9-5 ラムダ式

本節で学習するのは、“複合文”としての構造をもつ関数を、“単一の式”として実現するラムダ式です。

ラムダ式

関数定義が、一種の複合文であることを、本章の冒頭で学習しました。文ではなく、式として関数を実現するのが、ラムダ式（*lambda expression*）です。

ラムダ式の基本的な形式は、次のとおりです。

lambda 仮引数並び : 返却値 ラムダ式

▶ 複数の仮引数を受け取る場合は、コンマで区切って並べます。

ラムダ式と呼ばれるのは、lambda が演算子だからです。

▶ 既に学習したように、○○演算子を適用した式は、○○式と呼ばれるのでした。

一般に、式の中に文を入れることはできませんので、ラムダ式の中に文を含めることはできません。また、単なる式ですから、アノテーションを含めることもできません。

さて、lambda 演算子が行うことは、無名関数（*anonymous function*）と呼ばれる関数オブジェクトを生成することです。

▶ 無名関数は、匿名関数（とくめい）とも呼ばれます。

List 9-45 に示すのは、二つの値の和を求める関数オブジェクトをラムダ式によって生成して、それを呼び出すプログラムです。

List 9-45 chap09/list0945.py

```
"""２値の和を求めるラムダ式"""

a = int(input('整数a：'))
b = int(input('整数b：'))

add2 = lambda x, y: x + y
print('aとbの和は', add2(a, b), 'です。')
```

実行例

```
整数a：5
整数b：7
aとbの和は 12 です。
```

青網部がラムダ式です。ちょうど、以下の関数定義と、（文法上の扱いなどの詳細は異なるものの）同じような働きをします。

```
def 名前無し(x, y):
    return x + y
```

lambda 演算子の後ろに、上記の緑網部を並べると、青網部のラムダ式が完成します。

ラムダ式で作られるのは、関数オブジェクトであり、そのオブジェクトに add2 という名前が与えられています。赤網部では、関数オブジェクトを呼び出しています。なお、呼出しの形式は、通常の関数呼出しと同じです。

さて、プログラム中で1回しか呼び出さないのであれば、わざわざ関数に対して名前を与える必要がありません。

一般的に、式は、他の式の一部になり得ますので、ラムダ式も、他の式の一部として記述できます。それを利用して書きかえたのが、List 9-46 のプログラムです。

List 9-46 chap09/list0946.py

```
"""２値の和を求めるラムダ式（その２）"""

print('aとbの和は', (lambda x, y: x + y)(a, b), 'です。')
```

ラムダ式による関数定義と、その呼出しが、一つの式の中に収まりました。

＊

さて、二つの値の和を求める関数は、前節の高階関数のプログラム例でも使いました。

九九の掛け算表と足し算表を表示する List 9-44（p.285）のプログラムを、ラムダ式で書きかえたのが、List 9-47 のプログラムです。

List 9-47 chap09/list0947.py

```
"""九九の掛け算表・足し算表を表示"""

def kuku(func):
    """九九の表を表示"""
    for i in range(1, 10):
        for j in range(1, 10):
            print('{:3d}'.format(func(i, j)), end='')
        print()

n = int(input('掛け算[0]／足し算[1]：'))

if n == 0:
    print('九九の掛け算表')
    kuku(lambda x, y: x * y)
elif n == 1:
    print('九九の足し算表')
    kuku(lambda x, y: x + y)
```

List 9-44 と同じ

実行例

```
1 掛け算[0]／足し算[1]：0↵
九九の掛け算表
  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

2 掛け算[0]／足し算[1]：1↵
九九の足し算表
  2  3  4  5  6  7  8  9 10
  3  4  5  6  7  8  9 10 11
  4  5  6  7  8  9 10 11 12
  5  6  7  8  9 10 11 12 13
  6  7  8  9 10 11 12 13 14
  7  8  9 10 11 12 13 14 15
  8  9 10 11 12 13 14 15 16
  9 10 11 12 13 14 15 16 17
 10 11 12 13 14 15 16 17 18
```

書きかえたのは、関数 *kuku* ではなく、それを呼び出す青網部です。関数 *kuku* に与える実引数を、ラムダ式に変更しています。

そのため、2値の積あるいは和を求める（無名の）関数オブジェクトが生成されるとともに、その参照が関数 *kuku* に渡されます。

重要 ラムダ式を使えば、無名の関数オブジェクトを、文ではなく式として生成できる。

なお、ラムダ式では、関数の引数と同様に、引数のデフォルト値、キーワード引数、可変個引数なども指定できます。

map 関数とラムダ式

ラムダ式の応用例として、組込み関数である map 関数を学習します。この関数の基本的な呼出し形式は、次のとおりです。

```
map( 関数 , イテラブルオブジェクト )
```

map 関数は、第2引数に受け取ったイテラブルオブジェクトの全要素に対して、関数を適用した結果生成されるイテレータを、map 型の map オブジェクトとして返却します。

List 9-48 と List 9-49 に示すのが、プログラム例です。

List 9-48 chap09/list0948.py

```
# リストの全要素を２倍にする（関数）

def double(n):
    return 2 * n

x = [1, 2, 3, 4]
y = map(double, x)

print(list(y))
```

実行結果

```
[2, 4, 6, 8]
```

List 9-49 chap09/list0949.py

```
# リストの全要素を２倍にする（ラムダ式）

x = [1, 2, 3, 4]
y = map(lambda n: 2 * n, x)

print(list(y))
```

実行結果

```
[2, 4, 6, 8]
```

いずれも、リスト [1, 2, 3, 4] に入っている全要素の値を2倍したリストを生成・表示するプログラムです。

List 9-48 は、関数版です。受け取った引数の2倍の値を返却する関数 *double* を定義して関数オブジェクトを生成した上で、それを map 関数の第1引数として与えています。

map 関数が生成・返却するのは、*x* の全要素に関数 *double* を適用した結果の並びです。それは map オブジェクトですから、そのままでは表示できません。そのため、*y* をいったんリストに変換した上で出力しています（表示は [2, 4, 6, 8] とリスト形式で行われます）。

List 9-49 は、ラムダ式版です。前ページのプログラムと同様に、関数オブジェクトを生成するラムダ式を、そのまま引数として渡しています。

▶ ラムダ式が生成する関数オブジェクトが受け取る仮引数が *n* で、返却するのは 2 * *n* です。

*

map 関数を用いた変換の例をいくつか考えましょう（*x* は [1, 2, 3, 4] であるとします）。

- **全要素を文字列に変換する**（'chap09/map_sample01.py'）

```
list(map(str, x))
```

第2章で学習した str 関数は、受け取った引数を文字列に変換する組込み関数です。そのため、生成されるリストは、['1', '2', '3', '4'] となります。

- **全要素の単位をセンチメートルからインチに変換する**（'chap09/map_sample02.py'）

```
list(map(lambda n: 2.54 * n, x))
```

生成されるリストは、[2.54, 5.08, 7.62, 10.16] です。

filter 関数とラムダ式

次にラムダ式を応用するのは、組込み関数である filter 関数です。この関数の基本的な呼出し形式は、次のとおりです。

```
filter( 関数 , イテラブルオブジェクト )
```

この関数は、第2引数に受け取ったイテラブルオブジェクトの全要素の中で、関数を適用した結果が真となる要素のみを抽出したイテレータを、filter 型の filter オブジェクトとして返却します。

List 9-50 に示すのが、プログラム例です。

List 9-50 chap09/list0950.py

```
# 80点以上の点数のみを抽出

import random

number = int(input('学生の人数：'))

tensu = [None] * number

for i in range(number):
    tensu[i] = random.randint(0, 100)

print('全員の点数＝', tensu)
print('合格者の点数＝', list(filter(lambda n: n >= 80, tensu)))
```

実行例

```
学生の人数：11↵
全員の点数＝ [29, 71, 68, 57, 61, 50, 92, 98, 58, 71, 53]
合格者の点数＝ [92, 98]
```

tensu は、0 点～ 100 点のテストの点数を格納したリストです。要素数はキーボードから読み込んで、各要素の値は乱数で決定します。

filter 関数に行わせていることは、合格者（80 点以上）の点数の抽出です。そのために、受け取った引数の値が 80 以上であれば True を、そうでなければ False を返却する関数をラムダ式で生成して第1引数として与えています。

呼び出された filter 関数は、第2引数に受け取ったリストの全要素を順に走査します。その際、各要素に第1引数に受け取った関数を適用し、真となった（80 点以上になった）点数のみを抽出して、新しい並びを filter オブジェクトとして生成します。

返却された filter オブジェクトは、そのままでは表示できませんので、いったんリストに変換した上で表示しています。