19　リストの操作とリスト内包表記

いろいろなデータをまとめて取り扱うときに便利なのがリストです。リストとは [3,6,3,9] のように，いくつかの要素を括弧の中に並べたものです。短いリストであれば，次のように直接的に書くことで定義します。

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

さらに，リストに要素を付け加えるには，append メソッドを使います。

```
>>> a.append(7)
>>> a
[1,3,2,4,7]
```

もう少し多様なリストをある種のパターンで生成するためには，for 文などの繰り返しの処理を使って要素を付け加えます。次の例を見てみましょう。

```
aa = []                         # aaを空のリストとする
for i in range(10):             # iを0から9まで動かして
    aa.append(i*i)              # i*iをリストaに付け加える。

print aa                        # aaをプリントする
```

実行結果の例

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

上では，空のリストに i^2 $(i = 0, 1, 2, \cdots, 9)$ を付け加える事によって，平方数からなるリストを生成しています。次に $1, 9, 25, 49, 81, \cdots$ と奇数の平方数からなるリストを for 文を使って作ってみましょう。

```
bb = []                         # bbを空のリストとする
for i in range(20):             # iを0から19まで動かして
    if i%2 == 1:                # もしiが奇数なら
        bb.append(i*i)          # i*iをリストaに付け加える

print bb                        # bbをプリントする
```

実行結果の例

```
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361]
```

Python には<u>リスト内包表記</u>という非常に便利なリストの生成法があります。リスト内包表記を使えば，上のようなリストをたった一行で作ることができます。リスト内包表記を使って上と同じリストを作って表示させるプログラムがこちらです：

```
cc = [i*i for i in range(10)]       # リスト内包表記
print cc
```

実行結果の例

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

結果は上と同じですが，リストを簡潔に定義することができました。さらにリスト内包表記で要素を生成するときに**条件を付ける**こともできます。上で作ったリスト bb は次のように簡潔に作ることができます。

```
1 dd = [i*i for i in range(20) if i%2==1]
2 print dd
```

実行結果の例

```
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361]
```

さらに，次のように二つの変数を動かして要素を生成することもできます：

```
1 lis = ['a', 'b', 'c']
2 ee = [ (i,j) for i in range(1,5) for j in lis]
3 print ee
```

実行結果の例

```
[(1, 'a'), (1, 'b'), (1, 'c'), (2, 'a'), (2, 'b'), (2, 'c'),
 (3, 'a'), (3, 'b'), (3, 'c'), (4, 'a'), (4, 'b'), (4, 'c')]
```

20 関数

あるまとまった処理を単純に繰り返すときには for や while で繰り返せばよいのですが，その処理をいろんな場所で自由に呼び出して使いたいときには**関数**を使うのが便利です。関数はひとかたまりの処理に名前を付けて，簡単に使い回しできるようにしたものです。関数を使うことで一回書いたコードを再利用でき，スマートで読みやすいプログラムを作ることができるようになります。

図 6 関数でプログラムを再利用

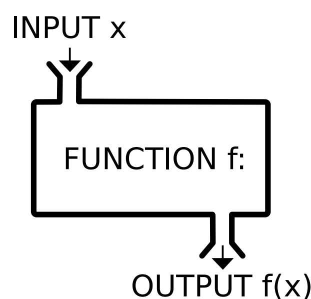

図 7 関数のイメージ

20.1 関数の定義のしかた

関数の定義

```
1 def 関数名(引数):
2     [処理 X を記述したブロック]      # 関数が実行するプログラム
3     return [戻り値]                  # 戻り値が必要ない場合は省略可能
```

- 関数は def 文で定義します。
- def の後に関数名と引数を書いたら右端にはコロン『:』を付けます。
- 関数の定義が終わるまでインデントを保つ。
- 関数名の大文字と小文字は区別されます。
- 引数や戻り値は不要なら省略できます。

それでは次の例題を見てみましょう。

- ファイル名：**func1.py**

```
1 def greeting():            # ここが
2     print "I'm fine."      # 関数の定義
```

```
3
4 print 'How are you?'         # How are you?と表示する
5 greeting()                   # 関数を呼び出す
```

実行結果の例

```
How are you?
I'm fine.
```

上のプログラムでは greeting() は呼び出されたら print "I'm fine" を実行する関数です。関数は定義された時点では実行されず，プログラム中で呼びだされたときにだけ動きます。この関数は引数も戻り値も持ちません。

次に引数と戻り値（return）があるプログラムの例を見てみましょう。

- ファイル名：**func2.py**

```
1 def sanjou(a):        # 関数名はsanjou，引数はa
2     return a*a*a      # aの3乗を返す
3
4 print sanjou(2)       # 返された8をプリントする
5 print sanjou(3)       # 返された27をプリントする
6 b = sanjou(1)+sanjou(2)+sanjou(3)+sanjou(4)
7 print b
```

実行結果の例

```
8
27
100
```

sanjou( ) は引数の三乗を返す関数ですが，上のように<u>返された値</u>をプリントしたり代入したりすることができます。次の例では引数 a が奇数なら odd，偶数なら even を返す関数を定義しています：

- ファイル名：**func3.py**

```
1 def guuki(a):
2   if a%2 == 0:          # もしaが偶数なら
3      return 'even'      # evenを返す
4   else:                 # そうでなければ
5      return 'odd'       # oddを返す
6
7 print guuki(1232), guuki(99)
```

実行結果の例

```
even odd
```

次にもう少し実用的な関数を作ってみます。西暦 n 年に対してその年が閏年なら True を，そうでなければ False を返す関数 uruuQ を作ります。さらに西暦 1000 年から 2017 年までの閏年をすべて表示します。

- ファイル名：**func4.py**

```
1 def uruuQ(n):
2     if n%400 == 0:
3        return True
4     elif n%100 == 0:
5        return False
6     elif n%4 == 0:
7        return True
8     else:
9        return False
10
```

```
11 for i in range(1000,2018):
12     if uruuQ(i):        # もしi年が閏年なら
13         print i,        # iをプリントする
```

実行結果の例

```
1004 1008 1012 1016 1020 1024 1028 .......... 1984 1988 1992 1996 2000 2004 2008 2012
```

次に羊を好きなだけ数える関数を定義してみます：

- ファイル名：**func5.py**

```
1 # -*- coding: utf-8 -*-
2 def CountSheeps(a):
3     for i in range(1,a+1):            #iを1からaまで繰り返す
4         print '羊が' + str(i) + '匹'    # 羊がi匹
5
6 CountSheeps(45)      # 関数を呼び出す
```

実行結果の例

```
羊が 1 匹
羊が 2 匹
・・・
・・・
羊が 45 匹
```

関数を別の変数に代入することも出来ます

```
1 def nobu():
2     print 'De aruka'
3
4 oda = nobu        # 関数nobuを新たな変数odaに代入
5 oda()             # odaを実行
```

実行結果の例

```
De aruka
```

20.2　名前のない関数 — lambda 式

lambda 式と呼ばれる特別な文法があり，名前がなくて引数と返り値の対応だけを指定して関数を定めることができます。これは対応だけを指定したいが，わざわざ名前を付けたくないときに用います。そして Python では `lambda` は特別な意味を持つ予約語なので変数名などで使ってはいけません。

lambda 式の文法

```
1 lambda x, y : (xとyを使った式)
```

引数と戻り値の間はコロン：で区切ります。

例えば n に n^2 を対応させる関数 f を lambda 式を使って次のように書くことができます：

```
1 >>> f = lambda a : a*a     # fは引数aに対してa*aを返す関数
2 >>> f(3)
3 9
```

また，lambda 式はリストの要素になることが出来ます。

```
1 a = [lambda x, y : x+y,   lambda x,y: x-y,   lambda x,y: x*y]
2
```

```
3 for f in a:
4     print f(3,5)
```

実行結果の例

```
8
-2
15
```

20.3 関数とローカル変数

関数の定義の中で新しく定義された変数は，その定義の中だけで**一時的に利用**できます。このような変数のことを**ローカル変数**といいます。これは関数の処理をそこだけで完結させるために補助的に用いるものです。ローカル変数はそれが定義された関数の外では使えません。これによって変数に使う文字を節約することができます。ローカル変数でない変数を**グローバル変数**といいます。

次のような数学の問題を考えてみましょう。$N = 100$ とするとき，$S = \sum_{k=1}^{N} k^2$ を求めなさい。ここで，

$$S = 1 + 2^2 + 3^2 + \cdots + 100^2 = \sum_{k=1}^{N} k^2 = \sum_{j=1}^{N} j^2 = \sum_{\ell=1}^{N} \ell^2 \tag{2}$$

なので，Σ の和を取るための変数 k, j, ℓ に特に意味は無く，他の文字[*6]を使っても構いません。そして (2) の和で使われている k, j, ℓ は Σ の外では意味を持ちません。一方，S と N には決まった数が代入されています。S と N に対応するものが変数がグローバル変数で，k のようにある処理の外では意味が無い変数がローカル変数に対応します。

次のプログラムは台形の面積を返す関数を定義していて，c と s はローカル変数になります。

- **ファイル名：daikei.py**

```
1 def daikei(a,b,h):    #a上底, b下底, h高さ
2     c = a+b           #cはローカル変数
3     s = 1.0*c*h / 2   #sはローカル変数
4     return s
5
6 S = daikei(3,5,8)     # Sはグローバル変数
7 print S
8
9 print c   # これはエラー
```

実行結果の例

```
32.0
Traceback (most recent call last):
  File "daikei.py", line 8, in ?
    print c
NameError: name 'c' is not defined
```

20.4 関数の説明の書き方

関数の定義を書くときに，その関数がどういう処理を行うのかをコメントとして書いておきましょう。コメントを書くことは次のような意味があります。

1. 関数の処理を書く前にコメントを書くことで，行いたい処理が明確になる。
2. 後になってプログラムを見た時に，何をやっていたのか思い出すことができる。。

[*6] ただし，他で使われていないもの

3. 他人が読むときの助けになる。

Python では関数の定義（def）の直後にコメントを書く慣習があります。そこでは関数が行う処理をコーテーション『""" と """』で囲んで文字列として書きます。

たとえば，羊を数える関数 gcd(a,b) を定義するときには，次のようにコメントを書くとよいでしょう：

```
1 # -*- coding: utf-8 -*-
2 def CountSheeps(a):
3     """ 羊を a 引き数える関数 """
4     for i in range(1,a+1):            # i を 1 から a まで繰り返す
5         print '羊が' + str(i) + '匹'   # 羊が i 匹とプリントする
6 
7 CountSheeps(45)
```

上では 3 行目が関数の説明ですが，これは単なる文字列なのでプログラム実行時には何もしません。

20.5 def 文の応用：素数を判定する関数

以前のプリントで，与えられた数字に対してそれが素数かどうかを判定するプログラム (while_primeQ.py) を作りましたが，それをもとに素数判定の関数を定義してみましょう。次で定義する関数は，引数 n が素数なら True を返し，そうでなければ False を返します。

- ファイル名：**primeq.py**

```
1  # -*- coding:utf-8 -*-
2  def primeQ(n):
3      """ nが素数ならTrue, 合成数ならFalseを返す関数を定義 """
4      if n < 2:
5          return False
6      else:
7          i=2
8          while i**2 <= n:
9              if n%i == 0:    #nがiで割り切れるなら
10                 return False
11             i=i+1
12     return True
13 
14 # リスト内包表記をつかって, 1から100までの素数を列挙する。
15 print [k for k in range(1,101) if primeQ(k)]
```

実行結果の例

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97]
```

上のプログラムは小さい数に対してはすぐに答えを出しますが，例えば

$$111\ 111\ 111\ 111\ 111\ 111\ 111\ 113$$

のような大きい数が素数かどうか判定するには長い時間がかかってしまいます（実際この数は素数です）。大きな数の素数判定を行うにはそれなりの工夫が必要になります。多項式時間で素数を判定する AKS 素数判定法や確率的だが高速で素数判定を行う Miller-Rabin 素数判定法といったものが知られています。一般に高速に動作するプログラムを作成するには，高度な数学的知識が必要になります。後半で解説する Sage では素数かどうかを高速に判定する関数 is_prime がデフォルトで用意されています。

20.6 練習問題

20.6.1 最大公約数を計算する関数

18.2 節の練習問題を参考にして，与えられた 2 つの自然数 a, b の最大公約数を返す関数を定義しなさい。そして，その関数を使って 23954187074819 と 8326543 の最大公約数を求めなさい。次のファイルの*****を推測すればよい。

- ファイル名：**def_gcd.py**

```
# -*- coding:utf-8 -*-
def gcd(a,b):
    """ aとbの最大公約数を求める関数
        ユークリッドの互除法により最大公約数を計算する
    """
    ************:        # while文開始
        *************    # (a,b)を(b,r)に同時の置き換える。rはaをbで割ったあまり
    return *

print gcd(23954187074819,8326543)
```

実行結果の例

```
32399
```

20.6.2 円周率を近似する分数

規約分数 a/b で円周率を近似するものを見つけたい。次のアルゴリズムを実行する Python プログラムを作成せよ。ファイル名は approxPi.py とすること。

(1) pi を 3.141592653589793 とする。
(2) a と b の最大公約数を返す関数 gcd(a,b) を定義する。
(3) $a = 3, b = 1, d = 1$ とする。(d は誤差評価に用いる)
(4) 次を $a < 320000$ かつ $b < 100000$ である間は次の (5)–(8) を繰り返す。
(5) apr を $apr = 1.0 * a/b$ とおく（円周率の近似値になる予定)。
(6) err を $err = |\pi - apr|$ とおく (絶対値は abs(-3) のように計算する)。
(7) もし a, b が互いに素かつ $err < d$ ならば a, b, apr, err の値をプリントし，err の値を d に入れる。
(8) もし $\pi > apr$ なら a の値を一つ増やし，そうでなければ b の値を一つ増やす。