第 2 章

Python の文法 (1/2): 変数，オブジェクト，条件分岐

今まで見てきたように，変数とはデータを格納しておくための名前付きの箱のようなもので，名前が重複しない限り，スクリプト中のどこでも使うことができます。大変便利なものですが，その中身，つまりどのようなデータが入っているか，ということについては人間の方で把握しておく必要があります。コンピュータが扱えるデータは例外なく 2 進数として表現される bit 列の塊ですが，それを文字として扱うのか，数として扱うのか，それともデータのありかを示す番地として扱うのか・・・つまり，どのように解釈すべきものかということは最低限，スクリプト実行時には確定していなくてはいけません。本章では「変数」の中身について，スクリプト例に基づいて解説していきます。

2.1 変数とデータ型

通常，コンピュータで扱うことのできる基本的なデータ型 (datatype) は，表 2.1 に示すように，数値データ型 (整数型 (integer), 実数型（real number, 浮動小数点型, floating-point)) とシーケンス (文字列型に 2 分されます。

表 2.1 主な Python の基本データ型

種別	**データ型名称**	**Python 表記**
数値	整数型 (Integer)	int
	論理型 (Boolean)	bool
	浮動小数点型 (floating-point)	float
	複素数型 (complex)	complex
シーケンス	文字列型	str
その他	データ型なし	NoneType

整数型と同一視できるものとして，ブール型 (boolian, true(1) か false(0) のみ) があります。単語や文章のように文字が結合されているものを文字列型 (string) と呼びます。

実数型の拡張としては，複素数型 (complex number) があり，二つの実数をそれぞれ実部 (real part) と虚部 (imaginary part) とし，実部と虚部のセットとして表現します。幾何学的には 2 次元座標と同一視できま

すので，実部を横軸，虚部を縦軸としたガウス平面として表現することもあります。

Python に限らず，主としてインタプリタを使ってプログラム（スクリプト）を動かすコンピュータ言語では，初めて使う変数の中身に相応しいデータ型自動的に割り当てられます。Python の場合は type 関数を使うことで，その変数のデータ型を知ることができます。

例えば次の variable.py を動作させてみましょう。変数 a から変数 f まで値を代入し，それにふさわしい s 変数型になっている稼働を可を確認することができます。

ソースコード 2.1　variable.py

```
1  # variable.py: 変数とデータ型
2  a = 1 # 整数 (integer)
3  b = True # bool 値 (True か False)
4  c = -3.0 # 実数 (float)
5  d = -2 + 5 * 1j # 複素数 (complex)
6  e = 'abcdefg' # 文字列 (ASCII 文字)
7  f = '吾輩は猫である' # 文字列 (UNICODE UTF-8)
8
9  print('a, type(a) = ', a, type(a))
10 print('b, type(b) = ', b, type(b))
11 print('c, type(c) = ', c, type(c))
12 print('d, type(d) = ', d, type(d))
13 print('e, type(e) = ', e, type(e))
14 print('f, type(e) = ', f, type(f))
```

この結果を下記に示します。それぞれ変数の値と共に，「<class ' データ型'>」という表記がなされていることが分かります。ここで class(クラス) とは，データ型と共に，そのデータに対する操作（関数や演算）を規定したオブジェクトの一種です。

variable.py の実行結果

```
a, type(a) =  1 <class 'int'>
b, type(b) =  True <class 'bool'>
c, type(c) =  -3.0 <class 'float'>
d, type(d) =  (-2+5j) <class 'complex'>
e, type(e) =  abcdefg <class 'str'>
f, type(e) =  吾輩は猫である <class 'str'>
```

それぞれ変数の値と共に，「<class ' データ型'>」という表記がなされていることが分かります。ここで class(クラス) とは，データ型と共に，そのデータに対する操作（関数や演算）を規定したものです。

例えば，整数型 (int)，実数型 (float)，複素数型 (complex) には，四則演算が定義されており，下記のスクリプトに示すように，相互に利用することができます。この場合，$a = 1, b = 2$ とし，整数型 (int_a, int_b)，実数型 (float_a, float_b)，複素数型 (complex_a, complex_b) にそれぞれ同じ値を代入して計算を行っています。

ソースコード 2.2　number_arithmetic.py

```
1  # number_arithmetic.py: 整数,実数,複素数の四則演算
2  int_a, int_b = 1, 2
3  float_a, float_b = 1.0, 2.0
4  complex_a, complex_b = 1.0 + 0 * 1j, complex(2.0, 0)
5
```

```
6  print('int_a,␣int_b␣=␣␣␣␣␣␣␣␣␣', int_a, int_b)
7  print('float_a,␣float_b␣=␣␣␣␣␣', float_a, float_b)
8  print('complex_a,␣complex_b␣=␣', complex_a, complex_b)
9
10 # 同じデータ型どうしの四則演算
11 print('a␣+␣b␣=␣', int_a + int_b, float_a + float_b, complex_a + complex_b)
12 print('a␣-␣b␣=␣', int_a - int_b, float_a - float_b, complex_a - complex_b)
13 print('a␣*␣b␣=␣', int_a * int_b, float_a * float_b, complex_a * complex_b)
14 print('a␣/␣b␣=␣', int_a / int_b, float_a / float_b, complex_a / complex_b)
15
16 # データ型混合四則演算
17 print('a␣+␣b␣=␣', int_a + float_b, float_a + complex_b, complex_a + int_b)
18 print('a␣-␣b␣=␣', int_a - float_b, float_a - complex_b, complex_a - int_b)
19 print('a␣*␣b␣=␣', int_a * float_b, float_a * complex_b, complex_a * int_b)
20 print('a␣/␣b␣=␣', int_a / float_b, float_a / complex_b, complex_a / int_b)
```

同じデータ型どうしはもちろん，異なるデータ型に対しても正確に計算されていることが分かります。もちろん，そのような組み合わせに対応できる仕組みが Python 側に備わっているから可能なのであって，想定されていないデータ型どうしの演算は実行できないこともままあります。

問題 2.1

$a = 20240403$, $b = 20250401$, $c = 20340331$ とする時，次の式の値を計算するスクリプト `eval_long_formula.py` を作り，その結果を確認せよ。なおべき乗は**を使用し，例えば a^2 は `a**2` と記述する。

1. $\frac{a+b}{c+b}$ [ヒント: (a + b) / (c + b)]
2. ab^5c
3. $(a-b)^2 + (b-c)^3 + (c-a)^4$

2.2 複素演算の復習

Python における複素数 c, d は，実部と虚部はそれぞれ `real` と `imag` という変数に格納されています。従って例えば $c = 3 + 4\mathrm{i}$, $d = -2 - 5\mathrm{i}$ を代入し，その実部 ($\mathrm{Re}\ c = 3$, $\mathrm{Re}\ d = -2$) と虚部 ($\mathrm{Im}\ c = 4$, $\mathrm{Im}\ d = -5$) を取り出すには，下記のように `c.real`, `c.imag` と指定します。

```
>>> c = complex(3, 4) ← c = 3 + 4i を代入
>>> c.real ← c の実部を取り出す
3.0
>>> c.imag ← c の虚部を取り出す
4.0
>>> d = -2 - 5 * 1j ← d= -2 - 5i を代入
>>> d.real ← d の実部を取り出す
-2.0
>>> d.imag ← d の虚部を取り出す
-5.0
```

よく使用される複素演算を復習しておきましょう。虚数単位 $\mathrm{i} = \sqrt{-1}$ は $\mathrm{i}^2 = -1$ であることに注意すると，

$$
\begin{aligned}
\text{[加減算]}\ & c \pm d = (\mathrm{Re}\ c \pm \mathrm{Re}\ d) + (\mathrm{Im}\ c \pm \mathrm{Im}\ d)\mathrm{i} \\
\text{[乗算]}\ & cd = \{(\mathrm{Re}\ c)\cdot(\mathrm{Re}\ d) - (\mathrm{Im}\ c)\cdot(\mathrm{Im}\ d)\} + \{(\mathrm{Re}\ c)\cdot(\mathrm{Im}\ d) - a(\mathrm{Im}\ c)\cdot(\mathrm{Re}\ d)\}\,\mathrm{i} \\
\text{[共役]}\ & \overline{c} = \mathrm{Re}\ c + (-\mathrm{Im}\ c)\mathrm{i} \\
\text{[絶対値]}\ & |d| = \sqrt{(\mathrm{Re}\ d)^2 + (\mathrm{Im}\ d)^2} = \sqrt{d\cdot\overline{d}} \\
\text{[逆数]}\ & d^{-1} = \frac{1}{d} = \frac{d\overline{d}}{|d|^2} \\
\text{[除算]}\ & c/d = c\cdot d^{-1}
\end{aligned}
$$

となります。Python ではそれぞれ次のように計算できます。

```
>>> print(c, d) ← c, d の値の確認
(3+4j) (-2-5j)
>>> c + d ←加算
(1-1j)
>>> c - d ←減算
(5+9j)
>>> c * d ←乗算
(14-23j)
>>> c.conjugate() ← 共役
(3-4j)
>>> abs(c) ← 絶対値
5.0
>>> d ** (-1) ←逆数 (1)
(-0.06896551724137931+0.1724137931034483j)
>>> 1 / d ←逆数 (2)
(-0.06896551724137931+0.1724137931034483j)
>>> c / d ←除算
(-0.896551724137931+0.24137931034482757j)
```

問題 2.2

$c = -3-4\mathrm{i}$, $d = 2+5\mathrm{i}$ として, $c+d$, $c-d$, cd, c/d の値を求める Python スクリプト `complex_arithmetic.py` を作り，その結果を手計算と比較して確認せよ。

2.3 文字列演算と標準入力

コンピュータはキーボード，マウス，ファイル等，データを入力する装置と，ディスプレイ，ファイル等，データを書き出す装置がついています。これらを合わせて入出力 (I/O, Input and output) 装置と呼びます。このうち，よく使用するものを標準入出力 (standard I/O) と呼び，キーボードは標準入力 (standard input),

ディスプレイを標準出力 (stardard output) もしくはコンソール (console) と呼び，Python では print 関数が標準出力関数となります。

では，Python の標準入力関数 input を使ってみましょう。キーボードから入力された値は全て文字列型になります。

ソースコード 2.3　input_string.py

```
1  # input_string.py: 文字列の入力
2  str_a = input('str_a␣=?␣') # str_a の入力
3  str_b = input('str_b␣=?␣') # str_b の入力
4
5  # str_a, str_b の確認
6  print('str_a,␣str_b␣=␣', str_a, str_b)
7
8  # 文字列の長さ
9  print('len(str_a),␣len(str_b)␣=␣', len(str_a), len(str_b))
10
11 # 文字列の連結
12 print('str_a␣+␣str_b␣=␣', str_a + str_b)
13
14 # 文字列の最初の一文字目と最後の文字を表示
15 print('First␣and␣last␣char␣of␣str_a␣=␣', str_a[0], str_a[len(str_a) - 1])
16 print('First␣and␣last␣char␣of␣str_b␣=␣', str_b[0], str_b[len(str_b) - 1])
```

これを実行すると，次のようになります。

input_string.py の実行結果:ASCII 文字

```
str_a =? Tomonori ←入力
str_b =? Kouya    ←入力
str_a, str_b =  Tomonori Kouya
len(str_a), len(str_b) =  8 5
str_a + str_b =  TomonoriKouya
First and last char of str_a =  T i
First and last char of str_b =  K a
```

日本語も処理できます。

input_string.py の実行結果: 日本語入力

```
str_a =? 幸谷 ←入力
str_b =? 智紀 ←入力
str_a, str_b =  幸谷 智紀
len(str_a), len(str_b) =  2 2
str_a + str_b =  幸谷智紀
First and last char of str_a =  幸 谷
First and last char of str_b =  智 紀
```

このように，標準入力関数 input は受け取ったデータを全て文字列として解釈します。そのため，整数型，実数型，複素数型として数値データを受け取った時には型キャスト関数 int, float, complex インスタンス

を使って目的とするカスデータ型に変換する必要があります。

ソースコード 2.4　typecast.py

```
1  # typecast.py: 型キャスト例
2  str_a = input('a␣=?␣')
3  str_b = input('b␣=?␣')
4
5  # 型キャスト
6  int_a, int_b = int(str_a), int(str_b) # 整数型へ
7  float_a, float_b = float(str_a), float(str_b) # 実数型へ
8  complex_a, complex_b = complex(str_a), complex(str_b) # 複素数型へ
9
10 # データとデータ型の確認
11 print('int_a,␣type(int_a)␣=␣', int_a, type(int_a))
12 print('float_a,␣type(float_a)␣=␣', float_a, type(float_a))
13 print('complex_a,␣type(fcomplex_a)␣=␣', complex_a, type(complex_a))
```

typecast.py の実行結果

```
a =? 5 ← '5' を入力
b =? 4 ← '4' を入力 k
int_a, type(int_a) =  5 <class 'int'>
float_a, type(float_a) =  5.0 <class 'float'>
complex_a, type(fcomplex_a) =  (5+0j) <class 'complex'>
```

問題 2.3

複素数の入力を次のように行うスクリプト input_complex.py を作れ。

1. a を実部として入力
2. b を虚部として入力
3. $c := a + b \times \mathrm{i}$ としてまとめる
4. print 文で c を出力

2.4　書式付き出力

内部データを標準出力（ディスプレイ等）に文字列として表示する際に，形式を整えたいことがあります。例えば下記のような指定をしたい時に活躍するのが書式付き出力 (formatting output) です。

- 表示文字数
- 左寄せ，右寄せ，中央寄せ
- 小数点以下の桁数
- 指数表示

これらの指定を施した出力を行うため，print 文に対して 3 つの方法が使用できます。

1. 書式化演算子 % を利用
2. format メソッド

3. f 文字列指定

以下のスクリプトで，文字列，整数，浮動小数点数（実数）の出力事例を見ていくことにしましょう。

ソースコード 2.5 print_format.py

```
# print_format.py: 書式付き出力

str_a = input('a␣=␣')
str_b = input('b␣=␣')

# 文字列の表示
print('str_a,␣str_b␣=␣', str_a, str_b) # 書式指定なし
print('str_a,␣str_b␣=␣%10s,␣%s' % (str_a, str_b)) # 書式化演算子 %
print('str_a,␣str_b␣=␣{:10s},␣{}'.format(str_a, str_b)) # format メソッド
print(f'str_a,␣str_b␣=␣{str_a:10s},␣{str_b:s}') # f 文字列

# 整数の表示
int_a = int(str_a) # 文字列→整数
int_b = int(str_b)
print('int_a,␣int_b␣=␣', int_a, int_b) # 書式指定なし
print('int_a,␣int_b␣=␣%10d,␣%d' % (int_a, int_b)) # 書式化演算子 %
print('int_a,␣int_b␣=␣{:10d},␣{:d}'.format(int_a, int_b)) # format メソッド
print(f'int_a,␣int_b␣=␣{int_a:10d},␣{int_b:d}') # f 文字列，10進表示
print(f'int_a,␣int_b␣=␣{int_a:b},␣{int_b:b}') # f 文字列，2進表示
print(f'int_a,␣int_b␣=␣{int_a:o},␣{int_b:o}') # f 文字列，8進表示
print(f'int_a,␣int_b␣=␣{int_a:#o},␣{int_b:#o}') # f 文字列，8進表示，0o-prefix
print(f'int_a,␣int_b␣=␣{int_a:x},␣{int_b:x}') # f 文字列，16進表示
print(f'int_a,␣int_b␣=␣{int_a:#x},␣{int_b:#x}') # f 文字列，16進表示，0x-prefix

# 浮動小数点数 (実数)の表示
float_a = float(str_a)
float_b = float(str_b)
print('float_a,␣float_b␣=␣', float_a, float_b) # 書式指定なし
print('float_a,␣float_b␣=␣%15g,␣%g' % (float_a, float_b)) # 書式化演算子 %
print('float_a,␣float_b␣=␣{:15g},␣{:g}'.format(float_a, float_b)) # format メソッド
print(f'float_a,␣float_b␣=␣{float_a:15g},␣{int_b:g}') # f 文字列
print(f'float_a,␣float_b␣=␣{float_a:15.2f},␣{int_b:f}') # f 文字列，小数表示
print(f'float_a,␣float_b␣=␣{float_a:20.15e},␣{int_b:e}') # f 文字列，指数表示
```

print_format.py の表示例

```
a = 254 ←入力値 str_a
b = 511 ←入力値 str_b
str_a, str_b =  254 511
str_a, str_b =        254, 511
str_a, str_b = 254       , 511
str_a, str_b = 254       , 511
int_a, int_b =  254 511
int_a, int_b =        254, 511
int_a, int_b =        254, 511
int_a, int_b =        254, 511
int_a, int_b = 11111110, 111111111
int_a, int_b = 376, 777
int_a, int_b = 0o376, 0o777
int_a, int_b = fe, 1ff
int_a, int_b = 0xfe, 0x1ff
float_a, float_b =  254.0 511.0
float_a, float_b =             254, 511
float_a, float_b =             254, 511
float_a, float_b =             254, 511
float_a, float_b =          254.00, 511.000000
float_a, float_b = 2.540000000000000e+02, 5.110000e+02
```

問題 2.4

複素数 c を入力し，書式付き出力するスクリプト output_complex.py を作れ。[ヒント: 実部 (c.real) と虚部 (c.imag) をそれぞれ float 型の書式付き出力を行って複素数の標準形として出力する。]

2.5 変数とオブジェクト

表 2.1 に示した通り，論理型 (Boolean type) は数値データ型の一種で，True(真, 1) と False(偽, 0) の 2 値しか取らないという特徴があり，後述する条件分岐で使用されます。「シーケンス」とは，文字列型のように複数の文字（データ）の連なりとして表現されるデータ型です。後述するリスト (list)，タプル (tuple)，集合 (set) もシーケンスの一種です。

Python では，変数は全てこれらのデータ型で規定されたオブジェクト (object) への参照 (reference) になっています。例えば a = 2024 と Python で記述すると，正確には個別の ID が付加された整数型のオブジェクトが生成され（メモリ上の領域が作られ）て 2024 という整数がこのオブジェクトに格納されます。「a」という変数は，この整数型のオブジェクトへの参照（矢印）が格納されており，代入 (=) はオブジェクトへの参照を変数に設定する処理に他なりません。

この動きを具体的に見るために，次の Python スクリプトを動かしてみましょう。変数が参照しているオブジェクトの ID を見るために id 関数を使用しています。

ソースコード 2.6　object.py

```
1  # object.py: 変数とオブジェクト
2  a = 2024 # (1)
3  print('a が参照しているオブジェクトのデータ型:␣', type(a))
4  print('a が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(a))
5  b = a # (2)
6  print('b が参照しているオブジェクトのデータ型:␣', type(b))
7  print('b が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(b))
8  print('a␣=␣', a)
9  print('b␣=␣', b)
10
11 a = 'This␣is␣a␣pen.' # (3)
12 print('a が参照しているオブジェクトのデータ型:␣', type(a))
13 print('a が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(a))
14 print('b が参照しているオブジェクトのデータ型:␣', type(b))
15 print('b が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(b))
16 print('a␣=␣', a)
17 print('b␣=␣', b)
```

これを動かすと下記のような実行結果が得られます。オブジェクトの ID は動かすたびに異なりますので，実際の ID 値は各自確認して下さい。

object.py の表示例

```
a が参照しているオブジェクトの ID      :  2228462526640
b が参照しているオブジェクトのデータ型:  <class 'int'>
b が参照しているオブジェクトの ID      :  2228462526640
a =  2024
b =  2024
a が参照しているオブジェクトのデータ型:  <class 'str'>
a が参照しているオブジェクトの ID      :  2228465708272
b が参照しているオブジェクトのデータ型:  <class 'int'>
b が参照しているオブジェクトの ID      :  2228462526640
a =  This is a pen.
b =  2024
```

代入を行っている (1), (2), (3) において実際に行われていることを図示したものが図 2.1 になります。(1) では整数型のオブジェクトへの参照が a に作られ，(2) でこの参照が b と共有されます。(3) では a の参照が文字列型へのオブジェクトに変更され，結果として a と b は異なるオブジェクトへの参照を保持することになります。

Python では全てのデータはオブジェクトとして扱い，変数はデータの実態ではなく，オブジェクトへの参照であると覚えておきましょう。ちなみに，変数からの参照を失ったオブジェクトは自動的に消去されます。このようなメモリ管理をガベージコレクション (gabage collection,「ゴミ集め」の意味) と呼び，不要なメモリ消費を極力抑えることができます。

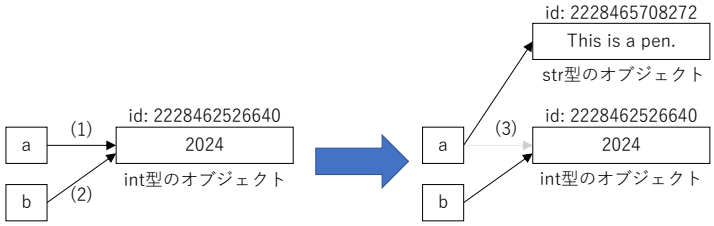


図 2.1　変数とオブジェクトの関係

2.6　数学関数の計算

実用上頻繁に使用される数学関数として，四則演算の他に，べき乗 (x^y)，平方根 $\sqrt{x} = x^{1/2}$, 指数関数 $\exp(x) = e^x$，自然対数関数 $\log_e x = \log x$，常用対数関数 $\log_{10} x$，三角関数 $\sin x$, $\cos x$, $\tan x$，逆三角関数 $\sin^{-1} x = \arcsin\ x$, $\cos^{-1}(x) = \arcsos\ x$, $\tan^{-1} x = \arctan\ x$ 等があります。べき乗以外は，実数関数は標準の math モジュール，複素関数は cmath モジュールが必要です。math, cmath モジュールは Python を標準インストーすれば使える標準モジュールですが，後述する NumPy では両方使えますので，pip コマンドなどで NumPy が使用できるようになったらこちらを使うようにして下さい。

2.6.1　math モジュール

良く使用する実数関数を使うためのモジュールです。下記のように，平方根，指数関数，三角関数，対数関数が使えます。

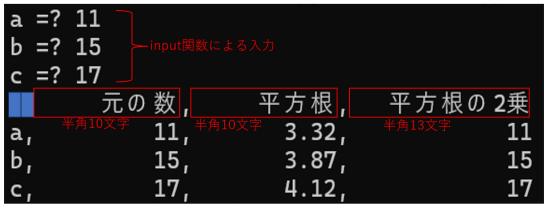
ソースコード 2.7　float_calc.py

```
1  # float_calc.py: 実数型の計算
2  import math # 数学関数
3
4  a = input('a␣=␣')
5  b = input('b␣=␣')
6  a, b = float(a), float(b)
7  # 四則演算
8  print('a␣+␣b␣=␣', a + b)
9  print('a␣-␣b␣=␣', a - b)
10 print('a␣*␣b␣=␣', a * b)
11 print('a␣/␣b␣=␣', a / b)
12
13 # 平方根: math.sqrt
14 print('sqrt(a)␣=␣', math.sqrt(a))
15
16 # べき乗
17 c = math.sqrt(b)
18 print('(sqrt(b))^2␣=␣', c ** 2)
19
20 # 指数関数,三角関数,対数関数
21 print('exp(a)␣=␣', math.exp(a))
22 print('sin(a)␣=␣', math.sin(a))
23 print('cos(a)␣=␣', math.cos(a))
```

```
24 print('tan(a)␣=␣', math.tan(a))
25 print('log(a)␣=␣', math.log(a))
26 print('log10(a)␣=␣', math.log10(a))
```

問題 2.5

キーボードから数値 a, b, c を入力し，float 型に変換した上で $\sqrt{a}$, $\sqrt{b}$, $\sqrt{c}$ を求め，さらにこれらの 2 乗を計算して元に戻ることを確認する sqrt_abs.py を作成せよ。



2.6.2 cmath モジュール

複素数を引数とする数学関数を使うには，cmath モジュールを使います。実数関数と同じ名前の複素関数は，基本，引数 x が実数である時は実数関数と同じ値を返すようになっています。

ソースコード 2.8 complex_func.py

```
1  # complex_func.py: 複素関数の計算
2  import cmath # 複素関数
3
4  a = 1.2 + 3.4j # 1.2 + 3.4 i
5  print('a␣=␣', a)
6
7  c = cmath.sqrt(a)
8  print('sqrt(a)␣=␣', cmath.sqrt(a))
9  print('sqrt(a)^2␣=␣', c ** 2)
10
11 # 指数関数,三角関数,対数関数
12 print('exp(a)␣=␣', cmath.exp(a))
13 print('sin(a)␣=␣', cmath.sin(a))
14 print('log(a)␣=␣', cmath.log(a))
15 print('log10(a)␣=␣', cmath.log10(a))
```

問題 2.6

次の計算を実行するスクリプト calc_formula.py を作れ。複素数として求める必要のあるものも含まれるので，エラーが出ない形で計算結果を出力できるように配慮せよ。

1. $\exp((-4/3)^3) = e^{(-4/3)^3}$
2. $\sqrt{-5}$
3. $\exp(\log -10)$
4. $\mathrm{i}^{\mathrm{i}} = (\sqrt{-1})^{\sqrt{-1}}$

2.7 条件分岐

未来のことは分かりませんが，あらかじめ起こり得る状況に応じて判断を変えることは日常的によくあります。このように，条件に応じてプログラムの流れを帰る際には if 文を使用します。

基本形は，ブール型を返す条件文を使用して

```
if 条件文:
    インデントブロック 1
else:
    インデントブロック 2
```

という形で定義します。条件文が真の時はインデントブロック 1 が実行され，そうでない時にはインデントブロック 2 の方が実行されます。

例えば，整数型の変数 a が偶数 (2 で割り切れる) か奇数 (2 で割った時の余りが 1) かを判断して「a is even.」(偶数の時), 「a is odd.」(奇数の時) と表示するスクリプト `odd_even.py` は，a を 2 で割った時の余りを\%（剰余演算）で求め，余りが 0 か (`a == 0`)，0 ではないか (`a != 0`) で判断します。

ソースコード 2.9　`odd_even.py`

```
1  # odd_even.py: 偶奇判断
2  a = int(input('a␣=␣')) # 文字列入力→整数キャスト
3
4  # 書き方 1
5  if a % 2 == 0: # 偶数
6      print(a, '␣is␣even.')
7  else: # 奇数
8      print(a, '␣is␣odd.')
9
10 # 書き方 2
11 if a % 2 != 0: # 奇数
12     print(a, '␣is␣odd.')
13 else: # 偶数
14     print(a, '␣is␣even.')
```

条件分岐が 3 つ以上になる場合は if〜else の間に elif 文を

```
if 条件文 1:
    インデントブロック 1
elif 条件文 2:
    インデントブロック 2
....
....
elif 条件文 n:
    インデントブロック n
else:
    インデントブロック n+1
```

のように挟みます。

例えば，a が

- 2 と 3 の公倍数
- 2 の倍数 (偶数)
- 3 の倍数
- 上記以外

の 4 つに分類する時には次のように elif 文をはさみます。

ソースコード 2.10　common_mult23.py.py

```
1  # common_mult23.py: 2と 3の公倍数
2  a = int(input('a␣=␣')) # 文字列入力→整数キャスト
3
4  if (a % 2 == 0) & (a % 3 == 0): # 2と 3の公倍数
5      print(a, '␣is␣a␣common␣multiples␣of␣2␣and␣3.')
6  elif a % 2 == 0: # 偶数
7      print(a, '␣is␣even.')
8  elif a % 3 == 0: # 3の倍数
9      print(a, '␣is␣a␣multiples␣of␣3.')
10 else: # 上記以外
11     print(a, '␣is␣not␣even␣and␣a␣multiples␣of␣3.')
```

数値データの大小関係を表現するためには，表 2.2 のような比較演算子が使用できます。

表 2.2　Python の比較演算子

演算子	意味	例（a=5, b=3）
==	等しい	a == b ⇒ False
!=	等しくない	a != b ⇒ True
<	より小さい	a < b ⇒ False
>	より大きい	a > b ⇒ True
<=	以下	a <= b ⇒ False
>=	以上	a >= b ⇒ True

問題 2.7

入力データが 10 のべき乗 ($1 = 10^0$, $10 = 10^1$, $1000 = 10^3$ 等) かどうかを判断する Pytho スクリプト power10.py を作成せよ。[ヒント: 10 で割った余りが 0 かどうかで判断する。但し 1 もきちんと処理できるようにすること。]

2.8　条件分岐の応用：2 次方程式の解法

ではここで 2 次方程式 $ax^2 + bx + c = 0$ を解くプログラム quadratic_eq.py を作ってみましょう。係数 a, b, c を標準入力（キーボード）から与え，解の公式 $x = (-b \pm \sqrt{b^2 - 4ac})/(2a)$ を使用して解を求めています。

ソースコード 2.11　quadratic_eq.py

```
1  # quadratic_eq.py: 2次方程式を解く
2  import math # sqrt 関数
3
4  # 係数入力
5  a = input('a␣=?␣')
6  b = input('b␣=?␣')
7  c = input('c␣=?␣')
8  a, b, c = float(a), float(b), float(c)
9  print(f'␣␣{a:25.17g}␣*␣x^2')
10 print(f'+␣{b:25.17g}␣*␣x')
11 print(f'+␣{c:25.17g}␣=␣0')
12
13 # 判別式
14 d = b ** 2 - 4 * a * c
15
16 x1 = (-b + math.sqrt(d)) / (2 * a)
17 x2 = (-b - math.sqrt(d)) / (2 * a)
18 # 出力
19 print(f'x1␣=␣{x1:25.17e}')
20 print(f'x2␣=␣{x2:25.17e}')
```

判別式 $d = b^2 - 4ac$ の値に応じてそれぞれ計算式を変えてみましょう。16 行目以降で if 文を使い，$d \geq 0$ の時は従来の計算式，$d < 0$ の時は実部と虚部の計算を行って複素数型として解を求めます。

ソースコード 2.12　quadratic_eq_mod.py

```
16 # 判別式
17 d = b ** 2 - 4 * a * c
18
19 if d >= 0: # 実数解の場合
20     print('Real␣solutions:␣\n')
21     x1 = (-b + math.sqrt(d)) / (2 * a)
22     x2 = (-b - math.sqrt(d)) / (2 * a)
23     # 出力
24     print(f'x1␣=␣{x1:25.17e}')
25     print(f'x2␣=␣{x2:25.17e}')
26 else: # 複素数解の場合
27     print('complex␣solutions:␣\n')
28     x1 = complex(-b / (2 * a), math.sqrt(-d) / (2 * a))
29     x2 = complex(-b / (2 * a), -math.sqrt(-d)) / (2 * a)
30     # 出力
31     print(f'x1␣=␣{x1:25.17e}')
32     print(f'x2␣=␣{x2:25.17e}')
```

問題 2.8

2 次方程式を解くスクリプトに次の改良を加えよ。

1. $a = 0$ の時，1 次方程式 $bx + c = 0$ を解けるようにせよ。
2. 複素数を係数とする 2 次方程式を解くプログラムを，cmath モジュールの sqrt 関数を使って作れ。この場合，判別式による場合分けは必要なくなる。

演習問題

1. 三角形 ABC の 3 辺の長さを $a = \mathrm{BC}$, $b = \mathrm{CA}$ とし，角 ACB を α とする。a, b, α をキーボードから入力し，次の手順で c, s, S を計算して出力する Python スクリプト `triangle.py` を作れ。
   (a) 入力された a, b, α から $c = \mathrm{AB}$ を余弦定理

   $$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

   を用いて求める（式変形は自力で行うこと）。
   (b) $s := (a + b + c)/2$ を求める。
   (c) ヘロンの公式 $S := \sqrt{s(s-a)(s-b)(s-c)}$ で三角形の面積を求める。
   余力のある人は，角度を度 (°) で入力できるようにスクリプトに一工夫加えること。
2. 3 つの異なる複素数 c_1, c_2, c_3 を入力し，この 3 点が描く複素平面上の三角形の面積を求める Python スクリプト `complex_triangle.py` を作れ。