プログラミング講習

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出来る．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
x + y	x と y の和
x - y	x と y の差
x * y	x と y の積
x / y	x 割る y の商
x % y	x 割る y の余剰
x ** y	x の y 乗
x // y	切り捨て除算

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（+）を使った結合や（*）による繰り返しが出来る．またクォート記号三つで複数行に分けて書くことや [n] で n 番目の文字を取り出すことが可能である （ソースコード 1）．

ソースコード 1: 文字列の操作

```
s = "str"
print s[1] # 「t」が表示される
print 'abc' + 'def' # 「abcdef」が表示される
print "YO" * 5 #「YOYOYOYOYO」が表示される
print """HE
LL
O!
"""
「HE
  LL
  O!」が表示される
```

1.3 Bool 型

Bool は「真偽値」とも呼ばれる「True/False」の二値を持つ型である．Bool は表 2 に示した比較演算子と呼ばれる記号で二つの値を比較した際に返される．

また，複数の比較演算子の結果を組み合わせて評価する場合，表 3 の論理演算子を用いる．

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と以下のようになる（ソースコード 2）．

表 2: 比較演算子

操作	意味
x == y	x と y は等しい
x is y	x と y は等しい
x != y	x と y は等しくない
x is not y	x と y は等しくない
x >y	x は y より小さい
x <y	x は y より大きい
x >= y	x は y 以下
x <= y	x は y 以上
x in y	x の要素が y に含まれる
x not in y	x の要素が y に含まれない

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

ソースコード 2: リストを使わない場合

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

ソースコード 2 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．

ソースコード 3: リストを使った場合

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

ソースコード 3 の 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要素数を返す関数である．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．

ソースコード 4: リストの使い方 1

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

ソースコード 4 の 1 行目のように，[] の中に何も入れない場合は空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られ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．また，指定する順序は 1 からではなく 0 からである（ソースコード 5）．

ソースコード 5: リストの使い方 2

```
リスト名[要素の番号]

>>>b = [1,2,3]
>>>print(b[0]) # 1
>>>print(b[2]) # 3
>>>b[1] = 5
>>>print(b[1]) # 5
>>> b.append(4)    # 末尾へ4を挿入
>>> print(b)
[1, 5, 3, 4]
>>> b.insert(1,10)    # 1番目の要素に10を挿入
>>> print(b)
[1, 10, 5, 3, 4]
>>> b.remove(1)
>>> print(b)
[10, 5, 3, 4]
```

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．インデントとはスペースやタブで字下げすることをいう．また，同じインデント幅で揃えられた部分をブロックという．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（ソースコード 6）．

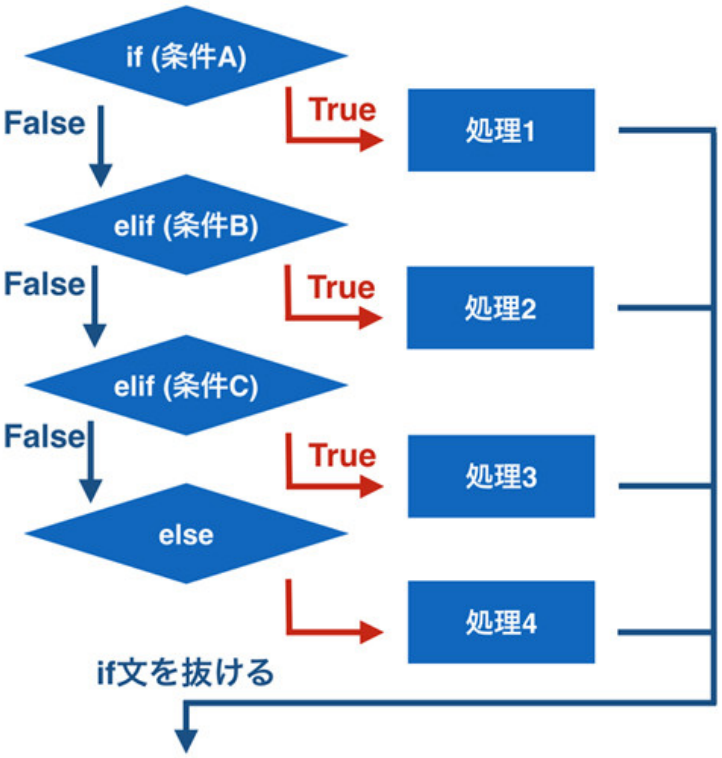

図 1: 条件分岐の仕組み

ソースコード 6: if の使い方

```
if 条件A:
    処理1  # 条件 A が True の時に実行される処理
elif 条件B:
    処理2  # 条件 A が False で条件 B が True の時に実行される処理
elif 条件C:
    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
else:
    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 の二つがある．

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（ソースコード 7）．

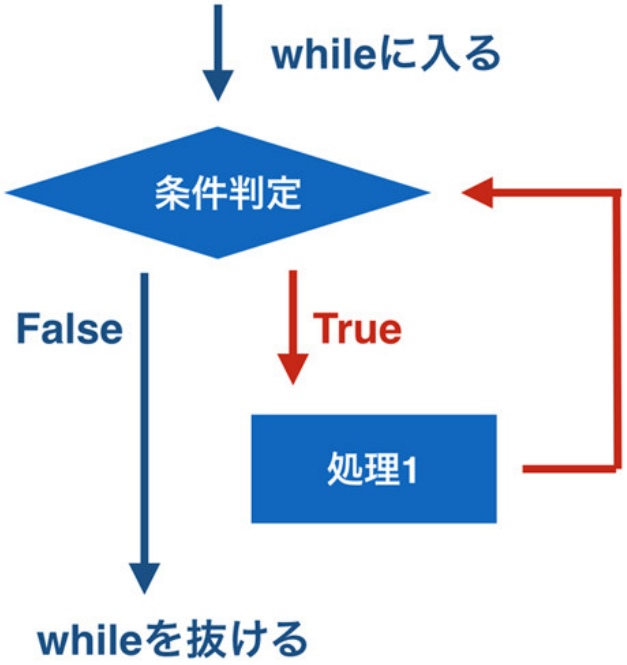

図 2: Python の while 文

ソースコード 7: while の使い方

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

3.2 for

for は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．range 関数を使うことによって指定した回数だけ繰り返し処理を行うことも可能である（ソースコード 8）．

ソースコード 8: for の使い方

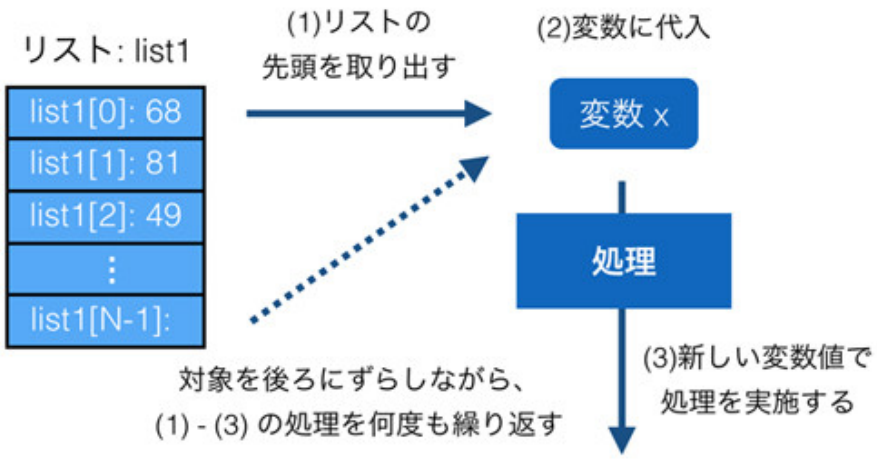

図 3: Python の for 文

```
n = [1,2,3]
for x in n:
  print x    # 1, 2, 3
for n in range(10):
  print n # 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（ソースコード 9）.

ソースコード 9: break

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

4 関数

python には様々な組み込み関数が用意されている．これまで使用してきた range() や sum(), len() などがそれである．関数を使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（ソースコード 10）.

ソースコード 10: 絶対値の計算をするコード

```
x = -5
if(x<0):
    x = -x
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のようになる（ソースコード 11）.

ソースコード 11: abs()

```
x = abs(-5)
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（ソースコード 12）.

ソースコード 12: 絶対値の検索をするコードを使った場合

```
x = 5
y = -10

if(x<0):
    x = x * -1
if(y<0):
    y = y * -1

print 'abs x > abs y ?'
print x > y
```

関数を使って書き直すとこのようになる（ソースコード 13）.

ソースコード 13: abs() を使った場合

```
x = 5
y = -10

x = abs(x)
y = abs(y)

print 'abs x > abs y ?'
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．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．(ソースコード 14) .

ソースコード 14: 関数の作成

```
# 引数がない関数
def my_func1():
  return 0

# 返り値がない関数
def my_func2(x):
  x = x * -1

print my_func1()  # 0 と表示される
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．

ソースコード 15: return の動作

```
def my_func3(x, y):
    print "A"
    if(x > y):
        return x
    print "B"
    return y

print my_func3(5,1)
# A
# 5

print my_func3(2,4)
# A
# B
# 4
```

上記のソースコード 15 では二つの 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
```