# リストと関数

## リスト

　BasicやCには配列というデータ構造がありました。pythonには配列より便利なリストというデータ構造があります。BasicやCの配列は要素数を決めて宣言し、要素数を後から変更することはできませんでした。それに対してpythonのリストは要素数が固定されていません。例を示します。

my\_list = [10, 20, 30]

print(my\_list) # 配列全体を表示

a = my\_list[1] # 2 個目の要素を取り出す

n = len(my\_list) # 配列の要素数

my\_list2 = []

my\_list2.append(40) # リストの末尾に追加

print(my\_list2)

my\_list2.append(50)

print(my\_list2)

my\_list2.append(60)

print(my\_list2)

　リストは

my\_list = [要素1, 要素2, 要素3]

のように作成します。この場合my\_listというリストは3個の要素を持ちます。

　最初に空のリストを作成し、要素を追加する方法もあります。空のリストを作成し、末尾に要素を追加するとき、以下のように書きます。

my\_list = []

my\_list.append(要素)

　リストの要素数は

n = len(my\_list)

で得られます。リスト中の個々の要素を取り出すには

element = my\_list[2]

のように書きます。[ ] の中の添字は0からはじまります。すなわち1番目の要素の添字は0、n番目の要素の添字はn-1です。n個の要素を持つリストのとき、添字の範囲は0 ～ n-1です。

　my\_listという配列変数の3番目の要素を削除するには

my\_list.pop(2)

と書きます。削除した要素は

element = my\_list.pop(2)

のように、取り出すこともできます。

　ここまでの例ではリストの中に数値が入っていました。リストの要素は文字列が入ってもOKです。文字列と数値が混在してもOKです。

　リストの連結は以下のように書きます。

my\_list1 = ["a", "b", 10] # 文字列と数値を混在させてもよい

my\_list2 = ["c", 20, "d"]

my\_list = my\_list1 + my\_list2

print(my\_list) # ["a", "b", 10, "c", 20, "d"] となる

　リストから要素を1個ずつ取り出すには以下のように書きます[[1]](#footnote-1)。

my\_list = ["a", "b", "c"]

for element in my\_list:

print(element)

　以前学習したfor文のfor i in range(0, 4) という書き方は、以下のように書くのと同じ動作をします。

for i in [0, 1, 2, 3]:

print(i)

　このようにリストやrangeオブジェクトから1個ずつ値を取りだして、ループを回すことができます[[2]](#footnote-2)。

　インデックスを使って以下のように書くこともできますが、冗長であり、pythonでは推奨されません。

my\_list = [10, 20, 30]

n = len(my\_list)

for i in range(0, n):

print(i, my\_list[i])

### 【例題】

　次のリストの平均、最大、最小を求めて下さい。値の範囲は0～100です。

my\_list = [10, 53, 29, 83, 90, 40, 73]

### 【解答】

sum = 0

max = -999

min = 999

for value in my\_list:

sum = sum + value

if value > max:

max = value

if value < min:

min = value

average = sum / len(my\_list)

print("average =", average)

print("max =", max)

print("min =", min)

　上の例ではmaxとminを愚直に求めましたが、以下のようにすると簡単に求まります。

max = max(my\_list)

min = min(my\_list)

## アンパック代入

　pythonでは以下のように代入することができます。

a = 10

b = "abc"

c, d = a, b

print(c, d)

　イコールの右辺にある値が左辺の各変数に順番に割り当てられます。ここではcに「aの値」が代入され、dに「bの値」が代入されます。以下のようにリストの値を割り当てることもできます。

my\_list = ["aaa", "bbb"]

a, b = my\_list

print(a, b)

## 関数

　pythonにおける関数はExcel VBAのプロシージャ（Sub）と関数（Function）のどちらとも少し異なります。まずは次の例題を見てください。

def func1(c, d):

print(c, d)

c = 30

d = 40

print(c, d)

a = 10

b = 20

func1(a, b)

print(a, b)

　defで定義しているのがExcel VBAのSubやFunctionに対応する機能です。この場合、func1という名前の関数を定義しています。名前は好きな名前を付けて構いません。ただし、「数字からはじまる」「スペースを含む」「特殊文字（!, @, $) を含む」「def, if, elseなど予約語を含む」名前は禁止です。defの行の行末には : を付けます。インデントの範囲が関数の範囲です。この場合は、defから2回目のprint(c, d) までが関数func1の範囲です。

　pythonでは「関数」は「関数を呼び出す場所（ここではfunc1(a, b)）より手前にないといけません。また、関数の定義は関数を呼び出す場所より手前であれば、どこに書いてもかまいません。この場合「def ～ 2回目のprint(c, d)」の部分を、たとえば「b = 20 と func1(a, b) の間」に書いても問題なく動作します。

　関数の引数は順番に結合されるので、呼び出す側の変数aとbが関数func1において、cとdとして扱われます。ですから、1回目のprint(c, d) の実行時においてc = 10, b = 20です。2回目のprint(c, d) の実行時においてはc = 30, d = 40に更新されています。関数の実行を終了して、呼び出し元に戻ってきてからprint(a, b)を実行しています。このときa = 10, b = 20です。

　Excel VBAとは異なりpythonでは関数側で引数の値を書き換えても、その変更が呼び出し元に影響しません。

● 呼び出し元へ情報を返す

　関数側から呼び出し元へ情報を渡すにはreturn文を使います。以下の例を見てください。

def func1(c, d):

print(c, d)

sum = c + d # 和

sub = c - d # 差

return sum, sub

a = 10

b = 20

e, f = func1(a, b) # 2個の数値をe, fに受け取る

print(e, f)

　returnで2個の数値sumとsubを返し、それをeとfで受け取っています。複数の値を代入するとき、並び順に結合されます。sumの値がeに入り、subの値がfに入ります。

【例題】

　次のinput\_listを入力とするとき、output\_listとして0以上の数だけを取り出して、新しいリストとして返す関数を作りなさい。

input\_list = [10, -20, 3, 0, -5, 50]

output\_list = [10, 3, 0, 50]

【解答】

def delete\_minus\_value(in\_list):

out\_list = []

for element in in\_list:

if element >= 0:

out\_list.append(element)

return out\_list

input\_list = [10, -20, 3, 0, -5, 50]

output\_list = delete\_minus\_value(input\_list)

print(output\_list)

1. my\_listという変数名を使っています。当初listという変数名でしたが、組み込み関数のlistを上書きしてしまうので、変数名をmy\_listに変更した。 [↑](#footnote-ref-1)
2. range(0, 4) はrangeオブジェクトと呼ばれ、[0, 1, 2, 3] と同等に扱えますが、厳密には異なります。list(range(0, 4)) とすると、[0, 1, 2, 3] というリストに変換できます。 [↑](#footnote-ref-2)