List Module

Namespace: FSharp.Collections
Assembly: FSharp.Core.dll

Contains operations for working with values of type list.

Functions and values

Function or value	Description
`List.allPairs list1 list2`	Returns a new list that contains all pairings of elements from two lists. ▶
`List.append list1 list2`	Returns a new list that contains the elements of the first list followed by elements of the second list. ▶
`List.average list`	Returns the average of the values in a non-empty list. ▶
`List.averageBy projection list`	Returns the average of values in a list generated by applying a function to each element of the list. ▶
`List.choose chooser list`	Applies a function to each element in a list and then returns a list of values `v` where the applied function returned `Some(v)`. Returns an empty list when the input list is empty or when the applied chooser function returns `None` for all elements. ▶

Function or value	Description	
`List.chunkBySize chunkSize list`	Divides the input list into lists (chunks) of size at most `chunkSize`. Returns a new list containing the generated lists (chunks) as its elements. Returns an empty list when the input list is empty. ▶	XML MD
`List.collect mapping list`	For each element of the list, applies the given function. Concatenates all the results and returns the combined list. ▶	XML MD
`List.compareWith comparer list1 list2`	Compares two lists using the given comparison function, element by element. ▶	XML MD
`List.concat lists`	Returns a new list that contains the elements of each of the lists in order. ▶	XML MD
`List.contains value source`	Tests if the list contains the specified element. ▶	XML MD
`List.countBy projection list`	Applies a key-generating function to each element of a list and returns a list yielding unique keys and their number of occurrences in the original list. ▶	XML MD
`List.distinct list`	Returns a list that contains no duplicate entries according to generic hash and equality comparisons on the entries. If an element occurs multiple times in the list then the later occurrences are discarded.	XML MD

Function or value	Description
	▶
List.distinctBy projection list	Returns a list that contains no duplicate entries according to the generic hash and equality comparisons on the keys returned by the given key-generating function. If an element occurs multiple times in the list then the later occurrences are discarded. ▶
List.empty	Returns an empty list of the given type. ▶
List.exactlyOne list	Returns the only element of the list. ▶
List.except itemsToExclude list	Returns a new list with the distinct elements of the input list which do not appear in the itemsToExclude sequence, using generic hash and equality comparisons to compare values. ▶
List.exists predicate list	Tests if any element of the list satisfies the given predicate. ▶

Function or value	Description
`List.exists2 predicate list1 list2`	Tests if any pair of corresponding elements of the lists satisfies the given predicate. ▶
`List.filter predicate list`	Returns a new collection containing only the elements of the collection for which the given predicate returns "true" ▶
`List.find predicate list`	Returns the first element for which the given function returns True. Raises `KeyNotFoundException` if no such element exists. ▶
`List.findBack predicate list`	Returns the last element for which the given function returns True. Raises `KeyNotFoundException` if no such element exists. ▶
`List.findIndex`	Returns the index of the first element in the list that satisfies the given predicate. Raises `KeyNotFoundException` if no

Function or value	Description
predicate list	the given predicate. Raises `KeyNotFoundException` if no such element exists. ▶
`List.findIndexBack predicate list`	Returns the index of the last element in the list that satisfies the given predicate. Raises `KeyNotFoundException` if no such element exists. ▶
`List.fold folder state list`	Applies a function to each element of the collection, threading an accumulator argument through the computation. Take the second argument, and apply the function to it and the first element of the list. Then feed this result into the function along with the second element and so on. Return the final result. If the input function is `f` and the elements are `i0...iN` then computes `f (... (f s i0) i1 ...) iN`. ▶
`List.fold2 folder state list1 list2`	Applies a function to corresponding elements of two collections, threading an accumulator argument through the computation. The collections must have identical sizes. If the input function is `f` and the elements are `i0...iN` and `j0...jN` then computes `f (... (f s i0 j0)...) iN jN`. ▶
`List.foldBack folder list state`	Applies a function to each element of the collection, starting from the end, threading an accumulator argument through the computation. If the input function is `f` and the elements are `i0...iN` then computes `f i0 (...(f iN s))`. ▶

Function or value	Description
`List.foldBack2 folder list1 list2 state`	Applies a function to corresponding elements of two collections, threading an accumulator argument through the computation. The collections must have identical sizes. If the input function is `f` and the elements are `i0...iN` and `j0...jN` then computes `f i0 j0 (...(f iN jN s))`. ▶
`List.forall predicate list`	Tests if all elements of the collection satisfy the given predicate. ▶
`List.forall2 predicate list1 list2`	Tests if all corresponding elements of the collection satisfy the given predicate pairwise. ▶
`List.groupBy`	Applies a key-generating function to each element of a list and yields a list of unique keys. Each unique key contains a list

Function or value	Description
projection list	of all elements that match to this key. ▶
List.head list	Returns the first element of the list. ▶
List.indexed list	Returns a new list whose elements are the corresponding elements of the input list paired with the index (from 0) of each element. ▶
List.init length initializer	Creates a list by calling the given generator on each index. ▶
List.insertAt index value source	Return a new list with a new item inserted before the given index. ▶
List.insertManyAt index values source	Return a new list with new items inserted before the given index. ▶
List.isEmpty list	Returns true if the list contains no elements, false otherwise. ▶

Function or value	Description
`List.item index list`	Indexes into the list. The first element has index 0. ▶
`List.iter action list`	Applies the given function to each element of the collection. ▶
`List.iter2 action list1 list2`	Applies the given function to two collections simultaneously. The collections must have identical sizes. ▶
`List.iteri action list`	Applies the given function to each element of the collection. The integer passed to the function indicates the index of the element. ▶
`List.iteri2 action list1 list2`	Applies the given function to two collections simultaneously. The collections must have identical sizes. The integer passed to the function indicates the index of the element. ▶
`List.last list`	Returns the last element of the list. ▶
`List.length list`	Returns the length of the list. ▶

Function or value	Description
`List.map mapping list`	Builds a new collection whose elements are the results of applying the given function to each of the elements of the collection. ▶
`List.map2 mapping list1 list2`	Builds a new collection whose elements are the results of applying the given function to the corresponding elements of the two collections pairwise. ▶
`List.map3 mapping list1 list2 list3`	Builds a new collection whose elements are the results of applying the given function to the corresponding elements of the three collections simultaneously. ▶
`List.mapFold mapping state list`	Combines map and fold. Builds a new list whose elements are the results of applying the given function to each of the elements of the input list. The function is also used to accumulate a final value. ▶
`List.mapFoldBack mapping list state`	Combines map and foldBack. Builds a new list whose elements are the results of applying the given function to each of the elements of the input list. The function is also used to accumulate a final value. ▶

Function or value	Description	
`List.mapi mapping list`	Builds a new collection whose elements are the results of applying the given function to each of the elements of the collection. The integer index passed to the function indicates the index (from 0) of the element being transformed. ▶	XML MD
`List.mapi2 mapping list1 list2`	Like mapi, but mapping corresponding elements from two lists of equal length. ▶	XML MD
`List.max list`	Return the greatest of all elements of the list, compared via Operators.max. ▶	XML MD
`List.maxBy projection list`	Returns the greatest of all elements of the list, compared via Operators.max on the function result. ▶	XML MD
`List.min list`	Returns the lowest of all elements of the list, compared via Operators.min. ▶	XML MD
`List.minBy projection list`	Returns the lowest of all elements of the list, compared via Operators.min on the function result ▶	XML MD
`List.ofArray array`	Builds a list from the given array. ▶	XML MD

Function or value	Description
`List.ofSeq source`	Builds a new list from the given enumerable object. ▶
`List.pairwise list`	Returns a list of each element in the input list and its predecessor, with the exception of the first element which is only returned as the predecessor of the second element. ▶
`List.partition predicate list`	Splits the collection into two collections, containing the elements for which the given predicate returns True and False respectively. Element order is preserved in both of the created lists. ▶
`List.permute indexMap list`	Returns a list with all elements permuted according to the specified permutation. ▶
`List.pick chooser list`	Applies the given function to successive elements, returning the first result where function returns `Some(x)` for some x. If no such element exists then raise KeyNotFoundException ▶
`List.reduce reduction list`	Apply a function to each element of the collection, threading an accumulator argument through the computation. Apply the function to the first two elements of the list. Then feed this result into the function along with the third element and so on. Return the final result. If the input function is `f` and the elements are `i0...iN` then computes `f (... (f i0 i1) i2 ...) iN`. ▶

Function or value	Description	
	▶	
List.reduceBack reduction list	Applies a function to each element of the collection, starting from the end, threading an accumulator argument through the computation. If the input function is `f` and the elements are `i0...iN` then computes `f i0 (...(f iN-1 iN))`. ▶	
List.removeAt index source	Return a new list with the item at a given index removed. ▶	
List.removeManyAt index count source	Return a new list with the number of items starting at a given index removed. ▶	
List.replicate count initial	Creates a list by replicating the given initial value. ▶	
List.rev list	Returns a new list with the elements in reverse order. ▶	
List.scan folder state list	Applies a function to each element of the collection, threading an accumulator argument through the computation. Take the second argument, and apply the function to it and the first element of the list. Then feed this result into the function along with the second element and so on. Returns the list of intermediate results and the final result. ▶	
List.scanBack folder list state	Like `foldBack`, but returns both the intermediary and final results ▶	

Function or value	Description	
`List.singleton value`	Returns a list that contains one item only. ▶	
`List.skip count list`	Returns the list after removing the first N elements. ▶	
`List.skipWhile predicate list`	Bypasses elements in a list while the given predicate returns True, and then returns the remaining elements of the list. ▶	
`List.sort list`	Sorts the given list using Operators.compare. ▶	
`List.sortBy projection list`	Sorts the given list using keys given by the given projection. Keys are compared using Operators.compare. ▶	
`List.sortByDescending projection list`	Sorts the given list in descending order using keys given by the given projection. Keys are compared using Operators.compare. ▶	
`List.sortDescending list`	Sorts the given list in descending order using Operators.compare. ▶	
`List.sortWith comparer list`	Sorts the given list using the given comparison function. ▶	

Function or value	Description	
List.splitAt index list	Splits a list into two lists, at the given index. ▶	
List.splitInto count list	Splits the input list into at most `count` chunks. ▶	
List.sum list	Returns the sum of the elements in the list. ▶	
List.sumBy projection list	Returns the sum of the results generated by applying the function to each element of the list. ▶	
List.tail list	Returns the list after removing the first element. ▶	
List.take count list	Returns the first N elements of the list. ▶	
List.takeWhile predicate list	Returns a list that contains all elements of the original list while the given predicate returns True, and then returns no further elements. ▶	
List.toArray list	Builds an array from the given list. ▶	
List.toSeq list	Views the given list as a sequence. ▶	

Function or value	Description
`List.transpose lists`	Returns the transpose of the given sequence of lists. ▶
`List.truncate count list`	Returns at most N elements in a new list. ▶
`List.tryExactlyOne list`	Returns the only element of the list or `None` if it is empty or contains more than one element. ▶
`List.tryFind predicate list`	Returns the first element for which the given function returns True. Return None if no such element exists. ▶
`List.tryFindBack predicate list`	Returns the last element for which the given function returns True. Return None if no such element exists. ▶
`List.tryFindIndex predicate list`	Returns the index of the first element in the list that satisfies the given predicate. Return `None` if no such element exists. ▶
`List.tryFindIndexBack predicate list`	Returns the index of the last element in the list that satisfies the given predicate. Return `None` if no such element exists. ▶

Function or value	Description
`List.tryHead list`	Returns the first element of the list, or `None` if the list is empty. ▶
`List.tryItem index list`	Tries to find the nth element in the list. Returns `None` if index is negative or the list does not contain enough elements. ▶
`List.tryLast list`	Returns the last element of the list. Return `None` if no such element exists. ▶
`List.tryPick chooser list`	Applies the given function to successive elements, returning `Some(x)` the first result where function returns `Some(x)` for some x. If no such element exists then return `None`. ▶
`List.unfold generator state`	Returns a list that contains the elements generated by the given computation. The generator is repeatedly called to build the list until it returns `None`. The given initial `state` argument is passed to the element generator. ▶
`List.unzip list`	Splits a list of pairs into two lists. ▶
`List.unzip3 list`	Splits a list of triples into three lists. ▶

Function or value	Description
`List.updateAt index value source`	Return a new list with the item at a given index set to the new value. ▶
`List.where predicate list`	Returns a new list containing only the elements of the list for which the given predicate returns "true" ▶
`List.windowed windowSize list`	Returns a list of sliding windows containing elements drawn from the input list. Each window is returned as a fresh list. ▶
`List.zip list1 list2`	Combines the two lists into a list of pairs. The two lists must have equal lengths. ▶
`List.zip3 list1 list2 list3`	Combines the three lists into a list of triples. The lists must have equal lengths. ▶