9 拡張ユークリッド互除法（再訪）　2024年度　総合教育科目演習 I（金曜 3 限）

入学年度		学部	学 科		組		番　号			検	フリガナ	
		B		1							氏 名	

Python を用いて，正の整数 a と b の最大公約数を求めるプログラムを書くことを考える．

ユークリッド互除法の原理

2つの正の整数 a, b の最大公約数を $\gcd(a, b)$ とする．a を b で割った余りを r とすると

$b \neq 0$ のとき　　$\gcd(a, b) = \gcd(b, r)$

$b = 0$ のとき　　$\gcd(a, 0) = a$

このユークリッドの互除法の原理を応用して，2 つの数の最大公約数が計算できる．例えば 899 と 696 の最大公約数を求める次のようになる．

$$
\begin{aligned}
899 &= 696 \cdot 1 + 203 && \gcd(899, 696)\\
696 &= 203 \cdot 3 + 87 && = \gcd(696, 203)\\
203 &= 87 \cdot 2 + 29 && = \gcd(203, 87)\\
87 &= 29 \cdot 3 + 0 && = \gcd(87, 29)\\
29 &= 0 && = \gcd(29, 0) = 29
\end{aligned}
$$

Python に限らず，一般のプログラム言語で関数を定義するテクニックの一つとして、**再帰呼び出し** (recursive call) と呼ばれるものがある．これは，ある関数 f の定義の中で f 自身を呼び出している箇所がというもので，このような定義は**再帰的** (recursive) であると言う．上の計算の右側の gcd に関する式をみれば，この再帰的定義が役に立ちそうであることが容易に想像できるだろう．

Python では，整数 a を正の整数 b で割ったときの商は，`a // b`, 余りは `a % b` で計算されることを思い出しておく．あとは，上の互除法の原理をそのままコードにするだけである．

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

「関数の定義と返値」については，東京大学「Python プログラミング入門」の §1-2，「if 文による条件分岐」については，§1-3 を参照のこと．また，「再帰」についても終わりの方の節で詳しく説明されている．

途中の結果も次々に表示するには，次のようにすればよい．

```
def gcd1(a, b):
    if b == 0:
        print("=", a)
    else:
        print(f'gcd({a}, {b}) = gcd({b}, {a % b})')
        return gcd1(b, a % b)
```

2024 年 6 月 28 日

1 再帰呼び出しを用いて，1 から n までの和を求める関数 `s(n)` を作れ．同様に，1 から n までの積である $n!$ を計算する関数 `factorial(n)` を作れ．

2 フィボナッチ数列はイタリアの数学者フィボナッチ（Fibonacci，1170 1259 年頃）が紹介した数列で，$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$ のように，どの数も前2つの数を足した数であるという規則で定義された数列である．言い換えると，フィボナッチ数列 $\{F_n\}$ は次の漸化式で定義される：

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad (n \geqq 2)$$

a) フィボナッチ数列の第 n 項を計算する関数を `fibonacci(n)` を再帰呼び出しを用いて書け．

b) a) で作った `fibonacci(n)` を用いて，F_{30} と F_{31} を求めよ．

c) 途中の結果も次々に表示するバージョンの関数 `gcd` を用いて $\gcd(F_{31}, F_{30})$ を計算せよ．

3 一般に，次の漸化式で定義される数列 $\{a_n\}$ を考える．

$$a_0 = a, \quad a_1 = b, \quad a_n = pa_{n-1} + qa_{n-2} \qquad (n \geqq 2)$$

a, b, p, q, n を入力すれば，一般項 a_n が求まる Python プログラムを書け．

先ほどの，最大公約数を求めるプログラムを改良して，a, b を与えたとき，$d = \gcd(a, b)$ と同時に $ax + by = d$ を満たす整数 x, y を求めるプログラムを作ろう．

いま，仮に $ax + by = d$ を満たす x, y が見つかっているとする．このとき，a を b で割り算して $a = bq + r$ と表し，これを $ax + by = d$ に代入すると，

$$(bq + r)x + by = d \Leftrightarrow b(qx + y) + ry = d$$

いま，$bx' + ry' = d$ の整数解 x', y' が求まっていれば，$qx + y = x', x = y'$ より，$x = y', y = x' - qy'$ とすれば，$ax + by = d$ の解 x, y が求まる．すなわち，

$$
\begin{aligned}
d &= b \cdot x' + r \cdot y'\\
&= b \cdot x' + (a - b \cdot q) \cdot y' = a \cdot y' + b \cdot (x' - qy')
\end{aligned}
$$

最初に挙げた具体例では，次のように次々に代入していくことになる．

$$
\begin{aligned}
29 &= 203 - 87 \cdot 2\\
&= 203 - (696 - 203 \cdot 3) \cdot 2 = 696 \cdot (-2) + 203 \cdot 7\\
&= 696 \cdot (-2) + (899 - 696 \cdot 1) \cdot 7 = 899 \cdot 7 + 699 \cdot (-9)
\end{aligned}
$$

と表せることがわかる．実際にプログラムにする場合には，最初は $d \cdot 1 + 0 \cdot 0 = d$ から始める（ユークリッドの互除法で $\gcd(a, b) = \cdots = \gcd(d, 0) = d$ となるまで続ける）．すなわち，上の式の前に次の式を付け加える．

$$
\begin{aligned}
29 &= 29 - 0 \cdot 0\\
&= 29 - (87 - 29 \cdot 3) \cdot 0 = 87 \cdot 0 + 29 \cdot 1\\
&= 87 \cdot 0 + (203 - 87 \cdot 2) \cdot 1 = 203 \cdot 1 + 87 \cdot (-2)
\end{aligned}
$$

【裏に続く】

そこで，$x, y, \gcd(a,b)$ を再帰呼び出しで求める．最後は $b = 0$ となったときに a が最大公約数になり，そのときの方程式は $a \cdot 1 + b \cdot 0 = a$ となる．

```
def extgcd(a, b):
    if b == 0:
        return 1, 0, a

    q, r = a // b, a % b
    x1, y1, d = extgcd(b, r)
    x, y = y1, x1 - q*y1
    return x, y, d
```

このコードの無駄を省いてすっきりさせると，

```
def extgcd1(a, b):
    if b == 0:
        return 1, 0, a

    x, y , d = extgcd1(b, a % b)
    return y, x - (a // b)*y, d
```

途中の計算結果も表示するようにすると，

```
def extgcd1(a, b):
    if b == 0:
        print(f'{a}*1+{b}*0={a}, gcd({a},{b})={a}')
        return 1, 0, a

    x, y, d = extgcd1(b, a % b)
    print(f'{a}*{y}+{b}*{x - (a//b)*y}={d}, gcd({a},{b})={d}')
    return y, x - (a//b)*y, d
```

4 次の 1 次不定方程式の 1 組の整数解を求めよ．

a) $163x + 78y = 1$　　b) $987x + 610y = 1$

再帰的定義は非常に便利なものであるが，互除法の過程，すなわち各段階での商と余りの式を逐一メモリに蓄えておき，最大公約数が求まったあと，それらの式を逆に辿ることが必要となるため，a, b が大きな数である場合，プログラムを実行するのにある程度のメモリが必要となり，実行時間も長くかかるようになる．

フィボナッチ数列の計算でも，再帰法では F_{40} を求めるのにかなりの時間がかかる．したがって，フィボナッチ数列のような漸化式による数列の計算は，再帰法ではなく，反復法を用いるのが普通である．

反復法（繰り返し）については，東京大学「Python プログラミング入門」の §3-2 などを参照のこと．どのようなプログラミング言語でも `for` 文と `while` 文は用意されており，これらが反復（繰り返し）の基本である．

フィボナッチ数列の第 n 項を求めるプログラムを，それぞれ `for` と `while` を用いて書くと，例えば次のようにできる．

```
def fibonacci1(n):
    a0, a1 = 0, 1
    for i in range(n):
        a0, a1 = a1, a0 + a1
    return a0
```

```
def fibonacci2(n):
    a0, a1 = 0, 1
    while n:
        a0, a1 = a1, a0 + a1
        n-=1
    return a0
```

5 再帰法によるプログラムと上のプログラムでそれぞれ，F_{40} を計算し，速さの違いを体感せよ．

担当：鍬田 政人 (kuwata.04f@g.chuo-u.ac.jp)