# Espressioni

## Generazione di espressioni

**{e1, e2, ...}** is a list of elements.

**Array[f, n]** generates a list of length n, with elements f[i].
Array[f, {n1, n2, ...}] generates an n1 X n2 X ... array of nested lists, with elements f[i1, i2, ...].
Array[f, dims, origin] generates a list using the specified index origin (default 1).
Array[f, dims, origin, h] uses head h, rather than List, for each level of the array.

**Range[imax]** generates the list {1, 2, ..., imax}.
Range[imin, imax] generates the list {imin, ..., imax}.
Range[imin, imax, di] uses step di.

**Table[expr, {imax}]** generates a list of imax copies of expr.
Table[expr, {i, imax}] generates a list of the values of expr when i runs from 1 to imax.
Table[expr, {i, imin, imax}] starts with i = imin.
Table[expr, {i, imin, imax, di}] uses steps di.
Table[expr, {i, imin, imax}, {j, jmin, jmax}, ...] gives a nested list. The list associated with i is outermost.

**Permutations[list]** generates a list of all possible permutations of the elements in list.

## Analisi delle espressioni

**AtomQ[expr]** yields True if expr is an expression which cannot be divided into subexpressions, and yields False otherwise.

**ListQ[expr]** gives True if expr is a list, and False otherwise.

**expr[[i]]** or Part[expr, i] gives the ith part of expr.
expr[[-i]] counts from the end.
expr[[0]] gives the head of expr.
expr[[i, j, ...]] or Part[expr, i, j, ...] is equivalent to expr[[i]] [[j]] ....
expr[[ {i1, i2, ...} ]] gives a list of the parts i1, i2, ... of expr.

**Head[expr]** gives the head of expr.

**First[expr]** gives the first element in expr.

**Rest[expr]** gives expr with the first element removed.

**Last[expr]** gives the last element in expr.

**Length[expr]** gives the number of elements in expr.

**LeafCount[expr]** gives the total number of indivisible subexpressions in expr.

**Depth[expr]** gives the maximum number of indices needed to specify any part of expr, plus one.

**OrderedQ[h[e1, e2, ...]]** gives True if the ei are in canonical order, and False otherwise.

**Order[expr1, expr2]** gives 1 if expr1 is before expr2 in canonical order, and -1 if expr1 is after expr2 in canonical order. It gives 0 if expr1 is identical to expr2.

**FreeQ[expr, form]** yields True if no subexpression in expr matches form, and yields False otherwise.
FreeQ[expr, form, levelspec] tests only those parts of expr on levels specified by levelspec.

**MemberQ[list, form]** returns True if an element of list matches form, and False otherwise.
MemberQ[list, form, levelspec] tests all parts of list specified by levelspec.

**Count[list, pattern]** gives the number of elements in list that match pattern.
Count[expr, pattern, levelspec] gives the total number of subexpressions matching pattern that appear at the levels in expr specified by levelspec.

**Position[expr, pattern]** gives a list of the positions at which objects matching pattern appear in expr.
Position[expr, pattern, levspec] finds only objects that appear on levels specified by levspec.

## Manipolazione di espressioni

**Append[expr, elem]** gives expr with elem appended.

**AppendTo[s, elem]** appends elem to the value of s, and resets s to the result.

**Prepend[expr, elem]** gives expr with elem prepended.

**PrependTo[s, elem]** prepends elem to the value of s, and resets s to the result.

**Insert[list, elem, n]** inserts elem at position n in list. If n is negative, the position is counted from the end.
Insert[expr, elem, {i, j, ...}] inserts elem at position {i, j, ...} in expr.
Insert[expr, elem, {{i1, j1, ...}, {i2, j2, ...}, ...}] inserts elem at several positions.

**Delete[expr, n]** deletes the element at position n in expr. If n is negative, the position is counted from the end.
Delete[expr, {i, j, ...}] deletes the part at position {i, j, ...}.
Delete[expr, {{i1, j1, ...}, {i2, j2, ...}, ...}] deletes parts at several positions.

**ReplacePart[expr, new, n]** yields an expression in which the nth part of expr is replaced by new. ReplacePart[expr, new, {i, j, ...}] replaces the part at position {i, j, ...}. ReplacePart[expr, new, {{i1, j1, ...}, {i2, j2, ...}, ...}] replaces parts at several positions by new.

**Take[list, n]** gives the first n elements of list.
Take[list, -n] gives the last n elements of list.
Take[list, {m, n}] gives elements m through n of list.

**Drop[list, n]** gives list with its first n elements dropped.
Drop[list, -n] gives list with its last n elements dropped.
Drop[list, {n}] gives list with its nth element dropped.
Drop[list, {m, n}] gives list with elements m through n dropped.

**Cases[{e1, e2, ...}, pattern]** gives a list of the ei that match the pattern.
Cases[{e1, ...}, pattern -> rhs] or Cases[{e1, ...}, pattern :> rhs] gives a list of the values of rhs corresponding to the ei that match the pattern.
Cases[expr, pattern, levelspec] gives a list of all parts of expr on levels specified by levelspec which match the pattern.

**DeleteCases[expr, pattern]** removes all elements of expr which match pattern.
DeleteCases[expr, pattern, levspec] removes all parts of expr on levels specified by levspec which match pattern.

**Select[list, crit]** picks out all elements ei of list for which crit[ei] is True.
Select[list, crit, n] picks out the first n elements for which crit[ei] is True.

**Join[list1, list2, ...]** concatenates lists together. Join can be used on any set of expressions that have the same head.

**Flatten[list]** flattens out nested lists.
Flatten[list, n] flattens to level n.
Flatten[list, n, h] flattens subexpressions with head h.

**Transpose[list]** transposes the first two levels in list.
Transpose[list, {n1, n2, ...}] transposes list so that the nk-th level in list is the k-th level in the result.

**Complement[eall, e1, e2, ...]** gives the elements in eall which are not in any of the ei.

**Intersection[list1, list2, ...]** gives a sorted list of the elements common to all the listi.

**Union[list1, list2, ...]** gives a sorted list of all the distinct elements that appear in any of the listi.
Union[list] gives a sorted version of a list, in which all duplicated elements have been dropped.

**Sort[list]** sorts the elements of list into canonical order.
Sort[list, p] sorts using the ordering function p.

**Split[list]** splits list into sublists consisting of runs of identical elements.
Split[list, test] treats pairs of adjacent elements as identical whenever applying the function test to them yields True.

**Reverse[expr]** reverses the order of the elements in expr.

**RotateRight[expr, n]** cycles the elements in expr n positions to the right.
RotateRight[expr] cycles one position to the right.
RotateRight[expr, {n1, n2, ...}] cycles elements at successive levels ni positions to the right.

**RotateLeft[expr, n]** cycles the elements in expr n positions to the left.
RotateLeft[expr] cycles one position to the left.
RotateLeft[expr, {n1, n2, ...}] cycles elements at successive levels ni positions to the left.

**Normal[expr]** converts expr to a normal expression, from a variety of special forms.