だいたい正しい Python 入門

初心者のための基本入出力演習

出原至道 (idehara@tama.ac.jp)

2022 年 5 月 9 日

目 次

1 プログラムを組む準備をする

1.1 Repl.it を使う

1. Repl.It に接続する
2. Sign Up を選ぶ
3. アカウントを作成する
   - GitHub 連携を使うなら、GitHub アカウントでログイン
   - 大学の gmail アカウントを使うなら、Google アカウントでログイン
   - それ以外なら、適切に情報を入力する

1.2 Google Colaboratory を使う

1. Colaboratory に接続する
2. Google アカウントでログインする

2 表示ができるようになろう

2.1 プリント命令で計算しよう

```
print(1+1)
```

print は、「( ) の中のもの」を表示してくれる命令です。まず、いろいろ計算してみましょう。

- 「かける」は *、「割る」は / という記号を使います。

- ** は、「〜乗」を表す記号です。
- // と % は「整数を整数で割った商と余り」を表します。

例：

```
print(1*2*3*4*5)
print(1/3)
print(10//3)
print(10%3)
print(2**3)
print(2**0.5)
```

2.2　プリント命令で文字を表示しよう

```
print('hello')
```

print( ) の中には色々なものを入れることができます。今度は「文字列」を入れてみましょう。文字列とは、「文字が 0 個以上並んだモノ」です。

- python で文字列をプログラム中に書くときは " " か' ' のどちらかで囲います。
- 文字列に " が出てくるときは ' で囲い、' が出てくるときは " で囲うといいでしょう。

例：

```
print("I'm fine")
```

2.3　プリント命令で、複数のものを表示しよう

```
print('答え：', 3)
```

print( ) の中には、複数のモノを , で並べて書くことができます。

- 出力では、モノの間に　が入ります。
- 文字列や数字を混ぜてもかまいません。
- 計算式は、（最初の例のように）計算した結果を表示します。
- 文字列の中が計算式になっていても、「計算してあげようかな」などと思ってくれません。ただの文字として表示されます。

例：

```
print('1+2', 1+2)
print(' 円周率はだいたい', 3.1415926)
```

2.4　文字列の計算をしよう

```
print('abc' + 'efg')
print('123' * 10)
```

計算記号は、プログラム言語の「決めごと」なので、普通に使う意味とちがう動作をすることもあります。

- 文字列同士のあいだで + が計算できます。「文字列をつなぐ」処理をします。
- 文字列と整数の間で * が計算できます。「文字列を繰り返す」処理をします。

例：

```
print('Hello' + '!' * 20)
```

2.5　（ちょっと高度）改行したくないときは

```
print(' この言語は', end='')
print(' Python です。')
```

- python の print 命令は、何も指定しないと、表示後に改行します。
- print 命令の（ ）の中で、最後に end='' と書くと、改行しなくなります。これは、表示後に自動的に追加される文字を変更する指定です。

例：

```
print(' 計算結果は', end='')
print(1*2*3*4*5, end='')
print(' です')
```

3　変数を理解しよう

3.1　変数とは

```
x = 13
y = 11
print(x+y)
print(x/y)
```

- 「変数」とは、計算結果などを一時的に書き留めておけるメモのようなものです。（メモに書き込むことを「変数に代入する」といいます）
- 変数は名前をつけて区別します。最初の 1 文字がアルファベット（と _）で、その後にいくらでもアルファベット、数字、_ を並べることができます。
  - 習慣として、普通の変数はアルファベットの小文字で始めます。
  - 単語を組み合わせて一つの変数名にするときは、 2 個め以降の単語の最初を大文字にするのが python 風です。（たとえば `playerLifePoint`）
- 変数の代入には = 記号を使います。「右の値を左の変数に書き込む」という意味です。
- 上の例では、x というメモに 13、y というメモに 11 と書いてあります。
- あとから、そのメモを見ながら計算することができます。（変数を「参照する」といいます）
- 変数には、数字や文字列以外にも色々なモノを書き留めておけますが、それは出てきたときに。

例：

```
s1 = 'Hello'
s2 = 'World'
s3 = '!'
x = 10
y = 20
print(s1,s2,s3*x,x*y)
```

3.2 変数を書き換えよう

```
x = 10
x = x + 1
print(x)
```

変数はメモ書きです。後でいくらでも書き換えることができます。

- 新しく書き換えると、前の値はなくなります。

- 今のメモを見ながら計算して、あたらしい結果を書き込むので、`x = x + 1` のような（数学的には変な）式も全く問題ありません。
  - これは「今の `x` に書いてある数字を 1 増やした値を計算して、それを `x` に書き込む（まえの数字は消す）」を意味します。

例：

```
x = 2
print(x)
x = x * x
print(x)
x = x * x
print(x)
x = x * x
print(x)
```

3.3 変数の型を理解しよう

```
x = 123
s = '123'
print( x * 10 )
print( s * 10 )
```

変数に代入している（＝書き留めている）モノの種類のことを「型」といいます。

- `+` のように、「型によって動作が違う」演算があります。
- `-` のように、「ある型には定義されていない」演算があります。
  - このような場合 `TypeError` などのエラーが表示されてプログラムの実行が止まります。
- 文字列を繰り返す `*` は、左に文字列、右に整数を置かなければいけません。場所によって必要な型が決まります。
- `type( )` のカッコの中にモノを入れると、そのモノの型を教えてくれます。

例（最後の行でエラー）：

```
x = 123
s = '123'
print( type(x) )
print( type(s) )
```

```
print(x/10)
print(s/10)
```

3.4　型を変えよう

```
s = '123'
x = int(s)
print(s*10)
print(x*10)
```

ある型を別の型として扱いたいことが（よく）あります。このようなとき、「型変換」を行います。

- 文字列から整数に変換したいときは int( ) のカッコの中に文字列を入れます。もちろん、変数でも構いません。
- 整数から文字列に変換したいときは、str( ) のカッコの中に文字列を入れます。

例：

```
x = 123
s = str(x)
print(x*10)
print(s*10)
```

4　関数を理解しよう

4.1　関数とは

```
x = len('Hello')
print( x )
```

関数とは「モノを渡すと、なにか決まったお仕事をしてくれるヒト」のことです。これまで「( ) の中に〜を入れる」と書いてきたものが、すべて「関数」です。

- 関数には、大きく分けて２種類あります。
  - 仕事をお願いするだけで、返事が返ってくることは期待していないもの（たとえば print()）
  - 何かお仕事の結果を返事してくれるもの（たとえば int() len()）

- `print()` などの「返事が返ってこない」タイプの関数は、勝手に関数の方で仕事をしてくれます。（画面にモノを表示する）
- `len()` などの「返事をしてくれる」タイプの関数は、それを= の右において返事をメモ書きしたり、いきなりその返事を他の関数に渡してさらに仕事を進めたりします。
- 「関数に何を渡したら、何を返してくれるのか」について、覚える必要はありません。ネットで調べましょう。
- 関数には、正しい型を渡さないと怒られます。（エラーが出ます）
- なかには、なにもモノを渡さなくても仕事だけしてくれる関数もあります。（() の中が空）

例（最後の行でエラー）：

```
print( len('Hi') )
print( len(123) )
```

4.2 モジュールを使えるようになろう

```
import math
print( math.sin(1.57) )
```

Python にあらかじめ用意されている関数が「組み込み関数」です。組み込み関数以外にも、必要に応じて「これからこの種類の機能を使います」と宣言することで、使える機能を増やすことができます。

- この機能のかたまりが「モジュール」です。
- モジュールを使うときは、最初に `import` モジュール名で「これからこのモジュールを使います！」と宣言します。
  - モジュールを読み込むときに別名を指定したり、モジュールの一部分だけ読み込むようなワザもありますが、ここでは触れません。
- 読み込んだモジュールの機能は モジュール名 . 機能で使うことができます。
- 「機能」には、関数だけでなく、定数や、その他便利なモノが定義されています。
- 例によって、モジュールの名前や機能を覚える必要はありません。なんとなく「こんなのがあった」とだけ覚えておいて、あとは検索しましょう。

```
import math
print( math.factorial(5) )
print( math.pi )
```

```
print( math.e )
```

5　リストを理解しよう

5.1　リストとは

```
x = [1,2,3]
print(x)
print(x[1])
```

リストとは、「モノが順番に並んだもの」を表す型です。１個１個のモノを「要素」と呼びます。

- python では、リストの中の１個１個の要素は、型が違っていても構いません。
- リストをプログラム中で直接書くときは、[ ] で囲い、要素を , で区切ります。
- 順番があるので、リストから「最初の要素」「２番めの要素」などと読み出すことができます。読み出すときはリスト [順番] で読み出します。
- 順番は「０番」から数えるので、注意しましょう。
- 順番に、マイナスの数を使えます。[-1] が１番最後の要素、[-2] が後ろから２番め……のように指定します。

例：

```
d = ['hello! ', 3]
print(d)
print(d[0] * d[1])
```

5.2　落とし穴に注意しよう！！！

```
a = [1,2,3]
b = a
print(a,b)
a[0] = 999
print(a,b)
```

ここまで読んできた人は、上のコードを実行すると混乱するはずです。「a と b は別のメモですよね？　b に a を書き写したあとで、a を書き換えたんですけど？」　ここが python のリストの落とし穴です。

- 変数 = リストで代入すると、変数には「リストが置いてある場所」をメモします。リストの中身ではありません。
- 上の例では a も b も、同じ「場所」をメモしています。
- したがって、a にメモしてある場所のリストを書き換えると、(b も同じ場所をメモしているので) b からみたリストも書き換わります。
- これは、そうとう python を使い込んでいてもハマる落とし穴です。注意してください。
- これを避けるには、リスト.copy() で内容が同じ別のリストを作っておいて、この場所を記録します。

```
a = [1,2,3]
b = a.copy()
print(a,b)
a[0] = 999
print(a,b)
```

5.3 リストから別のリストを作ろう

```
a = [1,2,3,4,5]
b = a[1:3]
c = a[::2]
print(b)
print(c)
```

リストから別のリストを作るには、前に出てきた リスト.copy() の他に、取り出し場所・取り出し間隔を指定する方法があります。

- リストから要素を１個だけものを取り出すときは [順番] でした。リストからリストを取り出すときは [最初の場所: 何個目までか: 何個ごとか] を指定します。
  - 「最初の場所」は、要素を１個取り出すときと同じで、０番から始まります。
  - 「何個目までか」は、最初を１個めと数えた個数です。「場所」とは１個ずれます。
  - 「何個ごとにするか」を１以外にすると、飛び飛びに要素を取り出します。
- マイナスの数を指定すると、色々と便利な使い方ができます。
- 省略すると、それぞれ「最初から」「おしまいまで」「１個ごと（全て）」を指定したことになります。

例：

```
a = [1,2,3,4,5]
d = a[::-1]
e = a[:3]
f = a[3::-2]
print(d)
print(e)
print(f)
```

5.4　文字列をリストだと思って使えるようになろう

```
s = list('Hello!')
t = s.copy()
t[1] = 'a'
print(s)
print(t)
```

文字列を `list()` によって、１文字１文字がバラバラの要素になったリストに型変換することができます。

- リストにしてしまえば、リストで使えたワザが全て使えます。
- 別の変数に代入するとき、文字列だと別々のモノになりましたが、リストは場所を記録しているだけなところに注意しましょう。
- リストの要素を、間に特定の文字列をいれながらつないで文字列に型変換したいときは区切り文字`.join(リスト)` と書きます。
  - 区切り文字を `''` にすると、すべての文字がつながった文字列になります。

例：

```
s = list('Hello!')
t = s.copy()
t = t[::-1]
print(t)
print(''.join(t))
print(' -> '.join(t))
```

5.5　（ちょっと高度）`map()` 関数を使おう

```
s = ['12','34','56']
a = list(map(int, s))
```

```
print(s)
print(a)
```

「リストのすべての要素にある処理を加えたい」というとき、その処理が関数一つなら、map() 関数で簡単に書くことができます。

- map(関数，リスト) は、() の中に関数名を書く特殊な関数です。
- 「リストのすべての要素に関数を適用した結果を map 型で返してください」を意味します。
- 実際には、返ってきた map 型の値を、list() でリストに型変換するなどして使用します。
- 特に、データを文字列で読み込んでおいて、すべてを数値に変換するときなどに多用します。

例：

```
s = '123456'
print(s)
t = list(s)
print(t)
u = list(map(int, t))
print(u)
```

6　入力を読み取ろう

6.1　キーボードからの文字入力を受け取ろう

```
s = input()
print(s, 'さん、こんにちは')
```

input() で、キーボードからの１行の入力を受け取ることができます。

- 受け取った結果は、文字列型です。
- 受け取った結果を変数に書き込むのが普通です。
- input() を複数回書くことで、複数行の入力を受け取ることができます。

例：

```
print('名字？')
m = input()
print('名前？')
```

```
n = input()
print(m+n, ' さん、こんにちは')
```

6.2　一行で複数のデータを受け取れるようになろう

```
print(' スペースで区切って、名字と名前を入れてください')
s = input().split()
print(s)
```

文字列を、特定の文字でバラバラにしてリストにしてくれる関数 split() を使うと、一行に複数のデータが入っているときに、データを分割して取り出すことができます。

- split() の（ ）の中には、区切り文字に使いたい文字を指定します。何も指定しなければ「スペース」を使います。
- 結果はリスト型になります。

例：

```
print(' スペースで区切りながら、いくつか数字を入れてください')
s = input().split()
print(s)
print(s[-1], ' が最後の数字ですね')
```

6.3　数値データを受け取れるようになろう

```
print(' スペースで区切りながら、４つの数字を入れてください')
s = input().split()
print(s)
a = list( map(int,s) )
print(a)
```

これまでの集大成です。

1. 一行の文字列を読み込み
2. スペースでバラバラにして
3. それぞれの要素を文字列から整数に変換して
4. リストに変換しています。

これで、データを受け取る準備ができました。

例（sum はリストの合計を計算してくれる関数）：

```
print(' スペースで区切りながら、４つの数字を入れてください')
s = list( map(int, input().split()) )
print(' 合計', sum(s))
```

7　条件分岐

7.1　あるときだけ実行できるようになろう

```
print(' あなたの年齢を入れてください')
y = int( input() )
if y<6:
    print(' 本当ですか？')
print( y, ' 歳ですね' )
```

条件判断をする if 命令を使うと、「このときにだけ命令を実行して欲しい」という指示を書けます。「このときに」を表す式を「条件式」と呼びます。

書き方：

```
if 条件式:
    条件に当てはまったときに実行したい命令１
    条件に当てはまったときに実行したい命令２
    条件に当てはまったときに実行したい命令３
    ……
普通に実行したい命令
……
```

- python では、特定のプログラムの要素の影響にある部分（ブロック）を、行の最初を下げて表します。
- スペースの数は、通常４つのスペースを使います。
  - 他の人と一緒に作業するときには、別の指定があるかもしれません。それに従ってください。
- 条件式で使える代表的な記号：
  - 大小比較 <, >, <=, >=（＝を後ろに書くことに注意）
  - 等しい・等しくない ==, !=（「等しい」は＝を２個書く。１個だと「代入」）

例（色々な数を入力して、すべての場合を試してみよう）：

```
print('好きな自然数を入れてください')
s = input()
ni = int(s)
print(ni)
if ni % 7 == 0:
    print('7の倍数です')
if ni % 11 == 0:
    print('11 の倍数です')
if ni % 13 == 0:
    print('13 の倍数です')
```

7.2 if を組み合わせよう

```
print('好きな自然数を入れてください')
n = int( input() )
if n % 7 == 0:
    print('7の倍数です')
    if n % 11 == 0:
        print('しかも 11 の倍数です')
```

if のブロックの中に if を書くことができます。

- どんどんブロックが深くなります。
- ブロックの中は、外側の if が成り立たないと実行されないことに注意してください。

例（色々な数を入力して、すべての場合を試してみよう）：

```
print('好きな自然数を入れてください')
n = int( input() )
if n % 7 == 0:
    print('7の倍数です')
    if n % 11 == 0:
        print('しかも 11 の倍数です')
        if n % 13 == 0:
            print('しかも 13 の倍数です')
```

7.3　リストと if を組み合わせよう

```
print('好きな自然数を入れてください')
s = input()
ns = list(s)
print(ns)
if '0' in ns:
    print('０が入っています')
```

リストで使える演算子に in, not in があります。

- 要素 in リストの形で「リストに要素が含まれるならば」を表します。
- 要素 not in リストの形で「リストに要素が含まれないならば」を表します。
- 要素の項目を [] で取り出して条件式に使うこともできます。

例（色々な数を入力して、すべての場合を試してみよう）：

```
print('好きな自然数を入れてください')
s = input()
ns = list(s)
print(ns)
if '0' not in ns:
    print('０が入っていません')
if ns[-1] == '1':
    print('１で終わっています')
```

7.4　「……でないとき」が書けるようになろう

```
print('あなたの年齢を入れてください')
y = int( input() )
if y<18:
    print('未成年です')
else:
    print('成年です')
```

if の条件ブロックが終了した直後に else: を置くことで、「そうでないならば」というブロックを始められます。

- if,else などに限らず、「次の行から新しいブロックを始める」とき、:（コロン）を行の最後に書きます。

- else は、if と同じ高さまで文字を下げます。

書き方：

```
if 条件式:
    条件にあったときのブロック
else:
    条件に合わなかったときのブロック
```

例（色々な数を入力して、すべての場合を試してみよう）：

```
print(' あなたの年齢を入れてください')
y = int( input() )
if y<18:
    print(' 未成年です')
else:
    if y<20:
        print(' 成年ですが、お酒はまだダメです')
    else:
        print(' お酒 OK です')
```

7.5 （ちょっと高度）「そうでなくて〜ならば」が書けるようになろう

```
print(' あなたの年齢を入れてください')
y = int( input() )
if y<18:
    print(' 未成年です')
elif y<20:
    print(' 成年ですが、お酒はまだダメです')
else:
    print(' お酒 OK です')
```

「〜でないときに、〜ならば」という条件が、プログラミングではよく出てきます。これまでの書き方では、条件が多くなると、

```
if xxx:
    ...
else:
    if yyy:
        ...
```

```
    else:
        if zzz:
            ...
        else:
            ...
```

となって、どんどん行の頭が下がっていってしまいます。これを簡単に書くために「そうでなくて〜ならば」を書くための命令が `elif:` です。これを使うと、

```
if xxx:
    ...
elif yyy:
    ...
elif zzz:
    ...
else:
    ...
```

と簡単に書くことができます。（とくにこれを使わなくても、if - else で頑張って書くこともできます）

例：とくになし

ここまでで、基礎的な入出力の命令をマスターしました。TOPSIC のレベル１、AtCoder のレベル A がほとんど解けるはずです。