```
(*Instructor：Hiroyuki Akama
```

# Basic Built–in Functions of Mathematica Linear Algebra– List Vector and Matrix (1)

```
*)
```

In[57]:=
```
lst1 = {"a", "b", "c", "a", "d"}
lst2 = {"a", "b", "c", "e", "f"}
{lst1, lst2}
```

Out[57]= {a, b, c, a, d}

Out[58]= {a, b, c, e, f}

Out[59]= {{a, b, c, a, d}, {a, b, c, e, f}}

In[60]:=
```
Length[lst1]
? Length
```

Out[60]= 5

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

In[62]:=
```
Position[lst1, "a"]
? Position
```

Out[62]= {{1}, {4}}

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

In[64]:=
```
Count[lst1, "a"]
? Count
```

Out[64]= 2

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*. ≫

In[66]:=
```
lst1
Part[lst1, 3]
lst1[[3]]
Take[lst1, {2, 3}]
Drop[lst1, 2]
First[lst1]
Last[lst1]
? Part
? Take
? Drop
? First
? Last
```

Out[66]= {a, b, c, a, d}

Out[67]= c

Out[68]= c

Out[69]= {b, c}

Out[70]= {c, a, d}

Out[71]= a

Out[72]= d

*expr*[[*i*]] or Part[*expr*, *i*] gives the $i^{\text{th}}$ part of *expr*.
*expr*[[−*i*]] counts from the end.
*expr*[[*i*, *j*, …]] or Part[*expr*, *i*, *j*, …] is equivalent to *expr*[[*i*]][[*j*]] ….
*expr*[[{$i_1$, $i_2$, …}]] gives a list of the parts $i_1$, $i_2$, … of *expr*.
*expr*[[*m* ;; *n*]] gives parts *m* through *n*.
*expr*[[*m* ;; *n* ;; *s*]] gives parts *m* through *n* in steps of *s*. ≫

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*.
Take[*list*, $seq_1$, $seq_2$, …] gives a nested list in which elements specified by $seq_i$ are taken at level *i* in *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 $n^{\text{th}}$ element dropped.
Drop[*list*, {*m*, *n*}] gives *list* with elements *m* through *n* dropped.
Drop[*list*, {*m*, *n*, *s*}] gives *list* with elements *m* through *n* in steps of *s* dropped.
Drop[*list*, $seq_1$, $seq_2$, …] gives a nested list
in which elements specified by $seq_i$ have been dropped at level *i* in *list*. ≫

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

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

In[78]:=
```
Sort[lst1]
? Sort
```

Out[78]= {a, a, b, c, d}

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

In[80]:=
```
Split[lst1]
Split[Sort[lst1]]
? Split
```

Out[80]= {{a}, {b}, {c}, {a}, {d}}

Out[81]= {{a, a}, {b}, {c}, {d}}

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. ≫

In[83]:=
```
Flatten[Split[Sort[lst1]]]
Prepend[lst1, "x"]
? Flatten
? Prepend
```

Out[83]= {a, a, b, c, d}

Out[84]= {x, a, b, c, a, d}

Flatten[*list*] flattens out nested lists.
Flatten[*list*, *n*] flattens to level *n*.
Flatten[*list*, *n*, *h*] flattens subexpressions with head *h*.
Flatten[*list*, {{$s_{11}$, $s_{12}$, …}, {$s_{21}$, $s_{22}$, …}, …}]
flattens *list* by combining all levels $s_{ij}$ to make each level *i* in the result. ≫

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

In[87]:=
```
{lst1, lst2}
Union[lst1]
Union[lst1, lst2]
Join[lst1, lst2]
Intersection[lst1, lst2]
Complement[lst1, lst2]
? Union
? Join
? Intersection
? Complement
```

Out[87]= {{a, b, c, a, d}, {a, b, c, e, f}}

Out[88]= {a, b, c, d}

Out[89]= {a, b, c, d, e, f}

Out[90]= {a, b, c, a, d, a, b, c, e, f}

Out[91]= {a, b, c}

Out[92]= {d}

Union[$list_1$, $list_2$, …] gives a sorted list of all the distinct elements that appear in any of the $list_i$.
Union[*list*] gives a sorted version of a list, in which all duplicated elements have been dropped. ≫

Join[$list_1$, $list_2$, …] concatenates lists or other expressions that share the same head.
Join[$list_1$, $list_2$, …, $n$] joins the objects at level $n$ in each of the $list_i$. ≫

Intersection[$list_1$, $list_2$, …] gives a sorted list of the elements common to all the $list_i$. ≫

Complement[$e_{all}$, $e_1$, $e_2$, …] gives the elements in $e_{all}$ which are not in any of the $e_i$. ≫

In[97]:=
```
{lst1, lst2}
{lst1, lst2} // MatrixForm
Transpose[{lst1, lst2}]
? MatrixForm
? Transpose
```

Out[97]= {{a, b, c, a, d}, {a, b, c, e, f}}

Out[98]//MatrixForm=

$$\begin{pmatrix} a & b & c & a & d \\ a & b & c & e & f \end{pmatrix}$$

Out[99]= {{a, a}, {b, b}, {c, c}, {a, e}, {d, f}}

MatrixForm[*list*] prints with the elements of *list* arranged in a regular array. ≫

Transpose[*list*] transposes the first two levels in *list*.
Transpose[*list*, {$n_1$, $n_2$, …}] transposes *list* so that the $k^{th}$ level in *list* is the $n_k{}^{th}$ level in the result. ≫

In[102]:=
```
Map[f, lst1]
f /@ lst1
? Map
```

Out[102]= {f[a], f[b], f[c], f[a], f[d]}

Out[103]= {f[a], f[b], f[c], f[a], f[d]}

Map[*f*, *expr*] or *f* /@ *expr* applies *f* to each element on the first level in *expr*.
Map[*f*, *expr*, *levelspec*] applies *f* to parts of *expr* specified by *levelspec*. ≫

In[105]:=
```
Split[Sort[lst1]]
Map[Length, Split[Sort[lst1]]]
Length /@ Split[Sort[lst1]]
Length [#] & /@ Split[Sort[lst1]]
```

Out[105]= {{a, a}, {b}, {c}, {d}}

Out[106]= {2, 1, 1, 1}

Out[107]= {2, 1, 1, 1}

Out[108]= {2, 1, 1, 1}

In[109]:=
```
square[x_] := x^2
square[6]
6 // square
```

General::spell1 :
New symbol name "square" is similar to existing symbol "Square" and may be misspelled. ≫

Out[110]= 36

Out[111]= 36

In[112]:=
```
Function[z, z^2];
Function[z, z^2][6]
(#^2) &[6]
```

Out[113]= 36

Out[114]= 36

In[115]:=
```
Slot[]^2
Slot[]^2 &[6]
Slot[1]^2
Slot[1]^2 &[6]
(#^2) &
(#^2) &[6]
Function[Slot[]^2][6]
Function[Slot[1]^2][6]
? Slot
```

Out[115]= $\text{Slot}[\,]^2$

Out[116]= 36

Out[117]= $\#1^2$

Out[118]= 36

Out[119]= $\#1^2$ &

Out[120]= 36

Out[121]= 36

Out[122]= 36

# represents the first argument supplied to a pure function.
#*n* represents the $n^{\text{th}}$ argument. ≫

In[124]:=
```
procSqr = (#^2) &;
procSqr[6]
```

Out[125]= 36

In[126]:=
```
testdata = Table[i, {i, 10}]
(#^2) & /@ testdata
Map[#^2 &, testdata]
Map[Function[Slot[1]^2], testdata]
```

Out[126]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Out[127]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

Out[128]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

Out[129]= {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

In[130]:=
```
str1 = "word"
str2 = "1"
str3 = "The"
str4 = "cat"
str5 = "Word"
```

Out[130]= word

Out[131]= 1

Out[132]= The

Out[133]= cat

Out[134]= Word

In[135]:=
```
LetterQ[str1]
LetterQ[str2]
StringQ[str2]
StringQ[ToInteger[str2]]
StringQ[1]
StringQ[ToString[1]]
? LetterQ
? StringQ
```

Out[135]= True

Out[136]= False

Out[137]= True

Out[138]= False

Out[139]= False

Out[140]= True

LetterQ[*string*] yields True if all the characters in the string are letters, and yields False otherwise. ≫

StringQ[*expr*] gives True if *expr* is a string, and False otherwise. ≫

In[143]:=
```
StringLength[str2]
? StringLength
```

Out[143]= 1

StringLength["*string*"] gives the number of characters in a string. ≫

In[145]:=
```
str3 <> " " <> str4
StringJoin[str3, " ", str4]
? StringJoin
```

Out[145]= The cat

Out[146]= The cat

"$s_1$" <> "$s_2$" <> …, StringJoin["$s_1$", "$s_2$", …] or
StringJoin[{"$s_1$", "$s_2$", …}] yields a string consisting of a concatenation of the $s_i$. ≫

In[148]:=
```
ToUpperCase[str1]
ToLowerCase[str5]
? ToUpperCase
? ToLowerCase
```

Out[148]= WORD

Out[149]= word

ToUpperCase[*string*] yields a string in which all letters have been converted to uppercase. ≫

ToLowerCase[*string*] yields a string in which all letters have been converted to lowercase. ≫

In[152]:=
```
text1 = {"aa", "bb", "cc", "aa", "aa", "ee"};
Sort[text1]
Union[text1]
? Union
```

Out[153]= {aa, aa, aa, bb, cc, ee}

Out[154]= {aa, bb, cc, ee}

Union[$list_1$, $list_2$, …] gives a sorted list of all the distinct elements that appear in any of the $list_i$.
Union[*list*] gives a sorted version of a list, in which all duplicated elements have been dropped. ≫

In[156]:=
```
getFreq[lis_] := {lis, Count[text1, lis]};
Function[x, {x, Count[text1, x]}];
({#, Count[text1, #]}) &;
Map[getFreq, Union[text1]]
getFreq[#] & /@ Union[text1]
```

Out[159]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

Out[160]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

In[161]:=
```
Function[x, {x, Count[text1, x]}][#] & /@ Union[text1]
Map[Function[x, {x, Count[text1, x]}][#] &, Union[text1]]
Function[x, {x, Count[text1, x]}][Slot[]] & /@ Union[text1]
Map[Function[x, {x, Count[text1, x]}][Slot[]] &, Union[text1]]
Map[Function[Function[x, {x, Count[text1, x]}][Slot[1]]], Union[text1]]
```

Out[161]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

Out[162]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

Out[163]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

Out[164]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

Out[165]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

In[166]:=
```
{#, Count[text1, #]} & /@ Union[text1]
```

Out[166]= {{aa, 3}, {bb, 1}, {cc, 1}, {ee, 1}}

In[167]:=
```
text1 = {"a", "b", "c", "a", "n1", "d", "n2"};
text2 = {"a", "b", "c", "a", "n1", "e"};
noiseList = {"n1", "n2", "n3"};
```

In[170]:=
```
wordList = Union[text1, text2]
```

Out[170]= {a, b, c, d, e, n1, n2}

In[171]:=
```
newWordList = Complement[wordList, noiseList]
freq4text1 = Count[text1, #] & /@ newWordList
freq4text2 = Count[text2, #] & /@ newWordList
```

Out[171]= {a, b, c, d, e}

Out[172]= {2, 1, 1, 1, 0}

Out[173]= {2, 1, 1, 0, 1}

In[174]:=
```
data = {newWordList, freq4text1, freq4text2}
data // MatrixForm
Transpose[data] // MatrixForm
```

Out[174]= {{a, b, c, d, e}, {2, 1, 1, 1, 0}, {2, 1, 1, 0, 1}}

Out[175]//MatrixForm=

$$\begin{pmatrix} a & b & c & d & e \\ 2 & 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Out[176]//MatrixForm=

$$\begin{pmatrix} a & 2 & 2 \\ b & 1 & 1 \\ c & 1 & 1 \\ d & 1 & 0 \\ e & 0 & 1 \end{pmatrix}$$

In[177]:=
```
text1 = {"a", "b", "c", "a", "n1", "d", "n2"};
text2 = {"a", "b", "c", "a", "n1", "e"};
text3 = {"a", "b", "c", "a", "n1", "e", "n4"};
```

In[180]:=
```
textList = {text1, text2, text3};

noiseList = {"n1", "n2", "n3", "n4"};
wordList = Union[text1, text2, text3]
{"a", "b", "c", "d", "e", "n1", "n2", "n4"}
```

Out[182]= {a, b, c, d, e, n1, n2, n4}

Out[183]= {a, b, c, d, e, n1, n2, n4}

In[184]:=
```
getFreq[lis_, allLis_] := Map[Count[lis, #] &, allLis]
getFreq[text1, wordList]
```

Out[185]= {2, 1, 1, 1, 0, 1, 1, 0}

In[186]:=
```
data = getFreq[#, wordList] & /@ textList
```

Out[186]= {{2, 1, 1, 1, 0, 1, 1, 0}, {2, 1, 1, 0, 1, 1, 0, 0}, {2, 1, 1, 0, 1, 1, 0, 1}}

In[187]:=
```
data = Prepend[data, wordList]
Transpose[data] // MatrixForm
```

Out[187]= {{a, b, c, d, e, n1, n2, n4}, {2, 1, 1, 1, 0, 1, 1, 0},
{2, 1, 1, 0, 1, 1, 0, 0}, {2, 1, 1, 0, 1, 1, 0, 1}}

Out[188]//MatrixForm=

$$\begin{pmatrix} a & 2 & 2 & 2 \\ b & 1 & 1 & 1 \\ c & 1 & 1 & 1 \\ d & 1 & 0 & 0 \\ e & 0 & 1 & 1 \\ n1 & 1 & 1 & 1 \\ n2 & 1 & 0 & 0 \\ n4 & 0 & 0 & 1 \end{pmatrix}$$

In[189]:=
```
newWordList = Complement[Union[wordList], noiseList]
{"a", "b", "c", "d", "e"}
data = getFreq[#, newWordList] & /@ textList
data = Prepend[data, newWordList];
Transpose[data] // MatrixForm
```

Out[189]= {a, b, c, d, e}

Out[190]= {a, b, c, d, e}

Out[191]= {{2, 1, 1, 1, 0}, {2, 1, 1, 0, 1}, {2, 1, 1, 0, 1}}

Out[193]//MatrixForm=

$$\begin{pmatrix} a & 2 & 2 & 2 \\ b & 1 & 1 & 1 \\ c & 1 & 1 & 1 \\ d & 1 & 0 & 0 \\ e & 0 & 1 & 1 \end{pmatrix}$$