

Python

情 13-209 坂本 要

指導教員 小林 孝史

1 型

プログラミングで使われるデータには「型」と呼ばれるものがある。例えば 1 や 2 は「数値」で”Hello”というテキストは「文字列」という型である。Python には様々なデータ型が用意されている。今回はその中でも重要な数値、文字列、Bool、リストについて説明する。

1.1 数値型

C や Java では「整数」と「少数」は別物であり、表現できる上限値が決まっている。Java の整数型である `int` は整数を表現する型であるため、小数点は扱うことができない。Python でも正確には整数型や少数型は存在するものの、それらは同じ「数値型」のようなイメージで扱うことができる。数値計算では表 1 の演算子が利用可能である。

表 1: 算術演算子

操作	結果
<code>x + y</code>	x と y の和
<code>x - y</code>	x と y の差
<code>x * y</code>	x と y の積
<code>x / y</code>	x 割る y の商
<code>x % y</code>	x 割る y の余剰
<code>x ** y</code>	x の y 乗

1.2 文字列型

文字列は’シングルクォーテーションか”ダブルクォーテーションで囲む。テキストも数値と同様に「+」を使うことで結合ができる

1.3 Bool 型

Bool は「真偽値」とも呼ばれる「True/False」の 2 値を持つ型である。Bool は表 2 に示した比較演算子と呼ばれる記号で 2 つの値を比較した際に返される。

表 2: 比較演算子

操作	意味
<code>x == y</code>	x と y は等しい
<code>x != y</code>	x と y は等しくない
<code>x < y</code>	x は y より小さい
<code>x > y</code>	x は y より大きい
<code>x <= y</code>	x は y 以下
<code>x >= y</code>	x は y 以上

また、複数の比較演算子の結果を組み合わせる場合、表 3 の論理演算子を用いる。

表 3: 論理演算子

操作	意味
not x	x が偽であれば真
x and y	x も y も真ならば真
x or y	x または y が真ならば真

1.4 リスト

リスト型はデータを「リスト」状に複数個並べたような型である。リストを使わずに 3 人の学生の平均点を求めようとする以下のようなコードがかかる。

```
1 student1 = 68
2 student2 = 81
3 student3 = 49
4 average = (student1 + student2 + student3) / 3
5 print average
```

このコードの場合、学生の人数が 4 人になった場合、修正する箇所が多いことが問題点としてあげられる。このような問題はリストを使うことである程度解消することが出来る。リストは「リストというデータの中に複数のデータを格納できる」という型なので、「学生達の点数」というデータに「具体的な各学生の点数」を格納する。

```
1 results = [68, 81, 49]
2 average = sum(results)/len(results)
3 print average
```

1 行目で「学生たちの点数」というリストを作成している。2 行目にある `sum()` はリスト内にあるデータの合計値を算出する関数、`len()` はリストに格納されるデータの数を返す関数である。2 つめのコードは学生一人ひとりの点数毎に変数を作成していないので、「複数の学生たちの点数の格納」が簡単なことに加え、平均値の算出方法が学生の数に依存していない。リストを使うことで「ひとつのグループに属するデータ」を便利に扱うことが出来る。

```
1 >>> a = []
2 >>> b = [1, 2, 3]
3 >>> c = [1, "2", False]
```

`[]` の中に何も入れない場合はからのリストを作成する。3 行目のようにリストの中に様々なデータが入れるることには注意したい。リストの要素を取り出すにはリスト内の `x` 番目の要素を指定する必要がある。

```
1 リスト名 [要素の番号]
```

指定する順序は 1 からではなく 0 からである。

```
1 >>>b = [1,2,3]
2 >>>print(b[0]) # 1
3 >>>print(b[2]) # 3
4 >>>b[1] = 5
5 >>>print(b[1]) # 5
6 >>> b.append(4) # 末尾への追加
7 >>> print(b)
8 [1, 2, 3, 4]
9 >>> b.insert(1,10) # 1番目の要素に 10を追加
10 >>> print(b)
11 [1, 10, 2, 3, 4]
12 >>> b = [1,2,3]
13 >>> b.remove(1)
14 >>> print(b)
15 [2, 3]
```

2 分岐処理

ifを使うことで条件に応じた分岐処理を行うことが可能である。ifに続く式が真であればインデントされたブロックの処理が実行される。複数の条件を指定する場合 elifを用いる。elifは任意の数繰り返すことが可能である。いずれにも該当しない場合の処理は else に続けて書くが、省略することも出来る。Bool 型が条件判定に利用され、その値が True か False かで実行するプログラムが変わる。

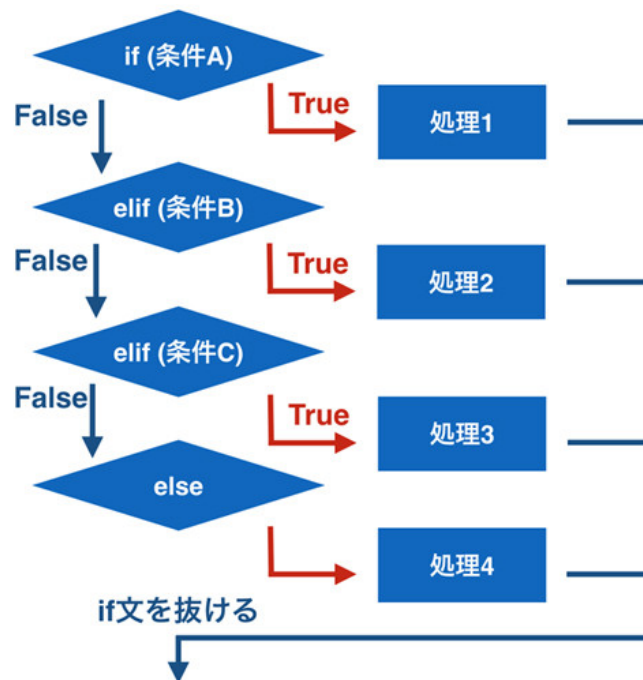


図 1: 条件分岐の仕組み

```
1 if 条件A:
2     処理1 # 条件 A が True の時に実行される処理
3 elif 条件B:
4     処理2 # 条件 A が False で条件 B が True の時に実行される処理
5 elif 条件C:
6     処理3 # 条件 A,B が False で条件 C が True の時に実行される処理
7 else:
8     処理4 # 全ての条件が False の時に実行される処理
```

3 ループ処理

ループ処理は「同じ処理を何度も繰り返す」という処理である。ループ処理の制御構造には for と while の 2 つがある。

3.1 while

while は条件が真である間、インデントされたブロックの処理を繰り返し実行する。

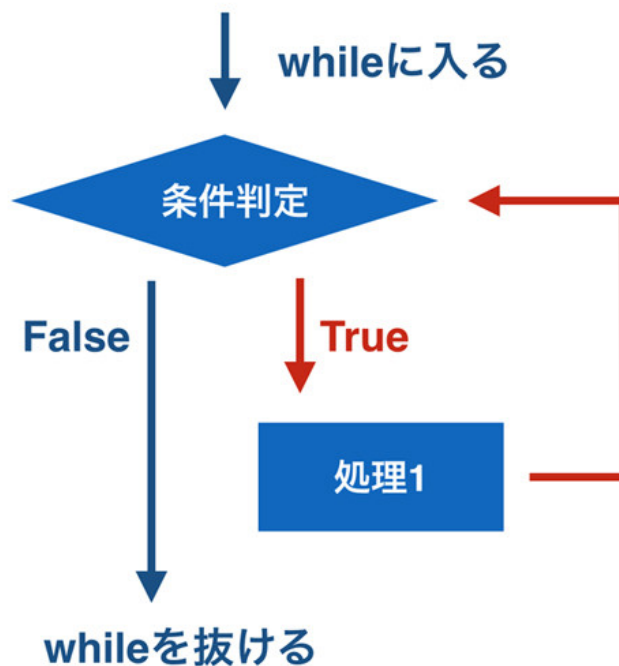


図 2: Python の while 文

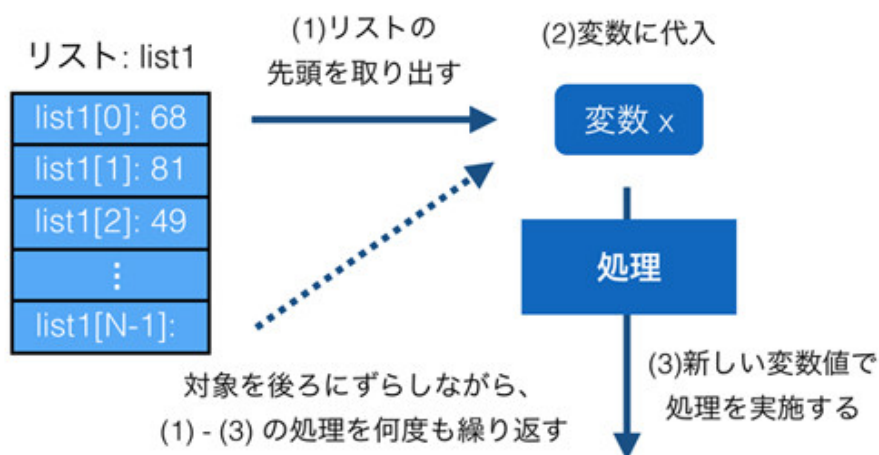
```

1  n = 0
2  while n < 10:
3      print n
4      n += 1

```

3.2 for

for はグループにある要素を取り出し繰り返し処理を実行するといった時に使われる。リストに格納されている要素をすべてチェックするような処理でよく使われる



```
1 for x in range(10):
2     print x # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

3.3 break

while や for を使った繰り返し処理は break を使うことで抜け出すことができる。

```
1 for x in range(10):
2     if x == 5:
3         break
4     print x # 0, 1, 2, 3, 4, 5
```

4 関数

python には様々な組み込み関数が用意されている [1]。これまで使用してきた raw_input() や range(), sum(), len() などがそれである。関数を利用する利点としてプログラムの可読性が向上することがあげられる。例えば絶対値を得ようと思った場合、以下のように if を使って条件分岐させることで実現が可能である。

```
1 x = -5
2 if(x<0):
3     x = x * -1
4 print(x)
```

同じ処理を組み込み関数である abs() を使って書くと以下ようになる

```
1 x = abs(-5)
2 print(x)
```

関数には同じコードを何度も書かなくてすむといったメリットもある。

```
1 x = 5
2 y = -10
3
4 if(x<0):
5     x = x * -1
6 if(y<0):
7     y = y * -1
8
9 print 'abs x > abs y ?'
10 print x > y
```

関数を使って書き直すと

```
1 x = 5
2 y = -10
3
4 x = abs(x)
5 y = abs(y)
6
7 print 'abs x > abs y ?'
8 print x > y
```

関数は自分で作成することも可能である。関数は入力を受け取り、それを加工して出力する。入力と出力はなくてもかまわず、入力がない場合は関数の宣言の引数をなくし、出力が不要な場合は return 文をなくす。

```
1 # 引数がない関数
2 def my_func1():
3     return 0
4
5 # 戻り値がない関数
6 def my_func2(x):
7     x = x * -1
```

関数は宣言した引数に対応する箇所に入力値を入れることで呼び出す。引数がない関数に関しては () に何も入れずに、引数を取る場合は () に値を入力する。

```
1 print my_func1() # 0 と表示される
2 print my_func2(5) # None と表示される
```

引数は複数指定できるが、**return** 文は一度しか実行されない。

```
1 def my_func3(x, y):
2     print "A"
3     if(x > y):
4         return x
5     print "B"
6     return y
7
8 print my_func3(5, 1)
9 # A
10 # 5
11
12 print my_func3(2, 4)
13 # A
14 # B
15 # 4
```

上記のコードでは 2 つの **return** 文が確認できる。x > y の条件が満たされた場合 B が出力されていないことに注目して欲しい。return はいくつあっても構わないが、return されたあとの関数の処理は一切無視される。

4.1 グローバル変数とローカル変数

```
1 x = 5
2 def square(x):
3     return x * x
4 print x
5 square(10)
6 print x
```

参考文献

- [1] 2. 組み込み関数 — Python 2.7.x ドキュメント, <http://docs.python.jp/2/library/functions.html>, 2016 年 3 月 17 日確認。