学期の前半で，合同類の乗法の表やべき乗の表を作った．これらを **Python** を用いて作ることを考える．表を効率よく作るには，「NumPy ライブラリ」と呼ばれるライブラリを利用する．「ライブラリ」については，京都大学「プログラミング演習 Python 2023」§1.9 (p. 33) に説明がある．また，NumPy ライブラリについては，東京大学「Python プログラミング入門」§5-3 に詳しく説明されている．

NumPy を用いるには，まず，numpy モジュールをインポートする必要がある．ふつう慣習として，np という省略した別名をつけて利用する．

```
import numpy as np
```

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．

```
def mult_table(m):
    table = np.array([[(k*l % m) for k in range(1,m)]
                      for l in range(1,m)]);
    print(table)
```

上の [(k の式) **for** k **in range**(1,m)] は for 文によってリストを作る定番の式で，range(1,m) は 1 から $m-1$ までのリスト $1, 2, \ldots m-1$ を表す．range(m) とすると 1 からではなく，0 から $m-1$ までのリストを表す．リストと for 文による繰り返しは，東京大学「Python プログラミング入門」§2-2 に詳しい解説がある．

2 上のプログラムで，m をいろいろ変えて乗法表を作ってみよ．

Python の普通の設定では，m を大きくすると，結果が ··· を用いて省略された形で出力される．これを避けるために，次のコマンドを実行しておく．

```
np.set_printoptions(linewidth=1000,threshold=10000)
```

Python では（大抵のプログラミング言語でも）整数 a と正の整数 m について，$a \bmod m$ の値 r は $0 \leqq r < m$ を満たすよう定義されている．これを，$-\frac{m}{2} < r \leqq \frac{m}{2}$ となるようにした方が便利なことがある．そこで，次のような関数 sym_mod を定義しておく．

```
def sym_mod(a,m):
  r = a % m;
  if r > m/2:
    return(r-m)
  else:
    return(r)
```

1 mult_tabe の定義のなかの (k*l % m) を sym_mod(k*l,m) に変えて，もう一度乗法表を作れ．

Fermat の小定理を扱った際，素数 p を 1 つ取り，$a^k \bmod p$ を次々に計算してその様子をみた．この表を，Python で作ってみよう．基本的な方法は乗法表の場合と同じである．

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

```
def kth_power(p):
    m = np.array([[sym_mod(a**k, p) for k in range(1,p)]
                  for a in range(1,p)])
    print(m);
```

3 p をいろいろ変えて $a^k \bmod p$ の表を作ってみよ．

● 文字コード

array（配列）の仕組みを利用して，いろいろな文字の表を作ってみよう．Python の組み込み関数 chr は与えられた整数を文字の番号（コード）とする文字を返す．逆に，組み込み関数 ord は与えられた文字の番号（コード）を整数として返す．これについては，東京大学「Python プログラミング入門」§3-2 の中の「for 文による繰り返し」に説明がある．

まず，文字コード 0 から 127 までの文字（ASCII コード文字などと呼ばれる）を 16 字ごとに分けて表示してみよう．

```
print(np.array([[chr(k) for k in range(16*l,16*(l+1))] for l in range(8)]))
```

この結果得られる表は次のようなものになる．

```
[['' '\x01' '\x02' '\x03' '\x04' '\x05' '\x06' '\x07' '\x08' '\t' '\n' '\x0b' '\
  x0c' '\r' '\x0e' '\x0f']
['\x10' '\x11' '\x12' '\x13' '\x14' '\x15' '\x16' '\x17' '\x18' '\x19' '\x1a' '\
  x1b' '\x1c' '\x1d' '\x1e' '\x1f']
[' ' '!' '"' '#' '$' '%' '&' "'" '(' ')' '*' '+' ',' '-' '.' '/']
['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' ':' ';' '<' '=' '>' '?']
['@' 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O']
['P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z' '[' '\\' ']' '^' '_']
['`' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o']
['p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z' '{' '|' '}' '~' '\x7f']
['\x80' '\x81' '\x82' '\x83' '\x84' '\x85' '\x86' '\x87' '\x88' '\x89' '\x8a' '\
  x8b' '\x8c' '\x8d' '\x8e' '\x8f']]
```

最初の 32 文字（0 から 31 まで）は特殊文字で，プログラムの制御用にのみ用いられるので，上の表ではおかしな記号で表されている．32 番スペース（空白）であり，48 番から数字が始まる．また，大文字の A は 65 番，小文字の a は 97 番である．

4 1 行目にアルファベットの大文字 26 文字，2 行目に小文字 26 文字が並ぶような array を表示せよ．

5 東京大学「Python プログラミング入門」§2-1 にある文字列（sting）の説明を読み，次のプログラムの実行結果がどうなるかを推測せよ．

```
text='I came, I saw, I conquer.'
print([text[k] for k in range(len(text))])
print([ord(text[k]) for k in range(len(text))])
```

【裏に続く】

【前回の補足】前回再帰的定義により，拡張ユークリッド互除法のプログラムを書いた．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．実は，最大公約数を求めると同時に x, y も求まるアルゴリズムがある．このアルゴリズムを用いれば，2 項間漸化式で定義されるフィボナッチ数列の一般項を求める方法をまねて簡潔なプログラムが書ける．それを見るために，まず，ユークリッドの互除法の証明の過程を振り返って見る．

いま，除法を表す式

$$a = bq + r, \qquad 0 \leq r < 0$$

において，$a = r_0, b = r_1$ とおき，q, r をそれぞれ q_1, r_2 と書き直すと，

$$r_0 = r_1 q_1 + r_2, \qquad 0 \leq r_2 < r_1$$

と表せる．そして，r_1 を r_2 で割ることにより，

$$r_1 = r_2 q_2 + r_3, \qquad 0 \leq r_3 < r_2$$

と表せる．さらにこれを繰り返し，

$$\begin{aligned} r_2 &= r_3 q_3 + r_4, & 0 \leq r_4 < r_3 \\ r_3 &= r_4 q_4 + r_5, & 0 \leq r_5 < r_4 \\ &\vdots & \vdots \\ r_{n-2} &= r_{n-1} q_{n-1} + r_n, & 0 \leq r_n < r_{n-1} \end{aligned}$$

そして，r_{n+1} が 0 になるとき，すなわち r_n が r_{n-1} を割り切るとき，$r_n = \gcd(a, b)$ となるのであった．この互除法の各段階において

$$ax_k + by_k = r_k$$

をみたす整数 x_k, y_k を見つけていく．そして，$r_{n+1} = 0$ になり互除法が終了したとき，$r_n = \gcd(a, b) = d$ なので，x_n, y_n が求める $ax + by = d$ の整数解となる．

まず，$r_0 = a, r_1 = b$ だから，

$$\begin{aligned} a \cdot 1 + b \cdot 0 &= r_0 \\ a \cdot 0 + b \cdot 1 &= r_1 \end{aligned}$$

が成り立つので，$(x_0, y_0) = (1, 0)$，$(x_1, y_1) = (0, 1)$ と定義する．いま，(x_k, y_k) まで定義されているとすると，r_{k+1} は r_{k-1} を r_k で割った余りなので，$r_{k-1} = r_k q_k + r_{k+1}$，すなわち，

$$r_{k+1} = r_{k-1} - r_k q_k$$

が成り立つ．ここで，$ax_{k-1} + by_{k-1} = r_{k-1}$ から $ax_k + by_k = r_k$ の両辺に q_k をかけた式を辺々引き算すると，

$$\begin{array}{rlcll} & ax_{k-1} & + & by_{k-1} & = r_{k-1} \\ -) & ax_k q_k & + & by_k q_k & = r_k q_k \\ \hline & a(x_{k-1} - x_k q_k) & + & b(y_{k-1} - y_k q_k) & = r_{k-1} - r_k q_k = r_{k+1} \end{array}$$

となり，x_{k+1}, y_{k+1} を次のように定義すればよいことがわかる．

$$\begin{cases} x_{k+1} = x_{k-1} - x_k q_k \\ y_{k+1} = y_{k-1} - y_k q_k \end{cases}$$

これをまとめると，数列 $\{r_n\}, \{q_n\}, \{x_n\}, \{y_n\}$ が次のような漸化式で定義されていることがわかる．

$$\begin{aligned} &q_1 = r_0 // r_1 && q_k = r_{k-1} // r_k \\ &r_0 = a, \quad r_1 = b, && r_{k+1} = r_{k-1} \ \% \ r_k, \\ &x_0 = 1, \quad x_1 = 0, && x_{k+1} = x_{k-1} - x_k q_k, \\ &y_0 = 1, \quad y_1 = 0, && y_{k+1} = y_{k-1} - y_k q_k. \end{aligned}$$

これをもとに，2 項間漸化式のプログラムをまねて，プログラムを書こうとすると，$\{q_n\}$ だけを少しだけ別扱いして次のようにすればよい．

```
def extended_gcd(a,b):
    # Initialization
    r0, r1, x0 , x1, y0, y1  = a, b, 1, 0, 0, 1

    # Loop
    while r1:
        q = r0 // r1
        r0, r1, x0 , x1, y0, y1 = r1, r0 % r1, x1, x0 - q*x1, y1, y0 - q*y1

    # Output
    print(f'{a}*({x0}) + {b}*({y0}) = {r0}')
```

担当：鍬田 政人 (kuwata.04f@g.chuo-u.ac.jp)