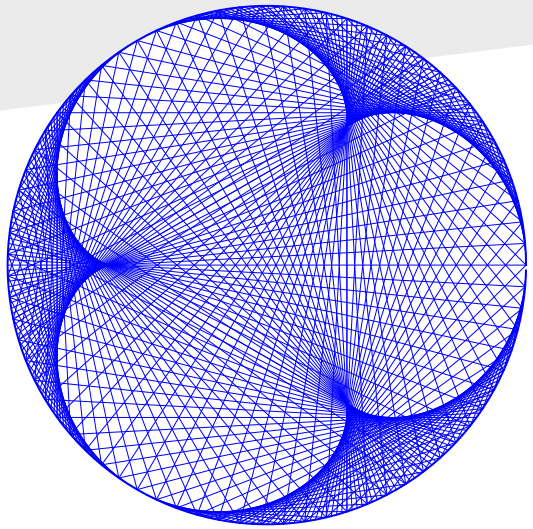
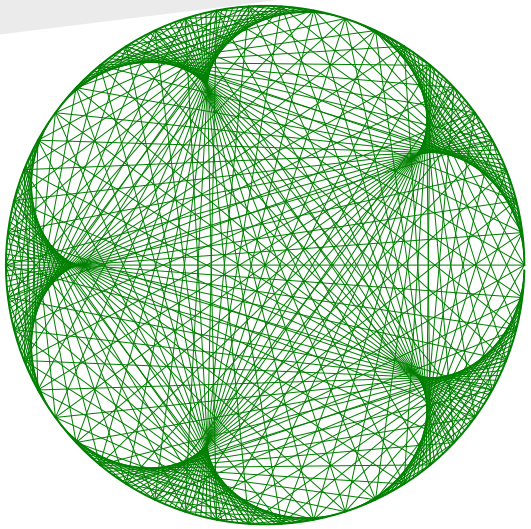
Python pour le supérieur

4) Arithmétique





Arithmétique

- Fonctions disponibles
  - Basiques
  - Avancées
  - Réciproque du petit théorème de Fermat ?
- Algorithmes d'Euclide
  - Différentes versions
  - Application à l'identité de Bézout
    - Calcul direct
    - Calcul en deux temps
- Théorème des restes chinois
  - Rappel et aperçu historique
  - Résolution d'un système de congruences

Quelques bases

Dans cette vidéo, `a, b, c, m, n, k,` ... sont des variables entières.

- `pow(a, n, m)` est le calcul optimisé de $a^n \mod m$
- `m//n` est le **quotient** de la division euclidienne de `m` par `n`.
- `m%n` est le **reste** de la division euclidienne de `m` par `n`.
  - `(m%n == 0)` teste si `m` est multiple de `n`.

Dans le module `math` :

- `gcd(m, n)` est le **PGCD** de `m` et `n` (*greatest common divisor*).
- `lcm(m, n)` est le **PPCM** de `m` et `n` (*least common multiple*).
- `(gcd(m, n)==1)` teste si `m` et `n` sont premiers entre eux.

Fonctions de sympy

Dans le module `sympy` :

Fonction	Description
`binomial(n, k)`	Coefficient binomial C_n^k
`divisors(n)`	Liste de tous les diviseurs de n
`factorint(n)`	Décomposition en facteurs premiers
`factorial(n)`	Factorielle ($n!$)
`mod_inverse(a,m)`	Inverse modulaire
`isprime(n)`	Teste si n est premier

Fonctions de sympy

Dans le module `sympy` :

Fonction	Description
`binomial(n, k)`	Coefficient binomial C_n^k
`divisors(n)`	Liste de tous les diviseurs de n
`factorint(n)`	Décomposition en facteurs premiers
`factorial(n)`	Factorielle ($n!$)
`mod_inverse(a,m)`	Inverse modulaire
`isprime(n)`	Teste si n est premier

Fonction	Description
`prime(n)`	n-ième nombre premier
`nextprime(n)`	Premier nombre premier supérieur à n
`prevprime(n)`	Premier nombre premier inférieur à n
`totient(n)`	Indicatrice d'Euler $\varphi(n)$
	etc.

Première application : contre-exemples à la réciproque du petit théorème de Fermat ?

Petit théorème de Fermat.

Soit p un nombre premier. Alors, pour tout entier a, on a :

$$a^p \equiv a \quad [p]$$

La réciproque est-elle vraie ? A-t-on : « **Soit un entier $n \geq 2$. Si, pour tout entier a, $a^n \equiv a\,[n]$, alors n est premier** » ?

Recherche de contre-exemples :

- faire une liste de nombres composés
- tester si `pow(a,n,n) == a` pour tout `a` entre `1` et `n-1`.

Première application : contre-exemples à la réciproque du petit théorème de Fermat ?

```
1  from sympy import *
2
3  C = [] # liste de nombres composés
4
5  for n in range(301, 3001, 2):
6      if(not isprime(n)):
7          C.append(n)
8
9  for n in C:
10     contre_ex = True
11     for a in range(1, n):
12         if(pow(a, n, n) != a):
13             contre_ex = False
14             break
15     if(contre_ex):
16         print(n)
```



Robert Daniel Carmichael (1879-1967)

Exécution du code

```
561, 1105, 1729, 2465, 2821
```

← **Nombres de Carmichael**

Algorithme d'Euclide

Exemple 1 : Calcul détaillé du PGCD de 364 et 63

```
1 a, b = 364, 63
2
3 while(a%b!=0):
4     print(f"{a} = {a//b}*{b} + {a%b}")
5     a, b = b, a%b
6
7 print(f"Le PGCD vaut {b}.")
```

Exécution du code

```
364 = 5*63 + 49
63 = 1*49 + 14
49 = 3*14 + 7
Le PGCD vaut 7.
```

Algorithme d'Euclide

Version historique (différences successives) :

```
1  a, b = 364, 63        # a > 0, b > 0
2
3  while(a!=b):
4      if(a>b):
5          a = a-b
6      else:
7          b = b-a
8
9  print(a)
```

Version récursive :

```
1  def pgcd(a, b): return a if b == 0 else pgcd(b, a%b)
2
3  print(pgcd(364, 63))
```

Coefficients de Bézout : la théorie

Théorème de Bézout.

Soient a et b deux entiers.

a et b sont premiers entre eux si, et seulement si, il existe $(u, v) \in \mathbb{Z}^2$ tel que $au + bv = 1$ (E)

Question.

Comment trouver un tel couple (u, v) ?

Quel est l'ensemble des couples qui sont solutions de (E) ?

$\rightarrow$ Calcul direct ou « en deux temps »

Coefficients de Bézout : la théorie

Calcul direct. Soient a et b deux entiers premiers entre eux.

On définit 3 « suites » $(r_n)_n, (u_n)_n$ et $(v_n)_n$ avec :

- $r_0 = a,\ u_0 = 1$ et $v_0 = 0$
- $r_1 = b,\ u_1 = 0$ et $v_1 = 1$
- Pour $n \geq 2$. Soit q le quotient de la division euclidienne de r_{n-2} par r_{n-1}. On pose : $r_n = r_{n-2} - q\, r_{n-1}$ ainsi que $u_n = u_{n-2} - q\, u_{n-1}$ et $v_n = v_{n-2} - q\, v_{n-1}$

En itérant jusqu'au dernier reste non nul, on obtient un couple $(u_N, v_N) \in \mathbb{Z}^2$ vérifiant (E). L'ensemble des couples de $\mathbb{Z}^2$ vérifiant (E) est donné par $\{(u_N + bk,\ v_N - ak),\ k \in \mathbb{Z}\}$

Coefficients de Bézout : calcul direct

Exemple 2 : Trouver les couples (u, v) vérifiant $141u + 34v = 1$

- $141 = \mathbf{4} \times 34 + 5$
  - pour u : $1 - 4 \times 0 = 1$
  - pour v : $0 - 4 \times 1 = -4$
- $34 = \mathbf{6} \times 5 + 4$
  - pour u : $0 - 6 \times 1 = -6$
  - pour v : $1 - 6 \times (-4) = 25$
- $5 = \mathbf{1} \times 4 + 1$
  - pour u : $1 - 1 \times (-6) = 7$
  - pour v : $-4 - 1 \times 25 = -29$
- $4 = \mathbf{4} \times 1 + 0$
  - reste nul

n	q	r_n	u_n	v_n
0		141	1	0
1		34	0	1
2	4	5	1	−4
3	6	4	−6	25
4	1	1	7	−29

Coefficients de Bézout : calcul direct

Exemple 2 : Trouver les couples (u, v) vérifiant $141u + 34v = 1$

D'où : $141 \times 7 + 34 \times (-29) = 1$

L'ensemble des solutions est :

$\{(7 + 34k,\ -29 - 141k), k \in \mathbb{Z}\}$

Tableau de vérification pour $k \in [\![-4\,;4]\!]$ $\longrightarrow$

k	u	v	$au + bv$
-4	-129	535	1
-3	-95	394	1
-2	-61	253	1
-1	-27	112	1
0	7	-29	1
1	41	-170	1
2	75	-311	1
3	109	-452	1
4	143	-593	1

Coefficients de Bézout : code itératif

```python
a, b = 141, 34
R = [a, b]
Q, U, V = ["", ""], [1, 0], [0, 1]

while (b!=0):
    q = a // b
    Q.append(q)
    R.append(a%b)
    U.append(U[-2] - q*U[-1])
    V.append(V[-2] - q*V[-1])
    print(f"{a} = {q}*{b} + {a%b}")
    a, b = b, a%b

print("\nn", "q", "r_n", "u_n", "v_n", sep='\t|\t')

for n in range(len(Q)-1):
    print(n, Q[n], R[n], U[n], V[n], sep='\t|\t')

print(f"\nD'où : {R[0]}*{U[-2]} + {R[1]}*{V[-2]} =
{R[-2]}")
```

Coefficients de Bézout : code itératif

Exécution du code

```
141 = 4*34 + 5
34 = 6*5 + 4
5 = 1*4 + 1
4 = 4*1 + 0

n   |   q   |   r_n  |   u_n  |   v_n
0   |       |   141  |   1    |   0
1   |       |   34   |   0    |   1
2   |   4   |   5    |   1    |   -4
3   |   6   |   4    |   -6   |   25
4   |   1   |   1    |   7    |   -29

D'où : 141*7 + 34*-29 = 1
```

Exercices

1) Donner l'ensemble des couples $(u, v) \in \mathbb{Z}^2$ vérifiant $58u + 27v = 3$ (calcul à la main)

2) Je possède des bidons de 17 litres et 26 litres.

Combien en faut-il de chaque pour remplir exactement 500 litres ?

Coefficients de Bézout : calcul en deux temps

Calcul en deux temps.

1) On détermine la « suite » (q_n) des N quotients $q_0, \ldots, q_{N-1}$ de l'algorithme d'Euclide (simple).

2) On construit les suites (u_n) et (v_n) avec les relations suivantes :
   - $u_0 = 1,\ v_0 = 0$
   - $u_{n+1} = v_n$ et $v_{n+1} = u_n - q_{N-1-n} v_n$

Exemple 3 : Trouver les couples (u, v) vérifiant $141u + 34v = 1$

- $141 = \mathbf{4} \times 34 + 5$
- $34 = \mathbf{6} \times 5 + 4$
- $5 = \mathbf{1} \times 4 + 1$
- $4 = \mathbf{4} \times 1 + 0$

D'où $(q_n) = (4, 6, 1, 4)$

$N = 4$

Coefficients de Bézout : calcul en deux temps

Exemple 3 : Trouver les couples (u, v) vérifiant $141u + 34v = 1$

- Avec $(q_n) = (4, 6, 1, 4)$, $u_0 = 1$, $v_0 = 0$,
  $u_{n+1} = v_n$ et $v_{n+1} = u_n - q_{N-1-n} v_n$

n	**Euclide (inversé)**	u	v	**Expression du PGCD**
0		1	0	$1 = 1 \times 1 + 0 \times 0$
1	$4 = \mathbf{4} \times 1 + 0$	0	1	$1 = 4 \times 0 + 1 \times 1$
2	$5 = \mathbf{1} \times 4 + 1$	1	-1	$1 = 5 \times 1 + 4 \times (-1)$
3	$34 = \mathbf{6} \times 5 + 4$	-1	7	$1 = 34 \times (-1) + 5 \times 7$
4	$141 = \mathbf{4} \times 34 + 5$	7	-29	$1 = 141 \times 7 + 34 \times -29$

L'ensemble des solutions est : $\{(7 + 34k, -29 - 141k),\ k \in \mathbb{Z}\}$

Coefficients de Bézout : code récursif

Version récursive.

```python
def bezout(a, b):
    if(b==0):
        return a, 1, 0
    d, u, v = bezout(b, a%b)
    return d, v, u-(a//b)*v

print(bezout(141, 34))
```

Descente :

```
bezout(141, 34) ?
d, u, v = bezout(34, 5)
d, u, v = bezout(5, 4)
d, u, v = bezout(4, 1)
d, u, v = bezout(1, 0)
```

Remontée :

```
d,  u,  v
1,  1,  0
1,  0,  1
1,  1, -1
1, -1,  7
1,  7, -29
```

Exercices

Implémenter la version « deux temps » avec un code itératif. Le code affiche les étapes de l'algorithme d'Euclide (simple) puis les différentes expressions du PGCD lors de la remontée.

Exécution du code

```
141 = 4*34 + 5
34 = 6*5 + 4
5 = 1*4 + 1
4 = 4*1 + 0

1 = 4*0 + 1*1
1 = 5*1 + 4*-1
1 = 34*-1 + 5*7
1 = 141*7 + 34*-29
```

Théorème des restes chinois

Énigme. *« Soient des objets en nombre inconnu. Si on les range par 3 il en reste 2. Si on les range par 5, il en reste 3 et si on les range par 7, il en reste 2. Combien a-t-on d'objets ? »* (Sun Zi, III^{e} siècle apr. J.-C.)

Théorème des restes chinois.

Soient $n_1, ..., n_k$ des entiers deux à deux premiers entre eux et $n = \prod_{i=1}^{k} n_i$. Alors, pour tout $(a_1, ..., a_k) \in \mathbb{Z}^k$, il existe un entier x, unique modulo n, tel que $\forall i \in [\![1\,;k]\!],\ x \equiv a_i \quad [n_i]$

Exemple 4 : Trouver $x \in \mathbb{N}$ tel que $x \equiv 3\,[5]$; $x \equiv 1\,[7]$; $x \equiv 4\,[11]$

- 5, 7 et 11 sont deux à deux premiers entre eux.
  Donc x existe et il est unique modulo $5 \times 7 \times 11 = 385$

Résolution d'un système de congruences

Exemple 4 : Trouver $x \in \mathbb{N}$ tel que $x \equiv 3\,[5]; x \equiv 1\,[7]; x \equiv 4\,[11]$

1) On cherche un x_1 tel que $x_1 \equiv 3\ [5]$ et $x_1 \equiv 0\ [77]$

   Par l'algorithme d'Euclide étendu, on a : $77 \times (-2) + 31 \times 5 = 1$

   Donc $77 \times (-2) \equiv 1\ [5]$ et $77 \times (-2) \times 3 \equiv 3\ [5]$.

   On pose $x_1 = 77 \times (-2) \times 3 = -462$

2) On cherche un x_2 tel que $x_2 \equiv 1\ [7]$ et $x_2 \equiv 0\ [55]$

   On a : $55 \times (-1) \times 1 \equiv 1\ [7]$. On pose $x_2 = -55$

3) On cherche un x_3 tel que $x_3 \equiv 4\ [11]$ et $x_3 \equiv 0\ [35]$

   On a : $35 \times (-5) \times 4 \equiv 4\ [11]$ On pose $x_3 = -700$

4) On a : $x_1 + x_2 + x_3 = -462 - 55 - 700 = -1217$

   Donc $x = 4 \times 385 - 1217 = 323$ est solution.

Résolution d'un système de congruences

```python
import numpy as np

def bezout(a,b):
    if(b==0):
        return a, 1, 0
    d, u, v = bezout(b,a%b)
    return (d, v, u-(a//b)*v)

A, N = [3, 1, 4], [5, 7, 11]
k, n = len(A), np.prod(N)

Q = [n//N[i] for i in range(k)]
Y = [bezout(Q[i],N[i])[1] for i in range(k)]  #2e argumt

x = 0
sol = "x = "    # expression d'une solution

for i in range(k):
    x += (Q[i]*Y[i]*A[i])
    sol += f"{Q[i]}*{Y[i]}*{A[i]} + "

print(f"{sol[0:-3]} = {x%n} [{n}]")
```

Exécution du code

```
x = 77*-2*3 + 55*-1*1
+ 35*-5*4 = 323 [385]
```

Exercices

1) Trouver à la main la solution à l'énigme de Sun Zi.
   « Soient des objets en nombre inconnu. Si on les range par 3 il en reste 2. Si on les range par 5, il en reste 3 et si on les range par 7, il en reste 2. Combien a-t-on d'objets ? (au minimum) »

2) Trouver l'ensemble des x vérifiant $x \equiv 9\ [11]$, $x \equiv 8\ [17]$ et $x \equiv 2\ [25]$ en utilisant, au plus, la fonction `bezout`.

3) Ajouter une vérification au dernier code présenté :

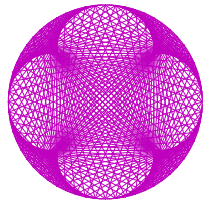
Exécution du code

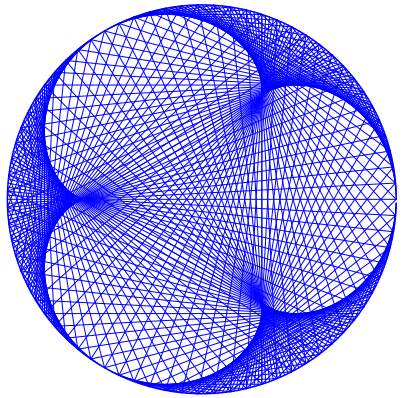
```
VÉRIFICATION
323 = 3 [5]
323 = 1 [7]
323 = 4 [11]
```

Conclusion

Approfondissements possibles

- Reproduire les illustrations du titre (« *La face cachée des tables de multiplication* », vidéo de Mickaël Launay)
- Test de primalité de Miller-Rabin (probabiliste)
- Chiffrements RSA et El Gamal





Quelques conseils

- Connaitre, *a minima*, les fonctions de base en arithmétique
- Savoir retrouver les fonctions avancées
- Être capable de programmer de mémoire une fonction `bezout`

Bibliographie

- Gilles Bailly-Maitre, *Arithmétique et cryptologie (2e édition)*, Ellipses, 2021
- Philippe Caldero et Marie Peronnier, *Carnet de voyage en Algébrie*, Calvage et Mounet, 2022
- Xavier Gourdon, *Les maths en tête. Algèbre et probabilités (3e édition)*, Ellipses, 2021
- Sébastien Peronno, *40 thèmes illustrés par le numérique pour l'agrégation interne de mathématiques*, Ellipses, 2026
- `joliesmaths.fr/sup` → téléchargement des codes et diapos

Solutions

1) $58 \times 7 + 27 \times (-15) = 1$

$58 \times 21 + 27 \times (-45) = 3$

L'ensemble des solutions est :
$\{(21 - 27k,\ 58k - 45),\ k \in \mathbb{Z}\}$

2) $26 \times 2 + 17 \times (-3) = 1$. Pour tout $k \in \mathbb{Z}$, on a :

$26 \times (1000 - 17 \times k) + 17 \times (26k - 1500) = 500$

```python
for k in range(100):
    if(1000-17*k>0 and 26*k-1500>0):
        print(k, 1000-17*k, 26*k-1500)

# Renvoie -> 58 14 8
```

Avec $k = 58$, on trouve 14 bidons de 26 litres et 8 bidons de 17 litres.

Solutions

Coefficients de Bézout : deux temps, itératif

```python
a, b = 141, 34
A, B, Q = [a], [b], []

while (b != 0):
    Q.append(a // b)
    print(f"{a} = {a//b}*{b} + {a%b}")
    a, b = b, a % b
    A.append(a)
    B.append(b)

d, u, v = a, 1, 0
N = len(Q)
print(" ")

for n in range(N):
    u, v = v, u-Q[N-1-n]*v
    print(f"{d} = {A[N-1-n]}*{u} +
{B[N-1-n]}*{v}")
```

Exécution du code

```
141 = 4*34 + 5
34 = 6*5 + 4
5 = 1*4 + 1
4 = 4*1 + 0

1 = 4*0 + 1*1
1 = 5*1 + 4*-1
1 = 34*-1 + 5*7
1 = 141*7 + 34*-29
```

Solutions

Théorème des restes chinois

1) 23

2) $x \equiv 425 \times (-3) \times 9 + 275 \times 6 \times 8 + 187 \times (-2) \times 2 \ [4675]$
$\equiv 977 \ [4675]$
L'ensemble des solutions est donc $\{977 + 4675k,\ k \in \mathbb{Z}\}$

3) Code :

```python
for i in range(k):
    print(f"{x%n} = {x%N[i]} [{N[i]}]")
```

Fin