7 関数を施す

リストに関数を施す方法がいく通りかあり，それらを行うために， Apply, Map, MapAll, MapAt, MapThread などがある．これらはリストだけではなく，式に対しても同様に働く．関数に引き数として与えれたリストにその関数を縫い込むために Thread がある．また，関数を繰り返して施して得られる値のリストを得るために NestList, FoldList, ComposeList などがある．

■ 式の全体に関数を適用する

■ Apply @@

たとえば，リスト {1, 2, 3, 4} のすべての要素の和を求めたいとする． C などのプログラミング言語ならば，要素を逐一加えて総和を計算するプログラムを書く必要があるが， Mathematica では簡単に行う方法がある．リスト {1, 2, 3, 4} の完全形式は List[1, 2, 3, 4] であるが，この頭部である List を Plus という関数でおきかえてやれば，そのまま総和を与えることになる．このような操作を，Plus を List[1, 2, 3, 4] に適用する，と言う．

● Apply[f, g[a, b, ...]] または f @@ g[a, b, ...] は， f[a, b, ...] を返す．

```
In[1]:= Apply[f, g[a, b, c]]
Out[1]= f[a, b, c]

In[2]:= Plus @@ {1, 2, 3, 4}
Out[2]= 10
```

Apply はリスト以外のものに対しても施すことができる．

● Apply[f, g[a, b, ...]] または f @@ g[a, b, ...] は f[a, b, ...] を返す．

```
In[3]:= Apply[f, g[a, b, c]]
Out[3]= f[a, b, c]
```

要するに， Apply[f, expr] は式 expr の頭部を f に付け替えたものを返す．

次のように，関数を適用するレベルを指定することもできる．

● Apply[f, expr, {lev}] は式 expr のレベル lev の要素すべてに f を適用する．

```
In[4]:= Apply[f, {{1, 2, 3}, {4, 5, 6}}, {1}]
Out[4]= {f[1, 2, 3], f[4, 5, 6]}
```

練習問題： {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}} から {{f[1, 2], f[3, 4]}, {f[5, 6], f[7, 8]}} を作れ．

■ 各々の要素に関数を施す

リスト {1, 2, 3, 4} の要素の平方根のリストを求めたいとする．それには各々の要素に関数 Sqrt を施す．すなわち Sqrt を {1, 2, 3, 4} に写像 (map) すればよい．このようにリストの各々の要素に関数を施すために Map, MapAll, MapAt などが用意されている．

■ Map/@

● Map[f,{a,b,...}] または f/@{a,b,...} は {f[a],f[b],...} を返す．

```
In[5]:= Map[f, {a, b, c}]

Out[5]= {f[a], f[b], f[c]}

In[6]:= Map[Sqrt, {1, 2, 3, 4}]

Out[6]= {1, √2, √3, 2}
```

Map はリスト以外のものに対しても施すことができる．

● Map[f,g[a,b,...]] または f/@g[a,b,...] は g[f[a],f[b],...] を返す．

```
In[7]:= Map[f, g[a, b, c]]

Out[7]= g[f[a], f[b], f[c]]
```

Map は，指定しないと，リストあるいは式の第 1 レベルの要素に関数を施すものであるが，レベルを指定することもできる．

● Map[f, expr, {lev}] は式 expr の第 lev レベルの要素に関数 f を施した式を返す．

```
In[8]:= Map[f, {{1, 2, 3}, {4, 5, 6}}, {2}]

Out[8]= {{f[1], f[2], f[3]}, {f[4], f[5], f[6]}}
```

練習問題： {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}} から {{{f[1], f[2]}, {f[3], f[4]}}, {{f[5], f[6]}, {f[7], f[8]}}} を作れ．

■ MapAll //@

リストあるいは式のあらゆる部分に関数を施すときには MapAll を用いる．

● MapAll[f, expr] または f //@ expr は，式 expr のすべてのレベルのすべての要素に関数 f を施した式を返す．

```
In[9]:= MapAll[f, {{1, 2, 3}, {4, 5, 6}}]

Out[9]= f[{f[{f[1], f[2], f[3]}], f[{f[4], f[5], f[6]}]}]
```

■ MapAt

リストあるいは式の特定の要素に関数を施すときには MapAt を用いる.

- MapAt[f, expr, {part1, part2, ...}] は，式 expr の part1, part2, ... にある各部分に関数 f を施した式を返す.

```
In[10]:= MapAt[f, {1, 2, 3, 4}, {{2}, {3}}]

Out[10]= {1, f[2], f[3], 4}

In[11]:= MapAt[f, {{1, 2}, 3, {4, {5, 6}}}, {{1, 2}, {3, 2, 1}}]

Out[11]= {{1, f[2]}, 3, {4, {f[5], 6}}}
```

MapAt で使用する位置指定の記法は Position などが返す値の記法と同じである．したがって，Position が返した値をそのまま MapAt にわたすことができる．次の計算例は多重リストの中のすべての a に関数 f を施すものである.

```
In[12]:= Position[{{a, b}, a, {b, {a, b}, a}}, a]

Out[12]= {{1, 1}, {2}, {3, 2, 1}, {3, 3}}

In[13]:= MapAt[f, {{a, b}, a, {b, {a, b}, a}}, %]

Out[13]= {{f[a], b}, f[a], {b, {f[a], b}, f[a]}}
```

式においても同様に使える．次の計算例は式 y にあるすべての x に平方根関数 Sqrt を施すものである.

```
In[14]:= y = (1 + x + Sin[x]) / (1 - x^2)

Out[14]= (1 + x + Sin[x]) / (1 - x^2)

In[15]:= Position[y, x]

Out[15]= {{1, 1, 2, 2, 1}, {2, 2}, {2, 3, 1}}

In[16]:= MapAt[Sqrt, y, %]

Out[16]= (1 + Sqrt[x] + Sin[Sqrt[x]]) / (1 - x)

In[17]:= Clear[y]
```

■ インデックスをつけて写像する

■ MapIndexed

リストの中の何番目の要素であるか，というインデックスをつけて関数を施したいときもある．そのようなときには MapIndexed を用いる.

- MapIndexed[f, {a, b, ...}] は {f[a, {1}], f[b, {2}], ...} を返す.

```
In[18]:= MapIndexed[f, {a, b, c, d}]

Out[18]= {f[a, {1}], f[b, {2}], f[c, {3}], f[d, {4}]}
```

■ 複数個のリストの要素を組にして関数を施す

■ MapThread

{a, b, c} と {x, y, z} という二つのリストに対して，リストの中で同じ位置にある要素 a,x b,y c,z を糸で縫い取る (thread) ように組にして {f[a, x], f[b, y], f[c, z]} というリストを作りたいとする．この操作を f を二つのリストに縫い込む，あるいは f で二つのリストを縫い取るという．これを行うには MapThread を用いる.

- MapThread[f,{list1,list2,...}] は list1,list2,... の同じ位置にある要素 e1,e2,... の組に対して関数 f を施した f[e1,e2,...] のリストを返す．ただし，list1,list2,... はすべて同じ長さでなければならない.

```
In[19]:= MapThread[f, {{a, b, c}, {aa, bb, cc}}]

Out[19]= {f[a, aa], f[b, bb], f[c, cc]}

In[20]:= MapThread[Power, {{1, 2, 3, 4}, {5, 6, 7, 8}}]

Out[20]= {1, 64, 2187, 65536}
```

■ 引き数に与えられたリストを関数で縫い取る

■ Thread

f[{a,b,c},{aa,bb,cc}] という形で関数 f に二つのリストが引き数として与えられているとき，f でそれらのリストを縫い取り {f[a,aa],f[b,bb],f[c,cc]} というリストを得たいとする．そのためには，これに Thread を施せばよい.

- Thread[f[{a, b, ...}, {aa, bb, ...}, ...]] は {f[a, aa, ...], f[b, bb, ...], ...} を返す．ただし， {a, b, ...}, {aa, bb, ...}, ... はすべて同じ長さのリストである必要がある.

```
In[21]:= Thread[f[{a, b, c}, {aa, bb, cc}]]

Out[21]= {f[a, aa], f[b, bb], f[c, cc]}
```

関数 f の引き数の中にリストでない要素があった場合，それは他のリストと同じ個数だけその要素が並んだリストとみなされる.

```
In[22]:= Thread[f[{a, b, c}, x, {aa, bb, cc}]]

Out[22]= {f[a, x, aa], f[b, x, bb], f[c, x, cc]}
```

練習問題： {f[a, x, y, aa], f[b, x, y, bb], f[c, x, y, cc]} というリストを Thread を用いて作れ.

■ 自動的にThreadを行うListable属性

Plus,Times,Power,Sin,Cos,Log などの組み込みの数学関数のほとんどは Listable 属性と呼ばれる属性をもっている．この属性をもつ関数は，引き数にリストが与えられると，自動的に Thread が適用される.

```
In[23]:= Plus[{1, 2, 3}, {4, 5, 6}]

Out[23]= {5, 7, 9}
```

これは {1,2,3}+{4,5,6} と同じである.

```
In[24]:= Times[{1, 2, 3}, {4, 5, 6}]

Out[24]= {4, 10, 18}
```

これは {1, 2, 3}*{4, 5, 6} あるいは {1, 2, 3} {4, 5, 6} と同じである.

```
In[25]:= Power[{1, 2, 3}, {4, 5, 6}]

Out[25]= {1, 32, 729}
```

これは {1, 2, 3}^{4, 5, 6} と同じである.

```
In[26]:= Sin[{1, 2, 3}]

Out[26]= {Sin[1], Sin[2], Sin[3]}
```

Thread と同様の働きをするので，リストでない要素は，その要素が並んだリストとして扱われる.

```
In[27]:= Plus[{1, 2, 3}, 4]

Out[27]= {5, 6, 7}
```

これは {1, 2, 3} + 4 と同じである.

■ すべての組み合わせの組に関数を施す

複数個のリストから 1 個ずつ要素を選び組みにして関数を施す，ということをすべての組み合わせにわたっておこなうには， Outer を用いればよい．この操作は，外積という演算の一般化になっていることから， Outer と名前が付いている.

■ Outer

● Outer[f, list1, list2, ...] は複数個のリスト list1, list2, ... から一つずつ要素 e1, e2, ... を選び，それらを要素とし f を頭部とする式 f[e1, e2, ...] を作り，それをすべての組み合わせにわたって多重リストにしたものを返す．その多重リストのレベル i では ei が変化する．

```
In[28]:= Outer[f, {a, b, c}, {w, x, y, z}, {1, 2}]

Out[28]= {{{f[a, w, 1], f[a, w, 2]}, {f[a, x, 1], f[a, x, 2]},
           {f[a, y, 1], f[a, y, 2]}, {f[a, z, 1], f[a, z, 2]}},
          {{f[b, w, 1], f[b, w, 2]}, {f[b, x, 1], f[b, x, 2]}, {f[b, y, 1], f[b, y, 2]},
           {f[b, z, 1], f[b, z, 2]}}, {{f[c, w, 1], f[c, w, 2]},
           {f[c, x, 1], f[c, x, 2]}, {f[c, y, 1], f[c, y, 2]}, {f[c, z, 1], f[c, z, 2]}}}

In[29]:= TableForm[%]

Out[29]//TableForm=
       f[a, w, 1]      f[a, x, 1]      f[a, y, 1]      f[a, z, 1]
       f[a, w, 2]      f[a, x, 2]      f[a, y, 2]      f[a, z, 2]
       f[b, w, 1]      f[b, x, 1]      f[b, y, 1]      f[b, z, 1]
       f[b, w, 2]      f[b, x, 2]      f[b, y, 2]      f[b, z, 2]
       f[c, w, 1]      f[c, x, 1]      f[c, y, 1]      f[c, z, 1]
       f[c, w, 2]      f[c, x, 2]      f[c, y, 2]      f[c, z, 2]
```

練習問題： Outer[a, Range[2], Range[3], Range[4]] は Array[a, {2, 3, 4}] と同じ結果を返すことを確認せよ．

■ 関数を繰り返し施して得られる値のリストを作る

関数を繰り返し施して得られる値をリストにしたいときがある．そのようなときには NestList, FoldList, ComposeList を用いる．

■ NestList

ある値から始めて，同じ関数を繰り返し施すことにより得られる値のリストを作るには NestList を用いる．

● NestList[f, x, n] は {x, f[x], f[f[x]], ..., f[f[...f[x]...]]} を返す．ただし，結果の最後の要素は f が n 回施されている．

```
In[30]:= NestList[f, x, 4]

Out[30]= {x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}

In[31]:= NestList[Cos, 1, 4]

Out[31]= {1, Cos[1], Cos[Cos[1]], Cos[Cos[Cos[1]]], Cos[Cos[Cos[Cos[1]]]]}
```

これは 1 から始めて余弦関数を繰り返し施したリストである．その近似値は次のようになる．

```
In[32]:= N[%]

Out[32]= {1., 0.540302, 0.857553, 0.65429, 0.79348}
```

NestList[f, x, n] で得られるリストの最後の要素は Nest[f, x, n] で得られる.

Nest[f, x, n] は x に f を n 回続けて施したものを返す.

```
In[33]:= Nest[f, x, 4]

Out[33]= f[f[f[f[x]]]]
```

■ FoldList

ある値から始めて，同じ関数を複数のパラメータと共に施すことにより得られる値のリストを作るには FoldList を用いる.

● FoldList[f, x, {a, b, ...}] は {x, f[x, a], f[f[x, a], b], ...} を返す.

```
In[34]:= FoldList[f, x, {a, b, c}]

Out[34]= {x, f[x, a], f[f[x, a], b], f[f[f[x, a], b], c]}
```

次の計算例は 3 に 2, 4, 5, 9 を次々と加えたときの値のリストである.

```
In[35]:= FoldList[Plus, 3, {2, 4, 5, 9}]

Out[35]= {3, 5, 9, 14, 23}
```

FoldList[f, x, {a, b, ...}] で得られるリストの最後の要素は Fold[f, x, {a, b, ...}] で得られる.

```
In[36]:= Fold[f, x, {a, b, c}]

Out[36]= f[f[f[x, a], b], c]
```

■ ComposeList

ある値から始めて，複数の関数を次々と施すことにより得られる値のリストを作るには ComposeList を用いる.

● ComposeList[{f1, f2, ...}, x] は {x, f1[x], f2[f1[x]], ...} を返す.

```
In[37]:= ComposeList[{f, g, h}, x]

Out[37]= {x, f[x], g[f[x]], h[g[f[x]]]}
```

■ FixedPoint

● FixedPoint[f, x] は x から始めて f を繰り返し施し，その関数値がある値に留まり変化しなくなればその値を返す.

```
In[38]:= FixedPoint[Cos, 1.0]

Out[38]= 0.739085
```

練習問題： NestList を用いて 1.0 に Cos を繰り返して施したリストを作り，それが 0.739085 に近づいて行く様子を見よ．

■ 演習問題

[7-1] 1 から 100 までの整数の和と積とを， Range, Apply, Plus, Times を用いて求めよ．

[7-2] 1 から 100 までの整数の平方根の近似値のリストを Range, Sqrt, Map, N を用いて求めよ．またその総和を Apply, Plus を用いて求めよ．

[7-3] {f[1, 10], f[2, 9], f[3, 8], f[4, 7], f[5, 6], f[6, 5], f[7, 4], f[8, 3], f[9, 2], f[10, 1]} を MapThread, Range, Reverse を用いて作れ．

[7-4] 次の三つの式は，二つのサイコロの目の和を表す 6 × 6 の配列 {{2,3,4,5,6,7}, {3,4,5,6,7,8}, {4,5,6,7,8,9}, {5,6,7,8,9,10}, {6,7,8,9,10,11}, {7,8,9,10,11,12}} を作るためのものである．それぞれを完成せよ．

(1) Table[i+j, ...
(2) Array[Plus, ...
(3) Outer[Plus, ...

[7-5] 3個のサイコロの目の和を表す 6 × 6 × 6 の配列を作れ．その中で値が 10 であるものは何個あるかを， Mathematica の関数を使って調べよ．

[7-6] 1 から始めて余弦関数 cos を繰り返し施した値の近似値のリストを作れ．その値は収束しそうか．もしそうならば，その値の概略値は何か．