目次へ

リスト、辞書など

オマケ 乱数で遊ぶ

人がつくった関数を使う、ということについて、「ライブラリ」とか「モジュール」とかそんな言葉をつかって表現することがあります。「最新のモジュールを導入してインポートしました」とか言われたら、ああ、人が書いてくれたやつを使ったんだね、という程度に理解して大丈夫です。

普通にpythonをインストールすると、このテの便利なモジュールがいきなりたくさん付属してきます。ここでは、ちゃんとした説明に先立って、ためしにひとつそのモジュールのひとつを使ってみます。

まずは、対話シェルで下の命令を入れてください。まだ意味はわからなくてもよいです。

```
>>> from random import randint
```

これが正常に実行されると、今までになかった randint という関数が使えるようになっているはずです。randintはふたつの数を引数としてとる関数で、最小値と最大値の間からランダムに値を作ってくれます。六面のサイコロを振るとすれば、下のような感じ。

```
>>> randint(1, 6)
5
```

ここでは５の目が出ましたが、実行する人によって、色々な値が出ているはずです。また、二回実行したら、また違う値が出るでしょう。

うーん、乱数を計算するのって、どういう仕組みなんだろう、と思ったりする人もいるでしょう。そういう人はpythonのインストールされているディレクトリを探検して色々と勉強することができます。でも、たとえば僕だってちゃんと調べたことはありません。randomモジュールの使い方だけはマニュアルで読んで、あとは問題なく便利に活用しています。

さて、乱数をつかって、簡単な占いをしてくれる関数を書くとすればどんな感じでしょうか。randintは使える状態と仮定して書いてよいです。

たとえばこんな風でしょうか…

```
>>> def uranai():
...     r = randint(1, 100)
...     if r > 90:
...       return "daikichi!"
...     elif r > 50:
...       return "kichi!"
...     else:
...       return "maybe bad..."
```

uranai（占い）関数をつくりました、引数は、とくに使いません。カッコ開くとカッコ閉じるを単に並べて() と書いて、引数がいらないことを明示しましょう。これは省略不可です。

r にはまず、1から100の値をランダムに選んで入れます。次のif では、まず、rが90より大きい場合のことを記述しています。90より大きい場合は、「大吉」というメッセージを報告します。この場合、ここで関数の実行は終わり。次に、ここで初めて出てくる elif というものがあります。これは if と else のあいのこみたいなもので、「そうでなくて、もし」という意味を持ちます。つまり、90より大きい場合は大吉なんだけど、「そうじゃなくて、もし」rが50より大きかったら、「吉」ということにしよう、というのがこの書き方の意味です。それにも該当しない「その他」の場合は、else: で書いてあるところが実行されます。「やばいかもね」というメッセージを返すことにしました。

何回も実行してみましょうか。引数がいらない場合にも () は必ず表記してくださいね。

```
>>> uranai()
'maybe bad...'
>>> uranai()
'kichi!'
>>> uranai()
'maybe bad...'
>>> uranai()
'maybe bad...'
>>> uranai()
'maybe bad...'
>>> uranai()
'daikichi!'
>>> uranai()
'daikichi!'
>>> uranai()
'maybe bad...'
>>> uranai()
'maybe bad...'
>>> uranai()
'maybe bad...'
>>> uranai()
'kichi!'
>>> uranai()
'kichi!'
```

大吉が出てくる確率は10%くらいといったところですね。100までの乱数が90より大きい確率なんですから、大体妥当そうです。

もっときめ細かく段階を作りたければ、elif をもっとたくさん増やすのがよいでしょう。90より大なら大吉、70より大きければ吉、40以上なら末吉、あとは凶、大凶、大大凶... まあ、お遊びですから、あまり凝ることもないですが。うまく作れば、イベント会場で福引きアプリケーションとして使えるんじゃないでしょうか。または、twitterで流行している「○○ったー」とかそんな感じの占い系サービスの原型もね。

リスト

関数まで書けるようになったので、かなり色々なことをするための基礎までを習得できたことになります。ただ、もうひとつ重要な道具として、リストというものがどんなものでどういった使い方ができるのかということを知っておくことが重要です。

説明より、まず見てみるのが早いかもしれません。下の例を対話シェルに打ち込んでみましょう。

```
>>> a = [1, 2, 3, 4, 5]
>>> a
[1, 2, 3, 4, 5]
```

角カッコ[ ]で囲むのがミソです。それぞれの値は、コンマ「,」で区切ります。いかがでしょうか。aという変数に、値が一度に５個入ったようですね。で、そのまま下の例も続けて試してください。

```
>>> a[0]
1
```

aに入っている５つのうち、最初のものを表示させてみました。

おわかりでしょうか。リストというのは、値がたくさん順番にならんだ状態のことです。

この何番目という指定を見て、「文字列」に似てるかな、と思いついた人は鋭いです。文字列は、文字がひとつづつ並んだものですが、リストは、文字に限らず、いろいろなものを並べて扱うことができます。その意味で、文字列に似ていて文字列以上の使い道を秘めたモノです。

文字列そのものを、リストのひとつずつにセットするようなこともできます。

```
>>> a = ["my", "name", "is", "python"]
>>> a
['my', 'name', 'is', 'python']
>>> a[0]
'my'
```

この例では、リストの中のひとつの要素が、独立した文字列ですね。ひとつの変数に、リストという形でたくさんの値を入れることができるという証拠のひとつです。

リストの中のひとつに、別の値を入れるという操作も書くことができます。下のとおり。

```
>>> a[1] = "favorite"
>>> a
['my', 'favorite', 'is', 'python']
```

[1]は、ゼロから始まるから実質二番目の値ということです。変数に値を入れるときの書き方にとても似ていますね。違うのは、変数がリストと仮定して、そこの一部分にのみ入れるんだと指定されているところです。

リストでは、並びの途中をこのように変更することができますが、文字列の途中の一文字だけを直すこともできそうに思えてきますね。文字列とリストって似てるんだから。でも、結論だけいうと、文字列の途中の一文字を今のような方法で直すことはできません。

```
>>> b = "12345"   これは「文字列」です。クオーテーションで囲んでるから
>>> b[2] = "Z"    で、３文字目を 'Z' に直せるかな…
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment （ダメだってさ）
```

これと似たことを実現する方法はもちろんあるのですが、とりあえずは、文字列は一部分を変更することはできないんだと覚えておいてください。

さて、リストは、文字列のときと同じように、「どっから、どこまで」という記法 [:] も使用可能です。

```
>>> a[2:]   ←二番目の「切れ目」からすべて
['is', 'python']
```

文字列の長さを知るのに使った、len という関数は、リストに使われたときはその並びの数を返してきます。

```
>>> len(a)
4
```

リストに新しく値をひとつ追加するときは、appendという命令を下のように書きます。リストのおしまいに追加されます。

```
>>> a.append('scripting')
>>> a
['my', 'favorite', 'is', 'python', 'scripting']  ←ひとつ追加された
```

この、変数の次にピリオド「.」をつけて、その後に関数のような感じの命令を書くというのが、変な感じがするかもしれません。今は、「aに、appendするんだ」というくらいにイメージして、丸呑みに覚えてしまえばよいです。

リストからひとつ値をはずしたいときは、popという命令を下のように書きます。リストのおしまいから取り去られます。

```
>>> a.pop()  ←カッコを単に開いて閉じる。引数がとくにないから
>>> a
['my', 'favorite', 'is', 'python']  ←再び減った
```

リストがどういう性質をもつものか、なんとなく見当がついてきたでしょうか。

一応注記しておきますと、文字列に対してはこういった append や pop の操作をすることはできません。さっき、何文字目かを他の文字に入れ替えてみるのに失敗したのと要領は同じで、文字列は自分自身を変更してしまうような操作ができません。リストと似ていながら似ていない点ですね。

さて、リストの中のそれぞれの要素は、数だったり文字列だったりすることができました。さらに、リストの中のある要素にリストが入っているということも可能となっています。

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [10, 20, a, 30]  ← aが途中の要素として入っている
>>> b
[10, 20, [1, 2, 3, 4, 5], 30]
```

bに入っている要素を順に挙げると、最初が10、次が20、次が[1, 2, 3, 4, 5]、次が30 といった具合になります。リストの中にまた別のリストが入っている、というのは、なんだかややこしいですが、結構よくあることです。

リストの中のひとつひとつが変数のような性質をもっていて、色々な種類の値を格納することができるのだ、と理解してもよいかも知れません。さっきは、リストの一部分に値を入れなおしたりすることができましたしね。

しかし、それにしても、実際に実用的に使ってみないと、これらのひととおりの使い道だけではどう便利なのかがわからないでしょう。使い道は、特に決められているわけではありません。これを使うとあんなデータの処理に使えそうだな、と思いついたら、それが使い道なわけです。

このテキストは続編が書かれる予定です。そこでは実際の事例を想像させる練習問題がいろいろと出題されますから、そこでリストを活用するための様々な例が紹介される予定です。

ああそうだ、リストをつかった処理として、ループを表現するという大事なものがありました。

「1から10まで何かする」というような処理が必要になったとします。たとえば、それぞれの数の二乗をそれぞれ一度に計算したいようなとき。下のような書き方をします。

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for i in a:
...     print i * i
...
1
4
9
16
25
36
49
64
81
100
```

ここではaがリストでそれぞれ数が入っているとして、for i in a: といったお決まりの書き方をしながら、字下げ部分にこの i なら i を使って行いたい処理を書く、という感じです。これが、決まった回数だけ何かを実行したいときの基本的な書き方になります。

ちょっとまった、10回ならこんな書き方でよいけど、10000回とかだったらどうするの？

そんなときは、もちろん、別の方法がありますよ。それもまた、練習問題編で、ふさわしいときに紹介しようと思います。

辞書

今まですべてを読んでくれた方は、変数にはいろいろなものが入るんだなと理解されたことでしょう。

```
a
```

という表記をひとつ見るだけでも、これには数字が入っているかも知れないし、文字列が入っているかも知れない、と注意することができます。または、aの中身はリストかもしれません。

はたまた、ここではaというのは実は関数として定義されているかもしれません。aっていう関数を作ることは可能ですからね。下のように。

```
def a(k):
  return k * 2
```

受け取った値を二倍してみる、という、どうでもいい感じの関数ですが、ちゃんと通用します。こうなるとaはただの変数どころでなく、立派な関数です。（変数と関数には実はたいした違いがないのですが、この話はまだ早いので、カッコ内の独り言だけ...）

なんせ、

```
a
```

とだけあっても、なかなか油断がなりません。

油断がならないついでに、もうひとつだけ、aに新しい種類のモノが入りうるということを紹介します。“辞書”(dictionary)です。辞書ってなんだそりゃ？

まずは、簡単な辞書を作って、変数に入れてみます。

```
>>> a = {'name': 'yama', 'age': 36}
>>> a
{'age': 36, 'name': 'yama'}
>>>
```

このくにゃっと曲がったカッコ記号（ブレース）は、シフトキーを押しながら「[」とか「]」で入力できます。また、コロン「:」やコンマ「,」がどういう順に入っているかに注意しながらこの例をまねてください。できましたでしょうか。

ブレース記号で囲んだばあい、pythonは「ああ、“辞書”をつくろうとしているのか」とわかってくれます。だから、この時点で、a には辞書という種類のデータがめでたく（？）入っているわけです。

'name': 'yama' というのがひとつの表記単位で、次の単位が 'age': 36 です。ここでは a というのがまるで一人の人間であるかのようにみなされて、'name' は 'yama'、'age' は 36 と、そんな表現になっています。yamaさん36歳。'name' とか 'age' といったほうを、〝キー〞と呼びます。'yama' とか 36 といった部分は、キーに対応する〝値〞とでも呼んでおきます。

pythonにおいて、〝辞書〞ってのはこういうことです。dogは犬、catは猫…といった感じに、一対一に値のセットが並ぶことから、開発者が、辞書に似たようなデータだと思ったんでしょうね。プログラム言語によっては、こういう種類のデータ表現を「連想配列」とか「ハッシュ」とか、最近は「Key-Value」なんて言うときもありますが、まあ同じようなものを指していると思って差し支えないです。

こういうものが、たとえば a なんていう変数にすっぽり入ってしまうことが可能なんです。怖ろしいことですね。（そんなこともないって？）

辞書という種類のデータについて、こんな操作ができるんだぞ、というのを、いくつか例として挙げていきます。a には、直前の例で打ち込んでもらった、{'age': 36, 'name': 'yama'} が入っているとします。

```
>>> a['name']  ←名前は？
'yama'      ← yamaっていいます
>>> a['age']  ← 何歳？
36          ← 36歳
>>> a['job'] = 'janitor'    ←君は今から清掃員だ
>>> a['job']  ←仕事は？
'janitor'   ← 掃除してます…
>>> a.keys()   ←どんな属性がセットされているのか、すべて見たい
['job', 'age', 'name']  ←こんな属性について、答えを持っています
```

角カッコで囲んで指定するところは、リストに似ていますね。でも辞書の場合は、角カッコの中身が文字列になることが多いです（実は数でも可能ですが）。

例で示したとおり、新しい属性（キー）を指定して、それに対応する新しい値をセットするようなことも可能です。

```
>>> a
{'job': 'janitor', 'age': 36, 'name': 'yama'}
```

aの中身をすべて表示してみたら、'job' が増えていますね。そういうことを、必要に応じてできるのです。

ところで、表示させてみた結果が、'job' 'age' 'name' の順に必ずしも並ぶとは限りません。辞書は、表示されるときにそれぞれの属性の表示順には頓着しないという仕様になっているのです。

辞書というものがどんなものか、大体のイメージはつかめましたでしょうか。リストとはかなり違う性質をもっていることがわかると思います。

ここでは、キーに対応する値には文字列か数が入ることが例として示されましたが、リストのときと同じく、値の部分にリストが入ったり、また値の部分に別の辞書が入ったりということが可能です。

辞書の使い道もなかなか多彩なもので、活用できれば、極めて便利なものであることがおいおい理解されていくことでしょう。奥は深いです。ですが、その説明は練習問題編にゆずりたいと思います。

まずは、ここまで

こんなところで、次からつづく「練習問題編」を読むために必要な最小限の知識をここに書いたつもりです。

実は、まだ重要な習得事項のうちいくつかが不足しているという状態です。 たとえば、

- ファイルの操作
- 日本語の扱い（スクリプトファイルの作成と取り扱い）
- whileなどを使う、もうすこし詳しいループの扱い
- 例外処理
- モジュールの一般的な利用法

など。さらに、言葉をご存知のかたがいるかもしれませんが、〝オブジェクト指向”とかいうものを 実現するための色々な知識も必要になってくることがあるでしょう。

ただ、今知らなくたって、必要に応じて習得するための準備はもうできてますよ。多分。 追ってそこらへんの話も書き足していく予定。

後半、ちょっと駆け足だったかも知れません。随時直していきます。