目次へ

条件分岐、関数など

真偽値(1)

真偽値（しんぎち）ってのは、「はい」か「いいえ」かどっちか、ってことです。ここでは、pythonに「はい」か「いいえ」か答えてもらうようなものについて例をいろいろと挙げて見ていきます。

pythonは、「はい」のことをTrue、「いいえ」のことをFalseと言います。先頭一文字が大文字になっているのに注意してください。

たとえば、「1足す1は2ですか？」とたずねてみます。

```
>>> 1 + 1 == 2
True    ←そうだよ
```

イコール記号がふたつ重なった、== というものに注意してください。変数のことを説明した章で、「=」一文字だと、変数に値を入れる、という意味でした。「何かと何かが等しい」という表現をするには、これと見分けるために「==」とします。

じゃあ、「1000引く333は、777ですか？」とたずねてみます。

```
>>> 1000 - 333 == 777
False   ←そりゃちがうだろ
```

ちゃんと期待通りに答えてくれていますね。（1000 - 333は、667)

今の例は、数をそのまま挙げてpythonに真偽をたずねました。普段、こんなことをわざわざ聞いたりはしません。

もうちょっと役に立つのは、変数を含む式について真偽値を尋ねるときです。

```
>>> a = 199
>>> a == 200   ←aは200ですか
False    ←ちがうよ
>>> a = a + 1  ←aの値をひとつ増やす書き方
>>> a == 200   ←aは200ですか
True     ←そうだよ
```

といった感じで、aに何が入っているかがそのたびに違うような場合、こういう聞きかたはそれなりに意味があることです。

同じかどうかをたずねるだけでなくて、「より大きいか」「より小さいか」というたずねかたもあります。

```
>>> a = 100
>>> a > 70  ←aは70より大きいですか
True   ←そうだよ
>>> a < 95  ←aは95より小さいですか
False  ←そりゃあ違うね
```

== の代わりに < とか > とかを入れた式を作ると、こんなたずね方をすることができます。中学校とかで習う数学と、だいたい同じですね。

「より大きい」には二種類の書き方があるので、これも覚えてください。

```
>>> a = 100
>>> a > 100  ←aは100より大きいですか
False    ←いいえ
>>> a >= 100  ←aは100より大きいですか（等しいときも含む）
True     ←それなら、はい
```

> を使うか、>= を使うかで、値がちょうど同じだったときにTrueになるかFalseになるかという境目が変わってきます。これも中学数学でやる記号に似てますね。「より小さい」についても全く同じです。 < という記号と、 <= という記号があります。例は挙げませんので、適当に試しておきましょう。

あと、「等しくない」という記号もあります。「!=」です。

```
>>> a = 100
>>> a != 160   ←aは160と「等しくない」ですか
True  ←はい、「等しくない」です
```

等しくないときにTrue。等しいときにFalse。やや混乱しますが、なんとかわかってもらえるのではないかと思います。

これら真偽値をたずねる書き方は、そのうち出てくる条件分岐や繰り返しを表現するためにとても重要です。今のところは、「で、それが何よ」って感じかもしれませんが…

さて、数について真偽値をたずねてきましたが、次は文字列について同じように真偽値をたずねる書き方に移ります。

実は数のときとほとんど同じです。「=」「!=」「<」「＜＝」「>」「>=」がすべて使えます。ただし、文字列と文字列を比較するというときに限りますので注意。文字列と数を比較しても、あまり意味のある答えは返ってきません。以下、例をいろいろ。

```
>>> a = "Hello"
>>> a == "Hello"  ←aは"Hello"に等しいですか（はい）
True
>>> a != "Goodbye"  ←aは"Goodbye"と「等しくない」ですか（はい）
True
>>> a > "Ohayou"  ←aは"Ohayou"より大きいですか（いいえ）
False
>>> a < "Saraba"   ←aは"Saraba"より小さいですか（はい）
True
```

ちょっとまった、文字列が文字列より大きいとか小さいって何よ。

これは、辞書の順番に並べたときに、より後ろに来るなら「より大」、逆なら「より小」ということになっています。

python的には、全部の文字がそれぞれ大小順に並べることが可能なものとして見ています。'a'が一番小さくて、'z'が一番大きいよ、という感じです。もっと色々言うと、'A'は'a'よりも小さいです。'1'は'9'より小さいです。'1'は'A'より小さいです。'!'は'1'より小さいです。空白記号は'!'より小さいです。

まとめて言うに、下のような順に大小関係が並びます。

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

そんなの覚えきれるかって？　まあそうですねえ。英数字や記号が全部順番に並んだASCII（アスキー）コード表ってのを見たことがあれば、あれに則ってるんだとすぐ分かるんですが、見たことがあるとは限りませんね。

これを全部把握している必要があるような仕事は、まずないでしょう。大体、AよりZが大きい、大文字より小文字が大きい、というくらいの覚え方で構いません。とにかく、文字列どうしには大小関係がある、と知ってください。

文字列をつかった真偽値の問い合わせには、もう少し変わったものもあります。いくつか、よく使われるものを紹介します。といっても、まずはふたつかな。

```
>>> a = "ultraman"
>>> a.startswith("super")  ←aは、"super"という文字列で始まっていますか
False    ←いいえ
>>> a.endswith("man")   ←aは、"man"という文字列で終わっていますか
True    ←はい
```

とりあえず真偽値(1)の章はここまで。真偽値(2)の章では、今挙げたような割と単純な真偽値問い合わせを組み合わせてさらに込み入った論理演算をする例をやってみることになるでしょう。

「ＡまたはＢ」とか、「ＡかつＢ」とかそんなやつですね。ここらへんは、プログラムの習得がむつかしい、というよりは、論理的な思考がむつかしい、という種類の難しさを持つものになっていきます。

※真偽値(2)の章は、このテキストにはありません。

条件分岐

真偽値について色々と説明してみたのは、ここで出る条件分岐の書き方についてちょっとスムーズになるんじゃないかなとの考えからです。本当にそんな期待どおりに わかってもらえるのかは分かりませんが。

この書き方をまず見ていただきます。

```
>>> if 真偽値:
...     何かする
```

〝真偽値〞のところには、さっきまでに書いた真偽値問い合わせにあたるものを書きます。書いたら、コロン記号をつけて[Enter]を押します。

今までと違って、この場で命令が決定されるわけではありません。＞＞＞ の次の行に、... と出てきました。ifって書き出しではじまる命令は大抵二行以上かかるものなので、pythonは「まだ全部じゃないね、続きがあるんじゃない？」って感じで、入力をうながしてきます。これが ... です。

今までにも、なんかの入力ミスで、期待したように返事が返ってくるかわりに ... が出てきて困惑したことが、もしかするとすでにあるかもしれません。そんなときは、コントロールキーを押しながら[c]を押せば、今入力しているやつをまとめて撤回できます。

今回のifの入力では、二行目を書かされることは最初から予定済みのことです。この書き方の意味は、「これこれこういう条件のときは、何々をしなさい」ということ。

「何々をしなさい」の部分は、左のほうに空白をつけて書いていることに注意してください。本当にスペースキーを何回か押してスキマを作ってから、「何か」の命令を書きましょう。大体、空白二つ分か四つ分書くのが慣習です。実際には何文字空けても構いませんが、必ず一文字以上はあけるようにしてください。また、必ず空白文字を使うようにしましょう。タブキーを押すと手っ取り早く何文字も空白をあけることができますが、最初は素直にスペースキーを使って空白文字をポチポチ打ち込むことからはじめましょう。タブキーとかタブ文字の意味が分かっているなら構いませんが。

前置きが長くなりましたが、条件分岐の書き方の説明です。

今までは、とにかく順番に何かする、という例の繰り返しでした。今回のワザでは、「これこれこういうときだけ何かする」ということが記述できます。ゲームなんかをやっていて、分かれ道を右に進むか左に進むかで、行動の結果が変わってきたりすることがあるでしょう。こういうものを実現するための基本です。

```
>>> a = 10
>>> if a > 5:
...   print "larger than 5"
...
larger than 5
```

最初にaに10を入れておいて、次の行で「aが5より大きければ」と書きました。この真偽値問い合わせのあとにコロンを書くのがミソです。これを忘れるといきなりエラーが出て入力がパーになります。 ... で続きが入力できるようになったら、何でもやらせたいことを今までのワザを駆使して（空白をあけた後に）書きます。最初はこのくらいだけ書きましょう。二回目の ... が出てきたら、書きたいことはこれで全部だよ、とばかりに単に[Enter]を押します。これで書き方に問題がなければ実行されるでしょう。

ここではaがたまたま5より大きかったから、その下の命令が実行されました。

もしも条件を満たさなかった場合は、

```
>>> if a > 100:
...   print "larger than 100"
...
```

何もしません。これで、特定の場合だけ何かさせる書き方のできあがり。

次に、「さもなくば」を含む書き方です。下のような感じです。

```
>>> a = 10
>>> if a > 20:
...   print "larger than 20"
... else:
...   print "not so large"
...
not so large
```

... の部分を入力している間は、ずっとひとまとまりのif文を書いていると思ってください。途中で一文字でもミスしたら、ぜんぶまとめてパーです。段々と対話シェルでやるには無理が出てきそうな感じですが、大変なのはせいぜいこの程度なのでご辛抱ください。

else: というものが増えました。ここが「さもなくば」です。ここは行頭に空白がいりません。次の、print "not so large" の部分は行頭に空白をあけてください。コロンでおわる行の次は空白を空ける、と最初は覚えておいてもよいかもしれません。

もうあまり説明しなくてもよいかも知れませんが、この書き方の意味は、「aが20より大きかったら larger than 20 と表示せよ、さもなくば not so large と表示せよ」ということです。

いかにもプログラミングっぽくて、いろいろな場合によって違うことをするといったことができるようになってきた、という感じになってきませんか。そうでもないですか、そうですか。

ところで、こんな場合はうまく書けるでしょうか。「aが20より大きかったら何もしないが、そうでなかった場合は何か表示する」

こんな感じ？

```
>>> if a > 20:
... else:
```

```
...   print "not so large"
...
```

やってみるとわかりますが、この書き方はダメだといわれます。コロンで終わった行のあとは空白をあけた命令が書かれているとpythonが期待するせいです。最初の if a > 20: のあとに「何も書かない」という書き方が許されないのです。いきなりelse: を実行したいんだから、こう書かせてくれてよさそうなものですが。

もちろん、こんな感じに書き換えてしまって回避してもいいのですが…

```
>>> if a <= 20:
...   print "not so large"
...
```

つまり、20より大きい、の逆は、20より小さいか同じ、ということですから、こっちを条件にして、else: に書く予定だったものを前にもってくる、という方法です。

それでもどうしても a > 20 という書き出しでうまくif文にまとめたい、というときは、こう書けばよいです。

```
>>> if a > 20:
...   pass
... else:
...   print "not so large"
...
```

何もしない、と明示的に記述するには、pass と書きます。七並べのパスと一緒ですね。

さて、さっきから、5より大きかったら、とか、20より大きかったら、とかいちいち数字を決めてから書き出していて、あまり使い道がない感じです。この条件の書き方も変数で表現できるように直してみましょう。

```
>>> a = 20
>>> b = 5
>>> if a > b:
...   print a, " is larger than", b
... else:
...   print "not so large"
20  is larger than 5
```

printの書き方で、今まで紹介していないものが出てきました。コンマ記号「,」です。二つ以上のものを一度に表示したいときは、コンマでつなげて横に並べれば、実際の表示も横につながった形で出てくるので便利です。ここでは、aの中身と、" is larger than" という文字列と、bの中身の合計三つが一度に表示されます。

つぎに b に 20 より大きい数を入れて動作を試す…としたいところですが、いくら入力履歴をうまく使うとはいえ、もうこんなたくさんの文をいちいち打ち込みながら動作を試すなんてことはしたくありません。間違えるたびに … の入力行を全部中断して ＞＞＞ の部分から書きなおすなんてことも、いいかげんくたびれてきます。

こんな手間は一度だけにして、それ以降はもっと楽にこういう入力を再現できないもんか、と考えるのは自然でしょう。どんなプログラム言語にも、そういうことを実現するための機能があり

ます。呼び方はいろいろありますが、サブルーチンとか関数とか、メソッドとかマクロとか、そんなものです。ぜんぶ、こういう楽をするために作られた機能です。

pythonでは、そういう仕組みを function といってます。訳して「関数」なんていいますが、functionはもともと"機能"とかそんな程度の意味です。変に小難しくしないで、いったん書いた命令（の集まり）を楽して何度も実行すること、くらいに理解しましょう。次の章で説明します。

関数

関数が作れる／使えるようになれば、プログラミングの習得についてある程度の段階に到達したと言えると思います。よろしく。

まず例を見てください。

```
>>> def addtest(a, b):
...     return a+b
...
>>>
>>> addtest(10, 20)
30
>>>
```

def addtest というのが、「今からaddtestという名前の何か（関数）を作りますよ」という宣言です。def というのは define（定義する）という意味でしょうね、たぶん。addtestっていう名前は、適当に今考えました。足し算をやってくれる関数(add)というのを例として書こうとしていますが、所詮テスト的なクオリティのものでしょうから、testという単語をくっつけたまでです。

で、次の(a, b)というのは、この関数はふたつの引数（ひきすう）を使いますよ、ということです。引数は、関数を使う（使われる）ために必要な値のことです。この名前のつけかたは、a でも b でも x でも y でも何でもいいです。変数の名前のつけかたと同じで、あとでわかりやすい名前にするのがいいです。ここは例なので適当にしていますが。

で、def文の一行目のおしまいにはコロン「:」をつけて、それ以降の字下げ部分でこの a と b をつかった動作を書いてあげて、結果を今までのような print じゃなくて return という書き方で記述すれば関数のできあがりです。字下げの要領は、さきに条件分岐の書き方で示したときと同じです。

ここでは、a とか b は、すでにこういう名前の変数があるものとみなして書けばよいです。今回は字下げ部分は一行だけですが、これが二行以上になるときもあります。

で、例のおしまいのところでは、たった今作った関数を実際に使ってみる例を挙げています。addtestという関数に、10と20という数を引数として伝えました。すると10+20の結果として30が "return" された、というわけです。

いくつか前に、文字列の長さを知る書き方として、 len(何かの文字列) というのを紹介しました。これも関数でした。ただし、pythonが初めから用意してくれていた関数です。今作った

addtest 関数は、手作りでユーザーが作った関数です。どれが出来合いの関数でどれが手作りの関数か、知らない人にはもう見分けはつきません。ある意味、pythonというプログラム言語を拡張したんだとも考えられます。

例をいくつかこなせばもうちょっと分かってくると期待して、いつかやったBMI係数を計算 するやつを、関数として仕立て直してみます。

```
>>> def calcBMI(height, weight):
...     h = height / 100
...     return weight / (h * h)
...
```

関数の名前の付け方は、BMIを計算する、と言う程度の意味で、calcBMIとしました。大文字が混じっていますが、まあここはこういうことでアリとしてください。

受け取る引数は、身長(height)と体重(weight)です。わかりやすいような名前の引数にしました。体重（キログラム）に対して、身長をメートルに換算して計算をしないと正しいBMIは出ません。先の例ではひとつの計算式に「割る100」という表現を混ぜ込みましたが、今回の例ではあらかじめ h という変数に100で割った値をあらかじめ入れておくことにして、あとの計算ではこっちを使うことにします。

で、最終的な計算結果は、return を使って、関数を使った誰かに報告します。

使ってみましょう。

```
>>> calcBMI(190.5, 80.0)
22.044488533421511
```

身長190センチくらい、体重80キロくらいのこの人のBMI係数は、22くらいですね。

```
>>> calcBMI(129.3, 129.3)
77.339520494972916
```

ドラえもんのBMIも簡単な書き方ですぐに計算できるようになりました。

関数を一旦上手につくってしまえば、あとはこれの中身がどうなっているかを「忘れて」必要な処理を簡単に再現することができます。今は自分で関数をつくって自分で使うんだから大したことはありませんが、人がつくってくれた関数を自分のために使うということができるようになってくると、まことに楽でよいものです。

たとえば、誰かが「ピザの出前を頼む」なんていう関数を作ってくれたとします。中身がどう書かれているかはわかりません。でも、たとえば下のように書いて実行すると

```
>>> order_pizza("large tomato")
```

本当に最寄のピザ屋に「トマトピザのラージ」を（多分インターネット経由で）注文してくれたりするでしょう。どうやって実現されたか知らなくても、これでもう「pythonでピザが注文できるよ」と胸を張って言えるようになったわけです。

（この架空の例で、引数にさりげなく「文字列」を使っていることに注意。引数っていうから"数"しか使えないのかな、と考えそうになりますが、実際は何を関数に与えてもよいです。変"数"になんでも入るのと同じ。）

さて、関数の中でつかう計算に、あらかじめ作った別の関数を使うこともできます。calcBMIという関数をすでに作ってあるものと仮定して、下のような新しい関数を作ってみます。（pythonを終わらせると、今まで作っていた関数がすべて消えます。変数のときと同じですね。注意）

```
>>> def is_yasegata(height, weight):
...     bmi = calcBMI(height, weight)
...     if bmi < 20.0:
...       return "yasegata"
...     else:
...       return "not yasegata"
```

... でちょっと長めに続いていますので、打ち込むのはややつらいです。ご辛抱ください。スペースキーでの字下げを確実に行いながら写してください。字下げが、二段階になっているのにお気づきでしょうか。def is_yasegata... にかかる字下げ部分は、下にある 5 行分全部です。その中に、条件分岐のために if を書きました。これはこれで、別の字下げが入ります。これによって、二段階のデコボコした字下げができあがりました。このデコボコが正しくないと、きっと記述中にエラーが出て打ち込みに失敗します。

さて、is_yasegata という関数を作りました。中身は、calcBMIを使っているようです。bmiという変数に、受け取った身長と体重を使ってBMI係数を計算して、入れます。次に、if を使って場合分けしながら、計算した係数が20より小さかったらメッセージをreturn、さもなければ別のメッセージをreturnするという仕組みになっています。今までの章をごらんになっていればすべて分かる、と思いますが...

さっそく使いましょう。

```
>>> is_yasegata(190.0, 80.0)
not yasegata
>>> is_yasegata(190.0, 70.0)
yasegata
```

身長190センチのとき、体重80キロは痩せ型とはいえない、でも70キロなら痩せ型らしい、ということがわかるようになりました。

もし「ピザを注文する」という関数が存在して、それの使いかたを知っているなら、身長と体重を与えて、痩せ型のときだけピザを注文する、という変な処理を書くこともはや自由自在です。まあ、そんな例はすごく意味がありませんが。

スクリプトファイルを書いて実行する、というやり方はまだ説明していません。対話シェルを使っているうちは、pythonを終わらせるたびに作った関数もすべて消えてしまいますから、今はまだあまり複雑なものは書きません。