Arithmétique 1

Nous utilisons couramment le système décimal. Cependant, nous pouvons aussi être amenés à utiliser l'écriture binaire (base 2) ou hexadécimale (base 16). Pour convertir à la main un nombre décimal dans l'une de ces bases, disons la base n, on procède par divisions successives du nombre par n. Le nombre converti commence par le dernier reste trouvé, les autres chiffres viennent des autres restes, lus du dernier au premier.

La conversion peut se faire de manière automatique avec Sage : si d est un nombre décimal,

- `bin(int(d))` renvoie l'écriture binaire de d sous la forme `0b...`,
- `hex(int(d))` renvoie l'écriture hexadécimale de d sous la forme `0x...`.

Inversement, pour retrouver le nombre décimal correspondant à un nombre sous forme binaire, il faut lancer `Integer('0b...')` et pour un nombre sous forme hexadécimale, il faut lancer `Integer('0x...')`.

Exercice 1.
Tester les commandes précédentes en convertissant le nombre décimal 31 en écriture binaire et hexadécimale.
Que valent :

- le nombre écrit en binaire : $0b10100$?
- le nombre écrit en hexadécimal : $0xA2E4F$?

Vérifier les calculs.

Si m est un nombre défini auparavant, la commande `m.str(n)` renvoie une chaine de caractères correspondant à l'écriture de m dans la base n. Cette commande permet ainsi de convertir m dans n'importe quelle base (pas uniquement 2 ou 16), et accepte les nombres m décimaux (pas uniquement entiers).

Exercice 2.
Définir x la valeur de π avec 3 chiffres significatifs (voir `numerical_approx`).
Définir y égal à 3.14 (avec 3 chiffres significatifs également).
Observer que, malgré les apparences, $x \neq y$. Comparer les écritures binaires de x et de y.
Cela est lié au fait que les nombres sont stockés en mémoire en base 2, et non en base 10.

Exercice 3.
Ecrire :

- le nombre 23 en base 11 ;
- le nombre 45 en base 7 ;
- le nombre 32 en base 5 ;
- le nombre 65 en base 13 ;

et vérifier les calculs.

Lorsque l'on cherche à crypter des messages, on utilise souvent des calculs modulo un entier n. Pour obtenir des résultats modulo n (dans $\mathbb{Z}/n\mathbb{Z}$), on peut prendre le modulo à la fin du calcul ou bien travailler directement dans $\mathbb{Z}/n\mathbb{Z}$.

Exercice 4.

- On pose $a = 5$. Calculer $2a$ modulo 6.
- On choisit maintenant de définir $b = 5$ comme étant un élément de $\mathbb{Z}/6\mathbb{Z}$ par la commande : `b = mod(5,6)`. Calculer $2b$. Qu'observe-t-on ?
  Calculer également la valeur de $b + 8$.
  Noter enfin que `1/b` renvoie la valeur 5 puisque $5 \times b = 25 \equiv 1 \mod 6$.
  Attention, le module $n = 6$ n'apparaît pas explicitement dans les résultats. On peut retrouver cette valeur en utilisant la commande `b.base_ring()`.

Exercice 5.

Soient $n = 3^{100000}$ et $a = n - 1$.

- Calculer a^{100} et donner le résultat modulo n.
- Définir maintenant $b = n - 1 \mod n$ et (re)faire le calcul de b^{100}.
- Comparer les temps de chaque calcul avec la commande `timeit('···')`.

Étant donné un entier n, on peut tester s'il s'agit d'un nombre premier avec la commande `is_prime`. Cependant, ce test peut être assez coûteux et, parfois, on n'effectue qu'un **test de pseudo-primalité** `is_pseudoprime`. On sait en effet (variante du petit théorème de Fermat) que si n est premier, alors, pour tout entier $0 < k < n$, on a $k^{n-1} \equiv 1 \mod n$. On peut ainsi tester plusieurs valeurs de k, et si pour l'une d'elle, $k^{n-1} \not\equiv 1 \mod n$, on peut affirmer que n n'est pas premier, sinon, on ne peut pas conclure théoriquement (le test répond que le nombre est pseudo-premier). En pratique, aucun contre-exemple n'a été trouvé à ce jour.

Exercice 6.

Tester les fonctions `is_prime` et `is_pseudoprime` sur quelques exemples et comparer les temps de chaque calcul.

L'**algorithme d'Euclide étendu** (variante de l'algorithme d'Euclide) permet, à partir de deux entiers a et b, de calculer non seulement leur plus grand commun diviseur (PGCD), mais aussi un couple de coefficients de Bézout : deux entiers u et v tels que $au + bv = \text{PGCD}(a, b)$. Quand a et b sont premiers entre eux, u est alors l'inverse de a pour la multiplication modulo b (et v est de la même façon l'inverse modulaire de b, modulo a), ce qui est un cas particulièrement utile.

On commence par calculer le reste de la division de a par b, qu'on note r ; puis on remplace a par b, puis b par r, et on réapplique le procédé depuis le début. On obtient ainsi une suite, qui vaut 0 à un certain rang ; le PGCD cherché est le terme précédent de la suite. En exprimant à chaque itération le reste comme une combinaison linéaire de a et b, le pgcd est alors exprimé comme une combinaison linéaire de a et b (d'où les coefficients u et v).

Exemple : prenons $a = 77$ et $b = 5$.
La division de a par b s'écrit : $a = \mathbf{77} = \mathbf{5} \times 15 + 2$, d'où $2 = \mathbf{77} - \mathbf{5} \times 15$.
On remplace a par b, donc $a = 5$, et b par r, donc $b = 2$.
La division de a par b s'écrit : $a = \mathbf{5} = 2 \times 2 + 1$, d'où $1 = \mathbf{5} - 2 \times 2 = \mathbf{5} - 2 \times (\mathbf{77} - \mathbf{5} \times 15) = \mathbf{77} \times (-2) + \mathbf{5} \times 31$.
On remplace a par b, donc $a = 2$, et b par r, donc $b = 1$.
La division de a par b s'écrit : $a = 2 = 1 \times 2 + 0$, l'algorithme est terminé, le dernier reste non nul est $1 = \text{PGCD}(a, b)$, et $1 = 77 \times (-2) + 5 \times 31$.

Exercice 7 (*).
Programmer l'algorithme d'Euclide étendu et le tester sur l'exemple précédent.

Exercice 8.
Rechercher dans l'aide le fonctionnement de la commande `xgcd` et la tester sur l'exemple précédent.

L'algorithme d'Euclide étendu fournit également une méthode efficace non seulement pour déterminer quand une équation diophantienne $ax + by = c$ possède une solution, ce que permet déjà l'algorithme d'Euclide simple, mais également pour en calculer dans ce cas une solution particulière, dont on déduit facilement la solution générale.

Une autre application utile des calculs modulaires est ce qu'on appelle communément les "restes chinois". Étant donnés m et n premiers entre eux, soit x un entier inconnu tel que $x \equiv a \mod m$ et $x \equiv b \mod n$. Alors le **théorème des restes chinois** permet de reconstruire de façon unique la valeur de x modulo le produit mn. En effet, on déduit de $x \equiv a \mod m$ que x s'écrit sous la forme $x = a + km$ avec $k \in \mathbb{Z}$. En remplaçant cette valeur dans l'expression $x \equiv b \mod n$, on obtient $k \equiv k_0 \mod n$, où [1] $k_0 \equiv \frac{(b-a)}{m} \mod n$. Il en résulte $x = x_0 + snm$, où $x_0 = a + k_0 m$, et s est un entier quelconque.

Exercice 9.
Vérifier que la commande `crt` permet bien de retrouver la valeur x_0. On testera la commande dans le cas où $a = 2, b = 3, m = 7, n = 8$.

On peut aussi s'intéresser à la recherche d'un entier x tel que, pour g, a, n donnés (avec a et n premiers entre eux), on ait : $g^x \equiv a \mod n$: il s'agit du problème du **logarithme discret**.

Exercice 10.
On choisit $g = 2, n = 53, a = 14$ (on pourra définir a modulo n). Après avoir recherché l'aide de `a.log()`, résoudre le problème du logarithme discret.

1. La division par m doit être interprétée comme le calcul de l'inverse modulo n, voir exercice 4.

Exercice 11.

Afin de coder un message, on assimile chaque lettre de l'alphabet à un nombre entier comme l'indique le tableau ci-dessous (on ne considèrera que les minuscules) :

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

- Tester les opérations suivantes :
  - ∗ transformer un mot en une liste de lettres,
  - ∗ transformer chaque lettre en nombre comme indiqué ci-dessus avec la commande `ord`,
  - ∗ transformer une liste de nombres entre 0 et 25 en lettres comme indiqué ci-dessus avec la commande `chr`,
  - ∗ transformer une liste de lettres `L` en un mot avec la commande `''.join(L)`.

Un message étant donné en langue naturelle, le chiffrement ou cryptage consiste à le traduire en langage codé et le déchiffrement consiste à traduire le message codé en langage naturelle.
On choisit le chiffrement suivant : à partir du nombre x, on calcule $11x + 8$ puis le reste de la division euclidienne de $11x + 8$ par 26 noté y (y est compris entre 0 et 25). Un tel chiffrement est dit "affine".
Exemple :
la lettre m est assimilée à $x = 12$: 11*12+8=140 et $140 \equiv 10 \mod 26$. Ainsi m est codée k.

- Tester ce chiffrement.

On s'intéresse maintenant du déchiffrement :

- Quel est l'inverse de 11 modulo 26 ?
- Comment calculer x à partir de y et de la réponse précédente ?
- Coder le déchiffrement et le tester sur l'exemple que vous avez chiffré précédemment.
- Comment déchiffrer le message "tnifg" ?

Exercice 12 (*).

On ne veut plus se restreindre aux seules minuscules, on souhaite désormais pouvoir utiliser tous les caractères. Reprendre l'exercice précédent, toujours avec un chiffrement affine donné par $11x + 8$ mais changer le modulo pour accéder à l'ensemble des caractères possibles.
Programmer le chiffrement et le déchiffrement correspondants.