目次へ

単純な計算と変数

足し算、引き算、その他の計算

しばらく、計算ばっかりにこだわってドリル的に進めてみます。さっきまでと同様に、pythonの対話シェルを開始してから下の例を順次マネしてみてください。

足し算や引き算は、直感どおりでたぶん大丈夫でしょう。

```
>>> print 4379 + 234890
237269
```

```
>>> print 45-345
-300
```

プラス記号やマイナス記号のまわりに、空白はあってもなくても構いません。

```
>>> print 1+2+3+4+5
15
```

いくつも項が増えたとしても大丈夫です。

掛け算の記号は * 、割り算の記号は / です。

```
>>> print 9*9
81
```

```
>>> print 1000/4
250
```

楽なもんです。

ところで、対話シェル（＞＞＞に打ち込むやつ）のときは、print を省略してもいいです。理由はちょっと難しくなるかもしれませんので、今は詳しい説明しませんが。

だから、

```
>>> print 1000/4
```

は、

```
>>> 1000/4
```

とだけしても結果は同じです。

割り算の例は、わざと割り切れる数で示しました。小数が生じたり、割り切れなくなる場合はどんなふうになるでしょう。

```
>>> 10 / 4
2
>>> 10 / 3
3
```

む、これは、期待と違う数字ですねえ。2.5とか、3.333333… とか表示してほしいところです。

これはpython固有の特徴なんですが、整数 割る 整数 のときは、割り算して小数部分を切り捨てた結果を教えてくれます。なにこれ不便…と言われそうですが、まあこれはこれで色々都合がよかったりすることも多いのです。そのうち分かっていくでしょう。

小数を含んだ実数の結果が必要なら、割り算の書き方にちょっと工夫をします。

```
>>> 10.0 / 4
2.5
>>> 10.0 / 3
3.3333333333333335
```

これで大体、当初の期待どおりの結果が出てきました。整数 割る 整数 が自動的に整数に丸められてしまうというのなら、実数 割る 整数 といった書き方にしてみたら どうか、ということです。10じゃなくてわざわざ10.0と書いたんだから、いかにも整数じゃないでしょ。

このときはpythonも、「おお、実数の答えを期待しているのか」と気づいて、大体期待にこたえてくれます。今のところは、このくらいの理解をしておいてください。

ところで、10/3は、3.3333333333… となるべきであって、やむなく途中で区切るにしても、…3335 といった感じに5で終わるのは変じゃないか、と感じることでしょう。「素」の状態のpythonは、実数の扱いがそんなに精密なわけではないのです。下のような入力をしてみたとき

も、明らかに数字変わってんじゃないか、と言われそうです。まあ、ご愛嬌と考えておいてください。

```
>>>0.1
0.10000000000000001 （えー）
```

もちろん、整数を扱うときにこんな変なことは起こりませんので安心してください。一億円のつもりで入力したら一億と一円になっていた、とかそういうのはありません。（あ、でも為替レートとかを扱うときに、こんな調子の小数が出るとちょっと困るかもね...）

今のうちにいっておくと、pythonは文句なしに精密な実数を扱う手段を完備しています。ただ、この手段を使うとちょっぴり遅くなりますので、そこまで用がない人は、まずこんなもんで充分でしょう、というものが「素」では用意されているというわけです。

筆者自身の経験でも、（仕事にもよるのですが）あまり小数点以下を扱う必要も生じませんし、生じたにしても上の程度の精度で充分です。為替レートを扱いたいときは、あらためて筆者なりに相談してください。：）

割り算の話題に、ひとつおまけ。/ のかわりに、% という演算子を使って割り算をしてみましょう。（演算子という言葉をはじめて使いました。+ とか - とか * とかを総称してかっこよく言っているだけです。でもよく出てくる言葉ですよ）

```
>>> 10 % 3
1
```

は？　どういう意味？

いろいろ数字を変えて、この % がやっていることを推理してみてください。

```
>>> 100 % 3
1
>>> 99 % 3
0
>>> 98 % 3
2
>>> 97 % 3
1
```

まあそういうことです。割ったときの「余り」を計算するための演算子です。

なんだこりゃ。こんな変な計算する必要ってあるのかな。

これもまた、使うようになってみると意外に侮れない、便利なものなんですよ。今はまあ、存在だけを頭の片隅にとどめてくれていても結構です。

次に、演算子の優先順序です。

これもわざと難しく言っているようなもので、つまり、例えば足し算や引き算よりも掛け算が先に計算されますよ、ってことです。

```
>>> 10 + 3 * 6
```

これを頭から計算していったら、13 * 6 となって、結果は78でしょう。でも、実際には 3 * 6 のほうが先に計算されて、あとで10と足されます。

```
>>> 10 + 3 * 6
28
```

このとおり。小学校で習う、計算順序のルールにpythonは従ってくれます。まあ、賢い…

とはいえ、あまりこういった「暗黙の」計算順序に頼っているだけだと、間違いが紛れ込みやすくなるときがあります。スクリプトを書くにしても、その他のプログラム言語を使うにしても、あとで必ず「デバッグ」と呼ばれるミス修正をする羽目になるに決まっていますから、わざわざ「掛け算は先で足し算はあと…」とか頭でワンクッション置いて自分や他人の書いたものを眺めるのは、つらいときがあります。

で、あんまり計算順序の暗黙のルールをアテにするのはやめて、

```
>>> 10 + (3 * 6)
28
```

といったように、カッコを使って、ここは先にやってね、と明示しておくのがよいでしょう。

```
>>> (10 + 3) * 6
78
```

といったように、足し算を先にやらせたいときも、同様にカッコでくるんでしまえばOKです。

カッコは入れ子状態になっても、期待したとおりに動いてくれます。

```
>>> ((10+3) * (2+9)) - 43
100
```

とりあえずここまでで、pythonで計算をすることについての区切りとします。

ここまでの内容だけでも、もうpythonを活用できるようになったと威張ってよいでしょう。Windows標準の電卓アプリよりは、よほど込み入った計算ができるようになったわけですから。特に、Windowsでpythonする場合、命令を打ち込むときに矢印キーの[↑]を押すと、前回入力した行がそのまま出てきますから、あとは左右キーを使って一部を修正してから、ちょっとづつ違った計算をさせるといった技が使えて便利さ倍増ですよ。（いや、Linuxなどでpythonするときにも、ほとんどの場合同じ技が 使えますからね）

実際、筆者もこの程度の使い方で用を済ませたりすることは多いのです。

変数

今までのように計算だけしていてもそれなりに使いではあるんですが、変数というものの使い方を覚えると、もうすこし込み入ったことが可能になっていきます。

数学とかそういうので使った「変数」という用語は、別に思い出さないでもよいでしょう。あとで「関数」なんてのも出てきますが、それも数学的な定義を思い出すとかえって邪魔なので、はじめて使う言葉だと考えるほうが無難です。なにが"変"なの？　なにが"関"なの？　というのも気にするのはやめましょう。こんな言葉は、丸呑みでＯＫ。

まずは、対話シェルの上で下のようなものを打ち込んで動作を試してください。

```
>>> a = 500
>>> a
500
```

出ましたか。aは500だそうです。ここでは、aが変数。aの中身が500ってことです。「aっていうラベルの箱に入っている」というたとえでとらえてもよいでしょう。

なんでaの中身が500かとうと、最初の a = 500 でpythonにそう指示したからです。このイコール記号 = は、右と左が同じだよ、という常識的なイコール記号とはまったく違う意味で使われます。「aに500を入れますよ」という感じの意味です。「代入」なんて言葉で表現することもあります。aに500を代入します、って感じ？こんな言葉なくても通じるから、今後は代入なんていいませんけどね。数学でいう「代入」と全く同じことをするわけでもないようですし。（筆者は数学に詳しくないです）

とにかく、変数に"なんか"を入れておいてあとで使いますよ、という考え方は今までの電卓モドキな使い方とは一線を画します。よく理解してください。理解できないときは筆者に教えてください。

ところで、a なんてのがどこから出てきたのか。

べつに、bでもcでもよかったのです。pythonは、数字に見えないものがこうして出てきたら、変数じゃないかな、とみなして、よろしく振舞ってくれるのです。（もっと正確には、数字かな、文字列かな、リストかな…といった、もうちょっと細かい判断をします。でもこれらの説明をなんにもしてませんので、今のうちだけそんな理解をしておいてください）変数は一文字だけじゃなくてもよいです。

```
>>> budget = 10000
```

こうすると、budgetという変数に10000が入ります。打ち込むのはめんどくさいですが、あとで見直したときに、「なんか、予算が一万円なんだろうな」とか予想がつきやすくて人間にとっては親切な感じです。

変数に数字を入れた。じゃあ、それはいつまで残り続けるか。それは、今動かしている対話シェルが生きている間だけです。一旦今の画面を終わらせてしまったら、変数の中身は全部「なんもなし」に戻ってしまいます。というか、「そんな変数そもそも知らねえよ」といった感じに戻ります。

何も入れたことのない変数には何が入っているか、ためしに見てみましょう。

```
>>> nanimonasi
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
NameError: name 'nanimonasi' is not defined
```

怒られました。「nanimonasiってのが何なのか、まだ教えてもらっていないがな」ってことです。

```
>>> nanimonasi = 0
>>> nanimonasi
0
```

こうやってなにかを入れるフリをしてはじめて、変数として使えるようになるわけです。単純なことですが、ちょっと込み入ったスクリプトを作り始めると、結構このミスはやりがちです。

変数に入っている数字は、その後の計算に現れることができます。

```
>>> x = 100
>>> y = 1
>>> (x+y) * (x-y)
9999
```

三行目の計算式では、あらかじめ「xには100、yには1」が入ってましたから、それの中身をつかって計算ができたんですね。

じゃあこれではどうか。

```
>>> x = 100
>>> x = x + 1
>>> x
101
```

まあ、理解はできると思います。xに、x+1の結果を入れるんですから。改めて101が入ったと。

つづけて、今度はxの中身をひとつ減らしたかったら？　わかりますね。

```
>>> x=x-1
>>> x
100
```

これで減らすの成功。(記号の前の隙間は縮めてもいいので、この例ではわざとそうしました)

こんなふうに、変数に一度入れた値は、あとでいろいろと変化させていく使い方ができます。というか、変化させながら活用するのが普通です。"変"数というだけのことはあります。

そんじゃあまあ、練習として、色々な例でBMI係数でも算出することでもやってみましょうか。体重を身長の二乗で割ると出てくるのがBMI係数とかいうものだそうで。

えーと、身長を height、体重をweightという変数にして、適当な値をまずは入れます。ここは、小数点を含む実数で入れましょう。

```
>>> height = 195.5
>>> weight = 81.8
```

で、こいつをBMIの定義で計算する。身長の二乗だから二回掛けて、で、体重をそれで割る。

```
>>> weight / (height * height)
0.0021402267122794852
```

おお？　あ、身長はメートル換算でやらなきゃいけないのね。じゃあ身長は100で割るように 書き換えましょう。

```
>>> weight / ((height/100) * (height/100))
21.402267122794854
```

ふーん。この「身長195センチ、体重82キロ程度の架空のおっさん」は、大体標準体重に近い感じらしいですね。20が分かれ目らしいから。よく知らないんですけどね、BMI係数。

heightやweightにいろいろな値を代入してみて、さっきの計算式をもう一回入力しなおして遊びましょう。わりと打ち込む量が多くてだるいので、前章で紹介した[↑]を何度か押して入力履歴を再利用するテクニックを活用しながらやるとよいでしょう。

```
>>> height = 129.3
>>> weight = 129.3
>>> weight / ((height/100) * (height/100))    ←こいつは「履歴」からそのまま入力しましょう
77.339520494972916
```

ほう、ドラえもんは肥満体型と…

変数の命名ルールについては、大体「アルファベットで始まって、途中は数字が混じってもいい」くらいに理解すれば大丈夫です。注意点としては、大文字と小文字は区別されますよ。

```
>>> price = 2980
>>> Price
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Price' is not defined
```

ちょっと格好をつけて頭を大文字にしてみたりすると、それだけで「違う変数」とみなされます。もしもこんな命名をしたければ、全部そういう規則で書かなくちゃいけません。まあ、慣習上、ぜんぶ小文字で書くのが普通です。大文字で書くべきときは、別にありますんで。

あと、変数の名前にマイナス記号「-」や空白文字が入るのは禁止です。マイナスだと、引き算かと思われますので。空白をつけたら別モノに見えますし、このルールは理解できますよね。どうしても二つ以上の単語をくっつけた感じの名前を作りたかったら、下線記号「_」ならＯＫです。

```
>>> remaining_time_by_second = 60
```

こんな感じ。ここでは、残り時間はあと60秒とかそんな感じなんだろうな、と、読む人に伝わりやすいでしょ。打ち込むのはめんどいですけどね。

ああそうだ、変数のつもりで作った名前が、pythonで使える命令等に"かぶる"場合に注意しましょう。今までに紹介した命令には print がありましたが、これを変数の名前として使うのは一般的には禁止です。まだ詳しくは説明しませんが、不都合なことが多いので避けるに越したことはありません。

「じゃあその"命令"の一覧を見せてくれよ、注意するから」なんて言われそうですね。好奇心のある人は、下のような文字を打ち込んで[Enter]してみましょう。builtinsのまわりの下線文字

は、二箇所とも二文字分です。

```
>>> dir(__builtins__)
```

表示される結果は省略します。「ふーん」といったくらいに眺めておいてください。

これら以外にも、どうやっても変数として使えない「予約語」と呼ばれるものがあって、and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield が（python2.6では）全てです。"from"なんて、開始時間とかスタート地点とかそんなのを表現しようとしてついつい使いがちな単語なので注意しましょう。

変数の説明は、まずはこんなところです。キリをつけたい人は、ここで対話シェルを終わらせてしまってください。次に起動したときには、今まで入力した変数がサッパリ消えてしまっていますが、対話シェルをつかっている間はこんなものです。スクリプトファイルという形で保存してから使う、という方法に移るまでに、もう何章かはこんな調子でおつきあいいただきます。