擬似コードを使ったPyhon入門用教材 (資料編)

竹内 いずみ

2006年1月11日

1 はじめに

――――――――――――――――――――――――――――――――

この教材は、オブジェクト指向プログラミング言語である Python の基礎を学ぶものである。ここで習う Python は、無料で入手できるだけでなく、コピーや配布が自由なパブリックドメインでもある。またオブジェクト指向言語であるために人間にとってプログラムを捉えやすくまた作成しやすい、作成したオブジェクトを再利用しやすいといった特徴がある。

教材の構成は、まず問題を提示する『問題』、その問題の捉え方を説明していく『問題把握』、プログラム内容を説明する『解説』、プログラムの擬似コードを提示する『擬似コード』、そして実際のプログラムを提示する『プログラム』の 4 つから成っている。その構成の中でも、『問題把握』では問題の入力方法と特に注意する点、そして簡単な処理の流れを説明している。さらに『解説』ではその処理の流れを詳しく説明していくなかで、使用したツールの解説をしている。Python の構文は簡潔なため、説明もまた Python のプログラム処理の流れに沿うようにしている。

そのため、この教材は構成ごとに順を追って学習して行くことで、プログラムがどのようにして作られていくのかといったアルゴリズムを自然に身に付けることができ、さらにツールの使い方も学べるようになっている。

この教材の中で表してある[　　]で囲ったもの、また**太字**で表されたものは、Python でよく使うステートメントや関数などのツールである。ツールは 13 章の『付録』でも紹介されている。

この教材は予備知識としては福田先生のプログラミングの初めての授業を受け、「変数への代入」が理解できる程度の知識から、第 1 学年の前期の授業を修めた程度の知識を持った人文情報学生を対象としている。

この教材を利用することで、Python のプログラミングに親しみ、Python を使ったプログラムを自分で作り出せるようになる切っ掛けとして欲しい。

目　次

2 条件判定

――――――――――――――――――――――――――――――――

(1) ＜＜問題 1 ＞＞

ユーザーに身長と体重の入力を求め、そこから標準体重と肥満度を割り出すプログラム。

(2) (問題把握)

身長と体重は対話的に入力を求めるようにし、それをもとに演算していく。

標準体重と肥満度を割り出すには実数での演算になるが、それを表示する時は、見やすいように小数点１以下で四捨五入する。また、肥満度が高かった場合、低かった場合、標準だった場合の３パターンのコメントも書き出すようにする。

処理の流れは、まず各質問を変数に代入し材料を取得した後、実際に計算していく。その結果を表示し、結果に応じて 3 パターンいずれかに当て嵌まったコメントを書き出す、となる。

(3) (解説)

まずユーザーから対話的に入力を求め、身長と体重をそれぞれ変数に代入する。身長が入る変数は height、体重が入る変数は weight としておく。

対話的な入力を受け取るには、**input(”質問”) 関数**を使う。これは質問に対しての答え入力を返す関数である。input() は数値を扱う時に用い、文字列を扱う場合は **raw_input(”質問”) 関数**を使う。

受け取った数値が標準体重と肥満度の演算でそのまま使うと、小数点以下は切り捨てられ、詳細な演算の答えが得られない。よって小数点以下の演算も行うように、実数に直しておく。これには **float(数値)** を使う。括弧の中には数値が入っていれば変数でもよい。同じように、文字列に直すには **str(文字列)**、整数に直すには **int(整数)** を用いる。

これらの材料が揃ったら、標準体重と肥満度を割り出す。標準体重を入れる変数は standard_weight とし、肥満度を入れる変数は fat とする。

標準体重を割り出す演算式は、「((身長　100) ** 2) * 22」。肥満度を割り出す式は「(体重　標準体重 - 1) * 100」なので、それぞれの式に対応する変数を当て嵌めた式を、変数 standard_weight と変数 fat に代入する。

次に演算結果を実際に書き出していくが、肥満度の結果については『太りすぎである』、『痩せ過ぎである』、『標準である』の３つに分類しコメントを書き出す。

まずはユーザーが解りやすいよう、演算によって得られた標準体重と、ユーザーの体重、肥満度を書き出す。ここで見やすいように、演算

結果をそれぞれ小数点第 1 位で四捨五入するが、2 通りのやり方で四捨五入する。

1 つ目、標準体重の書き出しでは組み込み関数の **round(数値)** を使う。組み込み関数とは、最初から Python に用意された関数のこと特別な宣言やインポートの必要ない。大抵のプログラムは組み込み関数を用いて作ることができる。

2 つ目、肥満度は**文字列フォーマット**を使う。文字列フォーマットとは、文字列の中に \ 演算子とキーを組み合わせて書き込んでおき、そこに挿入したいデータを文字列の後方、% 演算子で区切った後に置くと、プログラム実行時に文字列のなかにそのデータが置換され表示する処理を行う。置換したいデータが複数の場合は、( ) で括って挿入したい順に入れていく。

この文字列に書き込む % 演算子のキーには、文字列である **%s**、整数である **%d**、実数である **%f** などがある。さらに % 演算子とキーの間に、左右詰めをあらわす + −か空いた桁を 0 で埋めるフラグや、数値の場合全桁数を表す整数、小数点以下の桁数で四捨五入する . の後の整数、という形式で入れることができ、必要なキーだけ詰めて書く。つまり『実数で、全桁数が 4 で小数点以下が 2 で四捨五入する』を表すには、『%4.2f』となる。このプログラムの場合は『実数で、小数点以下が 1 で四捨五入』となる。

これら肥満度に応じた 3 パターンのコメントを比較し、その結果ごとに書き出すには、**if ステートメント**を使う。if ステートメントは、「もし何らかの条件に当てはまったら、以下の処理をする」という条件分岐に使用する。

if ステートメントで特定の条件に当てはまらなかったら、その下にある **elif ステートメント**の条件に、さらにそれが当てはまらなかったら、その下の **else ステートメント**の処理を実行する、というように、条件判定処理が上から下へ流れていく。

```
if　最初に判定される条件：
  実行される処理
elif　次に判定される条件：
  実行される処理
else：
  上の条件がいずれも満たされなかったとき実行される条件
  (必ずしも必要ではない)
```

条件判定では、『肥満度が 10 より大きい』の時に『太りすぎである』、と書き出し、また『肥満度が 10 より小さい』の時『痩せすぎである』、『それ以外』の時、つまり肥満度が-10 から 10 の間の時は『標準である』、と書き出すようにする。

この条件判定の『大きい・小さい』にはブール値を使う。

A ＜ B	：A は B より小さい
A ＞ B	：A は B より大きい
A ＜= B	：A は B 以下
A ＞= B	：A は B 以上
A == B	：A は B と同じ

(4) ＜擬似コード＞

変数 height **に、『あなたの身長は何** cm **ですか』に対し対話的に入力された答えを整数で代入**

変数 height **に、先に代入された変数** height **を実数に直して代入**

変数 weight **に、『あなたの体重は何** kg **ですか』に対し対話的に入力された答えを整数で代入**

変数 weight **に、先に代入された変数** weight **を実数に直して代入**

変数 standard_weight **に、標準体重を割り出す式（(変数** height / 100) ** 2) * 22 **を代入**

変数 fat **に、肥満度を割り出す式（変数** weight / standard_weight - 1) * 100 **を代入**

『標準体重は、』四捨五入した変数 standard_weight **『kg』を、書き出す**

『あなたの体重は、』変数 weight**『kg』を、書き出す**

『肥満度は、 %.1f **%です。』の % 以下を変数** fat **に置換して書き出す**

もし 変数 fat **が** 10 **より大きいならば：**
　　『太りすぎです。少し体重を減らしましょう』と書き出す

もしくは 変数 fat **が－** 10 **より小さいならば：**
　　『やせ過ぎです。健康的な食生活を心がけましょう』と書き出す

それ以外は：
　　『標準です。このままの体重を維持しましょう』と書き出す

(5) ＜プログラム＞

```
height = input("あなたの身長は、何 cm ですか？")

height =  float(height)

weight = input("あなたの体重は何 kg ですか？")

weight = float(weight)


standard_weight = ((height / 100) ** 2) * 22

fat = (変数 weight / standard_weight - 1) * 100


print "標準体重は、", round(standard_weight), "kg"

print "あなたの体重は、", weight,"kg"

print "肥満度は、%.1f %です。" % fat


if fat > 10:
   print "太りすぎです。少し体重を減らしましょう。"

elif fat < -10:
   print "やせ過ぎです。健康的な食生活を心がけましょう。"

else:
   print "標準です。このままの体重を維持しましょう。"
```

3 ループ

――――――――――――――――――――――――――――――――

(1) ＜＜問題 2 ＞＞
1 から 100 までの数字の中から、ある特定の数字を当てるゲーム。

(2) (問題把握)
当てる数字はランダムに決定されるようにし、その数字をユーザーに対話的に入力してもらう。またその数字にについて正解の数字とどれくらい離れているか大まかなヒントを書き出す。
正解するか、何も入力されない限りは繰り返しユーザーに対話的に数字の入力を求め、それが 10 より大きい数字ならば『大きすぎ』。10 より小さいなら『小さすぎ』。5 より大きいなら『やや大きい』、5 より小さいなら『やや小さい』と表示する。5 以上 5 以下の場合は『惜しい』と表示されるようにする。
質問に対して何も入力されなかった場合に終了する。

処理の流れは、まず正解となるランダム数字を変数に代入する。
次にそのランダム数字を当てるため、繰り返しユーザーに入力を求め、その結果が正解のランダム数字と同じか、もしくは何も入力されなかったか、さらにはどれくらい離れているか、を判定してヒントを返す。
終了する際はその旨を表示する。

(3) (解説)
整数をランダムに生成するには、**random** モジュールを使う。これは**「random.randint(整数 a から, 整数 b まで)」**として使う。返される整数は変数 number に代入する。
正解するか、何も入力されない限り繰り返すには、**while ステートメント**を使う。While ステートメントは、「ある一定の条件が満たされている間、ずっと同じ処理を繰り返し続ける」という場合のループ処理に使われる。この「ある一定の条件」が満たされなかった時は、else ステートメントの処理を実行する。大抵このブロックでは、終了の処理を行う。

```
while 条件が満たされている間：
    行われる処理
else:
    条件が満たされなかった場合の処理
    (必ずしも必要ではない)
```

while ループは無限ループとも呼ばれるように、条件が満たされてさえいれば永遠に同じ処理を繰り返してしまうので注意が必要である。もし無限ループを発生させてしまった場合は、コントロールキーを押しながら **C** をタイプすれば処理を中断することができる。
さらにループには for ループというものがあり、これには **for ステー**

トメントを使う。この for ループは while ループと違い、繰り返す回数が「数えられるデータの集合」として決まっている場合に使われ、1 つずつ「数えられるデータの集合」から取り出され、変数に代入される。この変数に代入されたデータに関する処理が終了すると次のデータが変数に代入され、数えられるデータの集合がなくなるまで繰り返される。

```
for　 1 つずつ代入していく変数　 in　 数えられるデータの集合：
　　代入された 1 つのデータに関する処理
```

while の条件には、『真であるあいだ』にしておく。ここで条件を終了のための『正解か、何も入力されない場合』にしてしまうと、以下のコードが冗長となってしまうためこれは避ける。ループ内では、最初に対話的にユーザーから入力を求めるが、これは変数名 usr に、入力された答えを代入する。

次に条件分岐に入っていくが、まずは終了するための条件を作成する。これは、終了条件は『何も入力されなかった場合』であるが、「何も入力されない」と指定するには「“”」と書き込むことになる。これは『空の文字列』と呼ばれるように文字列の型である。そのため、比較する変数 usr も同じ型である文字列に直さなければならない。よって、最初にユーザーから対話的に受け取る際、文字列で受け取っておき、後で数値に直すことになる。比較には if ステートメントを使用する。

文字列との比較はこれのみであり、他は整数との比較になる。よって、最初の比較はこの文字列同士の比較にしておく。

この終了条件の処理は、正解を文字列フォーマットを用いて書き出した後、ループから抜け出すようにする。ループから無条件で抜け出すには **break ステートメント**を使う。このステートメントのほかには、ループの先頭に戻る **continue ステートメント**や、何の働きもしない **pass ステートメント**がある。これらはインデントが同じとき、そのインデント内の処理の 1 つとすることができるので、抜け出すループは必ずしも直前に書かれたものではない。

以降は整数との比較になるので、ユーザーから受け取った文字列変数 usr を整数に直し、再び同じ変数 usr に代入する。

まず正解だった場合の条件を作成する。ユーザーから受け取った整数の変数 usr が、ランダム整数の変数 number と同じだった時、文字列フォーマットで正解数であるランダム整数の変数 number と、正解であるという旨を書き出し、終了のためループから無条件で抜け出る break ステートメントを入れる。

次にユーザーから受け取った整数の変数 usr が、ランダム整数の変数 number より 10 以上大きいものと比較させていく。その条件判定で偽だった場合は、5 以上大きいものと比較させ、同じようにして次は小さいものを比較していく。それらがすべて偽だった場合に、変数 usr がランダム整数変数 number より 5 以上で、かつ 5 以下の間の条件か、もしくは変数 usr が変数 number より大きく、かつ小さい条件を指定することで、正解に 1 番近い真に当て嵌める。

最後に、while ループから抜け出て終了する時、『終了します』と書き出す。

(4) ＜擬似コード＞

乱数生成のモジュール random をインポートする

変数 number に、random.randint(1, 100) で返されるランダム数値を代入

真であるあいだ：
　　変数 usr に、『１から１００までの数字を入力してください。』に対し対話的に入力された答えを文字列で代入

　　もし 変数 usr が「何も入力されていない」と同じならば：
　　　　『正解は\%d でした。』の\% 以下を変数 number に置換して書き出す
　　　　無条件でループから抜け出る

　　変数 number に、先に代入された変数 number を整数に直して代入

　　もし 変数 usr が変数 number と同じならば：
　　　　『**** %d 正解です！****』の % 以下を変数 number に置換して書き出す
　　　　無条件でループから抜け出る

　　もしくは 変数 usr が変数 number に 10 足したものより大きいならば：
　　　　『大きすぎです』と書き出す

　　もしくは 変数 usr が変数 number に 5 足したものより大きいならば：
　　　　『やや大きいです』と書き出す

　　もしくは 変数 usr が変数 number より 10 引いたものより小さいならば：
　　　　『小さすぎです』と書き出す

　　もしくは 変数 usr が変数 number より 5 引いたものより小さいならば：
　　　　『やや小さいです』と書き出す

　　もしくは 変数 usr が、変数 number に 5 足したもの以下で、変数 number から 5 引いたもの以上ならば：
　　　　『惜しい』と書き出す

『終了します。』と書き出す

(5) **＜プログラム＞**

```
import random

number = random.randint(1, 100)

while 1:
    usr = raw_input("１から１００までの数字を入力してくださ
い。")

    if usr == "":
        print "正解は %d でした。" % number
        break

    usr = int(usr)

    if usr == number:
        print "**** %d 正解です！****" % number
        break

    elif usr > (number + 10):
        print "大きすぎです"

    elif usr > (number + 5):
        print "やや大きいです"

    elif usr < (number - 10):
        print "小さすぎです"

    elif usr < (number - 5):
        print "やや小さいです"

    elif (number + 5) >= usr >= (number - 5):
        print "惜しい"

print "終了します。"
```

4 ファイルの操作

――――――――――――――――――――――――――――――――

(1) ＜＜問題 3 ＞＞

メモ帳に書かれたデータを新規ファイルに HTML で書き込み、HTML ファイルとして開くプログラム。

(2) (問題把握)

読み込むメモ帳のファイルと書き込む新規ファイルは、コマンドプロンプトからそれぞれ第 1 引数、第 2 引数として受け取る。

実行時の処理の流れは、まず変数に代入された HTML のヘッダ部を新規ファイルに書き込み、次にメモ帳の内容にタグをつけて書き込む。

最後にフッタ部を書き込み、それを開く、となる。

そこで使用するタグは、１つの文章を表す 〜 と、改行を表す
の 2 つにする。実行は以下の通りにする。

＞＞＞ python 実行するプログラム メモ帳名 新規 HTML ファイル名

(3) (解説)

コマンドプロンプト上から引数を指定するには、**sys モジュール**をインポートしてから使う。引数の指定は実行ファイル名の後に入力されたものとなり、右から第 1 引数、第 2 引数、となる。

＞＞＞ python 実行ファイル名 第 1 引数 第 2 引数

引数は、**sys.argv[引数番号]** 関数で取得する。この引数で得たデータを使うには、一度文字列や数値に変換し使用する同じ型に揃える必要がある。そのためまず引数を指定するための、sys モジュールをインポートする。さらに HTML ファイルを開くために、**os モジュール**もインポートする。これはシステムレベルでの操作が出来るプログラムで、**「os.system(”実行文”)」**と指定することで、コマンドプロンプトで実行できる作業がプログラム中でもできるようになる関数である。

次に、新規ファイルに書き込むヘッダーとフッターをそれぞれ変数に代入する。ヘッダーを入れる変数を header、フッターを入れる変数を footer とする。

メモ帳や新規ファイルを扱うためのファイルの操作には、ファイル関数の **open(”ファイル名”, ”読み込み・書き込み”)** を使う。ファイル名の後の”読み込み”・”書き込み”を指定することで、開く際の処理を決める。読み込みには**”r”**、書き込みには**”w”**を指定する。

ここでは、第 2 引数を書き込みで開いて、変数 w_file に代入しておく。引数はファイル名なので、ここで特別な型に変更していく必要はない。

これにまずヘッダーを書き込む。書き込みで開いたファイルにデータを書き込むには、**「ファイル名.write(”文字列”)」**を用いる。また、読み込みにはファイル全体を読み込み、まとめて表示する**「ファイル名.read()」**などがあるが、ここの本文にあたる部分ではファイルを 1 行ずつ読み込んで、それに対して処理を行うようにする。これは 1 行ずつ繰り返し読み込んで、変数に代入していく処理であり、for ループを使用する。またこの for ステートメント中の「数えられるデータの集合」には、この場合読み込みで開いたファイルを指定する。開いたファイルの中身を 1 行ずつ代入していく変数名は line とする。

ファイルが 1 行読み込まれた時行われる処理は、タグを付け加え、書き込みで開いた新規ファイル変数 w_file に書き込む作業となる。このとき、読み込んだ 1 行に何かしら書き込まれていた場合は「<p> 」タグを、改行のみの場合は「
」タグを書き込む。

その後、フッターを書き込み、その書き込んだ新規ファイル変数 w_file を閉じる。

開いたままだと、そのままプログラムが終了する分には問題ないが、まだファイルを使うというときエラーになってしまう。よって**「ファイル名.close()」**で閉じる。

書き込んだ HTML ファイルをブラウザで開くには、**「os.system(操作)」**の引数に、コマンドプロンプトで操作するのと同じようにしてそのファイルの完全パスか、同じフォルダ内にある場合にはファイル名を指定する。

このプログラムでは、ブラウザで開いている間はコマンドプロンプトが実行中となってしまうが、ブラウザを閉じると使えるようになる。

(4) ＜擬似コード＞

sys モジュールと os モジュールをインポートする

変数 header に、『<!DOCTYPE HTML PUBLIC
"-///W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

```
<html>
<head>
<meta http-equiv="Content-Type"
                content="text/html; charset=shift-jis">
<meta http-equiv="Content-Style-Type" content="text/css">
<title>HTML</title>
<style type="text/css">
<!--

body { color: Silver; background-color: Teal}
-->
</style>
</head>
<body>
```

』を、複数行で代入

変数 footer に『</body>\n</html>\n』を代入

変数 w_file に、書き込みで開いた第 2 引数を代入

ファイル変数 w_file に変数 header を書き込む

変数 line に、第１引数を 1 行ずつ繰り返し読み込みで開いて代入：
もし 変数 line が「\n」と同じならば：
　ファイル変数 w_file に「
」を書き込む
それ以外は：
変数 LINE に「 」 + 変数 line +「 」を代入
ファイル変数 w_file に変数 LINE を書き込む

ファイル変数 w_file に変数 footer を書き込む
ファイル変数 w_file を閉じる

os.system 関数の引数に第２引数を指定する

(5) **＜プログラム＞**

```
import sys, os

header = '''<!DOCTYPE HTML PUBLIC
            "-///W3C//DTD HTML 4.01 Transitional//EN"
             "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type"
                content="text/html; charset=shift-jis">
<meta http-equiv="Content-Style-Type" content="text/css">
<title>HTML</title>
<style type="text/css">
<!--

body { color: Silver; background-color: Teal}
-->
</style>
</head>
<body>'''


footer = '''
</body>
</html>'''

w_file = open(sys.argv[2], "w")

w_file.write(header)

for line in open(sys.argv[1], "r"):
    if line == "\n":
        w_file.write("<br>")

    else:
        LINE = "<p>" + line + "</p>"
        w_file.write(LINE)

w_file.write(footer)
w_file.close()

os.system(sys.argv[2])
```

5 ファイルの操作の応用

――――――――――――――――――――――――――――――――

(1) ＜＜問題 4 ＞＞

コマンドラインで指定された複数ファイルの内容を、同じく指定された行数分表示するプログラム。

また行数をユーザーが指定する場合はオプションを指定した後に行数を受け取り、行数をユーザーが指定しない場合は、デフォルト値として 10 行を書き出すようにする。

オプションでヘルプを指定すると、このプログラムの説明が書き出されるようにする。

(2) (問題把握)

実行パターンは、オプションが指定され行数が指定された場合、オプションが指定されずデフォルト値を適用する場合、オプションでヘルプが指定された場合の 3 パターンとなる。以下はプロンプト上での実行方法である。

オプションは、行数を指定する場合は「-n」を、ヘルプの場合は「–help」とする。

```
＞＞＞ python 実行ファイル名 -n 20 ファイル名 ファイル名 …
＞＞＞ python 実行ファイル名 ファイル名 ファイル名 …
＞＞＞ python 実行ファイル名 –help
```

処理の流れは、まずオプションについて条件判断をする。

オプションが「–help」の時は説明を書き出すだけなので、「–help」が指定されなかった時に、オプションが「-n」であるか判断する。「-n」でなかったときはデフォルトの値の処理になる。

次にファイルの操作だが、書き出す形式は

『ファイル名』

『001 行目：1 行の内容』

『002 行目：1 行の内容』

として書き出していく。

よって、まず複数のファイル名が入った変数から、1 つずつファイル名を繰り返し取り出し、その取り出したファイルから 1 行ずつ読み込んで、処理をしていくことになる。

(3) (解説)

コマンドラインから引数を受け取るので、sys モジュールをインポートする。

まず if ステートメントを使い、第 1 引数がオプション「–help」と入力されない時は、オプション「-n」やデフォルトのファイルの操作をする。

次に第 1 引数に「 - n」が入力されたときの条件分岐を作る。ここで

は、材料となる行数が第 2 引数に、読み込む複数ファイルが第 3 引数以降となるので、それぞれ変数に代入する。行数は、整数に直してから変数 n に、複数ファイルを入れる変数は files とするが、引数の 3 番目以降全てを指定するには、[**3：**] と入力する。逆に 3 までの場合は [：3] となる。

「- n」が指定される以外は、デフォルト値となり行数は 10 行となる。そこで行数の変数 n に 10 を代入する。また、デフォルトの場合は第 1 引数からファイル名が書き込まれるので、第 1 引数以降を、複数ファイルの変数 files に代入しておく。

材料が揃ったら、まず変数 file に、複数ファイルの変数 fails から 1 つずつ繰り返し代入する。これで変数 fail には、複数ファイルの中から 1 つのファイル名だけが順番に入っていくことになる。このループ内で、まずファイル名である変数 fail を書き出す。これで以下に書き出される内容がどのファイルのものなのか判るようになる。

次に行数を数えるための変数 i に 0 を代入し初期化しておく。このインデントに置くことで、1 つのファイルについての行数を加算し終わり、新しいファイルが読み込まれた時にまた初期化されるようになる。

変数 line に、ファイル変数 fail を 1 行ずつ繰り返し読み込みで開いて代入する。行数を数えるため 1 行が代入されるたびに、1 つずつ加算されるようにするには、まず変数 i に 1 を足して、それをまた変数 i に代入し直す。つまり『変数 i = 変数 i + 1』となるが、これを短く表記すると『**変数 i += 1**』となる。

行数と 1 行の内容を書き出すが、文字列フォーマットで『全桁 3 つで空桁は 0 で埋めた整数：　文字列』として書き出す。書き出される際の改行を無くすには、括弧で括った変数の後ろに、カンマを入れればよい。

そして、行数を数える変数 i と、指定された行数の変数 n が同じになった時、無条件でループから抜けるようにする。

最後に、第 1 引数に「–help」が入力され時の、ヘルプを書き込む。

(4) ＜擬似コード＞

sys モジュールをインポートする

もし 第 1 引数が、『--help』ではないならば：
　　もし 第 1 引数が『- n』と同じならば：
　変数 n に、整数に直した第 2 引数を代入
　変数 files に、第 3 引数以降を代入

　　それ以外は：
　変数 n に、10 を代入
　変数 files に、第 1 引数以降を代入

　　変数 file に、ファイル変数 files を 1 つずつ繰り返し代入：
　『　<<%s>>』の % 以下を、ファイル変数 file に置換して、文字列で書き出す
　変数 i に 0 を代入し初期化
　　変数 line に、ファイル変数 file を 1 行ずつ繰り返し読み込みで開いて代入：
　　行を数える変数 i に 1 を足して代入
　　『%03d：　%s』の % 以下を、変数 i と変数 line に順番に置換して全桁が 3 で空桁を 0 で埋めた整数と文字列で書き出す
　　もし 行を数える変数 i と変数 n が同じならば：
　　無条件でループから抜け出す

それ以外は：
　　『　このプログラムは、ファイルの中身を書き出すプログラムです。
第 1 引数に -n オプション、第 2 引数に出力する行数、第 3 引数以降にファイル名を
指定してください。
オプションと行数を指定しない場合は、10 行目までを書き出します。
オプションで --help を指定した場合に、このヘルプが表示されます。』を複数行で書き出す

(5) **＜プログラム＞**

```
import sys

if sys.argv[1] != "--help":

    if sys.argv[1] == "-n":
        n = int(sys.argv[2])
        files = sys.argv[3:]

    else:
        n = 10
        files = sys.argv[1:]

    for file in files:
        print "    <<%s>>" % file
        i = 0
        for line in open(file, "r"):
            i += 1
            print "%03d: %s" % (i, line),
            if i == n:
                break

else:
    print''' このプログラムは、ファイルの中身を書き出すプログ
ラムです。
第 1 引数に -n オプション、第 2 引数に出力する行数、第 3 引数以降
にファイル名を
指定してください。
オプションと行数を指定しない場合は、10 行目までを書き出します。
オプションで --help を指定した場合に、このヘルプが表示されます。
'''
```

6 ファイルの操作の応用 2

――――――――――――――――――――――――――――――――

(1) ＜＜問題 5 ＞＞

指定したファイルから、指定した行のみ書き出すプログラム。またリダイレクトで書き込みファイルを指定すると、その書き出される行がファイルに書き込まれる。

行の指定には、1 行のみ、ハイフンで区切った行から行まで、カンマで区切った各行ずつ、の 3 パターンを用意する。

(2) (問題把握)

書き出すファイルと行の指定はコマンドプロンプトの引数から受け取る。

ハイフンで区切った行の間全てを表示するには、ハイフンの両側の数字を取り出し、その数字と数字の間でのみ書き出すようにする。

カンマで区切った各行を表示させる場合も、同じように数字を取り出し、その行になったときに書き出すようにする。

```
＞＞＞ python 実行ファイル名 対象ファイル名 ”行”
＞＞＞ python 実行ファイル名 対象ファイル名 ”行 - 行”
＞＞＞ python 実行ファイル名 対象ファイル名 ”行、行、行”

＞＞＞ python 実行ファイル名 対象ファイル名 ”行 - 行” ¿ 書き込むファイル
```

処理の流れは、ファイル名と行数を変数に代入し、またその行数と比較するため、ファイルの行を数える変数を初期化しておく。

つぎにファイルを 1 行ずつ読み込み、その行数を数えておく。

あとは 3 パターンの書き出す処理を比較して書き出す。

(3) (解説)

引数からファイルを指定するので、sys モジュールをインポートする。

対象ファイル名が入る第 1 引数を、変数 file_name に代入しておき、また行数の指定が入る第 2 引数を変数 Lines に代入しておく。さらにファイルの行数を数えるため、変数 n に 0 を代入しておき、整数の変数を初期化する。

まず、比較のためにファイル変数 file_name を読み込みで 1 行ずつ繰り返し変数 line に代入し、その 1 行ごとに変数 n に 1 を足し代入する。これで変数 n は、読み込んだファイルの行数を表すことになる。

その 1 行ごとのループ内で、3 パターンに分岐させる。まず 1 行のみの指定の場合、対象ファイル行の変数 n と指定行数の変数 Lines で、指定した行が同じ場合を条件にしそれを書き出すようにする。

ファイル行の変数 n と指定行数の変数 Lines を比較するとき、同じオブジェクトの型に当て嵌めなければならない。つまり整数同士、文字列

同士での比較にしなければならない。ここでは変数 n を文字列に変換しておく。

２つ目の、行と行の間が指定された場合は、間のハイフンがあるかを条件判定にかける。

ハイフンの両側の数字を取り出すには、「文字列.split(目印)」を使う。これは以前使ったものと同じように、文字列を目印で分解した後、アンパック代入を用いて左の数字を変数 froms に、右の数字を変数 to に代入する。そして、ファイル行の変数 n が、左の小さい数の変数 from 以上で、かつ右の多きい数の変数 to 以下の場合に書き出す。

３つ目の、各行を指定した場合、間のカンマがあるかを条件判定にかける。これも同じようにカンマを目印として区切って各数字を取り出すが、カンマの間に空白を入れる場合があるので、まず空白を無くす必要がある。これには **replace() メソッド**を使う。これは**「文字列.replace(対象文字列，置換文字列)」**という形式で使うが、この場合は対象文字列は空白で、それを「何もない」を表す「””」に置換すればよい。

その次にカンマを目印として分解する。この置換の処理と分割の処理を 2 つに分けても問題ないが、短く１つにまとめて記述するようにする。処理は通常上から下へ、また左から右へ順に流れる。よって、これらはまず空白を消す置換処理を行ってから、それを対象文字列として目印で分割したリストを変数 Lines に代入し直す。しかしこのままでは、リストで代入されているため１つずつ行数と比較することができない。そのため、１つ１つのリストの要素を繰り返し取り出し、再び同じ変数に代入し直し、そうやって取り出された各指定行数の変数とファイル行の変数 n とを比較させる。

比較するには、変数の型を揃えるため、変数 n を文字列に直しておき、ファイル行の変数 n と指定行数の変数が同じとき、書き出すようにする。

(4) ＜擬似コード＞

sys をインポートする

変数 file_name に第１引数を代入
変数 Lines に第２引数を代入
変数 n に 0 を代入し初期化し

変数 line に、ファイル変数 file_name を１行ずつ繰り返し読み込みで開いて代入：
行を数える変数 n に１を足し代入
もし 文字列に直した変数 n が、変数 Lines と同じならば：
　　『%d：%s』の % 以下を、行を数える変数 n と変数 line に順番に置換して整数と文字列で書き出す

もしくは "－"がファイル変数 write_line に含まれていたならば：
　　変数 froms と変数 to に、変数 Lines を『－』を目印に分割して、それぞれに代入
　　もし 整数に直した変数 froms が変数 n 以下であり、変数 n が整数に直した変数 to 以上であるならば：
　『%d：%s』の % 以下を行を数える変数 n と変数 line に順番に置換して整数と文字列で書き出す

もしくは "，" が変数 write_line に含まれていたならば：
　　変数 Lines に、変数 Lines の空白を消して『,』を目印に分割したリストを代入
　変数 Lines に、変数 Lines の要素を１つずつ繰り返し代入：
　もし 変数 n と数値に変換した変数 Lines が同じならば：
『%d：%s』の % 以下を、行を数える変数 n と変数 line に順番に置換して整数と文字列で書き出す

(5) **＜プログラム＞**

```
import sys

file_name = sys.argv[1]
Lines = sys.argv[2]
n = 0

for line in open(file_name, "r"):
    n += 1
    if str(n) == Lines:
        print "%d:%s" % (n, line),

    elif "-" in Lines:
        froms, to = Lines.split("-")
        if int(froms) <= n <= int(to):
            print "%d:%s" % (n, line),

    elif "," in Lines:
        Lines = Lines.replace(" ", "").split(",")
        for Lines in Lines:
            if str(n) == Lines:
                print "%d:%s" % (n, line),
```

7 関数作成

――――――――――――――――――――――――――――――――

(1) ＜＜問題 6 ＞＞

引数から受け取った 2 つの数値から、足し算・引き算・割り算・掛け算を演算し、その答えが一度に表示されるプログラム。

(2) (問題把握)

コマンドプロンプト上から 2 つの数字を受け取って、それを 4 種類の演算式に当て嵌めていく。

演算には、簡単な演算のみを行う関数を作成し、それを実行させて出た答えを、見やすいよう一覧表示する。

書き出す時は文字列フォーマットを使用する。コマンドプロンプトでの指定方法は以下の通りである。

```
＞＞＞ python 実行ファイル名 数字 数字
```

処理の流れは、まず演算の材料となる数字を引数から受け取り、変数に代入する。

次に各演算を実行する関数を作成し、それを見やすいよう書き出していくことになる。

(3) (解説)

コマンドラインから引数として数字を受け取りたいので、まず sys モジュールをインポートする。受け取った数字は演算に使うので数値に変換するが、割り算があるので整数でなく実数に直し、書き出す時に割り算以外は文字列フォーマットで整数に直すことにする。

第 1 引数を実数に直したものは変数 x に、同じく実数に直した第 2 引数は変数 y に代入しておく。

関数を作成するには、**def** 関数を使う。これは 1 度作成しておけば何度でもプログラム内で実行でき、また引数で使用する変数は数値のみ、文字列のみと固定されていないので、どちらでも使える。

```
def 関数名 (引数名のリスト)：
    関数内の処理
     return 実行時に返す処理 (必ずしも必要ではない)
```

この関数は、関数名と引数名を指定し実行してはじめて関数内の処理が行われる。よって関数の中で使用した変数などは、何らかの処理をしない限りは関数外に影響しない。引数名のリストは、関数内の処理で実行する同じ引数名に代入される。この引数は無い場合もある。そこで得られた結果は、**return ステートメント**で本来のコードに渡される。実行の際は**「関数名 (引数名のリスト)」**と指定する。

まず関数名を tasizan にした、足し算の演算をする関数を作成する。

引数名のリストには、プロンプトから受け取る数値 2 つで演算していくため、引数名は (a, b) としておく。

関数内の処理は、単純に a + b となる。これを一度別の変数に代入してから、その変数を return で結果として返してもよいが、簡単に return a + b とする。return は『結果を返す』ステートメントであるので、a + b の演算が実行されて返されることになる。

以下の関数作成も同じようにしていくが、前にも述べたように関数内の変数は基本的に他に影響しないので、引数名のリストは同じものを使用する。

次に実際に関数を実行させ、結果を書き出す。その際 1 つ 1 つ書き出してもよいが、今回は複数行に渡って、書き込んだ通りに書き出されるトリプルクォーテーションを使う。

これは、シングルクォーテーション (') やダブルクォーテーション (") を 3 つ重ねたもの **(”’)** で、この 2 種類のどちらを使ってもよい。書き出すには、『何の演算か：結果』の形式にし、結果については文字列フォーマットで書き込んでいく。複数の文字列フォーマットでは、置換したい文字列に書かれたフォーマットとその以下に () で閉じられたデータは、その左から書かれた順番ごとに置換されていく。この場合は実行する関数が順番ごとに置かれていく。

関数の実行は**関数名 (引数名)** で行う。このとき、引数には関数作成で指定した引数が当て嵌められる。ここでは、引数にはプロンプトから受け取った数値の入った第 1 引数と第 2 引数の変数 x、y が入る。

(4) ＜擬似コード＞

sys **モジュールをインポートする**

変数 x **に、コマンドラインの第** 1 **引数を実数に直して代入**
変数 y **に、コマンドラインの第** 2 **引数を実数に直して代入**

関数名 tashizan **で引数名** (a, b) **の関数作成：**
　　a + b **の結果を返す**

関数名 hikizan **で引数名** (a, b) **の関数作成：**
　　a ー b **の結果を返す**

関数名 kakezan **で引数名** (a, b) **の関数作成：**
　　a * b **の結果を返す**

関数名 warizan **で引数名** (a, b) **の関数作成：**
　　a / b **の結果を返す**

『足し算結果：%d
引き算結果：%d
掛け算結果：%d
割り算結果：%.1f』 **の % 以下を、関数** tashizan(**変数** x, **変数** y)**、関数** hikizan(**変数** x, **変数** y)**、関数** kakezan(**変数** x, **変数** y) **をそれぞれ整数に直したもの、関数** warizan(**変数** x, **変数** y) **を小数点１以下を四捨五入たものを、順番に置換して複数行で書き出す**

(5) **＜プログラム＞**

```
import sys

x = float(sys.argv[1])
y = float(sys.argv[2])

def tashizan(a, b):
    return a + b

def hikizan(a, b):
    return a - b

def kakezan(a, b):
    return a * b

def warizan(a, b):
    return a / b

print '''
足し算結果：%d
引き算結果：%d
掛け算結果：%d
割り算結果：%.1f''' % (tashizan(x, y), hikizan(x, y),
                                    kakezan(x, y), warizan(x, y))
```

8 関数作成の応用

――――――――――――――――――――――――――――――――

(1) ＜＜問題 7 ＞＞

円・三角形・正方形・長方形・台形の 5 パターンの図形の面積の演算が、コマンドラインの引数で決まったオプションを指定することで、表示される。また、このプログラムのヘルプとなる説明を表示するプログラム。

(2) (問題把握)

実行できる演算は、1 つのみとする。

またコマンドラインで指定する引数は、第 1 引数に図形を選択するオプション、第 2・3 引数に演算に用いる数値を入力するようにする。

オプションは、円が『-e』三角形が『-s』正方形が『-h』長方形が『-t』台形が『-d』とし、指定された図形の面積を表示する。

ヘルプの場合は、オプションに『–help』とだけ入力すると表示されるようにする。

面積の演算は関数作成で行う。実行は以下の通りにする。

```
>>> python 実行ファイル名 -t 4 6
```

処理の流れは、まず図形を指定するオプションを引数から受け取り、変数に代入しておく。次に各演算を実行する関数を作成しておく。

全図形のオプションについて、その特定のオプションが指定された時に関数を実行させ、そこから得た結果を表示させる、となる。

(3) (解説)

まずコマンドラインで引数を取得するので、sys モジュールをインポートしておく。次にオプションとなる第 1 引数を変数 shurui に代入する。これは、後で実行する関数を選択する際の条件との比較対象になる。

次に面積を求める演算を関数で作成する。

・ 円の面積を求める公式は「半径×半径× 3.14」
・ 三角形の公式は「底辺×高さ÷ 2」
・ 正方形の公式は「１辺×１辺」
・ 長方形の公式は「底辺×高さ÷ 2」
・ 台形の公式は「(上底＋下底) ×高さ÷ 2」である。

これらを参考にし、引数名を公式に当て嵌めた演算結果を返すようにする。

関数ができたら、オプションで実行する処理を選択させるため、条件分岐を作る。これは if ステートメントで、変数 syurui と各オプションが同じだった場合に、関数を実行させる条件分岐を作る。円の場合は、変数 syurui が「-e」と同じだった時、コマンドラインから受け取った

半径を整数に直し、変数 hankei に代入する。後は円の面積である旨と、引数名を変数 hankei に指定した関数を書き出せばよい。同じようにして他の面積も書き出していく。

　ヘルプは、このプログラムについての説明が書き出されるだけでよい。

(4) ＜擬似コード＞

sys モジュールをインポートする

変数 syurui にコマンドラインの第 1 引数を代入

関数名 en_menseki で引数名（hankei）の関数作成：
　　3.14 * hankie * hankei の結果を返す

関数名 sankaku_menseki で引数名（teihen, takasa）の関数作成：
　　teihen * takasa / 2 の結果を返す

関数名 seihou_menseki で引数名（ippen）の関数作成：
　　ippen * ippen の結果を返す

関数名 tyouhou_menseki で引数名（takasa, teihen）の関数作成：
　　takasa * teihen の結果を返す

関数名 daikei_menseki で引数名（jyouhen, teihen, takasa）の関数作成：
　　（jyouhen + teihen） * takasa / 2 の結果を返す

もし 変数 syurui と『-e』が同じならば：
　　変数 hankei に、コマンドラインの第 2 引数を整数に直して代入
　　『円の面積：』関数 en_menseki(変数 hankei)『cm2』を、書き出す

もしくは 変数 syurui と『-s』が同じならば：
　　変数 teihen に、コマンドラインの第 2 引数を整数に直して代入
　　変数 takasa に、コマンドラインの第 3 引数を整数に直して代入
　　『三角形の面積：』関数 sankaku_menseki(変数 teihen, 変数 takasa)『cm2』を、書き出す

もしくは 変数 syurui と『-h』が同じならば：
　　変数 ippen に、コマンドラインの第 2 引数を整数に直して代入
　　『正方形の面積：』関数 seihou_menseki(変数 ippen)『cm2』を、書き出す

もしくは 変数 syurui と『-t』が同じならば：
　　変数 teihen に、コマンドラインの第 2 引数を整数に直して代入
　　変数 takasa に、コマンドラインの第 3 引数を整数に直して代入
　　『長方形の面積：』関数 tyouhou_menseki(変数 teihen, 変数 takasa)『cm2』を、書き出す

もしくは 変数 syurui と『-d』が同じならば：
　　変数 jyouhen に、コマンドラインの第 2 引数を整数に直して代入
　　変数 teihen に、コマンドラインの第 2 引数を整数に直して代入

変数 takasa に、コマンドラインの第 2 引数を整数に直して代入
『台形の面積：』関数 daikei_menseki(変数 jyouhen, 変数 teihen, 変数 takasa)『cm2』を、書き出す

もしくは 変数 syurui と『--help』が同じならば：
『このプログラムは、円・三角形・正方形・長方形・台形の面積を求めます。
円の面積を求めたいなら "-e 半径"
三角形の面積を求めたいなら "-s 底辺 高さ"
正方形の面積を求めたいなら "-h 一辺"
長方形の面積を求めたいなら "-t 底辺 高さ"
台形の面積を求めたいなら "-d 上底 下底 高さ"

以上の形式を用いて、第１引数から各オプションを入力してください。』
を、複数行で書き出す

(5) **＜プログラム＞**

```
import sys

shurui = sys.argv[1]

def en_menseki(hankei):
    return 3.14 * hankei * hankei

def sankaku_menseki(teihen, takasa):
    return teihen * takasa / 2

def seihou_menseki(ippen):
    return ippen * ippen

def tyouhou_menseki(takasa, teihen):
    return takasa * teihen

def daikei_menseki(jyouhen, teihen, takasa):
    return (jyouhen + teihen) * takasa / 2

if shurui == "-e":
    hankei = int(sys.argv[2])
    print "円の面積:", en_menseki(hankei),"cm2"

elif shurui == "-s":
    teihen = int(sys.argv[2])
    takasa = int(sys.argv[3])
    print "三角形の面積", sankaku_menseki(teihen, takasa), "cm2"

elif shurui == "-h":
    ippen =int(sys.argv[2])
    print "正方形の面積", seihou_menseki(ippen), "cm2"

elif shurui == "-t":
    teihen = int(sys.argv[2])
    takasa = int(sys.argv[3])
    print "長方形の面積", tyouhou_menseki(takasa, teihen), "cm2"

elif shurui == "-d":
    jyouhen = int(sys.argv[2])
    teihen = int(sys.argv[3])
    takasa = int(sys.argv[4])
    print "台形の面積", daikei_menseki(jyouhen, teihen, takasa), "cm2"

elif shurui == "--help":
    print ''' このプログラムは、円・三角形・正方形・長方形・台形
の面積を求めます。
```

円の面積を求めたいなら "-e 半径"
三角形の面積を求めたいなら "-s 底辺 高さ"
正方形の面積を求めたいなら "-h 一辺"
長方形の面積を求めたいなら "-t 底辺 高さ"
台形の面積を求めたいなら "-d 上底 下底 高さ"

以上の形式を用いて、第１引数から各オプションを入力してください。
’’’

9 ディクショナリ

――――――――――――――――――――――――――――――――

(1) ＜＜問題 8 ＞＞

ID とパスワードの入力を求め、もし ID が登録されていなかったら新規に登録する。

ID とパスワードが一致するまで繰り返し、何も入力されなかった場合か、一致した場合に終了するプログラム。

(2) (問題把握)

ID とパスワードは対話的に入力してもらう。それらを保存しておくデータベースとなるファイルを、ファイル名「id_password_data.txt」として新規に作っておく。

データベースに保存する際、ディクショナリのキーを ID、要素をパスワードに指定して「id_password_data.txt」に書き込むようにする。

ユーザーに対する質問は、最初に ID の入力を求め、ID が見つかったらパスワードの入力に移り、それが見つかったら確認できた旨を表示し終了する。パスワードが見つからなかった場合は、その旨表示し再び ID の入力に戻る。

最初の ID が見つからなかったら、登録されていない旨と、新たに登録するかの確認を取る。登録するのであれば、パスワードの入力を求め、先に入力した ID とパスワードを表示し、これらを本当に登録するか確認する。登録が終わった場合と登録を中止する場合は、再び ID 入力に戻る。

処理の流れは、ディクショナリの変数とデータベースファイルの変数、何度も同じ質問を使うので質問のみを入れてた変数を用意する。

次に、データベースに書き込まれた ID とパスワードをそれぞれ取り出し、ディクショナリ変数に代入する関数を作成する。また、同じようにしてデータベースにディクショナリ変数に溜まったデータを書き込む関数も作成する。

まずデータベースから読み込む関数を実行し、ID の入力を対話的に求め、何も入力されることがない間、上記のユーザーに対する質問の処理を繰り返すようにする。

最後に、ループが終了したところで新規ファイルに書き込む関数を実行し、終了する旨を書き出すようにする。

(3) (解説)

まず、一時的に ID とパスワードを貯めておくための空のディクショナリを作成し、変数 data に代入しておく。ディクショナリとは、キーとそれに対応する要素が複数集まって 1 つのデータとなるもので、**{' キー' : ' 要素', ' キー' : ' 要素'}** と表記する。ディクショナリの特徴として、キーから要素を抽出することができ、検索が容易であることが挙げられる。

次に、データベースとなるファイルの操作のために、ファイル名

「id_password_data.txt」を変数 data_file に保存しておく。
　ID とパスワードに関する質問も、同様に変数に代入しておく。質問は、ID の入力を求めるもの、パスワードの入力を求めるもの、新規登録をするかどうか、入力した ID とパスワードを登録するかどうかの確認、の 4 パターン用意する。新規登録と登録確認は、「登録する」の場合は 1、「登録しない」場合は 2 を選択するようにする。

　データベースファイルから ID とパスワードを読み込む関数を作成する。関数名は read_id_password_file とし、引数名は (dic) とする。ここでの処理は、データベースファイルを 1 行ずつ繰り返し開いて、そのたびに行の末尾の改行を文字列メソッド**「文字列.rstrip( )」**で消す。
　データベースには『ID\t パスワード\n』の形式で書き込ようにするので、文字列を目印で分解する「文字列.split(”目印”)」メソッドを使う。普通「\t」はタブ、「\n」は改行を表す。そのためここでは目印をタブに指定すれば、タブで区切られた文字列が返ってくる。ここでは ID とパスワードは別々の変数 a、b に代入したいので、アンパック代入を用い「(変数 a、変数 b) = 文字列.split(”\t”)」とし、タブで区切られた ID が変数 a へ、パスワードが変数 b へ代入されるようにする。
　これを、ディクショナリで ID となるキーと要素となるパスワードとして保存するが、データを追加するには、**「ディクショナリの変数[キー] = 要素」**と指定する。この場合、ディクショナリの変数は、関数の引数名 (dic) として指定しておき、実行時はここにディクショナリ変数 data を入れるようにする。

　次にデータベースファイルに、ディクショナリに保存しておいた ID とパスワードを書き込む関数を作成する。関数名は write_id_password_file で、引数名は (dic) にしておく。
　まずデータベースファイルを書き込みで開き、変数 output_file に代入する。次に引数名 (dic) を変数 i に 1 つずつ繰り返しに代入していく。これで、引数 (dic) にディクショナリ変数 data を入れることで、キーを 1 つずつ取り出し、変数 i に代入していくことになる。そのループ内で、ファイル変数 output_file に文字列『ID\t パスワード』の形式で書き込んでいくが、ここは文字列フォーマットを使う。変数 i に代入されているのはキーのみなので、そこから要素を抽出するディクショナリの**変数[キー]** も用いて置換する。
　全て書き込みが終了したら、ファイルを閉じておく。

　ここから実際にプログラムが実行されていく。
　まず関数 read_id_password_file(data) を実行し、データベースからデータを読み込んでおく。次に ID の入力を対話的に求め、変数 id に代入する。変数 id に何かしら入力されている間は、ループを続ける。これで、ID に何も入力されなかった場合に終了することになる。
　ID となる特定のキー (変数 id) が、ディクショナリ変数 data に存在するかの確認には、**「ディクショナリ変数.has_key(キー)」**を使う。この条件分岐で、もしキーが存在したら、パスワードを対話的に入力してもらい変数 pw に代入しておく。次にキー (変数 id) から要素を抽出し、それが pw と同じならば、確認が取れた旨を書き出し、無条件でループから抜け出ることで終了する。それ以外は、『パスワードが一致しませ

ん』と書き出す。

最初の条件分岐でキーが存在しなかった場合、新規登録をするか確認のため対話的に 1 か 2 を入力してもらい、変数 answer に代入する。もし変数 answer が 1 の場合、対話的にパスワードの入力を求め、変数 pw に代入する。さらにこれで登録してよいか確認し、対話的に 1 か 2 で入力してもらい、変数 answer に代入しする。そこに 1 が入力された場合、ディクショナリ変数 data に、キーを ID の変数 id で要素をパスワードの変数 pw にして追加する。これら if ステートメントの条件分岐の処理が終わったら、再び ID の入力を対話的に求め、ID の変数 id に代入する。これで新しい ID を条件判定にかけることができる。

最後にループ外で関数 write_id_password_file(data) を実行し、『ID パスワード確認を終了します。』と書き出す。

(4) ＜擬似コード＞

変数 data に、空のディクショナリを代入

変数 data_file に、「id_password_data.txt」を代入

変数 id_nyuryoku に、『ID を入力してください：』を代入

変数 pw_nyuryoku に、『パスワードを入力してください：』を代入

変数 toroku_kakunin に、『あなたの名前は登録されていません。
登録しますか。1. はい　2. いいえ：』を複数行で代入

変数 id_pw_kakunin に、『ID: %s
パスワード: %s
を登録します。1. はい　2. いいえ：』を複数行で代入

関数名 read_id_password_file で引数名 dic の関数作成：
　　変数 line に、変数 data_file を 1 行ずつ繰り返し読み込みで開いて代入：
　　　　変数 line に、変数 line の末尾の改行を削除して代入
　　変数 a と変数 b に、変数 line のタブを目印に分解して、それぞれに代入
　　ディクショナリ変数 dic にキーが「変数 a」、要素が「変数 b」で追加

関数名 write_id_password_file で引数名 dic の関数作成：
　　変数 output_file に書き込みで開いた変数 data_file を代入
　　ディクショナリ変数 dic を , 変数 i に繰り返し代入：
　　ファイル変数 output_file に『%s\t%s\n』の % 以下を、変数 i とディクショナリ変数 dic キー「変数 i」に対応する要素を、順番に置換して文字列で書き出す
　　ファイル変数 output_file を閉じる

関数 read_id_password_file(data) を実行

変数 id に、変数 id_nyuryoku に対し対話的に入力された答えを文字列で代入

変数 id が「何も入力されていない」ではないあいだ：
　　もし ディクショナリ変数 data に変数 id と合致するキーがあったならば：
　　変数 pw に、変数 pw_nyuryoku に対し対話的に入力された答えを文字列で代入
　　もし ディクショナリ変数 data のキー「変数 id」の要素と変数 pw が同じならば：

『%s さん、確認が取れました。』の % 以下を、変数 id と置換して文字列で書き出す。
無条件でループから抜ける
　　それ以外は：
『%s さん、パスワードが一致しません。』の % 以下を、変数 id と置換して文字列で書き出す。

　　それ以外は：
　　変数 answer に、変数 toroku_kakunin 似たいし対話的に入力された答えを文字列で代入
　　もし 変数 answer が『1』と同じならば：
　変数 pw に、変数 pw_nyuryoku に対し対話的に入力された答えを文字列で代入する
　変数 answer に、変数 id_pw_kakunin に対し対話的に入力された文字列の、% 以下を、変数 id と変数 pw に順番に置換したものを代入
　もし 変数 answer が『1』と同じならば：
　　ディクショナリ変数 data に、変数 id をキー、変数 pw を要素として追加
　　変数 id に、変数 id_nyuryoku に対し対話的に入力された答えを文字列で代入

関数 write_id_password_file(data) を実行
「ID パスワード確認を終了します。」を書き出す

(5) ＜プログラム＞

```
data = {}

data_file = "id_password_data.txt"

id_nyuryoku = "ID を入力してください："

pw_nyuryoku = "パスワードを入力してください:"

toroku_kakunin = ''' あなたの名前は登録されていません。
登録しますか。1. はい　2. いいえ:'''

id_pw_kakunin = '''ID: %s
パスワード: %s
を登録します。1. はい　2. いいえ:'''


def read_id_password_file(dic):
    for line in open(data_file, 'r'):
        line = line.rstrip()
        (a, b) = line.split('\t')
        dic[a] = b

def write_id_password_file(dic):
    output_file = open(data_file, 'w')
    for i in dic:
        output_file.write("%s\t%s\n" % (i, dic[i]))
    output_file.close()


read_id_password_file(data)

id = raw_input(id_nyuryoku)

while id != '':
    if data.has_key(id):
        pw = raw_input(pw_nyuryoku)
        if data[id] == pw:
            print "%s さん、確認が取れました。" % (id)
            break

        else:
            print "%s さん、パスワードが一致しません。" % (id)

    else:
        answer = raw_input(toroku_kakunin)
        if answer == "1":
```

```
            pw = raw_input(pw_nyuryoku)
            answer = raw_input(id_pw_kakunin % (id, pw))
            if answer == "1":
                data[id] = pw
    id =raw_input(id_nyuryoku)

write_id_password_file(data)
print "ID パスワード確認を終了します。"
```

10 ディクショナリに似たモジュール

―――――――――――――――――――――――――――――――

(1) ＜＜問題 9 ＞＞

語句の登録と検索ができるプログラム。

繰り返し検索か登録かを最初に選択し、検索の場合は検索語句が見つかったらその意味を表示する。登録の場合は、登録語句と意味を入力する。

選択に指定されなかった文字を指定した場合は終了する。

(2) (問題把握)

検索や登録の選択や、語句の入力はユーザーから対話的に受け取るようにする。登録語句は別のファイルにデータベースとして書き込む。データベースファイルへのアクセスには、ディクショナリと同じようにキーとして行える shelve モジュールを使用する。検索語句の照合や登録にはディクショナリを使い、繰り返し検索か登録かの選択を求めるようにする。選択で示さなかった文字を入力した場合には、終了となる。

処理の流れは、まず以下の処理をループで繰り返すようにする。その中で最初に登録か検索か質問し、対話的にユーザーから答えを受け取り、変数に代入する。

その変数で、検索が選択された場合、検索語句の入力を求め、それと合致する要素がデータベースに存在するか確認し、あったらキーと要素を書き出すが、なかったらその旨表示する。

登録が選択された場合は、登録語句の入力を求め、さらにその意味の入力も求めて、それらをデータベースに追加する。

それ以外では、終了する旨を書き出し、ループから無条件に抜け出ることで終了することになる。

(3) (解説)

shelve モジュールは、ファイルへの書き出し、書き込みがキーによって操作できる。ファイルの指定方法は、**「ファイル変数 ＝ shelve.open(' ファイル名')」**となる。この代入されたファイル変数を、ディクショナリの入った変数と同じように指定して要素の追加や要素の確認などを行う。

ループは、「真の間」はずっと繰り返すようにしておく。このループ内の初めに、対話的に『検索は" f "キー、登録は"n"キーを入力してください』と、文字列での入力を求め、「f」キーが押された場合は検索、「n」キーが押された場合は登録、それ以外が押された場合は終了する、の 3 つの条件分岐を作る。

検索の場合、文字列で検索語句の入力を対話的に求め、それを変数 key に代入しておく。

次に shelve で、データベースファイル名を指定して開き、変数 file に

代入する。指定するファイル名は「data」にする。その検索語句の変数 key をキーとして、キーの要素がファイル変数 faile に存在するか確認し、その結果を変数 youso に代入する。

変数 youso に返されるデータは、真か偽、つまり 1 か 0 なので、もし変数 youso が 1 で、存在した場合は検索語句の変数 key と、その要素を書き出す。それ以外の場合は、検索語句の変数 key『は登録されていません』、と書き出す。登録の場合も同様に、登録語句と意味を対話的に入力してもらい、それぞれ変数 toroku と変数 imi に代入しておく。

データベースファイル data を shelve で同じように開いて変数 file に代入しておく。次に変数 toroku をキーに、変数 youso を要素としてデータベース変数 file に追加する。そして変数 toroku を登録した旨を書き出し、データベース変数 file を閉じる。

終了するには、それ以外の時に『終了します』と書き出し、無条件にループを抜け出る。

(4) ＜擬似コード＞

shelve モジュールをインポートする

真であるあいだ：
　　変数 question に、『<<検索は"f"キー、登録は"n"キーを入力してください>>→』に対し対話的に入力された答えを文字列で代入

　　もし 変数 question が『ｆ』と同じならば：
　変数 key に、『<<検索語句を入力してくださいい>>→』に対し対話的に入力された答えを文字列で代入
　変数 file に、ファイル名「data」を shelve で開いて代入
　変数 youso に、shelve ファイル変数 file にキーが「変数 key」と合致するものがあるか確認し代入
　もし 変数 youso が 1 と同じならば：
　変数 key、『：[』、shelve ファイル変数 file のキーを変数 key に指定して要素を抽出、『]』を、書き出す

　それ以外は：
　変数 key、『は登録されていません』を書き出す

　　もしくは 変数 question が『n』と同じならば：
　変数 toroku に、『<<登録語句を入力してくださいい>>→』に対し対話的に入力された答えを文字列で代入
　変数 imi に、『<<意味を入力してくださいい>>→』に対し対話的に入力された答えを文字列で代入
　変数 file に、ファイル名「data」を shelve で開いて代入
　shelve ファイル変数 file に変数 toroku をキー、変数 imi を要素として追加
　変数 toroku、『を登録しました』と書き出す

　shelve ファイル変数 file を閉じる

　　それ以外は：
　『終了します。』と書き出す
　無条件でループから抜け出す

(5) ＜プログラム＞

```
import shelve
while 1:
    question = raw_input('<<検索は"f"キー、登録は"n"キーを入
力してください>>→')

    if question == 'f':
        key = raw_input('<<検索語句を入力してくださいい>>
→')
        file = shelve.open('data')
        youso = file.has_key(key)
        if youso == 1:
            print key, ": [", file[key], "]"

        else:
            print key, "は登録されていません"

    elif question == 'n':
        toroku = raw_input('<<登録語句を入力してください>>
→')
        imi = raw_input('<<意味を入力してください>>→')
        file = shelve.open('data')
        file[toroku] = imi
        print toroku, "を登録しました。"

        file.close()

    else:
        print "終了します。"
        break
```

11 正規表現

――――――――――――――――――――――――――――――

(1) ＜＜問題 10 ＞＞

対象ファイル内に書かれた英単語と、その出現回数を書き出すプログラム。

(2) (問題把握)

対象ファイルは第 1 引数で指定し、結果は『英単語：出現回数』をソートした形で書き出す。

```
＞＞＞ python 実行ファイル名 対象ファイル名
```

対象ファイル内で英単語を検索するには、正規表現を用いる。正規表現は re モジュールをインポートすることで使用できる。

出現回数を加算していくには、ディクショナリを使う。

処理の流れは、まず検索する英単語にマッチする全ての要素について、正規表現としてコンパイルし変数に代入する。また出現回数を加算するためのディクショナリと対象ファイル名の入った変数も作成しておく。

次に、出現回数を加算する関数を作成する。マッチしたオブジェクトに対し、ディクショナリ変数に要素があるかの条件を判定し、あったら加算、なかったら 1 を入れる。

対象ファイルを 1 行ずつ繰り返し読み込んでいき、その 1 行に正規表現とマッチするものがあるか検索する。マッチしたものは作成した関数に回されるようにする。

最後に、ソートしたキーと、それに対応する要素を『英単語：出現回数』として書き出す。

(3) (解説)

最初に、sys モジュールと正規表現の **re モジュール**をインポートしておく。正規表現とは、ごく単純な文字から複雑な文字まで、文字列や決められた記号を組み合わせて検索条件や置換指示を表現する文字列操作方法のことである。

文字列をそのまま正規表現として使うためには、コンピュータが処理できるよう、一度機械語に翻訳して渡す必要がある。この翻訳のことを「コンパイル」する、という。コンパイルには、正規表現モジュールの**「compile(文字列)」**を使う。文字列を指定していくには、決まった文字列と記号を組み合わせる。

ここで検索したい文字列は、大文字の英字、小文字の英字、単語の間のハイフン、シングルクォーテーションである。これを正規表現として表すには、『シングルクォーテーションと、ハイフンと、大文字の英字と小文字の英字が 1 回以上繰り返す』ということになる。

クォーテーションは**特殊文字**であるので、それを記号として認識させ

ずにただの文字列として無視させるには、その直前に「\」記号をつける。正規表現の特殊記号を用いて英字の『AからZまで』と表すには、「A-Z」と表記する。これらの『1 回以上の繰り返し』の場合は、「[ ]」で文字の集合として指定したい文字を括った後に、「+」をつければよい。つまり、これらを正規表現で表すと『[-a-zA-Z]+』となる。このとき、文字列の前後に raw string 記法となる**「r' 文字列 '」**をつけるようにする。raw string 記法で書くことで、特殊記号などの意味を無視するときに使う「\」は記号として認識されなくなる。コンパイルしたこの文字列は、変数 re_alpha に代入する。

次に空のディクショナリを作成しておく。変数名は EnglishWords とする。これは英単語をキー、出現回数を要素として貯めておくためである。

引数で対象ファイルを指定するので、引数の 1 番目を変数 file_name に代入する。

出現回数を加算するための関数を作成する。関数名は toroku とし、引数名は (m) にする。この引数は、正規表現でマッチした文字列のグループを貯めておくマッチオブジェクト**「m.group(0)」**に代入させるためである。m.group( ) の引数が「0」で、グループ全てを貯めておくことになる。

関数内での処理は、まずディクショナリ変数 EnglishWords を、関数内外で共通して使えるようにするため、**global ステートメント**で宣言する。これは **global 変数名**として使う。

次に、マッチしたオブジェクトを関数の引数から受け取り、全て変数 word に代入しておく。さらに、英単語が入った変数 word を全て小文字に直しておく。これには**変数.lower( )** メソッドを使い、同じ変数 word に代入し直す。

関数が実行された際、単語が代入されてくるたびにその回数を加算していくようにするには、まず英単語と出現回数のディクショナリ変数 EnglishWord に、その単語が存在するか確認し、すでにあったならば 1 を加算し、なかったならば、1 を代入する。

これらの材料が揃ったら、プログラム本体を作る。まず、対象ファイルの入った変数 file_name を読み込みで開いて、1 行ずつ繰り返し変数 line に代入する。

そのループ内で、この変数 line にある英単語を検索して、作成した関数 toroku に当て嵌めていく。これには本来置換の処理を行う**「正規表現.sub(置換文字列、検索文字列)」**を使い、「置換文字列」の代わりに作成した関数 toroku を代入しておく。これで、検索対象となる変数 line に、正規表現の変数 re_alpha にマッチした文字列が関数 toroku に送られることになる。

またディクショナリのキーを全て英単語として表示させるため、キーの全リストを返す**「ディクショナリ変数.keys(キー)」**を用い、変数 words に代入しておく。これも見やすいようにリスト用のメソッド**「対象変数.sort( )」**でソートしておく。

ソートした英単語の変数 words を、1 つずつ繰り返し変数 tango に代入し、そのたびに代入された英単語変数 tango と、その変数 tango に対応する要素となる出現回数を、文字列フォーマットを用いて書き出す。

(4) ＜擬似コード＞

sys モジュールと re モジュールをインポートする

変数 re_alpha に、正規表現『r'[\'-a-zA-z]+'』をコンパイルして代入
変数 EnglishWords に、空のディクショナリを代入
変数 file_name に、第 1 引数を代入

関数名 toroku で引数名 m の関数作成：
　　ディクショナリ変数 EnglishWords をグローバル変数として宣言
　　変数 word に、マッチオブジェクトを全て代入
　　変数 word に、変数 word を全て小文字に変換したものを代入

　　もし ディクショナリ変数 EnglishWords のキーに変数 word と合致するものがあったならば：
　ディクショナリ変数 EnglishWords のキーを変数 word に、1 を足して代入したものを要素として追加
　　それ以外：
　ディクショナリ変数 EnglishWords のキーを変数 word に、1 を要素として追加

変数 line に、ファイル変数 file_name を 1 行ずつ繰り返し読み込みで開いて代入：
　　変数 line に含まれる、正規表現変数 re_alpha とマッチするものを検索し、関数 toroku で処理する
　　変数 words に、ディクショナリ変数 EnglishWords のキーの全リストを代入
　　変数 words をソート

変数 tango に、変数 words を 1 つずつ繰り返し代入：
　　『%s：%d』の % 以下を、変数 tango とディクショナリ変数のキー変数 tango に対応する要素、に順番に置換して文字列と整数で書き出す

(5) **＜プログラム＞**

```
import sys, re

re_alpha = re.compile(r'[\'-a-zA-z]+')
EnglishWords = {}
file_name = sys.argv[1]

def toroku(m):

    global EnglishWords
    word = m.group(0)
    word = word.lower()
    if EnglishWords.has_key(word):
        EnglishWords[word] += 1
    else:
        EnglishWords[word] = 1


for line in open(file_name, "r"):
    re_alpha.sub(toroku, line)
words = EnglishWords.keys()
words.sort()

for tango in words:
    print "%s: %d" % (tango, EnglishWords[tango])
```

―――――――――――――――――――――――――――――――

(1) ＜＜問題 11 ＞＞

指定したファイルから、そこに書かれた単語を対話的に受け取り検索し、その単語とマッチした行と行番号、合計単語数を表示するプログラム。

対象ファイルの指定は、コマンドラインから引数で受け取り、検索単語の入力は対話的に求めるようにする。

(2) (問題把握)

コマンドプロンプトの引数から、対象ファイルを受け取る。

また検索語句はユーザーから対話的に受け取る。

検索文字列の入力は、何も入力されない場合以外繰り返し単語入力を求める。何も入力されなかったら、終了する。

検索する単語が一つも合致しなかったら、その旨を表示し、再び単語の入力を求める。

見つかった場合の表示は、『行数：行』、『合計単語数』とする。

検索単語とファイル内のデータをマッチさせるには、正規表現を用いる。

```
＞＞＞ python 実行ファイル名 対象ファイル名
```

処理の流れは、質問内容を変数に代入した後、その内容をユーザーから対話的に受け取り、変数に代入する。

終了条件に当て嵌まらないあいだ、繰り返すループを作り、その中で正規表現に直した検索語句を探す。

ファイル行と合計単語数を加算する変数を初期化しておく。

また、単語の検索に使用するため、受け取った変数をコンパイルしておくが、その際エラー処理すようにする。

検索語句が見つかった場合と見つからなかった場合を条件判定にかけ、見つかったら『行：内容』と『計何個』だったかの合計単語数を書き出す。

ループの最後で、再び繰り返し新しい検索語句を求め、ループから抜け出た終了時に『SEE YOU!!』と書き出す。

(3) (解説)

sys モジュールをインポートしておく。また正規表現のため re モジュールもインポートする。

ファイル名を入れておくため、変数 file_name に第 1 引数を代入する。またループ内で同じ質問を入力するため、先に質問を、変数 question に代入しておき、それを変数 match に文字列の形で対話的に入力を求め代入する。

先の変数 match に『何も入力されない』間は、ループを続けるようにしておく。また、ループの最後に変数 match を設けることで、行の表

示が終わるか、単語が見つからなかった後、そこで入力された新たな単語についての処理ができるようにする。これで検索単語は何度も新しく入力されていくので、その検索単語を元にした全処理は、ループ内で実行していく。
　次にファイルの行数と単語数を加算していくため、それぞれ変数 i と変数 x に 0 を代入し、初期化しておく。

　検索単語を正規表現でコンパイルし、それがファイルの内容と一致するか調べるが、検索単語には、片括弧などコンパイルできない文字列が含まれる可能性がある。コンパイルできなければ以下の処理が動かないので、それを回避するため、例外処理を挿入する。
　例外処理とは、実行された処理でエラーが発生した場合、そのエラーに遭遇したときの処理を予想し実行させていくことで、エラーによるプログラムの中断を回避することである。

```
try：
　　処理できるか最初に実行されるコード
except　特定のエラー：
　　特定のエラーが発生したときに実行されるコード
except：
　　上記以外のエラーが発生したときに実行されるコード
else：
　　try ブロックでエラーが発生しなかったら実行されるコード
　　(else は必ずしも入れる必要はない)
```

　この場合、**try ステートメント**の処理でコンパイルできるか実行した後、**except ステートメント**でコンパイルできなかったことを表示させ、else：以下に実行できた場合のプログラムを書いていく。

　単語の検索がヒットした場合の処理は、else：以下に書く。
　まずファイル変数 file_name を読み込みで開き、1 行ずつ繰り返し変数 line に代入していく。また 1 行読み込まれるごとに、変数 i に 1 を足し、代入する。
　コンパイルした検索単語の入った変数と、ファイルの 1 行が入った変数 line とがマッチするものを検索し、それを変数 found に代入する。
　正規表現の検索には、**「検索文字列.findall(対象文字列)」**を使用する。これはリストの形で結果を返すので、合計単語数を数えるには**「len(要素)」**を使い、リストの個数を抽出し、変数 list に代入しておく。さらにこれを変数 x に足し、代入する。

　検索単語とマッチしたとき表示させるには、リストの個数の変数 list に 1 つ以上入っていた場合、という条件を作り、それに該当した場合に書き出すようにする。
　合計単語数が、ファイルを全部読み込んだ後に 1 つもなかった場合には、『合致する単語が見つかりません』と書き出す。
　最後にループから抜け出たとき、『SEE YOU!!』と書き出す。

(4) ＜擬似コード＞

sys モジュールと re モジュールをインポートする

変数 file_name に、第 1 引数を代入
変数 question に、『検索文字列を指定→』を代入
変数 match に、変数 question に対し対話的に入力された答えを文字列で代入

変数 match に「何も入力されていない」ことがないあいだ：
　　変数 x に 0 を代入し初期化
　　変数 i に 0 を代入し初期化

　　エラーがあるか：
　　変数 re_match に、コンパイルした正規表現変数 match を代入
　　エラーがあれば：
　　『入力 error』と書き出す
　　それ以外は：
　　変数 line に、ファイル変数 file_name を 1 行ずつ繰り返し読み込みで開いて代入：
行を数える変数 i に、1 を足し代入
変数 found に、変数 line に含まれる正規表現変数 re_match を検索し代入
変数 list に、マッチした正規表現変数 found の個数を数えて代入
合計単語数を数える変数 x に変数 list を足し代入

もし 変数 list が 1 以上ならば：
　　変数 i,『行目：』, 変数 line, を、書き出す
　　もし 合計単語数を数える変数 x が 0 と同じならば：
　　『合致する単語が見つかりません。』と書き出す

　　変数 match に、変数 question に対し対話的に入力された答えを文字列で代入

『SEE YOU!!』と書き出す

(5) ＜プログラム＞

```
import sys, re

file_name = sys.argv[1]
question = "検索文字列を指定→"
match = raw_input(question)

while match != "":
    x = 0
    i = 0

    try:
        re_match = re.compile(match)

    except:
        print "入力 error"

    else:
        for line in open(file_name, "r"):
            i += 1
            found = re_match.findall(line)
            list = len(found)
            x += list

            if list >= 1:
                print i, "行目:", line,
                print "*計", x, "個*"
        if x == 0:
            print "合致する単語が見つかりません。"

    match = raw_input(question)

print " SEE YOU !!"
```

13 付録

――――――――――――――――――――――――――――――――

(1) Python のツール集

ここでは、Python で基本的に使用する関数やステートメントなど、
プログラム作成のためのツールを簡単にまとめてある。

・オブジェクトの型

型	説明
int( )	整数に変換
str( )	文字列に変換
float( )	実数に変換

・数値と文字列の処理

演算子	説明
+	加算・連結
-	減算
/(//)	除算
*	乗算・繰り返し
**	べき算
%	剰余
A > B	B より大きい・小さい
A >= B	B 以上・以下
A == B	A と B は等しい
A != B	A と B は等しくない
not B	B は偽である
A is B	A と B は全く同じもの
A in B	B は A の中のものと同じ
%s	文字列の型の文字列フォーマット
%d	整数の型の文字列フォーマット
%f	実数の型の文字列フォーマット
input( )	入力されたデータを整数で代入
raw_input( )	入力されたデータを文字列で代入

・リストとディクショナリの操作

操作	説明
L = [ ]	リスト L の初期化
L[1]	リスト L の 1 番目を指定 (インデクシング)
L[1:]	リスト L の 1 番目以降を指定 (スライシング)
L.append(1)	リスト L に 1 を追加
L.extend(1, 2)	リスト L に 1, 2 を複数追加
L.pop( )	リスト L の末尾を削除

del L[1]	リストの 1 番目を削除
L.sort( )	リスト L をソート
L.reverse( )	リスト L を反転
range(2)	2 までの整数のリストを作成
D = { }	ディクショナリ D の初期化
D[’key’] = ’ 値’	ディクショナリのキーに要素を追加・変更
D[’key’]	キーに対する要素を取り出す
del D[’key’]	キーに対する要素の削除
D.has_key(’ 値’)	特定の要素が含まれているかの確認
D.keys( )	D の全キーを取り出す
D.values（ ）	D の全値を取り出す
D.update(D1)	別々のディクショナリを連結

・その他のメソッド

関数	説明
S.replace(’old’, ’new’)	文字列 S から’old’ を検索して、全て’new’ と置換
S.count(’ 文字’)	文字列 S から’ 文字’ の出現する回数を取り出す
S.find(’ 文字’)	文字列 S から’ 文字’ の出現する最初のインデックスを返す
S.lower( )	文字列 S を小文字に変換
S.upper( )	文字列 S を大文字に変換
S.strip( )	文字列 S 内の空白、もしくは引数で指定された文字列を除去
S.rstrip( )	文字列 S 内の末尾を除去
S.split(’ 文字’)	文字列 S 内の、’ 文字’ を目印に分割
S.join(L)	リスト L を文字列に変換
len(S)	変数 S のデータ数を数える関数

・ファイル操作の処理

操作	説明
FILE = open(’file_name’, ’r’)	ファイル名変数 file_name を読み込みで開く
file = open(’file_name’, ’w’)	ファイル名変数 file_name を書き込みで開く
FILE.read( )	ファイル変数 FILE 全体を、1 つの文字列として読み込む
FILE.readline( )	ファイル変数 FILE の行を読み込む
FILE.readlines( )	ファイル変数 FILE の各行を文字列のリストとして読み込む
file.write(S)	ファイル変数 file に文字列変数 S を書き込む
file.writelines(L)	ファイル変数 file にリスト変数 L を全て書き込む
FILE.close( )	ファイル変数 FILE を閉じる

・**条件判定**

ステートメント	説明
if ～： elif ～： else：	もし～ならば： もしくは～ならば： それ以外は：

・**繰り返し処理**

ステートメント	説明
while ～： for ～： else： break continue	～であるあいだ: ～を繰り返す それ以外は 無条件でループから抜ける そのループの先頭に無条件で戻る
for A in B：	A に、B から 1 つずつ取り出し繰り返し代入：

・**関数作成**

ステートメント	説明
def A(b)： return global A	関数名 A 引数名 (b) の関数を作成： 結果を返す 変数 A をグローバル変数に変換

・**正規表現モジュール**

関数	説明
re.compile(r’ 文字列’) re.sub(r’ 正規表現’, ’ 置換文字’, ’ 対象文字列’) re.finditer(r’ 正規表現’, 対象文字列) re.search(r’ 正規表現’, 対象文字列)	文字列をコンパイル 対象文字列から 検索用正規表現を検索し、全て置換文字に置き換える 対象文字列から検索された全ての正 規表現をマッチオブジェクトで返す 対象文字列から検索された最初の正規 表現をマッチオブジェクトで返す
特殊文字	説明
. (ドット) ^ (キャレット)	改行以外のあらゆる文字列 文字列の先頭

$	文字列の末尾
*	直前の文字が 0 回以上使われているもの
+	直前の文字と 1 回以上使われているもの
?	直前の文字と 0 回か 1 回使われているもの
\w・[a-zA-Z0-9]	あらゆる英数字
\d・[0-9]	あらゆる整数
[ ]	文字の集合
[^5]	5 以外のあらゆる文字

・例外処理

ステートメント	説明
try A： except B： else：	A のエラーが発生したら： B のエラーが発生したら： それ以外：

・その他モジュール

モジュールと関数	説明
sys.argv[1]	sys モジュール. コマンドラインの第 1 引数を返す
sys.exit( )	Python を終了する
os.system( )	os モジュール. 引数で指定されたコマンドを実行
path = os.cwd( )	使用中のディレクトリのパス名を返す
os.rename(file, file_name)	ファイル名 file を file_name に変更
os.remove(path)	指定されたパスのファイルを削除
shelve.open(file_name)	shelve モジュール. 指定されたファイルをディクショナリに似た形で永続性を持たせて開く
random.randint(a, b)	random モジュール. 整数 a から整数 b までの乱数を生成

(2) 参考文献

文献表

Mark Lutz・David Ascher(著)・夏目大 (訳)
　2004　『初めての Python 第 2 版』オライリー・ジャパン
アラン ゴールド (著)・松葉素子 (訳)
　2001　『Python で学ぶプログラム作法』ピアソン・エデュケーション
福田洋一
　2004　『プログラミング演習』人文情報学演習、授業配布資料