Arithmétique – Boucle tant que – II

On approfondit notre étude des nombres avec la boucle « tant que ». Pour cette fiche tu as besoin d'une fonction `est_premier()` *construite dans la fiche « Arithmétique – Boucle tant que – I ».*

Activité 1 (Conjecture(s) de Goldbach)**.**

Objectifs : étudier deux conjectures de Goldbach. Une conjecture est un énoncé que l'on pense vrai mais que l'on ne sait pas démontrer.

1. **La bonne conjecture de Goldbach :** *Tout entier pair plus grand que* 4 *est la somme de deux nombres premiers.*

   Par exemple $4 = 2+2$, $6 = 3+3$, $8 = 3+5$, $10 = 3+7$ (mais aussi $10 = 5+5$), $12 = 5+7$,… Pour $n = 100$ il y a 6 solutions : $100 = 3+97 = 11+89 = 17+83 = 29+71 = 41+59 = 47+53$.

   Personne ne sait démontrer cette conjecture, mais tu vas voir qu'il y a de bonnes raisons de penser qu'elle est vraie.

   (a) Programme une fonction `nombre_solutions_goldbach(n)` qui pour un entier pair n donné, trouve combien il existe de décompositions $n = p+q$ avec p et q deux nombres premiers et $p \leqslant q$.

   Par exemple pour $n = 8$, il n'y a qu'une seule solution $8 = 3+5$, par contre pour $n = 10$ il y a deux solutions $10 = 3+7$ et $10 = 5+5$.

   Indications.
   - Il faut donc tester tous les p compris 2 et $n/2$;
   - poser $q = n-p$;
   - on a une solution quand $p \leqslant q$ et que p et q sont tous les deux des nombres premiers.

   (b) Prouve avec la machine que la conjecture de Goldbach est vérifiée pour tous les entiers n pairs compris entre 4 et 10 000.

2. **La mauvaise conjecture de Goldbach :** *Tout entier impair n peut s'écrire sous la forme*

$$n = p + 2k^2$$

   où p est un nombre premier et k un entier (éventuellement nul).

   (a) Programme une fonction `existe_decomposition_goldbach(n)` qui renvoie « vrai » lorsqu'il existe une décomposition de la forme $n = p+2k^2$.

   (b) Montre que cette seconde conjecture de Goldbach est fausse ! Il existe deux entiers plus petits que 10 000 qui n'admettent pas une telle décomposition. Trouve-les !

Activité 2 (Nombres ayant 4 ou 8 diviseurs)**.**

Objectifs : réfuter une conjecture en faisant beaucoup de calculs !

Conjecture : *Entre* 1 *et N, il y a plus d'entiers qui ont exactement* 4 *diviseurs que d'entiers qui ont exactement* 8 *diviseurs.*
Tu vas voir que cette conjecture a l'air vrai pour N assez petit, mais tu vas montrer que cette conjecture est fausse en trouvant un N grand qui contredit cet énoncé.

1. **Nombre de diviseurs.**
   Programme une fonction `nombre_de_diviseurs(n)` qui renvoie le nombre d'entiers divisant n.
   Par exemple : `nombre_de_diviseurs(100)` renvoie 9 car il y a 9 diviseurs de $n = 100$:
   $$1, 2, 4, 5, 10, 20, 25, 50, 100$$
   Indications.
   - N'oublie pas 1 et n comme diviseurs.
   - Essaie d'optimiser ta fonction car tu l'utiliseras intensivement : par exemple il n'y a pas de diviseurs strictement plus grands que $\frac{n}{2}$ (à part n).
2. **4 ou 8 diviseurs.**
   Programme une fonction `quatre_et_huit_diviseurs(Nmin,Nmax)` qui renvoie deux nombres : (1) le nombre d'entiers n avec $N_{\text{min}} \leqslant n < N_{\text{max}}$ qui admettent exactement 4 diviseurs et (2) le nombre d'entiers n avec $N_{\text{min}} \leqslant n < N_{\text{max}}$ qui admettent exactement 8 diviseurs.
   Par exemple `quatre_et_huit_diviseurs(1,100)` renvoie `(32, 10)` car il y a 32 entiers entre 1 et 99 qui admettent 4 diviseurs, mais seulement 10 entiers qui en admettent 8.
3. **Preuve que la conjecture est fausse.**
   Expérimente que pour des « petites » valeurs de N (jusqu'à $N = 10\,000$ par exemple) il y a plus d'entiers ayant 4 diviseurs que 8. Mais calcule que pour $N = 300\,000$ ce n'est plus le cas.
   Indications. Comme il y a beaucoup de calculs, tu peux les séparer en tranches (la tranche des entiers $1 \leqslant n < 50\,000$, puis $50\,000 \leqslant n < 100\,000$,...) puis fais la somme. Tu peux ainsi partager tes calculs entre plusieurs ordinateurs.

Activité 3 (121111… n'est jamais premier ?)**.**

Objectifs : étudier une nouvelle conjecture fausse !

On appelle U_k l'entier :
$$U_k = 12\underbrace{111\ldots111}_{k \text{ occurrences de } 1}$$
formé du chiffre 1, puis du chiffre 2, puis de k fois le chiffre 1.
Par exemple $U_0 = 12$, $U_1 = 121$, $U_2 = 1211$,…

1. Écris une fonction `un_deux_un(k)` qui renvoie l'entier U_k.
   Indications. Tu peux remarquer qu'en partant de $U_0 = 12$, on a la relation $U_{k+1} = 10 \cdot U_k + 1$. Donc tu peux partir de `u = 12` et répéter un certain nombre de fois `u = 10*u + 1`.
2. Vérifie à l'aide de la machine que $U_0, \ldots, U_{20}$ ne sont pas des nombres premiers.

   On pourrait croire que c'est toujours le cas, mais ce n'est pas vrai ! L'entier U_{136} est un nombre premier ! Malheureusement il est trop grand pour qu'on puisse le vérifier avec nos algorithmes. Dans la suite on va définir ce qu'est un nombre presque premier *pour pouvoir pousser plus loin les calculs.*

3. Programme une fonction `est_presque_premier(n,r)` qui renvoie « vrai » si l'entier n n'admet aucun diviseur d tel que $1 < d \leqslant r$ (on suppose $r < n$).
   Par exemple : $n = 143 = 11 \times 13$ et $r = 10$, alors `est_presque_premier(n,r)` est « vrai » car n n'admet aucun diviseur inférieur ou égal à 10. (Mais bien sûr, n n'est pas un nombre premier.)
   Indications. Modifie ta fonction `est_premier(n)` !
4. Trouve tous les entiers U_k avec $0 \leqslant k \leqslant 150$ qui sont presque premiers pour $r = 1\,000\,000$ (c'est-à-dire qu'ils ne sont divisibles par aucun entier d avec $1 < d \leqslant 1\,000\,000$).
   Indications. Dans la liste tu dois retrouver U_{136} (qui est un nombre premier) mais aussi U_{34} qui n'est pas premier mais dont le plus petit diviseur est 10 149 217 781.

Activité 4 (Racine carrée entière)**.**

Objectifs : calculer la racine carrée entière d'un entier.

Soit $n \geqslant 0$ un entier. La ***racine carrée entière de n*** est le plus grand entier $r \geqslant 0$ tel que $r^2 \leqslant n$. Une autre définition est de dire que la racine carrée entière de n est la partie entière de $\sqrt{n}$.
Exemples :

- $n = 21$, alors la racine carrée entière de n est 4 (car $4^2 \leqslant 21$, mais $5^2 > 21$). Autre façon, $\sqrt{21} = 4.58\ldots$, on ne retient que la partie entière (l'entier à gauche de la virgule), c'est donc 4.
- $n = 36$, alors la racine carrée entière de n est 6 (car $6^2 \leqslant 36$, mais $7^2 > 36$). Autre façon, $\sqrt{36} = 6$ et la racine carrée entière est bien sûr aussi 6.

1. Écris une première fonction qui calcule la racine carrée entière d'un entier n, en calculant d'abord $\sqrt{n}$, puis en prenant la partie entière.
   Indications.
   - Pour cette question uniquement, tu peux utiliser le module `math` de `Python`.
   - Dans ce module `sqrt()` renvoie la racine carrée réelle.
   - La fonction `floor()` du même module renvoie la partie entière d'un nombre.
2. Écris une deuxième fonction qui calcule la racine carrée entière d'un entier n, mais cette fois selon la méthode suivante :
   - Pars de $p = 0$.
   - Tant que $p^2 \leqslant n$, incrémente la valeur de p.

   Teste bien quelle doit être la valeur renvoyée (attention au décalage !).
3. Écris une troisième fonction qui calcule encore la racine carrée entière d'un entier n avec l'algorithme décrit ci-dessous. Cet algorithme s'appelle la méthode babylonienne ou bien méthode de Héron ou bien encore méthode de Newton !

   Algorithme.
   Entrée : un entier positif n
   Sortie : sa racine carrée entière
   - Partir avec $a = 1$ et $b = n$.
   - Tant que $|a - b| > 1$:
     — $a \leftarrow (a + b)//2$;
     — $b \leftarrow n//a$
   - Renvoyer le minimum entre a et b : c'est la racine carrée entière de n.

Nous n'expliquons pas comment fonctionne cet algorithme, mais il faut savoir que c'est l'une des méthodes les plus efficaces pour calculer les racines carrées. Les nombres a et b fournissent, au cours de l'exécution, un encadrement de plus en plus précis de $\sqrt{n}$.

Voici un tableau qui détaille un exemple de calcul pour la racine carrée entière de $n = 1664$.

Étape	a	b
$i = 0$	$a = 1$	$b = 1664$
$i = 1$	$a = 832$	$b = 2$
$i = 2$	$a = 417$	$b = 3$
$i = 3$	$a = 210$	$b = 7$
$i = 4$	$a = 108$	$b = 15$
$i = 5$	$a = 61$	$b = 27$
$i = 6$	$a = 44$	$b = 37$
$i = 7$	$a = 40$	$b = 41$

À la dernière étape, l'écart entre a et b est inférieur ou égal à 1, donc la racine carrée entière est 40. On peut vérifier que c'est exact car : $40^2 = 1600 \leqslant 1664 < 41^2 = 1681$.

Bonus. Compare les vitesses d'exécution des trois méthodes à l'aide de `timeit()`. Voir la fiche « Fonctions ».

Cours 1 (Quitter une boucle)**.**

Il n'est pas toujours facile de trouver la condition adéquate pour une boucle « tant que ». Python possède une commande pour quitter immédiatement une boucle « tant que » ou une boucle « pour » : c'est l'instruction `break`.

Voici des exemples qui utilisent la commande `break`. Comme c'est rarement une façon élégante d'écrire son programme, des alternatives sont aussi présentées.

Exemple.

Voici différents codes pour un compte à rebours de 10 à 0.

```
# Compte à rebours
n = 10
# Boucle infinie
while True:
    print(n)
    n = n - 1
    if n < 0:
        break # Arrêt
```

```
# Mieux (avec un drapeau)
n = 10
termine = False
while not termine:
    print(n)
    n = n - 1
    if n < 0:
        termine = True
```

```
# Encore mieux
n = 10
while n >= 0:
    print(n)
    n = n - 1
```

Exemple.
Voici des programmes qui cherchent la racine carrée entière de 777, c'est-à-dire le plus grand entier i qui vérifie $i^2 \leqslant 777$. La recherche est limitée aux entiers i entre 0 et 99.

```
# Racine carrée entière
n = 777
for i in range(100):
    if i**2 >= n:
        break
print(i-1)
```

```
# Mieux
n = 777
i = 0
while (i**2 < n)  and (i < 100):
    i = i + 1
print(i-1)
```

Exemple.
Voici des programmes qui calculent les racines carrés réelles des éléments d'une liste, sauf bien sûr si le nombre est négatif. Le code de gauche s'arrête avant la fin de la liste, alors que le code de droite gère proprement le problème.

```
# Racines carrées des éléments
# d'une liste
liste = [3,7,0,10,-1,12]
for element in liste:
    if element < 0:
        break
    print(sqrt(element))
```

```
# Mieux avec try/except
liste = [3,7,0,10,-1,12]
for element in liste:
    try:
        print(sqrt(element))
    except:
        print("Problème avec",element)
```