プログラミング演習I
– 第9回 –
復習と応用問題

1　はじめに

第 9 回の演習の目標は,

- これまでの総復習をして，プログラミング演習 I の知識の定着を図る
- 応用問題で力をつける（余力のある人向け）

です.

2　総復習

演習 9-1
配列の要素を 0 番目から順に表示し，足し合わせていくプログラムを作成しよう.

```
-------------------------------------------------
# 配列の準備
kazu = [

# 配列の要素を全て表示しながら，足し合わせていく
goukei =
i = 0
while









end

print("配列の要素の合計は，", goukei, "です￥n")
-------------------------------------------------
```

演習 9-2
配列を与えると合計を計算し返すメソッドを作成し，上記と同様の結果を表示するプログラムを完成させよう．

```
-----------------------------------------------
# 配列の値を表示しながら，合計を計算し，最後に合計を返すメソッド
#    引数：_____________
#    返り値：_____________
def









end

# 配列の準備
kazu =

# 配列を引数としてメソッドを呼び，返り値を変数 goukei に保存する


# 表示
print("配列の要素の合計は，", goukei, "です￥n")
-----------------------------------------------
```

演習 9-3
ハッシュのキーと値を表示し，ハッシュの値を足し合わせていくプログラムを作成せよ．

```
-----------------------------------------------
# ハッシュの準備
yaoya = {

# ハッシュのキーと値を全て表示しながら，値を足し合わせていく
goukei =









print("ハッシュの値の合計は，", goukei, "です￥n")
-----------------------------------------------
```

演習 9-4
ハッシュを与えると値の合計を計算し返すメソッドを作成し，上記と同様の結果を表示するプログラムを完成させよ．

```
-----------------------------------------------
# ハッシュのキーと値を表示しながら，値の合計を計算し，最後に合計を返すメソッド
def




end

# ハッシュの準備
yaoya =

# ハッシュを引数として，メソッドを呼び，返り値を変数 goukei に保存する


# 表示
print("ハッシュの値の合計は，", goukei, "です￥n")
-----------------------------------------------
```

演習 9-5
要素数 5，初期値 0 の配列を準備し，キーボードから整数を入力すると配列の要素として順に保存し，すべての要素の入力を終えると，配列の要素を後ろから順に表示するプログラムを完成させよう．

```
-----------------------------------------------
# 配列の準備
kazu =

# 配列 kazu に整数を 0 番目から順に保存していく
# ただし，配列の要素数を変更しても，プログラムを変更せずに済む方法とする




# 配列 kazu の要素を後ろから順に表示する
# ただし，配列の要素数を変更しても，プログラムを変更せずに済む方法とする




-----------------------------------------------
```

演習 9-6
配列を与えると後ろから順に表示するメソッドを作成し，上記のプログラムと同様の動きをするプログラムを作成しよう．

```
-----------------------------------------------
# 引数として配列を受け取ると後ろから順に表示するメソッド
def









end

# 配列の準備
kazu =

# 配列 kazu に整数を 0 番目から順に保存していく




# 引数を配列 kazu とし，メソッドを呼ぶ

-----------------------------------------------
```

演習 9-7
上記プログラムを数字と数字の間に=>を挟んで表示するように変更しよう．たとえば，8，2，3，1，9 をキーボードから順に入力すると，9=>1=>3=>2=>8 と表示するメソッドを作成してみよう．

演習 9-8

キーには野菜の名前 (文字列)，値には値段 (整数）を保存するためのハッシュ yaoya を作成し，キーボードからキーと値を順に入力し，ハッシュyaoya に保存したあとで，値段の合計を表示するプログラムを作成しよう．なお，プログラムは以下の手順で少しずつ拡張していこう．

STEP1: 空のハッシュを用意して，3 組のキーと値をキーボードから入力できるようにする．

STEP2: 何組でも入力できるようにする．ただし，キーとして .(ピリオド）が入力されたら入力を終える．

STEP3: 同じキーが入力されたら，値を保存しない．

STEP4: 同じキーが入力されたら，値を比較し，もし小さければハッシュに保存するようにする．

```
------------------------------------------------
# STEP1 のテンプレート
# ハッシュの準備
yaoya =

# キーボードから入力されたキーと値をハッシュyaoya に順に保存することを 3 回繰り返す




# ハッシュ yaoya のキーと値を全て表示しながら，値を足し合わせていく
goukei =




# ハッシュの値の合計を表示する
print("ハッシュの値の合計は，", goukei, "です\n")
------------------------------------------------
```

3　応用（余力のある人向け）

3.1　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 9-9

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 9-10

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，

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

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，

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

となる．なお，式 (4) は式 (2) から導かれる．

結局,『ある年の**うさぎの数は,（**$a.$ _____**）と（**$b.$ _____**）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．

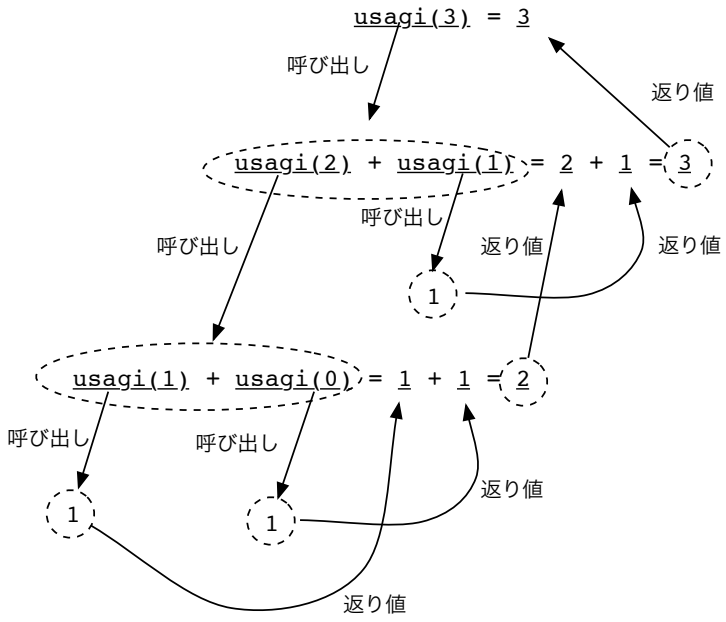
これをプログラムにすると以下のようになる．

```
------------------------------------------------
# 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 9-11

usagi(4) の場合について図示してみよう.

演習 9-12

[**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 9-13
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 9-14
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))
-----------------------------------------------
```