第 8 回　再帰的アルゴリズム

再帰的アルゴリズムについて学ぶ。

【関数の定義】

これまでは、print()や range()のように Python に組み込まれている関数を利用してきたが、関数を自作することができる。たとえば、第 4 回の授業で作成した身長と体重から BMI を計算する次のプログラムを、関数を自作することで書き換えてみよう。

```
a, b = map(float,input().split())
c = 10000 * b / (a ** 2)
print(c)
```

このプログラムは、関数 bmi を定義することで次のように書き変えることができる。

```
def bmi(a,b):
    return 10000 * b / (a ** 2)

a, b = map(float,input().split())
c = bmi(a,b)
print(c)
```

1 行目の def bmi(a,b): からはじまるブロックが、関数の定義である。2 行目までがインデントされていることで、2 行目までが関数の定義であることがわかる。def は関数を「定義(define)しますよ」ということであり、bmi は関数の名前である。そして、(a,b)では関数には 2 つの引数があり、それぞれを a と b という引数として受け取る。つまり、def BMI(a,b)は BMI という名前の関数には引数 a と b がある、ということを意味する。

2 行目の return とは、関数を終了することを意味する。return とだけ書かれると、戻り値がなく、return の後に書かれているものを「戻り値」として返す。つまり、

```
def bmi(a,b):
    return 10000 * b / (a ** 2)
```

の 2 行で、a と b を引数として受け取り、10000 * b / (a ** 2)の計算結果を戻り値として返す BMI という名前の関数が定義された。

その後の 2 行では、入力を a と b という変数で受け取り、この bmi 関数を使って計算した結果を出力する。

つまり、このプログラムは最初の BMI 計算プログラムとまったく同じ動きをする。bmi 関数は 1 回しか使われていないので、ここでは関数を定義するメリットはそれほど感じないが、同じ関数を何回か使うときには便利であり、関数に名前をつけて整理することでプログラムがわかりやすくなる。

【再帰的呼び出し】

関数の再帰的呼び出し(recursive call)とは、ある関数から自分自身を呼び出す処理である。例として、階乗 n!（n は非負整数）を考える。n! は次の定義で得られる。

(1) 0! = 1

(2) n! = n × (n - 1)!　　(ただし n > 0)

例えば5!は、5 × 4! と計算される。つまり、5! を求めるという計算の中で4!を求めるという関数を再帰的に呼び出すこととなる。4!は 4 × 3! なので、さらに再帰的に階乗を求める関数が呼び出される。これが 0! に到達するとようやく 1 という数値が得られ、結果として，5! = 5 × 4 × 3 × 2 × 1 になる。このように、自分自身を呼び出すアルゴリズムを再帰的アルゴリズムと言う。

再帰的定義とは、定義の中に、更にその定義されるべきものが簡単化されて含まれていることである。まったく同じ定義が含まれるときには、循環論法といって定義が定まらない。再帰的とはその部分に含まれるものが全く同じものではなくて、簡単になったもので、最終的終了条件が明示されているものである。

★階乗を求めるプログラムを実行してみよう。

```
def factorial(n):
    if n > 0:
        return n * factorial(n-1)
    else:
        return 1

n = int(input())
print(factorial(n))
```

実行例

入力：5
出力：120

入力：50
出力：30414093201713378043612608166064768844377641568960512000000000000

★この階乗の関数を利用して、ネイピア数（自然対数の底）の近似計算をしてみよう。

```
def factorial(n):
    if n > 0:
        return n * factorial(n-1)
    else:
        return 1

import math
e = 0
for k in range(20):
    e += 1 / factorial(k)
    print("k={0}: {1}".format(k,e))
print("  e = {0}".format(math.e))
```

ネイピア数は次の極限となることを使っている。

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$$

【ハノイの塔】

再帰アルゴリズムを用いて、ハノイの塔を解くプログラムを作成する。ハノイの塔とは、3 本ある杭を用いて，以下のルールに従ってすべての円盤を右端の杭に移動させられれば完成、というゲームである。

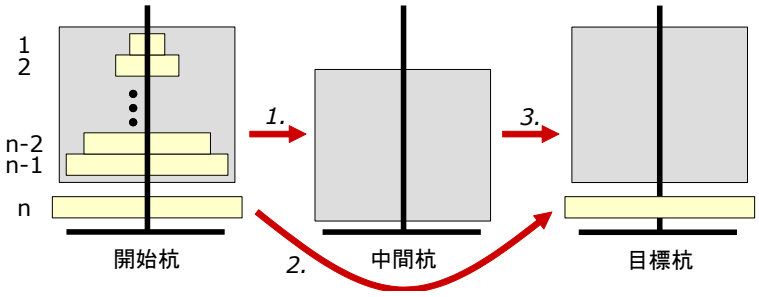
(1) 3 本の杭と、中央に穴の開いた大きさの異なる複数の円盤から構成される。
(2) 最初はすべての円盤が左端の杭に小さいものが上になるように順に積み重ねられている。
(3) 円盤を一回に一枚ずつどれかの杭に移動させることができるが、小さな円盤の上に大きな円盤を乗せることはできない。

https://hanoi.aimary.com/ で、ハノイの塔を実際にやってみよう。

この問題は、次のように考えれば解決できる。n 枚の円盤を開始杭から目標杭へと移動させる問題を、(n-1)枚の円盤のグループと、n 枚目の円盤 1 枚に分ける。そして、

(1) (n-1)枚の円盤のグループを開始杭から中間杭に移動させる
(2) n 枚目の円盤を開始杭から目標杭に移動させる
(3) (n-1)枚の円盤のグループを中間杭から目標杭に移動させる

という問題としてとらえる。



このように考えると、次に考えるべき問題は、(n-1)枚の円盤のグループを、開始杭から中間杭にどうやって移動させるか？という問題になる。つまり、n 枚の円盤を移動させる問題から、n-1 枚の円盤を移動させる問題へと、問題の規模が縮小したということである。これが再帰的アルゴリズムである。

【★課題】
ハノイの塔を実行する次のプログラムの????を埋めて完成させ、実行してみよう。正常に実行されたら、ToyoNet-ACE に提出してください。# で始まる行はコメントであり入力してもしなくても構いません。

```
def move(n, start, end):
    # n-1枚の円盤を start 番目の杭から中間杭へと移動する
    if n > 1:
        move(n-1, start, ????)
    # n 枚目の円盤を移動（移動結果を出力）
    print("円盤{0}を杭{1}から杭{2}へと移動".format(n,start,end))
    # n-1 枚の円盤を中間杭から end 番目の杭へと移動する
```

```
    if n > 1:
        move(n-1, ????, ????)

n = int(input())
move(n,1,3)
```

関数 move には,3 つのパラメータが必要である。第 1 パラメータが移動する円盤の枚数、第 2 パラメータが移動元（開始杭）の杭番号、第 3 パラメータが移動先（目標杭）の杭番号である。杭には 1 から 3 までの番号がついていて、はじめは開始杭の番号が 1 で目標杭の番号が 3 であるが、再帰的な呼び出しをするときには、開始杭と目標杭の番号が変わるために、それぞれをパラメータとして関数を呼び出し、move 関数の中では start と end という変数に格納される。

このプログラムのポイントは，「中間杭の番号を start と end の組み合わせでどう表現するか？」である。杭の番号は 1，2，3 で、これらの合計は 6 なので、もう分かるであろう。例えば 2 が中間杭だとすると，2 = 6 − 1 − 3，すなわち，中間杭 ＝ 6 − 開始杭 − 目標杭で求められる。あとは、これをプログラム中の変数を使った式で表現すれば良い。

【発展：NumPy と線形代数】

NumPy（ナンパイ）は、科学技術計算でよく利用される数値計算ライブラリである。NumPy の配列は、ベクトル、行列、テンソルなどの線形代数の計算を高速に実行できるところがその強みである。特に、最近の流行である人工知能（AI）の機械学習は、線形代数がその「すべて」であると言っても過言ではなく、線形代数の理解なしには語れない。Google が提供している機械学習ライブラリ TensorFlow は、その名の通り要するにテンソルの計算を高速にするためのライブラリである。Google Colaboratory[1]（略称: Colab）を使うと、Google のクラウドサーバー上で Python と TensorFlow を使って計算をすることができる。

近年は高校の数学でベクトルや行列を学ぶ機会がなくなっているため（ベクトルが教えられる「数学 B」は履修者が少ない）、線形代数になじみのない学生が多いであろう。この授業では線形代数や NumPy について深入りはしないが、以下のサンプルコードを動かしてみよう。

https://paiza.io/projects/AP4Xuxa5ZQ9Ndxxnmz48bw

```
import numpy as np
a = np.random.randint(0, 10, size=(3,3))
print('行列A')
print(a)
print('Aの転置行列')
print(np.transpose(a))
print('Aの逆行列')
print(np.linalg.inv(a))
w, v = np.linalg.eig(a)
print('Aの固有値')
print(w)
print('Aの固有ベクトル')
print(v)
```

[1] https://colab.research.google.com/