Exercice 1. Suite de Fibonacci

La suite de Fibonacci $(F_n)_{n\geqslant 0}$ est définie récursivement : $\begin{cases} F_0 = 0, F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n & \text{pour tout } n \geqslant 0. \end{cases}$

1. Sur feuille, calculer F_8.
2. Écrire une fonction `fibo` qui prend en entrée un entier n et renvoie la valeur de F_n. Vérifier que $F_0 = 0$, $F_1 = 1$, $F_{10} = 55$ et $F_{20} = 6765$.

Exercice 2. Plus petit commun multiple

Soit $a \in \mathbb{N}^*$ et $b \in \mathbb{N}^*$ deux entiers. Le plus petit commun multiple (PPCM) de a et b est le plus petit entier m tel que m soit un multiple de a et un multiple de b. Ce nombre peut être calculé à l'aide de la fonction `lcm` du module `math`, mais le but de cet exercice est de réécrire cette fonction à l'aide d'une boucle `while`. L'idée est d'initialiser une variable m à 1 et d'incrémenter m tant que m n'est pas le PPCM de a et b.

1. Écrire une fonction `PPCM` qui calcule le PPCM de ses paramètres d'entrée `a` et `b`. Si `a` ou `b` est négatif ou nul, votre fonction déclenchera une erreur. Par exemple :

   `PPCM(0, 1)` déclenche une erreur, `PPCM(1, 0)` déclenche une erreur,
   `PPCM(1, 1)` vaut `1`, `PPCM(1, 123)` vaut `123`,
   `PPCM(4, 6)` vaut `12`, `PPCM(17, 204)` vaut `204`,
   `PPCM(179, 64)` vaut `11456`.

2. En utilisant la fonction `randint` du module `random`, écrire une fonction `test_PPCM` (sans argument) qui tire aléatoirement 1000 couples `(a,b)` avec `a` et `b` compris entre 1 et 1000, et renvoie `True` si les fonctions `PPCM` et `math.lcm` renvoient les mêmes résultats pour ces 1000 couples. Si votre fonction s'exécute instantanément, c'est qu'il y a un problème (elle doit prendre quelques secondes avant de renvoyer `True`).

Exercice 3. Approximation de la fonction cosinus

Soit $x \in \mathbb{R}$ fixé. Le réel $\cos(x)$ peut être défini par $\cos(x) = \lim_{n\to+\infty} u_n$ où pour tout $n \in \mathbb{N}$: $u_n = \sum_{k=0}^{n} \frac{(-1)^k}{(2k)!} x^{2k}$.

1. Écrire une fonction `get_u` qui prend en arguments un flottant x ainsi qu'un entier n et renvoie la valeur de u_n. Vous pouvez utiliser la fonction `factorial` du module `math`. Par exemple, à l'aide de la constante `pi` et de la fonction `sqrt` du module `math`, vérifiez que :

   Pour $x = 0$: $u_0 = u_1 = 1$,
   Pour $x = \pi/4$: $u_1 \approx 0.691\,574\ldots$, $u_{10} \approx \sqrt{2}/2$,
   Pour $x = 2$: $u_3 \approx -0.422\,22\ldots$, $u_{15} \approx -0.416\,146\ldots$

2. Écrire une fonction `seuil` qui prend en entrée deux flottants x et ϵ, et renvoie le plus petit entier $n \in \mathbb{N}$ tel que $|u_n - cos(x)| \leqslant \epsilon$. Pour cette question, utilisez la fonction `cos` du module `math` et la fonction `abs` (qui n'appartient pas au module `math`). Par exemple, vérifier que :

   La fonction renvoie 0 pour $x = 0$ et $\epsilon = 10^{-10}$,
   La fonction renvoie 3 pour $x = \pi/6$ et $\epsilon = 10^{-6}$,
   La fonction renvoie 18 pour $x = -10$ et $\epsilon = 10^{-6}$.

Exercice 4. Approximation de π

Le but de cet exercice est d'approcher le nombre π à l'aide d'un processus aléatoire. On considère un repère orthonormé dans le plan, ainsi qu'un disque $\mathcal{D}$ de rayon 2 centré sur l'origine du repère et $\mathcal{C}$ le carré plein $[-2,2] \times [-2,2]$.

1. Sur une feuille de papier, dessiner $\mathcal{C}$ et $\mathcal{D}$ et calculer leurs aires.
2. On choisit un point aléatoire dans $\mathcal{C}$. Quelle est la probabilité p_0 que ce point appartienne à $\mathcal{D}$?

Afin de calculer une approximation de π, on approche p_0 de la manière suivante : on choisit aléatoirement n points dans $\mathcal{C}$ et on approche p_0 par m/n où m est le nombre de points appartenant à $\mathcal{D}$. Pour information, cette méthode pour estimer la valeur de π est très lente. Il existe des méthodes beaucoup plus rapides.

3. À l'aide de la fonction `uniform` du module `random`, écrire une fonction `point_aleatoire` qui renvoie un couple (x, y) représentant l'abscisse et l'ordonnée d'un point choisi aléatoirement et uniformément dans $\mathcal{C}$.
4. Écrire un fonction `est_dans_D` qui prend en entrée un point `p`, et renvoie `True` si `p` appartient à $\mathcal{D}$ ou `False` sinon. Le point `p` sera un tuple `(x,y)` représentant l'abscisse et l'ordonnée du point. Votre fonction ne doit prendre qu'un argument `p` qui est un couple, pas deux arguments de type `float`.
5. Implémenter une fonction `approx_pi` qui prend en entrée un entier `n` et renvoie une approximation de π en utilisant la méthode décrite ci-dessus avec `n` points aléatoires.
6. Donner une fonction `diff_approx_pi` qui prend en entrée un entier `n` et renvoie la valeur absolue de la différence entre `approx_pi(n)` et la constante `pi` du module `math`. Vérifiez que `diff_approx_pi(n)` s'approche de `0` quand `n` est grand.

Exercice 5. Suite de Conway

La suite de Conway est la suite $(C_n)_{n \geqslant 0}$ dont le premier terme est $C_0 = 1$ et telle que C_{n+1} s'obtient en lisant les chiffres de C_n à haute voix (ce processus est appelé “look-and-say”). Par exemple :

- Le terme $C_0 = 1$ contient un 1 donc $C_1 = 11$.
- Le terme C_1 contient deux 1 donc $C_2 = 21$.
- Le terme C_2 contient un 2 et un 1 donc $C_3 = 1211$.

On a ainsi :

$$C_0 = 1, \qquad C_1 = 11, \qquad C_2 = 21, \qquad C_3 = 1211, \qquad C_4 = 111221, \qquad C_5 = 312211, \qquad \ldots$$

1. Écrire une fonction `look_and_say` qui prend en entrée un entier naturel C et renvoie l'entier C' obtenu en appliquant à C le processus look-and-say décrit ci-dessus. En particulier, pour tout $n \geqslant 0$, `look_and_say`(C_n) doit renvoyer C_{n+1}. Vérifiez que :

   `look_and_say(10112111311114111111)` vaut `111021123113411451`.

   Proposez deux autres tests pour votre fonction.
2. Implémenter une fonction `valeur_Conway` qui prend en entrée un entier naturel n, et renvoie la valeur de C_n. Testez avec les valeurs C_0, C_1, C_2, C_3, C_4 et C_5 données ci-dessus et vérifiez que $C_{10} = 11131221133112132113212221$.

Pour tout $n \geqslant 0$, soit L_n le nombre de chiffres de C_n et soit $u_n = L_{n+1}/L_n$, alors la suite $(u_n)_{n \geqslant 0}$ converge vers une valeur appelée ***constante de Conway***.

3. Écrire une fonction `approx_cst_Conway` qui prend en entrée un entier naturel n, et renvoie la valeur de u_n. Vérifiez que :

$$u_0 = 2 \qquad u_1 = 1 \qquad u_2 = 2$$
$$u_{10} \approx 1.307\,692\ldots \qquad u_{30} \approx 1.306\,129\ldots \qquad u_{60} \approx 1.303\,592\ldots.$$

Dans la suite, on modifie la valeur de C_0, mais la valeur C_{n+1} est toujours obtenue à partir de C_n grâce au processus look-and-say.

4. Sur une feuille de papier, trouver l'unique valeur de C_0 pour laquelle la suite $(C_n)_{n\geqslant 0}$ est constante.
5. Choisir cinq valeurs différentes pour C_0 et pour chacune de ces valeurs, calculer une approximation de la constante de Conway comme dans la question 3. Que remarquez vous ?

Exercice 6. Classes et objets

Les notions de classe et d'objet. En Python, il est possible de définir des classes (sorte de nouveau type) et des objets de cette classe. Le but n'est pas de calculer plus de choses qu'on ne pourrait le faire sans la notion de classe, mais d'organiser le code pour qu'il soit plus facile à lire, à écrire et à corriger. Voici une classe pour représenter les nombres complexes :

```
class C():
    def __init__(self, re, im):
        self.re = re
        self.im = im
    def __repr__(self):
        return str(self.re) + "+" + str(self.im) + "i"
    def __add__(self, other):
        return C(self.re + other.re, self.im + other.im)
    def module(self):
        return (self.re**2 + self.im**2)**0.5

z1 = C(4, 1)      # z1 = 4 + i
z2 = C(0, 2)      # z2 = 2i
z = z1+z2         # z = 4 + 3i
print(z1, z2, z)  # Affiche 4+1i, 0+2i et 4+3i
print(z.module()) # Affiche 5.0
```

Dans ce programme, le nom de la classe est « `C` » et trois objets sont définis : « `z1` », « `z2` » et « `z` ». De manière générale, une classe contient des ***attributs*** (des valeurs associées à un objet) et des ***méthodes*** (équivalent d'une fonction). Dans la classe `C`, les deux attributs sont `re` et `im` : ils représentent la partie réelle et la partie imaginaire du nombre complexe. Pour chaque objet `z` de la classe `C`, on peut accéder à ces deux valeurs avec la syntaxe `z.re` et `z.im`. Les quatre méthodes sont :

- ⋆ La méthode `__init__` : c'est cette méthode qui est appelée lors de la création d'un nouvel objet. Par exemple, lorsqu'on créé `z1` à la ligne 12, la méthode `__init__` est appelée avec `re = 4` et `im = 1`. Notez que le nom de la méthode (`__init__`) n'est jamais écrit explicitement, il suffit d'invoquer le nom de la classe (`C`) comme aux lignes 12 et 13.
  Le premier paramètre d'une méthode, usuellement appelé `self`, fait référence à l'objet manipulé. Pour la ligne 12, le paramètre `self` de la méthode `__init__` correspond à la variable `z1`. Les lignes 3 et 4 initialisent donc `z1.re` à `4` et `z1.im` à `1`.
- ⋆ La méthode `__repr__` : elle est notamment appelée par la fonction `print`. À la ligne 6, on fait en sorte qu'un nombre complexe soit affiché sous la forme `a + ib`, d'où le résultat de l'exécution de la ligne 15.
- ⋆ La méthode `__add__` : elle permet de définir l'opérateur `+` sur les objets de la classe. Par exemple, lors de l'exécution de la ligne 14, la méthode `__add__` est appelée avec `self = z1` et `other = z2`. Elle renvoie le nouveau nombre complexe de la ligne 8.
- ⋆ La méthode `module` : elle renvoie le module du nombre complexe. On y accède avec la syntaxe `z.module`.

But de l'exercice. On souhaite écrire une classe `Q` pour représenter les nombres rationnels. Un objet `x` de la classe `C` aura deux attributs `x.num` et `x.denom` représentant respectivement le numérateur et le dénominateur.

`x`	$\frac{1}{3}$	4	$-\frac{2}{5}$
`x.num`	1	4	-2
`x.denom`	3	1	5

Afin de garantir l'unicité de notre représentation :

- On ne s'intéresse qu'à des fractions irréductibles. On rappelle qu'une fraction p/q est sous forme irréductible si p et q sont premiers entre eux. De plus, pour tous entiers a et b, la forme irréductible de a/b est p/q où $p = a/d$ et $q = b/d$ avec d est le PGCD de a et b.
- Le dénominateur est strictement positif.
- Le rationnel 0 a pour numérateur 0 et pour dénominateur 1.

Une fraction qui satisfait les trois conditions ci-dessus est appelée une représentation valide. Par exemple, les fractions suivantes ne sont pas des représentations valides : $\frac{2}{4}$, $\frac{2}{-5}$ et $\frac{0}{3}$. Toutes les méthodes à implémenter doivent renvoyer uniquement des représentations valides.

1. Définir la classe `Q` et ses méthodes `__init__`, `__repr__`, `__add__`, `__neg__` (renvoie l'opposé d'un rationnel), `__mul__` (calcule le produit de deux rationnels), `__sub__` (calcule la différence de deux rationnels), `approx` (renvoie une approximation d'un rationnel sous la forme d'un flottant).
2. À l'aide de la classe `Q`, vérifiez que : $\frac{135}{26}\left(\frac{1}{3}+4\right)^3 - \frac{33}{6} = 417$.
3. Soit $(u_n)_{n\in\mathbb{N}^*}$ la suite définie par $u_n = \sum_{i=1}^{n} \frac{(-1)^i}{i}$. Cette suite s'appelle la « série harmonique alternée ».
   (a) Écrire une fonction `sha` qui prend en entrée un entier n strictement positif et renvoie le rationnel u_n. Votre fonction doit renvoyer un objet de la classe `Q` et non un flottant. Vérifiez que :
   $$u_1 = -1, \qquad u_2 = -1/2, \qquad u_3 = -5/6, \qquad u_4 = -7/12, \qquad u_{10} = -1627/2520.$$
   (b) En utilisant la fonction `log` du module `math`, calculer les valeurs successives de $u_n + ln(2)$. Que remarquez vous ?

Exercices à rendre au plus tard le 05/11/2023 à 20h

Exercice 7. Compréhensions de listes

Dans cet exercice, il est demandé d'utiliser des compréhensions de listes. Si vous ne vous souvenez plus de ce qu'est une compréhension, relisez le cours sur les listes.

1. À l'aide d'une compréhensions de liste, écrire une fonction `repetition_elem` qui prend en entrée une variable `e` ainsi qu'un entier `k`, et renvoie la liste ayant `k` éléments tous égaux à `e`. Vérifiez que :

   `repetition_elem(12345, 0)` vaut `[]`,
   `repetition_elem("bonjour", 3)` vaut `["bonjour", "bonjour", "bonjour"]`,
   `repetition_elem(12, 1)` vaut `[12]`,
   `repetition_elem(-1, 9)` vaut `[-1, -1, -1, -1, -1, -1, -1, -1, -1]`.

2. À l'aide d'une compréhensions de liste, écrire une fonction `diviseurs_impairs` qui prend en entrée un entier naturel n non nul, et renvoie la liste des diviseurs impairs de n. Par exemple :

   `diviseurs_impairs(1)` vaut `[1]`,
   `diviseurs_impairs(2**10)` vaut `[1]`,
   `diviseurs_impairs(3**5)` vaut `[1, 3, 9, 27, 81, 243]`,
   `diviseurs_impairs(2**4 * 3**2 * 5**2)` vaut `[1, 3, 5, 9, 15, 25, 45, 75, 225]`.

Exercice 8. Coefficients binomiaux

Le but de cet exercice est de calculer les coefficients binomiaux à l'aide de deux méthodes différentes : d'abord en utilisant une formule explicite puis avec le triangle de Pascal.

Dans tout l'exercice, vous devez tester vos fonctions

Calcul des coefficients binomiaux avec une formule explicite

Pour $n, k \in \mathbb{N}$, le coefficient binomial $\binom{n}{k}$ (lire "k parmi n") est défini par :

$$\binom{n}{k} = 0 \text{ si } n < k \qquad \text{et} \qquad \binom{n}{k} = \frac{\prod_{i=n-k+1}^{n} i}{\prod_{i=1}^{k} i} \text{ sinon.}$$

On rappelle que $\prod_{i=k}^{\ell} i = k \times (k+1) \times (k+2) \times \ldots \times (\ell - 1) \times \ell$.

De plus, par convention, pour tout $n \in \mathbb{N}$: $\prod_{i=n+1}^{n} i = 1$.

1. Que valent $\binom{6}{3}$, $\binom{4}{1}$, $\binom{5}{0}$ et $\binom{3}{4}$?
2. Écrire une fonction `binomExplicite` qui prend en entrée deux entiers naturels n et k, et renvoie la valeur du coefficient binomial $\binom{n}{k}$ en utilisant la formule ci-dessus. Votre fonction doit renvoyer un `int`, son premier argument doit être n et son second k.

Calcul des coefficients binomiaux à l'aide du triangle de Pascal

Afin de calculer les coefficients binomiaux, on peut utiliser le triangle de Pascal représenté ci-dessous :

n \ k	0	1	2	3	4	5	6	$\cdots$
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Dans le triangle de Pascal, pour tout $n, k \in \mathbb{N}$ où $0 \leqslant k \leqslant n$, le coefficient binomial $\binom{n}{k}$ se trouve sur la ligne d'indice n et sur la colonne d'indice k. Le triangle de Pascal satisfait les conditions suivantes :

- Les cases telles que $k = 0$ ou $n = k$ contiennent un 1.
- La ligne d'indice $n + 1$ se construit à partir de la ligne d'indice n en utilisant la formule suivante :

$$\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}.$$

En d'autres termes, une case du tableau avec $k \neq 0$ et $n \neq k$ est égale à la somme de la case au-dessus à gauche et de la case au-dessus.

3. Quelles sont les lignes d'indices $n = 7$ et $n = 8$ du triangle de Pascal ?

En Python, on représente une ligne du triangle de Pascal à l'aide d'une liste d'entiers. Par exemple, la ligne d'indice $n = 4$ est représentée par `[1, 4, 6, 4, 1]`.

4. Écrire une fonction de signature « `ligneSuivante(L: list[int]) -> list[int]` » telle que si `L` représente la ligne d'indice n pour un certain $n \in \mathbb{N}$, alors la liste renvoyée représente la ligne d'indice $n + 1$. Bien sûr, vous devez utiliser la méthode décrite ci-dessus et non la fonction `binomExplicite`. De plus, votre fonction ne doit pas modifier `L`. Par exemple :

   `ligneSuivante([1, 5, 10, 10, 5, 1])` vaut `[1, 6, 15, 20, 15, 6, 1]`.

5. En utilisant la fonction `ligneSuivante`, écrire une fonction `binomPascal` qui prend en entrée deux entiers naturels n et k tels que $0 \leqslant k \leqslant n$, et renvoie la valeur du coefficient binomial $\binom{n}{k}$. Vous n'avez pas besoin de vérifier la condition $0 \leqslant k \leqslant n$.

Testons nos fonctions. Le but de cette partie est de tester les fonctions précédentes.

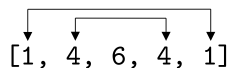
6. Écrire une fonction sans argument `milleComparaisons` qui teste si les valeurs `binomExplicite(n,k)` et `binomPascal(n,k)` sont égales pour mille valeurs de `n` et `k` choisies aléatoirement avec $\texttt{0} \leqslant \texttt{k} \leqslant \texttt{n} \leqslant \texttt{123}$. Votre fonction reverra `True` si les valeurs sont égales pour les mille tests et `False` sinon. Vous pouvez utiliser la fonction `randint` du module `random` qui prend en entrée deux entiers `a` et `b`, et renvoie un entier aléatoire choisi dans $[\![\texttt{a}, \texttt{b}]\!]$.

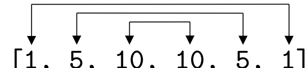
On remarque que si `L` est une liste contenant la ligne d'indice `n` du triangle de Pascal, alors :

(i) La première valeur et la dernière valeur de `L` sont égales à 1.

(ii) Par la formule du binôme de Newton, pour tous réels $\texttt{a}, \texttt{b} \in \mathbb{R}$: $(\texttt{a}+\texttt{b})^{\texttt{n}} = \sum_{\texttt{k}=0}^{\texttt{n}} \texttt{a}^{\texttt{k}}\texttt{b}^{\texttt{n-k}}\texttt{L[k]}$.

(iii) Les valeurs sont symétriques par rapport au milieu de `L` :





(iv) Les valeurs sont strictement croissantes jusqu'au milieu de `L` puis strictement décroissantes.

7. Écrire une fonction `test1` qui prend en entrée une liste `L` non vide, et renvoie `True` si `L` respecte la condition (i) ou `False` sinon.
8. Écrire une fonction `test2` qui prend en entrée une liste `L` non vide ainsi que deux entiers a et b, et renvoie `True` si `L`, a et b respectent la condition (ii) ou `False` sinon.
9. À l'aide d'une boucle `for`, écrire une fonction `test3` qui prend en entrée une liste `L` non vide, et renvoie `True` si `L` respecte la condition (iii) ou `False` sinon.
10. Écrire une fonction `test4` qui prend en entrée une liste `L` non vide, et renvoie `True` si `L` respecte la condition (iv) ou `False` sinon.

Exercice 9. Sous-ensembles de $[\![0; n-1]\!]$ (exercice facultatif)

Écrire une fonction `sous_ensembles` qui prend en entrée deux entiers $(\texttt{n}, \texttt{k}) \in \mathbb{N}^2$, et renvoie une liste contenant toutes les sous-listes triées de `[0, 1, ..., n-1]` de taille `k`. Par exemple avec `n = 4`, on obtient (l'ordre des sous-listes n'a pas d'importance) :

Pour `k = 0` : `[[]]`,
Pour `k = 1` : `[[0], [1], [2], [3]]`,
Pour `k = 2` : `[[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]`,
Pour `k = 3` : `[[0,1,2], [0,1,3], [0,2,3], [1,2,3]]`,
Pour `k = 4` : `[[0,1,2,3]]`,
Pour `k = 5` : `[]`.