F#
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 [])
Array.Parallel.partition : (('a -> bool) -> 'a [] -> 'a [] * 'a [])
Array.allPairs : ('a [] -> 'b [] -> ('a * 'b) [])
Array.append : ('a [] -> 'a [] -> 'a [])
Array.average : ???
Array.averageBy : ???
Array.blit : ('a [] -> int -> 'a [] -> int -> int -> unit)
Array.choose : (('a -> 'b option) -> 'a [] -> 'b [])
Array.chunkBySize : (int -> 'a [] -> 'a [] [])
Array.collect : (('a -> 'b []) -> 'a [] -> 'b [])
Array.compareWith : (('a -> 'a -> int) -> 'a [] -> 'a [] -> int)
Array.concat : (seq<'a []> -> 'a [])
Array.contains : ('a -> 'a [] -> bool) when 'a : equality
Array.copy : ('a [] -> 'a [])
Array.countBy : (('a -> 'b) -> 'a [] -> ('b * int) []) when 'b : equality
Array.create : (int -> 'a -> 'a [])
Array.distinct : ('a [] -> 'a []) when 'a : equality
Array.distinctBy : (('a -> 'b) -> 'a [] -> 'a []) when 'b : equality
Array.empty : 'a []
Array.exactlyOne : ('a [] -> 'a)
Array.except : (seq<'a> -> 'a [] -> 'a []) when 'a : equality
Array.exists : (('a -> bool) -> 'a [] -> bool)
Array.exists2 : (('a -> 'b -> bool) -> 'a [] -> 'b [] -> bool)
Array.fill : ('a [] -> int -> int -> 'a -> unit)
Array.filter : (('a -> bool) -> 'a [] -> 'a [])
Array.find : (('a -> bool) -> 'a [] -> 'a)
Array.findBack : (('a -> bool) -> 'a [] -> 'a)
Array.findIndex : (('a -> bool) -> 'a [] -> int)

F#
Array.findIndexBack : (('a -> bool) -> 'a [] -> int)
Array.fold : (('a -> 'b -> 'a) -> 'a -> 'b [] -> 'a)
Array.fold2 : (('a -> 'b -> 'c -> 'a) -> 'a -> 'b [] -> 'c [] -> 'a)
Array.foldBack : (('a -> 'b -> 'b) -> 'a [] -> 'b -> 'b)
Array.foldBack2 : (('a -> 'b -> 'c -> 'c) -> 'a [] -> 'b [] -> 'c -> 'c)
Array.forall : (('a -> bool) -> 'a [] -> bool)
Array.forall2 : (('a -> 'b -> bool) -> 'a [] -> 'b [] -> bool)
Array.get : ('a [] -> int -> 'a)
Array.groupBy : (('a -> 'b) -> 'a [] -> ('b * 'a []) []) when 'b : equality
Array.head : ('a [] -> 'a)
Array.indexed : ('a [] -> (int * 'a) [])
Array.init : (int -> (int -> 'a) -> 'a [])
Array.insertAt : ???
Array.insertManyAt : ???
Array.isEmpty : ('a [] -> bool)
Array.item : (int -> 'a [] -> 'a)
Array.iter : (('a -> unit) -> 'a [] -> unit)
Array.iter2 : (('a -> 'b -> unit) -> 'a [] -> 'b [] -> unit)
Array.iteri : ((int -> 'a -> unit) -> 'a [] -> unit)
Array.iteri2 : ((int -> 'a -> 'b -> unit) -> 'a [] -> 'b [] -> unit)
Array.last : ('a [] -> 'a)
Array.length : ('a [] -> int)
Array.map : (('a -> 'b) -> 'a [] -> 'b [])
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)
Array.mapi : ((int -> 'a -> 'b) -> 'a [] -> 'b [])
Array.mapi2 : ((int -> 'a -> 'b -> 'c) -> 'a [] -> 'b [] -> 'c [])
Array.max : ('a [] -> 'a) when 'a : comparison
Array.maxBy : (('a -> 'b) -> 'a [] -> 'a) when 'b : comparison
Array.min : ('a [] -> 'a) when 'a : comparison
Array.minBy : (('a -> 'b) -> 'a [] -> 'a) when 'b : comparison
Array.ofList : ('a list -> 'a [])

F#
Array.ofSeq : (seq<'a> -> 'a [])
Array.pairwise : ('a [] -> ('a * 'a) [])
Array.partition : (('a -> bool) -> 'a [] -> 'a [] * 'a [])
Array.permute : ((int -> int) -> 'a [] -> 'a [])
Array.pick : (('a -> 'b option) -> 'a [] -> 'b)
Array.reduce : (('a -> 'a -> 'a) -> 'a [] -> 'a)
Array.reduceBack : (('a -> 'a -> 'a) -> 'a [] -> 'a)
Array.removeAt : ???
Array.removeManyAt : ???
Array.replicate : (int -> 'a -> 'a [])
Array.rev : ('a [] -> 'a [])
Array.scan : (('a -> 'b -> 'a) -> 'a -> 'b [] -> 'a [])
Array.scanBack : (('a -> 'b -> 'b) -> 'a [] -> 'b -> 'b [])
Array.set : ('a [] -> int -> 'a -> unit)
Array.singleton : ('a -> 'a [])
Array.skip : (int -> 'a [] -> 'a [])
Array.skipWhile : (('a -> bool) -> 'a [] -> 'a [])
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.splitAt : (int -> 'a [] -> 'a [] * 'a [])
Array.splitInto : (int -> 'a [] -> 'a [] [])
Array.sub : ('a [] -> int -> int -> 'a [])
Array.sum : ???
Array.sumBy : ???
Array.tail : ('a [] -> 'a [])
Array.take : (int -> 'a [] -> 'a [])
Array.takeWhile : (('a -> bool) -> 'a [] -> 'a [])
Array.toList : ('a [] -> 'a list)

F#
Array.toSeq : ('a [] -> seq<'a>)
Array.transpose : (seq<'a []> -> 'a [] [])
Array.truncate : (int -> 'a [] -> 'a [])
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) [])
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)

F#
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 -> '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 [,,,])
List.allPairs : ('a list -> 'b list -> ('a * 'b) list)
List.append : ('a list -> 'a list -> 'a list)
List.average : ???
List.averageBy : ???
List.choose : (('a -> 'b option) -> 'a list -> 'b list)
List.chunkBySize : (int -> 'a list -> 'a list list)
List.collect : (('a -> 'b list) -> 'a list -> 'b list)

F#
List.compareWith : (('a -> 'a -> int) -> 'a list -> 'a list -> int)
List.concat : (seq<'a list> -> 'a list)
List.contains : ('a -> 'a list -> bool) when 'a : equality
List.countBy : (('a -> 'b) -> 'a list -> ('b * int) list) when 'b : equality
List.distinct : ('a list -> 'a list) when 'a : equality
List.distinctBy : (('a -> 'b) -> 'a list -> 'a list) when 'b : equality
List.empty : 'a list
List.exactlyOne : ('a list -> 'a)
List.except : (seq<'a> -> 'a list -> 'a list) when 'a : equality
List.exists : (('a -> bool) -> 'a list -> bool)
List.exists2 : (('a -> 'b -> bool) -> 'a list -> 'b list -> bool)
List.filter : (('a -> bool) -> 'a list -> 'a list)
List.find : (('a -> bool) -> 'a list -> 'a)
List.findBack : (('a -> bool) -> 'a list -> 'a)
List.findIndex : (('a -> bool) -> 'a list -> int)
List.findIndexBack : (('a -> bool) -> 'a list -> int)
List.fold : (('a -> 'b -> 'a) -> 'a -> 'b list -> 'a)
List.fold2 : (('a -> 'b -> 'c -> 'a) -> 'a -> 'b list -> 'c list -> 'a)
List.foldBack : (('a -> 'b -> 'b) -> 'a list -> 'b -> 'b)
List.foldBack2 : (('a -> 'b -> 'c -> 'c) -> 'a list -> 'b list -> 'c -> 'c)
List.forall : (('a -> bool) -> 'a list -> bool)
List.forall2 : (('a -> 'b -> bool) -> 'a list -> 'b list -> bool)
List.groupBy : (('a -> 'b) -> 'a list -> ('b * 'a list) list) when 'b : equality
List.head : ('a list -> 'a)
List.indexed : ('a list -> (int * 'a) list)
List.init : (int -> (int -> 'a) -> 'a list)
List.insertAt : ???
List.insertManyAt : ???
List.isEmpty : ('a list -> bool)
List.item : (int -> 'a list -> 'a)
List.iter : (('a -> unit) -> 'a list -> unit)
List.iter2 : (('a -> 'b -> unit) -> 'a list -> 'b list -> unit)
List.iteri : ((int -> 'a -> unit) -> 'a list -> unit)
List.iteri2 : ((int -> 'a -> 'b -> unit) -> 'a list -> 'b list -> unit)

F#
List.last : ('a list -> 'a)
List.length : ('a list -> int)
List.map : (('a -> 'b) -> 'a list -> 'b 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)
List.mapFold : (('a -> 'b -> 'c * 'a) -> 'a -> 'b list -> 'c list * 'a)
List.mapFoldBack : (('a -> 'b -> 'c * 'b) -> 'a list -> 'b -> 'c list * 'b)
List.mapi : ((int -> 'a -> 'b) -> 'a list -> 'b list)
List.mapi2 : ((int -> 'a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list)
List.max : ('a list -> 'a) when 'a : comparison
List.maxBy : (('a -> 'b) -> 'a list -> 'a) when 'b : comparison
List.min : ('a list -> 'a) when 'a : comparison
List.minBy : (('a -> 'b) -> 'a list -> 'a) when 'b : comparison
List.nth : ('a list -> int -> 'a)
List.ofArray : ('a [] -> 'a list)
List.ofSeq : (seq<'a> -> 'a list)
List.pairwise : ('a list -> ('a * 'a) list)
List.partition : (('a -> bool) -> 'a list -> 'a list * 'a list)
List.permute : ((int -> int) -> 'a list -> 'a list)
List.pick : (('a -> 'b option) -> 'a list -> 'b)
List.reduce : (('a -> 'a -> 'a) -> 'a list -> 'a)
List.reduceBack : (('a -> 'a -> 'a) -> 'a list -> 'a)
List.removeAt : ???
List.removeManyAt : ???
List.replicate : (int -> 'a -> 'a list)
List.rev : ('a list -> 'a list)
List.scan : (('a -> 'b -> 'a) -> 'a -> 'b list -> 'a list)
List.scanBack : (('a -> 'b -> 'b) -> 'a list -> 'b -> 'b list)
List.singleton : ('a -> 'a list)
List.skip : (int -> 'a list -> 'a list)
List.skipWhile : (('a -> bool) -> 'a list -> 'a list)
List.sort : ('a list -> 'a list) when 'a : comparison
List.sortBy : (('a -> 'b) -> 'a list -> 'a list) when 'b : comparison
List.sortByDescending : (('a -> 'b) -> 'a list -> 'a list) when 'b : comparison

F#
List.sortDescending : ('a list -> 'a list) when 'a : comparison
List.sortWith : (('a -> 'a -> int) -> 'a list -> 'a list)
List.splitAt : (int -> 'a list -> 'a list * 'a list)
List.splitInto : (int -> 'a list -> 'a list list)
List.sum : ???
List.sumBy : ???
List.tail : ('a list -> 'a list)
List.take : (int -> 'a list -> 'a list)
List.takeWhile : (('a -> bool) -> 'a list -> 'a list)
List.toArray : ('a list -> 'a [])
List.toSeq : ('a list -> seq<'a>)
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 -> 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)
List.unfold : (('a -> ('b * 'a) option) -> 'a -> 'b 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)
Map.add : ('a -> 'b -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
Map.change : ('a -> ('b option -> 'b option) -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
Map.containsKey : ('a -> Map<'a,'b> -> bool) when 'a : comparison
Map.count : (Map<'a,'b> -> int) when 'a : comparison

F#
Map.empty : Map<'a,'b> when 'a : comparison
Map.exists : (('a -> 'b -> bool) -> Map<'a,'b> -> bool) when 'a : comparison
Map.filter : (('a -> 'b -> bool) -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
Map.find : ('a -> Map<'a,'b> -> 'b) when 'a : comparison
Map.findKey : (('a -> 'b -> bool) -> Map<'a,'b> -> 'a) when 'a : comparison
Map.fold : (('a -> 'b -> 'c -> 'a) -> 'a -> Map<'b,'c> -> 'a) when 'b : comparison
Map.foldBack : (('a -> 'b -> 'c -> 'c) -> Map<'a,'b> -> 'c -> 'c) when 'a : comparison
Map.forall : (('a -> 'b -> bool) -> Map<'a,'b> -> bool) when 'a : comparison
Map.isEmpty : (Map<'a,'b> -> bool) when 'a : comparison
Map.iter : (('a -> 'b -> unit) -> Map<'a,'b> -> unit) when 'a : comparison
Map.keys : ???
Map.map : (('a -> 'b -> 'c) -> Map<'a,'b> -> Map<'a,'c>) when 'a : comparison
Map.maxKeyValue : ???
Map.minKeyValue : ???
Map.ofArray : (('a * 'b) [] -> Map<'a,'b>) when 'a : comparison
Map.ofList : (('a * 'b) list -> Map<'a,'b>) when 'a : comparison
Map.ofSeq : (seq<'a * 'b> -> Map<'a,'b>) when 'a : comparison
Map.partition : (('a -> 'b -> bool) -> Map<'a,'b> -> Map<'a,'b> * Map<'a,'b>) when 'a : comparison
Map.pick : (('a -> 'b -> 'c option) -> Map<'a,'b> -> 'c) when 'a : comparison
Map.remove : ('a -> Map<'a,'b> -> Map<'a,'b>) when 'a : comparison
Map.toArray : (Map<'a,'b> -> ('a * 'b) []) when 'a : comparison
Map.toList : (Map<'a,'b> -> ('a * 'b) list) when 'a : comparison
Map.toSeq : (Map<'a,'b> -> seq<'a * 'b>) when 'a : comparison
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.values : ???
Option.bind : (('a -> 'b option) -> 'a option -> 'b option)
Option.contains : ('a -> 'a option -> bool) when 'a : equality
Option.count : ('a option -> int)
Option.defaultValue : ('a -> 'a option -> 'a)
Option.defaultWith : ((unit -> 'a) -> 'a option -> 'a)
Option.exists : (('a -> bool) -> 'a option -> bool)
Option.filter : (('a -> bool) -> 'a option -> 'a option)

F#
Option.flatten : ('a option option -> 'a option)
Option.fold : (('a -> 'b -> 'a) -> 'a -> 'b option -> 'a)
Option.foldBack : (('a -> 'b -> 'b) -> 'a option -> 'b -> 'b)
Option.forall : (('a -> bool) -> 'a option -> bool)
Option.get : ('a option -> 'a)
Option.isNone : ('a option -> bool)
Option.isSome : ('a option -> bool)
Option.iter : (('a -> unit) -> 'a option -> unit)
Option.map : (('a -> 'b) -> 'a option -> 'b option)
Option.map2 : (('a -> 'b -> 'c) -> 'a option -> 'b option -> 'c option)
Option.map3 : (('a -> 'b -> 'c -> 'd) -> 'a option -> 'b option -> 'c option -> 'd option)
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
Option.orElse : ('a option -> 'a option -> 'a option)
Option.orElseWith : ((unit -> 'a option) -> 'a option -> 'a option)
Option.toArray : ('a option -> 'a [])
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
Printf.bprintf : (System.Text.StringBuilder -> Printf.BuilderFormat<'a> -> 'a)
Printf.eprintf : (Printf.TextWriterFormat<'a> -> 'a)
Printf.eprintfn : (Printf.TextWriterFormat<'a> -> 'a)
Printf.failwithf : (Printf.StringFormat<'a,'b> -> 'a)
Printf.fprintf : (System.IO.TextWriter -> Printf.TextWriterFormat<'a> -> 'a)
Printf.fprintfn : (System.IO.TextWriter -> Printf.TextWriterFormat<'a> -> 'a)
Printf.kbprintf : ((unit -> 'a) -> System.Text.StringBuilder -> Printf.BuilderFormat<'b,'a> -> 'b)
Printf.kfprintf : ((unit -> 'a) -> System.IO.TextWriter -> Printf.TextWriterFormat<'b,'a> -> 'b)
Printf.kprintf : ((string -> 'a) -> Printf.StringFormat<'b,'a> -> 'b)
Printf.ksprintf : ((string -> 'a) -> Printf.StringFormat<'b,'a> -> 'b)
Printf.printf : (Printf.TextWriterFormat<'a> -> 'a)
Printf.printfn : (Printf.TextWriterFormat<'a> -> 'a)
Printf.sprintf : (Printf.StringFormat<'a> -> 'a)
Result.Error : arg0:'a -> Result<'b,'a>
Result.Ok : arg0:'a -> Result<'a,'b>

F#
Result.bind : (('a -> Result<'b,'c>) -> Result<'a,'c> -> Result<'b,'c>)
Result.map : (('a -> 'b) -> Result<'a,'c> -> Result<'b,'c>)
Result.mapError : (('a -> 'b) -> Result<'c,'a> -> Result<'c,'b>)
Seq.allPairs : (seq<'a> -> seq<'b> -> seq<'a * 'b>)
Seq.append : (seq<'a> -> seq<'a> -> seq<'a>)
Seq.average : ???
Seq.averageBy : ???
Seq.cache : (seq<'a> -> seq<'a>)
Seq.cast : (System.Collections.IEnumerable -> seq<'a>)
Seq.choose : (('a -> 'b option) -> seq<'a> -> seq<'b>)
Seq.chunkBySize : (int -> seq<'a> -> seq<'a []>)
Seq.collect : (('a -> #seq<'c>) -> seq<'a> -> seq<'c>)
Seq.compareWith : (('a -> 'a -> int) -> seq<'a> -> seq<'a> -> int)
Seq.concat : (seq<#seq<'b>> -> seq<'b>)
Seq.contains : ('a -> seq<'a> -> bool) when 'a : equality
Seq.countBy : (('a -> 'b) -> seq<'a> -> seq<'b * int>) when 'b : equality
Seq.delay : ((unit -> seq<'a>) -> seq<'a>)
Seq.distinct : (seq<'a> -> seq<'a>) when 'a : equality
Seq.distinctBy : (('a -> 'b) -> seq<'a> -> seq<'a>) when 'b : equality
Seq.empty : seq<'a>
Seq.exactlyOne : (seq<'a> -> 'a)
Seq.except : (seq<'a> -> seq<'a> -> seq<'a>) when 'a : equality
Seq.exists : (('a -> bool) -> seq<'a> -> bool)
Seq.exists2 : (('a -> 'b -> bool) -> seq<'a> -> seq<'b> -> bool)
Seq.filter : (('a -> bool) -> seq<'a> -> seq<'a>)
Seq.find : (('a -> bool) -> seq<'a> -> 'a)
Seq.findBack : (('a -> bool) -> seq<'a> -> 'a)
Seq.findIndex : (('a -> bool) -> seq<'a> -> int)
Seq.findIndexBack : (('a -> bool) -> seq<'a> -> int)
Seq.fold : (('a -> 'b -> 'a) -> 'a -> seq<'b> -> 'a)
Seq.fold2 : (('a -> 'b -> 'c -> 'a) -> 'a -> seq<'b> -> seq<'c> -> 'a)
Seq.foldBack : (('a -> 'b -> 'b) -> seq<'a> -> 'b -> 'b)
Seq.foldBack2 : (('a -> 'b -> 'c -> 'c) -> seq<'a> -> seq<'b> -> 'c -> 'c)
Seq.forall : (('a -> bool) -> seq<'a> -> bool)

F#
Seq.forall2 : (('a -> 'b -> bool) -> seq<'a> -> seq<'b> -> bool)
Seq.groupBy : (('a -> 'b) -> seq<'a> -> seq<'b * seq<'a>>) when 'b : equality
Seq.head : (seq<'a> -> 'a)
Seq.indexed : (seq<'a> -> seq<int * 'a>)
Seq.init : (int -> (int -> 'a) -> seq<'a>)
Seq.initInfinite : ((int -> 'a) -> seq<'a>)
Seq.insertAt : ???
Seq.insertManyAt : ???
Seq.isEmpty : (seq<'a> -> bool)
Seq.item : (int -> seq<'a> -> 'a)
Seq.iter : (('a -> unit) -> seq<'a> -> unit)
Seq.iter2 : (('a -> 'b -> unit) -> seq<'a> -> seq<'b> -> unit)
Seq.iteri : ((int -> 'a -> unit) -> seq<'a> -> unit)
Seq.iteri2 : ((int -> 'a -> 'b -> unit) -> seq<'a> -> seq<'b> -> unit)
Seq.last : (seq<'a> -> 'a)
Seq.length : (seq<'a> -> int)
Seq.map : (('a -> 'b) -> seq<'a> -> seq<'b>)
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.mapi : ((int -> 'a -> 'b) -> seq<'a> -> seq<'b>)
Seq.mapi2 : ((int -> 'a -> 'b -> 'c) -> seq<'a> -> seq<'b> -> seq<'c>)
Seq.max : (seq<'a> -> 'a) when 'a : comparison
Seq.maxBy : (('a -> 'b) -> seq<'a> -> 'a) when 'b : comparison
Seq.min : (seq<'a> -> 'a) when 'a : comparison
Seq.minBy : (('a -> 'b) -> seq<'a> -> 'a) when 'b : comparison
Seq.nth : (int -> seq<'a> -> 'a)
Seq.ofArray : ('a [] -> seq<'a>)
Seq.ofList : ('a list -> seq<'a>)
Seq.pairwise : (seq<'a> -> seq<'a * 'a>)
Seq.permute : ((int -> int) -> seq<'a> -> seq<'a>)
Seq.pick : (('a -> 'b option) -> seq<'a> -> 'b)
Seq.readonly : (seq<'a> -> seq<'a>)

F#
Seq.reduce : (('a -> 'a -> 'a) -> seq<'a> -> 'a)
Seq.reduceBack : (('a -> 'a -> 'a) -> seq<'a> -> 'a)
Seq.removeAt : ???
Seq.removeManyAt : ???
Seq.replicate : (int -> 'a -> seq<'a>)
Seq.rev : (seq<'a> -> seq<'a>)
Seq.scan : (('a -> 'b -> 'a) -> 'a -> seq<'b> -> seq<'a>)
Seq.scanBack : (('a -> 'b -> 'b) -> seq<'a> -> 'b -> seq<'b>)
Seq.singleton : ('a -> seq<'a>)
Seq.skip : (int -> seq<'a> -> seq<'a>)
Seq.skipWhile : (('a -> bool) -> seq<'a> -> seq<'a>)
Seq.sort : (seq<'a> -> seq<'a>) when 'a : comparison
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.splitInto : (int -> seq<'a> -> seq<'a []>)
Seq.sum : ???
Seq.sumBy : ???
Seq.tail : (seq<'a> -> seq<'a>)
Seq.take : (int -> seq<'a> -> seq<'a>)
Seq.takeWhile : (('a -> bool) -> seq<'a> -> seq<'a>)
Seq.toArray : (seq<'a> -> 'a [])
Seq.toList : (seq<'a> -> 'a list)
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)

F#
Seq.tryPick : (('a -> 'b option) -> seq<'a> -> 'b option)
Seq.unfold : (('a -> ('b * 'a) option) -> 'a -> seq<'b>)
Seq.updateAt : ???
Seq.where : (('a -> bool) -> seq<'a> -> seq<'a>)
Seq.windowed : (int -> seq<'a> -> seq<'a []>)
Seq.zip : (seq<'a> -> seq<'b> -> seq<'a * 'b>)
Seq.zip3 : (seq<'a> -> seq<'b> -> seq<'c> -> seq<'a * 'b * 'c>)
Set.add : ('a -> Set<'a> -> Set<'a>) when 'a : comparison
Set.contains : ('a -> Set<'a> -> bool) when 'a : comparison
Set.count : (Set<'a> -> int) when 'a : comparison
Set.difference : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
Set.empty : Set<'a> when 'a : comparison
Set.exists : (('a -> bool) -> Set<'a> -> bool) when 'a : comparison
Set.filter : (('a -> bool) -> Set<'a> -> Set<'a>) when 'a : comparison
Set.fold : (('a -> 'b -> 'a) -> 'a -> Set<'b> -> 'a) when 'b : comparison
Set.foldBack : (('a -> 'b -> 'b) -> Set<'a> -> 'b -> 'b) when 'a : comparison
Set.forall : (('a -> bool) -> Set<'a> -> bool) when 'a : comparison
Set.intersect : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
Set.intersectMany : (seq<Set<'a>> -> Set<'a>) when 'a : comparison
Set.isEmpty : (Set<'a> -> bool) when 'a : comparison
Set.isProperSubset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
Set.isProperSuperset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
Set.isSubset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
Set.isSuperset : (Set<'a> -> Set<'a> -> bool) when 'a : comparison
Set.iter : (('a -> unit) -> Set<'a> -> unit) when 'a : comparison
Set.map : (('a -> 'b) -> Set<'a> -> Set<'b>) when 'a : comparison and 'b : comparison
Set.maxElement : (Set<'a> -> 'a) when 'a : comparison
Set.minElement : (Set<'a> -> 'a) when 'a : comparison
Set.ofArray : ('a [] -> Set<'a>) when 'a : comparison
Set.ofList : ('a list -> Set<'a>) when 'a : comparison
Set.ofSeq : (seq<'a> -> Set<'a>) when 'a : comparison
Set.partition : (('a -> bool) -> Set<'a> -> Set<'a> * Set<'a>) when 'a : comparison
Set.remove : ('a -> Set<'a> -> Set<'a>) when 'a : comparison
Set.singleton : ('a -> Set<'a>) when 'a : comparison

F#
Set.toArray : (Set<'a> -> 'a []) when 'a : comparison
Set.toList : (Set<'a> -> 'a list) when 'a : comparison
Set.toSeq : (Set<'a> -> seq<'a>) when 'a : comparison
Set.union : (Set<'a> -> Set<'a> -> Set<'a>) when 'a : comparison
Set.unionMany : (seq<Set<'a>> -> Set<'a>) when 'a : comparison
String.collect : ((char -> string) -> string -> string)
String.concat : (string -> seq<string> -> string)
String.exists : ((char -> bool) -> string -> bool)
String.filter : ((char -> bool) -> string -> string)
String.forall : ((char -> bool) -> string -> bool)
String.init : (int -> (int -> string) -> string)
String.iter : ((char -> unit) -> string -> unit)
String.iteri : ((int -> char -> unit) -> string -> unit)
String.length : (string -> int)
String.map : ((char -> char) -> string -> string)
String.mapi : ((int -> char -> char) -> string -> string)
String.replicate : (int -> string -> string)