Base
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
Base.Array.blito : ('a t, 'a t) Base__Blit_intf.blito
Base.Array.cartesian_product : 'a t -> 'b t -> ('a * 'b) 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
Base.Array.copy_matrix : 'a t t -> 'a t t
Base.Array.count : 'a t -> f:('a -> bool) -> int
Base.Array.counti : 'a t -> f:(int -> 'a -> bool) -> int
Base.Array.create : len:int -> 'a -> 'a t
Base.Array.create_float_uninitialized : len:int -> float t
Base.Array.existsi : 'a t -> f:(int -> 'a -> bool) -> bool
Base.Array.filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t
Base.Array.filter_opt : 'a option t -> 'a t
Base.Array.filteri : 'a t -> f:(int -> 'a -> bool) -> 'a t
Base.Array.find : 'a t -> f:('a -> bool) -> 'a option
Base.Array.find_consecutive_duplicate : 'a t -> equal:('a -> 'a -> bool) -> ('a * 'a) option
Base.Array.find_exn : 'a t -> f:('a -> bool) -> 'a
Base.Array.find_map_exn : 'a t -> f:('a -> 'b option) -> 'b
Base.Array.find_mapi_exn : 'a t -> f:(int -> 'a -> 'b option) -> 'b
Base.Array.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option
Base.Array.findi_exn : 'a t -> f:(int -> 'a -> bool) -> int * 'a
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
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
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
Base.Array.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool
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
Base.Array.last : 'a t -> 'a

Base
Base.Array.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
Base.Array.max_length : int
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
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
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
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
Base.Array.reduce : 'a t -> f:('a -> 'a -> 'a) -> 'a option
Base.Array.reduce_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a
Base.Array.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> Sexplib0__.Sexp.t
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
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
Base.Array.to_sequence_mutable : 'a t -> 'a Base__.Sequence.t
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
Base.Array.zip : 'a t -> 'b t -> ('a * 'b) t option
Base.Array.zip_exn : 'a t -> 'b t -> ('a * 'b) t
Base.List.all : 'a t list -> 'a list t
Base.List.all_equal : 'a t -> equal:('a -> 'a -> bool) -> 'a option
Base.List.all_unit : unit t list -> unit t
Base.List.bind : 'a t -> f:('a -> 'b t) -> 'b t
Base.List.cartesian_product : 'a t -> 'b t -> ('a * 'b) t
Base.List.concat_mapi : 'a t -> f:(int -> 'a -> 'b t) -> 'b t
Base.List.concat_no_order : 'a t t -> 'a t

Base
Base.List.contains_dup : 'a t -> compare:('a -> 'a -> int) -> bool
Base.List.counti : 'a t -> f:(int -> 'a -> bool) -> int
Base.List.dedup_and_sort : 'a t -> compare:('a -> 'a -> int) -> 'a t
Base.List.drop_last : 'a t -> 'a t option
Base.List.drop_last_exn : 'a t -> 'a t
Base.List.exists2 : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool Or_unequal_lengths.t
Base.List.existsi : 'a t -> f:(int -> 'a -> bool) -> bool
Base.List.filter_mapi : 'a t -> f:(int -> 'a -> 'b option) -> 'b t
Base.List.filter_opt : 'a option t -> 'a t
Base.List.find_a_dup : 'a t -> compare:('a -> 'a -> int) -> 'a option
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
Base.List.find_map_exn : 'a t -> f:('a -> 'b option) -> 'b
Base.List.find_mapi_exn : 'a t -> f:(int -> 'a -> 'b option) -> 'b
Base.List.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option
Base.List.findi_exn : 'a t -> f:(int -> 'a -> bool) -> int * 'a
Base.List.fold : 'a t -> init:'accum -> f:('accum -> 'a -> 'accum) -> 'accum
Base.List.fold_result : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t
Base.List.fold_until : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
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
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
Base.List.for_all2 : 'a t -> 'b t -> f:('a -> 'b -> bool) -> bool Or_unequal_lengths.t
Base.List.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool
Base.List.groupi : 'a t -> break:(int -> 'a -> 'a -> bool) -> 'a t t
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
Base.List.ignore_m : 'a t -> unit t
Base.List.invariant : 'a Base__Invariant_intf.inv -> 'a t Base__Invariant_intf.inv
Base.List.is_prefix : 'a t -> prefix:'a t -> equal:('a -> 'a -> bool) -> 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
Base.List.iter2 : 'a t -> 'b t -> f:('a -> 'b -> unit) -> unit Or_unequal_lengths.t

Base
Base.List.join : 'a t t -> 'a t
Base.List.map2 : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t Or_unequal_lengths.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
Base.List.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
Base.List.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
Base.List.partition_result : ('ok, 'error) Base__.Result.t t -> 'ok t * 'error t
Base.List.partition3_map : 'a t -> f:('a -> [`Fst of 'b | `Snd of 'c | `Trd of 'd]) -> 'b t * 'c t * 'd t
Base.List.permute : ?random_state:Base__.Random.State.t -> 'a t -> 'a t
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
Base.List.reduce_balanced : 'a t -> f:('a -> 'a -> 'a) -> 'a option
Base.List.reduce_balanced_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a
Base.List.remove_consecutive_duplicates : ?which_to_keep:[`First | `Last] -> 'a t -> equal:('a -> 'a -> bool) -> '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
Base.List.rev_map_append : 'a t -> 'b t -> f:('a -> 'b) -> 'b t
Base.List.rev_map2 : 'a t -> 'b t -> f:('a -> 'b -> 'c) -> 'c t Or_unequal_lengths.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_mapi : 'a t -> f:(int -> 'a -> 'b) -> 'b t
Base.List.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> Sexplib0__.Sexp.t
Base.List.sort_and_group : 'a t -> compare:('a -> 'a -> int) -> 'a t 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
Base.List.sub : 'a t -> pos:int -> len:int -> 'a t
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
Base.List.tl : 'a t -> 'a t option
Base.List.to_array : 'a t -> 'a array
Base.List.to_list : 'a t -> 'a list
Base.List.transpose : 'a t t -> 'a t t option

Base
Base.List.transpose_exn : 'a t t -> 'a t t
Base.List.unordered_append : 'a t -> 'a t -> 'a t
Base.List.unzip : ('a * 'b) t -> 'a t * 'b t
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
Base.Map.add : ('k, 'v, 'cmp) t -> key:'k -> data:'v -> ('k, 'v, 'cmp) t Or_duplicate.t
Base.Map.add_multi : ('k, 'v list, 'cmp) t -> key:'k -> data:'v -> ('k, 'v list, 'cmp) t
Base.Map.append : lower_part:('k, 'v, 'cmp) t -> upper_part:('k, 'v, 'cmp) t -> [`Ok of ('k, 'v, 'cmp) t | `Overlapping_key_ranges]
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
Base.Map.change : ('k, 'v, 'cmp) t -> 'k -> f:('v option -> 'v option) -> ('k, 'v, 'cmp) t
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
Base.Map.compare_m__t : (module Compare_m) -> ('v -> 'v -> int) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> int
Base.Map.count : ('k, 'v, 'a) t -> f:('v -> bool) -> int
Base.Map.counti : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> int
Base.Map.data : ('a, 'v, 'b) t -> 'v list
Base.Map.equal_m__t : (module Equal_m) -> ('v -> 'v -> bool) -> ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> bool
Base.Map.existsi : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> bool
Base.Map.filter_keys : ('k, 'v, 'cmp) t -> f:('k -> bool) -> ('k, 'v, '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
Base.Map.find_multi : ('k, 'v list, 'cmp) t -> 'k -> 'v list
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
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
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.for_alli : ('k, 'v, 'a) t -> f:(key:'k -> data:'v -> bool) -> bool
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

Base
Base.Map.invariants : ('a, 'b, 'c) t -> bool
Base.Map.iter_keys : ('k, 'a, 'b) t -> f:('k -> unit) -> unit
Base.Map.iter2 : ('k, 'v1, 'cmp) t -> ('k, 'v2, 'cmp) t -> f:(key:'k -> data:('v1, 'v2) Merge_element.t -> 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
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
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
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
Base.Map.merge_skewed : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) t -> combine:(key:'k -> 'v -> 'v -> 'v) -> ('k, 'v, 'cmp) 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
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
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
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]
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
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
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

Base
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
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
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
Base.Map.to_tree : ('k, 'v, 'cmp) t -> ('k, 'v, 'cmp) Using_comparator.Tree.t
Base.Option.(*>) : unit t -> 'a t -> 'a t
Base.Option.(<*) : 'a t -> unit t -> 'a t
Base.Option.all : 'a t list -> 'a list t
Base.Option.all_unit : unit t list -> unit t
Base.Option.apply : ('a -> 'b) t -> 'a t -> 'b t
Base.Option.both : 'a t -> 'b t -> ('a * 'b) t
Base.Option.call : 'a -> f:('a -> unit) t -> unit
Base.Option.count : 'a t -> f:('a -> bool) -> int
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
Base.Option.fold_result : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t
Base.Option.fold_until : 'a t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__.Container.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
Base.Option.hash_fold_t : 'a Base__Ppx_hash_lib.hash_fold -> 'a t Base__Ppx_hash_lib.hash_fold
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
Base.Option.join : 'a t t -> 'a t
Base.Option.length : 'a t -> int
Base.Option.map3 : 'a t -> 'b t -> 'c t -> f:('a -> 'b -> 'c -> 'd) -> 'd t
Base.Option.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
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
Base.Option.sexp_of_t : ('a -> Sexplib0__.Sexp.t) -> 'a t -> Sexplib0__.Sexp.t

Base
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
Base.Option.try_with : (unit -> 'a) -> 'a t
Base.Option.try_with_join : (unit -> 'a t) -> 'a t
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
Base.Printf.bprintf : Buffer.t -> ('r, Buffer.t, unit) format -> 'r
Base.Printf.failwithf : (‘r, unit, string, unit -> 'a) format4 -> 'r
Base.Printf.ifprintf : 'a -> ('r, 'a, 'c, unit) format4 -> 'r
Base.Printf.invalid_argf : (‘r, unit, string, unit -> 'a) format4 -> 'r
Base.Printf.kbprintf : (Buffer.t -> 'a) -> Buffer.t -> ('r, Buffer.t, unit, 'a) format4 -> 'r
Base.Printf.ksprintf : (string -> 'a) -> ('r, unit, string, 'a) format4 -> 'r
Base.Printf.sprintf : (‘r, unit, string) format -> ‘r
Base.Result.all : ('a, 'e) t list -> ('a list, 'e) t
Base.Result.all_unit : (unit, 'e) t list -> (unit, 'e) t
Base.Result.bind : ('a, 'e) t -> f:('a -> ('b, 'e) t) -> ('b, 'e) 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
Base.Result.error : ('a, 'err) t -> 'err option
Base.Result.failf : ('a, unit, string, ('b, string) t) format4 -> 'a
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
Base.Result.of_either : ('ok, 'err) Base__.Either0.t -> ('ok, 'err) t
Base.Result.of_option : 'ok option -> error:'err -> ('ok, 'err) t
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
Base.Result.ok_or_failwith : ('ok, string) t -> 'ok
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
Base.Result.try_with : (unit -> 'a) -> ('a, exn) t
Base.Sequence.all : 'a t list -> 'a list t
Base.Sequence.all_unit : unit t list -> unit t
Base.Sequence.bind : 'a t -> f:('a -> 'b t) -> 'b t
Base.Sequence.bounded_length : 'a t -> at_most:int -> [`Greater | `Is of int]
Base.Sequence.cartesian_product : 'a t -> 'b t -> ('a * 'b) t
Base.Sequence.chunks_exn : 'a t -> int -> 'a list t
Base.Sequence.concat : 'a t t -> 'a 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
Base.Sequence.count : 'a t -> f:('a -> bool) -> int
Base.Sequence.counti : 'a t -> f:(int -> 'a -> bool) -> int
Base.Sequence.cycle_list_exn : 'a list -> 'a t
Base.Sequence.delayed_fold : 'a t -> init:'s -> f:('s -> 'a -> k:('s -> 'r) -> 'r) -> finish:('s -> 'r) -> 'r
Base.Sequence.drop_eagerly : 'a t -> int -> 'a t
Base.Sequence.drop_while_option : 'a t -> f:('a -> bool) -> ('a * 'a t) option
Base.Sequence.existsi : 'a t -> f:(int -> 'a -> bool) -> bool
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
Base.Sequence.find : 'a t -> f:('a -> bool) -> 'a option
Base.Sequence.find_consecutive_duplicate : 'a t -> equal:('a -> 'a -> bool) -> ('a * 'a) option
Base.Sequence.find_exn : 'a t -> f:('a -> bool) -> 'a
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
Base.Sequence.findi : 'a t -> f:(int -> 'a -> bool) -> (int * 'a) option
Base.Sequence.fold_m : bind:('acc_m -> f:('acc -> 'acc_m) -> 'acc_m) -> return:('acc -> 'acc_m) -> 'elt 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
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
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
Base.Sequence.for_alli : 'a t -> f:(int -> 'a -> bool) -> bool
Base.Sequence.force_eagerly : 'a t -> 'a t
Base.Sequence.ignore_m : 'a t -> unit t
Base.Sequence.init : int -> f:(int -> 'a) -> 'a t
Base.Sequence.interleaved_cartesian_product : 'a t -> 'b t -> ('a * 'b) t
Base.Sequence.intersperse : 'a t -> sep:'a -> 'a t
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
Base.Sequence.join : 'a t t -> 'a t
Base.Sequence.length_is_bounded_by : ?min:int -> ?max:int -> 'a t -> bool
Base.Sequence.max_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
Base.Sequence.mem : 'a t -> 'a -> equal:('a -> 'a -> bool) -> bool
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) -> ('a, 'b) Merge_with_duplicates_element.t t
Base.Sequence.min_elt : 'a t -> compare:('a -> 'a -> int) -> 'a option
Base.Sequence.next : 'a t -> ('a * 'a t) option
Base.Sequence.nth : 'a t -> int -> 'a option
Base.Sequence.nth_exn : 'a t -> int -> 'a
Base.Sequence.of_lazy : 'a t Base__.Lazy.t -> 'a t
Base.Sequence.of_seq : 'a Base__.Import.Caml.Seq.t -> 'a t
Base.Sequence.reduce : 'a t -> f:('a -> 'a -> 'a) -> 'a option
Base.Sequence.reduce_exn : 'a t -> f:('a -> 'a -> 'a) -> 'a
Base.Sequence.remove_consecutive_duplicates : 'a t -> equal:('a -> 'a -> bool) -> 'a t
Base.Sequence.round_robin : 'a t list -> 'a t
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
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
Base.Sequence.to_seq : 'a t -> 'a Base__.Import.Caml.Seq.t
Base.Sequence.unfold_step : init:'s -> f:('s -> ('a, 's) Step.t) -> 'a t

Base
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
Base.Sequence.zip_full : 'a t -> 'b t -> [`Both of 'a * 'b | `Left of 'a | `Right of 'b] t
Base.Set.are_disjoint : ('a, 'cmp) t -> ('a, 'cmp) t -> bool
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
Base.Set.comparator : ('a, 'cmp) t -> ('a, 'cmp) Base__.Comparator.t
Base.Set.comparator_s : ('a, 'cmp) t -> ('a, 'cmp) Base__.Comparator.Module.t
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
Base.Set.count : ('a, 'b) t -> f:('a -> bool) -> int
Base.Set.equal_m__t : (module Equal_m) -> ('elt, 'cmp) t -> ('elt, 'cmp) t -> bool
Base.Set.find_map : ('a, 'c) t -> f:('a -> 'b option) -> 'b option
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
Base.Set.fold_until : ('a, 'b) t -> init:'accum -> f:('accum -> 'a -> ('accum, 'final) Base__.Container.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
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
Base.Set.invariants : ('a, 'b) t -> bool
Base.Set.is_subset : ('a, 'cmp) t -> of_:('a, 'cmp) t -> bool
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
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
Base.Set.nth : ('a, 'b) t -> int -> 'a option
Base.Set.of_array : ('a, 'cmp) Base__.Comparator.Module.t -> 'a array -> ('a, 'cmp) t
Base.Set.of_increasing_iterator_unchecked : ('a, 'cmp) Base__.Comparator.Module.t -> len:int -> f:(int -> 'a) -> ('a, 'cmp) t
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
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

Base
Base.Set.stable_dedup_list : ('a, 'b) Base__.Comparator.Module.t -> 'a list -> 'a list
Base.Set.sum : (module Base__.Container.Summable with type t = 'sum) -> ('a, 'b) t -> f:('a -> 'sum) -> 'sum
Base.Set.symmetric_diff : ('a, 'cmp) t -> ('a, 'cmp) t -> ('a, 'a) Base__.Either.t Base__.Sequence.t
Base.Set.to_array : ('a, 'b) t -> 'a array
Base.Set.to_sequence : ?order:[`Decreasing | `Increasing] -> ?greater_or_equal_to:'a -> ?less_or_equal_to:'a -> ('a, 'cmp) t -> 'a Base__.Sequence.t
Base.Set.union_list : ('a, 'cmp) Base__.Comparator.Module.t -> ('a, 'cmp) t list -> ('a, 'cmp) t
Base.String.(^) : t -> t -> t
Base.String.ascending : t -> t -> int
Base.String.between : t -> low:t -> high:t -> bool
Base.String.chop_prefix_exn : t -> prefix:t -> t
Base.String.chop_prefix_if_exists : t -> prefix:t -> t
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_prefix_length : t list -> int
Base.String.common_prefix2 : t -> t -> t
Base.String.common_prefix2_length : t -> t -> int
Base.String.common_suffix : t list -> t
Base.String.common_suffix_length : t list -> int
Base.String.common_suffix2 : t -> t -> t
Base.String.common_suffix2_length : t -> t -> int
Base.String.comparator : (t, comparator_witness) Base__Comparator.comparator
Base.String.concat_array : ?sep:t -> t array -> t
Base.String.concat_map : ?sep:t -> t -> f:(char -> 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
Base.String.drop_prefix : t -> int -> t
Base.String.drop_suffix : t -> int -> t
Base.String.existsi : t -> f:(int -> elt -> bool) -> bool
Base.String.filteri : t -> f:(int -> char -> bool) -> t
Base.String.find_map : t -> f:(elt -> 'a option) -> 'a option

Base
Base.String.find_mapi : t -> f:(int -> elt -> 'a option) -> 'a option
Base.String.findi : t -> f:(int -> elt -> bool) -> (int * elt) option
Base.String.fold_result : t -> init:'accum -> f:('accum -> elt -> ('accum, 'e) Base__.Result.t) -> ('accum, 'e) Base__.Result.t
Base.String.fold_until : t -> init:'accum -> f:('accum -> elt -> ('accum, 'final) Base__Container_intf.Continue_or_stop.t) -> finish:('accum -> 'final) -> 'final
Base.String.for_alli : t -> f:(int -> elt -> bool) -> bool
Base.String.hash_fold_t : t Base__Ppx_hash_lib.hash_fold
Base.String.hashable : t Base__.Hashable.t
Base.String.index : t -> char -> int option
Base.String.invariant : t Base__Invariant_intf.inv
Base.String.is_prefix : t -> prefix:t -> bool
Base.String.is_substring : t -> substring:t -> bool
Base.String.is_suffix : t -> suffix:t -> bool
Base.String.lfindi : ?pos:int -> t -> f:(int -> char -> bool) -> int option
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
Base.String.max : t -> t -> t
Base.String.max_elt : t -> compare:(elt -> elt -> int) -> elt option
Base.String.max_length : int
Base.String.min : t -> t -> t
Base.String.min_elt : t -> compare:(elt -> elt -> int) -> elt option
Base.String.of_char_list : char list -> t
Base.String.of_string : string -> t
Base.String.rfindi : ?pos:int -> t -> f:(int -> char -> bool) -> int option
Base.String.rindex : t -> char -> int option
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
Base.String.split_lines : t -> t list
Base.String.split_on_chars : t -> on:char list -> t list
Base.String.strip : ?drop:(char -> bool) -> t -> t
Base.String.subo : (t, t) Base__.Blit.subo
Base.String.substr_index : ?pos:int -> t -> pattern:t -> int option

Base
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
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
Base.String.to_list_rev : t -> char list
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