プログラミング演習３（Python）

■Python の基本記法

○１．算術演算
加算　+
減算　-
乗算　*
除算　/
a を b で割ったときの余り
　　a%b
累乗　**
切り捨て除算　//

【例１】
```
1+2
3**3
17//5
```

【問１】
35 を 3 で割ったときのあまりは？

○２．比較演算子
大なり　>
以上　　>=
小なり　<
以下　　<=
等しい　==
等しくない　!=

※満たしている場合は True
　満たしていない場合は False

【例２】
```
2 > 1
1 != 1
```

○３．データ型
整数 int
浮動小数点数　float
文字列　str
など

※型は type()で確認できる

【例３―１】整数
```
var1=1
type(var1)
```
【例３－２】浮動小数点数
```
var2=1.56
type(var2)
```
【例３－３】文字列
```
var3='文字列'
type(var3)
```
【例３―４】その他
```
a = 1==1
type(a)
```

上記の「var1」,「var2」,「var3」は変数と呼ばれ、数値や文字列などが入る箱のようなものである。

○４．リスト
```
str_list = ['a','b','c']
num_list = [1,2,3,4,5]
```

リストは、上記のように複数の数値や文字列を格納することができる

要素番号は０から始まり
２番目を取り出す場合は

```
str_list[2]
```

と指定して取り出す
・リストの出力
for 変数名 in リスト名:
　処理
の書式を使ってリストを出力できる

数値や文字の出力には組み込み関数の
print()を使う

【例４－１】リストの出力
```
list[0,1,2]
for listout in list
  print(listout)
```

・リストの編集
【例４－２】
```
num_list[0]
num_list[0:2]
```

要素を削除するときは
remove や　del を使う
要素を指定するか、

要素番号を指定するか使い分ける

【例４－３】要素を指定する場合
```
str_list.remove('a')
print(str_list)
```

【例４－４】要素番号を指定する場合
```
del num_list[2]
print(num_list)
```

要素を追加するときは
append を使う

【例４－５】
```
str_list = ['a','b','c']

str_list.append('d')
print(str_list)
```

◎５．条件文

if ～(elif)else～のような形で
条件を指定する

```
a = "blue"
if a=='red':
  print('apple')
elif a=="blue":
  print('sky')
else:
  print('earth')
```

◎６．ループ処理

繰り返しの処理をする際には
for 文を使う
for 変数名　in データの集まり
という形にする

【例６－１】0-9 までの値をすべて足す
```
n=0
for i in range(10):
        n += i
print(n)
```

※range(10)は 0 から 9 までの整数を
順番に返す

ループ処理と条件文を合わせたような処理
をするときは
while を使う
while のある条件を満たす間は

while 以下の処理を実行し続ける
下記は n が 10 より小さい間は
その数を表示させ
10 になったら、その下の処理
に移る。

```
n=0
while n < 10:
        print(n)
        n += 1

print('finish:',n)
```

○7．関数

何度も発生する処理は関数として
定義しておくと可動性があがる

下記のように
def(引数 1,引数 2,引数 3,...)
と定義する

```
def sum(a,b):
  return a + b
```

関数を利用するときは
次のように引数を与える

```
sum(2,5)
```

（＋α）
■タプル

リストと同様に複数の値を
保存できる
タプルは（）で囲んで定義する

```
a = (0,1,8,12)
print(a)
```

【例】要素番号を使って取り出す

```
print(a[3])
```

※タプルは変更できないので
　下記のように書くとエラーになる

```
a = (0,1,8,12)
a[2] = 3
```

■辞書（ディクショナリ）

key と value をセットにした構造
{}で値を定義する

```
dict = {'a':20,'b':30,'c':15}
```

key を指定することで値を取り出す
事ができる

【例】

```
print(dict['a'])
```

■print()のキーワード引数を使う
　print()はいくつかのキーワード引数を
指定できる。
print()のキーワード引数
引数：説明
sep：指定した文字で区切りながら出力する
end：指定した文字を最後に出力する
file：write メソッドを持つオブジェクト
（ファイルなど）に出力する
flush：呼び出したときにメモリに一時的に
保持せず即時書き込む

・sep
sep=','のように区切り文字にカンマを指

定するとCSVデータのような出力になる
```
for i in range(10)
        print(i,i+1,i+2,sep=',')
```
（結果）
0,1,2
1,2,3
・・・
9,10,11

・end
　指定したデータを出力するときに最後の文字を指定する。デフォルトの値は改行文字の「¥n」である。Printが実行されるたびに改行されるのはデフォルトが改行になっているからである。ここではこの設定を変えて end=''として、改行されないようにする
（コード）
```
for i in range(10):
        print(i, end='')
print()
```
(結果)
0123456789
・fileとflush
通常 print を実行すると端末に出力されるが、引数の file を指定すると、その内容をファイルに出力できる。
多くの場合、file と flush は一緒に使い、flush は print()が実行されるごとに、即時書き込みするかどうかを決めるフラグである。デフォルトでは False となっていて、即時書き込みはしない設定になっている。それはある程度データ量がたまるまでメモリに保持しておいてまとめて出力したほうが効率がいいからである。そのため、print()を実行するごとに出力したい理由がなければ flush を指定する必要なない。
ここでは、flush=True に設定している。

```
filename = 'print.csv'
with open(filename,'w') as f:
  print(1,2,3, sep=',', file=f, flush=True)
  print(4,5,6, sep=',', file=f, flush=True)
  print(7,8,9, sep=',', file=f, flush=True)
```

■フォーマット済み文字列リテラル
　文字列に変数の値を表示するとき、フォーマット済み文字列リテラルを使うと楽である。
```
value = 'ABC'
print(f'変数の値は {value}')
```
従来は
```
print('変数の値は %s' % value)
```
のように文字列なら%s,整数なら%d と変数の方によって指定を変える必要があった。

■文字列の表現方法について
　文字列を扱う時、「'」「"」の２種類の記号で囲って扱えるようにすることで、どちらかの記号を表示させることができる。
```
value='ABC'
print(f'変数の値は"{value}"')
print(f"変数の値は'{value}'")
```
{}（波かっこ）はそのまま書けばよい
```
print(f"変数の値は{{value}}")
```

■便利な文字列表現

フォーマット済み文字列リテラルを使うと、次のようなちょっとしたカスタマイズができる。

・指定した桁数でゼロ埋め
```
value=5
print(f'3桁でゼロ埋め：{value:03}')
```

・３桁区切りでカンマを表示

```
value=123456789
print(f'3桁でカンマ区切り：{value:,}')
print()
```

・指定した桁数で右寄せ・中央・左寄せ

```
value='右寄せ'
print(f'8桁で：{value:_>8}')
value='左寄せ'
print(f'8桁で：{value:_<8}')
value='中央'
print(f'8桁で：{value:_^8}')
```

（参考）
森本哲也・中野正輝・池徹・岡田幸大著「できる仕事がはかどる Python 自動処理 全部入り」（インプレス）

■リスト内包表記

下記のような、for 文でリストに値を追加していくような処理をする際には、リスト内包記法を使う

```
list_a=[]
for i in range(10):
    list_a.append(i**2)
print(list_a)
```

というコードをリスト内に for 文を書くようなイメージで、リスト内包記法を使うと

```
list_b=[i**2 for i in range(10)]
print(list_b)
```