プログラミング演習I
– 第8回 –
文字列＋復習（配列，ハッシュ，条件判断，繰り返し）

1　はじめに

第 8 回の演習の目標は,

- 文字列についてのさまざまな操作を理解する.
- 配列，ハッシュ，条件判断，繰り返しについて理解を深める.

です.

2　文字列あれこれ

教科書

p.217- 文字列 (String) クラス

演習 8-1
次の 2 行の出力の違いを確認しよう．2 行目と同じ出力になるように 1 行目を変更するにはどうすればよいだろう．ただし，文字列は”(ダブルクォート) で括ることとする.

```
------------------------------------------------
print("hoge\n")
print('hoge\n')
------------------------------------------------
```

演習 8-2
キーボードから入力した回数だけ,「カッコー」と鳴く（表示する）プログラムを作ろう.

演習 8-3
キーボードからいろいろな文字列を入力して，出力を確かめよう．特に，日本語の場合について考えてみよう.

```
------------------------------------------------
line = gets.chomp
print(line.length, "\n")
print(line.size, "\n")
------------------------------------------------
```

演習 8-4
文字列がつながることを確かめよう.

```
-----------------------------------------------
who = "私が"
what = "宙返りを"
where = "実習室で"
why = "寝坊したので"
how = "華麗に"


phrase1 = who + what + "した"
print(phrase1, ". \n")

phrase2 = who + why + where + how + what + "した"
print(phrase2, ". \n")
-----------------------------------------------
```

演習 8-5
文字列と配列は同じ？以下の出力を確認してみよう.

```
-----------------------------------------------
string = "abcdef"
array = ["a", "b", "c", "d", "e", "f"]
print(string[0], "\n")
print(string[1], "\n")
print(string[0].chr, "\n")
print(string[1].chr, "\n")
print(string[1,2], "\n")
print(string[1,3], "\n")
print(array[0], array[1], array[2], "\n")
-----------------------------------------------
```

演習 8-6
以下の出力を確認しよう.

```
-----------------------------------------------
print("aaa" == "baa")
print("aaa" == "aaa")
print("aaa" != "bbb")
print("aaa" < "bbb")
-----------------------------------------------
```

演習 8-7
キーボードから入力した自分の姓名が正しいかを判定するプログラムを作ろう.

演習 8-8
chomp と chop の違いを確かめてみよう.

```
-----------------------------------------------
line = gets.chomp
print(line, "\n")
print(line.chomp, "\n")
print(line.chop, "\n")
-----------------------------------------------
```

演習 8-9
出力を確かめてメソッドの意味を確認しよう.

```
-----------------------------------------------
string = "012345:-)9012345678901:-)56789"
print(string.index(":-)"), "\n")
print(string.rindex(":-)"), "\n")
print(string.include?(":-<"), "\n")
-----------------------------------------------
```

演習 8-10
次のプログラムの出力を確認しよう.

```
-----------------------------------------------
string = "abcde"
print(string, "\n")
string[2] = "C"
print(string, "\n")
-----------------------------------------------
```

演習 8-11
次のプログラムの出力を確認しよう.

```
-----------------------------------------------
string = "abcde"
print(string, "\n")
print(string.slice!(2..3), "\n")
-----------------------------------------------
```

演習 8-12
つぎのプログラムの出力を確認しよう.

```
-----------------------------------------------
string = "abc"
print(string, "\n")
string.concat("def")
print(string, "\n")
-----------------------------------------------
```

演習 8-13
つぎのプログラムの出力を確認しよう.

```
-----------------------------------------------
s = "abcabc"
print(s, "\n")
print(s.delete("b"), "\n")
-----------------------------------------------
```

演習 8-14
つぎのプログラムの出力を確認しよう.

```
-----------------------------------------------
string = "A man, a plan, a canal --Panama!"
print(string, "\n")
print(string.reverse, "\n")
-----------------------------------------------
```

演習 8-15
つぎのプログラムの出力を確認しよう.

```
-----------------------------------------------
print("hogehoge.hogehoho".count("h"), "\n")
print("abababcababababc".scan(/abc/).length, "\n")
-----------------------------------------------
```

演習 8-16
文字列操作のためのメソッドについてまとめよう. (表 1)

表 1: 文字列操作のメソッド

メソッド名	説明
length	
size	
empty?	
split	
index	
rindex	
include?	
delete	
reverse	
strip	
upcase	
downcase	
swapcase	
capitalize	
tr	
concat	
count	

3　総合問題

演習 8-17
キーボードから姓と名を入力すると，以下のことを実行するプログラムを作ろう．姓名の入力はアルファベットで行うとする．

1. 姓と名それぞれの長さを表示する
2. 姓と名の長さの和を表示する
3. 入力形式に関わらず，姓は全て大文字で表示し，名前は先頭だけ大文字で表示する．
4. 姓と名をこの順で空白１文字で区切って表示する．
5. 二人分の姓名を入力できるようにして，それぞれ上記と同様の処理をする．
6. 二人の姓名をアルファベット順で表示する．

演習 8-18
要素数 10 の配列に整数を入れておく．キーボードから入力した整数と配列の各要素の大小を比較した結果を表示するプログラムを作成しよう．

演習 8-19
10 個の文字列の配列を用意し，それらのうち最大の長さの文字列を求めるプログラムを作成しよう．

演習 8-20
演習 4-8 のプログラムを変更して，一つも一致しなかった場合に,「一致しませんでした」と表示するようにしてみよう．

演習 8-21
変数名を入力すると，本演習での ruby コーディング規約の命名規約に違反しているかどうかをチェックして，良い変数名を提案してくれるプログラムを作ろう．まずは，以下のことをやってくれるようにしてみよう．

1. 記号類が含まれていたら，‘_’ で置き換えたものを推薦してくれる．
2. 大文字が含まれていたら，小文字で置き換えたものを推薦してくれる．

では，ローマ字方式か英単語方式かを判定する機能を実現するにはどうしたらよいだろう？

4　応用（余力のある人向け）

4.1　メソッドが自分自身を呼ぶ（再帰呼び出し）

メソッド内でメソッドを呼ぶことができることは既に学んだ．その応用で，メソッド内でそのメソッド自身を呼ぶこともできる．これを再帰呼び出しという．プログラミングにおけるとても面白い考え方である．

演習 8-22

子うさぎが 1 羽いる．1 年後に子うさぎは親うさぎになる．親うさぎは 1 年後に子うさぎを産む．こうして年々うさぎが増えていくと 10 年後には何羽になっているだろうか？まず，親を R，子を r として紙に書いてみよう．

最初	r
1 年後	R
2 年後	Rr
3 年後	RrR
4 年後	
5 年後	
6 年後	
7 年後	
8 年後	
9 年後	
10 年後	

演習 8-23

以下のカッコを埋めよう．

親うさぎの数は，前年の親うさぎがそのまま親として残るのと，子うさぎが親になるのとで，

$$\begin{aligned} \text{親うさぎの数} &= \text{前年の親うさぎの数} + \text{前年の子うさぎの数} && (1)\\ &= (\underline{\qquad\qquad\qquad\qquad}) \text{の数} && (2) \end{aligned}$$

となる．

一方，子うさぎの方は，前年の親うさぎが 1 羽ずつ産むので，

$$\begin{aligned} \text{子うさぎの数} &= \text{前年の親うさぎの数} && (3)\\ &= (\underline{\qquad\qquad\qquad\qquad}) \text{の数} && (4) \end{aligned}$$

となる．なお，式 (4) は式 (2) から導かれる．

結局,『**うさぎの数は，(_____）と（_____）の和となる**』ことが分かる．ただし，最初と 1 年後は 1 羽ずつである．

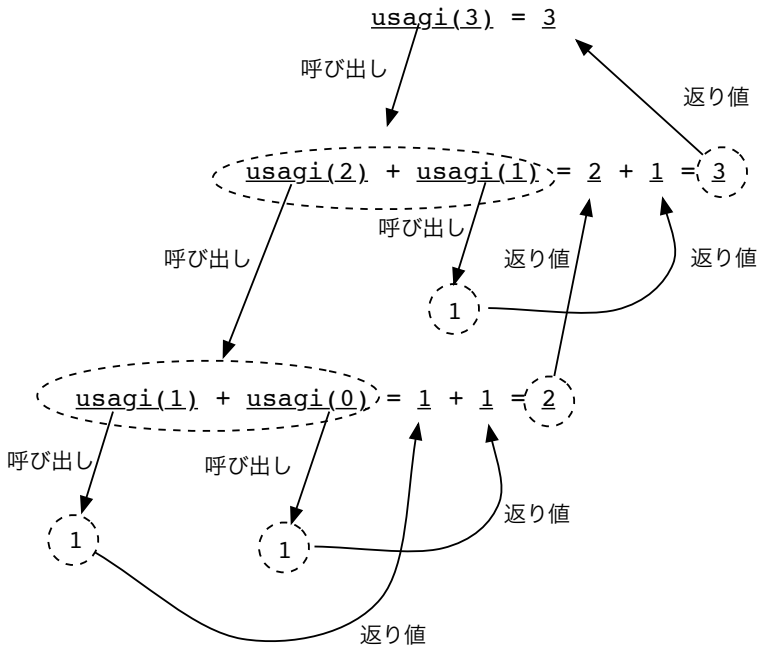
これをプログラムにすると以下のようになる.

```
-----------------------------------------------
# フィボナッチ数で増えていくうさぎの数を求めるメソッド
#   引数   n: 年数
#   返り値 n 年後のうさぎの数
#
def usagi(n)
  if n == 0 || n == 1
    return 1
  else
    return usagi(n - 1) + usagi(n - 2)
  end
end

print("10 年後にうさぎは", usagi(10), "羽だ!\n")
print("紙に書かせるなんてひどい！\n")
-----------------------------------------------
```

return usagi(n - 1) + usagi(n - 2) のところが再帰呼び出しである．メソッド usagi の中から usagi を呼び出している．ただし，引数に n - 2 があるため，n が 0 と 1 の時は，定義しておく必要がある.（数学が好きな人は，漸化式を思い出してもらえれば同じことである.）

これを n = 3 の場合の呼び出しと返り値の返り方の関係を図示すると下図のようになる.

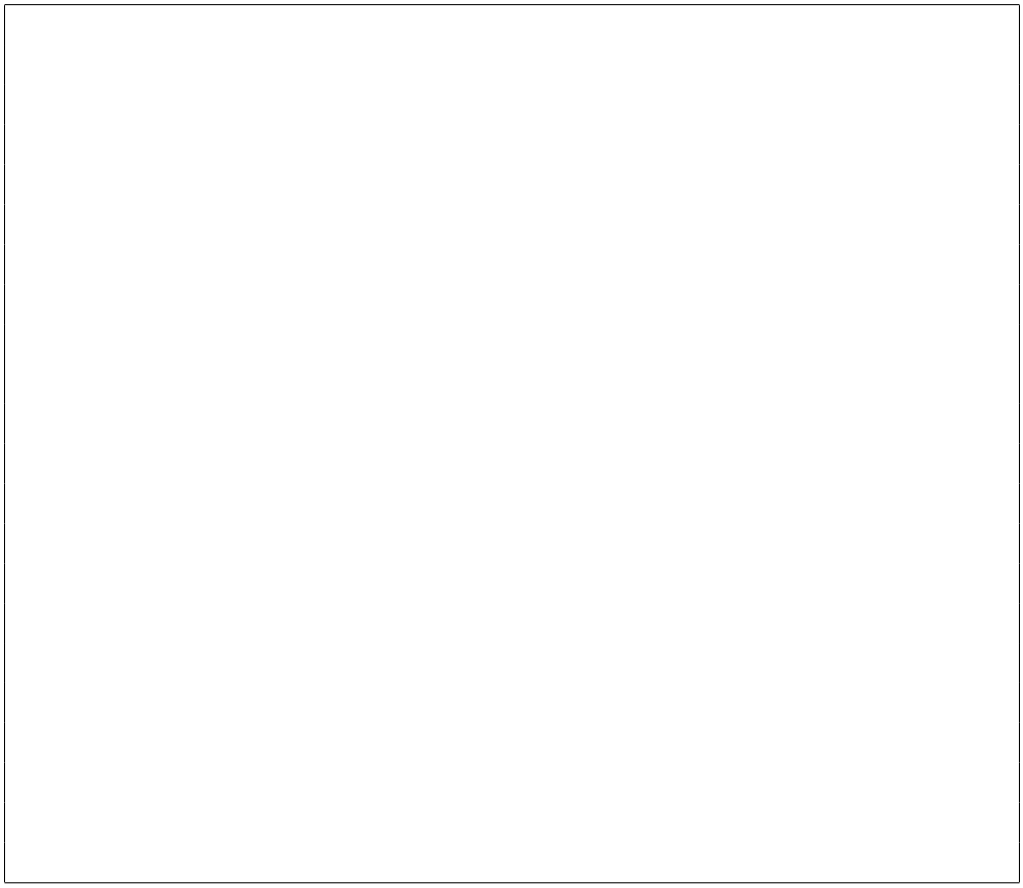


なお，このようにして求められる数列 1, 1, 2, 3, 5, 8, 13,... はフィボナッチ数列と呼ばれている.

ついでに補足しておくと，このうさぎ物語にオスうさぎは出てこない.

演習 8-24

`usagi(4)` の場合について図示してみよう．



演習 8-25

[**ハノイの塔**]「3 本の棒と，真ん中に穴のある大きさの異なる円盤 13 枚がある．初め，左の棒に大きい順に 13 枚穴を通して積み上げられている．この円盤を全て右の棒へ移動して，同じように大きい順に積み上げたい．ただし，円盤は他の棒にしか移動できず，移動できるのは 1 度に 1 枚である．しかも，自分より小さい円盤の上に積み上げることはできない．どのような動かし方をしたらよいか．手順を示せ.」

よく紹介される再帰呼び出しの問題である．ちょっと考えるとこんなことがプログラムでできるか思いつかないかもしれない．

まずは，次のように考えてみよう．

左の 13 枚を右に移動する手順は以下の通りである．

1. 左の 12 枚を中央に移動する．
2. 左の一番大きい円盤を右へ移動する．
3. 中央の 12 枚を右へ移動する．

絵に描いてみよう．

1
2
3

1 と 3 は枚数が 12 枚になっているだけで，もともとの問題である左の 13 枚を右に移動する手順と同じである．プログラムは以下のようになる．メソッド hanoi(n，from，to，tmp) は，n 枚の円盤を from 棒から to 棒へ tmp 棒を補助として使って移動する手順を求める．

```
-----------------------------------------------
# ハノイの塔を解く手順を表示するメソッド
#
#   引数   n:    円盤の数
#          from: 最初に円盤が積み上げられた棒の名前
#          to:   移動先の棒の名前
#          tmp:  作業用の棒の名前
#   返り値 なし
#
def hanoi(n, from, to, tmp)      # n 枚を from 棒から to 棒へ
  if n > 0
    hanoi(n - 1, from, tmp, to)        # n-1 枚を from 棒から tmp 棒へ
    print(n, "番目の円盤を" + from + "から" + to + "へ移動する\n")
    hanoi(n - 1, tmp, to, from)        # n-1 枚を tmp 棒から to 棒へ
  end
end

hanoi(3, "左", "右", "中")
hanoi(13, "左", "右", "中")
-----------------------------------------------
```

演習 8-26
階乗は以下のように定義される．再帰呼び出しを使って階乗を求めるメソッドを定義しよう．適切なコメントもつけるように．

$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

```
-----------------------------------------------
# 階乗を計算するメソッド
#   引数   ?????
#   返り値 ?????
#
def factorial(n)
  if n > 1
    return ???
  else
    return 1
  end
end

print(factorial(3))
-----------------------------------------------
```

演習 8-27
組み合わせの定義は次の通りである．組み合わせを求めるメソッドを定義して使ってみよう．
$({}_5C_3 = 10, {}_{10}C_7 = 120)$

$${}_nC_r = \begin{cases} 1 & r = 0 \text{ または } n = r \\ {}_{n-1}C_r + {}_{n-1}C_{r-1} & \text{それ以外} \end{cases}$$

```
-----------------------------------------------
# 組み合わせを計算するメソッド
#   引数   ?????
#          ?????
#   返り値 ?????
#
def combination(n, r)
  if ???
    return 1
  else
    return ???
  end
end

print(combination(5, 3))
-----------------------------------------------
```