Python 講座

【第3回】2015.05.19

きょうの話

7 ファイル操作

8 対話ループ

ファイル操作

ファイル操作とは何ぞや？

- 文字通りファイルの中身の操作を行う。
- とはいってもファイルを操作してword等のプログラムで直接開くのではなく、ファイルの中身（書かれている文字の読み込み、逆にファイルに書き込むなど）を直接操作する。Cygwinとかのコンソールを使ったことがある人はそっちの感覚でいるとわかりやすい…かも

open()

ファイル操作をするうえで必須となるメソッド。()の中には引数が入り、第一引数には開きたいファイル名が、第二引数には開くときのモードがそれぞれ入る。このときファイル名は拡張子も含めて指定しないと同名の別ファイルとなるので注意

Ex)test.txtを読み込み専用で開きたい場合

```
f=open(“test.txt”, ”r”)
```

(fは変数であるため、他の文字でも可)

open()のモードについて

- 第一引数（参照したいファイル）に対してどんなコマンドを受け付ける状態にするかを決めるのがモード。だいたい下の表を覚えておけば何とかなる。

r	読み込み専用。書き込みはできない。 ファイルが存在しないときはエラーになる。
w	書き込み専用。読み込みはできない。 ファイルが存在していない場合は新規作成。
a	追加書き込み専用。読み込みはできない。 ファイルがない場合は新規作成。
r+	読み書き両用。 ファイルがない場合はエラー。
w+	読み書き両用。 ファイルがある場合は「w」と同じ処理。
a+	追記・読み書き両用。 ファイルがない場合は新規作成。

※読み書きしたいファイルの場所について

できるだけpythonのexeファイルがあるところ（C:¥python34の中、C:¥Software¥Python¥Pythonの中等）と同階層に作成する。

一応他の場所、例えばデスクトップから呼び出すこともできるが、本講座では時間の都合上説明しない。興味がある人はosモジュールを検索してみよう。

close()

開いて使い終えたら必ず閉じる。

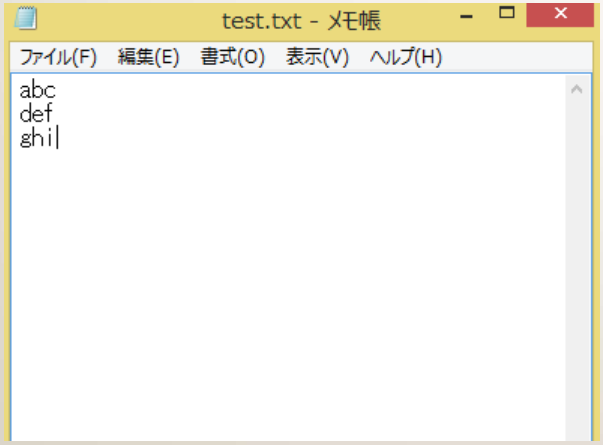
ここまでがファイル操作の一連の流れ。

この流れは大事なので必ず覚えること。これをしないと他のモードで開くことができないのはもちろん、他の作業に移ろうとしたときにエラーが出て警告を出してくる可能性も。

```
f=open(“test.txt”, “r”)   //読み込みモード
  f.read()                          //これは受け付ける
  f.write(“python”)        //これは受け付けない
f.close()
f=open(“test.txt”, “w”)   //ここで書き込みモードへ
  f.write(“python”)        //これは受け付ける
  f.read()                          //これは受け付けない
```

ファイルの読み込み

あらかじめpythonのexeファイルと同階層にabc(改行)def(改行)ghiという文字列を保持したtest.txtというファイルが存在するとする。




```
f=open(“test.txt”, “r”)
```

readline()

ファイルの改行までを１行読み込み、文字列として返す。

open()を繰り返し呼ぶことで１行ずつ読み込みを進めていける。

```
>>> f.readline()
'abc¥n'
>>> f.readline()
'def¥n'
>>> f.readline()
'ghi'
>>> f.readline()
''
```

read()

ファイルの初めから終わりまでをひとつの文字列として返す。読み込んだ後はポインタが一番最後まで行っているので、再度readで読み込もうとしても実行できない。

```
>>> f.read()
'abc¥ndef¥nghi'
```

Ex.ポインタって？

言葉では説明しづらいので画像で。
read()で読み込む前は右上の図。
一番左上にある｜が
read()で読み込むと右下の図のようになる
ここでもう一度readを実行すると｜は
動かずに右下の時点から読み込もうとする
そのため戻したりしないと文字列が出ない

readlines()

ファイルを複数行まとめて読み込み、リストとして返す。一行を読み込んだとき、改行の文字列は直前の列に保持される。

```
>>> f.readlines()
['abc¥n', 'def¥n', 'ghi']
```

ちなみに一部分が空欄だったりしたとき、例えばtest.txtの”def”部分だけを消してreadlinesを実行すると、改行の文字列のみ返される。

```
>>> f.readlines()
['abc¥n', '¥n', 'ghi']
```

seek()

一度read等で読み込むと読み込んだところまでポインタは移動してしまう。最後まで読み込んでしまうと再度読み込むには一度閉じるかこのseekメソッドを使って戻すことができる。が、これを説明しようとするとややこしいので、今はseek(0)を使うとポインタが簡単に頭に戻ってくる、ということだけで覚えていてほしい。いわゆるおまじない扱いで。

```
>>> f.readline()
'abc¥n'
>>> f.readline()
'def¥n'
>>> f.readline()
'ghi'
>>> f.readline()
''
```

```
>>>f.seek(0)
0
>>> f.readlines()
['abc¥n', 'def¥n', 'ghi']
```

ファイルの書き込み

ファイルの書き込みができるモードであることが前提。できないモードで書き込みのメソッドを実行しても エラーが吐かれる。ここでは確認をすぐ行えるよう、読み書き両用を使う。

```
f=open(“python.txt”, “w+”)
```

write("文字列")

引数の文字列を書き込む。改行なし。一度書き込むとポインタが移動しているのでseekで元に戻してからreadを実行する。

```
>>> f.write("python")
6
>>> f.read()
''
>>> f.seek(0)
0
>>> f.read()
'python'
```

writelines(シーケンス)

シーケンス部分（リスト、タプルなど）の要素を書き込む。改行なし。

```
f.writelines( ["1", "2", "3", "4", "5"] )
>>> f.read()
''
>>> f.seek(0)
0
>>> f.read()
'12345'
```

入力する際、改行を保持したい場合

基本的に入力部分の文字列中に“¥n”を入れることで、そのまま保持してファイルに書き込める。そのほかにも小技としてクォーテーション三つで囲むことで入力中の改行をそのまま保持できる。

```
>>> s="""python
java
C#"""
>>> print(s)
python
java
C#
>>> f.write(s)
14
```

```
>>> f.read()
'python¥njava¥nC#‘
>>> f.readline()
'python¥n'
>>> f.readline()
'java¥n'
>>> f.readline()
'C#'
>>> f.readline()
''
```

```
>>> f.readlines()
['python¥n', 'java¥n', 'C#']
>>>
```

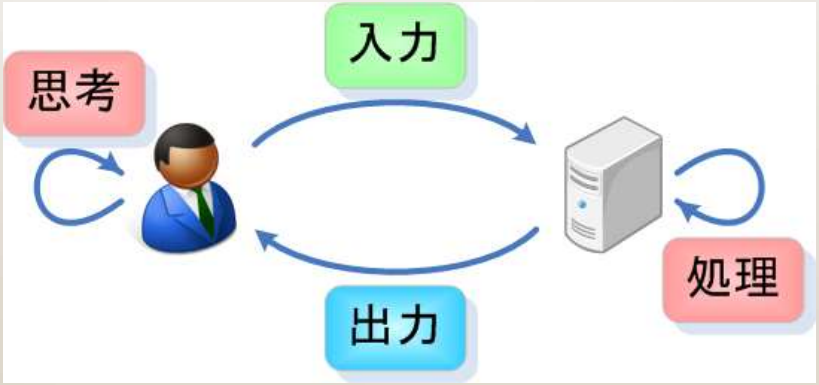
対話ループ

繰り返しの応用的な使い方のお話

対話ループって？

- 今までのプログラムは入力を一方的に受け付けるプログラム

- 人間とコンピュータが対話しながら処理を進めるプログラム



先頭から最後まで・for

（再掲）

- for <変数名> in <要素の集まり>:
  space (繰り返す処理)

```
members = ['ムサシ', 'コジロウ', 'ニャース'] ⏎

for x in members: ⏎
    print(x) ⏎
```

```
ムサシ
コジロウ
ニャース
```

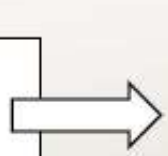
変数 x に 1 個放り込み、print して、
次の要素を放り込み、print して、
また次の要素を放り込み…… の繰り返し。

連番を詰め込んで返す・range

（再掲）

- range(★, ☆)
  - range 関数。★ から ☆-1 までの**整数を**順に詰め込んだリスト**っぽいもの**を返す。
  - ★ …… 開始値
  - ☆ …… 終了値 +1

```
for x in range(0, 3): ⏎
    print(x) ⏎
```



```
0
1
2
```

0、1、2 の詰まったリストのようなものを for に指定してあげた例。
x に 0 を放り込んで print 、次は 1 を、……という具合。
☆ = (終了値 + 1) = 3 なので、格納されるのは 2 まで。

条件を満たす限り・while

（再掲）

- while <*条件*>:
  (繰り返す処理)

```
name = ‘’
while name == ‘’ :      # 条件「name が空」⏎
    name = input(‘きみの なまえは? ’) ⏎
    print(‘ふむ……’) ⏎
print(name + ‘ と いうんだな!’) ⏎
```

```
きみの なまえは? ⏎
ふむ……
きみの なまえは? わしにしね⏎
ふむ……
わしにしね と いうんだな!
```

name 変数が空であれば、
ループに突入＆継続する。

ループ中に使える特殊な処理

1. continue … ループの先頭に戻る

2. break…ループを終了する

3. else…breakが呼び出されたときに使用

※こんなものあるんだ、へー程度の認識でOK!

continueって何？

continue文は...

- 繰り返し処理中に出てくる
- continue以下のループ内の処理を無視して繰り返しの条件に戻る

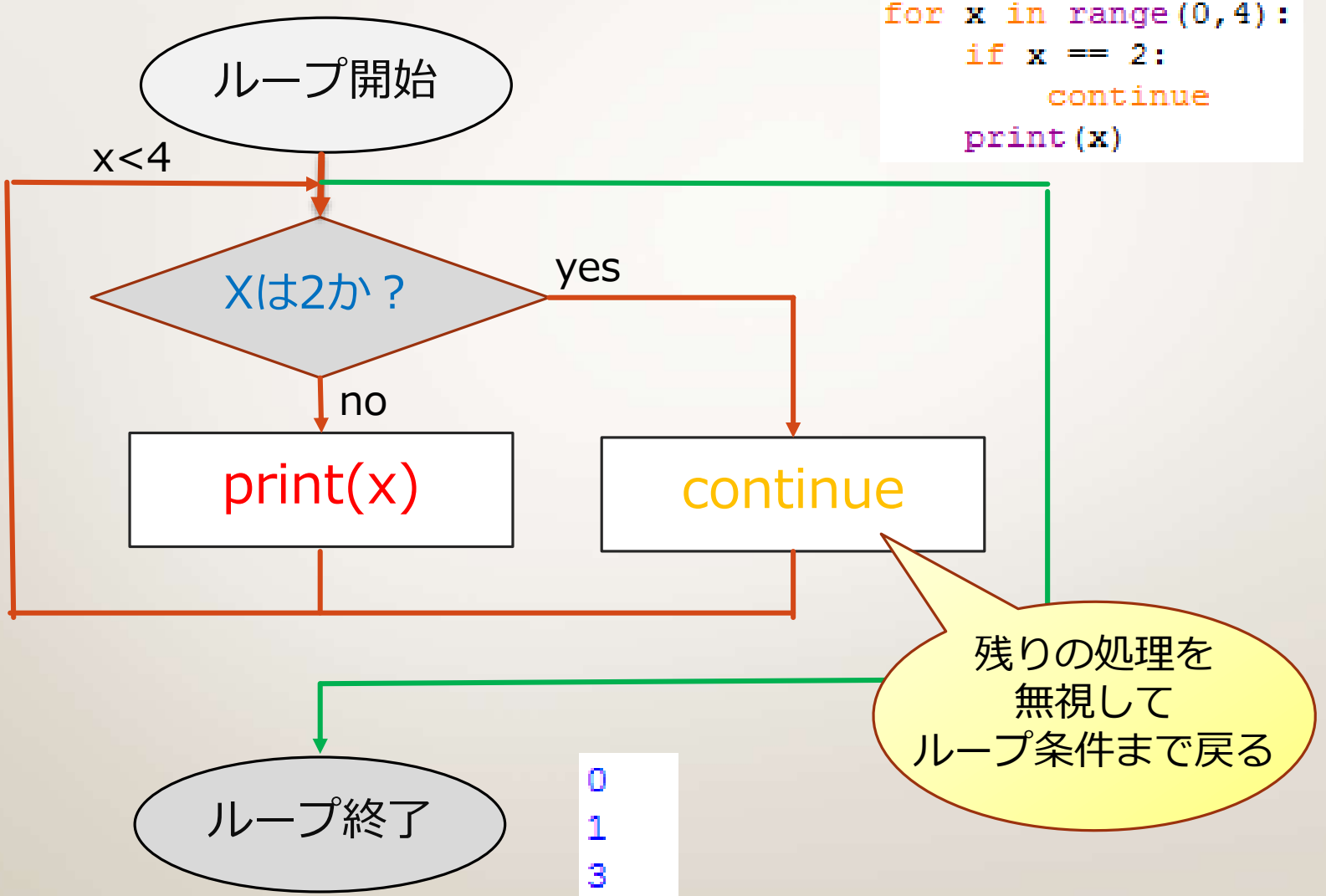
```
for x in range(0,4):
    if x == 2:
        continue
    print(x)
```

```
0
1
3
```

xが2の時だけcontinueより下にあるループ内の処理を無視次の条件で繰り返し続ける

continue

```
for x in range(0,4):
    if x == 2:
        continue
    print(x)
```




```
0
1
3
```

breakって何？

break文は...

- 繰り返し処理中に出てくる
- breakがきたところで残りの繰り返しの処理を全て無視してループを抜ける

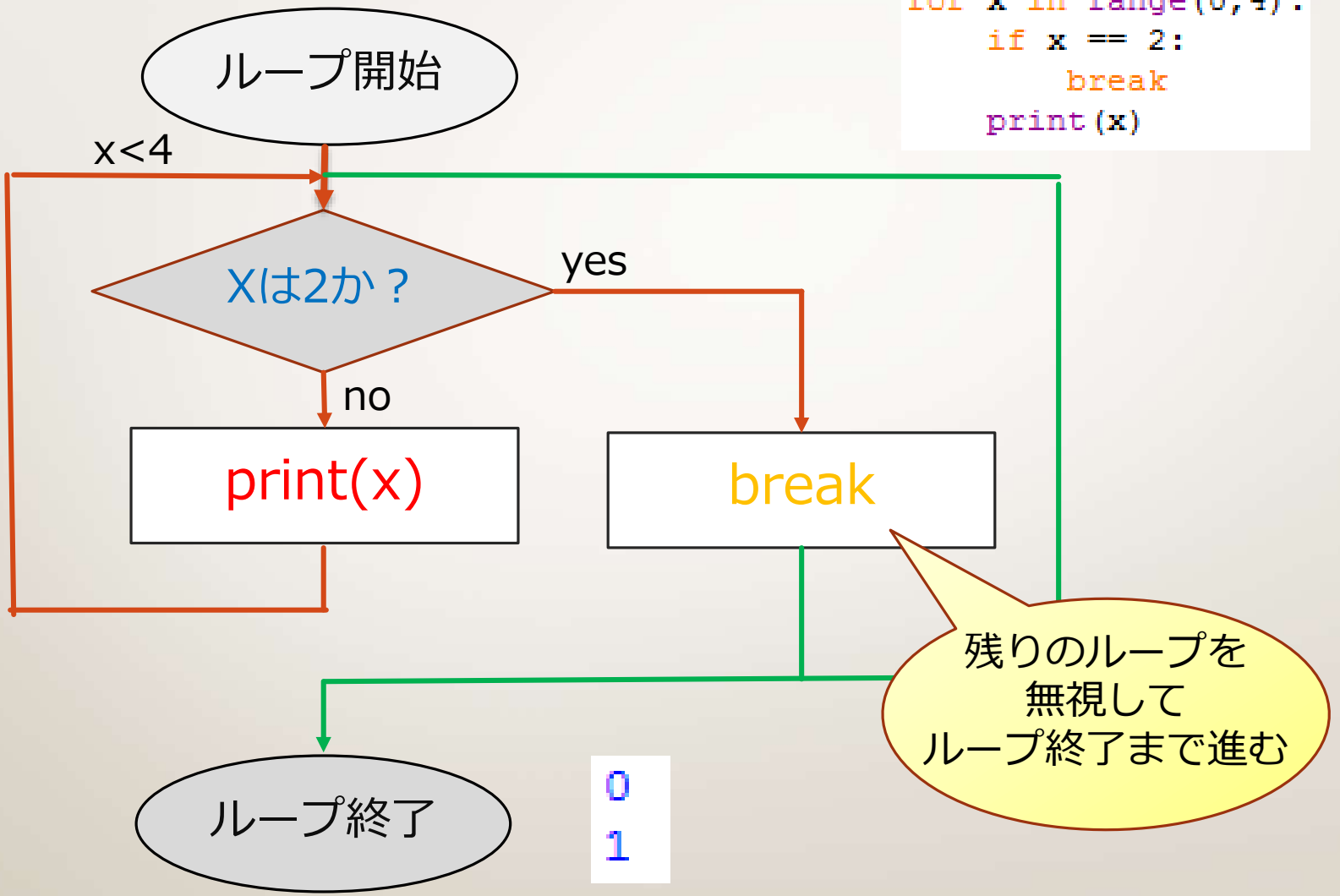
```
for x in range(0,4):
    if x == 2:
        break
    print(x)
```

```
0
1
```

xが2の時にbreakするのでxが2の時のprint()と3の処理を全て無視し繰り返しを終了する

break

```
for x in range(0,4):
    if x == 2:
        break
    print(x)
```




```
0
1
```

つまり…

- continue…呼び出されたらループの一番上に戻る
  （ループを終了するわけではない）

- break…呼び出されたらループの一番下の次の行に進む
  （ループは強制終了）

実際に対話ループする
プログラムを見てみましょう

対話プログラム（例１①）

```
msg=['晴れ','曇り','雨','雪']

while True:
    x=int(input('何番の天気にしますか？'))
    print('今日の天気は{}'.format(msg[x]))
```

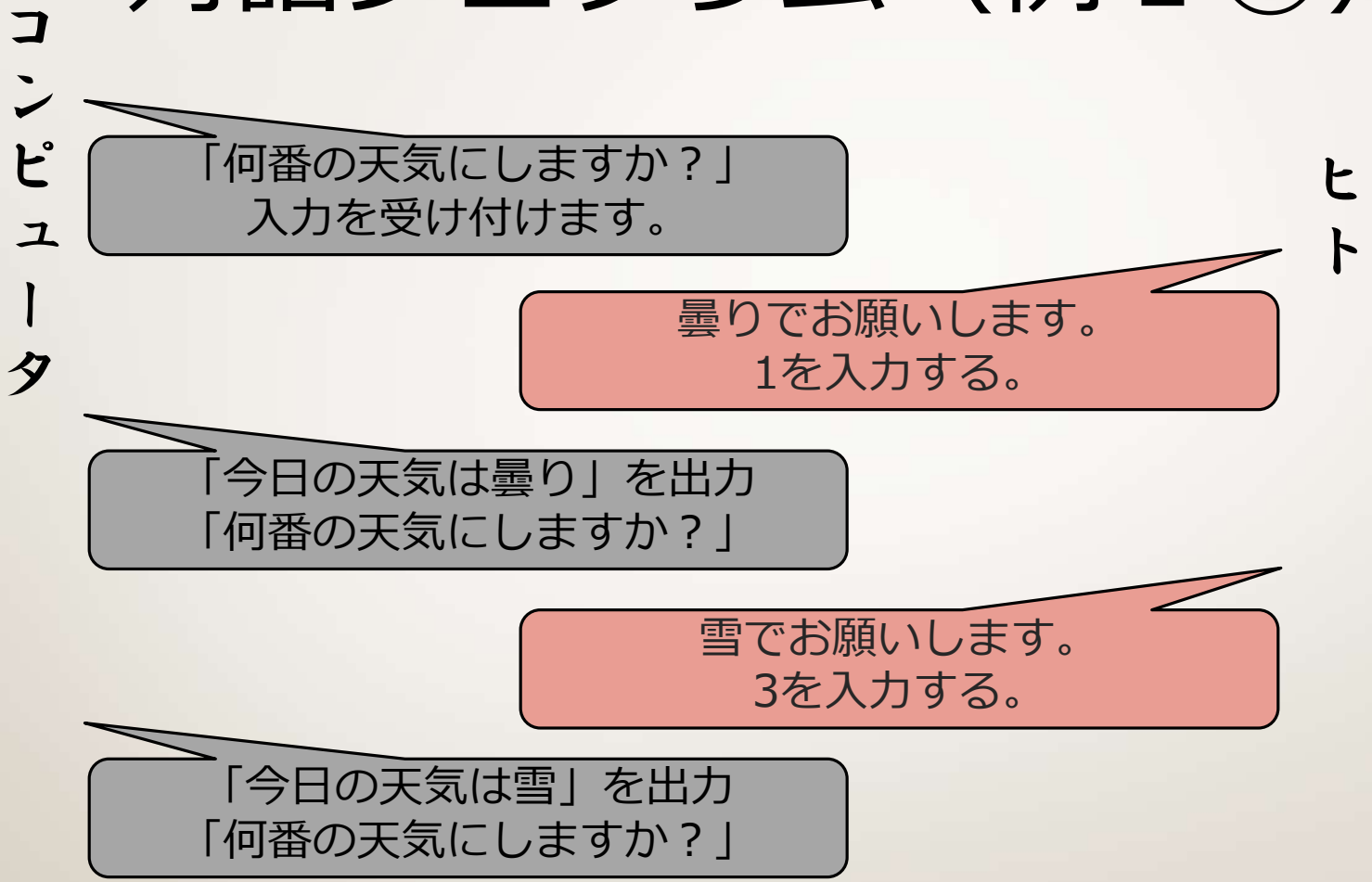
条件に「True」を設定すると無条件でループする

```
何番の天気にしますか？0
今日の天気は晴れ
何番の天気にしますか？1
今日の天気は曇り
何番の天気にしますか？2
今日の天気は雨
何番の天気にしますか？3
今日の天気は雪
何番の天気にしますか？1
今日の天気は曇り
何番の天気にしますか？
```

「何番の天気にしますか？」という文が表示され、それに対し入力をすると対応する文章が表示される。
↑の処理を繰り返す。

※ここでの入力は0~3までの数字のみとする

対話プログラム（例１②）



- ループ

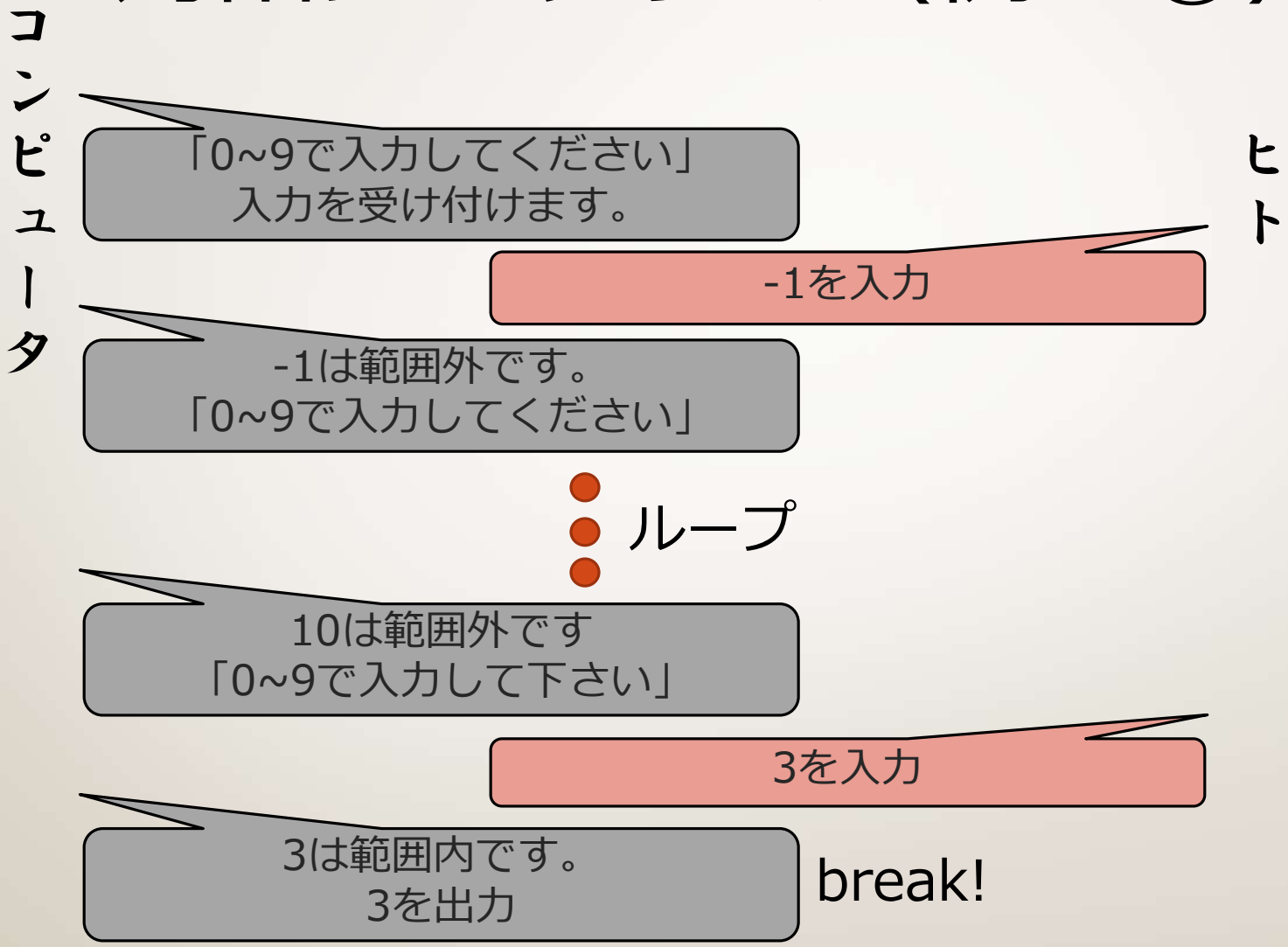
対話プログラム（例２①）

```
while True:
    x=int(input('0~9の間の数字を入力してください'))
    if x >= 0 and x <= 9:
        break
    else:
        print('{}は適切な入力ではありません'.format(x))
print(x)
```

```
0~9の間の数字を入力してください-1
-1は適切な入力ではありません
0~9の間の数字を入力してください10
10は適切な入力ではありません
0~9の間の数字を入力してください3
3
```

0~9の範囲の数字が入力されない限り入力をさせるループを続ける。
0~9の範囲の数字が入力されるとループを抜け、その値を表示する。

対話プログラム（例２②）


コンピュータ
ヒト
「0~9で入力してください」
入力を受け付けます。
-1を入力
-1は範囲外です。
「0~9で入力してください」
ループ
10は範囲外です
「0~9で入力して下さい」
3を入力
3は範囲内です。
3を出力
break!

演習 #7~9

演習 #7

自分の好きな名前のファイルをtxtファイルで作成し、
中に好きな文字列を書き込み、読みだしてください。
エラーがなければ成功です。

演習 #8

- ユーザが数字の入力を行い、それまでの数字の合計を入力一回ごとに「これまでの合計は～」というように表示するプログラムを作成しなさい。
- 入力は整数とします
- 余力のある人は入力の部分を関数で作ってみてね！

```
整数を入力してください：1
これまでの合計は1
整数を入力してください：-2
これまでの合計は-1
整数を入力してください：3
これまでの合計は2
整数を入力してください：4
これまでの合計は6
整数を入力してください：5
これまでの合計は11
整数を入力してください：
```

演習 #9

- 以下の結果が得られるように次のプログラムを書き直し、正しく動作するようにしなさい。

```
def getInput():
    while True:
        a=int(input('0~100の範囲で整数を入力してください：'))
        if 0 <= a and a <= 100:
            continue
            print('正しい入力を受け付けました')
        else:
            print('正しい入力ではありません')

score=getInput()
print('入力された点数は{}'.format(score))
```

```
0~100の範囲で整数を入力してください：101
正しい入力ではありません
0~100の範囲で整数を入力してください：-3
正しい入力ではありません
0~100の範囲で整数を入力してください：95
正しい入力を受け付けました
入力された点数は95
```