NOMBRES DE CARMICHAEL

Problème, 240 minutes

En 1640, Pierre de Fermat énonce ce qu'on appelle aujourd'hui le « petit théorème » dans une lettre. Il a l'idée de l'utiliser pour tester si un nombre est premier ou non. En 1899, l'allemand Alwin Korselt publie, sans doute en vue de démontrer la réciproque du petit théorème de Fermat, une caractérisation des nombres x qui seraient des contre-exemples (pour en déduire que de tels nombres n'existent pas). En 1910, l'américain Robert Carmichael trouve, en utilisant ce critère, les premiers contre-exemples (dont le plus petit d'entre eux : 561). Pour lui rendre hommage on a donné son nom à ces nombres.

Partie I — Factorisation des entiers naturels

On considère un nombre entier x au moins égal à 2. On cherche son plus petit diviseur (en excluant 1 évidemment). Le programme ci-dessous effectue cette tâche.

```
1  def PlusPetitDiviseur(x, d_min = 2) :
2      # Divisibilité par les nombres pairs
3      if d_min == 2 :
4          if x % 2 == 0 :
5              return 2
6          d = 3
7      elif d_min % 2 == 0 :
8          d = d_min + 1
9      else :
10         d = d_min
11     # Parcours des diviseurs potentiels
12     while d * d <= x :
13         if x % d == 0 :
14             return d
15         d += 2
16     return x
```

On peut voir que ce programme teste d'abord la divisibilité par 2 (ligne 4), puis essaie ensuite les diviseurs de deux en deux (ligne 15) à partir de 3 (ligne 6). Ce qui veut dire, en particulier, qu'on ne regarde pas si le nombre est divisible par 4, par 6, par 8, etc..

QUESTION 1.1 — *On suppose que x n'est pas divisible par 2. Démontrer qu'il n'est alors divisible par aucun nombre pair (il est donc inutile de tester la divisibilité par 4, par 6, par 8, etc.).*

Pour gagner du temps, on ajoute une deuxième entrée au programme : un entier $d_{\min}$ qui indique le plus petit diviseur à essayer. On fait l'hypothèse suivante : x n'a aucun diviseur entre 2 et $d_{\min} - 1$.

QUESTION 1.2 — *Soit alors d le plus petit diviseur renvoyé par le programme. Démontrer que d est un nombre premier. On fera une démonstration par l'absurde, en supposant que d possède un diviseur non trivial.*

QUESTION 1.3 — *On suppose que x n'est lui-même pas premier, c'est-à-dire qu'il admet une factorisation non triviale $x = a \times b$ avec $a, b \geqslant 2$. Pour fixer les idées, disons que $a \leqslant b$. Démontrer que $a \leqslant \sqrt{d}$.*

QUESTION 1.4 — *Est-il possible d'atteindre la ligne 16 du programme (c'est-à-dire que x n'ait aucun diviseur entre 2 et $\sqrt{x}$) ?*

De la question précédente, on déduit le programme ci-dessous, qui teste si x est premier et renvoie un booléen (`True` ou `False` en Python) pour le dire. C'est le test de primalité *naïf*.

```
def EstPremier(x) :
    if x <= 1 :
        return False
    else :
        return PlusPetitDiviseur(x) == x
```

QUESTION 1.5 — *À quel endroit y a-t-il une boucle dans ce programme ? Combien d'itérations (en fonction de x) cette boucle effectue-t-elle au maximum ?*

On admet que dans le cas le plus défavorable, c'est-à-dire lorsque x est un nombre premier, le temps d'exécution du test de primalité naïf est proportionnel à une puissance de x, disons x^{α}. On a mesuré le temps d'exécution pour quelques valeurs de x (toutes des nombres premiers).

```
>>> from time import time as Maintenant
>>> déb = Maintenant() ; EstPremier(10 ** 15 + 37) ; fin = Maintenant()
True
>>> fin - déb
1.7058842182159424
>>> déb = Maintenant() ; EstPremier(10 ** 16 + 61) ; fin = Maintenant()
True
>>> fin - déb
5.451136589050293
>>> déb = Maintenant() ; EstPremier(10 ** 17 + 3) ; fin = Maintenant()
True
>>> fin - déb
17.165472745895386
```

valeur de x	$10^{15}+37$	$10^{16}+61$	$10^{17}+3$
temps d'exécution (en s)	1,705	5,451	17,165

QUESTION 1.6 — *Déterminer la valeur de l'exposant α.*

On termine cette partie en donnant deux algorithmes de factorisation, basés (comme le test de primalité naïf) sur le programme auxiliaire `PlusPetitDiviseur`.

```
def Factoriser(x) :
    F = [] ; d_min = 2
    while x > 1 :
        p = PlusPetitDiviseur(x, d_min)
        v = 1 ; x //= p ; d_min = p + 1
        while x % p == 0 :
            v += 1 ; x //= p
        F.append( (p, v) )
    return F
```

```
def FacteursPremiers(x) :
    P = [] ; d_min = 2
    while x > 1 :
        p = PlusPetitDiviseur(x, d_min)
        P.append(p) ; x //= p ; d_min = p
    return P
```

QUESTION 1.7 — *Tester ces algorithmes sur quelques valeurs de x, puis expliquer en quoi ils diffèrent.*

```
>>> Factoriser(1001)
[(7, 1), (11, 1), (13, 1)]
>>> FacteursPpremiers(1001)
[7, 11, 13]

>>> Factoriser(1728)
[(2, 6), (3, 3)]
>>> FacteursPremiers(1728)
[2, 2, 2, 2, 2, 2, 3, 3, 3]
```

Partie II — Un test de primalité ?

On rappelle l'un des énoncés du *petit théorème de Fermat* : soit p un nombre premier et soit a un entier premier avec p, alors $a^{p-1} \equiv 1\ [p]$. Sans transition, on rappelle l'algorithme « des restes successifs » d'Euclide qui calcule le PGCD de deux entiers relatifs.

```
def PGCD(x, y) :
    x = abs(x) ; y = abs(y)
    while y > 0 :
        (x, y) = (y, x % y)
    return x
```

QUESTION 2.1 — *Rappeler la définition de « a est premier avec p » lorsque a et p sont deux entiers relatifs. Comment se simplifie cette définition dans le cas particulier où p est un nombre premier ?*

Maintenant on se lance dans une observation expérimentale : que se passe-t-il lorsque p n'est pas premier ? La fonction prédéfinie `pow(x, n, m)` renvoie le reste dans la division euclidenne de x^n par m.

```
>>> p = 21
>>> [(a, pow(a, p - 1, p)) for a in range(0, p)]
[( 0,  0), ( 1,  1), ( 2,  4), ( 3,  9), ( 4, 16), ( 5,  4), ( 6, 15),
 ( 7,  7), ( 8,  1), ( 9, 18), (10, 16), (11, 16), (12, 18), (13,  1),
 (14,  7), (15, 15), (16,  4), (17, 16), (18,  9), (19,  4), (20,  1)]
```

On a affiché ci-dessus les valeurs de a^{p-1} modulo p pour tous les a de 0 à $p-1$. On voit qu'en effet pour $a = 1$ et $a = p-1$ on obtient 1, et que lorsque a n'est pas premier avec p on n'obtient pas 1.

QUESTION 2.2 — *Pourquoi a-t-on $0^{p-1} \equiv 0$ et $1^{p-1} \equiv 1$ même lorsque p n'est pas premier ?*

QUESTION 2.3 — *Et, lorsque p est impair, pourquoi a-t-on $(p-1)^{p-1} \equiv 1$ même lorsque p n'est pas premier ?*

QUESTION 2.4 — *Dans le cas où $p = 21$, énumérer tous les a entre 2 et $p-2$ qui ne sont pas premiers avec p. Vérifier ce qui a été affirmé ci-dessus : pour ces valeurs de a, a^{p-1} n'est pas égal à 1 modulo p.*

QUESTION 2.5 — *Calculer, toujours dans le cas où $p = 21$, la proportion de valeurs de a entre 2 et $p-2$ pour lesquelles on a $a^{p-1} \equiv 1\ [p]$.*

QUESTION 2.6 — *Plus généralement, écrire un programme* `ÉtudeExpérimentale(p)` *qui pour un nombre composé p au moins égal à 4 calcule la proportion de valeurs de a entre 2 et $p-2$ pour lesquelles on a $a^{p-1} \equiv 1\ [p]$.*

```
>>> ÉtudeExpérimentale(89 * 257)
0.002710975076519458
```

Cette proportion étant très faible, on en déduit le test de primalité suivant : on choisit quelques valeurs de a au hasard et on calcule a^{x-1} modulo x. Si l'on obtient 1 à chaque fois, on affirme (avec une petite chance de se tromper) que x est premier. Sinon, on dit qu'il ne l'est pas (et là c'est une certitude). C'est le test de primalité de Fermat.

```
from random import randint as EntAléa
def Fermat(x, NB_ESSAIS = 5) :
    for _ in range(NB_ESSAIS) :
        a = EntAléa(2, x - 2)
        y = pow(a, x - 1, x)
        if y != 1 :
            return False
    return True
```

Partie III — Réciproque au petit théorème de Fermat

La réciproque du petit théorème de Fermat serait : soit p un entier au moins égal à 2, si pour tout a premier avec p on a $a^{p-1} \equiv 1\ [p]$ alors p est premier. Malheureusement cette réciproque est fausse : les contre-exemples x s'appellent les *nombres de Carmichael*, et on va les chercher dans cette partie.

```
def EstCarmichael(x) :
    if EstPremier(x) :
        return False
    for a in range(1, x) :
        if PGCD(a, x) == 1 :
            y = pow(a, x - 1, x)
            if y != 1 :
                return False
    return True
```

QUESTION 3.1 — *Écrire un programme* `ProchainCarmichael(`$x_{\min}$ `= 2)` *qui renvoie le plus petit nombre de Carmichael x supérieur ou égal à $x_{\min}$.*

```
>>> ProchainCarmichael()
561
```

Le plus petit contre-exemple à la réciproque du petit théorème de Fermat est donc $x = 561$.

QUESTION 3.2 — *En déduire le programme* `PetitsCarmichael(n)` *qui renvoie la liste des n plus petits nombres de Carmichael.*

```
>>> PetitsCarmichael(10)
[561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341]
```

QUESTION 3.3 — *Enfin écrire un programme* `NombresDeCarmichael(M)` *qui renvoie la liste de tous les nombres de Carmichael inférieurs ou égaux à* M.

```
>>> L = NombresDeCarmichael(10 ** 6)
>>> len(L)
43
```

Bon, il n'y en a pas tant que ça !

Partie IV — Nombres sans facteur carré

On dit qu'un entier x est *sans facteur carré* lorsque le seul $d \geqslant 1$ tel que d^2 divise x est $d = 1$. Par exemple 10 et 11 sont sans facteur carré, mais 12 ne l'est pas (puisqu'il se divise par $4 = 2^2$).

QUESTION 4.1 — *Soit x un entier strictement positif. Démontrer que x est sans facteur carré si seulement si dans sa décomposition en produit de nombres premiers, toutes les valuations valent 0 ou 1.*

QUESTION 4.2 — *Écrire le programme* `EstSansFacteurCarré(x)` *qui renvoie* `True` *ou* `False` *selon que x est sans facteur carré ou non.*

QUESTION 4.3 — *Écrire un programme* `SimplifierRacine(a)` *qui étant donné un entier strictement positif a renvoie le couple (k, d), avec d sans facteur carré, tel que $\sqrt{a} = k \times \sqrt{d}$.*

```
>>> SimplifierRacine(4000)
(20, 10)
```

Partie V — Caractérisation de Korselt

Il s'agit du théorème suivant : un entier naturel composé x est un nombre de Carmichael si et seulement si il est sans facteur carré et si, pour tout facteur premier p de x, le nombre $p - 1$ divise $x - 1$.

QUESTION 5 — *Écrire une nouvelle version du programme* `EstCarmichael(x)`*, basée sur ce critère.*

En utilisant cette nouvelle version, on peut trouver le plus petit nombre de Carmichael supérieur à un milliard (quelques minutes de calcul sur mon ordinateur).

```
>>> ProchainCarmichael(10 ** 9)
1001152801
```

FIN