関数の定義 (応用編)

関数定義の復習

• 自己定義関数の名前は小文字で始めるとよい．関数定義の際には _ と ： を忘れないようにする．

条件分岐を用いた関数定義

■ 関数 $f : \mathbf{N} \to \mathbf{N}$ を

$$f(x) = \begin{cases} x/2, & x \text{ が偶数のとき} \\ 3x+1, & x \text{ が奇数のとき} \end{cases}$$

と定義しよう．これは，Mathematica 上では次のようにする[†]．

```
f[x_] := If[EvenQ[x], x/2, 3x + 1]
```

(1)

一般に，If[condition, exp1, exp2] は condition が True を返すときに exp1 を，False を返すときに exp2 を評価する．If[condition, exp1] とすると，condition が True を返すときにのみ exp1 を評価し，False を返すときには何もしない．次のようになれば，定義が成功している．

```
f[10]
5

f[%]
16
```

(2)

■ **(変数の型による分岐)** 次のようにしてみよ[‡]．

```
g[x_Integer] := x + 2

g[x_Rational] := Numerator[x] + Denominator[x]

g[x_Real] := Floor[x]

g[x_Complex] := Abs[x]
```

(3)

これらの意味は，次を実行してみれば推測できるであろう (それでも分からなければ，ヘルプで調べてみよ).

```
g[0]

g[3/100]

g[1.2]

g[1 + I]
```

(4)

[†]英単語の学習．even=偶数.

[‡]integer=整数，rational number=有理数，real number=実数，complex number=複素数.

Mathematica では意識することはまれであるが，プログラム言語を扱う際に，数の型を認識しておくことは重要である．例えば先の例で，3/100 は Rational，1.2 は Real である．この辺り，数学とはやや言葉遣いが異なる (やや不正確な言い方をすれば，分数が Rational で小数が Real である)．また，数学的には「整数」は必ず「有理数」でもあるが，計算機の認識では Integer はあくまでも Integer であって，同時に Rational であることはない．数の型は関数 Head で確認することができる．数学的な知識とは必ずしも相容れないので注意せよ．

```
Head[Pi]

Head[N[Pi]]

Head[Sqrt[2]]

Head[I]

Head[Infinity]

Head[Pi/E]

Head[1/E]

Head[1 + E]
```

(5)

上の例で，Integer, Rational, Real, Complex 以外の出力が返ってきたものは，Mathematica ではある意味「数」とも認識されていない．関数 NumberQ は頭部 (Head) がこの四つのいずれかである場合にのみ True を出力する．一般に，**頭部**とは Mathematica 内での認識 (関数 FullForm で確認できる) の先頭部分のことである．これらの意味は，次を実行してみれば明らかになるであろう．

```
NumberQ[1 + E]

NumberQ[N[1 + E]]

FullForm[1 + E]
```

(6)

局所変数

■ やや話が脱線するが，Collatz 予想について簡単に述べる．それは，(1) で定義された関数を繰り返し適用すれば，どのような自然数もいつかは 1 になるであろう，という予想である．例えば，3 から出発したなら，3 → 10 → 5 → 16 → 8 → 4 → 2 → 1 という具合である．実験的に正しいと思われているが，未だに証明はされていない[§]．

さて，1 に至る途中経過を出力する関数を定義したいとする．次が一つの答えである．

```
collatz[x_] := (y = x; While[y > 1, y = f[y]; Print[y]])

collatz[3]
```

(7)

一応確認するなら，まず y に x の値を代入し，y が 1 より大なる限り，y を f[y] で置き換え，その値を出力することを繰り返せ，という命令である．collatz[3] でこの関数がうまく動くことを確かめてみよ．

[§]少し考えてみるだけならともかく，Collatz 予想を証明しようなどと本気で思わぬこと．時間の無駄になること請け合いである．

ヒント．collatz という綴りをタイプするのは面倒だろうが，一度定義してしまえば，次は c を入力してすぐに補完機能 (Ctrl+k) を利用すればよい．組み込み関数は全て大文字で始まるのだから，小文字の c で始まるのは今のところここで定義したものしか無いはずである．

(7) が問題なのは，変数 y を用いていることである．(7) を実行した後，y には 1 が代入されているはずである．

```
y
1
```

(8)

他の所でも y を用いたい場合，これでは少し困ったことになる．そこで，**自己定義関数内では局所変数を用いることを強く勧める**．そのためには関数 Module を用いる．用法は次の通り．

Module[{局所変数の宣言}, ...]

```
y = .

collatz[x_] := Module[{y}, y = x; While[y > 1, y = f[y]; Print[y]]]

collatz[3]

y
```

(9)

今度は y に何も代入されていないことが分かるであろう．局所変数は関数内のみで用いられ，他に影響を及ぼさない．

While という関数は注意して使わなければならない．安易な使い方をすると無限ループに陥ってしまう．実は上の (7) も，Collatz 予想が証明されていないのだから無限ループに陥る可能性がある．次は，1000 回繰り返すと error と表示してストップするように修正したものである．

```
collatz[x_] := Module[{y, c},
   y = x; c = 0;
   While[y > 1,
     y = f[y]; Print[y];
     c = c + 1; If[c == 1000, Print["error"]; y = 1]]]
```

(10)

練習問題 1

(1) の代わりに

$$f(x) = \begin{cases} x/2, & x \text{ が偶数のとき} \\ 5x+1, & x \text{ が奇数のとき} \end{cases}$$

と定義し，これについて Collatz 予想と同様のことが成り立つか，実験して調べよ．

リスト

■ Mathematica ではオブジェクト (数や関数など) を一つにまとめて扱うことができる．これを**リスト**という．数学における集合のようなものと思えばよい．ただし，数学の集合とは次の点で異なる．

• 数学では無限集合が特に重要であるが，無限個のオブジェクトを含むリストは扱えない．
• 例えば，集合としては {1,1} と {1} は区別しないが，リストとしては区別する．また，リストはオブジェクトの並ぶ順番が異なっても区別する．

以下，印刷上 1 と区別するために L (エル) は大文字で表すが，実際は小文字で書く方が望ましい．

```
L = {4, 1, 2, 1}
{4, 1, 2, 1}

Sort[L]
{1, 1, 2, 4}

Union[L]
{1, 2, 4}
```

(11)

リストは括弧 { } でくくり，オブジェクト間はコンマ , で区切る．Sort と Union は共にリストを加工する関数で，Sort は通常の順序に並べ替えをし，Union は並べ替えの上，重複するものは取り除く．その他，リストを加工する関数は沢山あるが，必要になったときにヘルプで調べて頂きたい．

■ リストを用いて，関数 collatz を改良しよう (Append[L,y] は「リスト L の末尾にオブジェクト y を付け加えたリスト」を表す)．このように，リストを用いれば，Print で出力するのに比べてスペースを省略できるし，その後の加工も容易である．Length はリストの長さ，Max はリスト内の最大数を出力する．なお，27 は関数 f で 1 に達するまでに比較的長くかかるので有名である．

```
collatz[x_] := Module[{y, L}, y = x; L = {x}; While[y > 1, y = f[y]; L = Append[L, y]]; L]

L = collatz[27]

Length[L]

Max[L]
```

(12)

プログラムを組もう

以上で，プログラムに必要な最低限の説明を終えたつもりである．ここまでの理解が十分で，かつ根気さえあれば，ボーリングで倒したピンの数を入力すればスコアが出力されるプログラムを書いたりできるだろう．

プログラムの技術を磨くには，人のプログラムを読んだり，自分でいろいろと試行錯誤することが必要である．最初のうちは (熟練したプログラマーでさえ) 思うようにプログラムが動いてくれず，その原因もなかなか見つからなかったりする．実際，プログラマーは，プログラムを組むよりもその修正 (デバッグ) により時間をかけるという．たった 5 回のこの授業ではそこまでの訓練はできないので，やる気のある者は関連する図書 (図書館に行けば Mathematica と名の付いた本がいくらでもある) を読むなどして自分で練習をするとよい．

次は，サンプルプログラムとその実行例である．平面上の 2 点を与えると，その 2 点を通る直線をプロットする．

```
plotline[a_, b_] := Module[{t},
    t = (b[[2]] - a[[2]])/(b[[1]] - a[[1]]);
    Plot[t(x - a[[1]]) + a[[2]], {x, -5, 5}]]

plotline[{1, 1}, {2, 3}]
```

(13)

練習問題 2

上のプログラムでは，2 点の x 座標が等しいとエラーとなる．その場合には `error` と表示されて終了するようにプログラムを修正せよ．

再帰的関数定義

■ Fibonacci 数列 `a[n]` を定義しよう．Mathematica は次のような直感的な関数の定義を許してくれる．

```
a[0] := 0; a[1] := 1; a[n_] := a[n - 2] + a[n - 1]

a[5]
5
```

(14)

`Timing` で計算にかかった時間を同時に出力させられる．`a[30]` の計算に意外に時間がかかることが分かる．

```
Timing[a[30]]
{4.156 Second, 832040}
```

(15)

• (ヒント) 上の例で括弧が多すぎて見にくいな，と感じたら，`a[30] //Timing` のようにしてもよい．これは `Timing` に限らず，任意の関数で使える方法である．例えば `0 //Cos` など．

なぜ `a[30]` の計算にこんなに時間がかかるのであろうか．29 回足し算をするだけではないのか．実はこのとき，Mathematica はそれまでの計算結果を覚えておくことはせず，必要になった時点でもう一度計算をやり直しているのである．

> (Mathematica の心の動き) `a[30]` を計算しろだって？じゃあ `a[28]` と `a[29]` が必要だな．`a[28]` を計算するには `a[26]` と `a[27]` が必要だな．(中略) よし，分かった! `a[28]` は 317811 だ．さて，最初に戻って `a[29]` も計算するんだったな．それには `a[27]` と `a[28]` が必要だ．やれやれ忙しいな．

まあ，実際には忙しいとも思ってないだろうし，命令がそうなのだから仕方がないのである．`a[n]` を計算させると，`a[n-2]` は 2 回，`a[n-3]` は 3 回，`a[n-4]` は 5 回計算される (これ自身も Fibonacci 数列である!).

■ ではどのようにするべきだろうか．次のようにすれば，それまでの計算結果を覚えておくので一瞬で済む．

```
b[0] := 0; b[1] := 1; b[n_] := (b[n] = b[n - 2] + b[n - 1])

Timing[b[30]]
{0. Second, 832040}
```

(16)

この方法を用いると，一度計算したものはカーネルを閉じるまで覚えている．よって，Fibonacci 数を何度も参照する場合には計算時間を節約できる．ただし，デメリットとして，メモリを消費するということと，次のようなエラーが出てしまう 2 点がある．

```
b[1000]
$RecursionLimit :: reclim : 最大再帰回数 256 を超えています. 詳細
$RecursionLimit :: reclim : 最大再帰回数 256 を超えています. 詳細
$RecursionLimit :: reclim : 最大再帰回数 256 を超えています. 詳細
General :: stop : 計算中, $RecursionLimit :: reclim のこれ以上の出力は表示されません. 詳細
```

(17)

$RecursionLimit は無限ループを避けるために定められた，再帰的計算の最大回数である．必要ならばこの値を書き換えることはできるが，面倒である．再帰的関数定義というテーマからは外れるが，次のようにプログラムを組めば問題は解決する．

```
c[n_] := Module[{m1, m2, m3},
   If[n == 0, Return[0]]; If[n == 1, Return[1]];
   m1 = 0; m2 = 1;
   Do[m3 = m1 + m2; m1 = m2; m2 = m3, {i, 1, n - 1}];
   m3]

c[1000]//Timing
{0. Second,
   43466557686937456435688527675040625802564660517371780402481729089536555417949051 8
   90403879840079255169295922593080322634775209689623239873322471161642996440906533 1
   87938298969649928516003704476137795166849228875}
```

(18)

課題

■ $a_{n+2} = a_{n+1} + 2a_n$, $a_0 = 0$, $a_1 = 1$ で定義される数列の一般項を計算するプログラムを作りなさい．なお，計算が速ければ速いほど評価は高い．

■ 上の問題に対する解答が math5.nb の最後に位置するようにしてセーブせよ．次にメーラーを起動し，今日の内容について何か感想を記せ (一言でもよい).

■ メールの Subject は「学籍番号+math5」とせよ．例えば，学籍番号が 6101999 ならば，`6101999+math5` となる．署名を付け，math5.nb を添付し，今日中に `j-goto` に送ること．