1 Arithmétique de base

Bloc-Note 1 *Divisibilité*

☞ **Diviseurs et multiples**

- Soit deux entiers strictement positifs a et b, il existe un unique couple d'entiers positifs (q,r) avec $0 \leqslant r < b$ tel que $a = qb + r$.
  q est le **quotient** et r est le **reste** de la division euclidienne de a par b.
- Soit deux entiers strictement positifs a et b, b divise a si le reste de la division euclidienne de a par b est 0 : $a = q \times b$.
  b est un **diviseur** de a et a est un **multiple** de b.
- Deux entiers a et b strictement positifs admettent **plus grand diviseur commun** qui s'appelle leur PGCD.
- Deux entiers a et b strictement positifs admettent un **plus petit commun multiple** qui s'appelle leur PPCM.
- Deux entiers strictement positifs sont **premiers entre eux** (ou **étrangers**) s'ils n'ont pas d'autre diviseur commun que 1 c'est-à-dire si leur PGCD est 1.

☞ **Critères de divisibilité**

- Un nombre entier est **divisible par** 2 ou pair, si son chiffre des unités est 0, 2, 4, 6 ou 8.
- Un nombre entier est **divisible par** 3 si la somme de ses chiffres est divisible par 3.
- Un nombre entier est **divisible par** 4 si le nombre formé par ses deux derniers chiffres est divisible par 4.
- Un nombre entier est **divisible par** 5 si son chiffre des unités est 0 ou 5.
- Un nombre entier est **divisible par** 6 s'il est divisible par 3 et 2.
  Plus généralement si p et q divisent a et que p et q sont premiers entre eux, alors leur produit pq divise a.
- Un nombre entier est **divisible par** 8 si le nombre formé par ses trois derniers chiffres est divisible par 8.
- Un nombre entier est d**ivisible par** 9 si la somme de ses chiffres est divisible par 9.
- Un nombre entier est **divisible par** 10 si son chiffre des unités est 0.

☞ **Nombres premiers**

- Un nombre entier, strictement plus grand que 1, qui n'est divisible que par 1 et lui-même est un **nombre premier**.
- Il existe une **infinité de nombres premiers**.
  La série des nombres premiers commence par 2, 3, 5, 7, 11, 13, 17, 19 ...
- Tout nombre entier strictement plus grand que 1 possède une unique **décomposition comme un produit de nombres premiers**.
  Par exemple : $28 = 2^2 \times 7$.

Exercice 1 *Divisibilité*

1. Démontrer les critères de divisibilité par 3 et 9.
2. Démontrer que tout produit $n(n+1)$ de deux entiers consécutifs est divisible par 2.
3. Démontrer que tout produit $(n-1)n(n+1)$ de trois entiers consécutifs est divisible par 3.
4. Démontrer que tout produit $2n \times (2n+2)$ de deux nombres pairs consécutifs est divisible par 8.
5. Démontrer que tout produit $(n-1)n(n+1)$ de trois entiers consécutifs, tel que le nombre central est impair, est divisible par 6 et par 24
6. Soit $n = 10d + u$ un entier avec d dizaines et u unités.
   Démontrer que 7 divise n si et seulement si 7 divise $d - 2u$.

Exercice 2 *Récréations mathématiques*

1. Un facteur apporte une lettre à un père de trois filles. Il demande l'âge des filles. Le père lui dit que le produit des âges est 36 et que la somme des âges est égal au numéro de la maison d'en face. Le facteur se retourne, regarde le numéro et dit « *Cela ne me suffit pas* ».
   Le père ajoute alors « *La plus jeune est brune* ».
   Trouver les âges des filles.
2. Trouver le plus petit nombre entier naturel qui est divisible par 45, dont la somme des chiffres est 45 et dont l'écriture décimale se termine par 45.
3. a et b sont deux entiers strictement positifs tels que $a + b + ab = 90$, trouver leur produit ab.
4. Deux nombres a et b sont entiers et strictement positifs. Parmi les quatre affirmations suivantes, trois sont vraies et une est fausse :
   - $a+1$ est divisible par b,
   - a est égal à $2b+5$,
   - $a+b$ est divisible par 3,
   - $a+7b$ est premier.

   Quels sont les couples (a,b) possibles ?

2 Arithmétique et programmation

Bloc-Note 2 *Rappels* `Python`

☞ Échange des valeurs de deux variables, et affichage avant puis après :

```
a = 634
b = 633
print(a, b)
a, b = b, a
print(a, b)
```

☞ Fonction $f : x \mapsto \dfrac{3 \times x - 1}{x^2 + 1}$:

```
def f(x):
    return (3 * x - 1)/ (x ** 2 + 1)
```

☞ Somme des cubes des entiers n tels que $2 \leqslant n < 10$:

```
somme = 0
for n in range(2, 10):
    somme = somme + n ** 3
```

☞ Plus petit entier n tel que $2^n \geqslant 98765$:

```
n = 0
puissance = 1
while puissance < 98765:
    n = n + 1
    puissance = puissance * 2
```

☞ Racines d'un trinôme :

```
from math import sqrt

def trinome(a, b, c):
    '''Retourne la liste des racines réelles de a*x**2 + b*x + c'''
    delta = b ** 2 - 4 * a * c
    if delta < 0:
        return []
    elif delta > 0:
        x1 = (-b - sqrt(delta))/(2 * a)
        x2 = (-b + sqrt(delta))/(2 * a)
        return [x1, x2]
    else:
        return [-b/(2 * a)]
```

☞ Pour les entiers n tels que $1 \leqslant n < 1000$, liste des carrés divisibles par 9 :

```
liste = []
for n in range(1, 1000):
    carre = n ** 2
    if carre % 9 == 0:
        liste.append(carre)
```

Exercice 3 *Projet Euler*

Résoudre le problème n° 1 du projet Euler.

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

Exercice 4 *Suite de Syracuse*

A la fin des années 20 du vingtième siècle, Lothar Collatz, mathématicien allemand (1910 - 1990), s'intéressait aux itérations de fonctions prenant des valeurs entières et inventa une suite de nombres définie de la manière suivante :

- on choisit un entier naturel ;
- si cet entier est pair, on le divise par deux ;
- sinon on le multiplie par 3 et on ajoute 1 ;
- on réitère le procédé avec l'entier obtenu ...

Cette suite est devenue célèbre à l'université de Syracuse aux États-Unis à partir des années 1950 car on peut formuler la conjecture qu'elle aboutit à l'entier 1 quel que soit l'entier choisi au départ, mais cette conjecture n'a toujours pas été démontrée. D'après le mathématicien hongrois Paul Erdos, « *Les mathématiques ne sont pas encore prêtes pour ce type de problème* ».

À ce jour, on a vérifié la conjecture avec l'informatique pour tout entier inférieur à $20 \times 2^{58} \approx 5,764 \times 10^{18}$.

Si on part d'un entier n, l'ensemble des valeurs prises par la suite de Syracuse jusqu'à l'obtention de 1 s'appelle un vol.

1. Écrire une fonction `syracuse(n)` qui pour un entier `n` retourne le terme suivant dans une suite de Syracuse.
2. Écrire une fonction `duree_vol(n)` qui retourne le nombre d'itérations nécessaires jusqu'à ce que la suite de Syracuse de départ `n` atterrisse en 1.
3. Écrire une fonction `altitude(n)` qui retourne la valeur maximale atteinte par la suite de Syracuse de départ `n` avant qu'elle atterrisse en 1.
4. On dit qu'une durée de vol pour une suite de Syracuse de départ n est record si elle est supérieure aux durées de vol à partir des entiers inférieurs à n.

   Écrire une fonction `record_duree(n)` qui retourne la liste des couples (départ,durée) de toutes les durées de vol records pour les entiers inférieurs ou égaux à `n`.
5. On dit qu'une altitude de vol pour une suite de Syracuse de départ n est record si elle est supérieure à toutes les altitudes de vol à partir des entiers inférieurs à n.

   Écrire une fonction `record_altitude(n)` qui retourne la liste des couples (départ,altitude) de toutes les altitudes de vol records pour les entiers inférieurs ou égaux à `n`.

Exercice 5 *Persistance additive*

On considère la fonction F qui à tout nombre entier, associe la somme de ses chiffres, il faut deux itérations à partir de 19 pour aboutir à un nombre comportant un seul chiffre.

En effet, on a : $19 \overset{F}{\longmapsto} 10 \overset{F}{\longmapsto} 1$.

En itérant la fonction F, on aboutit à un nombre avec un seul chiffre appelé *racine digitale additive*. La *persistance additive* est le nombre d'itérations nécessaires. Par exemple, la persistance additive de 19 est de 2.

1. Écrire une fonction `somme_liste(L)` qui retourne la somme des éléments d'une liste de nombres.
2. Commenter la fonction `liste_chiffres(n)` ci-dessous qui retourne la liste des chiffre en base 10 de l'entier `n` passé en paramètre.

```
def liste_chiffres(n):
    L = [n % 10]
    n = n // 10
    while n > 0:
        L.append(n % 10)
        n = n // 10
    L.reverse()
    return L
```

3. Écrire une fonction `persistance_additive(n)` qui retourne la persistance additive de l'entier `n` passé en paramètre.
4. Écrire un programme qui affiche le plus petit entier dont la persistance additive est de 3 [1].
5. Démontrer que la persistance additive d'un entier peut être rendue aussi grande que l'on veut.

Exercice 6 *Persistance multiplicative*

On considère la fonction G qui à tout nombre entier, associe la somme de ses chiffres, il faut quatre itérations à partir de 77 pour aboutir à un nombre comportant un seul chiffre.

En effet, on a : $77 \overset{G}{\longmapsto} 49 \overset{G}{\longmapsto} 36 \overset{G}{\longmapsto} 18 \overset{G}{\longmapsto} 8$.

En itérant la fonction G, on aboutit à un nombre avec un seul chiffre appelé *racine digitale multiplicative*. La *persistance multiplicative* est le nombre d'itérations nécessaires. Par exemple, la persistance multiplicative de 77 est de 4.

Selon un résultat de 2 001, il n'existe pas de nombre inférieur à $2^{333} \approx 1{,}38 \times 10^{70}$ dont la persistance multiplicative est strictement supérieure à 11.

1. Écrire une fonction `persistance_multiplicative(n)` qui retourne la persistance multiplicative de l'entier `n` passé en paramètre.
2. Écrire un programme qui affiche le plus petit entier dont la persistance multiplicative est de 6. Idem pour 7. Et pour $n > 8$? [2].

Exercice 7 *Nombres résistants*

Un *nombre résistant* est un nombre de n chiffres divisible par n et tel que les nombres obtenus en éliminant ses chiffres les uns après les autres restent divisibles par leur nombre de chiffres. Par exemple 123 252 est un nombre résistant à 6 chiffres car 6 divise 123 252, 5 divise 12 325, 4 divise 1 232 ...

1. 19 999 999 999 999 999 999 999 est le plus petit entier dont la persistance additive est 4
2. Le plus petit nombre dont la persistance multiplicative est 8 est 2 677 889, pour 9 c'est 26 888 999

1. Sans compter 0, quels sont les nombres résistants à un chiffre ? à deux chiffres ? à trois chiffres ?

2. Il existe 375 nombres résistants à quatre chiffres, combien en existe-t-il à cinq chiffres ?

3. Compléter le programme Python ci-dessous pour qu'il affiche combien il existe de nombres résistants à 1, 2, 3, ... 9, 10 chiffres.

```
n = 1
#resistant liste nombres résistants, initialisé avec les résistants à 1
    chiffre
resistant = [k for k in range(1, 10)]
print("Nombre de chiffres : 1 Nombre de résistants : ", len(resistant))
for nbchiffre in range(2, 10):
    nbresistant = len(resistant)
    for i in range(nbresistant):
        r = resistant.pop(0)
        for u in range(0, 10):
            #à  compléter
    print("Nombre de chiffres : ", nbchiffre, " Nombre de résistants : "
        , len(resistant))
```

Exercice 8 *Algorithme d'Euclide*

1. On donne ci-dessous un exemple d'exécution de l'algorithme d'Euclide pour déterminer le PGCD de 136 et 34.

   Le PGCD est le dernier reste non nul.

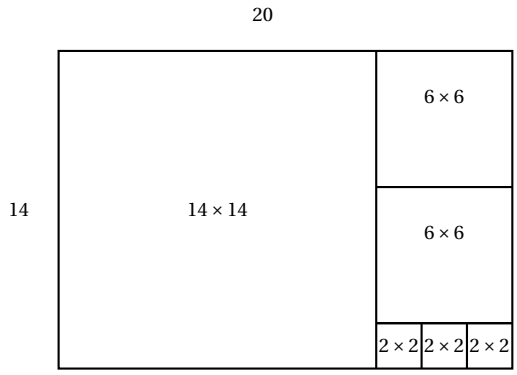
Calcul	Dividende	Quotient	Diviseur	Reste
$136 = 3 \times 34 + 32$	136	3	34	32
$34 = 1 \times 32 + 2$	34	1	32	2
$32 = 16 \times 2 + 2$	32	16	2	0

   Écrire en Python une fonction `pgcd(a, b)` qui retourne le PGCD de deux entiers strictement positifs a et b en implémentant l'algorithme d'Euclide.

2. L'algorithme des différences permet également de déterminer le PGCD de deux entiers.

   Le tableau ci-contre donne un exemple d'exécution de cet algorithme pour déterminer le PGCD des entiers $a = 75$ et $b = 30$.
   Écrire en Python une fonction de signature `pcgd_difference(a, b)` qui retourne le PGCD des paramètres entiers `a` et `b` par l'algorithme des différences.

a	b	différence
75	30	45
45	30	15
30	15	15
15	15	0



3. En s'inspirant de l'algorithme des différences pour calculer le PGCD de deux entiers, écrire une fonction `produit_to_sommecarre(a, b)` qui retourne la liste des nombres qui font partie de la décomposition du produit des deux entiers `a` et `b` passés comme paramètres.

Il faut implémenter l'algorithme de Babylone, décrit dans le graphique précédent :

$$14 \times 20 = 14^2 + 6^2 + 6^2 + 2^2 + 2^2 + 2^2$$

```
>>> produit_to_sommecarre(14,20)
[14, 6, 6, 2, 2, 2]
```

Plus précisément, d'après un théorème démontré par Lagrange en 1772, tout entier naturel peut s'écrire comme une somme de quatre carrés d'entiers.

Exercice 9 *Nombres premiers*

1. Justifier que si un entier strictement positif n ne possède aucun diviseur distinct de 1, inférieur ou égal à $\sqrt{n}$, alors n est un nombre premier.

2. En déduire une fonction `est_premier(n)` qui teste la primalité d'un entier n.

```
>>> est_premier(7)
True

>>> est_premier(14)
False
```

3. Écrire une fonction `decomposition_facteur_premier(n)` qui retourne la liste des facteurs premiers de l'entier strictement positif `n`.

```
>>> decomposition_facteur_premier(28)
[2, 2, 7]
```

4. Résoudre le problème n° 5 du projet Euler.

> 2520 *is the smallest number that can be divided by each of the numbers from* 1 *to* 10 *without any remainder.*
>
> ***What is the smallest positive number that is evenly divisible (divisible with no remainder) by all of the numbers from*** 1 ***to*** 20 ***?***

5. L'algorithme du *crible d'Ératosthène* permet de déterminer les entiers majorés par N qui sont premiers, plus efficacement que l'application en boucle du test de primalité précédent. Le principe en est le suivant :

- On initialise une liste (disons T) de $N+1$ booléens à `True` (sauf les deux premiers, qui correspondent aux entiers non premiers 0 et 1). En k-ème case, le `True` s'interprète « jusqu'ici, et jusqu'à preuve du contraire, k semble être premier ».
- Tous les multiples (stricts) de 2 sont composés : on réaffecte donc à `False` tous les $T[2i]$ pour $2 \leq i \leq N/2$.
- Tous les multiples de 3 sont composés : on passe donc à `False` tous les $T[3i]$ pour $3 \leq i \leq N/3$.
- On constate que 4 est composé ($T[4]$ a été mis à `False` lors du premier passage) : inutile de « rayer » les multiples de 4, qui l'ont déjà été.
- Tous les multiples de 5 sont composés : on passe donc à `False` tous les $T[5i]$ pour $5 \leq i \leq N/5$.
- ...

On notera que pour $k = 3$ par exemple, il est inutile de basculer $T[k*2]$ à `False` : ça a déjà été fait lors du traitement de $k = 2$.

On balaye ainsi toute la liste de nombreuses fois : environ $\sqrt{N}$ fois.

Compléter la fonction `crible(N)` qui retourne la liste obtenue par criblage des entiers inférieurs ou égaux à `N` .

```
from math import sqrt

def crible(N):
    T = [False] * 2 + [True] * (N - 1)
    for i in range(2, int(sqrt(N)) + 1):
        if .........:
            for k in range(k, N//i + 1):
                ......................
    return T
```

6. Résoudre le problème n° 3 du projet Euler.

The prime factors of 13 195 *are* 5, 7, 13 *and* 29.
What is the largest prime factor of the number 600 851 475 143 ***?***

7. Résoudre le problème n° 7 du projet Euler.

By listing the first six prime numbers : 2, 3, 5, 7, 11, *and* 13, *we can see that the 6th prime is* 13.
What is the 10 001***st prime number?***

8. Résoudre le problème n° 10 du projet Euler.

The sum of the primes below 10 *is* $2+3+5+7=17$.
Find the sum of all the primes below two million.