第 3 章

Python の文法 (2/2) リスト，ループ，辞書，集合，ファイル入出力

「退屈なことは Python にやらせよう」(Automate the boring stuff with Python) という本が出版されているぐらい，コンピュータは機械的てつまらない作業をさせるにはうってつけです。一番の強みは，何万回，何億回と繰り返すことも厭わないことで，プログラムではループ (loop) を使うことで実現できます。本章では代表的な for ループと，沢山のデータを格納するためのデータ構造（リスト，辞書，集合）について学んでいきます。

3.0.1 リストの基本操作と for ループ,while ループ

リスト (list) は，複数の値をまとめて格納できるデータ構造です。[値 1，値 2，...] の形式で定義し，先頭の要素が 0 番目，次が 1 番目，…とインデックスが付きます。

ソースコード 3.1 list_number.py：リストを使って 0 から 20 まで表示する

```
1 # list_number.py: リストを使って 0から 20まで表示する
2
3 # int_array リストに 0 から 20 を入れる
4 int_array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
5              13, 14, 15, 16, 17, 18, 19, 20]
6 print(int_array) # [, ]付きで出力
```

■for ループ　最初の章で使った print_number.py(p.3) は，for ループという繰り返し命令を使って簡単に書くことができます。

```
for 変数 in リスト:
    インデントブロック
```

リストの要素が順番に変数に引き渡され，これを使ってインデントブロック内で処理を行う，という処理を全てのリスト要素に対して実行されるイメージです。

この for 文を使い，print 関数に end=’ ’と指定することで，改行の代わりに半角スペースが入り，数字を横並びにしてリスト内のデータを全て出力できるようになります。

ソースコード 3.2 list_number.py

```
5 # int_array の中身をすべて表示
```

```
for num in int_array:
    print(num, end='␣')
print() # 最後に改行
```

リストを最初から生成するためには

```
# int_array を for ループで生成
int_array = [] # 空のリストを生成
for num in range(21):
  int_array.append(num)
print(int_array) # 出力して確認
```

のように，リストの最後尾に値を付加するための append 関数を使って書くことができます。また，後置形という形で 1 行で下記のように書くこともできます。

```
# 後置型for 文でも書ける
int_array = [num for num in range(21)]
print(int_array) # [, ]付きで出力
```

リストの並びを逆に表示するには下記のように reversed 文で対応できます。

```
# 逆順で表示
for num in reversed(int_array):
    print(num, end='␣')
print()
```

リストに対して使用できる主な操作をまとめます。

変数名 `[i]`　インデックス i 番目の要素を取り出す（先頭は 0 番目）。

`reversed(`**リスト**`)`　要素の並びを逆順にしたイテレータを返す。

リスト`.append(`**値**`)`　リストの最後尾に値を追加する。

append 関数はリストを動的に拡張する際によく使います。例えば，後の節で紹介するエラトステネスの篩では，新たに見つかった素数を `prime_number.append(i)` でリストに追記していきます。

■while **ループ**　条件文が真 (true) であるうちは繰り返しを続けるループを書く時には while ループが便利です。

```
while 条件文:
    インデントブロック
```

while ループを使うと下記のように順表示も逆順表示も可能です。

ソースコード 3.3　`while_ex.py`

```
# while_ex.py: while ループの例
# while 文で 0 から 20 を正順と逆順に表示
min_num = 0
while min_num < 21:
  print(min_num, end='␣')
  min_num += 1
print()

# 逆順
max_num = 20
while max_num >= 0:
```

```
12    print(max_num, end='␣')
13    max_num -= 1
14  print()
```

問題 3.1

n を与えた時，1 から n までの自然数の和と積を計算するスクリプト sum_product.py を作れ。リストを使っても使わなくても実装は可能。

3.1 タプル (tuple)：変更のできないリスト

タプル (tuple) は，リストと同様に複数の値をまとめたデータ構造ですが，**一度作成すると要素を変更できない**という点が異なります。(**値 1**，**値 2**，...) のように丸括弧で定義します。

ソースコード 3.4　tuple_number.py：タプルを使って 0 から 20 まで表示する

```
1  # tuple_number.py: タプルを使って 0から 20まで表示する
2
3  # int_array タプルに 0 から 20 を入れる
4  int_array = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
5               14, 15, 16, 17, 18, 19, 20)
6  int_array = (i for i in range(21)) # 後置型for 文（ジェネレータ）
7  print(int_array) # [, ]付きで出力
8
9  # int_array の中身をすべて表示
10 for num in int_array:
11     print(num, end='␣')
12 print() # 最後に改行
13
14 # 逆順で表示→エラーになる
15 for num in reversed(int_array):
16     print(num, end='␣')
17 print()
```

後置型 for 文を丸括弧で囲んだ場合，リストではなく**ジェネレータ (generator) オブジェクト**が生成されます。ジェネレータは一方向にしか走査できないため，reversed を適用しようとすると次のようなエラーが発生します。

tuple_number.py の実行結果（エラー）

```
<generator object <genexpr> at 0x00000020622676980>
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Traceback (most recent call last):
  File "...\tuple_number.py", line 14, in <module>
    for num in reversed(int_array):
TypeError: 'generator' object is not reversible
```

リストとタプルの主な違いを表 3.1 にまとめます。

特徴	リスト	タプル
記法	[ ]	( )
要素の変更	可能	不可
append による追加	可能	不可
reversed の適用	可能	リストに変換すれば可能
用途	動的に変化するデータ	変更されてはいけない固定データ

表 3.1　リストとタプルの比較

3.2　エラトステネスの篩：素数判定法

素数 (prime number) とは，1 と自分自身以外の約数を持たない自然数で，1 は除きます。小さい順に並べると, 2, 3, 5, 7, 11, 13, 17, 19, 23, ... となり，感覚的には，次第に間隔が開きながらも，無数に存在していることが分かります。素数を知ることで，整数の素因数分解 (prime factorization) が可能となり，最大公約数や最小公倍数を自動的に求めることも可能となります。ここでは，一番原始的な素数判定法であるエラトステネスの篩 (Sieve of Eratosthenes) をリストを使って実装していきます。

エラトステネスの篩は，2 以上 n 以下の素数を次のようにして割り出していきます。

1. 3 以上 n 以下の自然数のうち，2 で割り切れるものを取り除く
2. 残った自然数のうち，3 で割り切れるものを除く
3. 以下同様

下記のスクリプトでは，整数の剰余を求める演算子 %(パーセント）を使い，残った素数のうち小さいものを prime_number リストに追加しています。

ソースコード 3.5　e_sieve.py

```
1  # e_sieve.py: エラトステネスの篩,素数を取り出し
2  input_str = input('Input␣integer␣=?␣')
3  max_num = int(input_str)
4
5  # 素数を格納する
6  prime_number = [2] # 2以上を素数判定する
7
8  # メインループ
9  for i in range(3, max_num + 1):
10     flag_prime = 0 # 0のままだと素数
11     for pnum in prime_number:
12         if i % pnum == 0:
13             flag_prime = 1 # 素数でない
14             break
15
16     # 既存の素数にはないもの
17     if flag_prime == 0:
18         prime_number.append(i) # i を素数に追加
19
20 # 素数をリストアップ
```

```
21 print('Prime␣number␣(<', max_num, '):', prime_number)
```

このスクリプトを基に，n を入力値とし，2 以上 n 以下の素数リストを返す prime_list 関数を実装して上記スクリプトと同じ結果が返ってくることを確認して下さい。

問題 3.2

指定された整数を素因数分解するスクリプト prime_factorization.py を作れ。また，これに基づいて与えられた整数の素因数分解リストを返す prime_fact 関数を実装せよ。

なお，素因数分解のアルゴリズムは次のようになる。

1. 指定された整数までの素数リストを取得 [ヒント:prime_list 関数を使用]
2. 小さい素数から与えられた整数が割り切れるかどうかを確認し，割り切れれば素因数リスト factlist に追加。これを割り切れなくなるまで続ける

3.3 辞書と集合

要素の順序が不定でも構わないデータの塊を扱いたい時には，辞書 (dictionary) や集合 (set) が便利です。どちらも中カッコ ({ }) を用いて表記される点は共通ですが，辞書にはデータ一つ一つに索引にあたる key が付加されますが，集合は索引がありません。

3.3.1 辞書の例

例えば，「英単語: 日本語」の対を集合で定義すると次のように表記されます。データの取り出しはキー (key) を使って行います。

ソースコード 3.6 word_dict.py

```
1 # word_dict.py: 辞書の例
2 mammal_dict = {'cat':'猫', 'dog':'犬', 'horse':'馬', 'cow':'牛', 'squerrel':'リス'}
3
4 # key を使って値を取り出し
5 print('cat␣=>', mammal_dict['cat'])
```

辞書は，添え字として文字列が指定できるリストと考えると良いでしょう。

word_dictpy の実行結果 (1/2)

```
cat => 猫
```

ループで回すときには items() 関数を用いることで，キーと値を同時に取り出すことができ案す。

ソースコード 3.7 word_dict.py

```
6 # key と値を同時に取り出し
7 for key, val in mammal_dict.items():
8     print(key, '->', val)
```

これを実行すると，全ての辞書データがキーと値のセットとして表示されます。

word_dict.py の実行結果 (2/2)

```
cat -> 猫
dog -> 犬
horse -> 馬
cow -> 牛
squerrel -> リス
```

問題 3.3

word_dict.py を次の動作が可能なように改良せよ。

1. 「cat」を入力すると「猫」が表示される
2. 「猫」と入力すると「cat」が表示される

3.3.2 集合の例

先に述べたように，キーなしで値だけをまとめたものが集合になります。A, B が集合とすると，数学では

和集合 $A \cup B = \{c \mid c \in A \text{ または } c \in B\}$
積集合 $A \cap B = \{c \mid c \in A \text{ かつ } c \in B\}$
差集合 $A - B = \{c \mid c \in A \text{ かつ } c \notin B\}$

という定義がありますが，これと全く同じように集合を作ることができます。

ソースコード 3.8 set.py

```
1  # set.py: 集合の使い方
2  input_str = input('Input␣maxmum␣integer␣n␣=?␣')
3  n = int(input_str)
4
5  # A = { n 以下の偶数の集合 }
6  # B = { n 以下の 3 の倍数の集合 }
7  A = set() # 空集合
8  B = set() # 空集合
9  for i in range(1, n + 1):
10     if i % 2 == 0: A.add(i) # 要素 i を A に追加
11     if i % 3 == 0: B.add(i) # 要素 i を B に追加
12
13 print('A␣=␣', A)
14 print('B␣=␣', B)
15
16 # 和集合
17 print('A␣|␣B␣=␣', A | B)
18 print('A.union(B)␣=␣', A.union(B))
19
20 # 積集合
21 print('A␣&␣B␣=␣', A & B)
22 print('A␣intersection␣B␣=␣', A.intersection(B))
23
24 # 差集合
25 print('A␣-␣B␣=␣', A - B)
```

```
26 print(A.difference(B))
```

set.py の実行結果 (1/2)

```
Input maxmum integer n =? 10
A =  {2, 4, 6, 8, 10}
B =  {9, 3, 6}
A | B =  {2, 3, 4, 6, 8, 9, 10}
A.union(B) =  {2, 3, 4, 6, 8, 9, 10}
A & B =  {6}
A intersection B =  {6}
A - B =  {8, 2, 10, 4}
A.differernce(B) =  {8, 2, 10, 4}
```

要素の追加は add 関数，削除は discard 関数（エラーなし）か remove 関数（指定要素が見つからないとエラーになる）で実行できます。

ソースコード 3.9 set.py

```
28 # 要素の削除
29 print('A.discard(2)␣=␣', A.discard(2))
30 print('A.remove(2)␣␣=␣', A.remove(2)) # エラーになる
```

set.py の実行結果 (2/2)

```
A.discard(2) =  None
Traceback (most recent call last):
(略)
KeyError: 2
```

問題 3.4

上記のスクリプトで定義した集合 A と B を用いて対称差集合 $(A - (A \cap B)) \cup (B - (A \cap B))$ を求める部分を追記せよ。

3.4 ファイル入出力

ファイル (file) は，永続的にデータを補完するための入れ物です。コンピュータが扱えるデータは全て 2 進数 (0 と 1 の組）として表現できるデジタルデータですが，全てのデジタルデータが文字として認識される範囲であれば文字列となり，文字列だけからなるファイルのことをテキストファイルと呼びます。テキストファイル以外は全てバイナリファイルと呼びます。例えば，今まで作ってきた Python スクリプト（*.y) は全てテキストファイルですし，Word ファイル，画像ファイル (PNG, JPEG 等）や動画ファイル (MPEG 等）はバイナリファイルです。

Python でもファイルの入出力が可能です。ここではテキストファイルを扱う事例を見ていくことにします。

3.4.1 文字コードとテキストファイルの入出力

文字コード (character code) は，扱える文字集合（character set) と文字に割り当てる 0 以上の整数，すなわちエンコーディングスキーム (encoding scheme) がセットになったものの総称です。

コンピュータがまだ非力だったころは，欧米で使用するアルファベット・数字・記号のみを 7bit で表現できる ASCII コードで事足りていましたが，日本人向けには当然ひらがな・カタカナ・漢字が使用できないと困りますし，日本以上に膨大な感じを必要とする中国に至っては 16bit でも足りず，現在では全世界で標準的に使用される，絵文字も含めた文字集合としてユニコード (unicode) が標準となっています。このエンコードには 32bit が必要で，ASCII コードとの互換性を考慮して，これを 8bit ずつ区切って表現する形式が UTF-8 と呼ばれているものになります。以後，今まで作ってきた Python スクリプトにはコメントに日本語を使っていますが，これも UTF-8 で表現されたものになります。異なるエンコーディングスキームで読み書きを行うと，意図した文字が表示できないことになりますが，これが文字化けと呼ばれる現象の一因です。

ということで，下記のような UTF-8 で記述された日本語ファイル `student_grade_name.csv` を読み込み，1 行ごとに行番号を付加して表示する Python スクリプトを作ってみましょう。

```
学籍番号・氏名表, ,
, ,
学籍番号, 氏名, よみがな
A0001, 安藤流星, あんどうりゅうせい
A0002, 伊藤薫, いとうかおる
A0003, 井坂瑠々子, いさかるるこ
A0004, 梅坂武, うめさかたけし
```

図 3.1　`student_grade_name.csv` の冒頭部分

ファイルを扱う際には

1. ファイルを開く（open 関数）
2. ファイルのデータを読み書きする
3. ファイルを閉じる（close 関数）

というように，指定したファイルの開閉が必ず必要になります。Python の場合，open・close 関数でこの動作を行います。

ソースコード 3.10　`read_write_file.py`

```
1  # read_write_file.py: テキストファイルを読み書きする
2
3  # ファイル名の指定
4  input_filename = 'student_grade_name.csv' # 読み込み用
5  output_filename = 'linenum_student_grade_name.txt' # 書き込み用
6
7  # ファイルを開く
8  with open(input_filename, 'r', encoding='utf-8') as f_in: # 読み込み用
9      with open(output_filename, 'w', encoding='utf-8') as f_out: # 書き込み用
10         # 1行ずつ読み込み
11         num_line = 1
```

```
12          while line := f_in.readline():
13              # 標準出力
14              print(f'{num_line:5d}:{line}', end='␣')
15              # ファイル出力
16              f_out.write(f'{num_line:5d}:{line}')
17              num_line += 1
18
19          # ファイルを閉じる
20          f_out.close()
21  f_in.close()
```

このスクリプトを動作させた結果，出来上がるのが linenum_student_grade_name.txt でここに行番号が付加されたデータが書き込まれます。

問題 3.5

read_write_file.py を改良し，読み取ったテキストファイルの文字数をカウントして表示する機能を付加せよ。

演習問題

1. prime_factorization.py(P.28) の出力を Python のべき乗形式にせよ。例えば 2 2 3 3 3 5 という出力を得た時には，は 2**2 * 3**3 * 5**1 と表示する。
2. set.py を参照し，自然数 n を入力し，次の集合を求めるスクリプトを作れ。
   (a) $A = \{a \mid a \text{ は } 5 \text{ の倍数 }\}$
   (b) $B = \{b \mid b \text{ は } 6 \text{ の倍数 }\}$
   (c) $C = \{c \mid c \text{ は } 7 \text{ の倍数 }\}$
   (d) $D = A \cup B \cup C$
   (e) $E = A \cap B \cap C$