プログラミング講習

1 プログラミングとは

プログラミングとは，人間の意図した処理を行うようにコンピュータに指示を与える行為である．現在，身近にある様々なものはプログラムに従い動作している．コンピュータやスマートフォンはもちろん，炊飯器や洗濯機，自動車や信号機などを動かしているのもプログラムである．コンピュータは人間と違い，指示された通りのことしかできない．プログラミングはコンピュータに対して,「こうだったら，こうする」といった振る舞いのマニュアルを作成する行為であると言える．

2 プログラミングを学習するメリット

プログラミングを学ぶことで得られるメリットとして，作業の「効率化」が挙げられる．コンピュータは指示されたことしか実行できない代わりに，人間より高速かつ正確に処理を行うことができる．市販のコンピュータでさえ，1 秒間に 10 億回の計算を行うことができる．プログラムを書くことができれば，人間が時間をかけてやっている面倒な作業を，コンピュータに任せることができるというメリットがある．

3 Python

3.1 Python とは

Python とは，簡潔で読みやすい文法が特徴的なプログラミング言語の一つである．プログラムの可読性が高くなるように設計されているので，C や Java などの言語に比べてより少ないコード行数でプログラムを表現することができる．Google や NASA の内部で利用されているという実績があり，本格的なプログラミングにも対応できる言語である．

3.2 インタープリタ

プログラムをコンピュータが理解できるマシン語（バイナリ）に変換するには，インタープリタとコンパイラという 2 種類の方式がある．Python はプログラムのソースコードを 1 行読み込むたびにマシン語に変換し実行するインタープリタ型の言語である．C 言語や Java はソースコード全体をマシン語に翻訳してから実行するコンパイラ型の言語である．インタープリタ型の言語はコンパイラ型の言語に比べて実行が遅いという欠点がある．その反面，1 行ずつ処理を確認することができるので，プログラミング学習に向いていると言える．

3.3 対話モード

Python には，対話的にプログラムを入力，実行できる対話モードがある．対話モードを利用することで，1 行ずつソースコードを打ち込み，その都度実行結果を確認することができる．

対話モードを利用するためには，Windows の場合はコマンドプロンプトを，Mac の場合はターミナルを起動し，“python” と入力することで起動することができる．

ソースコード 1: 対話モード

```
1 $ python #コマンドプロンプトorターミナルでpythonと入力
2 Python 2.7.5 (default, Nov 20 2015, 02:00:19) #Pythonのバージョン情報など
3 [GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> #ユーザの入力を待ち受けるプロンプト
```

対話モードを起動した際のシェルの状態をソースコード 1 に示す．Python を対話モードで起動すると，バージョン情報などが表示された後に「>>>」という文字列が表示される．これはプロンプトと呼ばれるもので，Python がユーザの入力を待ち受けている状態であることを表す．

対話モードを終了したいときには，プロンプトが表示されている状態で “exit()” と入力するか，“Ctrl-D” キーを打ち込むことで終了できる．

4 Hello World

大抵のプログラミング言語で最初の例題とされている Hello World について解説する．

ソースコード 2: C 言語の Hello World

```c
#include <stdio.h>

int main(void)
{
    printf("Hello world!");
    return 0;
}
```

ソースコード 3: Python の Hello World

```
print "Hello World"
```

C 言語の Hello World（ソースコード 2）と Python の Hello World（ソースコード 3）を比較すると，コード量の差がわかる．二つのプログラムは同じ処理をしている．C 言語は Python に比べて処理速度が早いというメリットがあるが，文法的な制約が多く，簡単な処理でもコード量が多くなってしまう．Python は文法的な制約が少ないので，簡潔に処理を書くことができる．

Python プログラムの “print” は，半角スペースの後に続く文字列を出力するという命令である．Python では文字列はダブルクオートかシングルクオートで囲むことで表現する．

5 変数と代入

変数とはプログラムの中で扱われるデータを記憶し，必要なときに利用するために，データに固有の名前を与えたものである．例えば，ある計算の結果をまた別の計算に再利用したい時などに用いる事ができる．変数に実際のデータを関連付けることを代入という．

ソースコード 4: 変数と代入

```
>>> x = 3 #変数xに3を代入
>>> x + 5 #変数xには3が代入されているので，3+5となる
8 #3+5の計算結果が出力される
```

変数に使うことができる文字の種類は，アルファベット，数字，アンダースコアだけである．また，変数名はアルファベットの大文字と小文字が区別される．そのため xyz と XYZ は違う変数となる．さらに数字を変数名の先頭に使うことはできない．

この他，予約語と呼ばれる特別な単語は，変数として利用できない．予約語の例を表 1 に示す．

表 1: 予約語

and	as	assert	break	class
continue	def	del	elif	else
except	exec	finally	for	from
global	if	import	in	is
lambda	not	or	pass	print
raise	return	try	while	with
yield				

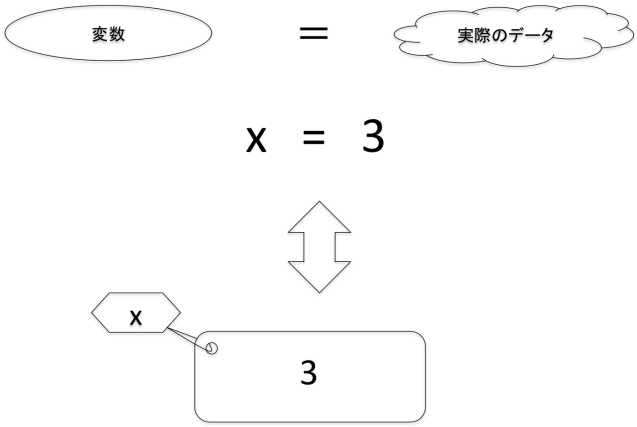

図 1: 変数と代入のイメージ

6 標準入力

世の中に存在するプログラムはすべて，なんらかのデータの入力を受け，そのデータを処理し，結果を出力する．例えば，電卓はユーザからの数値・演算子の入力，計算処理，結果の出力という流れで出来ている．

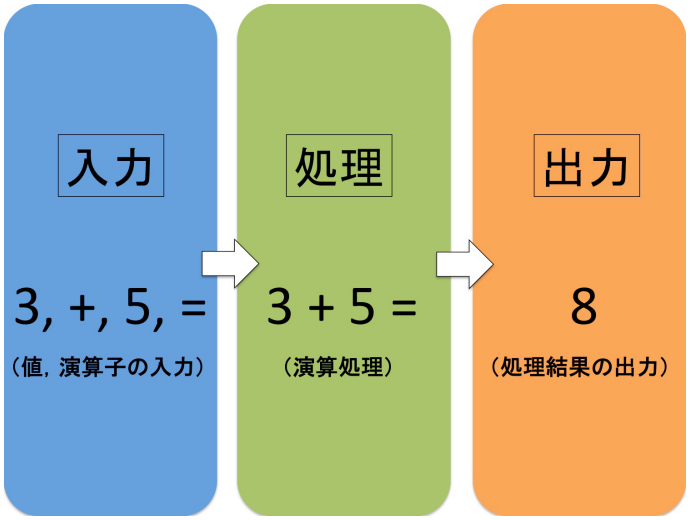

図 2: 電卓の入出力のイメージ

入力はユーザがキーボードやマウスから行う場合もあれば，他のプログラムや，ファイルから読み込むこともある．プログラムに対して，キーボードから値を入力する仕組みを標準入力という．Python における標準入力の方法をソースコード 5 に示す．

ソースコード 5: 標準入力

```
1 >>> input() #入力された値を式として扱う標準入力
2 3+5 #ユーザが入力した値
3 8 #入力された式を演算した結果
4
5 >>> raw\_input #入力された値を文字列として扱う標準入力
6 3+5 #ユーザが入力した値
7 '3+5' #入力された値を文字列として出力
```

1 行目の “raw_input” で，コマンドラインからの入力を取得している．Python の標準入力は 2 種類存在し，“raw_input” は文字列の入力を受け付け，と “input” は式の入力を受け付ける．

7 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．

7.1 数値型

C や Java では「整数」と「小数」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小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 2 の演算子が利用可能である．

表 2: 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

7.2 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 6）．

ソースコード 6: 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
```

7.3 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 3 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 4 の論理演算子を用いる．

表 3: 比較演算子

操作	意味
x == y	x と y は等しい
x is y	x と y は同一のオブジェクト
x != y	x と y は等しくない
x is not y	x と y は同一のオブジェクトではない
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 4: 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

7.4 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 7）.

ソースコード 7: リストを使わない場合

```
>>> student1 = 68
>>> student2 = 81
>>> student3 = 49
>>> average = (student1 + student2 + student3) / 3
>>> print average
```

ソースコード 7 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 8: リストを使った場合

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

ソースコード 8 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 9: リストの使い方 1

```
>>> a = []
```

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

ソースコード 9 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 10）．

ソースコード 10: 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]
```

8 ブロックとインデント

ブロックとは，複数個の文などのコードのまとまりのことである．if 文や for 文，関数定義やクラス定義などを利用する際は，ブロックの中に処理を記述することになる．

C 言語や Java が中括弧を用いて処理のブロックを表すのに対して，Python はインデント（行頭の空白文字の数）で処理のブロックを表す．コロンで終わる行がブロックの始まりを表し、それ以下のインデントが同じ行を同じブロックと見なす．インデントの空白文字の数は任意であるが，一般的に四つが推奨されている．

ソースコード 11: インデント

```
>>> x = 3
>>> if x == 3: #変数xが3である場合，if文以下のブロックを実行する
...     print "AAA" #インデント（スペース4つ）の後に，命令を記述
...     print "BBB"
... #if文を抜けるために，Enter
AAA #if文の条件が真だったため，if文以下のブロックが実行される
BBB

>>> if x == 5:
...     print "AAA"
...     print "BBB"
...
>>> #if文の条件が偽だったため，if文以下のブロックは実行されない
```

9 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 12）．

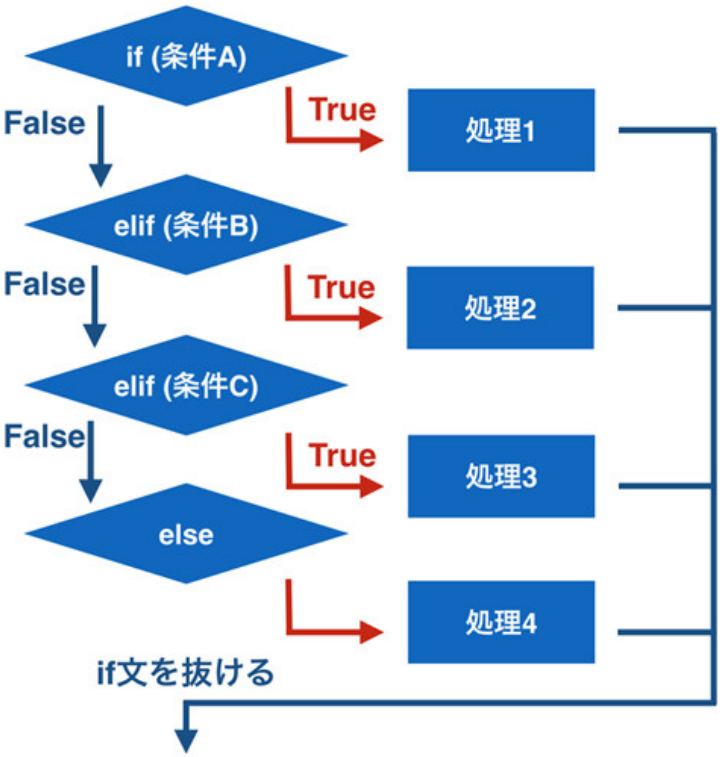

図 3: 条件分岐の仕組み

ソースコード 12: 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
```

10 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．

10.1 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 13）．

ソースコード 13: while の使い方

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

10.2 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 14）．

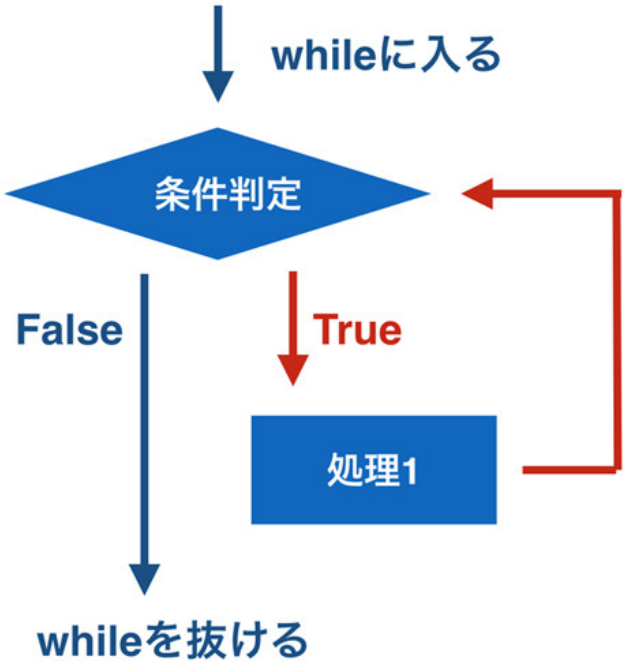

図 4: Python の while 文

ソースコード 14: for の使い方

```
>>> n = [1,2,3]
>>> for x in n:
... print x
...
1
2
3
>>> for m in range(10):
...  print m
...
0
1
2
3
4
5
6
7
8
9
```

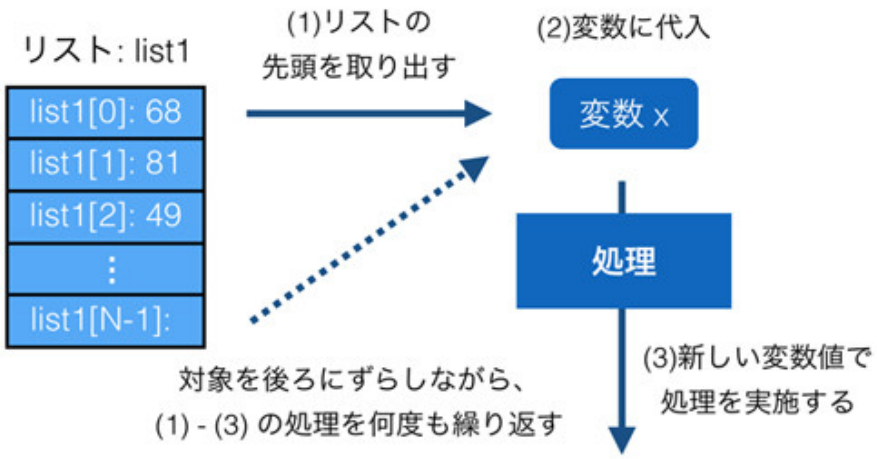

図 5: Python の for 文

10.3 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 15）.

ソースコード 15: break

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

11 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 16）.

ソースコード 16: 絶対値に変換するコード

```
>>> x = -5
>>> if(x<0):
...     x = -x
...
>>> print x
5
```

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 17）.

ソースコード 17: abs()

```
>>> x = abs(-5)
>>> print(x)
5
```

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 18）.

ソースコード 18: 絶対値に変換するコードを使った場合

```
>>> x = 5
>>> y = -10
>>> if(x<0):
...    x = -x
...
>>> if(y<0):
...    y = -y
...
>>> print x > y
False
```

関数を使って書き直すとこのようになる（ソースコード 19）.

ソースコード 19: abs() を使った場合

```
>>> x = 5
>>> y = -10
>>> x = abs(x)
>>> y = abs(y)
>>> print x > y
False
```

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 20) .

ソースコード 20: 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
...
>>> print my_func1()
0
>>> print my_func2(5)
None
```

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 21: 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 21 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．

12 スコープ

スコープとは，ある変数や関数が特定の名前で参照される範囲のことである．ある範囲の外に置いた変数等は，通常，その名前だけでは参照できない．このときこれらの変数はスコープ外であると言われる．プログラミングでは，予期しない誤作動を避けるためにも，それぞれの作業段階で必要のない名前はできるだけ参照されないようにすることが望ましい．

Python では if 文や for 文などの制御構造はスコープを作らない．つまり，if 文や for 文の中で宣言された変数は，if 文や for 文のブロックの外からも参照することができる．しかし，関数定義とクラス定義では新しいスコープが作られる．つまり，関数定義の内側と外側に同じ名前の変数が存在しても，両者は区別される．

ソースコード 22: スコープ

```
1 >>> x = 5 #一番外側のスコープで変数xに5を代入
2 >>> def scope_test(): #scope_test関数の定義
3 ...     x = 10 #関数定義のブロック内でxに10を代入
4 ...     print x #変数xの値を出力
5 ... #関数定義のブロックを抜けるためにEnter
6 >>> print x #変数xの値を出力
7 5
8 >>> scope_test() #scope_test関数の実行
9 10
```