6 リスト

波括弧{}はリストを表すということは以前に述べたが，ここではそのリストについて詳しく説明する．リストは， Mathematica を使う上でいたる所に現れる重要なものである．

■ リスト

■ リストとは

リストは，幾つかのものをカンマ","で区切って並べ，波括弧{}で括ったものである．

```
{a, b, c, d, e, f}
```

```
{a, b, c, d, e, f}
```

それの完全形式は，Listを頭部としてその中に要素が並んだものである．

```
FullForm[{a, b, c, d, e, f}]
```

```
List[a, b, c, d, e, f]
```

要素には左から順に，1,2,...と番号が付いている．それをインデックスという．たとえば {a,b,c,d,e,f} のインデックス 3 の要素は c である．

■ 多重リスト

リストの要素にリストを含めることができる．そのようなものは，ネストしたリスト，あるいは多重リストと呼ばれる．またリストの中にあるリストはサブリストと呼ばれる．

```
{{a, b}, c, {d, e, f}}
```

```
{{a, b}, c, {d, e, f}}
```

このリストの場合，1 番目の要素は {a,b}, 2 番目の要素は c, 3 番目の要素は {d,e,f} である．たとえば e は 3 番目の要素の中の 2 番目の要素である．この 3, 2 を {{a,b},c,{d,e,f}} の中の e のインデックスという．

多重リストの中にある各構成要素には，それが属しているレベルというものがある．一番外側のリストのすぐ内側にある要素のレベルは 1 である．レベル 1 にあるサブリストのすぐ内側にある要素のレベルは 2 である．以下，レベル n にあるサブリストのすぐ内側にある要素のレベルを n+1 と定める．また，一番外側のリスト自身は，特にレベル 0 にあると定める．

多重リスト中の要素のレベル

```
  {{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}
  {                                    }   --- レベル 0
   {      } c  {              } {   }     --- レベル 1
    { }  b       d                { }     --- レベル 2
```

```
a                e  {    }      h       --- レベル 3
                     f  g               --- レベル 4
```

■ リストの長さ

リストの要素の個数を，そのリストの長さといい，Length を用いて求められる．

```
Length[{a, b, c, d, e}]
```

```
5
```

多重リストの場合には，その長さとは，第1レベルにある要素の個数のことである．

```
Length[{{a, b}, c, {d, e, f, g}}]
```

```
3
```

■ 配列

各レベルにあるサブリストの長さがすべて同じ多重リストは，配列（Array）と呼ばれる．その各レベルにあるサブリストの長さが n1,n2,... nk であるとき，その配列を n1×n2×... の k 次元配列という．たとえば， {{a,b,c},{d,e,f}} は2×3の2次元配列である．

■ abcリストの綺麗な表示

リストを綺麗に表示するための関数にはColumnForm,TableForm,TreeFormなどがある．

■ ColumnForm

ColumnForm[list]はリストlistの各要素を縦に一列に並べて表示する．この関数は，リストの第 1 レベルの各要素を一覧したいときに便利である．

```
ColumnForm[{{a, b}, c, {d, {e, f}, g}}]
```

```
{a, b}
c
{d, {e, f}, g}
```

■ TableForm

TableFormは多重リストの全体像を，平面的にわかりやすく表示したいときに便利である．

```
TableForm[{{a, b}, c, {d, {e, f}, g}}]
```

```
a      b
c
d      e
       f      g
```

レベル1の要素が縦に，レベル2の要素が横に，レベル3の要素が縦に並ぶ．以後，レベルが深くなるごとに縦横が交互に入れ替る．

■ **TreeForm**

TreeFormは多重リストの木構造をわかりやすく表示したいときに便利である．

```
TreeForm[{{a, b}, c, {d, {e, f}, g}}]
```

```
List[ |          , c, |                      ]
     List[a, b]     List[d, |          , g]
                            List[e, f]
```

■ リストを作る

■ Tableを用いる

パラメータを用いてリストを作るにはTableを用いればよい．

- Table[expr, {i, imin, imax}]は，パラメータ i を imin から imax まで 1 刻みで動かしたときの，式 expr の値のリストを作る．

```
Table[i^2, {i, 3, 5}]
```

```
{9, 16, 25}
```

imin を省略すると 1 から imax までとなる．

```
Table[i^2, {i, 5}]
```

```
{1, 4, 9, 16, 25}
```

パラメータが必要ではないときには，次のように省略することもできる．

- Table[expr,{n}]は，式 expr を n 回評価してその値からなるリストを返す．

```
Table[3, {10}]
```

```
{3, 3, 3, 3, 3, 3, 3, 3, 3, 3}
```

式 expr は毎回評価されるので，それぞれが異なる値となることもある．

```
Table[Random[], {5}]
```

```
{0.230282, 0.49433, 0.606091, 0.434344, 0.400471}
```

Random[]は0から1の間にある乱数（疑似乱数）を返す．

- Table[expr,{i,imin,imax,istep}] は i を imin から imax まで istep 刻みで動かしたときのリストを作る．

```
Table[i, {i, 13, 56, 10}]
```

```
{13, 23, 33, 43, 53}
```

この実行例を見れば分かるとおり，imax までというよりも， imax を超えない範囲で i が動く．

- Table[expr,{i,imax,imin,-istep}] は i を iman から imin まで -istep 刻みで動かしたときのリストを作る．

```
Table[i, {i, 56, 13, -10}]
```

```
{56, 46, 36, 26, 16}
```

この場合には，imin を下回らない範囲でiが動く．

- imin, imax, istep は整数である必要はない．

```
Table[i, {i, 1 / 4, 5 / 4, 1 / 2}]
```

$$\left\{\frac{1}{4}, \frac{3}{4}, \frac{5}{4}\right\}$$

- Table[expr,{i,imin,imax},{j,imin,imax},...]は，多次元配列（長さのそろった多重リスト）を作る．

```
Table[a[i, j], {i, 3}, {j, 4}]
```

```
{{a[1, 1], a[1, 2], a[1, 3], a[1, 4]},
 {a[2, 1], a[2, 2], a[2, 3], a[2, 4]}, {a[3, 1], a[3, 2], a[3, 3], a[3, 4]}}
```

```
TableForm[%]
```

```
a[1, 1]      a[1, 2]      a[1, 3]      a[1, 4]
a[2, 1]      a[2, 2]      a[2, 3]      a[2, 4]
a[3, 1]      a[3, 2]      a[3, 3]      a[3, 4]
```

出来上がったリストの構造に注意すると，大きな範囲（外側のリスト）ではiが動き，小さな範囲（内側のリスト）ではjが動いている．

多次元配列の場合にも，imin,istepを指定することも同様にできる．

```
Table[a[i, j, k], {i, 2}, {j, 2, 3}, {k, 1, 5, 2}]
```

```
{{{a[1, 2, 1], a[1, 2, 3], a[1, 2, 5]}, {a[1, 3, 1], a[1, 3, 3], a[1, 3, 5]}},
 {{a[2, 2, 1], a[2, 2, 3], a[2, 2, 5]}, {a[2, 3, 1], a[2, 3, 3], a[2, 3, 5]}}}
```

練習問題： 次のリストをTableを用いて作れ．

```
{{{a[2, 3, 1], a[2, 2, 1], a[2, 1, 1]}, {a[2, 3, 2], a[2, 2, 2], a[2, 1, 2]}},
 {{a[4, 3, 1], a[4, 2, 1], a[4, 1, 1]}, {a[4, 3, 2], a[4, 2, 2], a[4, 1, 2]}}}
```

■ Array を用いる

{a[1],a[2],a[3],...}というリストを作るには，Arrayを用いるのが簡単である．Tableとの大きな違いはパラメータが不必要であるということであるが，その代わりに融通がききにくくなっている．

- Array[a,n] は {a[1],a[2],...,a[n]} というリストを返す.

```
Array[a, 5]
```

```
{a[1], a[2], a[3], a[4], a[5]}
```

- Array[a,n,start] は，添字が start から始まる長さが n のリストを返す.

```
Array[a, 5, 3]
```

```
{a[3], a[4], a[5], a[6], a[7]}
```

- Array[a,{m,n}] は m×n の２次元配列を，Array[a,{k,m,n}] は k×m×n の３次元の配列を返す.

```
Array[a, {3, 5}]
```

```
{{a[1, 1], a[1, 2], a[1, 3], a[1, 4], a[1, 5]},
 {a[2, 1], a[2, 2], a[2, 3], a[2, 4], a[2, 5]},
 {a[3, 1], a[3, 2], a[3, 3], a[3, 4], a[3, 5]}}
```

```
Array[a, {2, 3, 4}]
```

```
{{{a[1, 1, 1], a[1, 1, 2], a[1, 1, 3], a[1, 1, 4]},
  {a[1, 2, 1], a[1, 2, 2], a[1, 2, 3], a[1, 2, 4]},
  {a[1, 3, 1], a[1, 3, 2], a[1, 3, 3], a[1, 3, 4]}},
 {{a[2, 1, 1], a[2, 1, 2], a[2, 1, 3], a[2, 1, 4]},
  {a[2, 2, 1], a[2, 2, 2], a[2, 2, 3], a[2, 2, 4]},
  {a[2, 3, 1], a[2, 3, 2], a[2, 3, 3], a[2, 3, 4]}}}
```

- Array[a,{m,n},{s,t}] は添字が s,t から始まる m×n の２次元の配列を返す.

```
Array[a, {3, 5}, {-1, -2}]
```

```
{{a[-1, -2], a[-1, -1], a[-1, 0], a[-1, 1], a[-1, 2]},
 {a[0, -2], a[0, -1], a[0, 0], a[0, 1], a[0, 2]},
 {a[1, -2], a[1, -1], a[1, 0], a[1, 1], a[1, 2]}}
```

練習問題： Table[a[i,j],{i,3,5},{j,2,5}] と同じ結果を，Array を用いて作れ.

■ Rangeを用いる

連続する整数のリストを作るにはRangeが便利である.

- Range[n]は1からnまでの整数のリストを返す.

```
Range[10]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

- Range[m,n]はmからnまでの整数のリストを返す.

```
Range[5, 10]
```

```
{5, 6, 7, 8, 9, 10}
```

- Range[m,n,step]はmからstep刻みに増えていくn以下の整数のリストを返す.

```
Range[2, 50, 10]
```

```
{2, 12, 22, 32, 42}
```

■ インデックスを指定して，リストの一部を取り出す

リストの中の一つの要素あるいは一部分を取り出すための関数にはPart,First,Last,Take,Drop,-Delete,Rest,Extractなどがある.

■ Part

Partはリスト（あるは式）の一つの要素を取り出す関数で，別記号として [[]]とも表すことができる.

- Part[list,n]はlist[[n]]と同じで，listのn番目の要素を返す.

```
Part[{a, b, c, d}, 3]
```

```
c
```

```
{a, b, c, d}[[3]]
```

```
c
```

n が負のときにはリストの終わりから数える.

```
Part[{a, b, c, d}, -2]
```

```
c
```

- Part[list,{n1,n2,...,nk}] は list[[{n1,n2,...,nk}]]と同じで，リストlistのn1番目，n2番目，…nk番目の要素を取り出したリストを返す.

```
Part[{a, b, c, d}, {3, 1, 2}]
```

```
{c, a, b}
```

```
{a, b, c, d}[[{3, 1, 2}]]
```

```
{c, a, b}
```

- Part[list,n1,n2,...,nk]はlist[[n1,n2,...,nk]]と同じで，多重リストlistのn1番目の要素の中のn2番目の要素の中の... nk番目の要素，すなわちインデックスがn1,n2,...,nkの要素を返す. Part[list,n1,n2,n3,... nk]も同様である.

```
Part[{{a, b, c}, {d, e, f}, {g, h, i}}, 2, 3]
```

```
f
```

```
{{a, b, c}, {d, e, f}, {g, h, i}}[[2, 3]]
```

```
f
```

練習問題： Partを用いて{a,b,{c,{d,e},f},{g,h}}の中からeを取り出せ.

■ **First**

● First[list]はlistの最初の要素を取り出す．Part[list,1]すなわちlist[[1]]と同じである.

```
First[{a, b, c, d, e}]
```

```
a
```

■ **Last**

● Last[list]はlistの最後の要素を取り出す．Part[list,-1]すなわちlist[[-1]]と同じである.

```
Last[{a, b, c, d, e}]
```

```
e
```

■ **Take**

Takeは連続する要素を取り出すときに使用する.

● Take[list,n]はlistの1番目からn番目までの要素を取り出し，リストにする.

```
Take[{a, b, c, d, e, f, g}, 3]
```

```
{a, b, c}
```

● Take[list,-n]は最後から数えてn番目から最後までの要素を取り出す.

```
Take[{a, b, c, d, e, f, g}, -3]
```

```
{e, f, g}
```

● Take[list,{m,n}]はm番目からn番目までの要素を取り出す.

```
Take[{a, b, c, d, e, f, g}, {3, 5}]
```

```
{c, d, e}
```

練習問題： Takeを用いて{a,b,c,d,e,f,g}から{b,c,d,e}を取り出せ．また，同じことをPartを用いて行なえ.

■ **Drop**

Dropはリストの連続する幾つかの要素を取り除いた残りのリストを返す.

● Drop[list,n]はリストlistの先頭のn個の要素を取り除いた残りのリストを返す.

```
Drop[{a, b, c, d, e, f, g}, 3]
```

```
{d, e, f, g}
```

- Drop[list,-n]は最後のn個の要素を取り除いた残りのリストを返す.

```
Drop[{a, b, c, d, e, f, g}, -3]
```

```
{a, b, c, d}
```

- Drop[list,{m,n}]はm番目からn番目までの要素を取り除いた残りのリストを返す.

```
Drop[{a, b, c, d, e, f, g}, {3, 5}]
```

```
{a, b, f, g}
```

■ Delete

Deleteは一つあるいはいくつかの要素を削除するときに用いる.

- Delete[list,n]はn番目の要素を削除したリストを返す．これはDrop[list,{n,n}]と同じことである.

```
Delete[{a, b, c, d, e, f}, 3]
```

```
{a, b, d, e, f}
```

- Delete[list,{n1,n2,...,nk}]はインデックスがn1,n2,...,nkの要素を削除したリストを返す.

```
Delete[{{a, b}, c, {d, e, f}}, {3, 2}]
```

```
{{a, b}, c, {d, f}}
```

- Delete[List,{{m1,m2,...},{n1,n2,...},...}]は複数個の要素を削除したリストを返す.

```
Delete[{a, b, c, d, e, f}, {{2}, {4}}]
```

```
{a, c, e, f}
```

■ Rest

- Rest[list]はlistの最初の要素を取り除いた残りのリストを返す．Drop[list,1]と同じである.

```
Rest[{a, b, c, d, e, f, g}]
```

```
{b, c, d, e, f, g}
```

■ Extract

Extractは Partと同じように，リストの一部分を取り出す関数であるが，要素の指定の仕方が異なる．後で述べるPositionが返す値やMapAt,Delete,ReplacePart等の関数で使う要素指定の方式に従っている.

- Extract[list,{n1,n2,...,nk}]はPart[list,n1,n2,...,nk]と同じで，listのn1番目の要素の中のn2番目の要素の中の．．．nk番目の要素，すなわちインデックスがn1,n2,...,nkの要素を返す．

```
Extract[{{a, b, c}, {d, e, f}, {g, h, i}}, {2, 3}]
```

```
f
```

- Extract[list,{{m1,m2,...},{n1,n2,...},...}] は {Extract[list,{m1,m2,...}],-Extract[list,{n1,n2,...}],...}と同じで，list中の指定された複数個の要素のリストを返す．

```
Extract[{{a, b, c}, {d, e, f}, {g, h, i}}, {{2, 3}, {3, 1}}]
```

```
{f, g}
```

■ 要素の検索と検査

リスト中の要素について，検索や検査などを行うための関数には，MemberQ,FreeQ,Count,Position,-Select,Casesなどがある．

後の練習問題のために，次を実行して乱数のリストを作っておく．乱数であるので，ここに載せたものとは必ずしも同じではない．

```
rand = Table[Random[Integer, {0, 9}], {20}]
```

```
{0, 0, 2, 0, 7, 1, 1, 3, 7, 7, 9, 5, 4, 1, 1, 5, 4, 8, 5, 7}
```

■ MemberQ

MemberQは，指定されたものがリストの要素に含まれているかどうか，すなわち，そのリストのメンバーであるかどうかを返す真偽値関数である．

- MemberQ[list,elem]は，リストlistの第1レベルの中にelemが含まれているときにTrueを，含まれていないときにFalseを返す．

```
MemberQ[{a, b, c, d, e}, c]
```

```
True
```

```
MemberQ[{a, b, c, d, e}, f]
```

```
False
```

```
MemberQ[{{a, b}, {c, d, e}}, a]
```

```
False
```

練習問題： 先ほど作った乱数のリストrandに0が含まれているかどうかをMemberQを用いて調べよ．

■ **FreeQ**

FreeQはMemberQとは異なり，リストのすべてのレベルにわたって検索し，そのリストのいかなる部分にも指定された要素がないことを調べる．ちなみに，"X is free from Y." という英語は「X には Y がない」という意味である．

- FreeQ[list,elem]は，リストlistのすべてのレベルにelemがない場合にTrueを，どこかのレベルにelemがあればFalseを返す．

```
FreeQ[{{a, b}, {c, d, e}}, a]

False
```

■ **Count**

- Count[list,elem]はリストlistの中にあるelemの個数を返す．

```
Count[{a, b, c, b, a, c, a}, a]

3
```

練習問題： 先ほど作った乱数のリストrandには1がいくつ含まれているかをCountを用いて調べよ．

■ **Position**

- Position[list,elem]はリストlistの中のelemがある場所のインデックスのリストを返す．

```
Position[{a, b, c, b, a, c, a}, a]

{{1}, {5}, {7}}
```

この結果は，インデックスが1の場所と5の場所と7の場所にaがあることを意味する．

この関数は多重リストに対しても働き，インデックスのリストを返す．

```
Position[{{a, b}, c, {b, a, c}, a}, a]

{{1, 1}, {3, 2}, {4}}
```

この結果は，インデックスが1,1の場所と3,2の場所と4の場所にaがあることを意味する．

Positionが返す値の要素指定の方式はExtractに与えるものと同じであるので，Positionで得られた結果をExtractにそのまま渡すことができる．

```
Extract[{{a, b}, c, {b, a, c}, a}, %]

{a, a, a}
```

練習問題： 先ほど作った乱数のリスト rand のどこに 2 があるか，Position を用いて調べよ．

■ Cases

● Cases[list,elem]はリストlist中の要素elemをすべて取り出したリストを返す.

```
Cases[{a, b, c, b, a, c, a}, a]
```

```
{a, a, a}
```

通常，この関数はパターンを用いてelemを指定する．その場合，パターンにマッチした要素が取り出される.

```
Cases[{a + b, b + c, a + d, b + d, c + d}, a + _]
```

```
{a + b, a + d}
```

_は何にでもマッチするパターンである．したがってa+_はaと何かの和というものにマッチする.

練習問題： 先ほど作った乱数のリストrandから3をすべて取り出せ.

■ DeleteCases

● DeleteCases[list,elem]はリストlistの中でelem以外の要素をすべて取り出したリストを返す.

```
DeleteCases[{a, b, c, b, a, c, a}, a]
```

```
{b, c, b, c}
```

この関数のelemにもパターンを用いることができる.

```
DeleteCases[{a + b, b + c, a + d, b + d, c + d}, a + _]
```

```
{b + c, b + d, c + d}
```

パターンについては後の章で説明する．なお， MemberQ, FreeQ, Count, Position などの関数の引数にも，パターンが使用できる.

■ Select

Selectは与えられた条件を満たす要素をリストから抜き出す関数である．条件は真偽値関数を与えることにより指定する.

● Select[list,test]はリストlistの要素elemの中で，test[elem]が真となるようなものを取り出してリストにする.

```
Select[{1, 2, 3, 4, 5, 6, 7}, EvenQ]
```

```
{2, 4, 6}
```

EvenQ[x]はxが偶数であるときに真となる.

「偶数である」という条件はEvenQという用意された真偽値関数があったので簡単であったが，「5 以下である」などというような条件については，そのような真偽値関数を作る必要がある．このような場合に便利なものが，純関数と呼ばれるものである．ここで純関数について簡単に説明する．

純関数を作るには，与えられる引き数を # で表し， # を含む計算式の後に & を付け加えればよい．たとえば，与えられた引き数の3倍に1を加えた値を返す関数は (3 # + 1)& である．この関数の引き数に2を与えて計算するには， (3 # + 1)&[2] とすればよい．

```
(3 # + 1) &[2]
```

```
7
```

さて，引き数が5以下であるか否かを返す真偽値関数は(#≤5)&とすればよい．

```
(# <= 5) & [3]
```

```
True
```

したがって5以下の要素を取り出すには，条件の関数として (#<=5)& を与えて，次のようにすればよい．

```
Select[{1, 2, 3, 4, 5, 6, 7}, (# <= 5) &]
```

```
{1, 2, 3, 4, 5}
```

練習問題： 先ほど作った乱数のリストrandの要素で5以上であるもののリストを，Selectを用いて作れ．

■ 要素の追加，挿入，充填，変更

リストに要素の追加，挿入，充填，変更を行って新しいリストを作るにはAppend,AppendTo,Prepend,PrependTo,Insert,PadLeft,PadRight,ReplacePartを用いる．

■ 追加

- Append[list,elem]はlistの最後にelemを追加したリストを返す．

```
Append[{a, b, c, d}, x]
```

```
{a, b, c, d, x}
```

- Prepend[list,elem]はlistの最初にelemを追加したリストを返す．

```
Prepend[{a, b, c, d}, x]
```

```
{x, a, b, c, d}
```

■ 挿入

- Insert[list,elem,n]はlistの前からn番目の場所にelemを挿入したリストを返す．

```
Insert[{a, b, c, d, e}, x, 3]
```

```
{a, b, x, c, d, e}
```

- Insert[list,elem,-n]はlistの後ろからn番目の場所にelemを挿入したリストを返す.

```
Insert[{a, b, c, d, e}, x, -3]
```

```
{a, b, c, x, d, e}
```

- Insert[list,elem,{n1,n2,...}]は多重リストのn1,n2,...のインデックスの場所にelemを挿入したリストを返す.

```
Insert[{{a, b}, c, {d, e, f}}, x, {3, 2}]
```

```
{{a, b}, c, {d, x, e, f}}
```

練習問題： {{a,b},{c,d,e}}にxを挿入して{{a,b},{c,x,d,e}}を作れ.

■ 充填

- PadLeft[list,n,x]は全体の長さがnになるようにlistの左側にxを何個か追加したリストを返す.

```
PadLeft[{a, b, c, d}, 7, 0]
```

```
{0, 0, 0, a, b, c, d}
```

- PadRight[list,n,x]は全体の長さがnになるようにlistの右側にxを何個か追加したリストを返す.

```
PadRight[{a, b, c, d}, 7, 0]
```

```
{a, b, c, d, 0, 0, 0}
```

■ 変更

- ReplacePart[list,elem,n]はlistの前からn番目の要素をelemで置き換えたリストを返す.

```
ReplacePart[{a, b, c, d, e, f}, x, 3]
```

```
{a, b, x, d, e, f}
```

- ReplacePart[list,elem,-n]はlistの後ろからn番目の要素をelemで置き換えたリストを返す.

```
ReplacePart[{a, b, c, d, e, f}, x, -3]
```

```
{a, b, c, x, e, f}
```

- ReplacePart[list,elem,{n1,n2,...}]はlistのインデックスがn1,n2,...である要素をelemで置き換えたリストを返す.

```
ReplacePart[{{a, b}, c, {d, e, f}}, x, {3, 2}]
```

```
{{a, b}, c, {d, x, f}}
```

- ReplacePart[list,elem,{{m1,m2,...},{n1,n2,...},...}]はlistの中の複数個の要素をelemで置き換えたリストを返す.

```
ReplacePart[{a, b, c, d, e, f}, x, {{2}, {4}}]
```

```
{a, x, c, x, e, f}
```

- ReplacePart[list1,list2,{{m1,m2,...},{n1,n2,...},...},{{i1,i2,...},{j1,j2,...},...}]はlist1の中のインデックスがm1,m2,...の要素，インデックスがn1,n2,...の要素，...を，list2の中のインデックスがi1,i2,...の要素，インデックスがj1,j2,...の要素，...で置き換えたリストを返す.

```
ReplacePart[{a, b, c, d, e, f}, {x, y, z}, {{2}, {4}}, {{3}, {1}}]
```

```
{a, z, c, x, e, f}
```

練習問題： {{a,b},{c,e,f}} の c を x で置き換えたリストを作れ.

■ リスト変数の再割り当て

Append,Prepend,Insert,ReplacePartなどを行っても，引き数に与えられたリストから新しいリストをつくるだけで，引数のリスト変数の値自体を変化させることはできない.

```
lst = {a, b, c, d, e};
Append[lst, x]
```

```
{a, b, c, d, e, x}
```

```
lst
```

```
{a, b, c, d, e}
```

変数の値を変化させるには，関数を用いて得られた結果を変数に再割り当てする必要がある.

```
lst = Append[lst, x]
```

```
{a, b, c, d, e, x}
```

```
lst
```

```
{a, b, c, d, e, x}
```

AppendとPrependに関しては，計算した結果を引き数に与えられた変数に再割り当てする関数AppendToとPrependToがある．すなわちAppendTo[list,elem]はlist=Append[list,elem]と同じである.

```
AppendTo[lst, y]
```

```
{a, b, c, d, e, x, y}
```

```
lst
```

```
{a, b, c, d, e, x, y}
```

```
PrependTo[lst, z]
```

```
{z, a, b, c, d, e, x, y}
```

```
lst
```

```
{z, a, b, c, d, e, x, y}
```

リストの要素を直接変更するには， list[[n]]=elem とする．

```
lst[[3]] = w
```

```
w
```

```
lst
```

```
{z, a, w, c, d, e, x, y}
```

■ リストの結合

リストをそのまま結合するには Join を用いる．

- Join[list1,list2,...] は list1,list2,...をこの順序で結合したリストを返す．

```
Join[{a, b, c}, {1, 2, 3}, {x, y, z}]
```

```
{a, b, c, 1, 2, 3, x, y, z}
```

■ リストの並べ換え

リストの要素の順序を変更するには， Sort, Reverse, RotateLeft, RotateRight などを用いる．

- Sort[list] は list の要素を標準的順序（小さい順あるいはアルファベット順）に並べ換えたリストを返す．

```
Sort[{5, b, 1, 2, c, 4, 6, 3, 2, a}]
```

```
{1, 2, 2, 3, 4, 5, 6, a, b, c}
```

- Reverse[list] は list の要素の順序を逆にしたリストを返す．

```
Reverse[{a, b, c, d, e}]
```

```
{e, d, c, b, a}
```

- RotateLeft[list,n] は list の全要素を左に n 個だけずらしたリストを返す．ただし，左側にあぶれた要素は，右側に回して置く．n の省略値は 1 である．

```
RotateLeft[{a, b, c, d, e, f}, 2]
```

```
{c, d, e, f, a, b}
```

- RotateRight[list,n] は list の全要素を右に n 個だけずらしたリストを返す．ただし，右側にあぶれた要素は，左側に回して置く．n の省略値は 1 である．

```
RotateRight[{a, b, c, d, e, f}, 2]
```

```
{e, f, a, b, c, d}
```

練習問題： {1,2,3,4,5} を {3,2,1,5,4} に並べ変えることを，Reverse と RotateLeft を用いて行なえ．

■ 多重リストの操作

多重リストを操作する関数には， Flatten, FlattenAt, Partition, Split などがある．

■ Flatten

Flatten はネストしたリストのネストをはずす，すなわち多重リストの重なりを減らすことに用いる．

- Flatten[list] は多重リスト list のすべてのサブリストのネストをはずし，一重のリストにする．

```
Flatten[{{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}]
```

```
{a, b, c, d, e, f, g, h}
```

- Flatten[list,n] は多重リスト list のレベル 1 からレベル n までのサブリストのネストをはずす．

```
Flatten[{{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}, 1]
```

```
{{a}, b, c, d, {e, {f, g}}, {h}}
```

```
Flatten[{{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}, 2]
```

```
{a, b, c, d, e, {f, g}, h}
```

練習問題： {{{a, b}, {c, d}}, {{e, f}, {g, h}}} から Flatten を用いて {{a, b}, {c, d}, {e, f}, {g, h}} というリストを作れ．

■ FlattenAt

- FlattenAt[list,n] あるいは FlattenAt[list,{n1,n2,...}] は，リストlist中の指定された位置にあるサブリストのネストをはずす．

```
FlattenAt[{{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}, 3]
```

```
{{{a}, b}, c, d, {e, {f, g}}, {{h}}}
```

```
FlattenAt[{{{a}, b}, c, {d, {e, {f, g}}}, {{h}}}, {3, 2}]
```

```
{{{a}, b}, c, {d, e, {f, g}}, {{h}}}
```

■ **Partition**

Partition はリストの要素を何個かごとに区切って，多重リストにするために用いる．

- Partition[list,n] は，リスト list の要素を n 個ごとに区切りサブリストにした多重リストを返す．

```
Partition[{a, b, c, d, e, f}, 3]
```

```
{{a, b, c}, {d, e, f}}
```

- Partition[list,n,d] は，list の要素を n 個取り出してサブリストにすることを，d 個ずつずらして行って得られる多重リストを返す．

```
Partition[{a, b, c, d, e, f}, 3, 1]
```

```
{{a, b, c}, {b, c, d}, {c, d, e}, {d, e, f}}
```

```
Partition[Range[10], 5]
```

```
{{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}}
```

練習問題： {1,2,3,4,5,6,7,8,9,10} から Partition を用いて，要素を2個ずつ組みにしたリスト {{1,2}, {3,4}, {5,6}, {7,8}, {9,10}} を作れ．また4個ずつ組みにしたリスト {{1,2,3,4}, {3,4,5,6}, {5,6,7,8}, {7,8,9,10}} を作れ．

■ **Split**

Split はリストの隣り合う同一要素をまとめたいときに用いる．

- Split[list] は，隣り合う同一要素をサブリストにまとめた多重リストを返す．

```
Split[{a, a, b, b, b, a, c, c, c}]
```

```
{{a, a}, {b, b, b}, {a}, {c, c, c}}
```

■ リストを集合として扱う

リスト {...} を集合として見たときには，要素の順序は無視して，さらに重複も無視して扱う．すなわち {b,a,c,a,c} は {a,b,c} と同じ集合を表す．このようにリストを集合として取り扱う関数として Union, Intersection, Complement がある．これらの関数は，結果を標準的な順序に並べ替えて，重複は取り除いたリストとして返す．また記号 ⋃, ⋂ は，BasicInputパレット上にある記号をクリックするか， ⋂, ⋃ とタイプすることにより入力できる．

■ **Union ⋃**

いくつかの集合の和集合を求めるときに使用する.

● Union[list1,list2,...] あるいは list1⋃list2⋃...は，リスト list1, list2,... の少なくとも一つに含まれるような要素からなるリストを返す.

```
Union[{b, a, c, a, c}, {b, e, d, c, d, c}]
```

```
{a, b, c, d, e}
```

```
{b, a, c, a, c}⋃{b, e, d, c, d, c}
```

```
{a, b, c, d, e}
```

リストの要素の重複を取り除き，さらに標準的な順序に並べ換えるためだけに Union を使用することも多い.

```
Union[{b, a, c, a, c}]
```

```
{a, b, c}
```

■ **Intersection ⋂**

いくつかの集合の共通部分（交わり）を求めるときに使用する.

● Intersection[list1,list2,...] あるいは list1⋂list2⋂...は，リストlist1, list2,... のすべてに含まれるような要素からなるリストを返す.

```
Intersection[{b, a, c, a, c}, {b, e, d, c, d, c}]
```

```
{b, c}
```

■ **Complement**

集合の補集合を求めるときに使用する.

● Complement[list,list1,list2,...]は，list の要素で list1, list2,...のいずれにも含まれないようなものからなるリストを返す.

```
Complement[{a, b, c, d, e, f}, {a, b, c}, {c, d}]
```

```
{e, f}
```

練習問題： 三つの集合 X={1,2,3,4} Y={2,3,5,6} Z={3,4,6,7} に対して，分配法則 X⋂(Y⋃Z)=(X⋂Y)⋃(X⋂Z), X⋃(Y⋂Z)=(X⋃Y)⋂(X⋃Z) が成立していることを確かめよ.

■ リスト以外の式への応用

この章で出てきた関数には，リスト以外の式に対しても有効に働くものが多い.

式の一部を取り出す

リストの一部を取り出す関数 Part, Last, First, Take, Drop, Extract などは，そのままの使用法で式の一部を取り出すことに使える.

```
f = a + Cos[x] + Cos[y] + b (a + Sin[x])
```

```
a + Cos[x] + Cos[y] + b (a + Sin[x])
```

この f を使って用例を示す.

```
Last[f]
```

```
b (a + Sin[x])
```

```
Part[f, 4, 2, 2]
```

```
Sin[x]
```

```
Take[f, 2]
```

```
a + Cos[x]
```

```
Drop[f, 1]
```

```
Cos[x] + Cos[y] + b (a + Sin[x])
```

```
Extract[f, {4, 2}]
```

```
a + Sin[x]
```

■ 式の中から項を探す

リストの中の要素を検索，検査するための関数 Position, Count, MembreQ, FreeQ, Select, Cases などは，式の中から指定された項を探すことに使える.

```
f
```

```
a + Cos[x] + Cos[y] + b (a + Sin[x])
```

先ほどのfを使って用例を示す.

```
Position[f, a]
```

```
{{1}, {4, 2, 1}}
```

```
Count[f, a]
```

```
1
```

```
MemberQ[f, b]
```

```
False
```

fの第1レベルにはbはない．MemberQ は第1レベルだけを調べるので，結果はメンバーではない，すなわちFalseとなる．

```
FreeQ[f, b]
```

```
False
```

fの第2レベルにbがある．FreeQ はすべてのレベルにわたって調べるので，結果はfreeではない，すなわちFalseとなる．

```
Cases[f, Cos[_]]
```

```
{Cos[x], Cos[y]}
```

特に，FreeQ は数式を扱うときに大変重宝する．たとえば，FreeQ[f,x] がTrue ならば f を x で微分した D[f,x] は 0 になることがわかる．

■ 演習問題

[6-1] 2 から 100 までの偶数のリストを Table を用いて作れ．また Range を用いて作れ．

[6-2] リスト {{a,b},{c,{d,e,{f,g},h}},{i,j},k} の長さはいくらか．また，要素 f のレベルとインデックスは何か．また， f を Part を用いて取り出せ．

[6-3] {1,2,3,4,5,6,7,8,9} から {1,2,3,8,9} を取り出すことを，組み込み関数一つだけを，1回だけ用いて行なえ．

[6-4] 多重リスト {{a,b},c,{a,{b,{c,a},a}},c} の中のどの位置に a があるかを Position を用いて調べよ．

[6-5] Range と Reverse と Join とを用いて， {1, 2, 3, ..., 8, 9, 10, 10, 9, 8, ..., 3, 2, 1} を作れ．

[6-6] Table[Random[Integer, {0, 1}], {100}] を実行して， 0 と 1 からなる，長さが 100 の乱数の列を作れ．その数列において， 0 と 1 が何回入れ替っているかを Split と Length とを用いて数えよ．たとえば {1, 1, 0, 0, 0, 1, 0, 1} ならば 4 回入れ替っていると数える．

[6–7] Range を用いて，6 から 120 までの 6 の倍数のリストと，15 から 120 までの 15 の倍数のリストを作れ．それら二つの集合の交わりを求めて，それが 30 の倍数のリストになっていることを確かめよ．

[6–8] {{1,2,3},{3,4,5},{5,6,7},{7,8,9}} というリストを作れ．