14　For 文による繰り返し

14.1　For 文の文法

同じ処理の繰り返しを記述するには for 文を使います。たとえば，hello をプリントするという処理を 5 回繰り返すには手動で

```
>>> print 'hello'
hello
>>> print 'hello'
hello
>>> print 'hello'
hello
>>> print 'hello'
hello
>>> print 'hello'
hello
```

とやればよいのですが，次に説明する for 文を使えば，このような処理をより簡潔に書くことができます：

```
for j in range(5):
    print 'hello'
```

実際に，Python のインタラクティブシェルから上のプログラムを書けば次のような表示になるでしょう：

実行結果の例

```
>>> for j in range(5):
...   print 'hello'
...
hello
hello
hello
hello
hello
```

ここで，range(5) はリスト [0,1,2,3,4] だったことを思い出しましょう[*5]：

```
>>> range(5)
[0,1,2,3,4]
```

上の，for 文で『for j in range(5):』は j を 0 から 4 まで順番に変えて，その下のブロックの処理を繰り返しているので，次のように j そのものの値を使うこともできます：

```
for j in range(5):
   print j
```

実行結果の例

```
0
1
2
3
4
```

ここで j はダミー変数なので，他の文字に変えても結果全く同じです。例えば，次のプログラムは上のものと同じです：

```
for x in range(5):
   print x
```

[*5] Python3.x では range はリストを返しません。list(range(5)) がリスト [0,1,2,3,4] になります。

さて，for 文の構造は次のようになっています：

For 文の構造

同じ作業を 10 回繰り返すプログラムです：

```
1 for j in range(10):
2     |ここのブロックに|
3     |繰り返す       |
4     |プログラムを書く|
5
6 <次に行うプログラムはここに書く>
```

- インデント（字下げ）されている部分が繰り返す処理。
- 6 行目は，インデントから外れるので for 文の外にあるので繰り返されない。この部分は for 文の繰り返しが終わってから実行される。

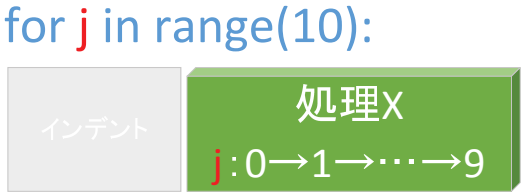

図 2　for 文の構造

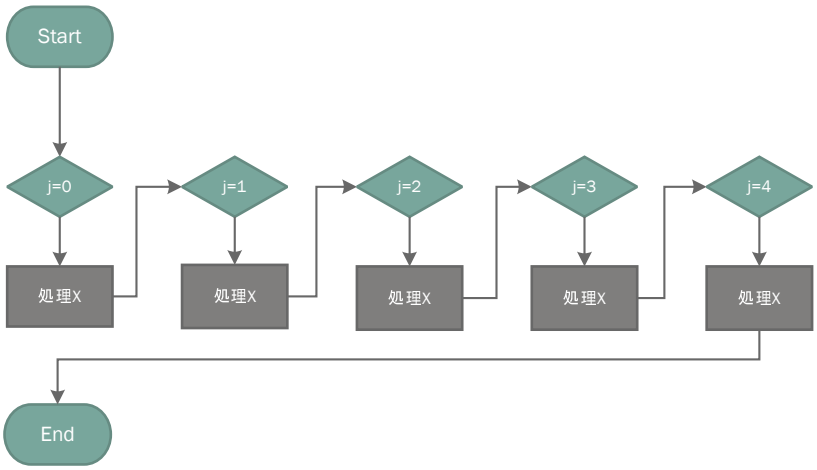

図 3　for 文で行われる処理 (5 回の繰り返し)

14.2　for 文のいくつかの例

for 文のプログラムをいくつか書いて実行結果を見てみましょう：

- ファイル名：**for1.py**

```
1 for j in range(5):          # 以下のブロックを5回繰り返す
2     print 'wanwan',        # これと
3     print 'nyannyan'       # これを繰り返す
```

実行結果の例

```
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
```

次のプログラムの最後の行は for 文のインデントから外れるので一回しか実行されません：

- ファイル名：**for2.py**

```
1 for j in range(5):            # 以下のブロックを5回繰り返す
2     print 'wanwan',           # この行と
3     print 'nyannyan'          # この行を繰り返すが
4
5 print 'gaogao'                # ここは繰り返さない
```

実行結果の例

```
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
wanwan nyannyan
gaogao
```

for 文を使って 0 から 9 までの数字の 2 乗を表示します：

- ファイル名：**for3.py**

```
1 for j in range(10):               # jを0から9まで変えて次を繰り返す
2     print j, j**2                 # jとjの二乗をプリントする
3
4 print 'owari'                     # ここは繰り返さない
```

実行結果の例

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
owari
```

数字を文字列にしてつなげれば次のような操作もすぐにできます：

- ファイル名：**for4.py**

```
1 # -*- coding: utf-8 -*-
2 for j in range(1,101):
3     print 'ひつじが'+str(j)+'匹'
```

実行結果の例

```
ひつじが 1 匹
ひつじが 2 匹
ひつじが 3 匹
.....
.....
.....
ひつじが 99 匹
ひつじが 100 匹
```

与えられたリストに対して，その要素について順番に同じ作業をすることができます：

- ファイル名：**for5.py**

```
1 aa = [ 'Alice', 'falls', 'down', 'a', 'rabbit', 'hole.']      # リストaaを定義
2
3 for i in aa:          # aaの中のiについて順番に
4     print i,          # iをプリントする
```

実行結果の例

```
Alice falls down a rabbit hole.
```

14.3 for 文のインデントのエラー

Python の文法の特徴にインデントで構文を判断するというものがありました。`if-else` 文ではインデントを間違えるとエラーになりましたが，`for` 文でも同じ事がおきます。次の例のように字下げが少し違うとエラーとなります：

- ファイル名：**for6.py**

```
1 for j in range(5):
2     print 'wanwan',
3   print 'nyannyan'
```

実行結果の例

```
  File "for6.py", line 3
    print 'nyannyan'
                ^
IndentationError: unindent does not match any outer indentation level
```

次のようにインデントを不当に下げた場合も同じエラーとなります：

- ファイル名：**for7.py**

```
1 for j in range(5):
2     print 'wanwan',
3      print 'nyannyan'
```

実行結果の例

```
  File "for7.py", line 3
    print 'nyannyan'
                ^
IndentationError: unindent does not match any outer indentation level
```

14.4 for 文の応用 1 ：(for 文の中に for 文を入れる)

もちろん for 文の中に for 文を入れて二重に繰り返す事もできます。

- ファイル名：**for8.py**

```
1 for i in range(5):              # i = 0 ~ 5に対して以下を繰り返す
2     for j in ['a','b','c']:     # j = a, b, cに対して次を繰り返す
3         print i, j,             # iとjをプリント
```

実行結果の例

```
0 a 0 b 0 c 1 a 1 b 1 c 2 a 2 b 2 c 3 a 3 b 3 c 4 a 4 b 4 c
```

次のようにすれば掛け算の表が出力されます：

- ファイル名：**for9.py**

```
1 for i in range(1,10):           # i=1,2,...,9と
2     for j in range(1,10):       # j=1,2,...,9に対して
3         print i*j,              # i*jをプリント
4     print ''                    # ここで改行
```

実行結果の例

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

14.5　for 文の応用 2： if 文と組合わせる

if 文と組合わせて，for 文の中で処理を分岐させるとより複雑なプログラムが書けます。次では j が偶数なら j is even，奇数なら j is odd と表示させるようなプログラムです：

- ファイル名：**for10.py**

```
1 for j in range(1,10):
2     if j%2 == 0:
3         print j, 'is even'
4     else:
5         print j, 'is odd'
```

実行結果の例

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
```

14.6　for 文の応用 3: break で処理を止める

for 文では与えられた回数だけ繰り返すことが簡単にできますが，処理を途中で止めたいときには break を使います。for 文の繰り返し中に break が現れたらその for 文から抜けます。例えば次のようなプログラムを考えます。

- 2^{2^j} で j を $1, 2, 3, \cdots, 10$ と動かして次々プリントしてゆきます，
- これは非常な早さで増加するので，もし 2^{2^j} の値が 100000000 を超えてしまった場合にはそこで for 文処理をやめたい。

- ファイル名：**for11.py**

```
1 for j in range(11):
2     a = 2**2**j
3     if a<1000000000000:
4         print a
5     else:
6         print 'Too big!!'
7         break    # これが実行されたらfor文はおしまい
```

実行結果の例

```
2
4
16
256
65536
4294967296
Too big!!
```

15　ファイルの書き込み・読み込み

Python の実行結果をファイルに書き込んで保存する方法を解説します。

15.1　ファイルの書き込み (その 1)

ファイルの書き込みで手軽なのは，端末の機能を使うことです，Python プログラムで書きたいものを print しておけば，

```
user@debian :~/Desktop/dataproc1/06$
```

のように，端末でプログラムの置いてあるディレクトリで

```
$ python ファイル名.py > 出力するファイル名.txt
```

とすることによりプリントされたものをファイルに書き出すことができます。たとえば

```
$ python for4.py > for4result.txt
```

すると，フォルダには for4result.txt というファイルが新たに作られ，for4.py の実行結果が書き込まれています。ファイルの末尾に『追記』するには，『>>』を使います。試しに

```
$ python for3.py >> for4result.txt
```

を実行してみましょう。これで，for4result.txt のファイルの最後に，for3.py の実行結果が書き込まれました。

15.2　ファイルの書き込み（その 2）

ファイルに書き込むもう一つの方法は Python 自体のファイル操作の機能を使うことです。

- ファイル名：**write1.py**

```
# -*- coding: utf-8 -*-
abc = 'hogehoge'                # 変数abcを定義
f = open('test.txt', 'a')       # 追記モード(a)で開く
f.write(abc)                    # abcをtext.txtに書き込む(追加)
f.close()                       # ファイルを閉じる
```

```
$ python write1.py
```

を実行すると test.txt が作られて hogehoge という文字列が書き込まれました。もう一度実行すると，文字列がさらに追加されます。追記ではなく，今あるものを消去してから書き込むには上の 3 行目の代わりに

```
f = open('test.txt', 'w')       # 書き込みモード(w)で開く
```

とすればよいです。

例として，ひつじが・・・のプログラムをこの手順でファイルに書き出すことをやってみましょう。そのためには，まず文字列を作成しなければなりません。

- ファイル名：**CountSheep.py**

```
# -*- coding: utf-8 -*-

abc = ''        # 変数abcを空の文字列とする
```

```
4
5  for i in range(10):
6      abc = abc + 'ひつじが'+str(i)+'匹\n'    # \nは改行を意味する
7
8  f = open('hitsuji.txt', 'w')     # ファイル(hitsuji.txt)をwriteモードで開く
9  f.write(abc)
10 f.close()
```

このファイルを実行すれば，自動的に hitsuji.txt というファイルが作られ，中に，『ひつじが・・・』と書き込まれます。

15.3　ファイルの読み込み

ファイルを読み込むには read や readlines を使います。前者はファイルの全部の内容を一つの文字列として読み込み，後者はファイルの各行のリストを作り出します。ここで，読み込むファイルを作るために次を実行します：

```
1 $ python for11.py > hoge.txt
```

ディレクトリには hoge.txt が生成されて，for11.py の実行結果が記録されました。さて，このファイルを read で読み込みます：

- ファイル名：**read1.py**

```
1  # -*- coding: utf-8 -*-
2
3  f = open('hoge.txt', 'r')  # 読み込みモードで開く
4  aaa =  f.read()            # hoge.txtの内容をaaaとする
5  print aaa                  # aaaの内容をプリントする
6
7  f.close()                  # hoge.txtを閉じる
```

実行結果の例

```
2
4
16
256
65536
4294967296
Too big!!
```

つぎに，各行を readlines で読み込みます：

- ファイル名：**read2.py**

```
1  # -*- coding: utf-8 -*-
2
3  f = open('hoge.txt', 'r')  # 読み込みモードで開く
4  aaa =  f.readlines()       # hoge.txtの各行からなるリストをaaaとする
5  print aaa                  # aaaの内容をプリントする
6
7  f.close()                  # hoge.txtを閉じる
```

実行結果の例

```
['\n','4\n','16\n','256\n','65536\n','4294967296\n','Too big!!\n',]
```

aaa は hoge.txt の各行が文字列になったリストである事がわかります。改行は「\n」になっています。

16　練習問題

16.1　問題：$3n+1$ 問題

$3n+1$ 問題というものがあります。自然数 n に対して

- もし n が偶数なら n を半分にする
- もし n が奇数なら n を $3n+1$ にする

という操作を繰り返すと，最終的にはどんな自然数も 1 になるであろうという予想です。現在も未解決の問題で角谷の問題とか Collatz 予想と呼ばれています。例えば，最初の数が 9 の場合上の手順で作られる数列は

9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

となります。入力された数 a に対して，上の手順で作られる数を次々に表示するプログラムを作りたい。プログラムは次のアルゴリズムに従うように作りたい。

1. 自然数 a の値を入力させる。
2. 最大の繰り返し回数 maxiter を 1000 とする。
3. 以下の 4 から 7 を for 文で maxiter 回繰り返す。
4. もし a の値が 1 なら break で for 文を終了する
5. もし a が偶数なら a を半分にする
6. もし a が奇数なら a を $3a+1$ にする
7. a をプリントする

以下の Python プログラムの****を自分で考えて上のアルゴリズムが実現するようにしなさい。

- ファイル名：**collatz.py**

```
a = input('a?: ')
print a,

maxiter = 1000
for i in range(maxiter):
    ***********
        break
    **************
        *******
    *****
        ***********
    print a,
```

プログラムが完成したら実行して a として 19 を入力してみましょう。

実行結果の例

```
a?: 19
19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

のようになればおそらく正しいプログラムです。

（注意）上のプログラムでは最大の繰り返し回数を 1000 に設定しましたが，何回繰り返すのかがやってみないとわからない処理を書くには while 文を使います (1000 回では足りないかもしれない！)。while 文は次の章で学習します。

16.2　数字当てゲーム

1 から 1000 までの数字 x をランダムに生成して，x を 8 回以内で当てるゲームを作りたい。

乱数はライブラリをインポートすることで手軽に使うことができます。次が乱数を使った簡単なプログラムの例です。

```
1 import random    # 乱数のライブラリrandomをインポートする
2 x = random.randint(1,10)    # 1から10までの数字をランダムに生成しxとする。
3 print x
```

さて，ここでは，次のような手順を行う数字当てゲームを作りたい。

(i) 1 から 1000 までの数を一つランダムに生成し，それを x とする。
(ii)『x を当ててみよう』と表示する。
(iii) キーボードから入力された数字を変数 a に入れる。
(iv) $a = x$ なら『正解』と表示しゲームは終了する。
(v) $a < x$ なら『もっと大きい』と表示し，$a > x$ なら『もっと小さい』と表示する。
(vi) 上の (iii)–(v) を 8 回繰り返し，当てることができなければ，『Game Over』と表示する。

より詳細な手順を示したアルゴリズムはつぎの通りです。

(1) ライブラリ `random` をインポートする。
(2) 1 から 1000 までの数を一つランダムに生成し，それを x とする。
(3)『1 から 1000 までの数字 x をランダムに生成されました。x を 8 回以内で当ててみよう！』と表示する。
(4) for 文で j を 0 から 7 まで変えながら，次の (5)–(8) の処理を繰り返す。
(5) `input` で『残り $8 - j$ 回です。x は何でしょう？：”』と表示して，キーボードからのデータを取得し，それを変数 a に入れる。
(6) もし $a = x$ なら『正解』と表示し，`break` で for 文から抜け，
(7) もし $a < x$ なら『もっと大きいよ』と表示し，
(8) そうでないなら『もっと小さいよ』と表示する。
(9) もし，$x \neq a$ なら次の (10),(11) を行う。
(10)『正解は x でした』と表示する。
(11)『残念 Game Over』と表示する。

上のアルゴリズムを Python で書きましょう。ファイル名は `find_number.py` とすること。