# *Partie n°1* : Primalité

a) Écrire un programme qui détermine si un nombre entier *n* est premier (si *n* n'a que deux diviseurs) ou s'il est composé. Dans ce dernier cas, donner la décomposition en facteurs premiers de *n*.

Pour déterminer si un nombre entier *n* est premier, il suffit de diviser ce nombre, successivement, par tous les entiers $a \geq 2$. Si une seule de ces divisions tombe juste alors le nombre est composé et on a déjà un des facteurs de la décomposition, sinon le nombre est premier. Cet algorithme n'est pas très raffiné car il oblige à faire des divisions inutiles : pourquoi tenter la division par 4 si la division par 2 ne tombe pas juste ? Mais, néanmoins, nous allons essayer ce programme, ne serait-ce que pour en mesurer les performances.

```
n=int(input(" Saisissez un nombre : "))
isPrem=True
listDiviseur=list()
for i in range(2,n):
    if n%i==0 :
        isPrem=False
        listDiviseur.append(i)   # liste des diviseurs de n
if isPrem : print("oui, "+str(n)+" est un nombre premier.")
else :
    s=decomposition(n,listDiviseur)
    print("non, "+str(n)+" n'est pas un nombre premier.")
    print("Voici sa décomposition :"+s)
```

```
 Saisissez un nombre : 7001
oui, 7001 est un nombre premier.

 Saisissez un nombre : 7003
non, 7003 n'est pas un nombre premier.
Voici sa décomposition :7003=47e1×149e1

 Saisissez un nombre : 7007
non, 7007 n'est pas un nombre premier.
Voici sa décomposition :7007=7e2×11e1×13e1
```

```
def decomposition(n,L) :# Décompose n en facteurs premiers
    s=str(n)+"="        # à partir de la liste de ces diviseurs
    decompo=list()
    Lpremier=[L[0]]     # liste contenant les facteurs premiers
    while len(L)!=0 :
        L=[a for a in L if a%Lpremier[-1]!=0] # expurgation de la liste
        if len(L)!=0 :Lpremier.append(L[0])   # des diviseurs
    i=0
    while n>1 :
        expo=0
        while n%Lpremier[i]==0 : # recherche de l'exposant pour
            expo+=1              # chaque facteur premier
            n//=Lpremier[i]
        decompo.append([Lpremier[i],expo])
        i+=1
    for rang,facteur in enumerate(decompo) :
        s+=str(decompo[rang][0])+"e"+str(decompo[rang][1])
        if rang<len(decompo)-1 :
            s+='\u00D7'          # écriture de la décomposition
    return s                     # dans une chaîne de caractères
```

L'algorithme principal (à gauche) examine si le nombre est premier et, s'il ne l'est pas, il stocke dans une liste les diviseurs. Cette liste est ensuite envoyée à la fonction *decomposition*() qui récupère tout d'abord les facteurs premiers. On stocke pour ce faire, le 1er diviseur (le plus petit, qui est forcément premier) dans une autre liste, et on expurge la liste des diviseurs de tous les diviseurs qui sont obtenus pas composition de ce facteur premier. On recommence jusqu'à avoir vidé la liste des diviseurs. On cherche alors la multiplicité (l'exposant) de chacun des diviseurs premiers.

Cet algorithme est très simple − il colle naïvement à la définition des nombres premiers − mais très laborieux. Il effectue *n* divisions : beaucoup de travail inutile. Testons-le avec un nombre plus grand que 7007 : pour $M_{11}=2^{11}-1=2047$ la réponse est instantanée, pour $M_{23}=2^{23}-1=8388607$, c'est beaucoup moins efficace : environ 8 secondes et pour $M_{29}=2^{29}-1=536870911$, il ne faut pas moins de 8 minutes. Ce nombre est 64 fois plus grand que le précédent, et il faut environ 64 fois plus de temps (il y a juste un peu plus de 536 millions de divisions à faire dans la partie principale du programme).

```
 Saisissez un nombre : 8388607
non, 8388607 n'est pas un nombre premier.
Voici sa décomposition :8388607=47e1×178481e1
```

8 secondes

```
 Saisissez un nombre : 536870911
non, 536870911 n'est pas un nombre premier.
Voici sa décomposition :536870911=233e1×1103e1×2089e1
```

8 minutes

La première amélioration va venir du fait que toutes les divisions ne sont pas nécessaires : s'il y a un facteur qui divise le nombre, ce facteur est forcément plus petit ou égal à la racine carrée du nombre . Car si $n=a\times b$, alors soit *a* soit *b* est inférieur ou égal à $\sqrt{n}$. On peut aussi supprimer tous les nombres pairs (sauf 2). Ces deux simples idées conduit à une amélioration qui diminue le nombre d'opérations à faire d'une manière très efficace. Pour examiner la primalité de $M_{29}$, on n'effectue les divisions que jusqu'à $\sqrt{M_{29}}\approx 23170$, et encore, on n'en effectue que la moitié, il y a donc environ 11585 divisions à effectuer au lieu de 536870911, ce qui revient à diviser le temps d'exécution d'un facteur 46342 environ. Les 8 minutes deviennent un centième de seconde !

Avec cette amélioration, il y a une adaptation à faire pour la fonction *decomposition*() qui prend en argument la liste des diviseurs du nombre *n* : en n'effectuant pas toutes les divisions, cette liste ne peut pas être complète ! Pour la compléter, nous avons mis au point la nouvelle fonction *complement*() qui ajoute les grands diviseurs (supérieurs à $\sqrt{n}$) obtenus en divisant *n* par la liste des premiers diviseurs. La liste complète des diviseurs a besoin d'être triée pour les besoins de la fonction *decomposition*(), nous utilisons donc cette méthode *sort*() de la classe *list*. Cette amélioration considérable ne suffit pourtant pas pour examiner, en temps raisonnable, la primalité de nombres beaucoup plus grands que $M_{29}$.

Pour commencer, notre instruction Ldiviseur=[2]+*list*(*range*(3,*int*(nRacine),2)) ne semblant pas plaire à Python pour certains grands nombres comme $M_{101}=2^{101}$ qui contient 31 chiffres (la liste à créer est trop

grande pour la mémoire), il fallait contourner ce problème technique. Une boucle *while* fait aussi bien l'affaire que cette liste et, cette fois, Python peut exécuter le programme mais sans nécessairement aboutir : il y a environ 796 131 459 065 721 opérations à effectuer pour examiner la primalité de $M_{101}$ et 8 minutes ne suffisent pas, loin de là. Si on fait le calcul, le processeur de notre ordinateur effectuant environ 1 118 481 opérations par secondes, il faudrait environ 542 années... Un peu trop long à attendre. Soyons plus raisonnable et examinons le cas de $M_{43}=2^{43}-1=8796093022207$ qui ne nécessite que 1 482 910 opérations : cette fois 1,4 secondes suffisent.

amélioration 1 :

forme utilisable pour les petits nombres

```
def complement(n,L) :
    for diviseur in L :
        nouvDiv=n//diviseur
        if nouvDiv not in L : L.append(n//diviseur)
    return L

... même début que le programme initial ....
Ldiviseur=[2]+list(range(3,int(nRacine),2))
for i in Ldiviseur:
    if n%i==0 :
        isPrem=False
        listDiviseur.append(i)
if isPrem : print("oui, "+str(n)+" est un nombre premier.")
else :
    listDiviseur=complement(n,listDiviseur)
    listDiviseur.sort()
    s=decomposition(n,listDiviseur)
    ... même fin que le programme initial ....
```

```
Saisissez un nombre : 536870911
non, 536870911 n'est pas un nombre premier.
Voici sa décomposition :536870911=233e1×1103e1×2089e1
```

0.01 seconde

forme utilisable pour les grands nombres

```
... même début que le programme initial ....
if n%2==0 :
    isPrem=False
    listDiviseur.append(2)
i=3
while i<=nRacine :
    if n%i==0 :
        isPrem=False
        listDiviseur.append(i)
    i+=2
if isPrem : print("oui, "+str(n)+" est un nombre premier.")
else :
    listDiviseur=complement(n,listDiviseur)
    listDiviseur.sort()
    s=decomposition(n,listDiviseur)
    ... même fin que le programme initial ....
```

```
Saisissez un nombre : 8796093022207
non, 8796093022207 n'est pas un nombre premier.
Voici sa décomposition :8796093022207=431e1×9719e1×2099863e1
```

1.40 secondes

b) Utiliser ce programme pour montrer qu'à partir de $k$=5 les nombres de Fermat $F_k=2^{2^k}+1$ ne sont pas premiers.

```
oui, F1 = 5 est un nombre premier.
oui, F2 = 17 est un nombre premier.
oui, F3 = 257 est un nombre premier.
oui, F4 = 65537 est un nombre premier.
non, F5 = 4294967297 n'est pas un nombre premier.
Voici sa décomposition :4294967297=641e1×6700417e1
non, F6 = 18446744073709551617 n'est pas un nombre premier.
Voici sa décomposition :18446744073709551617=274177e1×67280421310721e1
```

Ces nombres, Pierre de Fermat (1601-1665) pensait qu'ils étaient tous premiers. Il avait calculé les quatre premiers et, effectivement, les nombres 5, 17, 257 et 65537 sont tous premiers (on peut ajouter le nombre $F_0=2^0+1=3$ qui est premier aussi). La conjecture qu'il énonça alors (en 1640) s'est avérée fausse car le 5ème nombre n'est pas premier. Fermat ne l'avait pas vérifié, il faut dire que 4 294 967 297 est un grand nombre et que son premier facteur premier est assez grand aussi (c'est 641, le 116ème nombre premier, il fallait faire 116 divisions...). C'est Euler qui montra, en 1732, que ce nombre $F_5=2^{2^5}+1=2^{32}+1$ est composé $F_5=641\times 6\,700\,417$. Pour prouver cela, Euler démontra ce théorème : tout facteur premier d'un nombre de Fermat $F_n$ est de la forme $k\,2^{n+1}+1$ où $k$ est un entier. Ainsi, pour $F_5$, il suffisait de diviser par des nombres de la forme $64k+1$, et pour $k$=10, il trouva le premier diviseur de $F_5$. Quant aux autres nombres de Fermat, on en cherche encore des premiers (de $F_5$ jusqu'à $F_{32}$ ils sont tous composés). Voyons cela à l'aide de notre programme, en l'adaptant au problème : une petite boucle supplémentaire nous évitera d'entrer les valeurs successives de $k$. Nous n'irons pas bien loin car les nombres croissent très vite et pour $k$=6, le nombre a 20 chiffres, ce qui pose un problème de temps (plus de 45 minutes de calcul).

```
oui, M2 = 3 est un nombre premier.
oui, M3 = 7 est un nombre premier.
oui, M5 = 31 est un nombre premier.
oui, M7 = 127 est un nombre premier.
non, M11 = 2047 n'est pas un nombre premier.
Voici sa décomposition :2047=23e1×89e1
oui, M13 = 8191 est un nombre premier.
oui, M17 = 131071 est un nombre premier.
oui, M19 = 524287 est un nombre premier.
non, M23 = 8388607 n'est pas un nombre premier.
Voici sa décomposition :8388607=47e1×178481e1
non, M29 = 536870911 n'est pas un nombre premier.
Voici sa décomposition :536870911=233e1×1103e1×2089e1
oui, M31 = 2147483647 est un nombre premier.
non, M37 = 137438953471 n'est pas un nombre premier.
Voici sa décomposition :137438953471=223e1×616318177e1
non, M41 = 2199023255551 n'est pas un nombre premier.
Voici sa décomposition :2199023255551=13367e1×164511353e1
non, M43 = 8796093022207 n'est pas un nombre premier.
Voici sa décomposition :8796093022207=431e1×9719e1×2099863e1
non, M47 = 140737488355327 n'est pas un nombre premier.
Voici sa décomposition :140737488355327=2351e1×4513e1×13264529e1
non, M53 = 9007199254740991 n'est pas un nombre premier.
Voici sa décomposition :9007199254740991=6361e1×69431e1×20394401e1
non, M59 = 576460752303423487 n'est pas un nombre premier.
Voici sa décomposition :576460752303423487=179951e1×3203431780337e1
oui, M61 = 2305843009213693951 est un nombre premier.
```

c) Montrer, de même, que pour $k$=2, 3, 5, 7 les nombres de Mersenne, notés $M_k=2^k-1$, avec $k$ premier, sont premiers (on les note alors *M1, M2, M3* et *M4*), alors que pour $k$=11, $M_k$ n'est pas premier. Déterminer la valeur de $k$ pour le 5ème nombre de Mersenne premier, noté *M5*.
Nous avons déjà examiné la primalité de quelques nombres de Mersenne. Pour les examiner dans l'ordre, nous allons adapter notre boucle pour les tester tous à partir de $k$=2, 3, 5, etc. Pour ce faire, nous écrivons la liste *Lprem* des nombres premiers jusqu'à 101 et nous utilisons cette fois une boucle for k *in* Lprem. Les premiers résultats sont très rapides à s'inscrire, mais, dès que les nombres sont plus grands, les temps de calcul s'allongent : c'est quasi-instantané jusqu'à $M_{37}=2^{37}-1=137\,438\,953\,471$, mais il faut une vingtaine de minutes pour aller jusqu'à la primalité de $M_{59}=2^{59}-1=576460752303423487$, donc nous ne pouvons pas espérer aller beaucoup plus loin avec cet algorithme.
Quoi qu'il en soit, contrairement à ce qu'avait conjecturé Marin Mersenne à l'époque de Fermat, les nombres dits « de Mersenne » ne sont pas tous premiers. Le premier qui ne l'est pas est relativement petit puisqu'il s'agit de $M_{11}=2^{11}-1=2047=23\times89$. C'est encore Euler qui démonta en 1732 cette fausse conjecture. Il donna aussi d'autres contre-exemples : $M_{23}$, $M_{83}$, $M_{13}$, $M_{179}$, $M_{191}$ et $M_{239}$ et prouva par contre que $M_{31}$ était premier (c'était le 8ème nombre de Mersenne premier connu). Pourquoi s'intéressait-on tant aux nombres premiers de Mersenne à cette époque ? Parce qu'ils étaient liés aux nombres parfaits par la propriété suivante, connue depuis Euclide, au IVème siècle avant J.-C. : si $M=2^n-1$ est premier alors $\frac{M(M+1)}{2}=2^{p-1}(2^p-1)$ est un nombre parfait (l'intérêt pour les nombres parfaits est moins évident à comprendre). Ainsi, puisque $M4=M_7=127$ est premier, alors $\frac{127\times128}{2}=64\times127=2^6(2^7-1)=8128$ est un nombre parfait (le premier nombre parfait est 6=1+2+3, il correspond à $M1=M_2=3$). En effet, ses diviseurs propres sont 1, 2, 4, 8, 16, 32, 64, 127, 254, 508, 1016, 2032 et 4064 et leur somme est égale au nombre lui-même :
1+2+4+8+16+32+64+127+254+508+1016+2032+4064=8128.
Jusqu'à l'avènement des ordinateurs, au milieu du XXème siècle, on découvrit jusqu'à 12 nombres de Mersenne premiers $M12=M_{127}$ (c'est un nombre qui s'écrit avec 39 chiffres). Aujourd'hui, on continue à s'intéresser aux grands nombres de Mersenne premiers car ils sont utiles en cryptographie. Le plus grand nombre premier connu est un nombre de Mersenne, il s'agit de $M48=M_{57885161}$ (il s'écrit avec 17 425 170 chiffres) qui fut découvert en 2013 par un projet de recherche collaborative, la GIMPS, qui élevait pour la 14ème fois le record du nombre premier le plus grand en seulement 17 ans d'existence.

d) Il est possible de se focaliser sur la primalité des nombres de Mersenne car un théorème dû au mathématicien français Lucas (1842-1891), permet d'accroître notablement l'efficacité du test : étant donnée la suite $(s_n)$ définie par $s_1$=4 et, pour tout $n$>1, $s_n=(s_{n-1})^2-2$, le nombre $M_n=2^n-1$ est premier si et seulement si il divise $s_{n-1}$. La difficulté ici va être la croissance extrêmement rapide du nombre de chiffres des termes de la suite $(s_n)$. Par exemple, pour tester si $M_{11}=2^{11}-1=2047$ est premier, il suffit de tester si ce nombre divise $s_{10}$. Le problème est que $s_{10}$ s'écrit avec près de 300 chiffres ! Essayer de voir jusqu'où, avec Python sur une machine ordinaire et dans des temps raisonnables, cette simple suite permet d'examiner la primalité des nombres de Mersenne.
Pour ce programme nous allons oublier la décomposition des nombres de Mersenne non premiers, car la liste des diviseurs n'est pas accessible si on s'en tient au seul critère de Lucas. Nous nous bornons à calculer les termes de la suite $(s_n)$ et à chercher le reste de la division de $s_{n-1}$ par $M_n$. Si ce reste est nul, le nombre est premier. Lucas, à son époque sans ordinateur, parvint à prouver la primalité de $M_{127}$ qui s'écrit avec 39 chiffres, sans faire le calcul explicite de $s_{126}$. Mais nous allons faire les calculs, enfin Python va s'en charger. Le programme est court, la difficulté est d'insérer une boucle pour le calcul de $s_{n-1}$. On doit veiller à utiliser les bons indices. Une mise au point est parfois utile : en affichant le terme utilisé de cette suite qui commence par 4, 14, 194, 37 634, 1 316 317 954, etc. on s'aperçoit d'une erreur éventuelle. Une remarque : pour $n$=2, le test ne fonctionne pas, car 4 n'est pas divisible par 3.
La bonne formulation du théorème de Lucas serait : « pour $n$>2, le nombre $M_n=2^n-1$ est premier si et seulement si il divise $s_{n-1}$ » (ou bien pour $n$ impair...). Avec ce test fabuleux, qui permet d'aller si loin dans l'étude de la primalité des nombres de Mersenne, nous n'avons pas réussi à aller plus loin que $M_{23}$ ! Le nombre $s_{22}$ utilisé possède déjà un million deux cent mille chiffres... Sans doute qu'il faudrait procéder autrement pour améliorer cette bien piètre performance.

```
Lprem=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,\
       53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101] #quelques nombres premiers
s=4               # terme s(1)
i=1
for k in Lprem :
    n=2**(k)-1
    while i<k-1 : # calcul du terme s(k-1) de la suite
        s=s**2-2
        i+=1
    isPrem=True
    if s%n!=0 : isPrem=False
    if isPrem : print("oui, M"+str(k)+" = "+str(n)+" est un nombre premier.")
    elif n==2 : print("oui, M2 = 3 est un nombre premier.")# petite exception au test de Lucas
    else :
        print("non, M"+str(k)+" = "+str(n)+" n'est pas un nombre premier.")
        print("car s"+str(i)+" qui s'écrit avec "+str(len(str(s)))+" chiffres, n'est pas un multiple de "+str(n))
```

```
    s10=68729682406644277238837486231747530924247154108646671752192618583088487405799579647328830691025610434367796639355951720423573065949163446060745647128680782876080552030246583594390175808839109786661858757174155410844949265004751673811685059273781818997538392606094522653652748509018798812037140
```

```
non, M2 = 3 n'est pas un nombre premier.
car s1 qui s'écrit avec 1 chiffres, n'est pas un multiple de 3
oui, M3 = 7 est un nombre premier.
oui, M5 = 31 est un nombre premier.
oui, M7 = 127 est un nombre premier.
non, M11 = 2047 n'est pas un nombre premier.
car s10 qui s'écrit avec 293 chiffres, n'est pas un multiple de 2047
oui, M13 = 8191 est un nombre premier.
oui, M17 = 131071 est un nombre premier.
oui, M19 = 524287 est un nombre premier.
non, M23 = 8388607 n'est pas un nombre premier.
car s22 qui s'écrit avec 1199461 chiffres, n'est pas un multiple de 8388607
non, M29 = 536870911 n'est pas un nombre premier.
```

Cette méthode naïve d'application du test de Lucas-Lehmer ne marche pas car les nombres sont trop grands. Pour la faire fonctionner, on n'a pas réellement besoin de calculer ces nombres. Puisqu'on ne se sert que du reste de la division de $s_{n-1}$ par le nombre $M_n$, on peut se contenter des restes de la division des termes de la suite ($s_n$) par le nombre $M_n$. Justifier que le reste final (celui de $s_{n-1}$ par le nombre $M_n$) sera bien le même si on garde les vrais termes de la suite ($s_n$) ou si on ne conserve que les résidus modulo $M_n$ de ces termes n'est pas notre propos ici. Mais cette propriété va nous permettre d'appliquer efficacement le test. Nous pouvons tout de même essayer de comprendre ce qui se passe, en examinant un exemple : pour examiner la primalité de $M_7=127$, on doit examiner la divisibilité de $s_6$=2 005 956 546 822 746 114 par 127. Les termes précédents de la suite sont 4, 14, 194, 37 634 et 1 416 317 954. Si on examine la suite des restes (résidus) de la division par 127, on obtient 4, 14, 67 (194−127), ensuite on calcule $67^2-2=4\,487$ et le reste de 4 487 par 127 est 42 comme celui de 37 634 par 127. Ensuite on calcule $42^2-2=1\,762$ et le reste de 1 762 par 127 est 111 comme celui de 1 416 317 954 par 127 (et comme celui de $4\,487^2-2=20\,133\,167$ par 127 d'ailleurs). Enfin, on calcule $111^2-2=12\,319$ et le reste de 12 319 par 127 est 0 comme celui de

```
Lprem=[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101,\
       103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
r=1 #rang d'un nombre de Mersenne premier
for p in Lprem :
    s=4    # terme s(1)
    for k in range(3,p+1) :
        s=(s**2-2)%(2**p-1)
    if s==0 :
        r+=1
        print("oui, M"+str(k)+" = "+str(2**k-1)+" est un nombre premier. C'est le Mersenne no"+str(r))
    else :
        print("non, M"+str(k)+" = "+str(2**k-1)+" n'est pas un nombre premier. Le résidu est "+str(s))
```

```
oui, M3 = 7 est un nombre premier. C'est le Mersenne no2
oui, M5 = 31 est un nombre premier. C'est le Mersenne no3
oui, M7 = 127 est un nombre premier. C'est le Mersenne no4
non, M11 = 2047 n'est pas un nombre premier. Le résidu est 1736
oui, M13 = 8191 est un nombre premier. C'est le Mersenne no5
oui, M17 = 131071 est un nombre premier. C'est le Mersenne no6
oui, M19 = 524287 est un nombre premier. C'est le Mersenne no7
non, M23 = 8388607 n'est pas un nombre premier. Le résidu est 6107895
non, M29 = 536870911 n'est pas un nombre premier. Le résidu est 458738443
oui, M31 = 2147483647 est un nombre premier. C'est le Mersenne no8
non, M37 = 137438953471 n'est pas un nombre premier. Le résidu est 117093979072
non, M41 = 2199023255551 n'est pas un nombre premier. Le résidu est 856605019673
non, M43 = 8796093022207 n'est pas un nombre premier. Le résidu est 5774401272921
non, M47 = 140737488355327 n'est pas un nombre premier. Le résidu est 96699253829728
non, M53 = 9007199254740991 n'est pas un nombre premier. Le résidu est 5810550306096509
non, M59 = 576460752303423487 n'est pas un nombre premier. Le résidu est 450529175803834166
oui, M61 = 2305843009213693951 est un nombre premier. C'est le Mersenne no9
non, M67 = 147573952589676412927 n'est pas un nombre premier. Le résidu est 44350645312365507266
non, M71 = 2361183241434822606847 n'est pas un nombre premier. Le résidu est 271761692158955752596
non, M73 = 9444732965739290427391 n'est pas un nombre premier. Le résidu est 2941647823169311845731
non, M79 = 604462909807314587353087 n'est pas un nombre premier. Le résidu est 149246123283525944719226
non, M83 = 967140655691703339764907 n'est pas un nombre premier. Le résidu est 74263932232114893531232218
oui, M89 = 618970019642690137449562111 est un nombre premier. C'est le Mersenne no10
non, M97 = 158456325028528675187087900671 n'est pas un nombre premier. Le résidu est 138799132171283648987055810555
non, M101 = 25353012004564588029934064107510 n'est pas un nombre premier. Le résidu est 24574576398683058552749163448886
non, M103 = 10141204801825835211973625643007 n'est pas un nombre premier. Le résidu est 44732754599525451611885099651180
oui, M107 = 162259276829213363391578010288127 est un nombre premier. C'est le Mersenne no11
non, M109 = 649037107316853453566312041152511 n'est pas un nombre premier. Le résidu est 80310482899578688674364643057506
non, M113 = 10384593717069655257060992658440191 n'est pas un nombre premier. Le résidu est 66007912437406701327589192279933377
oui, M127 = 170141183460469231731687303715884105727 est un nombre premier. C'est le Mersenne no12
```

2 005 956 546 822 746 114 par 127. Le fait de pouvoir travailler avec les résidus de la suite modulo $M_n$ plutôt qu'avec la suite directement évite les nombres plus grands que $M_n$. Pourquoi $(194-127)^2-2$ et $(194)^2-2$ ont-ils le même reste dans la division par 127 ? Si on note $R$ ce reste, on a $\frac{194^2}{127}=Q+\frac{R}{127}$ ($Q$ est le quotient entier de cette division) et

$$\frac{(194-127)^2}{127}=\frac{194^2-2\times127\times194+127^2}{127}=\frac{194^2}{127}-2\times194+127=Q+\frac{R}{127}-2\times194+127=Q'+\frac{R}{127}$$

($Q'$ est le nouveau quotient entier de cette division). Donc $\frac{194^2}{127}$ et $\frac{(194-127)^2}{127}$ ont le même reste dans la division par 127. Retrancher 2 à ces deux nombres fera diminuer d'autant le reste commun modulo 127, donc ces deux nombres ont le même reste, et la propriété se propage aux autres termes de la suite.

Cette fois, pour tester les nombres de Mersenne jusqu'à $M_{127}$, cela ne prend qu'une fraction de seconde. Cela ne pose plus de problèmes d'aller plus loin dans la recherche. Pour éviter d'écrire une liste démesurée de nombres premiers (la liste *Lprem*), on supprime le contrôle de la boucle par cette liste et on le remplace par une boucle sur les nombres impairs à partir de 3.

Voici maintenant notre ultime proposition. On y a supprimé les affichages des nombres de Mersenne qui ne sont pas premiers. L'objectif est de lister les nombres de Mersenne premiers jusqu'à $k$=1 000. Nous en avons trouvé 14 en 2 secondes, le dernier étant $M_{607}$ qui comporte 183 chiffres.

```
r=1 #rang d'un nombre de Mersenne premier
p=3
while p < 1000 :
    s=4    # terme s(1)
    for k in range(3,p+1) :
        s=(s**2-2)%(2**p-1)
    if s==0 :
        r+=1
        print("M"+str(p)+" est un nombre premier. Il a "+str(len(str(2**p-1)))+" chiffres. C'est le Mersenne no"+str(r))
    p+=2
```

```
M3 est un nombre premier. Il a 1 chiffres. C'est le Mersenne no2
M5 est un nombre premier. Il a 2 chiffres. C'est le Mersenne no3
M7 est un nombre premier. Il a 3 chiffres. C'est le Mersenne no4
M13 est un nombre premier. Il a 4 chiffres. C'est le Mersenne no5
M17 est un nombre premier. Il a 6 chiffres. C'est le Mersenne no6
M19 est un nombre premier. Il a 6 chiffres. C'est le Mersenne no7
M31 est un nombre premier. Il a 10 chiffres. C'est le Mersenne no8
M61 est un nombre premier. Il a 19 chiffres. C'est le Mersenne no9
M89 est un nombre premier. Il a 27 chiffres. C'est le Mersenne no10
M107 est un nombre premier. Il a 33 chiffres. C'est le Mersenne no11
M127 est un nombre premier. Il a 39 chiffres. C'est le Mersenne no12
M521 est un nombre premier. Il a 157 chiffres. C'est le Mersenne no13
M607 est un nombre premier. Il a 183 chiffres. C'est le Mersenne no14
```

Comme nous pouvons facilement aller plus loin, nous essayons d'aller jusqu'à 10 000. Cela prend tout de même du temps, de plus en plus, au fur et à mesure de l'augmentation de la taille des nombres. Nous voyons progressivement la liste s'allonger : $M_{1279}$, $M_{2203}$, $M_{2281}$, $M_{3217}$, $M_{4253}$, $M_{4423}$. Le dernier nombre de Mersenne trouvé, le nombre M20, comporte 1332 chiffres ! Notre programme peine à en trouver davantage, nous en resterons donc là.

```
M1279 est un nombre premier. Il a 386 chiffres. C'est le Mersenne no15
M2203 est un nombre premier. Il a 664 chiffres. C'est le Mersenne no16
M2281 est un nombre premier. Il a 687 chiffres. C'est le Mersenne no17
M3217 est un nombre premier. Il a 969 chiffres. C'est le Mersenne no18
M4253 est un nombre premier. Il a 1281 chiffres. C'est le Mersenne no19
M4423 est un nombre premier. Il a 1332 chiffres. C'est le Mersenne no20
```