 'a t -> ('a -> 'b) -> 'b t	
	CCListLabels.(@) : 'a CCList.t -> 'a CCList.t -> 'a CCList.t			
	CCListLabels.(and&) : 'a list -> 'b list -> ('a * 'b) list			
	CCListLabels.(and*) : 'a CCList.t -> 'b CCList.t -> ('a * 'b) CCList.t			
	CCListLabels.(and+) : 'a CCList.t -> 'b CCList.t -> ('a * 'b) CCList.t			
	CCListLabels.(let*) : 'a CCList.t -> ('a -> 'b CCList.t) -> 'b CCList.t			
	CCListLabels.(let+) : 'a CCList.t -> ('a -> 'b) -> 'b CCList.t			
	CCListLabels.Assoc.get : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> 'b option			
	CCListLabels.Assoc.get_exn : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> 'b			
	CCListLabels.Assoc.keys : ('a, 'b) t -> 'a list			
	CCListLabels.Assoc.map_values : ('b -> 'c) -> ('a, 'b) t -> ('a, 'c) t			
	CCListLabels.Assoc.mem : ?eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> bool			
	CCListLabels.Assoc.remove : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> ('a, 'b) t			
	CCListLabels.Assoc.set : eq:('a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) t -> ('a, 'b) t			
	CCListLabels.Assoc.update : eq:('a -> 'a -> bool) -> f:('b option -> 'b option) -> 'a -> ('a, 'b) t -> ('a, 'b) t			
	CCListLabels.Assoc.values : ('a, 'b) t -> 'b list			
	CCListLabels.Ref.clear : 'a t -> unit			
	CCListLabels.Ref.create : unit -> 'a t			
	CCListLabels.Ref.lift : ('a list -> 'b) -> 'a t -> 'b			
	CCListLabels.Ref.pop : 'a t -> 'a option			
	CCListLabels.Ref.pop_exn : 'a t -> 'a			
	CCListLabels.Ref.push : 'a t -> 'a -> unit			
	CCListLabels.Ref.push_list : 'a t -> 'a list -> unit			
	CCList.add_nodup : eq:('a -> 'a -> bool) -> 'a -> 'a t -> 'a t			
	CCList.all_ok : ('a, 'err) result t -> ('a t, 'err) result			

Stdlib	Containers	Batteries	Base	F#
	CCList.all_some : 'a option t -> 'a t option			
				List.allPairs : ('a list -> 'b list -> ('a * 'b) list)
List.append : 'a list -> 'a list -> 'a list	CCList.append : 'a t -> 'a t -> 'a t	BatList.append : 'a list -> 'a list -> 'a list		List.append : ('a list -> 'a list -> 'a list)
List.assoc : 'a -> ('a * 'b) list -> 'b	CCList.assoc : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) t -> 'b	BatList.assoc : 'a -> ('a * 'b) list -> 'b		
		BatList.assoc_inv : 'b -> ('a * 'b) list -> 'a		
List.assoc_opt : 'a -> ('a * 'b) list -> 'b option	CCList.assoc_opt : eq:('a -> 'a -> bool) -> 'a -> ('a * 'b) t -> 'b option	BatList.assoc_opt : '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) t -> '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		
				List.average : ???
				List.averageBy : ???
		BatList.backwards : 'a list -> 'a BatEnum.t		
	CCList.cartesian_product : 'a t t -> 'a t t	BatList.n_cartesian_product : 'a list list -> 'a list list		
		BatList.cartesian_product : 'a list -> 'b list -> ('a * 'b) list		
				List.choose : (('a -> 'b option) -> 'a list -> 'b list)
	CCList.chunks : int -> 'a list -> 'a list list			List.chunkBySize : (int -> 'a list -> 'a list list)
				List.collect : (('a -> 'b list) -> 'a list -> 'b 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) gen			
	CCList.combine_shortest : 'a list -> 'b list -> ('a * 'b) list			
List.compare : ('a -> 'a -> int) -> 'a list -> 'a list -> int	CCList.compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int	BatList.compare : 'a BatOrd.comp -> 'a list BatOrd.comp		List.compareWith : (('a -> 'a -> int) -> 'a list -> 'a list -> int)
List.compare_length_with : 'a list -> int -> int	CCList.compare_length_with : 'a t -> int -> int	BatList.compare_length_with : 'a list -> int -> int		
List.compare_lengths : 'a list -> 'b list -> int	CCList.compare_lengths : 'a t -> 'b t -> 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 : (seq<'a 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 t -> 'a -> 'a t			
	CCList.cons_maybe : 'a option -> 'a t -> 'a t			
				List.contains : ('a -> 'a list -> bool) when 'a : equality
	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			
				List.countBy : (('a -> 'b) -> 'a list -> ('b * int) list) when 'b : equality

Stdlib	Containers	Batteries	Base	F#
	CCList.diagonal : 'a t -> ('a * 'a) t			
				List.distinct : ('a list -> 'a list) when 'a : equality
				List.distinctBy : (('a -> 'b) -> 'a list -> 'a list) when 'b : equality
	CCList.drop : int -> 'a t -> 'a t	BatList.drop : int -> 'a list -> 'a list		
	CCList.drop_while : ('a -> bool) -> 'a t -> 'a t	BatList.drop_while : ('a -> bool) -> 'a list -> 'a list		
		BatList.dropwhile : ('a -> bool) -> 'a list -> 'a list		
	CCList.empty : 'a t			List.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 t -> 'a t -> bool	BatList.equal : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool		
				List.exactlyOne : ('a list -> 'a)
				List.except : (seq<'a> -> 'a list -> 'a list) when 'a : equality
List.exists : ('a -> bool) -> 'a list -> bool	CCList.exists : ('a -> bool) -> 'a list -> bool	BatList.exists : ('a -> bool) -> 'a list -> bool		List.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.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 t -> 'a t	BatList.filter : ('a -> bool) -> 'a list -> 'a list		List.filter : (('a -> bool) -> 'a list -> 'a list)
List.filter_map : ('a -> 'b option) -> 'a list -> 'b list	CCList.filter_map : ('a -> 'b option) -> 'a t -> 'b t	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 : (('a -> bool) -> 'a list -> 'a)
				List.findBack : (('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 t -> (int * 'a) option			
				List.findIndex : (('a -> bool) -> 'a list -> int)
				List.findIndexBack : (('a -> bool) -> 'a list -> int)
List.find_map : ('a -> 'b option) -> 'a list -> 'b option	CCList.find_map : ('a -> 'b option) -> 'a t -> 'b option	BatList.find_map : ('a -> 'b option) -> 'a list -> 'b		
		BatList.find_map_opt : ('a -> 'b option) -> 'a list -> 'b option		
	CCList.find_mapi : (int -> 'a -> 'b option) -> 'a t -> 'b option			
List.find_opt : ('a -> bool) -> 'a list -> 'a option	CCList.find_opt : ('a -> bool) -> 'a t -> 'a option	BatList.find_opt : ('a -> bool) -> 'a list -> 'a option		
	CCList.find_pred : ('a -> bool) -> 'a t -> 'a option			
	CCList.find_pred_exn : ('a -> bool) -> 'a t -> 'a			

Stdlib	Containers	Batteries	Base	F#
		BatList.findi : (int -> 'a -> bool) -> 'a list -> int * 'a		
		BatList.first : 'a list -> 'a		
	CCList.flat_map : ('a -> 'b t) -> 'a t -> 'b t			
	CCList.flat_map_i : (int -> 'a -> 'b t) -> 'a t -> 'b t			
List.flatten : 'a list list -> 'a list	CCList.flatten : 'a t t -> 'a t	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 : (('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.fold2 : (('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		
		BatList.fold_lefti : ('a -> int -> 'b -> 'a) -> 'a -> 'b list -> 'a		
	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 t -> 'b t -> 'c			
List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b	CCList.fold_right : ('a -> 'b -> 'b) -> 'a t -> 'b -> 'b	BatList.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b		List.foldBack : (('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		List.foldBack2 : (('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 t -> 'a	BatList.fold_while : ('acc -> 'a -> bool) -> ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc * 'a list		
	CCList.foldi : ('b -> int -> 'a -> 'b) -> 'b -> 'a t -> 'b			
	CCList.foldi2 : ('c -> int -> 'a -> 'b -> 'c) -> 'c -> 'a t -> 'b t -> '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.forall : (('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		List.forall2 : (('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 t -> 'a option			

Stdlib	Containers	Batteries	Base	F#
	CCList.get_at_idx_exn : int -> 'a t -> 'a			
		BatList.group : ('a -> 'a -> int) -> 'a list -> 'a list list		
	CCList.group_by : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a t -> 'a list t			
				List.groupBy : (('a -> 'b) -> 'a list -> ('b * 'a list) list) when 'b : equality
	CCList.group_join_by : ?eq:('a -> 'a -> bool) -> ?hash:('a -> int) -> ('b -> 'a) -> 'a t -> 'b t -> ('a * 'b list) t	BatList.group_consecutive : ('a -> 'a -> bool) -> 'a list -> 'a 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		List.head : ('a list -> 'a)
	CCList.hd_tl : 'a t -> 'a * 'a t			
	CCList.head_opt : 'a t -> 'a option			
		BatList.index_of : 'a -> 'a list -> int option		
		BatList.index_ofq : 'a -> 'a list -> int option		
				List.indexed : ('a list -> (int * 'a) list)
List.init : int -> (int -> 'a) -> 'a list	CCList.init : int -> (int -> 'a) -> 'a t	BatList.init : int -> (int -> 'a) -> 'a list		List.init : (int -> (int -> 'a) -> 'a list)
	CCList.insert_at_idx : int -> 'a -> 'a t -> 'a t			
				List.insertAt : ???
				List.insertManyAt : ???
	CCList.inter : eq:('a -> 'a -> bool) -> 'a t -> 'a t -> 'a t			
	CCList.interleave : 'a list -> 'a list -> 'a list	BatList.interleave : ?first:'a -> ?last:'a -> 'a -> 'a list -> 'a list		
	CCList.intersperse : 'a -> 'a list -> 'a list			
	CCList.is_empty : 'a t -> bool	BatList.is_empty : 'a list -> bool		List.isEmpty : ('a list -> bool)
	CCList.is_sorted : cmp:('a -> 'a -> int) -> 'a list -> bool			
				List.item : (int -> 'a list -> 'a)
List.iter : ('a -> unit) -> 'a list -> unit	CCList.iter : ('a -> unit) -> 'a list -> unit	BatList.iter : ('a -> unit) -> 'a list -> unit		List.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		List.iter2 : (('a -> 'b -> unit) -> 'a list -> 'b list -> unit)
	CCList.iteri2 : (int -> 'a -> 'b -> unit) -> 'a t -> 'b t -> unit	BatList.iter2i : (int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit		List.iteri2 : ((int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit)
List.iteri : (int -> 'a -> unit) -> 'a list -> unit	CCList.iteri : (int -> 'a -> unit) -> 'a t -> unit	BatList.iteri : (int -> 'a -> unit) -> 'a list -> unit		List.iteri : ((int -> 'a -> unit) -> 'a list -> unit)
		BatList.kahan_sum : float list -> float		
	CCList.join : join_row:('a -> 'b -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCList.join_all_by : ?eq:('key -> 'key -> bool) -> ?hash:('key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:('key -> 'a list -> 'b list -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCList.join_by : ?eq:('key -> 'key -> bool) -> ?hash:('key -> int) -> ('a -> 'key) -> ('b -> 'key) -> merge:('key -> 'a -> 'b -> 'c option) -> 'a t -> 'b t -> 'c t			
	CCList.keep_ok : ('a, 'b) result t -> 'a t			
	CCList.keep_some : 'a option t -> 'a t			
	CCList.last : int -> 'a t -> 'a t	BatList.last : 'a list -> 'a		List.last : ('a list -> 'a)
	CCList.last_opt : 'a t -> 'a option			

Stdlib	Containers	Batteries	Base	F#
List.length : 'a list -> int	CCList.length : 'a list -> int	BatList.length : 'a list -> int		List.length : ('a list -> int)
		BatList.make : int -> 'a -> 'a list		
List.map : ('a -> 'b) -> 'a list -> 'b list	CCList.map : ('a -> 'b) -> 'a t -> 'b t	BatList.map : ('a -> 'b) -> 'a list -> 'b list		List.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		List.map2 : (('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list)
				List.map3 : (('a -> 'b -> 'c -> 'd) -> 'a list -> 'b list -> 'c list -> 'd list)
		BatList.map2i : (int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list		List.mapi2 : ((int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list)
				List.mapFold : (('a -> 'b -> 'c * 'a) -> 'a -> 'b list -> 'c list * 'a)
				List.mapFoldBack : (('a -> 'b -> 'c * 'b) -> 'a list -> 'b -> 'c list * 'b)
	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 t -> 'b t	BatList.mapi : (int -> 'a -> 'b) -> 'a list -> 'b list		List.mapi : ((int -> 'a -> 'b) -> 'a list -> 'b list)
		BatList.max : ?cmp:('a -> 'a -> int) -> 'a list -> 'a		List.max : ('a list -> 'a) when 'a : comparison
				List.maxBy : (('a -> 'b) -> 'a list -> 'a) when 'b : comparison
List.mem : 'a -> 'a list -> bool	CCList.mem : ?eq:('a -> 'a -> bool) -> 'a -> 'a t -> 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) t -> 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 t			
		BatList.min : ?cmp:('a -> 'a -> int) -> 'a list -> 'a		List.min : ('a list -> 'a) when 'a : comparison
				List.minBy : (('a -> 'b) -> 'a list -> 'a) when 'b : comparison
		BatList.min_max : ?cmp:('a -> 'a -> int) -> 'a list -> 'a * 'a		
		BatList.modify : 'a -> ('b -> 'b) -> ('a * 'b) list -> ('a * 'b) list		
		BatList.modify_at : int -> ('a -> 'a) -> 'a list -> 'a list		
		BatList.modify_def : 'b -> 'a -> ('b -> 'b) -> ('a * 'b) list -> ('a * 'b) list		
		BatList.modify_opt : 'a -> ('b option -> 'b option) -> ('a * 'b) list -> ('a * 'b) list		
		BatList.modify_opt_at : int -> ('a -> 'a option) -> 'a list -> 'a list		
		BatList.nsplit : ('a -> bool) -> 'a list -> 'a list list		
		BatList.ntake : int -> 'a list -> 'a list list		
List.nth : 'a list -> int -> 'a	CCList.nth : 'a list -> int -> 'a	BatList.nth : 'a list -> int -> 'a		List.nth : ('a list -> int -> 'a)
List.nth_opt : 'a list -> int -> 'a option	CCList.nth_opt : 'a t -> int -> 'a option	BatList.nth_opt : 'a list -> int -> 'a option		
				List.ofArray : ('a [] -> 'a list)

Stdlib	Containers	Batteries	Base	F#
		BatList.of_backwards : 'a BatEnum.t -> 'a list		
		BatList.of_enum : 'a BatEnum.t -> 'a list		
	CCList.of_gen : 'a gen -> 'a t			
	CCList.of_iter : 'a iter -> 'a t			
List.of_seq : 'a Seq.t -> 'a list	CCList.of_seq : 'a Seq.t -> 'a t	BatList.of_seq : 'a Seq.t -> 'a list		List.ofSeq : (seq<'a> -> 'a list)
	CCList.of_seq_rev : 'a Seq.t -> 'a t			
		BatList.ord : 'a BatOrd.ord -> 'a list BatOrd.ord		
				List.pairwise : ('a list -> ('a * 'a) list)
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		List.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			
List.partition_map : ('a -> ('b, 'c) Either.t) -> 'a list -> 'b list * 'c list	CCList.partition_map : ('a -> [< `Drop | `Left of 'b | `Right of 'c]) -> 'a list -> 'b list * 'c list	BatList.partition_map : ('a -> ('b, 'c) BatEither.t) -> 'a list -> 'b list * 'c list		
	CCList.partition_map_either : ('a -> ('b, 'c) CCEither.t) -> 'a list -> 'b list * 'c list			
				List.permute : ((int -> int) -> 'a list -> 'a list)
				List.pick : (('a -> 'b option) -> 'a list -> 'b)
	CCList.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> 'a printer -> 'a t 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 t -> 'b t -> 'c t			
	CCList.pure : 'a -> 'a t			
	CCList.random : 'a random_gen -> 'a t random_gen			
	CCList.random_choose : 'a t -> 'a random_gen			
	CCList.random_len : int -> 'a random_gen -> 'a t random_gen			
	CCList.random_non_empty : 'a random_gen -> 'a t random_gen			
	CCList.random_sequence : 'a random_gen t -> 'a t random_gen			
	CCList.range : int -> int -> int t	BatList.range : int -> [< `Downto | `To] -> int -> int list		
	CCList.range' : int -> int -> int t			
	CCList.range_by : step:int -> int -> int -> int t			
	CCList.reduce : ('a -> 'a -> 'a) -> 'a list -> 'a option	BatList.reduce : ('a -> 'a -> 'a) -> 'a list -> 'a		
	CCList.reduce_exn : ('a -> 'a -> 'a) -> 'a list -> 'a			List.reduce : (('a -> 'a -> 'a) -> 'a list -> 'a)
				List.reduceBack : (('a -> 'a -> 'a) -> 'a list -> 'a)
	CCList.remove : eq:('a -> 'a -> bool) -> key:'a -> 'a t -> 'a t	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) t -> ('a * 'b) t	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		

Stdlib	Containers	Batteries	Base	F#
	CCList.remove_at_idx : int -> 'a t -> 'a t	BatList.remove_at : int -> 'a list -> 'a list		List.removeAt : ???
				List.removeManyAt : ???
		BatList.remove_if : ('a -> bool) -> 'a list -> 'a list		
	CCList.remove_one : eq:('a -> 'a -> bool) -> 'a -> 'a t -> 'a t			
	CCList.repeat : int -> 'a t -> 'a t			
	CCList.replicate : int -> 'a -> 'a t			List.replicate : (int -> 'a -> 'a list)
	CCList.return : 'a -> 'a t			
List.rev : 'a list -> 'a list	CCList.rev : 'a list -> 'a list	BatList.rev : 'a list -> 'a list		List.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			List.scan : (('a -> 'b -> 'a) -> 'a -> 'b list -> 'a list)
				List.scanBack : (('a -> 'b -> 'b) -> 'a list -> 'b -> 'b list)
	CCList.set_at_idx : int -> 'a -> 'a t -> 'a t			
		BatList.shuffle : ?state:Random.State.t -> 'a list -> 'a list		
		BatList.singleton : 'a -> 'a list		List.singleton : ('a -> 'a list)
				List.skip : (int -> 'a list -> 'a list)
				List.skipWhile : (('a -> bool) -> 'a list -> '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_diff : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCList.sorted_diff_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCList.sorted_insert : cmp:('a -> 'a -> int) -> ?uniq:bool -> 'a -> 'a list -> 'a list			
	CCList.sorted_mem : cmp:('a -> 'a -> int) -> 'a -> 'a list -> bool			
	CCList.sorted_merge : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCList.sorted_merge_uniq : cmp:('a -> 'a -> int) -> 'a list -> 'a list -> 'a list			
	CCList.sorted_remove : cmp:('a -> 'a -> int) -> ?all:bool -> 'a -> 'a list -> 'a list			
				List.sort : ('a list -> 'a list) when 'a : comparison
				List.sortBy : (('a -> 'b) -> 'a list -> 'a list)

Stdlib	Containers	Batteries	Base	F#
				when 'b : comparison
				List.sortByDescending : (('a -> 'b) -> 'a list -> 'a list) when 'b : comparison
				List.sortDescending : ('a list -> 'a list) when 'a : comparison
				List.sortWith : (('a -> 'a -> int) -> '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) t -> 'a t * 'b t	BatList.split : ('a * 'b) list -> 'a list * 'b list		
		BatList.split_at : int -> 'a list -> 'a list * 'a list		List.splitAt : (int -> 'a list -> 'a list * 'a list)
				List.splitInto : (int -> 'a list -> 'a list 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 t -> 'a t -> bool	BatList.subset : ('a -> 'b -> int) -> 'a list -> 'b list -> bool		
		BatList.sum : int list -> int		
				List.sum : ???
				List.sumBy : ???
	CCList.tail_opt : 'a t -> 'a t option			
				List.tail : ('a list -> 'a list)
	CCList.take : int -> 'a t -> 'a t	BatList.take : int -> 'a list -> 'a list		List.take : (int -> 'a list -> 'a list)
	CCList.take_drop : int -> 'a t -> 'a t * 'a t	BatList.takedrop : int -> 'a list -> 'a list * 'a list		
	CCList.take_drop_while : ('a -> bool) -> 'a t -> 'a t * 'a t			
	CCList.take_while : ('a -> bool) -> 'a t -> 'a t	BatList.take_while : ('a -> bool) -> 'a list -> 'a list		List.takeWhile : (('a -> bool) -> '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		
				List.toArray : ('a list -> 'a [])
	CCList.to_gen : 'a t -> 'a gen			
	CCList.to_iter : 'a t -> 'a iter			
List.to_seq : 'a list -> 'a Seq.t	CCList.to_seq : 'a t -> 'a Seq.t	BatList.to_seq : 'a list -> 'a Seq.t		List.toSeq : ('a list -> seq<'a>)
	CCList.to_string : ?start:string -> ?stop:string -> ?sep:string -> ('a -> string) -> 'a t -> string			
		BatList.transpose : 'a list list -> 'a list list		List.transpose : (seq<'a list> -> 'a list list)
				List.truncate : (int -> 'a list -> 'a list)
				List.tryExactlyOne : ('a list -> 'a option)
				List.tryFind : (('a -> bool) -> 'a list -> 'a option)
				List.tryFindBack : (('a -> bool) -> 'a list -> 'a option)
				List.tryFindIndex : (('a -> bool) -> 'a list -> int option)
				List.tryFindIndexBack : (('a -> bool) -> 'a list

Stdlib	Containers	Batteries	Base	F#
				-> int option)
				List.tryHead : ('a list -> 'a option)
				List.tryItem : (int -> 'a list -> 'a option)
				List.tryLast : ('a list -> 'a option)
				List.tryPick : (('a -> 'b option) -> 'a list -> 'b option)
		BatList.unfold : 'b -> ('b -> ('a * 'b) option) -> 'a list		List.unfold : (('a -> ('b * 'a) option) -> 'a -> 'b list)
		BatList.unfold_exc : (unit -> 'a) -> 'a list * exn		
		BatList.unfold_exn : (unit -> 'a) -> 'a list * exn		
	CCList.union : eq:('a -> 'a -> bool) -> 'a t -> 'a t -> 'a t			
	CCList.uniq : eq:('a -> 'a -> bool) -> 'a t -> 'a t	BatList.unique : ?eq:('a -> 'a -> bool) -> 'a list -> 'a list		
		BatList.unique_cmp : ?cmp:('a -> 'a -> int) -> 'a list -> 'a list		
	CCList.uniq_succ : eq:('a -> 'a -> bool) -> 'a list -> 'a list			
		BatList.unique_hash : ?hash:('a -> int) -> ?eq:('a -> 'a -> bool) -> 'a list -> 'a list		
				List.unzip : (('a * 'b) list -> 'a list * 'b list)
				List.unzip3 : (('a * 'b * 'c) list -> 'a list * 'b list * 'c list)
				List.updateAt : ???
				List.where : (('a -> bool) -> 'a list -> 'a list)
				List.windowed : (int -> 'a list -> 'a list list)
				List.zip : ('a list -> 'b list -> ('a * 'b) list)
				List.zip3 : ('a list -> 'b list -> 'c list -> ('a * 'b * 'c) list)
	CCList.(--) : int -> int -> int t			
	CCList.(--^) : int -> int -> int t			
	CCList.(<$>) : ('a -> 'b) -> 'a t -> 'b t			
	CCList.(<*>) : ('a -> 'b) t -> 'a t -> 'b t			
	CCList.(>>=) : 'a t -> ('a -> 'b t) -> 'b t			
	CCList.(>|=) : 'a t -> ('a -> 'b) -> 'b t			
	CCList.(@) : 'a t -> 'a t -> 'a t	BatList.(@) : 'a list -> 'a list -> 'a list		
	CCList.(and&) : 'a list -> 'b list -> ('a * 'b) list			
	CCList.(and*) : 'a t -> 'b t -> ('a * 'b) t			
	CCList.(and+) : 'a t -> 'b t -> ('a * 'b) t			
	CCList.(let*) : 'a t -> ('a -> 'b t) -> 'b t			
	CCList.(let+) : 'a t -> ('a -> 'b) -> 'b t			
	CCList.Assoc.get : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> 'b option			
	CCList.Assoc.get_exn : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> 'b			
	CCList.Assoc.keys : ('a, 'b) t -> 'a list			
	CCList.Assoc.map_values : ('b -> 'c) -> ('a, 'b) t -> ('a			

Stdlib	Containers	Batteries	Base	F#
	CCList.Assoc.mem : ?eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> bool			
	CCList.Assoc.remove : eq:('a -> 'a -> bool) -> 'a -> ('a, 'b) t -> ('a, 'b) t			
	CCList.Assoc.set : eq:('a -> 'a -> bool) -> 'a -> 'b -> ('a, 'b) t -> ('a, 'b) t			
	CCList.Assoc.update : eq:('a -> 'a -> bool) -> f:('b option -> 'b option) -> 'a -> ('a, 'b) t -> ('a, 'b) t			
	CCList.Assoc.values : ('a, 'b) t -> 'b list			
	CCList.Ref.clear : 'a t -> unit			
	CCList.Ref.create : unit -> 'a t			
	CCList.Ref.lift : ('a list -> 'b) -> 'a t -> 'b			
	CCList.Ref.pop : 'a t -> 'a option			
	CCList.Ref.pop_exn : 'a t -> 'a			
	CCList.Ref.push : 'a t -> 'a -> unit			
	CCList.Ref.push_list : 'a t -> 'a list -> unit			
			Base.Map.add : ('k, 'v, 'cmp) t -> key:'k -> data:'v -> ('k, 'v, 'cmp) t Or_duplicate.t	
Map.add : key -> 'a -> 'a t -> 'a t	CCMap.add : key -> 'a -> 'a t -> 'a t	BatMap.add : 'a -> 'b -> ('a, 'b) t -> ('a, 'b) t	Base.Map.add_exn : ('k, 'v, 'cmp) t -> key:'k -> data:'v -> ('k, 'v, 'cmp) t	Map.add : ('a -> 'b -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
		BatMap.add_carry : 'a -> 'b -> ('a, 'b) t -> ('a, 'b) t * 'b option		
	CCMap.add_iter : 'a t -> (key * 'a) CCMap.iter -> 'a t			
	CCMap.add_iter_with : f:(key -> 'a -> 'a -> 'a) -> 'a t -> (key * 'a) CCMap.iter -> 'a t			
	CCMap.add_list : 'a t -> (key * 'a) list -> 'a t			
	CCMap.add_list_with : f:(key -> 'a -> 'a -> 'a) -> 'a t -> (key * 'a) list -> 'a t			
			Base.Map.add_multi : ('k, 'v list, 'cmp) t -> key:'k -> data:'v -> ('k, 'v list, 'cmp) t	
Map.add_seq : (key * 'a) Seq.t -> 'a t -> 'a t	CCMap.add_seq : 'a t -> (key * 'a) Seq.t -> 'a t	BatMap.add_seq : ('key * 'a) BatSeq.t -> ('key, 'a) t -> ('key, 'a) t		
	CCMap.add_seq_with : f:(key -> 'a -> 'a -> 'a) -> 'a t -> (key * 'a) Seq.t -> 'a t			
		BatMap.any : ('key, 'a) t -> 'key * 'a		
			Base.Map.append : lower_part:('k, 'v, 'cmp) t -> upper_part:('k, 'v, 'cmp) t -> [`Ok of ('k, 'v, 'cmp) t | `Overlapping_key_ranges]	
		BatMap.at_rank_exn : int -> ('key, 'a) t -> 'key * 'a		
		BatMap.backwards : ('a, 'b) t -> ('a * 'b) BatEnum.t		
			Base.Map.binary_search : ('k, 'v, 'cmp) t -> 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) t -> segment_of:(key:'k -> data:'v -> [`Left | `Right]) -> [`First_on_right | `Last_on_left] -> ('k * 'v) option	
			Base.Map.binary_search_subrange : ('k, 'v, 'cmp) t -> compare:(key:'k -> data:'v -> 'bound -> int) -> lower_bound:'bound Base__.Maybe_bound.t -> upper_bound:'bound Base__.Maybe_bound.t -> ('k, 'v, 'cmp) t	
Map.bindings : 'a t -> (key * 'a) list	CCMap.bindings : 'a t -> (key * 'a) list	BatMap.bindings : ('key, 'a) t -> ('key * 'a) list		
Map.cardinal : 'a t -> int	CCMap.cardinal : 'a t -> int	BatMap.cardinal : ('a, 'b) t -> int		

Stdlib	Containers	Batteries	Base	F#
			Base.Map.change : ('k, 'v, 'cmp) t -> 'k -> f:('v option -> 'v option) -> ('k, 'v, 'cmp) t	Map.change : ('a -> ('b option -> 'b option) -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
Map.choose : 'a t -> key * 'a	CCMap.choose : 'a t -> key * 'a	BatMap.choose : ('key, 'a) t -> 'key * 'a		
Map.choose_opt : 'a t -> (key * 'a) option	CCMap.choose_opt : 'a t -> (key * 'a) option	BatMap.choose_opt : ('key, 'a) t -> ('key * 'a) option		
			Base.Map.closest_key : ('k, 'v, 'cmp) t -> [`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) t -> ('k, 'v, 'cmp) t Base__.Or_error.t	
			Base.Map.comparator : ('a, 'b, 'cmp) t -> ('a, 'cmp) Base__.Comparator.t	
			Base.Map.comparator_s : ('a, 'b, 'cmp) t -> ('a, 'cmp) Base__.Comparator.Module.t	
Map.compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int	CCMap.compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int	BatMap.compare : ('b -> 'b -> int) -> ('a, 'b) t -> ('a, 'b) t -> int	Base.Map.compare_direct : ('v -> 'v -> int) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> int	
			Base.Map.compare_m__t : (module Compare_m) -> ('v -> 'v -> int) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> int	
				Map.containsKey : ('a -> Map<'a,'b> -> bool) when 'a : comparison
			Base.Map.count : ('k, 'v, 'a) t -> f:('v -> bool) -> int	Map.count : (Map<'a,'b> -> int) when 'a : comparison
			Base.Map.counti : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> int	
			Base.Map.data : ('a, 'v, 'b) t -> 'v list	
		BatMap.diff : ('a, 'b) t -> ('a, 'b) t -> ('a, 'b) t		
Map.empty : 'a t	CCMap.empty : 'a t	BatMap.empty : ('a, 'b) t	Base.Map.empty : ('a, 'cmp) Base__.Comparator.Module.t -> ('a, 'b, 'cmp) t	Map.empty : Map<'a,'b> when 'a : comparison
		BatMap.enum : ('a, 'b) t -> ('a * 'b) BatEnum.t		
Map.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool	CCMap.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool	BatMap.equal : ('b -> 'b -> bool) -> ('a, 'b) t -> ('a, 'b) t -> bool	Base.Map.equal : ('v -> 'v -> bool) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> bool	
			Base.Map.equal_m__t : (module Equal_m) -> ('v -> 'v -> bool) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> bool	
Map.exists : (key -> 'a -> bool) -> 'a t -> bool	CCMap.exists : (key -> 'a -> bool) -> 'a t -> bool	BatMap.exists : ('a -> 'b -> bool) -> ('a, 'b) t -> bool	Base.Map.exists : ('k, 'v, 'a) t -> f:('v -> bool) -> bool	Map.exists : (('a -> 'b -> bool) -> Map<'a,'b> -> bool) when 'a : comparison
			Base.Map.existsi : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> bool	
		BatMap.extract : 'a -> ('a, 'b) t -> 'b * ('a, 'b) t		
Map.filter : (key -> 'a -> bool) -> 'a t -> 'a t	CCMap.filter : (key -> 'a -> bool) -> 'a t -> 'a t	BatMap.filter : ('key -> 'a -> bool) -> ('key, 'a) t -> ('key, 'a) t	Base.Map.filter : ('k, 'v, 'cmp) t -> f:('v -> bool) -> ('k, 'v, 'cmp) t	Map.filter : (('a -> 'b -> bool) -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
			Base.Map.filter_keys : ('k, 'v, 'cmp) t -> f:('k -> bool) -> ('k, 'v, 'cmp) t	
Map.filter_map : (key -> 'a -> 'b option) -> 'a t -> 'b t	CCMap.filter_map : (key -> 'a -> 'b option) -> 'a t -> 'b t	BatMap.filter_map : ('key -> 'a -> 'b option) -> ('key, 'a) t -> ('key, 'b) t	Base.Map.filter_map : ('k, 'v1, 'cmp) t -> f:('v1 -> 'v2 option) -> ('k, 'v2, 'cmp) t	
			Base.Map.filter_mapi : ('k, 'v1, 'cmp) t -> f:(key:'k -> data:'v1 -> 'v2 option) -> ('k, 'v2, 'cmp) t	
			Base.Map.filteri : ('k, 'v, 'cmp) t -> f:(key:'k -> data:'v -> bool) -> ('k, 'v, 'cmp) t	
		BatMap.filterv : ('a -> bool) -> ('key, 'a) t -> ('key, 'a) t		
Map.find : key -> 'a t -> 'a	CCMap.find : key -> 'a t -> 'a	BatMap.find : 'a -> ('a, 'b) t -> 'b	Base.Map.find_exn : ('k, 'v, 'cmp) t -> 'k -> 'v	Map.find : ('a -> Map<'a,'b> -> 'b) when 'a : comparison
		BatMap.find_default : 'b -> 'a -> ('a, 'b) t -> 'b		
Map.find_first : (key -> bool) -> 'a t -> key * 'a	CCMap.find_first : (key -> bool) -> 'a t -> key * 'a	BatMap.find_first : ('a -> bool) -> ('a, 'b) t -> 'a * 'b		
Map.find_first_opt : (key -> bool) -> 'a t ->	CCMap.find_first_opt : (key -> bool) -> 'a t -> (key * 'a)	BatMap.find_first_opt : ('a -> bool) -> ('a, 'b) t -> ('a *		

Stdlib	Containers	Batteries	Base	F#
(key * 'a) option	option	'b) option		
Map.find_last : (key -> bool) -> 'a t -> key * 'a	CCMap.find_last : (key -> bool) -> 'a t -> key * 'a	BatMap.find_last : ('a -> bool) -> ('a, 'b) t -> 'a * 'b		
Map.find_last_opt : (key -> bool) -> 'a t -> (key * 'a) option	CCMap.find_last_opt : (key -> bool) -> 'a t -> (key * 'a) option	BatMap.find_last_opt : ('a -> bool) -> ('a, 'b) t -> ('a * 'b) option		
Map.find_opt : key -> 'a t -> 'a option	CCMap.find_opt : key -> 'a t -> 'a option	BatMap.find_opt : 'a -> ('a, 'b) t -> 'b option	Base.Map.find : ('k, 'v, 'cmp) t -> 'k -> 'v option	
			Base.Map.find_multi : ('k, 'v list, 'cmp) t -> 'k -> 'v list	
				Map.findKey : (('a -> 'b -> bool) -> Map<'a,'b> -> 'a) when 'a : comparison
Map.fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b	CCMap.fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b	BatMap.fold : ('b -> 'c -> 'c) -> ('a, 'b) t -> 'c -> 'c	Base.Map.fold : ('k, 'v, 'b) t -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a	Map.fold : (('a -> 'b -> 'c -> 'a) -> 'a -> Map<'b,'c> -> 'a) when 'b : comparison
			Base.Map.fold2 : ('k, 'v1, 'cmp) t -> ('k, 'v2, 'cmp) t -> init:'a -> f:(key:'k -> data:('v1, 'v2) Merge_element.t -> 'a -> 'a) -> 'a	
			Base.Map.fold_range_inclusive : ('k, 'v, 'cmp) t -> min:'k -> max:'k -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a	
			Base.Map.fold_right : ('k, 'v, 'b) t -> init:'a -> f:(key:'k -> data:'v -> 'a -> 'a) -> 'a	Map.foldBack : (('a -> 'b -> 'c -> 'c) -> Map<'a,'b> -> 'c -> 'c) when 'a : comparison
			Base.Map.fold_symmetric_diff : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> data_equal:('v -> 'v -> bool) -> init:'a -> f:('a -> ('k, 'v) Symmetric_diff_element.t -> 'a) -> 'a	
			Base.Map.fold_until : ('k, 'v, 'a) t -> init:'acc -> f:(key:'k -> data:'v -> 'acc -> ('acc, 'final) Base__.Container.Continue_or_stop.t) -> finish:('acc -> 'final) -> 'final	
		BatMap.foldi : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) t -> 'c -> 'c		
Map.for_all : (key -> 'a -> bool) -> 'a t -> bool	CCMap.for_all : (key -> 'a -> bool) -> 'a t -> bool	BatMap.for_all : ('a -> 'b -> bool) -> ('a, 'b) t -> bool	Base.Map.for_all : ('k, 'v, 'a) t -> f:('v -> bool) -> bool	Map.forall : (('a -> 'b -> bool) -> Map<'a,'b> -> bool) when 'a : comparison
			Base.Map.for_alli : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> bool	
	CCMap.get : key -> 'a t -> 'a option			
	CCMap.get_or : key -> 'a t -> default:'a -> 'a			
			Base.Map.hash_fold_direct : 'k Base__.Hash.folder -> 'v Base__.Hash.folder -> ('k, 'v, 'cmp) t Base__.Hash.folder	
			Base.Map.hash_fold_m__t : (module Hash_fold_m with type t = 'k) -> (Base__.Hash.state -> 'v -> Base__.Hash.state) -> Base__.Hash.state -> ('k, 'v, 'a) t -> Base__.Hash.state	
		BatMap.intersect : ('b -> 'c -> 'd) -> ('a, 'b) t -> ('a, 'c) t -> ('a, 'd) t		
			Base.Map.invariants : ('a, 'b, 'c) t -> bool	
Map.is_empty : 'a t -> bool	CCMap.is_empty : 'a t -> bool	BatMap.is_empty : ('a, 'b) t -> bool	Base.Map.is_empty : ('a, 'b, 'c) t -> bool	Map.isEmpty : (Map<'a,'b> -> bool) when 'a : comparison
Map.iter : (key -> 'a -> unit) -> 'a t -> unit	CCMap.iter : (key -> 'a -> unit) -> 'a t -> unit	BatMap.iter : ('a -> 'b -> unit) -> ('a, 'b) t -> unit	Base.Map.iter : ('a, 'v, 'b) t -> f:('v -> unit) -> unit	Map.iter : (('a -> 'b -> unit) -> Map<'a,'b> -> unit) when 'a : comparison
			Base.Map.iter2 : ('k, 'v1, 'cmp) t -> ('k, 'v2, 'cmp) t -> f:(key:'k -> data:('v1, 'v2) Merge_element.t -> unit) -> unit	
			Base.Map.iter_keys : ('k, 'a, 'b) t -> f:('k -> unit) -> unit	
			Base.Map.iteri : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> unit) -> unit	
			Base.Map.iteri_until : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> Continue_or_stop.t) -> Finished_or_unfinished.t	
	CCMap.keys : 'a t -> key CCMap.iter	BatMap.keys : ('a, 'b) t -> 'a BatEnum.t	Base.Map.keys : ('k, 'a, 'b) t -> 'k list	Map.keys : ???
			Base.Map.length : ('a, 'b, 'c) t -> int	
			Base.Map.m__t_of_sexp : (module M_of_sexp with type comparator_witness = 'cmp and type t = 'k) -> (Base__.Sexp.t -> 'v) -> Base__.Sexp.t -> ('k, 'v, 'cmp) t	

Stdlib	Containers	Batteries	Base	F#
			Base.Map.m__t_sexp_grammar : (module M_sexp_grammar with type t = 'k) -> 'v Sexplib0.Sexp_grammar.t -> ('k, 'v, 'cmp) t Sexplib0.Sexp_grammar.t	
Map.map : ('a -> 'b) -> 'a t -> 'b t	CCMap.map : ('a -> 'b) -> 'a t -> 'b t	BatMap.map : ('b -> 'c) -> ('a, 'b) t -> ('a, 'c) t	Base.Map.map : ('k, 'v1, 'cmp) t -> f:('v1 -> 'v2) -> ('k, 'v2, 'cmp) t	Map.map : (('a -> 'b -> 'c) -> Map<'a,'b> -> Map<'a,'c>) when 'a : comparison
			Base.Map.map_keys : ('k2, 'cmp2) Base__.Comparator.Module.t -> ('k1, 'v, 'cmp1) t -> f:('k1 -> 'k2) -> [`Duplicate_key of 'k2 | `Ok of ('k2, 'v, 'cmp2) t]	
			Base.Map.map_keys_exn : ('k2, 'cmp2) Base__.Comparator.Module.t -> ('k1, 'v, 'cmp1) t -> f:('k1 -> 'k2) -> ('k2, 'v, 'cmp2) t	
Map.mapi : (key -> 'a -> 'b) -> 'a t -> 'b t	CCMap.mapi : (key -> 'a -> 'b) -> 'a t -> 'b t	BatMap.mapi : ('a -> 'b -> 'c) -> ('a, 'b) t -> ('a, 'c) t	Base.Map.mapi : ('k, 'v1, 'cmp) t -> f:(key:'k -> data:'v1 -> 'v2) -> ('k, 'v2, 'cmp) t	
Map.max_binding : 'a t -> key * 'a	CCMap.max_binding : 'a t -> key * 'a	BatMap.max_binding : ('key, 'a) t -> 'key * 'a	Base.Map.max_elt_exn : ('k, 'v, 'a) t -> 'k * 'v	Map.maxKeyValue : ???
Map.max_binding_opt : 'a t -> (key * 'a) option	CCMap.max_binding_opt : 'a t -> (key * 'a) option	BatMap.max_binding_opt : ('key, 'a) t -> ('key * 'a) option	Base.Map.max_elt : ('k, 'v, 'a) t -> ('k * 'v) option	
Map.mem : key -> 'a t -> bool	CCMap.mem : key -> 'a t -> bool	BatMap.mem : 'a -> ('a, 'b) t -> bool	Base.Map.mem : ('k, 'a, 'cmp) t -> 'k -> bool	
Map.merge : (key -> 'a option -> 'b option -> 'c option) -> 'a t -> 'b t -> 'c t	CCMap.merge : (key -> 'a option -> 'b option -> 'c option) -> 'a t -> 'b t -> 'c t	BatMap.merge : ('key -> 'a option -> 'b option -> 'c option) -> ('key, 'a) t -> ('key, 'b) t -> ('key, 'c) t	Base.Map.merge : ('k, 'v1, 'cmp) t -> ('k, 'v2, 'cmp) t -> f:(key:'k -> ('v1, 'v2) Merge_element.t -> 'v3 option) -> ('k, 'v3, 'cmp) t	
			Base.Map.merge_skewed : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> combine:(key:'k -> 'v -> 'v -> 'v) -> ('k, 'v, 'cmp) t	
	CCMap.merge_safe : f:(key -> [`Both of 'a * 'b | `Left of 'a | `Right of 'b] -> 'c option) -> 'a t -> 'b t -> 'c t			
Map.min_binding : 'a t -> key * 'a	CCMap.min_binding : 'a t -> key * 'a	BatMap.min_binding : ('key, 'a) t -> 'key * 'a	Base.Map.min_elt_exn : ('k, 'v, 'a) t -> 'k * 'v	Map.minKeyValue : ???
Map.min_binding_opt : 'a t -> (key * 'a) option	CCMap.min_binding_opt : 'a t -> (key * 'a) option	BatMap.min_binding_opt : ('key, 'a) t -> ('key * 'a) option	Base.Map.min_elt : ('k, 'v, 'a) t -> ('k * 'v) option	
		BatMap.modify : 'a -> ('b -> 'b) -> ('a, 'b) t -> ('a, 'b) t		
		BatMap.modify_def : 'b -> 'a -> ('b -> 'b) -> ('a, 'b) t -> ('a, 'b) t		
		BatMap.modify_opt : 'a -> ('b option -> 'b option) -> ('a, 'b) t -> ('a, 'b) t		
			Base.Map.nth : ('k, 'v, 'a) t -> int -> ('k * 'v) option	
			Base.Map.nth_exn : ('k, 'v, 'a) t -> int -> 'k * 'v	
			Base.Map.of_alist : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> [`Duplicate_key of 'a | `Ok of ('a, 'b, 'cmp) t]	
			Base.Map.of_alist_exn : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> ('a, 'b, 'cmp) t	
			Base.Map.of_alist_fold : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> init:'c -> f:('c -> 'b -> 'c) -> ('a, 'c, 'cmp) t	
			Base.Map.of_alist_multi : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> ('a, 'b list, 'cmp) t	
			Base.Map.of_alist_or_error : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> ('a, 'b, 'cmp) t Base__.Or_error.t	
			Base.Map.of_alist_reduce : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) list -> f:('b -> 'b -> 'b) -> ('a, 'b, 'cmp) t	
				Map.ofArray : (('a * 'b) [] -> Map<'a,'b>) when 'a : comparison
		BatMap.of_enum : ('a * 'b) BatEnum.t -> ('a, 'b) t		
			Base.Map.of_increasing_iterator_unchecked : ('a, 'cmp) Base__.Comparator.Module.t -> len:int -> f:(int -> 'a * 'b) -> ('a, 'b, 'cmp) t	
			Base.Map.of_increasing_sequence : ('k, 'cmp) Base__.Comparator.Module.t -> ('k * 'v) Base__.Sequence.t -> ('k, 'v, 'cmp) t Base__.Or_error.t	
	CCMap.of_iter : (key * 'a) CCMap.iter -> 'a t			
	CCMap.of_iter_with : f:(key -> 'a -> 'a -> 'a) -> (key * 'a)			

Stdlib	Containers	Batteries	Base	F#
	CCMap.iter -> 'a t			
			Base.Map.of_iteri : ('a, 'cmp) Base__.Comparator.Module.t -> iteri:(f:(key:'a -> data:'b -> unit) -> unit) -> [`Duplicate_key of 'a | `Ok of ('a, 'b, 'cmp) t]	
			Base.Map.of_iteri_exn : ('a, 'cmp) Base__.Comparator.Module.t -> iteri:(f:(key:'a -> data:'b -> unit) -> unit) -> ('a, 'b, 'cmp) t	
	CCMap.of_list : (key * 'a) list -> 'a t			Map.ofList : (('a * 'b) list -> Map<'a,'b>) when 'a : comparison
	CCMap.of_list_with : f:(key -> 'a -> 'a -> 'a) -> (key * 'a) list -> 'a t			
			Base.Map.of_sequence : ('k, 'cmp) Base__.Comparator.Module.t -> ('k * 'v) Base__.Sequence.t -> [`Duplicate_key of 'k | `Ok of ('k, 'v, 'cmp) t]	
Map.of_seq : (key * 'a) Seq.t -> 'a t	CCMap.of_seq : (key * 'a) Seq.t -> 'a t	BatMap.of_seq : ('key * 'a) BatSeq.t -> ('key, 'a) t	Base.Map.of_sequence_exn : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) Base__.Sequence.t -> ('a, 'b, 'cmp) t	Map.ofSeq : (seq<'a * 'b> -> Map<'a,'b>) when 'a : comparison
			Base.Map.of_sequence_fold : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) Base__.Sequence.t -> init:'c -> f:('c -> 'b -> 'c) -> ('a, 'c, 'cmp) t	
			Base.Map.of_sequence_multi : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) Base__.Sequence.t -> ('a, 'b list, 'cmp) t	
			Base.Map.of_sequence_or_error : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) Base__.Sequence.t -> ('a, 'b, 'cmp) t Base__.Or_error.t	
			Base.Map.of_sequence_reduce : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) Base__.Sequence.t -> f:('b -> 'b -> 'b) -> ('a, 'b, 'cmp) t	
	CCMap.of_seq_with : f:(key -> 'a -> 'a -> 'a) -> (key * 'a) Seq.t -> 'a t		Base.Map.of_sorted_array : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) array -> ('a, 'b, 'cmp) t Base__.Or_error.t	
			Base.Map.of_sorted_array_unchecked : ('a, 'cmp) Base__.Comparator.Module.t -> ('a * 'b) array -> ('a, 'b, 'cmp) t	
			Base.Map.of_tree : ('k, 'cmp) Base__.Comparator.Module.t -> ('k, 'v, 'cmp) Using_comparator.Tree.t -> ('k, 'v, 'cmp) t	
Map.partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t	CCMap.partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t	BatMap.partition : ('a -> 'b -> bool) -> ('a, 'b) t -> ('a, 'b) t * ('a, 'b) t		Map.partition : (('a -> 'b -> bool) -> Map<'a,'b> -> Map<'a,'b> * Map<'a,'b>) when 'a : comparison
			Base.Map.partition_map : ('k, 'v1, 'cmp) t -> f:('v1 -> ('v2, 'v3) Base__.Either.t) -> ('k, 'v2, 'cmp) t * ('k, 'v3, 'cmp) t	
			Base.Map.partition_mapi : ('k, 'v1, 'cmp) t -> f:(key:'k -> data:'v1 -> ('v2, 'v3) Base__.Either.t) -> ('k, 'v2, 'cmp) t * ('k, 'v3, 'cmp) t	
			Base.Map.partition_tf : ('k, 'v, 'cmp) t -> f:('v -> bool) -> ('k, 'v, 'cmp) t * ('k, 'v, 'cmp) t	
			Base.Map.partitioni_tf : ('k, 'v, 'cmp) t -> f:(key:'k -> data:'v -> bool) -> ('k, 'v, 'cmp) t * ('k, 'v, 'cmp) t	
				Map.pick : (('a -> 'b -> 'c option) -> Map<'a,'b> -> 'c) when 'a : comparison
		BatMap.pop : ('a, 'b) t -> ('a * 'b) * ('a, 'b) t		
		BatMap.pop_max_binding : ('key, 'a) t -> ('key * 'a) * ('key, 'a) t		
		BatMap.pop_min_binding : ('key, 'a) t -> ('key * 'a) * ('key, 'a) t		
	CCMap.pp : ?pp_start:unit CCMap.printer -> ?pp_stop:unit CCMap.printer -> ?pp_arrow:unit CCMap.printer -> ?pp_sep:unit CCMap.printer -> key CCMap.printer -> 'a CCMap.printer -> 'a t CCMap.printer			
		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) t -> unit		

Stdlib	Containers	Batteries	Base	F#
			Base.Map.range_to_alist : ('k, 'v, 'cmp) t -> min:'k -> max:'k -> ('k * 'v) list	
			Base.Map.rank : ('k, 'v, 'cmp) t -> 'k -> int option	
Map.remove : key -> 'a t -> 'a t	CCMap.remove : key -> 'a t -> 'a t	BatMap.remove : 'a -> ('a, 'b) t -> ('a, 'b) t	Base.Map.remove : ('k, 'v, 'cmp) t -> 'k -> ('k, 'v, 'cmp) t	Map.remove : ('a -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
		BatMap.remove_exn : 'a -> ('a, 'b) t -> ('a, 'b) t		
			Base.Map.remove_multi : ('k, 'v list, 'cmp) t -> 'k -> ('k, 'v list, 'cmp) t	
			Base.Map.set : ('k, 'v, 'cmp) t -> key:'k -> data:'v -> ('k, 'v, 'cmp) t	
			Base.Map.sexp_of_m__t : (module Sexp_of_m with type t = 'k) -> ('v -> Base__.Sexp.t) -> ('k, 'v, 'cmp) t -> Base__.Sexp.t	
Map.singleton : key -> 'a -> 'a t	CCMap.singleton : key -> 'a -> 'a t	BatMap.singleton : 'a -> 'b -> ('a, 'b) t	Base.Map.singleton : ('a, 'cmp) Base__.Comparator.Module.t -> 'a -> 'b -> ('a, 'b, 'cmp) t	
Map.split : key -> 'a t -> 'a t * 'a option * 'a t	CCMap.split : key -> 'a t -> 'a t * 'a option * 'a t	BatMap.split : 'key -> ('key, 'a) t -> ('key, 'a) t * 'a option * ('key, 'a) t	Base.Map.split : ('k, 'v, 'cmp) t -> 'k -> ('k, 'v, 'cmp) t * ('k * 'v) option * ('k, 'v, 'cmp) t	
			Base.Map.subrange : ('k, 'v, 'cmp) t -> lower_bound:'k Base__.Maybe_bound.t -> upper_bound:'k Base__.Maybe_bound.t -> ('k, 'v, 'cmp) t	
			Base.Map.symmetric_diff : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> data_equal:('v -> 'v -> bool) -> ('k, 'v) Symmetric_diff_element.t Base__.Sequence.t	
			Base.Map.to_alist : ?key_order:[`Decreasing | `Increasing] -> ('k, 'v, 'a) t -> ('k * 'v) list	
				Map.toArray : (Map<'a,'b> -> ('a * 'b) []) when 'a : comparison
	CCMap.to_iter : 'a t -> (key * 'a) CCMap.iter			
	CCMap.to_list : 'a t -> (key * 'a) list			Map.toList : (Map<'a,'b> -> ('a * 'b) list) when 'a : comparison
Map.to_rev_seq : 'a t -> (key * 'a) Seq.t	CCMap.to_rev_seq : 'a t -> (key * 'a) Seq.t	BatMap.to_rev_seq : ('key, 'a) t -> ('key * 'a) BatSeq.t		
Map.to_seq : 'a t -> (key * 'a) Seq.t	CCMap.to_seq : 'a t -> (key * 'a) Seq.t	BatMap.to_seq : ('key, 'a) t -> ('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) t -> ('k * 'v) Base__.Sequence.t	Map.toSeq : (Map<'a,'b> -> seq<'a * 'b>) when 'a : comparison
Map.to_seq_from : key -> 'a t -> (key * 'a) Seq.t	CCMap.to_seq_from : key -> 'a t -> (key * 'a) Seq.t	BatMap.to_seq_from : 'key -> ('key, 'a) t -> ('key * 'a) BatSeq.t		
			Base.Map.to_tree : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) Using_comparator.Tree.t	
				Map.tryFind : ('a -> Map<'a,'b> -> 'b option) when 'a : comparison
				Map.tryFindKey : (('a -> 'b -> bool) -> Map<'a,'b> -> 'a option) when 'a : comparison
				Map.tryPick : (('a -> 'b -> 'c option) -> Map<'a,'b> -> 'c option) when 'a : comparison
Map.union : (key -> 'a -> 'a -> 'a option) -> 'a t -> 'a t -> 'a t	CCMap.union : (key -> 'a -> 'a -> 'a option) -> 'a t -> 'a t -> 'a t	BatMap.union : ('a, 'b) t -> ('a, 'b) t -> ('a, 'b) t		
		BatMap.union_stdlib : ('key -> 'a -> 'a -> 'a option) -> ('key, 'a) t -> ('key, 'a) t -> ('key, 'a) t		
Map.update : key -> ('a option -> 'a option) -> 'a t -> 'a t	CCMap.update : key -> ('a option -> 'a option) -> 'a t -> 'a t	BatMap.update : 'a -> 'a -> 'b -> ('a, 'b) t -> ('a, 'b) t	Base.Map.update : ('k, 'v, 'cmp) t -> 'k -> f:('v option -> 'v) -> ('k, 'v, 'cmp) t	
		BatMap.update_stdlib : 'a -> ('b option -> 'b option) -> ('a, 'b) t -> ('a, 'b) t		
	CCMap.values : 'a t -> 'a CCMap.iter	BatMap.values : ('a, 'b) t -> 'b BatEnum.t		Map.values : ???
		BatMap.(-->) : ('a, 'b) t -> 'a -> 'b		

Stdlib	Containers	Batteries	Base	F#
		BatMap.(<--) : ('a, 'b) t -> 'a * 'b -> ('a, 'b) t		
		BatMap.Exceptionless.any : ('a, 'b) t -> ('a * 'b) option		
		BatMap.Exceptionless.choose : ('a, 'b) t -> ('a * 'b) option		
		BatMap.Exceptionless.find : 'a -> ('a, 'b) t -> 'b option		
			Base.Option.all : 'a t list -> 'a list t	
			Base.Option.all_unit : unit t list -> unit t	
		BatOption.apply : ('a -> 'a) option -> 'a -> 'a	Base.Option.apply : ('a -> 'b) t -> 'a t -> 'b t	
Option.bind : 'a option -> ('a -> 'b option) -> 'b option	CCOption.bind : 'a t -> ('a -> 'b t) -> 'b t	BatOption.bind : 'a option -> ('a -> 'b option) -> 'b option	Base.Option.bind : 'a t -> f:('a -> 'b t) -> 'b t	Option.bind : (('a -> 'b option) -> 'a option -> 'b option)
			Base.Option.both : 'a t -> 'b t -> ('a * 'b) t	
			Base.Option.call : 'a -> f:('a -> unit) t -> unit	
	CCOption.choice : 'a t list -> 'a t			
	CCOption.choice_iter : 'a t iter -> 'a t			
	CCOption.choice_seq : 'a t Seq.t -> 'a t			
Option.compare : ('a -> 'a -> int) -> 'a option -> 'a option -> int	CCOption.compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int	BatOption.compare : ?cmp:('a -> 'a -> int) -> 'a option -> 'a option -> int	Base.Option.compare : 'a Base__Ppx_compare_lib.compare -> 'a t Base__Ppx_compare_lib.compare	
				Option.contains : ('a -> 'a option -> bool) when 'a : equality
			Base.Option.count : 'a t -> f:('a -> bool) -> int	Option.count : ('a option -> int)
		BatOption.default : 'a -> 'a option -> 'a		Option.defaultValue : ('a -> 'a option -> 'a)
				Option.defaultWith : ((unit -> '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	CCOption.equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool		Base.Option.equal : 'a Base__Equal.equal -> 'a t Base__Equal.equal	
	CCOption.exists : ('a -> bool) -> 'a t -> bool		Base.Option.exists : 'a t -> f:('a -> bool) -> bool	Option.exists : (('a -> bool) -> 'a option -> bool)
	CCOption.filter : ('a -> bool) -> 'a t -> 'a t	BatOption.filter : ('a -> bool) -> 'a option -> 'a option	Base.Option.filter : 'a t -> f:('a -> bool) -> 'a t	Option.filter : (('a -> bool) -> 'a option -> 'a option)
			Base.Option.find : 'a t -> f:('a -> bool) -> 'a option	
			Base.Option.find_map : 'a t -> f:('a -> 'b option) -> 'b option	
			Base.Option.first_some : 'a t -> 'a t -> 'a t	
	CCOption.flat_map : ('a -> 'b t) -> 'a t -> 'b t			
	CCOption.flatten : 'a t t -> 'a t			
				Option.flatten : ('a option option -> 'a option)
Option.fold : none:'a -> some:('b -> 'a) -> 'b option -> 'a	CCOption.fold : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a		Base.Option.fold : 'a t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum	Option.fold : (('a -> 'b -> 'a) -> 'a -> 'b option -> 'a)
			Base.Option.fold_result : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t	

Stdlib	Containers	Batteries	Base	F#
			Base.Option.fold_until : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__.Container.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final	
				Option.foldBack : (('a -> 'b -> 'b) -> 'a option -> 'b -> 'b)
	CCOption.for_all : ('a -> bool) -> 'a t -> bool		Base.Option.for_all : 'a t -> f:('a -> bool) -> bool	Option.forall : (('a -> bool) -> 'a option -> bool)
Option.get : 'a option -> 'a		BatOption.get : 'a option -> 'a		Option.get : ('a option -> 'a)
	CCOption.get_exn : 'a t -> 'a	BatOption.get_exn : 'a option -> exn -> 'a		
	CCOption.get_exn_or : string -> 'a t -> 'a			
	CCOption.get_lazy : (unit -> 'a) -> 'a t -> 'a			
	CCOption.get_or : default:'a -> 'a t -> 'a			
			Base.Option.hash_fold_t : 'a Base__Ppx_hash_lib.hash_fold -> 'a t Base__Ppx_hash_lib.hash_fold	
	CCOption.if_ : ('a -> bool) -> 'a -> 'a option			
			Base.Option.ignore_m : 'a t -> unit t	
			Base.Option.invariant : 'a Base__Invariant_intf.inv -> 'a t Base__Invariant_intf.inv	
			Base.Option.is_empty : 'a t -> bool	
Option.is_none : 'a option -> bool	CCOption.is_none : 'a t -> bool	BatOption.is_none : 'a option -> bool	Base.Option.is_none : 'a t -> bool	Option.isNone : ('a option -> bool)
Option.is_some : 'a option -> bool	CCOption.is_some : 'a t -> bool	BatOption.is_some : 'a option -> bool	Base.Option.is_some : 'a t -> bool	Option.isSome : ('a option -> bool)
Option.iter : ('a -> unit) -> 'a option -> unit	CCOption.iter : ('a -> unit) -> 'a t -> unit		Base.Option.iter : 'a t -> f:('a -> unit) -> unit	Option.iter : (('a -> unit) -> 'a option -> unit)
Option.join : 'a option option -> 'a option			Base.Option.join : 'a t t -> 'a t	
			Base.Option.length : 'a t -> int	
Option.map : ('a -> 'b) -> 'a option -> 'b option	CCOption.map : ('a -> 'b) -> 'a t -> 'b t	BatOption.map : ('a -> 'b) -> 'a option -> 'b option	Base.Option.map : 'a t -> f:('a -> 'b) -> 'b t	Option.map : (('a -> 'b) -> 'a option -> 'b option)
	CCOption.map2 : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t		Base.Option.map2 : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t	Option.map2 : (('a -> 'b -> 'c) -> 'a option -> 'b option -> 'c option)
			Base.Option.map3 : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t	Option.map3 : (('a -> 'b -> 'c -> 'd) -> 'a option -> 'b option -> 'c option -> 'd option)
		BatOption.map_default : ('a -> 'b) -> 'b -> 'a option -> 'b		
		BatOption.map_default_delayed : ('a -> 'b) -> (unit -> 'b) -> 'a option -> 'b		
	CCOption.map_lazy : (unit -> 'b) -> ('a -> 'b) -> 'a t -> 'b			
	CCOption.map_or : default:'b -> ('a -> 'b) -> 'a t -> 'b			
			Base.Option.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
		BatOption.may : ('a -> unit) -> 'a option -> unit		
			Base.Option.mem : 'a t -> 'a -> equal:('a -> 'a -> bool) -> bool	
			Base.Option.merge : 'a t -> 'a t -> f:('a -> 'a -> 'a) -> 'a t	
			Base.Option.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	
Option.none : 'a option	CCOption.none : 'a t			
		BatOption.of_enum : 'a BatEnum.t -> 'a option		
	CCOption.of_list : 'a list -> 'a t			
				Option.ofNullable : (System.Nullable<'a> -> 'a option) when 'a : (new : unit -> 'a) and 'a : struct and 'a :> System.ValueType
				Option.ofObj : ('a -> 'a option) when 'a : null

Stdlib	Containers	Batteries	Base	F#
	CCOption.of_result : ('a, 'b) result -> 'a t			
		BatOption.ord : 'a BatOrd.ord -> 'a option BatOrd.ord		
	CCOption.or_ : else_:'a t -> 'a t -> 'a t			Option.orElse : ('a option -> 'a option -> 'a option)
				Option.orElseWith : ((unit -> 'a option) -> 'a option -> 'a option)
	CCOption.or_lazy : else_:(unit -> 'a t) -> 'a t -> 'a t			
	CCOption.pp : 'a printer -> 'a t printer			
		BatOption.print : ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b t -> unit		
	CCOption.pure : 'a -> 'a t			
	CCOption.random : 'a random_gen -> 'a t random_gen			
	CCOption.return : 'a -> 'a t		Base.Option.return : 'a -> 'a t	
	CCOption.return_if : bool -> 'a -> 'a t			
	CCOption.sequence_l : 'a t list -> 'a list t			
			Base.Option.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> Sexplib0__.Sexp.t	
Option.some : 'a -> 'a option	CCOption.some : 'a -> 'a t	BatOption.some : 'a -> 'a option	Base.Option.some : 'a -> 'a t	
			Base.Option.some_if : bool -> 'a -> 'a t	
			Base.Option.sum : (module Base__.Container.Summable with type t = 'sum) -> 'a t -> f:('a -> 'sum) -> 'sum	
			Base.Option.t_of_sexp : (Sexplib0__.Sexp.t -> 'a) -> Sexplib0__.Sexp.t -> 'a t	
			Base.Option.t_sexp_grammar : 'a Sexplib0.Sexp_grammar.t -> 'a t Sexplib0.Sexp_grammar.t	
			Base.Option.to_array : 'a t -> 'a array	Option.toArray : ('a option -> 'a [])
	CCOption.to_gen : 'a t -> 'a gen			
	CCOption.to_iter : 'a t -> 'a iter			
Option.to_list : 'a option -> 'a list	CCOption.to_list : 'a t -> 'a list		Base.Option.to_list : 'a t -> 'a list	Option.toList : ('a option -> 'a list)
				Option.toNullable : ('a option -> System.Nullable<'a>) when 'a : (new : unit -> 'a) and 'a : struct and 'a :> System.ValueType
				Option.toObj : ('a option -> 'a) when 'a : null
Option.to_result : none:'e -> 'a option -> ('a, 'e) result	CCOption.to_result : 'e -> 'a t -> ('a, 'e) result			
	CCOption.to_result_lazy : (unit -> 'e) -> 'a t -> ('a, 'e) result			
Option.to_seq : 'a option -> 'a Seq.t	CCOption.to_seq : 'a t -> 'a Seq.t			
			Base.Option.try_with : (unit -> 'a) -> 'a t	
			Base.Option.try_with_join : (unit -> 'a t) -> 'a t	
Option.value : 'a option -> default:'a -> 'a	CCOption.value : 'a t -> default:'a -> 'a		Base.Option.value : 'a t -> default:'a -> 'a	
			Base.Option.value_exn : ?here:Base__.Source_code_position0.t -> ?error:Base__.Error.t -> ?message:string -> 'a t -> 'a	
			Base.Option.value_map : 'a t -> default:'b -> f:('a -> 'b) -> 'b	
			Base.Option.value_or_thunk : 'a t -> default:(unit -> 'a) -> 'a	

Stdlib	Containers	Batteries	Base	F#
	CCOption.wrap : ?handler:(exn -> bool) -> ('a -> 'b) -> 'a -> 'b option			
	CCOption.wrap2 : ?handler:(exn -> bool) -> ('a -> 'b -> 'c) -> 'a -> 'b -> 'c option			
			Base.Option.(*>) : unit t -> 'a t -> 'a t	
			Base.Option.(<*) : 'a t -> unit t -> 'a t	
	CCOption.(<$>) : ('a -> 'b) -> 'a t -> 'b t			
	CCOption.(<*>) : ('a -> 'b) t -> 'a t -> 'b t		Base.Option.(<*>) : ('a -> 'b) t -> 'a t -> 'b t	
	CCOption.(<+>) : 'a t -> 'a t -> 'a t			
	CCOption.(>>=) : 'a t -> ('a -> 'b t) -> 'b t		Base.Option.(>>=) : 'a t -> ('a -> 'b t) -> 'b t	
	CCOption.(>|=) : 'a t -> ('a -> 'b) -> 'b t		Base.Option.(>>|) : 'a t -> ('a -> 'b) -> 'b t	
	CCOption.(and*) : 'a t -> 'b t -> ('a * 'b) t			
	CCOption.(and+) : 'a t -> 'b t -> ('a * 'b) t			
	CCOption.(let*) : 'a t -> ('a -> 'b t) -> 'b t			
	CCOption.(let+) : 'a t -> ('a -> 'b) -> 'b t			
		BatOption.(|?) : 'a option -> 'a -> 'a		
		BatOption.Infix.(>>=) : 'a option -> ('a -> 'b option) -> '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		
		BatOption.Monad.bind : 'a m -> ('a -> 'b m) -> 'b m		
		BatOption.Monad.return : 'a -> 'a m		
Printf.bprintf : Buffer.t -> ('a, Buffer.t, unit) format -> 'a		BatPrintf.bprintf : Buffer.t -> ('a, Buffer.t, unit) t -> 'a	Base.Printf.bprintf : Buffer.t -> ('r, Buffer.t, unit) format -> 'r	Printf.bprintf : (System.Text.StringBuilder -> Printf.BuilderFormat<'a> -> 'a)
		BatPrintf.bprintf2 : Buffer.t -> ('b, 'a BatInnerIO.output, unit) t -> 'b		
Printf.eprintf : ('a, out_channel, unit) format -> 'a		BatPrintf.eprintf : ('b, 'a BatInnerIO.output, unit) t -> 'b		Printf.eprintf : (Printf.TextWriterFormat<'a> -> 'a)
				Printf.eprintfn : (Printf.TextWriterFormat<'a> -> 'a)
				Printf.failwithf : (Printf.StringFormat<'a,'b> -> 'a)
Printf.fprintf : out_channel -> ('a, out_channel, unit) format -> 'a		BatPrintf.fprintf : 'a BatInnerIO.output -> ('b, 'a BatInnerIO.output, unit) t -> 'b		Printf.fprintf : (System.IO.TextWriter -> Printf.TextWriterFormat<'a> -> 'a)
				Printf.fprintfn : (System.IO.TextWriter -> Printf.TextWriterFormat<'a> -> 'a)
Printf.ibprintf : Buffer.t -> ('a, Buffer.t, unit) format -> 'a				
Printf.ifprintf : 'b -> ('a, 'b, 'c, unit) format4 -> 'a		BatPrintf.ifprintf : 'c -> ('b, 'a BatInnerIO.output, unit) 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				
Printf.ikfprintf : ('b -> 'd) -> 'b -> ('a, 'b, 'c, 'd) format4 -> 'a				

Stdlib	Containers	Batteries	Base	F#
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 : (Buffer.t -> 'a) -> Buffer.t -> ('r, Buffer.t, unit, 'a) format4 -> 'r	Printf.kbprintf : ((unit -> 'a) -> System.Text.StringBuilder -> Printf.BuilderFormat<'b,'a> -> 'b)
		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		Printf.kfprintf : ((unit -> 'a) -> System.IO.TextWriter -> Printf.TextWriterFormat<'b,'a> -> 'b)
Printf.kprintf : (string -> 'b) -> ('a, unit, string, 'b) format4 -> 'a		BatPrintf.kprintf : (string -> 'a) -> ('b, unit, string, 'a) format4 -> 'b		Printf.kprintf : ((string -> 'a) -> Printf.StringFormat<'b,'a> -> 'b)
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	Printf.ksprintf : ((string -> 'a) -> Printf.StringFormat<'b,'a> -> 'b)
		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) t -> 'b		Printf.printf : (Printf.TextWriterFormat<'a> -> 'a)
				Printf.printfn : (Printf.TextWriterFormat<'a> -> 'a)
Printf.sprintf : ('a, unit, string) format -> 'a		BatPrintf.sprintf : ('a, unit, string) t -> 'a	Base.Printf.sprintf : (‘r, unit, string) format -> ‘r	Printf.sprintf : (Printf.StringFormat<'a> -> 'a)
		BatPrintf.sprintf2 : ('a, 'b BatInnerIO.output, unit, string) format4 -> 'a		
			Base.Printf.failwithf : (‘r, unit, string, unit -> 'a) format4 -> 'r	
			Base.Printf.invalid_argf : (‘r, unit, string, unit -> 'a) format4 -> 'r	
	CCResult.add_ctx : string -> ('a, string) t -> ('a, string) t			
	CCResult.add_ctxf : ('a, Format.formatter, unit, ('b, string) t -> ('b, string) t) format4 -> 'a			
			Base.Result.all : ('a, 'e) t list -> ('a list, 'e) t	
			Base.Result.all_unit : (unit, 'e) t list -> (unit, 'e) t	
Result.bind : ('a, 'e) result -> ('a -> ('b, 'e) result) -> ('b, 'e) result		BatResult.bind : ('a, 'e) t -> ('a -> ('b, 'e) t) -> ('b, 'e) t	Base.Result.bind : ('a, 'e) t -> f:('a -> ('b, 'e) t) -> ('b, 'e) t	Result.bind : (('a -> Result<'b,'c>) -> Result<'a,'c> -> Result<'b,'c>)
	CCResult.both : ('a, 'err) t -> ('b, 'err) t -> ('a * 'b, 'err) t			
	CCResult.catch : ('a, 'err) t -> ok:('a -> 'b) -> err:('err -> 'b) -> 'b	BatResult.catch : ('a -> 'e) -> 'a -> ('e, exn) t		
		BatResult.catch2 : ('a -> 'b -> 'c) -> 'a -> 'b -> ('c, exn) t		
		BatResult.catch3 : ('a -> 'b -> 'c -> 'd) -> 'a -> 'b -> 'c -> ('d, exn) t		
	CCResult.choose : ('a, 'err) t list -> ('a, 'err list) t			
			Base.Result.combine : ('ok1, 'err) t -> ('ok2, 'err) t -> ok:('ok1 -> 'ok2 -> 'ok3) -> err:('err -> 'err -> 'err) -> ('ok3, 'err) t	
			Base.Result.combine_errors : ('ok, 'err) t list -> ('ok list, 'err list) t	
			Base.Result.combine_errors_unit : (unit, 'err) t list -> (unit, 'err list) t	
Result.compare : ok:('a -> 'a -> int) -> error:('e -> 'e -> int) -> ('a, 'e) result -> ('a, 'e) result -> int	CCResult.compare : err:'err ord -> 'a ord -> ('a, 'err) t ord	BatResult.compare : ok:('a -> 'a -> int) -> error:('e -> 'e -> int) -> ('a, 'e) t -> ('a, 'e) t -> int	Base.Result.compare : 'a Base__Ppx_compare_lib.compare -> 'b Base__Ppx_compare_lib.compare -> ('a, 'b) t Base__Ppx_compare_lib.compare	
		BatResult.default : 'a -> ('a, 'b) t -> 'a		
Result.equal : ok:('a -> 'a -> bool) -> error:('e -> 'e -> bool) -> ('a, 'e) result -> ('a, 'e) result -> bool	CCResult.equal : err:'err equal -> 'a equal -> ('a, 'err) t equal	BatResult.equal : ok:('a -> 'a -> bool) -> error:('e -> 'e -> bool) -> ('a, 'e) t -> ('a, 'e) t -> bool	Base.Result.equal : 'a Base__Ppx_compare_lib.equal -> 'b Base__Ppx_compare_lib.equal -> ('a, 'b) t Base__Ppx_compare_lib.equal	

Stdlib	Containers	Batteries	Base	F#
Result.error : 'e -> ('a, 'e) result	CCResult.fail : 'err -> ('a, 'err) t	BatResult.error : 'e -> ('a, 'e) t	Base.Result.fail : 'err -> ('a, 'err) t	Result.Error : arg0:'a -> Result<'b,'a>
			Base.Result.error : ('a, 'err) t -> 'err option	
	CCResult.fail_fprintf : ('a, Format.formatter, unit, ('b, string) t) format4 -> 'a			
	CCResult.fail_printf : ('a, Buffer.t, unit, ('b, string) t) format4 -> 'a			
			Base.Result.failf : ('a, unit, string, ('b, string) t) format4 -> 'a	
	CCResult.flat_map : ('a -> ('b, 'err) t) -> ('a, 'err) t -> ('b, 'err) t			
	CCResult.flatten_l : ('a, 'err) t list -> ('a list, 'err) t			
Result.fold : ok:('a -> 'c) -> error:('e -> 'c) -> ('a, 'e) result -> 'c	CCResult.fold : ok:('a -> 'b) -> error:('err -> 'b) -> ('a, 'err) t -> 'b	BatResult.fold : ok:('a -> 'c) -> error:('e -> 'c) -> ('a, 'e) t -> 'c		
	CCResult.fold_iter : ('b -> 'a -> ('b, 'err) t) -> 'b -> 'a iter -> ('b, 'err) t			
	CCResult.fold_l : ('b -> 'a -> ('b, 'err) t) -> 'b -> 'a list -> ('b, 'err) t			
	CCResult.fold_ok : ('a -> 'b -> 'a) -> 'a -> ('b, 'c) t -> 'a			
		BatResult.get : ('a, exn) t -> 'a		
Result.get_error : ('a, 'e) result -> 'e		BatResult.get_error : ('a, 'e) t -> 'e		
	CCResult.get_exn : ('a, 'b) t -> 'a			
	CCResult.get_lazy : ('b -> 'a) -> ('a, 'b) t -> 'a			
Result.get_ok : ('a, 'e) result -> 'a		BatResult.get_ok : ('a, 'e) t -> 'a		
	CCResult.get_or : ('a, 'b) t -> default:'a -> 'a			
	CCResult.get_or_failwith : ('a, string) t -> 'a			
	CCResult.guard : (unit -> 'a) -> ('a, exn) t			
	CCResult.guard_str : (unit -> 'a) -> ('a, string) t			
	CCResult.guard_str_trace : (unit -> 'a) -> ('a, string) t			
			Base.Result.hash_fold_t : 'a Base__Ppx_hash_lib.hash_fold -> 'b Base__Ppx_hash_lib.hash_fold -> ('a, 'b) t Base__Ppx_hash_lib.hash_fold	
			Base.Result.ignore_m : ('a, 'e) t -> (unit, 'e) t	
			Base.Result.invariant : 'a Base__Invariant_intf.inv -> 'b Base__Invariant_intf.inv -> ('a, 'b) t Base__Invariant_intf.inv	
		BatResult.is_bad : ('a, 'e) t -> bool		
Result.is_error : ('a, 'e) result -> bool	CCResult.is_error : ('a, 'err) t -> bool	BatResult.is_error : ('a, 'e) t -> bool	Base.Result.is_error : ('a, 'b) t -> bool	
		BatResult.is_exn : exn -> ('a, exn) t -> bool		
Result.is_ok : ('a, 'e) result -> bool	CCResult.is_ok : ('a, 'err) t -> bool	BatResult.is_ok : ('a, 'e) t -> bool	Base.Result.is_ok : ('a, 'b) t -> bool	
Result.iter : ('a -> unit) -> ('a, 'e) result -> unit	CCResult.iter : ('a -> unit) -> ('a, 'b) t -> unit	BatResult.iter : ('a -> unit) -> ('a, 'e) t -> unit	Base.Result.iter : ('ok, 'a) t -> f:('ok -> unit) -> unit	
Result.iter_error : ('e -> unit) -> ('a, 'e) result -> unit	CCResult.iter_err : ('err -> unit) -> ('a, 'err) t -> unit	BatResult.iter_error : ('e -> unit) -> ('a, 'e) t -> unit	Base.Result.iter_error : ('a, 'err) t -> f:('err -> unit) -> unit	
Result.join : (('a, 'e) result, 'e) result -> ('a, 'e) result	CCResult.join : (('a, 'err) t, 'err) t -> ('a, 'err) t	BatResult.join : (('a, 'e) t, 'e) t -> ('a, 'e) t	Base.Result.join : (('a, 'e) t, 'e) t -> ('a, 'e) t	
Result.map : ('a -> 'b) -> ('a, 'e) result -> ('b, 'e) result	CCResult.map : ('a -> 'b) -> ('a, 'err) t -> ('b, 'err) t	BatResult.map : ('a -> 'b) -> ('a, 'e) t -> ('b, 'e) t	Base.Result.map : ('ok, 'err) t -> f:('ok -> 'c) -> ('c, 'err) t	Result.map : (('a -> 'b) -> Result<'a,'c> -> Result<'b,'c>)
	CCResult.map2 : ('a -> 'b) -> ('err1 -> 'err2) -> ('a, 'err1) t -> ('b, 'err2) t	BatResult.map_both : ('a1 -> 'a2) -> ('b1 -> 'b2) -> ('a1, 'b1) t -> ('a2, 'b2) t		
		BatResult.map_default : 'b -> ('a -> 'b) -> ('a, 'c) t ->		

Stdlib	Containers	Batteries	Base	F#
		'b		
Result.map_error : ('e -> 'f) -> ('a, 'e) result -> ('a, 'f) result	CCResult.map_err : ('err1 -> 'err2) -> ('a, 'err1) t -> ('a, 'err2) t	BatResult.map_error : ('e -> 'f) -> ('a, 'e) t -> ('a, 'f) t	Base.Result.map_error : ('ok, 'err) t -> f:('err -> 'c) -> ('ok, 'c) t	Result.mapError : (('a -> 'b) -> Result<'c,'a> -> Result<'c,'b>)
	CCResult.map_l : ('a -> ('b, 'err) t) -> 'a list -> ('b list, 'err) t			
	CCResult.map_or : ('a -> 'b) -> ('a, 'c) t -> default:'b -> 'b			
			Base.Result.of_either : ('ok, 'err) Base__.Either0.t -> ('ok, 'err) t	
	CCResult.of_err : ('a, 'b) error -> ('a, 'b) t			
	CCResult.of_exn : exn -> ('a, string) t			
	CCResult.of_exn_trace : exn -> ('a, string) t			
	CCResult.of_opt : 'a option -> ('a, string) t			
		BatResult.of_option : 'a option -> ('a, unit) t	Base.Result.of_option : 'ok option -> error:'err -> ('ok, 'err) t	
Result.ok : 'a -> ('a, 'e) result		BatResult.ok : 'a -> ('a, 'b) t		Result.Ok : arg0:'a -> Result<'a,'b>
			Base.Result.ok : ('ok, 'a) t -> 'ok option	
			Base.Result.ok_exn : ('ok, exn) t -> 'ok	
			Base.Result.ok_fst : ('ok, 'err) t -> ('ok, 'err) Base__.Either0.t	
			Base.Result.ok_if_true : bool -> error:'err -> (unit, 'err) t	
			Base.Result.ok_or_failwith : ('ok, string) t -> 'ok	
	CCResult.opt_map : ('a -> ('b, 'c) t) -> 'a option -> ('b option, 'c) t			
	CCResult.pp : 'a printer -> ('a, string) t printer			
	CCResult.pp' : 'a printer -> 'e printer -> ('a, 'e) t printer			
		BatResult.print : ('b BatInnerIO.output -> 'a -> unit) -> 'b BatInnerIO.output -> ('a, exn) t -> unit		
	CCResult.pure : 'a -> ('a, 'err) t			
	CCResult.retry : int -> (unit -> ('a, 'err) t) -> ('a, 'err list) t			
	CCResult.return : 'a -> ('a, 'err) t		Base.Result.return : 'a -> ('a, 'b) t	
			Base.Result.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> ('b -> Sexplib0__.Sexp.t) -> ('a, 'b) t -> Sexplib0__.Sexp.t	
			Base.Result.t_of_sexp : (Sexplib0__.Sexp.t -> 'a) -> (Sexplib0__.Sexp.t -> 'b) -> Sexplib0__.Sexp.t -> ('a, 'b) t	
			Base.Result.t_sexp_grammar : 'ok Sexplib0.Sexp_grammar.t -> 'err Sexplib0.Sexp_grammar.t -> ('ok, 'err) t Sexplib0.Sexp_grammar.t	
	CCResult.to_err : ('a, 'b) t -> ('a, 'b) error		Base.Result.to_either : ('ok, 'err) t -> ('ok, 'err) Base__.Either0.t	
	CCResult.to_iter : ('a, 'b) t -> 'a iter			
Result.to_list : ('a, 'e) result -> 'a list		BatResult.to_list : ('a, 'e) t -> 'a list		
Result.to_option : ('a, 'e) result -> 'a option	CCResult.to_opt : ('a, 'b) t -> 'a option	BatResult.to_option : ('a, 'b) t -> 'a option		
Result.to_seq : ('a, 'e) result -> 'a Seq.t	CCResult.to_seq : ('a, 'b) t -> 'a Seq.t	BatResult.to_seq : ('a, 'e) t -> 'a BatSeq.t		
			Base.Result.try_with : (unit -> 'a) -> ('a, exn) t	
Result.value : ('a, 'e) result -> default:'a -> 'a		BatResult.value : ('a, 'e) t -> default:'a -> 'a		
	CCResult.wrap1 : ('a -> 'b) -> 'a -> ('b, exn) t			
	CCResult.wrap2 : ('a -> 'b -> 'c) -> 'a -> 'b -> ('c, exn) t			
	CCResult.wrap3 : ('a -> 'b -> 'c -> 'd) -> 'a -> 'b -> 'c -> ('d,			

Stdlib	Containers	Batteries	Base	F#
	exn) t			
	CCResult.(<$>) : ('a -> 'b) -> ('a, 'err) t -> ('b, 'err) t			
	CCResult.(<*>) : ('a -> 'b, 'err) t -> ('a, 'err) t -> ('b, 'err) t			
	CCResult.(>>=) : ('a, 'err) t -> ('a -> ('b, 'err) t) -> ('b, 'err) t	BatResult.Infix.(>>=) : ('a, 'e) t -> ('a -> ('c, 'e) t) -> ('c, 'e) t	Base.Result.(>>=) : ('a, 'e) t -> ('a -> ('b, 'e) t) -> ('b, 'e) t	
	CCResult.(>|=) : ('a, 'err) t -> ('a -> 'b) -> ('b, 'err) t		Base.Result.(>>|) : ('a, 'e) t -> ('a -> 'b) -> ('b, 'e) t	
	CCResult.(and*) : ('a, 'e) t -> ('b, 'e) t -> ('a * 'b, 'e) t			
	CCResult.(and+) : ('a, 'e) t -> ('b, 'e) t -> ('a * 'b, 'e) t			
	CCResult.(let*) : ('a, 'e) t -> ('a -> ('b, 'e) t) -> ('b, 'e) t			
	CCResult.(let+) : ('a, 'e) t -> ('a -> 'b) -> ('b, 'e) t			
			Base.Sequence.all : 'a t list -> 'a list t	
				Seq.allPairs : (seq<'a> -> seq<'b> -> seq<'a * 'b>)
			Base.Sequence.all_unit : unit t list -> unit t	
Seq.append : 'a t -> 'a t -> 'a t	CCSeq.append : 'a t -> 'a t -> 'a t	BatSeq.append : 'a t -> 'a t -> 'a t	Base.Sequence.append : 'a t -> 'a t -> 'a t	Seq.append : (seq<'a> -> seq<'a> -> seq<'a>)
		BatSeq.assoc : 'a -> ('a * 'b) t -> 'b option		
		BatSeq.at : 'a t -> int -> 'a		
				Seq.average : ???
				Seq.averageBy : ???
			Base.Sequence.bind : 'a t -> f:('a -> 'b t) -> 'b t	
			Base.Sequence.bounded_length : 'a t -> at_most:int -> [`Greater | `Is of int]	
				Seq.cache : (seq<'a> -> seq<'a>)
			Base.Sequence.cartesian_product : 'a t -> 'b t -> ('a * 'b) t	
				Seq.cast : (System.Collections.IEnumerable -> seq<'a>)
				Seq.choose : (('a -> 'b option) -> seq<'a> -> seq<'b>)
			Base.Sequence.chunks_exn : 'a t -> int -> 'a list t	Seq.chunkBySize : (int -> seq<'a> -> seq<'a []>)
				Seq.collect : (('a -> #seq<'c>) -> seq<'a> -> seq<'c>)
		BatSeq.combine : 'a t -> 'b t -> ('a * 'b) t		
Seq.compare : ('a -> 'b -> int) -> 'a t -> 'b t -> int	CCSeq.compare : 'a ord -> 'a t ord	BatSeq.compare : ('a -> 'b -> int) -> 'a t -> 'b t -> int	Base.Sequence.compare : 'a Base__Ppx_compare_lib.compare -> 'a t Base__Ppx_compare_lib.compare	Seq.compareWith : (('a -> 'a -> int) -> seq<'a> -> seq<'a> -> int)
Seq.concat : 'a t t -> 'a t		BatSeq.concat : 'a t t -> 'a t	Base.Sequence.concat : 'a t t -> 'a t	Seq.concat : (seq<#seq<'b>> -> seq<'b>)
Seq.concat_map : ('a -> 'b t) -> 'a t -> 'b t		BatSeq.concat_map : ('a -> 'b t) -> 'a t -> 'b t	Base.Sequence.concat_map : 'a t -> f:('a -> 'b t) -> 'b t	
			Base.Sequence.concat_mapi : 'a t -> f:(int -> 'a -> 'b t) -> 'b t	
Seq.cons : 'a -> 'a t -> 'a t	CCSeq.cons : 'a -> 'a t -> 'a t	BatSeq.cons : 'a -> 'a t -> 'a t		
				Seq.contains : ('a -> seq<'a> -> bool) when 'a : equality
			Base.Sequence.count : 'a t -> f:('a -> bool) -> int	
			Base.Sequence.counti : 'a t -> f:(int -> 'a -> bool) -> int	

Stdlib	Containers	Batteries	Base	F#
				Seq.countBy : (('a -> 'b) -> seq<'a> -> seq<'b * int>) when 'b : equality
Seq.cycle : 'a t -> 'a t	CCSeq.cycle : 'a t -> 'a t	BatSeq.cycle : 'a t -> 'a t		
			Base.Sequence.cycle_list_exn : 'a list -> 'a t	
				Seq.delay : ((unit -> seq<'a>) -> seq<'a>)
			Base.Sequence.delayed_fold : 'a t -> init:'s -> f:('s -> 'a -> k:('s -> 'r) -> 'r) -> finish:('s -> 'r) -> 'r	
				Seq.distinct : (seq<'a> -> seq<'a>) when 'a : equality
				Seq.distinctBy : (('a -> 'b) -> seq<'a> -> seq<'a>) when 'b : equality
Seq.drop : int -> 'a t -> 'a t	CCSeq.drop : int -> 'a t -> 'a t	BatSeq.drop : int -> 'a t -> 'a t	Base.Sequence.drop : 'a t -> int -> 'a t	
			Base.Sequence.drop_eagerly : 'a t -> int -> 'a t	
Seq.drop_while : ('a -> bool) -> 'a t -> 'a t	CCSeq.drop_while : ('a -> bool) -> 'a t -> 'a t	BatSeq.drop_while : ('a -> bool) -> 'a t -> 'a t	Base.Sequence.drop_while : 'a t -> f:('a -> bool) -> 'a t	
			Base.Sequence.drop_while_option : 'a t -> f:('a -> bool) -> ('a * 'a t) option	
Seq.empty : 'a t	CCSeq.empty : 'a t	BatSeq.empty : 'a t	Base.Sequence.empty : 'a t	Seq.empty : seq<'a>
		BatSeq.enum : 'a t -> 'a BatEnum.t		
Seq.equal : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	CCSeq.equal : 'a equal -> 'a t equal	BatSeq.equal : ?eq:('a -> 'a -> bool) -> 'a t -> 'a t -> bool	Base.Sequence.equal : 'a Base__Ppx_compare_lib.equal -> 'a t Base__Ppx_compare_lib.equal	
		BatSeq.equal_stdlib : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool		
				Seq.exactlyOne : (seq<'a> -> 'a)
				Seq.except : (seq<'a> -> seq<'a> -> seq<'a>) when 'a : equality
Seq.exists : ('a -> bool) -> 'a t -> bool	CCSeq.exists : ('a -> bool) -> 'a t -> bool	BatSeq.exists : ('a -> bool) -> 'a t -> bool	Base.Sequence.exists : 'a t -> f:('a -> bool) -> bool	Seq.exists : (('a -> bool) -> seq<'a> -> bool)
			Base.Sequence.existsi : 'a t -> f:(int -> 'a -> bool) -> bool	
Seq.exists2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	CCSeq.exists2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	BatSeq.exists2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool		Seq.exists2 : (('a -> 'b -> bool) -> seq<'a> -> seq<'b> -> bool)
	CCSeq.fair_app : ('a -> 'b) t -> 'a t -> 'b t			
	CCSeq.fair_flat_map : ('a -> 'b t) -> 'a t -> 'b t			
Seq.filter : ('a -> bool) -> 'a t -> 'a t	CCSeq.filter : ('a -> bool) -> 'a t -> 'a t	BatSeq.filter : ('a -> bool) -> 'a t -> 'a t	Base.Sequence.filter : 'a t -> f:('a -> bool) -> 'a t	Seq.filter : (('a -> bool) -> seq<'a> -> seq<'a>)
Seq.filter_map : ('a -> 'b option) -> 'a t -> 'b t	CCSeq.filter_map : ('a -> 'b option) -> 'a t -> 'b t	BatSeq.filter_map : ('a -> 'b option) -> 'a t -> 'b t	Base.Sequence.filter_map : 'a t -> f:('a -> 'b option) -> 'b t	
			Base.Sequence.filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t	
			Base.Sequence.filter_opt : 'a option t -> 'a t	
			Base.Sequence.filteri : 'a t -> f:(int -> 'a -> bool) -> 'a t	
Seq.find : ('a -> bool) -> 'a t -> 'a option		BatSeq.find : ('a -> bool) -> 'a t -> 'a option	Base.Sequence.find : 'a t -> f:('a -> bool) -> 'a option	Seq.find : (('a -> bool) -> seq<'a> -> 'a)
			Base.Sequence.find_consecutive_duplicate : 'a t -> equal:('a -> 'a -> bool) -> ('a * 'a) option	
			Base.Sequence.find_exn : 'a t -> f:('a -> bool) -> 'a	
Seq.find_map : ('a -> 'b option) -> 'a t -> 'b option		BatSeq.find_map : ('a -> 'b option) -> 'a t -> 'b option	Base.Sequence.find_map : 'a t -> f:('a -> 'b option) -> 'b option	
			Base.Sequence.find_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b option	
				Seq.findBack : (('a -> bool) -> seq<'a> -> 'a)
			Base.Sequence.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option	

Stdlib	Containers	Batteries	Base	F#
				Seq.findIndex : (('a -> bool) -> seq<'a> -> int)
				Seq.findIndexBack : (('a -> bool) -> seq<'a> -> int)
		BatSeq.first : 'a t -> 'a		
Seq.flat_map : ('a -> 'b t) -> 'a t -> 'b t	CCSeq.flat_map : ('a -> 'b t) -> 'a t -> 'b t	BatSeq.flat_map : ('a -> 'b t) -> 'a t -> 'b t		
	CCSeq.flatten : 'a t t -> 'a t	BatSeq.flatten : 'a t t -> 'a t		
	CCSeq.fmap : ('a -> 'b option) -> 'a t -> 'b t			
Seq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a	CCSeq.fold : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a	BatSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a	Base.Sequence.fold : 'a t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum	Seq.fold : (('a -> 'b -> 'a) -> 'a -> seq<'b> -> 'a)
Seq.fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b t -> 'c t -> 'a	CCSeq.fold2 : ('acc -> 'a -> 'b -> 'acc) -> 'acc -> 'a t -> 'b t -> 'acc	BatSeq.fold_left2 : ('a -> 'b -> 'c -> 'a) -> 'a -> 'b t -> 'c t -> 'a		Seq.fold2 : (('a -> 'b -> 'c -> 'a) -> 'a -> seq<'b> -> seq<'c> -> 'a)
	CCSeq.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b t -> 'a			
Seq.fold_lefti : ('b -> int -> 'a -> 'b) -> 'b -> 'a t -> 'b		BatSeq.fold_lefti : ('b -> int -> 'a -> 'b) -> 'b -> 'a t -> 'b		
			Base.Sequence.fold_m : bind:('acc_m -> f:('acc -> 'acc_m) -> 'acc_m) -> return:('acc -> 'acc_m) -> 'elt t -> init:'acc -> f:('acc -> 'elt -> 'acc_m) -> 'acc_m	
			Base.Sequence.fold_result : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t	
		BatSeq.fold_right : ('a -> 'b -> 'b) -> 'a t -> 'b -> 'b		Seq.foldBack : (('a -> 'b -> 'b) -> seq<'a> -> 'b -> 'b)
				Seq.foldBack2 : (('a -> 'b -> 'c -> 'c) -> seq<'a> -> seq<'b> -> 'c -> 'c)
			Base.Sequence.fold_until : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final	
			Base.Sequence.foldi : ('a t, 'a, 'b) Base__Indexed_container_intf.foldi	
			Base.Sequence.folding_map : 'a t -> init:'b -> f:('b -> 'a -> 'b * 'c) -> 'c t	
			Base.Sequence.folding_mapi : 'a t -> init:'b -> f:(int -> 'b -> 'a -> 'b * 'c) -> 'c t	
Seq.for_all : ('a -> bool) -> 'a t -> bool	CCSeq.for_all : ('a -> bool) -> 'a t -> bool	BatSeq.for_all : ('a -> bool) -> 'a t -> bool	Base.Sequence.for_all : 'a t -> f:('a -> bool) -> bool	Seq.forall : (('a -> bool) -> seq<'a> -> bool)
			Base.Sequence.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool	
Seq.for_all2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	CCSeq.for_all2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool	BatSeq.for_all2 : ('a -> 'b -> bool) -> 'a t -> 'b t -> bool		Seq.forall2 : (('a -> 'b -> bool) -> seq<'a> -> seq<'b> -> bool)
			Base.Sequence.force_eagerly : 'a t -> 'a t	
Seq.forever : (unit -> 'a) -> 'a t		BatSeq.forever : (unit -> 'a) -> 'a t		
Seq.group : ('a -> 'a -> bool) -> 'a t -> 'a t t	CCSeq.group : 'a equal -> 'a t -> 'a t t	BatSeq.group : ('a -> 'a -> bool) -> 'a t -> 'a t t	Base.Sequence.group : 'a t -> break:('a -> 'a -> bool) -> 'a list t	
				Seq.groupBy : (('a -> 'b) -> seq<'a> -> seq<'b * seq<'a>>) when 'b : equality
	CCSeq.head : 'a t -> 'a option		Base.Sequence.hd : 'a t -> 'a option	
	CCSeq.head_exn : 'a t -> 'a	BatSeq.hd : 'a t -> 'a	Base.Sequence.hd_exn : 'a t -> 'a	Seq.head : (seq<'a> -> 'a)
			Base.Sequence.ignore_m : 'a t -> unit t	
				Seq.indexed : (seq<'a> -> seq<int * 'a>)
Seq.init : int -> (int -> 'a) -> 'a t		BatSeq.init : int -> (int -> 'a) -> 'a t	Base.Sequence.init : int -> f:(int -> 'a) -> 'a t	Seq.init : (int -> (int -> 'a) -> seq<'a>)
				Seq.initInfinite : ((int -> 'a) -> seq<'a>)
				Seq.insertAt : ???
				Seq.insertManyAt : ???
Seq.interleave : 'a t -> 'a t -> 'a t	CCSeq.interleave : 'a t -> 'a t -> 'a t	BatSeq.interleave : 'a t -> 'a t -> 'a t	Base.Sequence.interleave : 'a t t -> 'a t	

Stdlib	Containers	Batteries	Base	F#
			Base.Sequence.interleaved_cartesian_product : 'a t -> 'b t -> ('a * 'b) t	
			Base.Sequence.intersperse : 'a t -> sep:'a -> 'a t	
Seq.ints : int -> int t		BatSeq.ints : int -> int t		
Seq.is_empty : 'a t -> bool	CCSeq.is_empty : 'a t -> bool	BatSeq.is_empty : 'a t -> bool	Base.Sequence.is_empty : 'a t -> bool	Seq.isEmpty : (seq<'a> -> bool)
				Seq.item : (int -> seq<'a> -> 'a)
Seq.iter : ('a -> unit) -> 'a t -> unit	CCSeq.iter : ('a -> unit) -> 'a t -> unit	BatSeq.iter : ('a -> unit) -> 'a t -> unit	Base.Sequence.iter : 'a t -> f:('a -> unit) -> unit	Seq.iter : (('a -> unit) -> seq<'a> -> unit)
			Base.Sequence.iter_m : bind:('unit_m -> f:(unit -> 'unit_m) -> 'unit_m) -> return:(unit -> 'unit_m) -> 'elt t -> f:('elt -> 'unit_m) -> 'unit_m	
Seq.iter2 : ('a -> 'b -> unit) -> 'a t -> 'b t -> unit	CCSeq.iter2 : ('a -> 'b -> unit) -> 'a t -> 'b t -> unit	BatSeq.iter2 : ('a -> 'b -> unit) -> 'a t -> 'b t -> unit		Seq.iter2 : (('a -> 'b -> unit) -> seq<'a> -> seq<'b> -> unit)
Seq.iterate : ('a -> 'a) -> 'a -> 'a t		BatSeq.iterate : ('a -> 'a) -> 'a -> 'a t		
Seq.iteri : (int -> 'a -> unit) -> 'a t -> unit	CCSeq.iteri : (int -> 'a -> unit) -> 'a t -> unit	BatSeq.iteri : (int -> 'a -> unit) -> 'a t -> unit	Base.Sequence.iteri : ('a t, 'a) Base__Indexed_container_intf.iteri	Seq.iteri : ((int -> 'a -> unit) -> seq<'a> -> unit)
				Seq.iteri2 : ((int -> 'a -> 'b -> unit) -> seq<'a> -> seq<'b> -> unit)
			Base.Sequence.join : 'a t t -> 'a t	
		BatSeq.last : 'a t -> 'a		Seq.last : (seq<'a> -> 'a)
Seq.length : 'a t -> int	CCSeq.length : 'a t -> int	BatSeq.length : 'a t -> int	Base.Sequence.length : 'a t -> int	Seq.length : (seq<'a> -> int)
			Base.Sequence.length_is_bounded_by : ?min:int -> ?max:int -> 'a t -> bool	
		BatSeq.make : int -> 'a -> 'a t		
Seq.map : ('a -> 'b) -> 'a t -> 'b t	CCSeq.map : ('a -> 'b) -> 'a t -> 'b t	BatSeq.map : ('a -> 'b) -> 'a t -> 'b t	Base.Sequence.map : 'a t -> f:('a -> 'b) -> 'b t	Seq.map : (('a -> 'b) -> seq<'a> -> seq<'b>)
Seq.map2 : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t	CCSeq.map2 : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t	BatSeq.map2 : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t		Seq.map2 : (('a -> 'b -> 'c) -> seq<'a> -> seq<'b> -> seq<'c>)
				Seq.map3 : (('a -> 'b -> 'c -> 'd) -> seq<'a> -> seq<'b> -> seq<'c> -> seq<'d>)
				Seq.mapFold : (('a -> 'b -> 'c * 'a) -> 'a -> seq<'b> -> seq<'c> * 'a)
				Seq.mapFoldBack : (('a -> 'b -> 'c * 'b) -> seq<'a> -> 'b -> seq<'c> * 'b)
Seq.map_product : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t		BatSeq.map_product : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t		
Seq.mapi : (int -> 'a -> 'b) -> 'a t -> 'b t	CCSeq.mapi : (int -> 'a -> 'b) -> 'a t -> 'b t	BatSeq.mapi : (int -> 'a -> 'b) -> 'a t -> 'b t	Base.Sequence.mapi : 'a t -> f:(int -> 'a -> 'b) -> 'b t	Seq.mapi : ((int -> 'a -> 'b) -> seq<'a> -> seq<'b>)
				Seq.mapi2 : ((int -> 'a -> 'b -> 'c) -> seq<'a> -> seq<'b> -> seq<'c>)
		BatSeq.max : 'a t -> 'a		
			Base.Sequence.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	Seq.max : (seq<'a> -> 'a) when 'a : comparison
				Seq.maxBy : (('a -> 'b) -> seq<'a> -> 'a) when 'b : comparison
		BatSeq.mem : 'a -> 'a t -> bool	Base.Sequence.mem : 'a t -> 'a -> equal:('a -> 'a -> bool) -> bool	
Seq.memoize : 'a t -> 'a t	CCSeq.memoize : 'a t -> 'a t	BatSeq.memoize : 'a t -> 'a t	Base.Sequence.memoize : 'a t -> 'a t	
	CCSeq.merge : 'a ord -> 'a t -> 'a t -> 'a t		Base.Sequence.merge : 'a t -> 'a t -> compare:('a -> 'a -> int) -> 'a t	
		BatSeq.min : 'a t -> 'a	Base.Sequence.merge_deduped_and_sorted : 'a t -> 'a t -> compare:('a -> 'a -> int) -> 'a t	
			Base.Sequence.merge_sorted : 'a t -> 'a t -> compare:('a -> 'a -> int) -> 'a t	
			Base.Sequence.merge_with_duplicates : 'a t -> 'b t -> compare:('a -> 'b -> int) ->	

Stdlib	Containers	Batteries	Base	F#
			('a, 'b) Merge_with_duplicates_element.t t	
			Base.Sequence.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option	Seq.min : (seq<'a> -> 'a) when 'a : comparison
				Seq.minBy : (('a -> 'b) -> seq<'a> -> 'a) when 'b : comparison
			Base.Sequence.next : 'a t -> ('a * 'a t) option	
	CCSeq.nil : 'a t	BatSeq.nil : 'a t		
			Base.Sequence.nth : 'a t -> int -> 'a option	
			Base.Sequence.nth_exn : 'a t -> int -> 'a	Seq.nth : (int -> seq<'a> -> 'a)
	CCSeq.of_array : 'a array -> 'a t			Seq.ofArray : ('a [] -> seq<'a>)
Seq.of_dispenser : (unit -> 'a option) -> 'a t		BatSeq.of_dispenser : (unit -> 'a option) -> 'a t		
	CCSeq.of_gen : 'a gen -> 'a t			
			Base.Sequence.of_lazy : 'a t Base__.Lazy.t -> 'a t	
	CCSeq.of_list : 'a list -> 'a t	BatSeq.of_list : 'a list -> 'a t	Base.Sequence.of_list : 'a list -> 'a t	Seq.ofList : ('a list -> seq<'a>)
			Base.Sequence.of_seq : 'a Base__.Import.Caml.Seq.t -> 'a t	
	CCSeq.of_string : string -> char t	BatSeq.of_string : ?first:string -> ?last:string -> ?sep:string -> (string -> 'a) -> string -> 'a t		
Seq.once : 'a t -> 'a t		BatSeq.once : 'a t -> 'a t		
				Seq.pairwise : (seq<'a> -> seq<'a * 'a>)
Seq.partition : ('a -> bool) -> 'a t -> 'a t * 'a t		BatSeq.partition : ('a -> bool) -> 'a t -> 'a t * 'a t		
Seq.partition_map : ('a -> ('b, 'c) Either.t) -> 'a t -> 'b t * 'c t		BatSeq.partition_map : ('a -> ('b, 'c) Either.t) -> 'a t -> 'b t * 'c t		
				Seq.permute : ((int -> int) -> seq<'a> -> seq<'a>)
				Seq.pick : (('a -> 'b option) -> seq<'a> -> 'b)
	CCSeq.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> 'a printer -> 'a t printer			
		BatSeq.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'b -> unit) -> 'a BatInnerIO.output -> 'b t -> unit		
Seq.product : 'a t -> 'b t -> ('a * 'b) t	CCSeq.product : 'a t -> 'b t -> ('a * 'b) t	BatSeq.product : 'a t -> 'b t -> ('a * 'b) t		
	CCSeq.product_with : ('a -> 'b -> 'c) -> 'a t -> 'b t -> 'c t			
	CCSeq.pure : 'a -> 'a t			
	CCSeq.range : int -> int -> int t		Base.Sequence.range : ?stride:int -> ?start:[`exclusive | `inclusive] -> ?stop:[`exclusive | `inclusive] -> int -> int -> int t	
				Seq.readonly : (seq<'a> -> seq<'a>)
			Base.Sequence.reduce : 'a t -> f:('a -> 'a -> 'a) -> 'a option	
		BatSeq.reduce : ('a -> 'a -> 'a) -> 'a t -> 'a	Base.Sequence.reduce_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a	Seq.reduce : (('a -> 'a -> 'a) -> seq<'a> -> 'a)
				Seq.reduceBack : (('a -> 'a -> 'a) -> seq<'a> -> 'a)
			Base.Sequence.remove_consecutive_duplicates : 'a t -> equal:('a -> 'a -> bool) -> 'a t	
				Seq.removeAt : ???
				Seq.removeManyAt : ???
Seq.repeat : 'a -> 'a t	CCSeq.repeat : ?n:int -> 'a -> 'a t	BatSeq.repeat : 'a -> 'a t	Base.Sequence.repeat : 'a -> 'a t	

Stdlib	Containers	Batteries	Base	F#
				Seq.replicate : (int -> 'a -> seq<'a>)
Seq.return : 'a -> 'a t	CCSeq.return : 'a -> 'a t	BatSeq.return : 'a -> 'a t	Base.Sequence.return : 'a -> 'a t	
				Seq.rev : (seq<'a> -> seq<'a>)
			Base.Sequence.round_robin : 'a t list -> 'a t	
Seq.scan : ('b -> 'a -> 'b) -> 'b -> 'a t -> 'b t		BatSeq.scan : ('b -> 'a -> 'b) -> 'b -> 'a t -> 'b t		Seq.scan : (('a -> 'b -> 'a) -> 'a -> seq<'b> -> seq<'a>)
				Seq.scanBack : (('a -> 'b -> 'b) -> seq<'a> -> 'b -> seq<'b>)
			Base.Sequence.sexp_of_t : ('a -> Sexplib0.Sexp.t) -> 'a t -> Sexplib0.Sexp.t	
			Base.Sequence.shift_left : 'a t -> int -> 'a t	
			Base.Sequence.shift_right : 'a t -> 'a -> 'a t	
			Base.Sequence.shift_right_with_list : 'a t -> 'a list -> 'a t	
	CCSeq.singleton : 'a -> 'a t		Base.Sequence.singleton : 'a -> 'a t	Seq.singleton : ('a -> seq<'a>)
				Seq.skip : (int -> seq<'a> -> seq<'a>)
				Seq.skipWhile : (('a -> bool) -> seq<'a> -> seq<'a>)
	CCSeq.sort : cmp:'a ord -> 'a t -> 'a t			Seq.sort : (seq<'a> -> seq<'a>) when 'a : comparison
	CCSeq.sort_uniq : cmp:'a ord -> 'a t -> 'a t			
				Seq.sortBy : (('a -> 'b) -> seq<'a> -> seq<'a>) when 'b : comparison
				Seq.sortByDescending : (('a -> 'b) -> seq<'a> -> seq<'a>) when 'b : comparison
				Seq.sortDescending : (seq<'a> -> seq<'a>) when 'a : comparison
				Seq.sortWith : (('a -> 'a -> int) -> seq<'a> -> seq<'a>)
Seq.sorted_merge : ('a -> 'a -> int) -> 'a t -> 'a t -> 'a t		BatSeq.sorted_merge : ('a -> 'a -> int) -> 'a t -> 'a t -> 'a t		
Seq.split : ('a * 'b) t -> 'a t * 'b t		BatSeq.split : ('a * 'b) t -> 'a t * 'b t		
			Base.Sequence.split_n : 'a t -> int -> 'a list * 'a t	
				Seq.splitInto : (int -> seq<'a> -> seq<'a []>)
			Base.Sequence.sub : 'a t -> pos:int -> len:int -> 'a t	
			Base.Sequence.sum : (module Base__Container_intf.Summable with type t = 'sum) -> 'a t -> f:('a -> 'sum) -> 'sum	Seq.sum : ???
				Seq.sumBy : ???
	CCSeq.tail : 'a t -> 'a t option		Base.Sequence.tl : 'a t -> 'a t option	
	CCSeq.tail_exn : 'a t -> 'a t	BatSeq.tl : 'a t -> 'a t	Base.Sequence.tl_eagerly_exn : 'a t -> 'a t	Seq.tail : (seq<'a> -> seq<'a>)
Seq.take : int -> 'a t -> 'a t	CCSeq.take : int -> 'a t -> 'a t	BatSeq.take : int -> 'a t -> 'a t	Base.Sequence.take : 'a t -> int -> 'a t	Seq.take : (int -> seq<'a> -> seq<'a>)
Seq.take_while : ('a -> bool) -> 'a t -> 'a t	CCSeq.take_while : ('a -> bool) -> 'a t -> 'a t	BatSeq.take_while : ('a -> bool) -> 'a t -> 'a t	Base.Sequence.take_while : 'a t -> f:('a -> bool) -> 'a t	Seq.takeWhile : (('a -> bool) -> seq<'a> -> seq<'a>)
	CCSeq.to_array : 'a t -> 'a array		Base.Sequence.to_array : 'a t -> 'a array	Seq.toArray : (seq<'a> -> 'a [])
		BatSeq.to_buffer : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> Buffer.t -> (unit -> 'a node) -> unit		
Seq.to_dispenser : 'a t -> unit -> 'a option		BatSeq.to_dispenser : 'a t -> unit -> 'a option		
	CCSeq.to_gen : 'a t -> 'a gen			

Stdlib	Containers	Batteries	Base	F#
	CCSeq.to_iter : 'a t -> 'a iter			
	CCSeq.to_list : 'a t -> 'a list		Base.Sequence.to_list : 'a t -> 'a list	Seq.toList : (seq<'a> -> 'a list)
	CCSeq.to_rev_list : 'a t -> 'a list		Base.Sequence.to_list_rev : 'a t -> 'a list	
			Base.Sequence.to_seq : 'a t -> 'a Base__.Import.Caml.Seq.t	
		BatSeq.to_string : ?first:string -> ?last:string -> ?sep:string -> ('a -> string) -> 'a t -> string		
Seq.transpose : 'a t t -> 'a t t		BatSeq.transpose : 'a t t -> 'a t t		Seq.transpose : (seq<#seq<'b>> -> seq<seq<'b>>)
				Seq.truncate : (int -> seq<'a> -> seq<'a>)
				Seq.tryExactlyOne : (seq<'a> -> 'a option)
				Seq.tryFind : (('a -> bool) -> seq<'a> -> 'a option)
				Seq.tryFindBack : (('a -> bool) -> seq<'a> -> 'a option)
				Seq.tryFindIndex : (('a -> bool) -> seq<'a> -> int option)
				Seq.tryFindIndexBack : (('a -> bool) -> seq<'a> -> int option)
				Seq.tryHead : (seq<'a> -> 'a option)
				Seq.tryItem : (int -> seq<'a> -> 'a option)
				Seq.tryLast : (seq<'a> -> 'a option)
				Seq.tryPick : (('a -> 'b option) -> seq<'a> -> 'b option)
Seq.uncons : 'a t -> ('a * 'a t) option		BatSeq.uncons : 'a t -> ('a * 'a t) option		
Seq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a t	CCSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a t	BatSeq.unfold : ('b -> ('a * 'b) option) -> 'b -> 'a t	Base.Sequence.unfold : init:'s -> f:('s -> ('a * 's) option) -> 'a t	Seq.unfold : (('a -> ('b * 'a) option) -> 'a -> seq<'b>)
			Base.Sequence.unfold_step : init:'s -> f:('s -> ('a, 's) Step.t) -> 'a t	
			Base.Sequence.unfold_with : 'a t -> init:'s -> f:('s -> 'a -> ('b, 's) Step.t) -> 'b t	
			Base.Sequence.unfold_with_and_finish : 'a t -> init:'s_a -> running_step:('s_a -> 'a -> ('b, 's_a) Step.t) -> inner_finished:('s_a -> 's_b) -> finishing_step:('s_b -> ('b, 's_b) Step.t) -> 'b t	
	CCSeq.uniq : 'a equal -> 'a t -> 'a t			
Seq.unzip : ('a * 'b) t -> 'a t * 'b t	CCSeq.unzip : ('a * 'b) t -> 'a t * 'b t	BatSeq.unzip : ('a * 'b) t -> 'a t * 'b t		
				Seq.updateAt : ???
				Seq.where : (('a -> bool) -> seq<'a> -> seq<'a>)
				Seq.windowed : (int -> seq<'a> -> seq<'a []>)
Seq.zip : 'a t -> 'b t -> ('a * 'b) t	CCSeq.zip : 'a t -> 'b t -> ('a * 'b) t	BatSeq.zip : 'a t -> 'b t -> ('a * 'b) t	Base.Sequence.zip : 'a t -> 'b t -> ('a * 'b) t	Seq.zip : (seq<'a> -> seq<'b> -> seq<'a * 'b>)
				Seq.zip3 : (seq<'a> -> seq<'b> -> seq<'c> -> seq<'a * 'b * 'c>)
			Base.Sequence.zip_full : 'a t -> 'b t -> [`Both of 'a * 'b | `Left of 'a | `Right of 'b] t	
	CCSeq.zip_i : 'a t -> (int * 'a) t			
	CCSeq.(--) : int -> int -> int t	BatSeq.(--) : int -> int -> int t		
		BatSeq.(---) : int -> int -> int t		
		BatSeq.(--.) : float * float -> float -> float t		
	CCSeq.(--^) : int -> int -> int t	BatSeq.(--^) : int -> int -> int t		

Stdlib	Containers	Batteries	Base	F#
		BatSeq.(--~) : char -> char -> char t		
		BatSeq.(//) : 'a t -> ('a -> bool) -> 'a t		
		BatSeq.(//@) : 'a t -> ('a -> 'b option) -> 'b t		
		BatSeq.(/@) : 'a t -> ('a -> 'b) -> 'b t		
		BatSeq.(@/) : ('a -> 'b) -> 'a t -> 'b t		
		BatSeq.(@//) : ('a -> 'b option) -> 'a t -> 'b t		
	CCSeq.(<*>) : ('a -> 'b) t -> 'a t -> 'b t			
	CCSeq.(<.>) : ('a -> 'b) t -> 'a t -> 'b t			
	CCSeq.(>>-) : 'a t -> ('a -> 'b t) -> 'b t			
	CCSeq.(>>=) : 'a t -> ('a -> 'b t) -> 'b t		Base.Sequence.(>>=) : 'a t -> ('a -> 'b t) -> 'b t	
	CCSeq.(>|=) : 'a t -> ('a -> 'b) -> 'b t		Base.Sequence.(>>|) : 'a t -> ('a -> 'b) -> 'b t	
Set.add : elt -> t -> t	CCSet.add : elt -> t -> t	BatSet.add : 'a -> 'a t -> 'a t	Base.Set.add : ('a, 'cmp) t -> 'a -> ('a, 'cmp) t	Set.add : ('a -> Set<'a> -> Set<'a>) when 'a : comparison
	CCSet.add_iter : t -> elt iter -> t			
	CCSet.add_list : t -> elt list -> t			
Set.add_seq : elt Seq.t -> t -> t	CCSet.add_seq : elt Seq.t -> t -> t	BatSet.add_seq : 'a BatSeq.t -> 'a t -> 'a t		
		BatSet.any : 'a t -> 'a		
			Base.Set.are_disjoint : ('a, 'cmp) t -> ('a, 'cmp) t -> bool	
		BatSet.at_rank_exn : int -> 'a t -> 'a		
		BatSet.backwards : 'a t -> 'a BatEnum.t		
			Base.Set.binary_search : ('a, 'cmp) t -> compare:('a -> 'key -> int) -> [`First_equal_to | `First_greater_than_or_equal_to | `First_strictly_greater_than | `Last_equal_to | `Last_less_than_or_equal_to | `Last_strictly_less_than] -> 'key -> 'a option	
			Base.Set.binary_search_segmented : ('a, 'cmp) t -> segment_of:('a -> [`Left | `Right]) -> [`First_on_right | `Last_on_left] -> 'a option	
Set.cardinal : t -> int	CCSet.cardinal : t -> int	BatSet.cardinal : 'a t -> int		
		BatSet.cartesian_product : 'a t -> 'b t -> ('a * 'b) t		
Set.choose : t -> elt	CCSet.choose : t -> elt	BatSet.choose : 'a t -> 'a	Base.Set.choose_exn : ('a, 'b) t -> 'a	
Set.choose_opt : t -> elt option	CCSet.choose_opt : t -> elt option	BatSet.choose_opt : 'a t -> 'a option	Base.Set.choose : ('a, 'b) t -> 'a option	
			Base.Set.comparator : ('a, 'cmp) t -> ('a, 'cmp) Base__.Comparator.t	
			Base.Set.comparator_s : ('a, 'cmp) t -> ('a, 'cmp) Base__.Comparator.Module.t	
Set.compare : t -> t -> int	CCSet.compare : t -> t -> int	BatSet.compare : 'a t -> 'a t -> int	Base.Set.compare : 'a Base__Ppx_compare_lib.compare -> 'b Base__Ppx_compare_lib.compare -> ('a, 'b) t Base__Ppx_compare_lib.compare	
			Base.Set.compare_direct : ('a, 'cmp) t -> ('a, 'cmp) t -> int	
			Base.Set.compare_m__t : (module Compare_m) -> ('elt, 'cmp) t -> ('elt, 'cmp) t -> int	
				Set.contains : ('a -> Set<'a> -> bool) when 'a : comparison
			Base.Set.count : ('a, 'b) t -> f:('a -> bool) -> int	Set.count : (Set<'a> -> int) when 'a : comparison
Set.diff : t -> t -> t	CCSet.diff : t -> t -> t	BatSet.diff : 'a t -> 'a t -> 'a t	Base.Set.diff : ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'cmp) t	Set.difference : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
Set.disjoint : t -> t -> bool	CCSet.disjoint : t -> t -> bool	BatSet.disjoint : 'a t -> 'a t -> bool		
Set.elements : t -> elt list	CCSet.elements : t -> elt list	BatSet.elements : 'a t -> 'a list	Base.Set.elements : ('a, 'b) t -> 'a list	

Stdlib	Containers	Batteries	Base	F#
Set.empty : t	CCSet.empty : t	BatSet.empty : 'a t	Base.Set.empty : ('a, 'cmp) Base__.Comparator.Module.t -> ('a, 'cmp) t	Set.empty : Set<'a> when 'a : comparison
		BatSet.enum : 'a t -> 'a BatEnum.t		
Set.equal : t -> t -> bool	CCSet.equal : t -> t -> bool	BatSet.equal : 'a t -> 'a t -> bool	Base.Set.equal : ('a, 'cmp) t -> ('a, 'cmp) t -> bool	
			Base.Set.equal_m__t : (module Equal_m) -> ('elt, 'cmp) t -> ('elt, 'cmp) t -> bool	
Set.exists : (elt -> bool) -> t -> bool	CCSet.exists : (elt -> bool) -> t -> bool	BatSet.exists : ('a -> bool) -> 'a t -> bool	Base.Set.exists : ('a, 'b) t -> f:('a -> bool) -> bool	Set.exists : (('a -> bool) -> Set<'a> -> bool) when 'a : comparison
Set.filter : (elt -> bool) -> t -> t	CCSet.filter : (elt -> bool) -> t -> t	BatSet.filter : ('a -> bool) -> 'a t -> 'a t	Base.Set.filter : ('a, 'cmp) t -> f:('a -> bool) -> ('a, 'cmp) t	Set.filter : (('a -> bool) -> Set<'a> -> Set<'a>) when 'a : comparison
Set.filter_map : (elt -> elt option) -> t -> t	CCSet.filter_map : (elt -> elt option) -> t -> t	BatSet.filter_map : ('a -> 'b option) -> 'a t -> 'b t	Base.Set.filter_map : ('b, 'cmp) Base__.Comparator.Module.t -> ('a, 'c) t -> f:('a -> 'b option) -> ('b, 'cmp) t	
		BatSet.filter_map_endo : ('a -> 'a option) -> 'a t -> 'a t		
Set.find : elt -> t -> elt	CCSet.find : elt -> t -> elt	BatSet.find : 'a -> 'a t -> 'a		
Set.find_first : (elt -> bool) -> t -> elt	CCSet.find_first : (elt -> bool) -> t -> elt	BatSet.find_first : ('a -> bool) -> 'a t -> 'a	Base.Set.find_exn : ('a, 'b) t -> f:('a -> bool) -> 'a	
Set.find_first_opt : (elt -> bool) -> t -> elt option	CCSet.find_first_opt : (elt -> bool) -> t -> elt option	BatSet.find_first_opt : ('a -> bool) -> 'a t -> 'a option	Base.Set.find : ('a, 'b) t -> f:('a -> bool) -> 'a option	
Set.find_last : (elt -> bool) -> t -> elt	CCSet.find_last : (elt -> bool) -> t -> elt	BatSet.find_last : ('a -> bool) -> 'a t -> 'a		
Set.find_last_opt : (elt -> bool) -> t -> elt option	CCSet.find_last_opt : (elt -> bool) -> t -> elt option	BatSet.find_last_opt : ('a -> bool) -> 'a t -> 'a option		
			Base.Set.find_map : ('a, 'c) t -> f:('a -> 'b option) -> 'b option	
Set.find_opt : elt -> t -> elt option	CCSet.find_opt : elt -> t -> elt option	BatSet.find_opt : 'a -> 'a t -> 'a option		
Set.fold : (elt -> 'a -> 'a) -> t -> 'a -> 'a	CCSet.fold : (elt -> 'a -> 'a) -> t -> 'a -> 'a	BatSet.fold : ('a -> 'b -> 'b) -> 'a t -> 'b -> 'b	Base.Set.fold : ('a, 'b) t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum	Set.fold : (('a -> 'b -> 'a) -> 'a -> Set<'b> -> 'a) when 'b : comparison
			Base.Set.fold_result : ('a, 'b) t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t	
			Base.Set.fold_right : ('a, 'b) t -> init:'accum -> f:('a -> 'accum -> 'accum) -> 'accum	Set.foldBack : (('a -> 'b -> 'b) -> Set<'a> -> 'b -> 'b) when 'a : comparison
			Base.Set.fold_until : ('a, 'b) t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__.Container.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final	
Set.for_all : (elt -> bool) -> t -> bool	CCSet.for_all : (elt -> bool) -> t -> bool	BatSet.for_all : ('a -> bool) -> 'a t -> bool	Base.Set.for_all : ('a, 'b) t -> f:('a -> bool) -> bool	Set.forall : (('a -> bool) -> Set<'a> -> bool) when 'a : comparison
			Base.Set.group_by : ('a, 'cmp) t -> equiv:('a -> 'a -> bool) -> ('a, 'cmp) t list	
			Base.Set.hash_fold_direct : 'a Base__.Hash.folder -> ('a, 'cmp) t Base__.Hash.folder	
			Base.Set.hash_fold_m__t : (module Hash_fold_m with type t = 'elt) -> Base__.Hash.state -> ('elt, 'a) t -> Base__.Hash.state	
			Base.Set.hash_m__t : (module Hash_fold_m with type t = 'elt) -> ('elt, 'a) t -> int	
Set.inter : t -> t -> t	CCSet.inter : t -> t -> t	BatSet.intersect : 'a t -> 'a t -> 'a t	Base.Set.inter : ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'cmp) t	Set.intersect : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
				Set.intersectMany : (seq<Set<'a>> -> Set<'a>) when 'a : comparison
			Base.Set.invariants : ('a, 'b) t -> bool	
Set.is_empty : t -> bool	CCSet.is_empty : t -> bool	BatSet.is_empty : 'a t -> bool	Base.Set.is_empty : ('a, 'b) t -> bool	Set.isEmpty : (Set<'a> -> bool) when 'a : comparison
		BatSet.is_singleton : 'a t -> bool		
				Set.isProperSubset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
				Set.isProperSuperset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison

Stdlib	Containers	Batteries	Base	F#
			Base.Set.is_subset : ('a, 'cmp) t -> of_:('a, 'cmp) t -> bool	Set.isSubset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
				Set.isSuperset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
Set.iter : (elt -> unit) -> t -> unit	CCSet.iter : (elt -> unit) -> t -> unit	BatSet.iter : ('a -> unit) -> 'a t -> unit	Base.Set.iter : ('a, 'b) t -> f:('a -> unit) -> unit	Set.iter : (('a -> unit) -> Set<'a> -> unit) when 'a : comparison
			Base.Set.iter2 : ('a, 'cmp) t -> ('a, 'cmp) t -> f:([`Both of 'a * 'a | `Left of 'a | `Right of 'a] -> unit) -> unit	
			Base.Set.length : ('a, 'b) t -> int	
			Base.Set.m__t_of_sexp : (module M_of_sexp with type comparator_witness = 'cmp and type t = 'elt) -> Base__.Sexp.t -> ('elt, 'cmp) t	
			Base.Set.m__t_sexp_grammar : (module M_sexp_grammar with type t = 'elt) -> ('elt, 'cmp) t Sexplib0.Sexp_grammar.t	
Set.map : (elt -> elt) -> t -> t	CCSet.map : (elt -> elt) -> t -> t	BatSet.map : ('a -> 'b) -> 'a t -> 'b t	Base.Set.map : ('b, 'cmp) Base__.Comparator.Module.t -> ('a, 'c) t -> f:('a -> 'b) -> ('b, 'cmp) t	Set.map : (('a -> 'b) -> Set<'a> -> Set<'b>) when 'a : comparison and 'b : comparison
		BatSet.map_endo : ('a -> 'a) -> 'a t -> 'a t		
Set.max_elt : t -> elt	CCSet.max_elt : t -> elt	BatSet.max_elt : 'a t -> 'a	Base.Set.max_elt_exn : ('a, 'b) t -> 'a	Set.maxElement : (Set<'a> -> 'a) when 'a : comparison
Set.max_elt_opt : t -> elt option	CCSet.max_elt_opt : t -> elt option	BatSet.max_elt_opt : 'a t -> 'a option	Base.Set.max_elt : ('a, 'b) t -> 'a option	
Set.mem : elt -> t -> bool	CCSet.mem : elt -> t -> bool	BatSet.mem : 'a -> 'a t -> bool	Base.Set.mem : ('a, 'b) t -> 'a -> bool	
			Base.Set.merge_to_sequence : ?order:[`Decreasing | `Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'a) Merge_to_sequence_element.t Base__.Sequence.t	
Set.min_elt : t -> elt	CCSet.min_elt : t -> elt	BatSet.min_elt : 'a t -> 'a	Base.Set.min_elt_exn : ('a, 'b) t -> 'a	Set.minElement : (Set<'a> -> 'a) when 'a : comparison
Set.min_elt_opt : t -> elt option	CCSet.min_elt_opt : t -> elt option	BatSet.min_elt_opt : 'a t -> 'a option	Base.Set.min_elt : ('a, 'b) t -> 'a option	
			Base.Set.nth : ('a, 'b) t -> int -> 'a option	
		BatSet.of_array : 'a array -> 'a t	Base.Set.of_array : ('a, 'cmp) Base__.Comparator.Module.t -> 'a array -> ('a, 'cmp) t	Set.ofArray : ('a [] -> Set<'a>) when 'a : comparison
		BatSet.of_enum : 'a BatEnum.t -> 'a t		
			Base.Set.of_increasing_iterator_unchecked : ('a, 'cmp) Base__.Comparator.Module.t -> len:int -> f:(int -> 'a) -> ('a, 'cmp) t	
	CCSet.of_iter : elt iter -> t			
Set.of_list : elt list -> t	CCSet.of_list : elt list -> t	BatSet.of_list : 'a list -> 'a t	Base.Set.of_list : ('a, 'cmp) Base__.Comparator.Module.t -> 'a list -> ('a, 'cmp) t	Set.ofList : ('a list -> Set<'a>) when 'a : comparison
Set.of_seq : elt Seq.t -> t	CCSet.of_seq : elt Seq.t -> t	BatSet.of_seq : 'a BatSeq.t -> 'a t	Base.Set.of_sequence : ('a, 'cmp) Base__.Comparator.Module.t -> 'a Base__.Sequence.t -> ('a, 'cmp) t	Set.ofSeq : (seq<'a> -> Set<'a>) when 'a : comparison
			Base.Set.of_sorted_array : ('a, 'cmp) Base__.Comparator.Module.t -> 'a array -> ('a, 'cmp) t Base__.Or_error.t	
			Base.Set.of_sorted_array_unchecked : ('a, 'cmp) Base__.Comparator.Module.t -> 'a array -> ('a, 'cmp) t	
Set.partition : (elt -> bool) -> t -> t * t	CCSet.partition : (elt -> bool) -> t -> t * t	BatSet.partition : ('a -> bool) -> 'a t -> 'a t * 'a t	Base.Set.partition_tf : ('a, 'cmp) t -> f:('a -> bool) -> ('a, 'cmp) t * ('a, 'cmp) t	Set.partition : (('a -> bool) -> Set<'a> -> Set<'a> * Set<'a>) when 'a : comparison
		BatSet.pop : 'a t -> 'a * 'a t		
		BatSet.pop_max : 'a t -> 'a * 'a t		
		BatSet.pop_min : 'a t -> 'a * 'a t		
	CCSet.pp : ?pp_start:unit printer -> ?pp_stop:unit printer -> ?pp_sep:unit printer -> elt printer -> t printer			
		BatSet.print : ?first:string -> ?last:string -> ?sep:string -> ('a BatInnerIO.output -> 'c -> unit) -> 'a		

Stdlib	Containers	Batteries	Base	F#
		BatInnerIO.output -> 'c t -> unit		
Set.remove : elt -> t -> t	CCSet.remove : elt -> t -> t	BatSet.remove : 'a -> 'a t -> 'a t	Base.Set.remove : ('a, 'cmp) t -> 'a -> ('a, 'cmp) t	Set.remove : ('a -> Set<'a> -> Set<'a>) when 'a : comparison
		BatSet.remove_exn : 'a -> 'a t -> 'a t		
			Base.Set.remove_index : ('a, 'cmp) t -> int -> ('a, 'cmp) t	
			Base.Set.sexp_of_m__t : (module Sexp_of_m with type t = 'elt) -> ('elt, 'cmp) t -> Base__.Sexp.t	
Set.singleton : elt -> t	CCSet.singleton : elt -> t	BatSet.singleton : 'a -> 'a t	Base.Set.singleton : ('a, 'cmp) Base__.Comparator.Module.t -> 'a -> ('a, 'cmp) t	Set.singleton : ('a -> Set<'a>) when 'a : comparison
Set.split : elt -> t -> t * bool * t	CCSet.split : elt -> t -> t * bool * t	BatSet.split : 'a -> 'a t -> 'a t * bool * 'a t	Base.Set.split : ('a, 'cmp) t -> 'a -> ('a, 'cmp) t * 'a option * ('a, 'cmp) t	
		BatSet.split_le : 'a -> 'a t -> 'a t * 'a t		
		BatSet.split_lt : 'a -> 'a t -> 'a t * 'a t		
		BatSet.split_opt : 'a -> 'a t -> 'a t * 'a option * 'a t		
			Base.Set.stable_dedup_list : ('a, 'b) Base__.Comparator.Module.t -> 'a list -> 'a list	
Set.subset : t -> t -> bool	CCSet.subset : t -> t -> bool	BatSet.subset : 'a t -> 'a t -> bool		
			Base.Set.sum : (module Base__.Container.Summable with type t = 'sum) -> ('a, 'b) t -> f:('a -> 'sum) -> 'sum	
		BatSet.sym_diff : 'a t -> 'a t -> 'a t	Base.Set.symmetric_diff : ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'a) Base__.Either.t Base__.Sequence.t	
		BatSet.to_array : 'a t -> 'a array	Base.Set.to_array : ('a, 'b) t -> 'a array	Set.toArray : (Set<'a> -> 'a []) when 'a : comparison
	CCSet.to_iter : t -> elt iter			
	CCSet.to_list : t -> elt list	BatSet.to_list : 'a t -> 'a list	Base.Set.to_list : ('a, 'b) t -> 'a list	Set.toList : (Set<'a> -> 'a list) when 'a : comparison
Set.to_rev_seq : t -> elt Seq.t	CCSet.to_rev_seq : t -> elt Seq.t	BatSet.to_rev_seq : 'a t -> 'a BatSeq.t		
Set.to_seq : t -> elt Seq.t	CCSet.to_seq : t -> elt Seq.t	BatSet.to_seq : 'a t -> 'a BatSeq.t		Set.toSeq : (Set<'a> -> seq<'a>) when 'a : comparison
Set.to_seq_from : elt -> t -> elt Seq.t	CCSet.to_seq_from : elt -> t -> elt Seq.t	BatSet.to_seq_from : 'a -> 'a t -> 'a BatSeq.t		
			Base.Set.to_sequence : ?order:[`Decreasing | `Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) t -> 'a Base__.Sequence.t	
	CCSet.to_string : ?start:string -> ?stop:string -> ?sep:string -> (elt -> string) -> t -> string			
Set.union : t -> t -> t	CCSet.union : t -> t -> t	BatSet.union : 'a t -> 'a t -> 'a t	Base.Set.union : ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'cmp) t	Set.union : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
			Base.Set.union_list : ('a, 'cmp) Base__.Comparator.Module.t -> ('a, 'cmp) t list -> ('a, 'cmp) t	
				Set.unionMany : (seq<Set<'a>> -> Set<'a>) when 'a : comparison
			Base.String.ascending : t -> t -> int	
			Base.String.between : t -> low:t -> high:t -> bool	
StringLabels.blit : src:string -> src_pos:int -> dst:bytes -> dst_pos:int -> len:int -> unit	CCStringLabels.blit : src:t -> src_pos:int -> dst:Bytes.t -> dst_pos:int -> len:int -> unit			
StringLabels.capitalize : string -> string	CCStringLabels.capitalize : string -> string		Base.String.capitalize : t -> t	
StringLabels.capitalize_ascii : string -> string	CCStringLabels.capitalize_ascii : string -> string			

Stdlib	Containers	Batteries	Base	F#
StringLabels.cat : string -> string -> string	CCStringLabels.cat : string -> string -> string			
	CCStringLabels.chop_prefix : pre:string -> string -> string option		Base.String.chop_prefix : t -> prefix:t -> t option	
			Base.String.chop_prefix_exn : t -> prefix:t -> t	
			Base.String.chop_prefix_if_exists : t -> prefix:t -> t	
	CCStringLabels.chop_suffix : suf:string -> string -> string option		Base.String.chop_suffix : t -> suffix:t -> t option	
			Base.String.chop_suffix_exn : t -> suffix:t -> t	
			Base.String.chop_suffix_if_exists : t -> suffix:t -> t	
			Base.String.clamp : t -> min:t -> max:t -> t Base__.Or_error.t	
			Base.String.clamp_exn : t -> min:t -> max:t -> t	
			Base.String.common_prefix : t list -> t	
			Base.String.common_prefix2 : t -> t -> t	
			Base.String.common_prefix2_length : t -> t -> int	
			Base.String.common_prefix_length : t list -> int	
			Base.String.common_suffix : t list -> t	
			Base.String.common_suffix2 : t -> t -> t	
			Base.String.common_suffix2_length : t -> t -> int	
			Base.String.common_suffix_length : t list -> int	
			Base.String.comparator : (t, comparator_witness) Base__Comparator.comparator	
StringLabels.compare : t -> t -> int	CCStringLabels.compare : string -> string -> int		Base.String.compare : t -> t -> 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		Base.String.concat : ?sep:t -> t list -> t	
			Base.String.concat_array : ?sep:t -> t array -> t	
	CCStringLabels.concat_gen : sep:string -> string gen -> string			
	CCStringLabels.concat_iter : sep:string -> string iter -> string			
			Base.String.concat_map : ?sep:t -> t -> f:(char -> t) -> t	
	CCStringLabels.concat_seq : sep:string -> string Seq.t -> string			
StringLabels.contains : string -> char -> bool	CCStringLabels.contains : string -> char -> bool		Base.String.contains : ?pos:int -> ?len:int -> t -> char -> bool	
StringLabels.contains_from : string -> int -> char -> bool	CCStringLabels.contains_from : string -> int -> char -> bool			
StringLabels.copy : string -> string	CCStringLabels.copy : string -> string		Base.String.copy : t -> t	
			Base.String.count : t -> f:(elt -> bool) -> int	
			Base.String.counti : t -> f:(int -> elt -> bool) -> int	
			Base.String.descending : t -> t -> int	
	CCStringLabels.drop : int -> string -> string			

Stdlib	Containers	Batteries	Base	F#
			Base.String.drop_prefix : t -> int -> t	
			Base.String.drop_suffix : t -> int -> t	
	CCStringLabels.drop_while : f:(char -> bool) -> t -> t			
	CCStringLabels.edit_distance : ?cutoff:int -> string -> string -> int			
StringLabels.empty : string	CCStringLabels.empty : string			
StringLabels.ends_with : suffix:string -> string -> bool	CCStringLabels.ends_with : suffix:string -> string -> bool			
StringLabels.equal : t -> t -> bool	CCStringLabels.equal : string -> string -> bool		Base.String.equal : t -> t -> bool	
	CCStringLabels.equal_caseless : string -> string -> bool			
StringLabels.escaped : string -> string	CCStringLabels.escaped : string -> string		Base.String.escaped : t -> t	
StringLabels.exists : f:(char -> bool) -> string -> bool	CCStringLabels.exists : f:(char -> bool) -> string -> bool		Base.String.exists : t -> f:(elt -> bool) -> bool	
			Base.String.existsi : t -> f:(int -> elt -> bool) -> bool	
	CCStringLabels.exists2 : f:(char -> char -> bool) -> string -> string -> bool			
StringLabels.fill : bytes -> pos:int -> len:int -> char -> unit	CCStringLabels.fill : bytes -> pos:int -> len:int -> char -> unit			
	CCStringLabels.filter : f:(char -> bool) -> string -> string		Base.String.filter : t -> f:(char -> bool) -> t	
			Base.String.filteri : t -> f:(int -> char -> bool) -> t	
	CCStringLabels.filter_map : f:(char -> char option) -> string -> string			
	CCStringLabels.find : ?start:int -> sub:string -> string -> int		Base.String.find : t -> f:(elt -> bool) -> elt option	
	CCStringLabels.find_all : ?start:int -> sub:string -> string -> int gen			
	CCStringLabels.find_all_l : ?start:int -> sub:string -> string -> int list			
			Base.String.find_map : t -> f:(elt -> 'a option) -> 'a option	
			Base.String.find_mapi : t -> f:(int -> elt -> 'a option) -> 'a option	
			Base.String.findi : t -> f:(int -> elt -> bool) -> (int * elt) option	
	CCStringLabels.flat_map : ?sep:string -> f:(char -> string) -> string -> string			
	CCStringLabels.fold : f:('a -> char -> 'a) -> init:'a -> t -> 'a		Base.String.fold : t -> init:'accum -> f:('accum -> elt -> 'accum) -> 'accum	
	CCStringLabels.fold2 : f:('a -> char -> char -> 'a) -> init:'a -> string -> string -> 'a			
StringLabels.fold_left : f:('a -> char -> 'a) -> init:'a -> string -> 'a	CCStringLabels.fold_left : f:('a -> char -> 'a) -> init:'a -> string -> 'a			
			Base.String.fold_result : t -> init:'accum -> f:('accum -> elt -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t	
StringLabels.fold_right : f:(char -> 'a -> 'a) -> string -> init:'a -> 'a	CCStringLabels.fold_right : f:(char -> 'a -> 'a) -> string -> init:'a -> 'a			
			Base.String.fold_until : t -> init:'accum -> f:('accum -> elt -> ('accum, 'final) Base__Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final	
	CCStringLabels.foldi : f:('a -> int -> char -> 'a) -> 'a -> t -> 'a		Base.String.foldi : t -> init:'a -> f:(int -> 'a -> char -> 'a) -> 'a	
StringLabels.for_all : f:(char -> bool) -> string -> bool	CCStringLabels.for_all : f:(char -> bool) -> string -> bool		Base.String.for_all : t -> f:(elt -> bool) -> bool	

Stdlib	Containers	Batteries	Base	F#
			Base.String.for_alli : t -> f:(int -> elt -> bool) -> bool	
	CCStringLabels.for_all2 : f:(char -> char -> bool) -> string -> string -> bool			
StringLabels.get_int16_be : string -> int -> int	CCStringLabels.get_int16_be : string -> int -> int			
StringLabels.get_int16_le : string -> int -> int	CCStringLabels.get_int16_le : string -> int -> int			
StringLabels.get_int16_ne : string -> int -> int	CCStringLabels.get_int16_ne : string -> int -> int			
StringLabels.get_int32_be : string -> int -> int32	CCStringLabels.get_int32_be : string -> int -> int32			
StringLabels.get_int32_le : string -> int -> int32	CCStringLabels.get_int32_le : string -> int -> int32			
StringLabels.get_int32_ne : string -> int -> int32	CCStringLabels.get_int32_ne : string -> int -> int32			
StringLabels.get_int64_be : string -> int -> int64	CCStringLabels.get_int64_be : string -> int -> int64			
StringLabels.get_int64_le : string -> int -> int64	CCStringLabels.get_int64_le : string -> int -> int64			
StringLabels.get_int64_ne : string -> int -> int64	CCStringLabels.get_int64_ne : string -> int -> int64			
StringLabels.get_int8 : string -> int -> int	CCStringLabels.get_int8 : string -> int -> int			
StringLabels.get_uint16_be : string -> int -> int	CCStringLabels.get_uint16_be : string -> int -> int			
StringLabels.get_uint16_le : string -> int -> int	CCStringLabels.get_uint16_le : string -> int -> int			
StringLabels.get_uint16_ne : string -> int -> int	CCStringLabels.get_uint16_ne : string -> int -> int			
StringLabels.get_uint8 : string -> int -> int	CCStringLabels.get_uint8 : string -> int -> int			
StringLabels.get_utf_16be_uchar : t -> int -> Uchar.utf_decode	CCStringLabels.get_utf_16be_uchar : t -> int -> Uchar.utf_decode			
StringLabels.get_utf_16le_uchar : t -> int -> Uchar.utf_decode	CCStringLabels.get_utf_16le_uchar : t -> int -> Uchar.utf_decode			
StringLabels.get_utf_8_uchar : t -> int -> Uchar.utf_decode	CCStringLabels.get_utf_8_uchar : t -> int -> Uchar.utf_decode			
	CCStringLabels.hash : string -> int			
			Base.String.hash_fold_t : t Base__Ppx_hash_lib.hash_fold	
			Base.String.hashable : t Base__.Hashable.t	
StringLabels.index : string -> char -> int	CCStringLabels.index : string -> char -> int		Base.String.index_exn : t -> char -> int	
			Base.String.index : t -> char -> int option	
StringLabels.index_from : string -> int -> char -> int	CCStringLabels.index_from : string -> int -> char -> int		Base.String.index_from_exn : t -> int -> char -> int	
StringLabels.index_from_opt : string -> int -> char -> int option	CCStringLabels.index_from_opt : string -> int -> char -> int option		Base.String.index_from : t -> int -> char -> int option	
StringLabels.index_opt : string -> char -> int option	CCStringLabels.index_opt : string -> char -> int option			
StringLabels.init : int -> f:(int -> char) -> string	CCStringLabels.init : int -> f:(int -> char) -> string		Base.String.init : int -> f:(int -> char) -> t	
			Base.String.invariant : t Base__Invariant_intf.inv	

Stdlib	Containers	Batteries	Base	F#
	CCStringLabels.is_empty : string -> bool		Base.String.is_empty : t -> bool	
			Base.String.is_prefix : t -> prefix:t -> bool	
			Base.String.is_substring : t -> substring:t -> bool	
	CCStringLabels.is_sub : sub:string -> sub_pos:int -> string -> pos:int -> sub_len:int -> bool		Base.String.is_substring_at : t -> pos:int -> substring:t -> bool	
			Base.String.is_suffix : t -> suffix:t -> bool	
StringLabels.is_valid_utf_16be : t -> bool	CCStringLabels.is_valid_utf_16be : t -> bool			
StringLabels.is_valid_utf_16le : t -> bool	CCStringLabels.is_valid_utf_16le : t -> bool			
StringLabels.is_valid_utf_8 : t -> bool	CCStringLabels.is_valid_utf_8 : t -> bool			
StringLabels.iter : f:(char -> unit) -> string -> unit	CCStringLabels.iter : f:(char -> unit) -> string -> unit		Base.String.iter : t -> f:(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		Base.String.iteri : (t, elt) Base__Indexed_container_intf.iteri	
	CCStringLabels.iteri2 : f:(int -> char -> char -> unit) -> string -> string -> unit			
	CCStringLabels.length : t -> int			
			Base.String.lfindi : ?pos:int -> t -> f:(int -> char -> bool) -> int option	
	CCStringLabels.lines : string -> string list			
	CCStringLabels.lines_gen : string -> string gen			
	CCStringLabels.lines_iter : string -> string iter			
	CCStringLabels.lines_seq : string -> string Seq.t			
StringLabels.lowercase : string -> string	CCStringLabels.lowercase : string -> string		Base.String.lowercase : t -> t	
StringLabels.lowercase_ascii : string -> string	CCStringLabels.lowercase_ascii : string -> string			
			Base.String.lsplit2 : t -> on:char -> (t * t) option	
			Base.String.lsplit2_exn : t -> on:char -> t * t	
			Base.String.lstrip : ?drop:(char -> bool) -> t -> t	
	CCStringLabels.ltrim : t -> t			
StringLabels.make : int -> char -> string	CCStringLabels.make : int -> char -> string		Base.String.make : int -> char -> t	
StringLabels.map : f:(char -> char) -> string -> string	CCStringLabels.map : f:(char -> char) -> string -> string		Base.String.map : t -> f:(char -> char) -> t	
	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		Base.String.mapi : t -> f:(int -> char -> char) -> t	
			Base.String.max : t -> t -> t	
			Base.String.max_elt : t -> compare:(elt -> elt -> int) -> elt option	
			Base.String.max_length : int	
	CCStringLabels.mem : ?start:int -> sub:string -> string -> bool		Base.String.mem : t -> elt -> bool	
			Base.String.min : t -> t -> t	
			Base.String.min_elt : t -> compare:(elt -> elt -> int) -> elt option	
	CCStringLabels.of_array : char array -> string			

Stdlib	Containers	Batteries	Base	F#
StringLabels.of_bytes : bytes -> string	CCStringLabels.of_bytes : bytes -> string			
	CCStringLabels.of_char : char -> string		Base.String.of_char : char -> t	
			Base.String.of_char_list : char list -> t	
	CCStringLabels.of_gen : char gen -> string			
	CCStringLabels.of_hex : string -> string option			
	CCStringLabels.of_hex_exn : string -> string			
	CCStringLabels.of_iter : char iter -> string			
	CCStringLabels.of_list : char list -> string			
StringLabels.of_seq : char Seq.t -> t	CCStringLabels.of_seq : char Seq.t -> string			
			Base.String.of_string : string -> t	
	CCStringLabels.pad : ?side:[`Left | `Right] -> ?c:char -> int -> string -> string			
	CCStringLabels.pp : Format.formatter -> t -> unit		Base.String.pp : Base__.Formatter.t -> t -> unit	
	CCStringLabels.pp_buf : Buffer.t -> t -> unit			
	CCStringLabels.prefix : pre:string -> string -> bool		Base.String.prefix : t -> int -> t	
StringLabels.rcontains_from : string -> int -> char -> bool	CCStringLabels.rcontains_from : string -> int -> char -> bool			
	CCStringLabels.rdrop_while : f:(char -> bool) -> t -> t			
	CCStringLabels.repeat : string -> int -> string			
	CCStringLabels.replace : ?which:[`All | `Left | `Right] -> sub:string -> by:string -> string -> string			
	CCStringLabels.rev : string -> string		Base.String.rev : t -> t	
	CCStringLabels.rfind : sub:string -> string -> int			
			Base.String.rfindi : ?pos:int -> t -> f:(int -> char -> bool) -> int option	
			Base.String.rindex : t -> char -> int option	
StringLabels.rindex : string -> char -> int	CCStringLabels.rindex : string -> char -> int		Base.String.rindex_exn : t -> char -> int	
StringLabels.rindex_from : string -> int -> char -> int	CCStringLabels.rindex_from : string -> int -> char -> int		Base.String.rindex_from_exn : t -> int -> char -> int	
StringLabels.rindex_from_opt : string -> int -> char -> int option	CCStringLabels.rindex_from_opt : string -> int -> char -> int option		Base.String.rindex_from : t -> int -> char -> int option	
StringLabels.rindex_opt : string -> char -> int option	CCStringLabels.rindex_opt : string -> char -> int option			
	CCStringLabels.rtrim : t -> t			
	CCStringLabels.set : string -> int -> char -> string			
			Base.String.rsplit2 : t -> on:char -> (t * t) option	
			Base.String.rsplit2_exn : t -> on:char -> t * t	
			Base.String.rstrip : ?drop:(char -> bool) -> t -> t	
			Base.String.sexp_of_t : t -> Sexplib0__.Sexp.t	
	CCStringLabels.split : by:string -> string -> string list			
StringLabels.split_on_char : sep:char -> string -> string list	CCStringLabels.split_on_char : by:char -> string -> string list		Base.String.split : t -> on:char -> t list	
			Base.String.split_lines : t -> t list	
			Base.String.split_on_chars : t -> on:char list -> t list	
StringLabels.starts_with : prefix:string -> string -> bool	CCStringLabels.starts_with : prefix:string -> string -> bool			

Stdlib	Containers	Batteries	Base	F#
			Base.String.strip : ?drop:(char -> bool) -> t -> t	
StringLabels.sub : string -> pos:int -> len:int -> string	CCStringLabels.sub : string -> pos:int -> len:int -> string		Base.String.sub : (t, t) Base__.Blit.sub	
			Base.String.subo : (t, t) Base__.Blit.subo	
			Base.String.substr_index : ?pos:int -> t -> pattern:t -> int option	
			Base.String.substr_index_all : t -> may_overlap:bool -> pattern:t -> int list	
			Base.String.substr_index_exn : ?pos:int -> t -> pattern:t -> int	
			Base.String.substr_replace_all : t -> pattern:t -> with_:t -> t	
			Base.String.substr_replace_first : ?pos:int -> t -> pattern:t -> with_:t -> t	
	CCStringLabels.suffix : suf:string -> string -> bool			
			Base.String.suffix : t -> int -> t	
			Base.String.sum : (module Base__Container_intf.Summable with type t = 'sum) -> t -> f:(elt -> 'sum) -> 'sum	
			Base.String.t_of_sexp : Sexplib0__.Sexp.t -> t	
			Base.String.t_sexp_grammar : t Sexplib0.Sexp_grammar.t	
	CCStringLabels.take : int -> string -> string			
	CCStringLabels.take_drop : int -> string -> string * string			
	CCStringLabels.to_array : string -> char array		Base.String.to_array : t -> elt array	
StringLabels.to_bytes : string -> bytes	CCStringLabels.to_bytes : string -> bytes			
	CCStringLabels.to_gen : t -> char gen			
	CCStringLabels.to_hex : string -> string			
	CCStringLabels.to_iter : t -> char iter			
	CCStringLabels.to_list : t -> char list		Base.String.to_list : t -> elt list	
			Base.String.to_list_rev : t -> char list	
StringLabels.to_seq : t -> char Seq.t	CCStringLabels.to_seq : t -> char Seq.t			
StringLabels.to_seqi : t -> (int * char) Seq.t	CCStringLabels.to_seqi : t -> (int * char) Seq.t			
			Base.String.to_string : t -> string	
			Base.String.tr : target:char -> replacement:char -> t -> t	
			Base.String.tr_multi : target:t -> replacement:t -> (t -> t) Base__.Staged.t	
StringLabels.trim : string -> string	CCStringLabels.trim : string -> string			
StringLabels.uncapitalize : string -> string	CCStringLabels.uncapitalize : string -> string		Base.String.uncapitalize : t -> t	
StringLabels.uncapitalize_ascii : string -> string	CCStringLabels.uncapitalize_ascii : string -> string			
	CCStringLabels.uniq : eq:(char -> char -> bool) -> string -> string			
	CCStringLabels.unlines : string list -> string			
	CCStringLabels.unlines_gen : string gen -> string			
	CCStringLabels.unlines_iter : string iter -> string			
	CCStringLabels.unlines_seq : string Seq.t -> string			
StringLabels.uppercase : string -> string	CCStringLabels.uppercase : string -> string		Base.String.uppercase : t -> t	
StringLabels.uppercase_ascii : string -> string	CCStringLabels.uppercase_ascii : string -> string			

Stdlib	Containers	Batteries	Base	F#
	CCStringLabels.(<) : t -> t -> bool		Base.String.(<) : t -> t -> bool	
	CCStringLabels.(<=) : t -> t -> bool		Base.String.(<=) : t -> t -> bool	
	CCStringLabels.(<>) : t -> t -> bool		Base.String.(<>) : t -> t -> bool	
	CCStringLabels.(=) : t -> t -> bool		Base.String.(=) : t -> t -> bool	
	CCStringLabels.(>) : t -> t -> bool		Base.String.(>) : t -> t -> bool	
	CCStringLabels.(>=) : t -> t -> bool		Base.String.(>=) : t -> t -> bool	
			Base.String.(^) : t -> t -> t	
		BatString.backwards : string -> char BatEnum.t		
String.blit : string -> int -> bytes -> int -> int -> unit	CCString.blit : t -> int -> Bytes.t -> int -> int -> unit	BatString.blit : string -> int -> Bytes.t -> int -> int -> unit		
String.capitalize : string -> string	CCString.capitalize : string -> string	BatString.capitalize : string -> string		
String.capitalize_ascii : string -> string	CCString.capitalize_ascii : string -> string	BatString.capitalize_ascii : string -> string		
String.cat : string -> string -> string	CCString.cat : string -> string -> string	BatString.cat : string -> string -> string		
		BatString.chop : ?l:int -> ?r:int -> string -> string		
	CCString.chop_prefix : pre:string -> string -> string option			
	CCString.chop_suffix : suf:string -> string -> string option			
				String.collect : ((char -> string) -> string -> string)
String.compare : t -> t -> int	CCString.compare : string -> string -> int	BatString.compare : t -> t -> 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	BatString.concat : string -> string list -> string		String.concat : (string -> seq<string> -> string)
	CCString.concat_gen : sep:string -> string gen -> string			
	CCString.concat_iter : sep:string -> string iter -> string			
	CCString.concat_seq : sep:string -> string Seq.t -> string	BatString.contains : string -> char -> bool		
String.contains : string -> char -> bool	CCString.contains : string -> char -> bool	BatString.contains_from : string -> int -> char -> bool		
String.contains_from : string -> int -> char -> bool	CCString.contains_from : string -> int -> char -> bool			
String.copy : string -> string	CCString.copy : string -> string	BatString.copy : string -> string		
		BatString.count_char : string -> char -> int		
		BatString.count_string : string -> string -> int		
		BatString.cut_on_char : char -> int -> string -> string		
	CCString.drop : int -> string -> string			
	CCString.drop_while : (char -> bool) -> t -> t			
	CCString.edit_distance : ?cutoff:int -> string -> string -> int	BatString.edit_distance : t -> t -> int		
String.empty : string	CCString.empty : string	BatString.empty : string		
String.ends_with : suffix:string -> string -> bool	CCString.ends_with : suffix:string -> string -> bool	BatString.ends_with : string -> string -> bool		
		BatString.ends_with_stdlib : suffix:string -> string -		

Stdlib	Containers	Batteries	Base	F#
		> bool		
		BatString.enum : string -> char BatEnum.t		
String.equal : t -> t -> bool	CCString.equal : t -> t -> bool	BatString.equal : t -> t -> bool		
	CCString.equal_caseless : string -> string -> bool			
String.escaped : string -> string	CCString.escaped : string -> string	BatString.escaped : string -> string		
String.exists : (char -> bool) -> string -> bool	CCString.exists : (char -> bool) -> string -> bool	BatString.exists : string -> string -> bool		String.exists : ((char -> bool) -> string -> bool)
		BatString.exists_stdlib : (char -> bool) -> string -> bool		
	CCString.exists2 : (char -> char -> bool) -> string -> string -> bool			
		BatString.explode : string -> char list		
String.fill : bytes -> int -> int -> char -> unit	CCString.fill : bytes -> int -> int -> char -> unit	BatString.fill : Bytes.t -> int -> int -> char -> unit		
	CCString.filter : (char -> bool) -> string -> string	BatString.filter : (char -> bool) -> string -> string		String.filter : ((char -> bool) -> string -> string)
	CCString.filter_map : (char -> char option) -> string -> string	BatString.filter_map : (char -> char option) -> string -> string		
	CCString.find : ?start:int -> sub:string -> string -> int	BatString.find : string -> string -> int		
	CCString.find_all : ?start:int -> sub:string -> string -> int gen	BatString.find_all : string -> string -> int BatEnum.t		
		BatString.find_from : string -> int -> string -> int		
	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 -> t -> 'a			
	CCString.fold2 : ('a -> char -> char -> 'a) -> 'a -> string -> string -> 'a			
String.fold_left : ('a -> char -> 'a) -> 'a -> string -> 'a	CCString.fold_left : ('a -> char -> 'a) -> 'a -> string -> 'a	BatString.fold_left : ('a -> char -> 'a) -> 'a -> string -> 'a		
		BatString.fold_lefti : ('a -> int -> char -> 'a) -> 'a -> string -> 'a		
String.fold_right : (char -> 'a -> 'a) -> string -> 'a -> 'a	CCString.fold_right : (char -> 'a -> 'a) -> string -> 'a -> 'a	BatString.fold_right : (char -> 'a -> 'a) -> string -> 'a -> 'a		
		BatString.fold_righti : (int -> char -> 'a -> 'a) -> string -> 'a -> 'a		
	CCString.foldi : ('a -> int -> char -> 'a) -> 'a -> t -> 'a			
String.for_all : (char -> bool) -> string -> bool	CCString.for_all : (char -> bool) -> string -> bool	BatString.for_all : (char -> bool) -> string -> bool		String.forall : ((char -> bool) -> string -> bool)
	CCString.for_all2 : (char -> char -> bool) -> string -> string -> bool			
String.get_int16_be : string -> int -> int	CCString.get_int16_be : string -> int -> int	BatString.get_int16_be : string -> int -> int		
String.get_int16_le : string -> int -> int	CCString.get_int16_le : string -> int -> int	BatString.get_int16_le : string -> int -> int		
String.get_int16_ne : string -> int -> int	CCString.get_int16_ne : string -> int -> int	BatString.get_int16_ne : string -> int -> int		
String.get_int32_be : string -> int -> int32	CCString.get_int32_be : string -> int -> int32	BatString.get_int32_be : string -> int -> int32		
String.get_int32_le : string -> int -> int32	CCString.get_int32_le : string -> int -> int32	BatString.get_int32_le : string -> int -> int32		
String.get_int32_ne : string -> int -> int32	CCString.get_int32_ne : string -> int -> int32	BatString.get_int32_ne : string -> int -> int32		

Stdlib	Containers	Batteries	Base	F#
String.get_int64_be : string -> int -> int64	CCString.get_int64_be : string -> int -> int64	BatString.get_int64_be : string -> int -> int64		
String.get_int64_le : string -> int -> int64	CCString.get_int64_le : string -> int -> int64	BatString.get_int64_le : string -> int -> int64		
String.get_int64_ne : string -> int -> int64	CCString.get_int64_ne : string -> int -> int64	BatString.get_int64_ne : string -> int -> int64		
String.get_int8 : string -> int -> int	CCString.get_int8 : string -> int -> int	BatString.get_int8 : string -> int -> int		
String.get_uint16_be : string -> int -> int	CCString.get_uint16_be : string -> int -> int	BatString.get_uint16_be : string -> int -> int		
String.get_uint16_le : string -> int -> int	CCString.get_uint16_le : string -> int -> int	BatString.get_uint16_le : string -> int -> int		
String.get_uint16_ne : string -> int -> int	CCString.get_uint16_ne : string -> int -> int	BatString.get_uint16_ne : string -> int -> int		
String.get_uint8 : string -> int -> int	CCString.get_uint8 : string -> int -> int	BatString.get_uint8 : string -> int -> int		
String.get_utf_16be_uchar : t -> int -> Uchar.utf_decode	CCString.get_utf_16be_uchar : t -> int -> Uchar.utf_decode	BatString.get_utf_16be_uchar : t -> int -> Uchar.utf_decode		
String.get_utf_16le_uchar : t -> int -> Uchar.utf_decode	CCString.get_utf_16le_uchar : t -> int -> Uchar.utf_decode	BatString.get_utf_16le_uchar : t -> int -> Uchar.utf_decode		
String.get_utf_8_uchar : t -> int -> Uchar.utf_decode	CCString.get_utf_8_uchar : t -> int -> Uchar.utf_decode	BatString.get_utf_8_uchar : t -> int -> Uchar.utf_decode		
	CCString.hash : string -> int			
		BatString.head : string -> int -> string		
		BatString.icompare : t -> t -> int		
		BatString.implode : char list -> string		
		BatString.in_place_mirror : Bytes.t -> unit		
String.index : string -> char -> int	CCString.index : string -> char -> int	BatString.index : string -> char -> int		
		BatString.index_after_n : char -> int -> string -> int		
String.index_from : string -> int -> char -> int	CCString.index_from : string -> int -> char -> int	BatString.index_from : string -> int -> char -> int		
String.index_from_opt : string -> int -> char -> int option	CCString.index_from_opt : string -> int -> char -> int option	BatString.index_from_opt : string -> int -> char -> int option		
String.index_opt : string -> char -> int option	CCString.index_opt : string -> char -> int option	BatString.index_opt : string -> char -> int option		
String.init : int -> (int -> char) -> string	CCString.init : int -> (int -> char) -> string	BatString.init : int -> (int -> char) -> string		
				String.init : (int -> (int -> string) -> string)
	CCString.is_empty : string -> bool	BatString.is_empty : string -> bool		
	CCString.is_sub : sub:string -> int -> string -> int -> sub_len:int -> bool			
String.is_valid_utf_16be : t -> bool	CCString.is_valid_utf_16be : t -> bool	BatString.is_valid_utf_16be : t -> bool		
String.is_valid_utf_16le : t -> bool	CCString.is_valid_utf_16le : t -> bool	BatString.is_valid_utf_16le : t -> bool		
String.is_valid_utf_8 : t -> bool	CCString.is_valid_utf_8 : t -> bool	BatString.is_valid_utf_8 : t -> bool		
String.iter : (char -> unit) -> string -> unit	CCString.iter : (char -> unit) -> string -> unit	BatString.iter : (char -> unit) -> string -> unit		String.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	BatString.iteri : (int -> char -> unit) -> string -> unit		String.iteri : ((int -> char -> unit) -> string -> unit)
	CCString.iteri2 : (int -> char -> char -> unit) -> string -> string -> unit			
		BatString.join : string -> string list -> string		
		BatString.lchop : ?n:int -> string -> string		
		BatString.left : string -> int -> string		

Stdlib	Containers	Batteries	Base	F#
	CCString.length : t -> int			String.length : (string -> int)
	CCString.lines : string -> string list			
	CCString.lines_gen : string -> string gen			
	CCString.lines_iter : string -> string iter			
	CCString.lines_seq : string -> string Seq.t			
String.lowercase : string -> string	CCString.lowercase : string -> string	BatString.lowercase : string -> string		
String.lowercase_ascii : string -> string	CCString.lowercase_ascii : string -> string	BatString.lowercase_ascii : string -> string		
	CCString.ltrim : t -> t			
String.make : int -> char -> string	CCString.make : int -> char -> string	BatString.make : int -> char -> string		
String.map : (char -> char) -> string -> string	CCString.map : (char -> char) -> string -> string	BatString.map : (char -> char) -> string -> string		String.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	BatString.mapi : (int -> char -> char) -> string -> string		String.mapi : ((int -> char -> char) -> string -> string)
	CCString.mem : ?start:int -> sub:string -> string -> bool			
		BatString.nreplace : str:string -> sub:string -> by:string -> string		
		BatString.nsplit : string -> by:string -> string list		
		BatString.numeric_compare : t -> t -> int		
	CCString.of_array : char array -> string			
		BatString.of_backwards : char BatEnum.t -> string		
String.of_bytes : bytes -> string	CCString.of_bytes : bytes -> string	BatString.of_bytes : Bytes.t -> string		
	CCString.of_char : char -> string	BatString.of_char : char -> string		
		BatString.of_enum : char BatEnum.t -> string		
		BatString.of_float : float -> string		
	CCString.of_gen : char gen -> string			
	CCString.of_hex : string -> string option			
	CCString.of_hex_exn : string -> string			
		BatString.of_int : int -> string		
	CCString.of_iter : char iter -> string			
	CCString.of_list : char list -> string	BatString.of_list : char list -> string		
String.of_seq : char Seq.t -> t	CCString.of_seq : char Seq.t -> string	BatString.of_seq : char Seq.t -> t		
		BatString.ord : t -> t -> BatOrd.order		
	CCString.pad : ?side:[`Left | `Right] -> ?c:char -> int -> string -> string			
	CCString.pp : Format.formatter -> t -> unit			
	CCString.pp_buf : Buffer.t -> t -> unit			
	CCString.prefix : pre:string -> string -> bool			
		BatString.print : 'a BatInnerIO.output -> string -> unit		
		BatString.print_quoted : 'a BatInnerIO.output -> string -> unit		
		BatString.println : 'a BatInnerIO.output -> string -> unit		

Stdlib	Containers	Batteries	Base	F#
		BatString.quote : string -> string		
		BatString.rchop : ?n:int -> string -> string		
String.rcontains_from : string -> int -> char -> bool	CCString.rcontains_from : string -> int -> char -> bool	BatString.rcontains_from : string -> int -> char -> bool		
	CCString.rdrop_while : (char -> bool) -> t -> t			
	CCString.repeat : string -> int -> string	BatString.repeat : string -> int -> string		
	CCString.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		
				String.replicate : (int -> string -> string)
	CCString.rev : string -> string	BatString.rev : string -> string		
		BatString.rev_in_place : Bytes.t -> unit		
	CCString.rfind : sub:string -> string -> int	BatString.rfind : string -> string -> int		
		BatString.rfind_from : string -> int -> string -> int		
		BatString.right : string -> int -> string		
String.rindex : string -> char -> int	CCString.rindex : string -> char -> int	BatString.rindex : string -> char -> int		
String.rindex_from : string -> int -> char -> int	CCString.rindex_from : string -> int -> char -> int	BatString.rindex_from : string -> int -> char -> int		
String.rindex_from_opt : string -> int -> char -> int option	CCString.rindex_from_opt : string -> int -> char -> int option	BatString.rindex_from_opt : string -> int -> char -> int option		
String.rindex_opt : string -> char -> int option	CCString.rindex_opt : string -> char -> int option	BatString.rindex_opt : string -> char -> int option		
		BatString.rsplit : string -> by:string -> string * string		
	CCString.rtrim : t -> t			
	CCString.set : string -> int -> char -> string			
		BatString.slice : ?first:int -> ?last:int -> string -> string		
		BatString.splice : string -> int -> int -> string -> string		
	CCString.split : by:string -> string -> string list	BatString.split : string -> by:string -> string * string		
String.split_on_char : char -> string -> string list	CCString.split_on_char : char -> string -> string list	BatString.split_on_char : char -> string -> string list		
		BatString.split_on_string : by:string -> string -> string list		
String.starts_with : prefix:string -> string -> bool	CCString.starts_with : prefix:string -> string -> bool	BatString.starts_with : string -> string -> bool		
		BatString.starts_with_stdlib : prefix:string -> string -> bool		
		BatString.strip : ?chars:string -> string -> string		
String.sub : string -> int -> int -> string	CCString.sub : string -> int -> int -> string	BatString.sub : string -> int -> int -> string		
	CCString.suffix : suf:string -> string -> bool			
		BatString.tail : string -> int -> string		
	CCString.take : int -> string -> string			
	CCString.take_drop : int -> string -> string * string			
	CCString.to_array : string -> char array			

Stdlib	Containers	Batteries	Base	F#
String.to_bytes : string -> bytes	CCString.to_bytes : string -> bytes	BatString.to_bytes : string -> Bytes.t		
		BatString.to_float : string -> float		
	CCString.to_gen : t -> char gen			
	CCString.to_hex : string -> string			
		BatString.to_int : string -> int		
	CCString.to_iter : t -> char iter			
	CCString.to_list : t -> char list	BatString.to_list : string -> char list		
String.to_seq : t -> char Seq.t	CCString.to_seq : t -> char Seq.t	BatString.to_seq : t -> char Seq.t		
String.to_seqi : t -> (int * char) Seq.t	CCString.to_seqi : t -> (int * char) Seq.t	BatString.to_seqi : t -> (int * char) Seq.t		
String.trim : string -> string	CCString.trim : string -> string	BatString.trim : string -> string		
String.uncapitalize : string -> string	CCString.uncapitalize : string -> string	BatString.uncapitalize : string -> string		
String.uncapitalize_ascii : string -> string	CCString.uncapitalize_ascii : string -> string	BatString.uncapitalize_ascii : string -> string		
	CCString.uniq : (char -> char -> bool) -> string -> string			
	CCString.unlines : string list -> string			
	CCString.unlines_gen : string gen -> string			
	CCString.unlines_iter : string iter -> string			
	CCString.unlines_seq : string Seq.t -> string			
String.uppercase : string -> string	CCString.uppercase : string -> string	BatString.uppercase : string -> string		
String.uppercase_ascii : string -> string	CCString.uppercase_ascii : string -> string	BatString.uppercase_ascii : string -> string		
	CCString.(<) : t -> t -> bool			
	CCString.(<=) : t -> t -> bool			
	CCString.(<>) : t -> t -> bool			
	CCString.(=) : t -> t -> bool			
	CCString.(>) : t -> t -> bool			
	CCString.(>=) : t -> t -> bool			