付録**A.**関数型表記に関する解説

松田裕幸 matsuda@symbolics.jp (2011.4.48)

レシピ**A.1 #**と**&**の使い方

- 問題：『**Mathematica**クックブック』 を買ったけど関数型表記の意味と使い方がよく分からない。特に、**#** と **&** をどう使ったらいいんだろう。
- 解：まず基本的な話をします。**Mathematica**の典型的な関数を参考に説明します。

In[1]:=
```
f[a_, b_] := a + b;
f[3, 4]
```

Out[2]= 7

次にこの関数定義の表現を分解してみます。

```
f          ... 関数名
[a_, b_]   ... 引数
:=         ... 定義
a + b      ... 関数本体
```

これらの中で、どこが一番重要でしょう？もちろん、関数本体（a+b）です。しかし、実際に変数aと変数bに値が決まらなければ計算はできません。値はどこから持ってくるのでしょう？引数からです。

では、引数の役割はなんでしょう？いま、たまたまaとbという名前を使いましたが、これがarg1とarg2でも変わらないことは分かっていただけると思います。もちろん、その場合、関数本体の方もarg1+arg2に変更する必要があります。

ということは、引数名は任意の名称が使えることが分かります。もう一つ引数には重要な情報が含まれています。それは位置に関するものです。変数aは第1引数、変数bは第2引数に対応しています。

整理すると引数の名称は何でもいい、しかし、位置情報は守らなければならない。ということで、上記関数定義を次のように書き換えてみます。

```
f          ... 関数名
[#1, #2]   ... 引数
:=         ... 定義
#1+#2      ... 関数本体
```

ここまで来ると、実は引数部分は不要ではないかと思えませんか？つまり関数本体だけでいいのではないかということです。実際、そうなります。ただし、関数であることを示すために最後に&を付けます。

```
#1+#2&
```

こういう形の関数を純関数 pure function、と呼びます。また、#をplace holder（収納子）呼びます。

In[3]:=
```
#1 + #2 &[3, 4]
```

Out[3]= 7

- 解説

Lispの世界にはラムダ式（あるいはラムダ関数）という概念がはるか昔から存在していました。上記Mathematicaの式をラムダ式で表すとこんな感じになります。

```
(lambda (a b) (+ a b))
```

Mathematicaでも似たような表記が存在します。ただし、これを積極的に使うことはあまりありませんが。

In[4]:=
```
Function[{a, b}, a + b][3, 4]
```

Out[4]=
```
7
```

さて、ずっとほっておいた関数名はどうなったのでしょうか？すっかり忘れていました。関数名って何故必要なのでしょうか？それは定義に対し名前を付けなければ参照できないのだから当然だろうとうい反論が出てきそうです。しかし、本当にそうでしょうか？

Mathematicaではさきほどの純関数を「任意」の場所で使うことができます。このスタイルを on the flyと呼びます。たとえばこんな感じです。

In[5]:=
```
data = RandomInteger[{1, 100}, {10}];
Sort[data, #1 > #2 &]
```

Out[6]=
```
{81, 77, 75, 51, 35, 29, 28, 20, 14, 5}
```

これに対し従来のやり方は、次のように比較関数を定義し、それに「名前」を付け、それを使うというものでした。

In[7]:=
```
greater[a_, b_] := a > b;
Sort[data, greater[#1, #2] &]
```

Out[8]=
```
{81, 77, 75, 51, 35, 29, 28, 20, 14, 5}
```

あるいは次のように書きます。

In[9]:=
```
Sort[data, greater]
```

Out[9]=
```
{81, 77, 75, 51, 35, 29, 28, 20, 14, 5}
```

しかし、すでにみたように関数名を付けずにいきなり関数本体を、式の中にはめ込むことが可能です。これはLispのラムダ式も同じです。

それでも名前を付けておくと便利な時もあります。プログラムの意味がつかみやすくなるからです。

In[10]:=
```
second = #[[2]] &;
data = {a, b, c, d, e};
{First[data], second[data]}
```

Out[12]=
```
{a, b}
```

では実際に『クックブック』の中から例を持ってきて解説してみます。

例1

In[279]:=
```
NestList[Translate[Rotate[#, -d], {1.5, 0}] &, shape, 3] // TableForm
```

Out[279]//TableForm=
```
shape
Translate[Rotate[shape, -d], {1.5, 0}]
Translate[Rotate[Translate[Rotate[shape, -d], {1.5, 0}], -d], {1.5, 0}]
Translate[Rotate[Translate[Rotate[Translate[Rotate[shape, -d], {1.5, 0}], -d], {1.5, 0}],
```

この例のように未評価のシンボルを利用するとことで、#にどんな値が入ってくるのか確認できます。この例の場合は、#にshapeが代入されています。

それでもよく分からないということであれば、&（ここまでが関数）までを未評価のシンボルfに置き換えてみます。いかがでしょう？

In[280]:=
```
Clear[f];
NestList[f[#] &, shape, 3] // TableForm
```

Out[281]//TableForm=

shape
f[shape]
f[f[shape]]
f[f[f[shape]]]

例２

In[5]:=
```
data = RandomInteger[{1, 100}, {10}];
Sort[data, #1 > #2 &]
```

Out[6]= {81, 77, 75, 51, 35, 29, 28, 20, 14, 5}

もしかしたこのコード、よく考えると不思議にみえるかもしれません。なぜなら#1, #2って何に対応するんだろうという疑問です。並び替え(Sort)アルゴリズムは無数にありますが、通常、２つのデータの大小を比較します。アルゴリズムによって比較されるデータは隣合わせになることもあるし、離れた２つのデータが比較されることもありますが、どちらしても「２つ」のデータを比較する点には変わりありません。

その比較されるデータを#1と#2で表します。

同じ並び替え問題でも少し複雑な問題を考え見ますが、今までと同様簡単に対応できます。

In[13]:=
```
data = {{"桃太郎", 5}, {"ウルトラマン", 20}, {"一寸法師", 3}, {"花咲爺", 60}};
Sort[data]
```

Out[14]= {{桃太郎, 5}, {花咲爺, 60}, {一寸法師, 3}, {ウルトラマン, 20}}

何も考えないと漢字コードに従って並び替わります。では、年齢に応じて並び替えたいとしたらどうすればいいでしょう？

In[15]:=
```
Sort[data, #1[[2]] > #2[[2]] &]
```

Out[15]= {{花咲爺, 60}, {ウルトラマン, 20}, {桃太郎, 5}, {一寸法師, 3}}

例３

In[17]:=
```
Total[MapIndexed[#1 x^First[#2] &, {2, 0, 7, 3}]]
```

Out[17]= $2\,x + 7\,x^3 + 3\,x^4$

この場合も理解が難しければすでに試みたのと同じ方法で組み込み関数を未定義シンボルに置き換えてみます。

In[19]:=
```
Total[MapIndexed[#1 x^First2[#2] &, {2, 0, 7, 3}]]
```

Out[19]= $2\,x^{\text{First2}[\{1\}]} + 7\,x^{\text{First2}[\{3\}]} + 3\,x^{\text{First2}[\{4\}]}$

いかがでしょう？#1には順次、2,0,7,3が代入されているのがわかります。一方、#2にはなんか変なものが入っています。MathematicaではSortの例でも見たようにシステム側が持っている内部情報を#変数を使い、ユーザに開放しています。この#2もその１つです。

MapIndexedは写像する際、その位置も記憶しています。それを{}のに包んで返してきます。したがって、実際に位置を知るにはその中身を取る必要があり、Fristを使っています。上記コードは次のようにも書き直せます。

In[20]:=
```
Total[MapIndexed[#1 x^#2[[1]] &, {2, 0, 7, 3}]]
```

Out[20]= $2\,x + 7\,x^3 + 3\,x^4$

レシピA.2 @

- 問題： @って何？そもそもなんでこんな記号が必要なのか分からない。
- 解：説明難しいです（笑）。
- 解説：

本来@は関数Compositon（合成）の中置(infix)表記です。

```
In[31]:= Clear[f];
         f@g@h[a, b]
Out[32]= f[g[h[a, b]]]

In[2]:= Composition[f, g, h][a, b]

Out[2]= f[g[h[a, b]]]

In[3]:= Sqrt@Plus[3^2 + 4^2]

Out[3]= 5

In[9]:= data = RandomInteger[{1, 1000}, {10}]
        Length@Select[data, # > 500 &]
Out[9]= {343, 198, 938, 299, 16, 408, 729, 995, 171, 941}

Out[10]= 4
```

ではよく見かける@@はなんでしょう？実は@@@もあるのですが、これは後で説明します。いきなりですが、こんな例を載せます。

```
In[13]:= f @@ {a, b}

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

これを正確に書き直すと次のようになります。

```
In[15]:= f @@ List[a, b]

Out[15]= f[a, b]
```

@は合成の役割を果たすと書きました。では、@@で2回合成すると考えればいいのでしょうか？なぜ@@を2つ合わせることをWolframが思いついたのかは不明ですが、ここでは合成を2回行うという意味ではなく、後者を前者で乗っ取る意味に解釈します。つまり、Listをfで置き換えます。

```
In[14]:= f[a, b]

Out[14]= f[a, b]
```

Mathematicaにはすべての関数に前置、中置、後置の３種類の表現を持ちます。もちろん@@は中置形式です。では、この前置形式はなんでしょう？ Applyです。

```
In[16]:= Apply[f, {a, b}]

Out[16]= f[a, b]
```

Applyの例を２つほど挙げてみます。

```
In[19]:= Plus[{1, 2, 3}] (*エラー*)

Out[19]= {1, 2, 3}

In[22]:= {Plus @@ {1, 2, 3}, Apply[Plus, {1, 2, 3}], Plus[1, 2, 3]}

Out[22]= {6, 6, 6}
```

あるいはこんな例も。

```
In[23]:= data = {zaiko[10, 20, 30], zaiko[550, 20], zaiko[40, 2222]}

Out[23]= {zaiko[10, 20, 30], zaiko[550, 20], zaiko[40, 2222]}
```

In[24]:=
```
Map[Apply[Plus, #] &, data] (* Mapについては次に説明します *)
```
Out[24]=
```
{60, 570, 2262}
```

では@@@はなんでしょう？これもMapを解説した後に説明します。

レシピA.3 /@

- 問題：**Mathematica**の関数型の本質は**Map**（写像）だとよく聞くけど、今ひとつ理解できない。
- 解：**Map**の働き自体は単純です。

In[25]:=
```
Map[f, {a, b, c, d}]
```
Out[25]=
```
{f[a], f[b], f[c], f[d]}
```

Mapの中置表記が /@です。

In[26]:=
```
f /@ {a, b, c, d}
```
Out[26]=
```
{f[a], f[b], f[c], f[d]}
```

いままでのをおさらいするとこんなのも書けます。

In[27]:=
```
#^2 & /@ {1, 2, 3, 4}
```
Out[27]=
```
{1, 4, 9, 16}
```

In[29]:=
```
Map[#^2 &, {1, 2, 3, 4}]
```
Out[29]=
```
{1, 4, 9, 16}
```

あるいは

In[28]:=
```
#[Pi / 4] & /@ {Sin, Cos, Tan}
```
Out[28]=
$$\left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 1\right\}$$

In[30]:=
```
Map[#[Pi / 4] &, {Sin, Cos, Tan}]
```
Out[30]=
$$\left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 1\right\}$$

レシピA.4 @@@

ではずっととっておいた@@@の解説をしてこのレシピを閉めます。その前に次のようなMapの例を考えてみます。

In[41]:=
```
data = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
Map[f, data]
```
Out[42]=
```
{f[{1, 2, 3}], f[{4, 5, 6}], f[{7, 8, 9}]}
```

じつはこれは期待する結果ではなく、本来はf[1,2,3]を期待したとします。しかし、これはMapだけではできません。

In[44]:=
```
Apply[f, data, {1}] (* {1}は対象リストの操作対象レベルに対応 *)
```
Out[44]=
```
{f[1, 2, 3], f[4, 5, 6], f[7, 8, 9]}
```

こういうケースは非常に多いのでこれを簡単に表現できるよう@@@が用意されています。

In[37]:=
```
f @@@ {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```
Out[37]=
```
{f[1, 2, 3], f[4, 5, 6], f[7, 8, 9]}
```