Exercice 1. Représentation des nombres

Question 1.a – On a :

$$88 = 64 + 16 + 8 = \boxed{(101\ 1000)_2}$$

Question 1.b – On a :

$$(1001\ 0110)_2 = 2 + 4 + 16 + 128 = \boxed{150}$$

Question 2 –

⋆ Pour l'addition :

	1	1	0	1	1	1	0	1
+		1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	0

Donc $(1101\ 1101)_2 + (101\ 0101)_2 = \boxed{(1\ 0011\ 0010)_2}$.

⋆ Pour la soustraction :

	1	0	0	1	0	1	1	0
-		1	1	1	1	0	0	1
	0	0	0	1	1	1	0	1

Donc $(1001\ 0110)_2 - (111\ 1001)_2 = \boxed{(1\ 1101)_2}$.

⋆ Pour la multiplication :

					1	1	1	0	1	1
				×			1	1	0	1
					1	1	1	0	1	1
+			1	1	1	0	1	1	0	0
+		1	1	1	0	1	1	0	0	0
	1	0	1	1	1	1	1	1	1	1

Donc $(11\ 1011)_2 \times (1011)_2 = \boxed{(10\ 1111\ 1111)_2}$.

Question 3.a –

	1	1	1	1	1	0	1	0	1	0	1			
-	1	0	1						1	1	0	0	1	0
	0	1	0	1	1	0	1	0						
-		1	0	1										
		0	0	0	1	0	1	0						
		-			1	0	1							
					0	0	0	0						

Ainsi, dans la division euclidienne de $(1111\ 1010)_2$ par $(101)_2$:

$$\boxed{\text{le quotient est } (11\ 0010)_2 \text{ et le reste est } (0)_2}.$$

Question 3.b – On a :

$$(1111\ 1010)_2 = 2+8+16+32+64+128 = 250 \qquad \text{et} \qquad (101)_2 = 5$$

La division euclidienne de 218 par 5 donne : $250 = 5 \times 50 + 0$. De plus :

$$50 = 32+16+2 = (11\ 0010)_2 \qquad \text{et} \qquad 0 = (0)_2$$

Les valeurs obtenues sont cohérentes avec la question précédente.

Question 4.a – Soit $n = -143 < 0$. Par définition, la représentation complément à deux de n est la représentation binaire de $n + 2^{64}$. On a :

$$143 = 128+8+4+2+1 = (1000\ 1111)_2$$

Calculons $2^{64} + n = (1\ \underbrace{0\ldots0}_{64\text{ zéros}})_2 - (1000\ 1111)_2$:

	1	0	…	0	0	0	0	0	0	0	0	0	0
-						1	0	0	0	1	1	1	1
	0	1	…	1	1	0	1	1	1	0	0	0	1

Finalement la représentation complément à deux de -143 est $(\underbrace{1\ldots1}_{56\text{ uns}}\ 0111\ 0001)$

Question 4.b – Soit n l'entier dont la représentation complément à deux est $(\underbrace{1\ldots1}_{54\text{ uns}}01\ 1010\ 1001)$. Puisque le bit de poids fort de la représentation complément à deux est 1, on a $n < 0$ et $n+2^{64} = (\underbrace{1\ldots1}_{54\text{ uns}}01\ 1010\ 1001)_2$.

Ainsi :

$$-n = 2^{64} - (\underbrace{1\ldots1}_{54\text{ uns}}01\ 1010\ 1001)_2 = (1\ \underbrace{0\ldots0}_{64\text{ zéros}})_2 - (\underbrace{1\ldots1}_{54\text{ uns}}01\ 1010\ 1001)_2$$

Posons cette soustraction :

	1	0	…	0	0	0	0	0	0	0	0	0	0	0
-		1	…	1	1	0	1	1	0	1	0	1	0	0	1
	0	0	…	0	0	1	0	0	1	0	1	0	1	1	1

Ainsi :

$$-n = (10\ 0101\ 0111)_2 = 1+2+4+16+64+512 = 599$$

Donc $\boxed{n = -599}$

Exercice 2. Représentation de Zeckendorf

Question 1 – On a :

$$\mathcal{F}_0 = 1, \qquad \mathcal{F}_1 = 2, \qquad \mathcal{F}_2 = 3, \qquad \mathcal{F}_3 = 5, \qquad \mathcal{F}_4 = 8,$$
$$\mathcal{F}_5 = 13, \qquad \mathcal{F}_6 = 21, \qquad \mathcal{F}_7 = 34, \qquad \mathcal{F}_8 = 55, \qquad \mathcal{F}_9 = 89.$$

Question 2.a –

```
1  def fibo(k):
2      """fibo(k: int) -> list[int]"""
3      assert k >= 0
4      if k == 0:
5          return [1]
6      L = [1,2]
7      for _ in range(k-1):
8          L.append(L[-2] + L[-1])
9      return L
```

Question 2.b – Pour `k` $\leqslant$ `0`, l'appel à la fonction `fibo` termine. Pour `k` $\geqslant$ `1`, la boucle `for` effectue `k-1` tours et toutes les opérations utilisées dans la fonction sont élémentaires et terminent.

Donc l'appel à `fibo` termine pour toute entrée `k` $\in \mathbb{Z}$.

Question 2.c –

- Si `k < 0`, la fonction déclenche une erreur, ce qui est le comportement attendu.
- Si `k = 0`, la fonction renvoie la liste `[1]` = $[\mathcal{F}_0]$.
- Si `k = 1`, la boucle `for` fait 0 tour de boucle et la fonction renvoie `[1, 2]` = $[\mathcal{F}_0, \mathcal{F}_1]$.

Il reste donc à montrer que pour `k` $\geqslant$ `2`, la fonction renvoie :

$$[\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_k]$$

On numérote les tours de boucle de `i = 0` à `i = k - 2`. Pour chaque `i` $\in [\![0; k-2]\!]$, on note L_k la valeur de la variable `L` à la fin du tour de boucle numéro `i` et on définit $\mathcal{P}_i$ la propriété :

$$(\mathcal{P}_i) : L_i = [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{i+2}]$$

Montrons $\mathcal{P}_i$ par itération finie sur `i` $\in [\![0; k-2]\!]$.

$\star$ Pour `i = 0`, la valeur de la variable `L` à la fin du premier tour de boucle est (voir les lignes 6 et 8) :

$$L_0 = [1, 2, 3] = [\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2]$$

Donc $\mathcal{P}_0$ est vraie.

$\star$ Soit `i` $\in [\![1; k-2]\!]$. On suppose $\mathcal{P}_{i-1}$ et on montre $\mathcal{P}_i$. D'après $\mathcal{P}_{i-1}$, à la fin du tour de boucle numéro `i - 1`, la variable `L` vaut :

$$L_{i-1} = [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{i+1}]$$

D'après la ligne 8, à la fin du tour de boucle de boucle numéro `i`, la variable `L` vaut :

$$\begin{aligned} L_{i-1} + [L_{i-1}[-2] + L_{i-1}[-1]] &= [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{i+1}] + [\mathcal{F}_i + \mathcal{F}_{i+1}] \\ &= [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{i+1}] + [\mathcal{F}_{i+2}] \\ &= [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_{i+1}, \mathcal{F}_{i+2}] \end{aligned}$$

D'où $\mathcal{P}_i$.

$\star$ D'après $(\mathcal{P}_{k-2})$, à la fin du dernier tour de boucle, la variable `L` vaut :

$$L_{k-2} = [\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_k].$$

La liste renvoyée est bien celle attendue, d'où la correction.

Question 2.d –

⋆ Pour la complexité spatiale : la fonction `fibo` manipule des entiers et une liste `L` contenant `k+1` entiers à la fin de l'exécution. La complexité temporelle est donc en Θ`(k)`.

⋆ Pour la complexité temporelle : pour `k` $\geqslant$ `1`, la boucle `for` effectue `k - 1` tours et chaque ligne de la fonction s'exécute en temps constant. Ainsi, le temps d'exécution est en Θ`(k)`.

Question 3 – On a :

$$60 = 55 + 5 = \mathcal{F}_8 + \mathcal{F}_3.$$

Question 4 – En se basant sur la question précédente, la représentation de Zeckendorf de 60 est `"000100001"`.

Question 5 –

```
def zeckToInt(s):
    res = 0
    F = fibo(len(s)-1)
    for k in range(len(s)):
        if s[k] == '1':
            res += F[k]
    return res
```

Question 6.a –

```
# Pour obtenir la complexité demandée, on ne peut pas utiliser la
# fonction "fibo".
def tailleZeck(a):
    k = 1
    Fk = 1    # = F_{k}
    Fkp1 = 2  # = F_{k+1}
    while Fkp1 <= a:
        k += 1
        Fk, Fkp1 = Fkp1, Fk + Fkp1
    return k
```

Question 6.b – Remarque : par le troisième point de la définition d'une somme valide, si on note k l'entier tel que $\mathcal{F}_k \leqslant a < \mathcal{F}_{k+1}$, alors $\mathcal{F}_k$ apparaît dans la somme. Si $a \neq \mathcal{F}_k$, alors une somme valide pour a (qui existe d'après l'énoncé) est de la forme $a = \mathcal{F}_k + S$ avec S une somme valide pour $a - \mathcal{F}_k$. Par unicité de la décomposition en somme valide, on en déduit que **la** somme valide pour a est $a = \mathcal{F}_k + S$ avec S **la** somme valide pour $a - \mathcal{F}_k$ (en particulier, dans la fonction `intToZeckendorf`, il n'est pas nécessaire de vérifier explicitement le $4^{\text{ème}}$ point de la définition d'une somme valide).

```
def intToZeck(a):
    res = ""
    l = tailleZeck(a)
    F = fibo(l-1)
    for k in range(len(F)-1, -1, -1):
        if a >= F[k]:
            a = a-F[k]
            res = "1" + res
        else:
            res = "0" + res
    return res
```

Question 7.a –

```
def AtoCF(A):
    res = ""
    for a in A:
        res += intToZeck(a) + "1"
    return res
```

Question 7.b – On remarque que dans la chaîne de caractères `codeFibo`, deux `"1"` consécutifs signifie qu'on passe de la représentation d'un entier a_i à celle de a_{i+1}.

```
# Les indices deb et fin representent le debut et la fin d'une
# representation de Zeckendorf.
def CFtoA(codeFibo):
    A = []
    deb = 0
    while deb < len(codeFibo):
        fin = deb
        while codeFibo[fin] == "0" or codeFibo[fin + 1] == "0":
            fin += 1
        A.append(zeckToInt(codeFibo[deb:fin+1]))
        deb = fin + 2
    return A
```

Question 8.a – La représentation de Zeckendorf de a décrit comment écrire a comme une somme valide. Elle est donc représentée par un ensemble $A \subset \mathbb{N}$ tel que :

$$a = \sum_{k \in A} \mathcal{F}_k.$$

On obtient donc :

$$x^a = \prod_{k \in A} x^{\mathcal{F}_k}$$

Pour calculer les $x^{\mathcal{F}_k}$ on utilisera la formule de récurrence suivante :

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

```
### Solution 1. Dans cette fonction:
### xfimu vaut x**((i-1)eme terme de la suite de fibonacci)
### xfi vaut x**(ieme terme de la suite de fibonacci)
def puiss(x,s):
    if s == "1":
        return x
    xfimu = x; xfi = x*x
    res = 1
    if s[0] == "1":
        res *= x
    if s[1] == "1":
        res *= xfi
    for i in range(2,len(s)):
        xfimu, xfi = xfi, xfimu*xfi
        if s[i] == "1":
            res *= xfi
    return res
```

```
### Solution 2. Deuxieme solution plus courte pour laquelle on a posé F_{-2} = 0
### et F_{-1} = 1.
def puiss_bis(x,s):
    res = 1
    xfimu = 1; xfi = x
    for i in range(len(s)):
        xfimu, xfi = xfi, xfimu*xfi
        if s[i] == "1":
            res *= xfi
    return res
```

Question 8.b – Le temps d'exécution est clairement en Θ(`len(s)`). Il s'agit donc d'exprimer `len(s)` en fonction de a. D'après la définition de "somme valide" donnée dans l'énoncé, le plus grand terme $\mathcal{F}_k$ apparaissant dans la somme vérifie $\mathcal{F}_k \leqslant a < \mathcal{F}_{k+1}$. On a donc `len(s)` $= k+1$. Par une récurrence immédiate sur $k \in \mathbb{N}$, on montre que pour tout $k \in \mathbb{N}$:

$$\phi^{k-1} < \mathcal{F}_k < \phi^k$$

avec $\phi = \dfrac{1+\sqrt{5}}{2}$ (dans l'hérédité de la récurrence, on utilise la relation $\phi^2 = \phi + 1$). Ainsi :

$$k - 1 < \frac{\log_2(\mathcal{F}_k)}{\log_2(\phi)} < k$$

Donc :

$$\frac{\log_2(a)}{\log_2(\phi)} - 1 < k < \frac{\log_2(a)}{\log_2(\phi)} + 1$$

Finalement, $k = \Theta(\log_2(a))$ et le temps d'exécution de la fonction `puiss` est en $\Theta(\log_2(a))$.

Exercice 3. Plouf-plouf

Question 1 –

```
# k est l'indice du dernier enfant vu
# elim[i] vaut True lorsque l'enfant numéro i a été éliminé
# On élimine tous les enfants, le dernier éliminé est le capitaine.
def capitaine(n,m):
    """
    n, m: int
    Return: int
    """
    elim = [False]*n
    k = -1
    for _ in range(n):
        nb_vus = 0
        while nb_vus < m:
            k = (k + 1) % n
            if not elim[k]:
                nb_vus += 1
        elim[k] = True
        # print(k, end=' ')
    return k
```

Question 2.a – Montrons par récurrence sur $k \in \mathbb{N}$ que $\boxed{\text{le capitaine sera l'enfant numéro } 0}$:

- Si $k = 0$ alors $n = 1$ et la propriété est vraie.
- Étant donné $k \in \mathbb{N}$, on suppose la propriété vraie au rang k et on la montre au rang $k+1$. Lorsque $m = 2$ et $n = 2^{k+1}$, les enfants éliminés lors du premier tour sont ceux dont les numéros sont 1, 3, 5, …, $2^{k+1} - 1$. Il y a donc 2^k éliminations lors du premier tour, ce qui laisse $2^{k+1} - 2^k = 2^k$ enfants en lice. On recommence ensuite les éliminations en comptant à partir de l'enfant numéro 0. Ainsi, par hypothèse de récurrence, l'enfant qui deviendra capitaine est celui dont le numéro est 0.

Question 2.b – On écrit n sous la forme $n = 2^k + \ell$ où $0 \leqslant \ell < 2^k$. Les numéros des ℓ premiers enfants éliminés sont 1, 3, …, $2\ell - 1$. Après ces éliminations, il reste 2^k enfants et on commence à compter à partir de l'enfant numéro 2ℓ. Remarquez que 2ℓ est un numéro valide car $2\ell = \ell + \ell < 2^k + \ell = n$. D'après la question précédente, le capitaine sera l'enfant dont le numéro est 2ℓ.

On nous demande d'utiliser l'écriture en base 2 de n.

- L'écriture en base 2 de ℓ est l'écriture en base 2 de n où le bit de poids fort (qui vaut 1) a été supprimé.
- L'écriture en base 2 de 2ℓ est l'écriture en base 2 de ℓ où un 0 a été ajouté à droite.

En conclusion, pour obtenir l'écriture en base 2 du numéro du capitaine, on écrit n en base 2, on supprime le un qui se trouve à gauche et on ajoute un zéro à droite.

Question 2.c –

```
def capitaine2(n):
    """capitaine2(n: int) -> int"""
    n_base2 = bin(n)[2:]
    c_base2 = n_base2[1:] + "0"
    return int(c_base2, base = 2)
```