Exercice 1. Conversions et opérations en base 2

Répondez aux questions de cet exercice sur une feuille puis vérifiez vos résultats à l'aide de Python.

1. Écrire en base 10 les entiers $(1\ 0000\ 0000)_2$, $-(1\ 1011)_2$ et $(10\ 0100\ 0100)_2$. Vérifiez vos résultats à l'aide de Python.
2. Écrire en base 2 les entiers 103, 600 et -255. Vérifiez vos résultats à l'aide de Python.
3. Sans convertir en base 10, faire les calculs suivants :

$$(1010)_2 + (1100)_2 \qquad (1\ 0111)_2 + (1011\ 0011)_2 \qquad (1100)_2 - (111)_2 \qquad (110)_2 - (1100)_2$$

$$(1\ 1101)_2 \times (1110)_2 \qquad (1001\ 1011)_2 // (1110)_2 \qquad (1001\ 1011)_2 \% (1110)_2$$

Vérifiez vos résultats à l'aide de Python.

Exercice 2. Entiers binaires en Python

Dans cet exercice, un entier naturel $n \in \mathbb{N}$ est représenté par une liste notée $L(n)$ contenant les bits de la représentation binaire, en commençant par le bit de poids faible. Pour garantir l'unicité de cette représentation, le dernier élément de $L(n)$ est nécessairement un `1` sauf pour $n = 0$ (dans ce cas, $L(0)$ est égal à `[0]`). Par exemple :

n	$11 = (1011)_2$	$157 = (1001\ 1101)_2$	$0 = (0)_2$
$L(n)$	`[1,1,0,1]`	`[1,0,1,1,1,0,0,1]`	`[0]`

Un entier $n \in \mathbb{N}$ sera également représenté par une chaîne de caractères notée $s(n)$ de la forme `"0b" + t` où `t` est la représentation binaire de n, en commençant par le bit de poids fort. En Python, $s(n)$ s'obtient grâce à la commande `bin(n)`. Par exemple :

n	$11 = (1011)_2$	$157 = (1001\ 1101)_2$	$0 = (0)_2$
$s(n)$	`"0b1011"`	`"0b10011101"`	`"0b0"`

Notez que dans $s(n)$ le premier bit est le bit de poids fort, alors que dans $L(n)$ le premier bit est le bit de poids faible.

Conversion entre les différentes représentations

1. À l'aide de la fonction `bin`, écrire une fonction `int_to_list` qui prend en entrée n et renvoie $L(n)$. Vérifiez que les listes obtenues contiennent des entiers et non des chaînes de caractères.
2. (question facultative) Écrire une fonction `list_to_str` qui prend en entrée $L(n)$ et renvoie $s(n)$.
3. (question facultative) Écrire une fonction `list_to_int` qui prend en entrée $L(n)$ et renvoie n.

Nombre de bits dans la représentation binaire

Le nombre de bits dans la représentation binaire de $n \in \mathbb{N}^*$ est le plus grand entier $m \in \mathbb{N}^*$ tel que $2^{m-1} \leqslant n$. Par ailleurs, on considère qu'il y a un seul bit dans la représentation binaire de $n = 0$.

4. À l'aide d'une boucle `while`, écrire une fonction `nb_bits` qui prend en entrée un entier `n` et renvoie le nombre de bits dans sa représentation binaire. Vous devez utiliser la caractérisation donnée ci-dessus, en particulier vous ne devez pas construire $L(n)$ ou $s(n)$. Vérifiez que :

`n`	0	1	7	8	9	11	157	255	256
`nb_bits(n)`	1	1	3	4	4	4	8	8	9

Opérations en base 2

Dans cette partie, les fonctions doivent être écrites sans convertir de la base 2 vers la base 10. Si besoin, on pourra utiliser les indications à la fin de l'énoncé.

5. (a) Écrire une fonction `add` qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et renvoie $L(n_1 + n_2)$.
   (b) À l'aide de la fonction `int_to_list` (mais sans utiliser `list_to_str` ou `list_to_int`), testez la fonction `add` pour 100 000 valeurs de n_1 et n_2 comprises entre 0 et 100 000. Si votre fonction s'exécute instantanément, c'est qu'il y a un problème (l'exécution doit durer quelques secondes).
6. (a) Écrire une fonction `soust` qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et renvoie $L(n_1 - n_2)$. On pourra supposer que $n_1 \geqslant n_2$.
   (b) Répondre à la question 5b dans le cas de la fonction `soust`.
7. (a) Écrire une fonction `mult` qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et renvoie $L(n_1 \times n_2)$.
   (b) Répondre à la question 5b dans le cas de la fonction `mult`.
8. (a) Écrire une fonction `div` qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et renvoie les deux listes représentant le reste et le quotient dans la division euclidienne de n_1 par n_2.
   (b) Répondre à la question 5b dans le cas de la fonction `div` en ajoutant la condition $n_2 > 0$.
9. (a) À l'aide d'une exponentiation rapide, écrire une fonction `expo` qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et renvoie $L(n_1^{n_2})$.
   (b) Testez la fonction `expo` pour 200 valeurs de n_1 et n_2 comprises entre 0 et 200.

Indications (essayez de résoudre l'exercice sans lire ce qui suit).

On pourra commencer par écrire des fonctions intermédiaires :

- ⋆ Des fonctions `add_bis` et `sous_bis` qui fonctionnent dans le cas où `L1` et `L2` sont de même taille. Pour les fonctions `add` et `sous`, il suffira ensuite d'appeler `add_bis` ou `sous_bis` en ajoutant des 0 à la fin de $L(n_1)$ ou de $L(n_2)$.
- ⋆ Une fonction `suppr_zeros` qui prend en entrée une liste `L` non vide de `0` et de `1`, et supprime les `0` à la fin de `L` pour que le dernier élément soit un `1` ou bien que la liste devienne `[0]`. Par exemple :

`L`	`[0,1,0,1,1,1]`	`[0,0,0,0,0]`	`[0]`	`[0,1,0,1,0]`	`[0,1,0,0,1,0,0,0]`
`suppr_zero(L)`	`[0,1,0,1,1,1]`	`[0]`	`[0]`	`[0,1,0,1]`	`[0,1,0,0,1]`

  Cette fonction sera utilisée dans les fonctions `sous` et `div`.
- ⋆ Une fonction `leq` ("less or equal") qui prend en entrée deux listes $L(n_1)$ et $L(n_2)$, et indique si $n_1 \leqslant n_2$. Cette fonction sera utilisée dans la fonction `div`

Exercice 3. Représentation de Zeckendorf

Dans cet exercice, on s'intéresse à la célèbre suite de Fibonacci qu'on utilisera pour encoder des nombres, puis pour implémenter une exponentiation rapide. La suite de Fibonacci $(\mathcal{F}_k)_{k\in\mathbb{N}}$ est définie par :

$$\begin{cases} \mathcal{F}_0 = 1, \mathcal{F}_1 = 2 \\ \mathcal{F}_{k+2} = \mathcal{F}_{k+1} + \mathcal{F}_k \qquad \text{pour tout } k \geqslant 0 \end{cases}$$

Dans la suite, on dira qu'une somme d'entiers est ***valide*** si elle vérifie quatre conditions :

→ Chaque terme de la somme est un élément de la suite de Fibonacci.

→ Ces termes sont strictement décroissants.

→ Si on note $a \in \mathbb{N}^*$ la valeur de la somme et k l'entier tel que $\mathcal{F}_k \leqslant a < \mathcal{F}_{k+1}$, alors $\mathcal{F}_k$ apparaît dans la somme.

→ Pour tout $k \in \mathbb{N}$, si $\mathcal{F}_k$ apparaît dans la somme alors $\mathcal{F}_{k+1}$ n'y apparaît pas.

On admet que tout entier strictement positif $a \in \mathbb{N}^*$ peut s'écrire comme une unique somme valide. Par exemple, avec $a = 40$, on obtient :

$$a = 40 = 34 + 5 + 1 = \mathcal{F}_7 + \mathcal{F}_3 + \mathcal{F}_0.$$

À chaque entier $a \in \mathbb{N}^*$ on associe une chaîne de caractères `s` contenant une suite de `0` et de `1` appelée ***représentation de Zeckendorf***. Pour obtenir `s`, on écrit a comme une somme valide, puis on définit `s` comme étant la chaîne de caractères telle que :

→ Si $\mathcal{F}_k$ apparaît dans la somme alors `s[`k`]` vaut `"1"`.

→ Sinon, `s[`k`]` vaut `"0"`.

Afin de garantir l'unicité, on impose que le dernier caractère de `s` soit un `1`. Par exemple, la représentation de Zeckendorf de $a = 40$ est `"10010001"`.

1. Écrire une fonction `zeckToInt` qui prend en entrée une chaîne de caractères `s` et renvoie l'entier a dont `s` est la représentation de Zeckendorf.
2. Écrire une fonction `intToZeck` qui prend en entrée un entier et renvoie sa représentation de Zeckendorf.
3. Sans utiliser l'opérateur `**`, écrire une fonction `puiss` qui prend en entrée une chaîne de caractères `s` contenant la représentation de Zeckendorf d'un entier $a \in \mathbb{N}^*$, ainsi qu'un flottant $x \in \mathbb{R}$, et renvoie x^a. Votre fonction devra manipuler directement `s`, sans jamais calculer explicitement la valeur de a.

Exercice à rendre au plus tard le 02/03/2025 à 20h

Exercice 4. Générateurs pseudo-aléatoires

Un générateur pseudo-aléatoire est un algorithme qui génère des entiers $X_0, X_1, \ldots, X_{n-1}$ qui semblent avoir été tirés au hasard. En Python, ils seront stockés dans une liste `X` de taille n.

Exemple. La fonction `randint` du module `random` de Python est un générateur pseudo-aléatoire : un appel à `random.randint(a,b)` renvoie un entier aléatoire entre `a` et `b` inclus.

1. Écrire une fonction `alea1` qui prend en entrée l'entier n ainsi que a et b, et renvoie une liste `X` composée de n éléments de $[\![a, b]\!]$ générés par la fonction `randint` du module `random`. On pourra supposer que $a \leqslant b$.

Répartition des chiffres. On rappelle qu'un chiffre est un élément de $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ et qu'un nombre est un élément de $\mathbb{N}$. En Python, les fonctions `str` et `int` permettent de convertir un entier en chaîne de caractères et inversement. Par exemple `str(102030)` vaut `"102030"` et `int("102030")` vaut `102030`.

Dans cette partie, on va calculer les fréquences d'apparitions de chaque chiffre dans les nombres présents dans une liste « `X: list[int]` ». Par exemple, si `X = [0, 10, 3455, 9, 75]` alors il y a 10 chiffres présents dans `X`. Comme 5 y apparaît 3 fois, sa fréquence d'apparition est 0.3.

2. Écrire une fonction `XToStat` qui prend en entrée `X` (supposée non vide) et renvoie une liste « `stat: list[float]` » de taille 10 telle que `stat[i]` contient la fréquence d'apparition du chiffre `i` dans `X`. Par exemple :

`X = [0, 10, 3455, 9, 75]` ⇝ `stat = [0.2, 0.1, 0, 0.1, 0.1, 0.3, 0, 0.1, 0, 0.1]`.

Lorsqu'on génère la liste `X` à l'aide de `random.randint(0,9999)`, tous les chiffres à l'exception du chiffre `0` sont censés apparaître avec la même fréquence. La moyenne et l'écart type de n nombres $x_1, \ldots, x_n$ sont définis par :

$$\overline{x} = \frac{1}{n}\Big(x_1 + \ldots + x_n\Big) \qquad \text{et} \qquad \sigma = \sqrt{\frac{1}{n}\Big((x_1 - \overline{x})^2 + \ldots + (x_n - \overline{x})^2\Big)}$$

3. (a) Écrire une fonction `moy1a9` qui prend en entrée la liste `stat` et renvoie la moyenne des fréquences des chiffres de `1` à `9` (`0` n'est pas pris en compte). Par exemple :

`moy1a9(XToStat([0,10,3455,9,75]))` vaut `0.0888888...`

(b) Écrire une fonction `ET1a9` qui prend en entrée la liste `stat` et renvoie l'écart type des fréquences des chiffres de `1` à `9`. Par exemple :

`ET1a9(XToStat([0,10,3455,9,75]))` vaut `0.0874...`

Générateurs congruentiels linéaires. Un ***générateur congruentiel linéaire*** est défini par quatre entiers $a \in \mathbb{N}^*, c \in \mathbb{N}^*, m \in \mathbb{N}^*$ et $X_0 \in \mathbb{N}$ où X_0 est appelée la ***graine***. Pour (a, c, m, X_0) fixés, la suite $(X_k)_{k\in\mathbb{N}}$ est définie par :

$$X_{k+1} = (aX_k + c) \mod m.$$

Par exemple, pour $a = 4$, $c = 1$, $m = 10$ et $X_0 = 2$, on obtient :

$$X_0 = 2, \qquad X_1 = 9, \qquad X_2 = 7, \qquad X_3 = 9, \qquad X_4 = 7, \qquad X_5 = 9, \qquad \ldots$$

4. Écrire une fonction `alea2` qui prend en entrée les entiers n, a, c, m et X_0, et renvoie la liste contenant les entiers X_0, X_1, $\ldots X_{n-1}$.

5. Dans cette question :

   → La suite $(X_k)_{k\in\mathbb{N}}$ est issue d'un générateur congruentiel linéaire et on admet que cela implique qu'elle est périodique à partir d'un certain rang. Soit p la période associée. Par exemple pour la suite définie juste avant la question 4, on a $p = 2$.

   → On suppose disposer d'une fonction « `alea3() -> int` » telle que le premier appel à `alea3()` renvoie X_0, le deuxième appel à renvoie X_1, le troisième renvoie X_2, et ainsi de suite.

   Écrire une fonction `periode` (sans argument) de **complexité linéaire** qui renvoie p. On pourra tester avec :

```
def get_alea3(a, c, m, x0):
    x = x0
    def alea3():
        nonlocal x
        old_x = x
        x = (a*x+c) % m
        return old_x
    return alea3

alea3 = get_alea3( ... )
```

a	c	m	X_0	`periode()`
4	1	10	2	2
137	187	2^8	123	256
16807	0	$2^{17} - 1$	7	13107

Générateur de Von Neumann (facultatif). Le ***générateur de Von Neumann*** permet de générer des nombres avec 10 chiffres, c'est à dire des nombres appartenant à $[\![0, 10^{10} - 1]\!]$. Étant donné X_k, l'entier X_{k+1} est composé des dix chiffres du milieu de X_k^2. Par exemple, si $X_0 = 145$ alors (les espaces dans les nombres ci-dessous n'ont pas de signification mathématique, ils servent à faciliter la lecture) :

$$X_0^2 = 21\ 025, \qquad X_1 = 21\ 025,$$
$$X_1^2 = 442\ 050\ 625, \qquad X_2 = 442\ 050\ 625,$$
$$X_2^2 = 1954\ 08755\ 06289\ 0625, \qquad X_3 = 08755\ 06289 = 8755\ 06289,$$
$$X_3^2 = 7665\ 11262\ 07855\ 1521, \qquad X_4 = 11262\ 07855,$$
$$X_4^2 = 12683\ 44132\ 66370\ 1025, \qquad X_5 = 44132\ 66370.$$

En particulier, lorsque la taille de X_k^2 est impaire et supérieure à 10, on obtient X_{k+1} en enlevant un chiffre de plus à gauche qu'à droite.

6. Écrire une fonction `alea4` qui prend en entrée l'entier n ainsi que la graine X_0, et renvoie la liste contenant les entiers X_0, X_1, $\ldots X_{n-1}$.