Autour des nombres premiers

Merci d'indiquer au début de votre copie si vous avez déjà fait de l'informatique avant d'arriver en MPSI. Si oui, indiquez dans quel cadre. La notation de ce DS est la même pour tout le monde, elle ne sera pas affectée par votre réponse. **En revanche, si vous ne donnez pas de réponse, vous perdrez 1 point.**

Dans tout le sujet, on suppose que les modules `math` et `random` ont été importés grâce aux commandes ci-contre.

```
import math
import random
```

1 Tests de primalité

Définition 1. Un entier $n \in \mathbb{N}$ est appelé un ***nombre premier*** si et seulement si $n \geqslant 2$ et n admet exactement 2 diviseurs positifs distincts : 1 et lui-même.

Exemple 2. 21 n'est pas premier car $21 = 7 \times 3$. De plus, 1 n'est pas premier, mais 2, 3, 5 et 7 le sont.

Dans la première partie, on s'intéresse à quatre ***tests de primalité*** : étant donné un entier $n \in \mathbb{N}$, un test de primalité détermine si n est premier ou non.

1.1 Méthode naïve

On propose dans un premier temps de générer la liste de tous les diviseurs de n, puis de vérifier si elle contient exactement deux éléments distincts.

1. (a) Écrire une fonction de signature `listeDiv(n: int) -> list[int]` qui renvoie la liste des diviseurs positifs de `n`. Votre fonction déclenchera une erreur si `n` $\leqslant$ `0`. Par exemple :

`n`	`0`	`1`	`2`	`10`	`12`	`13`
`listeDiv(n)`	Erreur	`[1]`	`[1, 2]`	`[1, 2, 5, 10]`	`[1, 2, 3, 4, 6, 12]`	`[1, 13]`

   (b) À l'aide de la fonction précédente, écrire une fonction de signature `estPremier1(n: int) -> bool` qui renvoie `True` si `n` est premier et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$. Par exemple :

`n`	`1`	`2`	`10`	`12`	`13`
`estPremier1(n)`	`False`	`True`	`False`	`False`	`True`

2. À l'aide des fonction précédentes, écrire une fonction de signature `listeNP() -> list[int]` qui renvoie la liste des nombres premiers compris entre 1 et 4567. Dans cette question, vous devez utiliser uniquement une compréhension de liste. On obtient : `[2, 3, 5, 7, 11, 13, 17, 19, 23, ...]`.

1.2 Méthode naïve améliorée

Voici deux remarques permettant d'améliorer la méthode naïve de la partie précédente :

- Si un entier $n \in \mathbb{N}^*$ n'est pas premier, alors il peut être décomposé sous la forme $n = pq$ avec $p > 1$ et $q > 1$. Dans ce cas, on a nécessairement $p \leqslant \sqrt{n}$ ou $q \leqslant \sqrt{n}$. Ainsi, pour tester si un entier $n \in \mathbb{N}^*$ est premier, il suffit de vérifier s'il existe un entier $d \leqslant \sqrt{n}$ tel que $d \neq 1$ et d divise n.
- Soit $n \in \mathbb{N}^*$ un entier. Si n est pair, alors il est premier si et seulement si $n = 2$. Si n est impair, alors aucun diviseur de n n'est pair. Ainsi, pour tester si $n \in \mathbb{N}^*$ est premier, il suffit de vérifier si n est égal à 2 ou bien s'il possède un diviseur d impair.

Soit $n \in \mathbb{N}^*$. On note $D(n)$ l'ensemble de tous les entiers d impairs tels que $1 < d \leqslant \lfloor \sqrt{n} \rfloor$ où $\lfloor \sqrt{n} \rfloor$ désigne la partie entière de $\sqrt{n}$. D'après les deux remarques ci-dessus, la procédure suivante permet de tester si n est premier : si $n = 1$ alors il n'est pas premier ; sinon, si $n = 2$ alors il est premier ; sinon, si n est pair alors il n'est pas premier ; sinon, on parcourt tous les entiers $d \in D(n)$ (et uniquement ceux-là) ; si au moins l'un de ces d divise n alors n n'est pas premier ; sinon n est premier.

3. Écrire une fonction de signature `estPremier2(n: int) -> bool` qui renvoie `True` si `n` est premier et `False` sinon. On pourra supposer sans le vérifier que $\texttt{n} \in \mathbb{N}^*$. Bien sûr, vous devez utiliser la procédure décrite ci-dessus. Pour calculer la partie entière d'un flottant, on utilisera la fonction « `floor(x: float) -> int` » du module `math`.

1.3 Test de primalité de Fermat et nombres de Carmichael

Les programmes précédents étant assez lents, on souhaite utiliser un algorithme plus efficace. Le test de Fermat est une procédure très rapide, mais qui, malheureusement, peut parfois échouer. Ce test se base sur le théorème suivant :

Théorème 3 (Petit théorème de Fermat)**.** *Si p est un nombre premier, alors pour tout $a \in [\![1; p-1]\!]$, $a^{p-1} - 1$ est divisible par p.*

Soient $n \geqslant 2$ et $a \in [\![1; n-1]\!]$ deux entiers. Le test de Fermat consiste à vérifier si $a^{n-1} - 1$ est divisible par n. Si ce n'est pas le cas, on sait que n n'est pas premier. En revanche, si $a^{n-1} - 1$ est divisible par n, rien ne permet de décider si n est premier ou non.

Définition 4. Soient $n \in \mathbb{N}^*$ et $a \in \mathbb{N}^*$ deux entiers. On note $PGCD(n, a)$ le plus grand entier qui divise à la fois n et a. En Python, $PGCD(n, a)$ se calcule grâce à la commande `math.gcd(n,a)`.

Définition 5. Soit $n \in \mathbb{N}^*$ un entier. On note $P(n)$ l'ensemble des entiers $a \in [\![1; n-1]\!]$ tels que $PGCD(n, a) = 1$.

Définition 6. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre de Carmichael*** si le trois conditions suivantes sont vérifiées :

$$n \geqslant 2, \qquad n \text{ n'est pas premier}, \qquad \text{Pour tout } a \in P(n) : n \text{ divise } a^{n-1} - 1.$$

4. (a) Écrire une fonction de signature `makeP(n: int) -> list[int]` qui renvoie la liste $P(n)$ de la définition 5. On pourra supposer sans le vérifier que $\texttt{n} \in \mathbb{N}^*$.
   (b) Écrire une fonction de signature `estCarmic(n: int) -> bool` qui indique si `n` est un nombre de Carmichael. On pourra supposer sans le vérifier que $\texttt{n} \in \mathbb{N}^*$ et que `n` n'est pas un nombre premier.

On souhaite maintenant utiliser les fonctions `listeNP` (question 2) et `estCarmic` pour générer la liste de tous les nombres de Carmichael compris entre 1 et 4567. Dans un premier temps, il s'agit de construire la liste des entiers de $[\![1; 4567]\!]$ qui ne sont pas premiers.

5. (a) Écrire une fonction de signature : `listeBoolNP() -> list[bool]` qui renvoie une liste `L` de taille 4568 telle que pour tout $\texttt{i} \in [\![0, 4567]\!]$, `L[i]` vaut `True` si `i` est premier et `False` sinon. Vous devez utiliser la fonction `listeNP` de la question 2. On obtient :

```
[False, False, True, True, False, True, False, True, False, False, ...]
```

   (b) Écrire une fonction de signature `indicesFalse(L: list[bool]) -> list[int]` qui renvoie la liste de tous les indices $\texttt{i} \geqslant 2$ tels que `L[i]` vaut `False`. Par exemple, avec :

```
L = [False, False, True, True, False, True, False,
     True, False, False, False, True, False, True, False],
```

   on obtient `[4, 6, 8, 9, 10, 12, 14]`.
   (c) À l'aide des fonctions précédentes, écrire une fonction de signature `listeCarmic() -> list[int]` qui renvoie la liste des nombres de Carmichael compris entre 1 et 4567.

1.4 Test de primalité de Miller-Rabin

Le dernier test de primalité qu'on va étudier est le test de Miller-Rabin. À l'instar du test de Fermat, le test de Miller-Rabin ne peut pas garantir avec certitude qu'un entier $n \in \mathbb{N}^*$ est premier. Toutefois, s'il ne détecte pas que n n'est pas premier, alors la probabilité que n ne soit pas premier est tellement faible, qu'on considérera qu'il est premier. En résumer, le test de Miller-Rabin peut :

- Annoncer que n n'est pas premier. Dans ce cas, on sait avec certitude qu'il ne l'est pas.
- Annoncer que n est premier. Dans ce cas, il y a une chance infime que n ne soit pas premier, mais on considérera que n est bel et bien premier.

Définition 7. Soit $m \in \mathbb{N}^*$ un entier. On note k le plus grand entier de $[\![0; m]\!]$ tel que 2^k divise m. L'entier k s'appelle la ***valuation 2-adique*** de m et on le note $v_2(m)$.

6. Écrire une fonction de signature `val2Adique(m: int) -> int` qui renvoie la valuation 2-adique de `m`. On pourra supposer sans le vérifier que `m` $\in \mathbb{N}^*$. Par exemple :

`m`	1	2	3	4	8	112
`val2adique(m)`	0	1	0	2	3	4

Soient n et a deux entiers vérifiant la condition (C) suivante :

$$(C) : n \geqslant 4 \text{ est un entier impair et } a \in [\![2; n-2]\!].$$

Soient $d = \dfrac{n-1}{2^{v_2(n-1)}}$ et $(x_k)_{k\in\mathbb{N}}$ la suite définie par : $\begin{cases} x_0 = a^d \mod n \\ x_{k+1} = (x_k)^2 \mod n \end{cases}$ pour tout $k \geqslant 0$

Remarque 8. En Python, l'opération « $y \mod z$ » correspond à « `y % z` ».

7. (a) Écrire une fonction `verifC` qui prend en entrée n et a, qui renvoie `True` si la condition (C) est vérifiée et `False` sinon
   (b) Écrire une fonction `get_u` qui prend en entrée n et a, et renvoie la liste `L` de longueur $\ell = v_2(n-1)+1$ contenant les entiers $x_0, x_1, x_2, \ldots, x_{\ell-1}$. On pourra supposer sans le vérifier que la condition (C) est vérifiée. De plus, à des fins d'optimisation, on calculera la valeur de x_0 à l'aide de la fonction `pow` :

```
x0 = pow(a, d, n)
# x0 = (a**d) % n # L'exécution de cette ligne serait trop lente
```

   (c) Donner les signatures des fonctions `verifC` et `get_u`.

Définition 9. Soient n et a deux entiers vérifiant la condition (C) et $(x_k)_{k\in\mathbb{N}}$ la suite définie ci-dessus. On note (C_1), (C_2) et (C_3) les trois conditions suivantes :

$$(C_1) : x_0 \notin \{1, n-1\}.$$
$$(C_2) : x_{v_2(n-1)} \neq 1.$$
$$(C_3) : \text{il existe } k \in [\![1; v_2(n-1)]\!] \text{ tel que } x_{k-1} \notin \{1, n-1\} \text{ et } x_k = 1.$$

On dit que a est un ***témoin de Miller pour*** n, si : (C_1) est vraie et $\Big[(C_2)$ est vraie ou (C_3) est vraie$\Big]$.

8. Écrire une fonction de signature `estTemoinMiller(n: int, a: int) -> bool` qui indique si `a` est un témoin de Miller pour `n`. Si la condition (C) n'est pas vérifiée, votre fonction déclenchera une erreur.

Le test de Miller-Rabin consiste maintenant à choisir 100 entiers aléatoires $a \in [\![2; n-2]\!]$. Si l'un de ces entiers est un témoin de Miller pour n, c'est que n n'est pas premier, sinon c'est que n est premier.

9. Écrire une fonction `estPremier3` qui renvoie `True` si `n` est premier et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$ et utiliser la fonction « `randint(a1: int, a2: int) -> int` » du module `random` qui renvoie un entier aléatoire de l'intervalle $[\![$`a1; a2`$]\!]$. Bien sûr, vous devez utiliser le test de Miller-Rabin décrit ci-dessus.

2 Familles de nombres premiers

Dans cette partie, on s'intéresse à des nombres premiers dont les chiffres forment une suite remarquable. Voici quelques indications qui vous seront utiles :

- Attention à la différence entre « chiffre » et « nombre » : un chiffre est un élément de $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, un nombre est un élément de $\mathbb{N}$.
- Pour tester si un nombre est premier, utilisez la fonction `estPremier3` de la question 9.
- La fonction « `str(n: int) -> str` » convertit un entier vers une chaîne de caractères : `str(3137)` s'évalue en `"3137"`. De même, la fonction « `int(s: str) -> int` » convertit une chaîne de caractères vers un entier : `int("3137")` s'évalue en `3137`

Dans cette partie, on aura besoin à plusieurs reprises de tester si tous les éléments d'une liste sont premiers.

10. Écrire une fonction de signature `tousPremiers(L: list[int]) -> bool` qui renvoie `True` si tous les éléments de `L` sont premiers et `False` sinon.

2.1 Répunit

Définition 10. Un ***répunit*** (contraction de « repeated unit ») est un nombre dont tous les chiffres sont égaux à 1. Pour $k \in \mathbb{N}^*$, on note R_k le répunit composé de k fois le chiffre 1.

Exemple 11. On a $R_1 = 1, R_2 = 11, R_3 = 111, R_4 = 1\,111, R_5 = 11\,111, R_6 = 111\,111, \ldots$

11. Écrire une fonction de signature `repunitPremiers(k: int) -> list[int]` qui renvoie la liste de tous les répunit compris entre R_1 et R_k qui sont des nombres premiers.
    Par exemple, « `repunitPremiers(6)` » s'évalue en `[11]` car dans l'ensemble $\{R_1, R_2, R_3, R_4, R_5, R_6\}$, seul R_2 est un nombre premier.

2.2 Nombres premiers palindromes

Définition 12. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier palindrome*** si c'est un nombre premier et si la suite de ses chiffres est la même lorsqu'on la lit de gauche à droite que lorsqu'on la lit de droite à gauche.

Exemple 13. 2 et 1713302033171 sont des nombres premiers palindromes, mais 13 n'en est pas un. De plus, tout répunit premier est un nombre premier palindrome.

12. Écrire une fonction de signature `estNPPalind(n: int) -> bool` qui renvoie `True` si `n` est un nombre premier palindrome et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$. Dans cette question, il est demandé d'utiliser une boucle `for` et d'y faire le moins de tours possibles.

2.3 Nombres premiers circulaires

Définition 14. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier circulaire*** si les nombres obtenus en appliquant des permutations circulaires sur ses chiffres sont tous premiers.

Exemple 15. Le nombre $n = 193939$ est un nombre premier circulaire. En effet, en appliquant des permutations circulaires on obtient :

193939 939391 393919 939193 391939 919393

Étant donné que ces six nombres sont premiers, n est bien un nombre premier circulaire.
De plus, un répunit premier est toujours un nombre premier circulaire.

13. Écrire une fonction de signature `estNPCirc(n: int) -> bool` qui renvoie `True` si `n` est un nombre premier circulaire et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$. Dans cette question, vous devez utiliser la fonction `tousPremiers` de la question 10.

2.4 Nombres premiers tronquables

Définition 16. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier tronquable à droite*** si en supprimant un par un les chiffres de n de droite à gauche, tous les nombres obtenus sont premiers.

Exemple 17.

- ⋆ 3137 est un nombre premier tronquable à droite car 3137, 313, 31 et 3 sont tous premiers.
- ⋆ 31193 est un nombre premier tronquable à droite car 31193, 3119, 311, 31 et 3 sont tous premiers.
- ⋆ 37277 n'est pas un nombre premier tronquable à droite car 37277, 3727, 37 et 3 sont tous premiers, mais pas 372.
- ⋆ 31373 n'est pas un nombre premier tronquable à droite car 3137, 313, 31 et 3 sont tous premiers, mais pas 31373.

On souhaite d'abord vérifier si un entier $n \in \mathbb{N}^*$ donné est un nombre premier tronquable à droite.

14. Écrire une fonction de signature `estNPTD(n: int) -> bool` qui indique si `n` est un nombre premier tronquable à droite.

Essayons maintenant de générer la liste de tous les nombres premiers tronquables à droite. Pour cela, pour tout `k` $\in \mathbb{N}^*$, on note :

- $\texttt{L}_\texttt{k}$ la liste contenant tous les nombres premiers tronquables à droite composés d'exactement `k` chiffres. Par exemple : $\texttt{L}_1$ `= [2, 3, 5, 7]` et $\texttt{L}_2$ `= [23, 29, 31, 37, 53, 59, 71, 73, 79]`.
- $\texttt{M}_\texttt{k}$ la liste contenant tous les nombres obtenus en concaténant le chiffre 1, 3, 7 ou 9 à la fin d'un élément de $\texttt{L}_\texttt{k}$. Par exemple :

$\texttt{M}_1$ `=[21, 23, 27, 29, 31, 33, 37, 39, 51, 53, 57, 59, 71, 73, 77, 79]`

$\texttt{M}_2$ `=[231, 233, 237, 239, 291, 293, 297, 299, 311, 313, 317, 319, 371, 373, 377, 379,`
`531, 533, 537, 539, 591, 593, 597, 599, 711, 713, 717, 719, 731, 733, 737,`
`739, 791, 793, 797, 799]`

- $\texttt{T}_\texttt{k}$ la liste contenant tous les nombres premiers tronquables à droite composés d'au plus `k` chiffres. Par exemple : $\texttt{T}_1$ `= [2, 3, 5, 7]` et $\texttt{T}_2$ `= [2, 3, 5, 7, 23, 29, 31, 37, 53, 59, 71, 73, 79]`.

15. Soit `k` $\in \mathbb{N}^*$ et `n` un élément de $\texttt{L}_{\texttt{k+1}}$. Justifier que `n` appartient à $\texttt{M}_\texttt{k}$.
16. Soit `k` $\in \mathbb{N}^*$.
    (a) Dans cette question, on suppose que la liste $\texttt{L}_\texttt{k}$ a déjà été construite. Écrire une fonction `L_to_M` qui prend en entrée $\texttt{L}_\texttt{k}$ et renvoie $\texttt{M}_\texttt{k}$. Bien sûr, votre fonction doit utiliser $\texttt{L}_\texttt{k}$ pour construire $\texttt{M}_\texttt{k}$.
    (b) Dans cette question, on suppose que la liste $\texttt{M}_\texttt{k}$ a déjà été construite. Écrire une fonction `M_to_L` qui prend en entrée $\texttt{M}_\texttt{k}$ et renvoie $\texttt{L}_{\texttt{k+1}}$. Bien sûr, votre fonction doit utiliser $\texttt{M}_\texttt{k}$ pour construire $\texttt{L}_{\texttt{k+1}}$.
    (c) Écrire une fonction de signature `make_T8_T9() -> (list[int], list[int])` qui renvoie les listes $\texttt{T}_8$ et $\texttt{T}_9$. Vous devez utiliser les deux fonctions précédentes.

Lorsqu'on teste la fonction `make_T8_T9`, on remarque que le programme ci-contre affiche deux fois l'entier `83`.

```
T8, T9 = make_T8_T9()
print(len(T8), len(T9)) # Affiche "83 83"
```

17. Montrer qu'il y a un nombre fini de nombres premiers tronquables à droite.

Définition 18. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier tronquable à gauche*** si aucun de ses chiffres n'est égal à 0 et si en supprimant un par un les chiffres de n de gauche à droite, tous les nombres obtenus sont premiers.

Exemple 19. 1223 est un nombre premier tronquable à gauche car 1223, 223, 23 et 3 sont tous premiers.

18. Écrire une fonction de signature `listeNPTG() -> list[int]` qui renvoie la liste de tous les nombres premiers tronquables à gauche. On pourra utiliser le fait que le plus grand nombre premier tronquable à gauche est 357 686 312 646 216 567 629 137. Le programme doit être suffisamment efficace pour renvoyer le résultat dans un temps raisonnable. On expliquera succinctement la procédure utilisée.

2.5 Nombres premiers délicats

Définition 20. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier délicat*** si n est premier et lorsqu'on modifie n'importe quel chiffre de n, on obtient nécessairement un nombre qui n'est **pas** premier.

Exemple 21.

⋆ Le nombre $n = 13$ n'est pas un nombre premier délicat. En effet, lorsqu'on modifie un chiffre de n, voici tous les nombres qu'on peut obtenir :

3	23	33	43	53	63	73	83	93
10	11	12	14	15	16	17	18	19

Parmi ces 18 nombres, certains sont premiers (par exemple 17), donc n n'est pas un nombre premier délicat.

⋆ Le nombre $n = 294001$ est un nombre premier délicat. En effet, lorsqu'on modifie un chiffre de n, voici tous les nombres qu'on peut obtenir :

94001	194001	394001	494001	594001	694001	794001	894001	994001
204001	214001	224001	234001	244001	254001	264001	274001	284001
...								

Parmi ces $9 \times 6 = 54$ nombres, aucun n'est premier, donc n est un nombre premier délicat.

19. Écrire une fonction de signature `estNPDelicat(n: int) -> bool` qui renvoie `True` si `n` est un nombre premier délicat et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$.

2.6 Nombres premiers permutables

Définition 22. Soit $n \in \mathbb{N}^*$. On dit que n est un ***nombre premier permutable*** si les nombres obtenus en appliquant une permutation quelconque sur les chiffres de n sont tous premiers.

Exemple 23.

⋆ $n = 1379$ n'est pas un nombre premier permutable. En effet, lorsqu'on permute les chiffres de n, on obtient :

1379	1397	1739	1793	1937	1973
3179	3197	3719	3791	3917	3971
7139	7193	7319	7391	7913	7931
9137	9173	9317	9371	9713	9731

Parmi ces 24 nombres, certains ne sont pas premiers (par exemple 1397 n'est pas premier), donc n n'est pas un nombre premier permutable.

⋆ $n = 991$ est un nombre premier permutable. En effet, lorsqu'on permute les chiffres de n, on obtient :

991	919	991	919	199	199

Ces 6 nombres sont premiers, donc n est un nombre premier permutable.

⋆ Un répunit premier est un nombre premier permutable.

20. Écrire une fonction de signature `estNPPerm(n: int) -> bool` qui renvoie `True` si `n` est un nombre premier permutable et `False` sinon. On pourra supposer sans le vérifier que `n` $\in \mathbb{N}^*$.