1 Énoncés

Exercice 1 (Suite de Pell)
La suite de Pell est définie par $P_n = 2P_{n-1} + P_{n-2}$, $P_0 = 0$ et $P_1 = 1$.

1. Déterminer à la main les cinq termes suivants de la suite de Pell (de P_2 à P_6).

2. Mettre au point un programme qui détermine P_n connaissant n.
Vérifier votre programme en l'utilisant pour déterminer P_2, P_3 et P_6.
Utiliser ce programme pour déterminer P_{10} puis P_{20}.

3. Modifier votre programme pour qu'il détermine la 1re valeur de n pour laquelle P_n dépasse une valeur M donnée. Utiliser ce programme modifié pour déterminer la 1re valeur de n pour laquelle $P_n > 10^3$ puis, si possible, $P_n > 10^6$, $P_n > 10^9$ et $P_n > 10^{12}$.

4. On souhaite vérifier par le calcul l'hypothèse suivante : le rapport de deux nombres consécutifs de la suite de Pell s'approche de la proportion d'argent $\delta_{Ag} = 1 + \sqrt{2}$ quand n tend vers l'infini. Pour cela, on va modifier le programme pour qu'il détermine le rapport $R_n = \frac{P_n}{P_{n-1}}$ et qu'il détecte la 1re valeur de n pour laquelle $\epsilon_n = |R_n - \delta_{Ag}|$ devient inférieur à une valeur m donnée.
Utiliser ce second programme modifié pour déterminer la 1re valeur de n pour laquelle $\epsilon_n < 10^{-6}$.

Exercice 2 (Suites de Syracuse)
Le terme général de la suite de Syracuse (P) est $u_n = \begin{cases} 3u_{n-1} + 1 & \text{si } n \text{ est pair} \\ \frac{u_{n-1}}{2} & \text{si } n \text{ est impair} \end{cases}$
Le premier terme de cette suite est $u_0 = N$, un entier naturel.

1. Déterminer à la main les dix premiers termes de la suite de Syracuse associée à $N = 10$.
Au vu de ce résultat, quel type de comportement semble caractériser la suite (u_n) ?

2. L'observation effectuée pour $N = 10$ semble se généraliser : pour une certaine valeur de n, on arrive sur le terme $u_n = 1$. Á partir de là, le comportement de la suite est toujours le même.
Mettre au point un programme qui affiche les termes successifs de cette suite (pour plus de clarté, on affichera n et u_n sur une même ligne), jusqu'à l'obtention de $u_n = 1$.
Vérifier votre programme en l'utilisant pour déterminer la suite associée au nombre $N = 10$.
Utiliser ce programme pour déterminer la suite associée aux nombres $N = 15$ et $N = 18$.

3. Améliorer votre programme pour qu'il affiche seulement la plus grande valeur de u_n qui a été atteinte (paramètre appelé « altitude ») et la valeur de n pour laquelle $u_n = 1$ (paramètre appelé « longueur »). Par exemple, pour $N = 10$ il affiche $altitude = 16, longueur = 6$.
Utiliser ce programme amélioré pour déterminer ces paramètres pour $N = 15, 18$ et 127.

Exercice 3 (Un exercice du poly « suites »)
1. On désigne par d_n le ne chiffre de la partie décimale de $\frac{2018}{19}$.
Étudier la périodicité de la suite (d_n) et donner une définition explicite de d_n.
Donner en particulier la valeur de d_{2019}.
2. Mêmes questions si d_n est le ne chiffre de la partie décimale de $\frac{2018}{2019}$.
Indication : Pour cette dernière partie, on peut écrire un programme qui affiche la suite des chiffres d'un quotient jusqu'au retour d'un reste déjà obtenu.

Indication additionnelle
Ici, on ne s'intéressera qu'au programme qui est suggéré dans la question 2 : un programme qui détermine la séquence périodique associée au nombre rationnel $\frac{a}{b}$. Ce programme nécessite une liste L qui contiendra les restes successifs de la division euclidienne de a par b. Il faut pouvoir repérer si un reste que l'on vient d'obtenir appartient déjà à cette liste L des restes. Tant que le reste n'y appartient pas, on continue à diviser le reste (multiplié par dix) par le diviseur...

2 Corrections

Correction de l'exercice 1 (Suite de Pell)
1. Les deux premiers termes sont $P_0 = 0$ et $P_1 = 1$.
On a ensuite $P_2 = 2P_1+P_0 = 2\times1+0 = 2$, $P_3 = 2P_2+P_1 = 2\times2+1 = 5$, $P_4 = 2P_3+P_2 = 2\times5+2 = 12$, $P_5 = 2P_4 + P_3 = 2 \times 12 + 5 = 29$ et $P_6 = 2P_5 + P_3 = 2 \times 29 + 12 = 70$.
Après cela, on a $P_7 = 169$, $P_8 = 408$, $P_9 = 985$, $P_{10} = 2378$, $P_{11} = 5741$, $P_{12} = 13860$, $P_{13} = 33461$, $P_{14} = 80782$, $P_{15} = 195025$, $P_{16} = 470832$, $P_{17} = 1136689$, $P_{18} = 2744210$, etc.

2. Le programme est très simple. Il utilise une boucle « Pour », car on sait à l'avance combien de tours de boucle on doit faire (on connait le n final). Voici ce programme en Python :

```
def pell1(n):
    grandpere,pere=0,1
    for i in range(2,n+1):
        fils=2*pere+grandpere
        grandpere,pere=pere,fils
    print(pere)

def pell2(n):
    a,b=0,1
    for i in range(2,n+1):
        a,b=b,2*b+a
    print(b)
```

```
def pell3(n):
    a=0
    b=1
    for i in range(2,n+1):
        c=2*b+a
        a=b
        b=c
        print("P(",i,")=",b)
n=6
pell1(n)
pell2(n)
pell3(n)
```

```
70
70
P( 2 )= 2
P( 3 )= 5
P( 4 )= 12
P( 5 )= 29
P( 6 )= 70
```

J'ai proposé trois versions différentes du programme. La dernière utilise une variable tampon (la variable c) ce qui est inutile en Python (voir les deux autres versions) mais utile dans d'autres langages (le Basic de Casio ou de TI). Dans cette dernière version, j'ai aussi opté pour un affichage des différents termes de la suite, ce qui n'était pas demandé et qui peut s'avérer gênant s'il y a beaucoup de termes.
On vérifie avec ce programme que $P_{10} = 2378$ et $P_{20} = 2744210$.

3. La suite P étant très évidemment croissante et non majorée, les termes vont dépasser n'importe quelle valeur M fixée à l'avance. Pour montrer expérimentalement cela, on améliore le programme précédent en remplaçant la boucle « Pour » par une boucle « Tant que ». Cette fois, on ne sait pas à l'avance jusqu'où il faut aller (on ne connait pas la valeur du n final). On teste donc à chaque tour si on a dépassé ou non la valeur de M :

```
def pell(M):
    a,b,n=0,1,1
    while b<M:
        a,b,n=b,2*b+a,n+1
    print("P(",n,")=",b,">",M)
```

```
pell(10**6)
pell(10**9)
pell(10**12)
```

```
P( 17 )= 1136689 > 1000000
P( 25 )= 1311738121 > 1000000000
P( 33 )= 1513744654945 > 1000000000000
```

Ici, on doit gérer la valeur de l'indice n : au début, on l'initialise à 1 puisque P_1 est connu ; dans la boucle, on l'incrémente de 1. Il est extrêmement recommandé de tester son programme pour des valeurs connues avant de le lancer sur des valeurs inconnues : on essaie donc de déterminer n pour que $P_n > 50$ (on connait la réponse, c'est 6 d'après les questions précédentes).
Ensuite, on détermine que $P_{17} > 10^6$, $P_{25} > 10^9$ et $P_{33} > 10^{12}$.

4. Pour déterminer le nombre $\delta_{Ag} = 1+\sqrt{2}$ (noté `ag` dans le programme), il est nécessaire en Python d'importer la fonction `sqrt`de la bibliothèque mathématique, d'où l'instruction `from math import sqrt`.
Ici, la boucle « Tant que » est contrôlée par le test `abs(b/a-ag)>m`, mais vous remarquerez que a étant nul au début, cela va produire une erreur. J'ai donc mis un test supplémentaire : on entre dans la boucle si `a==0` ou bien si (`abs(b/a-ag)>m`). Lorsque $a = 0$, le test étant vrai, on n'examine pas la 2e condition (elle provoquerait une erreur). Pour éviter ce problème, on aurait pu s'arranger autrement : par exemple commencer au rang 2 (au lieu du rang 1) avec $a = 1, b = 2, n = 2$. Dans ce cas, on peut supprimer le test `a==0` qui est inutile puisque $a > 0$ dès que $n > 0$.

```
from math import sqrt
def pell4(m):
    a,b,n=0,1,1
    ag=1+sqrt(2)
    while a==0 or abs(b/a-ag)>m:
        a,b,n=b,2*b+a,n+1
    print("P(",n,")=",b,", P(",n-1,")=",a)
    print("epsilon(",n,")=",abs(b/a-ag),"<",m)

pell4(10**(-6))
pell4(10**(-9))
pell4(10**(-12))
```

```
P( 10 )= 2378 , P( 9 )= 985
epsilon( 10 )= 3.644e-07 < 1e-06
P( 14 )= 80782 , P( 13 )= 33461
epsilon( 14 )= 3.1577e-10 < 1e-09
P( 18 )= 2744210 , P( 17 )= 1136689
epsilon( 18 )= 2.73559e-13 < 1e-12
```

Correction de l'exercice 2 (Suites de Syracuse)

1. Les dix premiers termes de la suite de Syracuse associée à $N = 10$ sont indiqués dans le tableau.

$u_0 = 10$; $u_1 = 5$; $u_2 = 16$; $u_3 = 8$; $u_4 = 4$; $u_5 = 2$; $u_6 = 1$
$u_7 = 4$; $u_8 = 2$; $u_9 = 1$; $u_{10} = 4$; $u_{11} = 2$; $u_{12} = 1$; $\cdots$

On constate que la suite (u_n) est périodique à partir de $n = 4$: on obtient une répétition des trois termes $4, 2, 1$ jusqu'à l'infini.

2. La généralisation de cette propriété reste une conjecture sur laquelle se sont heurtées plusieurs générations de mathématiciens. En effet, cela fait 90 ans que le mathématicien Lothar Collatz a remarqué cette propriété : pour une certaine valeur de n, on arrive sur le terme $u_n = 1$ et ensuite, le comportement de la suite est toujours périodique. On n'a pas encore trouvé de contre-exemple qui infirme cette propriété (et pourtant on a essayé jusqu'à plus de six milliards de milliards) et on n'a pas non plus réussi à prouver qu'elle était vraie pour tout N.
Notre programme d'exploration est modeste : il affiche les termes successifs de la suite jusqu'à l'obtention de $u_n = 1$. Il s'agit donc, bien entendu, d'une boucle « Tant que », la condition étant évidemment `u!=1` (on ne s'arrête que si la condition est fausse, c'est-à-dire si $u = 1$).

```
def syracuse1(N):
    u,n=N,0
    while u!=1:
        n+=1
        if u%2==0 : u=u//2
        else : u=3*u+1
        print("u(",n,")=",u)

L=[10,15,18]
for N in L :
    print("u( 0 )=",N)
    syracuse1(N)
```

```
u( 0 )= 10
u( 1 )= 5
u( 2 )= 16
u( 3 )= 8
u( 4 )= 4
u( 5 )= 2
u( 6 )= 1
```

```
u( 0 )= 15
u( 1 )= 46
u( 2 )= 23
u( 3 )= 70
u( 4 )= 35
u( 5 )= 106
u( 6 )= 53
u( 7 )= 160
u( 8 )= 80
u( 9 )= 40
u( 10 )= 20
u( 11 )= 10
u( 12 )= 5
u( 13 )= 16
u( 14 )= 8
u( 15 )= 4
u( 16 )= 2
u( 17 )= 1
```

```
u( 0 )= 18
u( 1 )= 9
u( 2 )= 28
u( 3 )= 14
u( 4 )= 7
u( 5 )= 22
u( 6 )= 11
u( 7 )= 34
u( 8 )= 17
u( 9 )= 52
u( 10 )= 26
u( 11 )= 13
u( 12 )= 40
u( 13 )= 20
u( 14 )= 10
u( 15 )= 5
u( 16 )= 16
u( 17 )= 8
u( 18 )= 4
u( 19 )= 2
u( 20 )= 1
```

Les suites associées aux nombres $N = 15$ et $N = 18$ sont un peu plus longues à aboutir sur 1 (voir les résultats d'exécution). On remarque que la suite qui part de 18 passe par 10 et celle qui part de 15 aussi. Par contre, cette dernière ne passe pas par 18 : les deux suites sont distinctes jusqu'au terme 40 à partir duquel elles fusionnent. On pourrait ainsi dresser une cartographie précise de l'emplacement de chaque entier dans cette vaste dégringolade irrégulière vers l'unité. C'est ce qui a été fait pour les nombres qui aboutissent sur 1 en moins de 20 étapes, comme cet extrait de Wikipédia le montre ci-dessous (voir aussi l'animation sur ce thème [1] de Jason Davies).

3. L'altitude est la plus grande valeur de u_n qui a été atteinte, le maximum de u_n parmi toutes les valeurs atteintes par la suite. Pour la déterminer, on l'initialise à u_0 et ensuite, dans la boucle, on teste pour savoir si la nouvelle valeur calculée dépasse l'altitude, auquel cas on remplace l'altitude par cette nouvelle valeur. La longueur de la suite est la dernière valeur de n, celle qui conduit à l'égalité $u_n = 1$ pour la 1re fois.

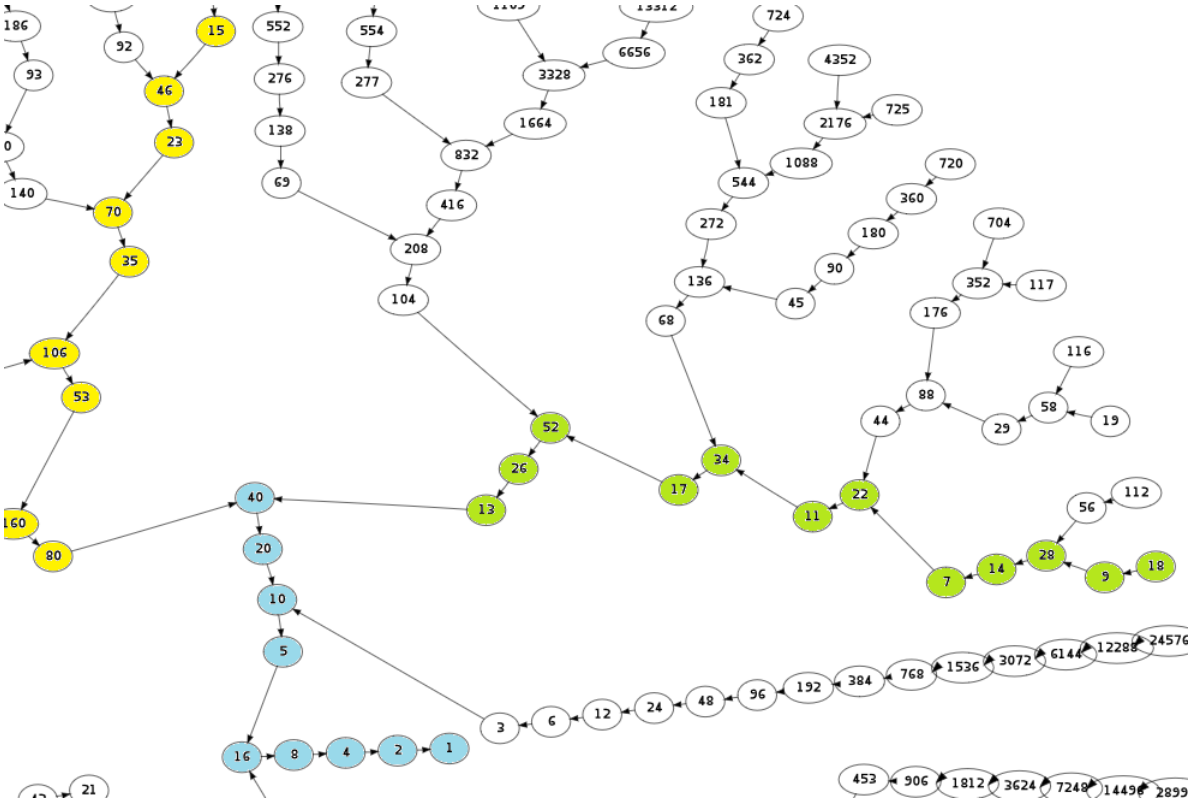
1. https://www.jasondavies.com/collatz-graph/

```
def syracuse2(N):
    u,n,a=N,0,N
    while u!=1:
        n+=1
        if u%2==0 : u=u//2
        else : u=3*u+1
        if u>a : a=u
    return a,n

L=[10,15,18,127]
for N in L :
    alt,lon=syracuse2(N)
    print("suite de ",N,": altitude=",alt,", longueur=",lon)
```

```
suite de  10 : altitude= 16 , longueur= 6
suite de  15 : altitude= 160 , longueur= 17
suite de  18 : altitude= 52 , longueur= 20
suite de  127 : altitude= 4372 , longueur= 46
```

Pour $N = 10$, on obtient bien *altitude* $= 16$, *longueur* $= 6$.
Pour $N = 15$, on obtient *altitude* $= 160$, *longueur* $= 17$; pour $N = 18$, on obtient *altitude* $= 52$, *longueur* $= 20$ et pour $N = 127$, on obtient *altitude* $= 4372$, *longueur* $= 46$.
Les suites les plus courtes sont celles qui partent d'une puissance de deux, puisque dans ce cas, à chaque étape le nombre est divisé par 2. Pour la même raison, ces suites n'ont jamais une altitude supérieure à leur point de départ. Pour $N = 871$, on obtient *altitude* $= 190996$, *longueur* $= 178$ (c'est le maximum pour les nombres inférieurs à 1161) ; par contre pour $N = 1024$, on obtient *altitude* $= 1024$, *longueur* $= 10$ (c'est le minimum pour les nombres inférieurs à 2048).



Correction de l'exercice 3 (Un exercice du poly « suites »)
1. Explicitons les premiers termes de (d_n) en effectuant la division à la calculatrice

$$\frac{2018}{19} \approx 106,21052631579$$

L'affichage ne permet pas de distinguer une répétition et pourtant, ce nombre étant rationnel, on sait qu'une séquence de chiffres doit se répéter. En effet, la suite des restes ne peut prendre que 18 valeurs différentes. Elle est donc nécessairement périodique, d'où également la suite des chiffres du quotient.
Pour obtenir la suite exacte des chiffres de la séquence périodique, effectuons la division à la main jusqu'à l'obtention d'un même reste deux fois :

$$
\begin{array}{rl|l}
2\,0\,1\,8 & & 19 \\
\cline{3-3}
2\,0\,1\,4 & & 106,210526315789473684 \\
\cline{1-1}
\quad 4\,0 & & \\
\quad\;\; 2\,0 & & \\
\quad\;\;\; 1\,0\,0 & & \\
\quad\quad\; 5\,0 & & \\
\quad\quad\; 1\,2\,0 & & \\
\quad\quad\quad 6\,0 & & \\
\quad\quad\quad\; 3\,0 & & \\
\quad\quad\quad\; 1\,1\,0 & & \\
\quad\quad\quad\quad 1\,5\,0 & & \\
\quad\quad\quad\quad\; 1\,7\,0 & & \\
\quad\quad\quad\quad\quad 1\,8\,0 & & \\
\quad\quad\quad\quad\quad\quad 9\,0 & & \\
\quad\quad\quad\quad\quad\quad 1\,4\,0 & & \\
\quad\quad\quad\quad\quad\quad\quad 7\,0 & & \\
\quad\quad\quad\quad\quad\quad\quad 1\,3\,0 & & \\
\quad\quad\quad\quad\quad\quad\quad\quad 1\,6\,0 & & \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad 8\,0 & & \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 4 & &
\end{array}
$$

Ainsi, comme 4 est un reste qui a déjà été obtenu, la suite va se répéter avec un 2, puis 1, etc.
On a $\frac{2018}{19} = 106,210526315789473684\cdots$ la suite de 18 chiffres 210526315789473684 se répétant jusqu'à l'infini.
La suite (d_n) est périodique de période 18, c'est-à-dire que pour tout $n > 0$ on a $d_{n+18} = d_n$.
La définition explicite de d_n dépend donc du reste r_n de la division par 18 de n.

r_n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
d_n	4	2	1	0	5	2	6	3	1	5	7	8	9	4	7	3	6	8

Comme $2019 = 112 \times 18 + 3$, on a $d_{2019} = d_3 = 0$.

Si d_n est le n$^{\text{e}}$ chiffre de la partie décimale de $\frac{2018}{2019} \approx 0.9995047052997$, on risque de peiner pour trouver à la main le développement décimal exact de $\frac{2018}{2019}$.
Pour cette raison, on va écrire un programme adapté.

```
a=int(input("Entrer le dividende : "))
b=int(input("Entrer le diviseur  : "))
if a//b>0 : ecriture=str(a//b)       #la partie entière du quotient      .
else : ecriture="0"
if a%b==0: print("Ce nombre est un entier : "+ecriture)
else :
    reste,longueur,suiteRestes,fin=a%b,0,list(),0
    ecriture+=",";
    suiteRestes.append(reste)
    while fin==0:
        reste*=10
        chiffreQuotient=reste//b
        suiteRestes.append(reste%b)
        ecriture+=str(chiffreQuotient)
        if suiteRestes[-1]==0 : fin=1
        else:
            for i in range(len(suiteRestes)-1) :
                if suiteRestes[-1]==suiteRestes[i] :
                    fin=2
                    longueur=len(suiteRestes)-i-1
                    sequence=str(ecriture[-longueur:])
                    ecriture+="..."
                else : reste=suiteRestes[-1]
    print("Voici le développement décimal de ce quotient : "+ecriture)
    if fin==2 :
        print("L'écriture décimale a une période de "+str(longueur))
        print("La séquence qui se répète est "+sequence)
    else : print("Ce nombre est décimal : "+ecriture)
```

```
Entrer le dividende : 2018
Entrer le diviseur  : 19
Voici le développement décimal de ce quotient : 106,210526315789473684...
L'écriture décimale a une période de 18
La séquence qui se répète est 210526315789473684
```

Ce programme nous permet de vérifier nos calculs précédents (voir la sortie d'exécution en bleu). Il nous donne ensuite la séquence des chiffres se répétant dans le quotient $\frac{2018}{2019}$:

$$\begin{aligned}&999504705299653293709757305596830113917781079742446755 81\\ &971272907379891035165923724616146607231302625061911837 54\\ &333828628033680039623576027736503219415552253590886577 51\\ &362060425953442298167409608717186726102030708271421495 78\end{aligned}$$

Cette suite de chiffres en contient 224, ainsi la suite (d_n) associé à ce quotient est périodique de période 224. La définition explicite de d_n dépend donc du reste r_n de la division par 224 de n.
On a $d_{n+224} = d_n$, par exemple $d_{225} = d_1 = 9$, $d_{226} = d_2 = 9$, $d_{227} = d_3 = 9$ et $d_{228} = d_4 = 5$.
Comme $2019 = 3 \times 224 + 3$, on a $d_{2019} = d_3 = 9$.

Pour simplifier ce programme, et notamment pour supprimer l'emploi de la liste `suiteRestes`, voici un programme que l'on peut utiliser pour obtenir la séquence périodique d'un nombre rationnel non-décimal. Ce programme a été mis au point collectivement lors de la séquence du vendredi 18 janvier. On teste simplement le retour du premier reste (celui de la division euclidienne initiale). Dès qu'on obtient ce reste, on est certain que les chiffres du quotient vont se présenter dans le même ordre. On enregistre la séquence périodique dans une variable `t` qui est une chaîne de caractères (une « String » en Python) pour éviter la perte des zéros éventuels (par exemple $1 \div 11 = 0,090909\cdots$ et dans ce cas la chaîne `t` sera `09`, si on conserve le nombre 09, il sera affiché `09`).

```
a=int(input("a="))
b=int(input("b="))
q=a//b     # quotient entier
r=a%b      # premier reste
s=r        # premier reste mobile
p=1        # période initialisée à 1
t=str((s*10)//b)    # chiffre 1 de la séquence periodique du quotient
while (s*10)%b!=r: # tant que le reste n'est pas égal au premier reste
    s=(s*10)%b     # nouveau reste mobile
    p+=1           # incrément de la période
    t+=str((s*10)//b) # ajout d'un chiffre dans la séquence periodique
print("Période  :",p)
print("Séquence :",t)
print("Quotient :"+str(q)+","+t+t+"...")
```

```
a=1                          a=2018
b=7                          b=19
Période  : 6                 Période  : 18
Séquence : 142857            Séquence : 210526315789473684
Quotient :0,142857142857...  Quotient :106,21052631578947368421052631578947368 4...
```

Ce programme est écrit en Python. Comment fait-on avec une Casio ou une TI ? Il faut retrouver le codage des différentes fonctions utilisées :

- La fonction `str` convertit un nombre en chaîne de caractères. Dans certains langages, on peut remplacer `str(q)` par `""+q`.
- La fonction `int` convertit une chaîne de caractères en un nombre entier. Par exemple `int("12")` donnera le nombre 12. Cette fonction donne aussi, en Python, la partie entière d'un nombre décimal. Par exemple `int(12.5)` donne 12 également. Dans le programme ci-dessus j'ai employé la syntaxe `int(input())` car la fonction `input()` renvoit une chaîne de caractères, même si on entre un nombre.
- L'opération `a//b` donne le quotient entier de `a` par `b`. On peut réaliser cela en faisant `int(a/b)` (on prend la partie entière du quotient décimal).
- L'opération `a%b` donne le reste de la division euclidienne de `a` par `b`. Cette fonction est souvent codée `mod(a,b)` (mod pour abréger modulo) ou aussi `rmdr(a,b)` (rmdr pour abréger reminder, le reste). Cette opération est très importante en programmation ; elle permet notamment de tester si un nombre `a` est pair : `if a%2==0`.
- je rappelle qu'en Python un test d'égalité se code avec deux signes `=`. Dans les autres langages, ce n'est généralement pas le cas. Le test d'inégalité se code, quant à lui, en Python `!=`.

Vous serez satisfaits de ce programme simplifié tant que vous l'essayez sur des nombres rationnels comme $\frac{1}{3}$, $\frac{1}{7}$, $\frac{1}{17}$ ou $\frac{2018}{19}$. Mais il vous montrera ses limites si vous l'essayer sur un nombre décimal (le reste devient nul) comme $\frac{1}{8}$ ou bien sur un nombre dont la séquence périodique ne commence pas directement après la virgule, comme $0,122\cdots$ dont l'écriture rationnelle est $\frac{11}{90}$. Pour $\frac{11}{9}$ il fonctionne et donne $1,22\cdots$ mais pour $\frac{11}{90}$, alors que c'est juste une question de virgule, il n'aboutit pas. Pourquoi ?
En se limitant à tester simplement le retour du premier reste, on élimine le cas où le reste est finalement nul (nombre décimal) et aussi le cas où la répétition des restes ne concerne pas le premier reste. C'est tout l'intérêt de la liste des restes : collectionner tous les restes jusqu'au retour d'un reste déjà obtenu. On examine si le reste figure déjà dans la liste : si oui, c'est fini, sinon on met le dernier reste dans la liste et on continue.
Le premier cas est facile à traiter. Voici une solution : je teste juste si le reste est nul à un moment donné, dans ce cas la division s'arrête et il y aura des zéros au quotient.

```
a=int(input("a="))
b=int(input("b="))
q=a//b     # quotient entier
r=a%b      # reste 1
s=r        # premier reste mobile
p=1        # période initialisée à 1
t=str((s*10)//b)    # chiffre 1 de la séquence periodique du quotient
while (s*10)%b!=r and s!=0: # tant que reste n'est ni reste 1 ni zéro
    s=(s*10)%b      # nouveau reste mobile
    p+=1            # incrément de la période
    t+=str((s*10)//b) # ajout d'un chiffre dans la séquence periodique
if s==0:            # si le nombre est décimal
    print("Quotient :"+str(q)+","+t)
else:               # si la séquence commence après la virgule...
    print("Période  :",p)
    print("Séquence :",t)
    print("Quotient :"+str(q)+","+t+t+t+"...")
```

```
a=1
b=8
Quotient :0,1250

a=12
b=4
Quotient :3,0

a=11
b=9
Période  : 1
Séquence : 2
Quotient :1,22...
```

Le dernier cas à traiter concerne les divisions dont le reste qui revient n'est pas le premier reste obtenu. Pour traiter cela sans liste, on peut penser à essayer de tester si ce n'est pas le second reste qui revient, et si ce n'est pas le second c'est peut-être le troisième... Finalement on peut faire une grande boucle « tant que » qui s'arrête quand on a (enfin) trouvé un reste qui revient. La boucle intérieure doit être arrêtée lorsqu'on a épuisé toutes les possibilités de restes (sinon on n'en sortira jamais) donc il faut l'arrêter quand elle a été utilisée b fois (b est le diviseur, on ne peut pas avoir plus de restes que le diviseur ; en divisant par 90 par exemple, il ne peut y avoir que 89 reste non nuls différents).
Voici une version plus élaborée de notre programme initial qui réalise cela (sans liste). Peut-on l'améliorer encore ? le simplifier ? Sans doute, je vous laisse y réfléchir...

```
a=int(input("a="))
b=int(input("b="))
f=0       # indicateur de fin (fin=1 : on s'arrête)
n=0       # on teste si le reste qui revient est le nième
while f==0:
    r=(a*10**n)%b         # reste 1
    q=(a*10**n)//b/10**n  # partie fixe du quotient
    s=r       # premier reste mobile
    p=1       # période initialisée à 1
    t=str((s*10)//b)    # chiffre 1 de la séquence periodique du quotient
    while (s*10)%b!=r and s!=0 and p<b: # tant que reste n'est ni reste 1 ni zéro
        s=(s*10)%b      # nouveau reste mobile
        p+=1            # incrément de la période
        t+=str((s*10)//b) # ajout d'un chiffre dans la séquence periodique
    if p<b: f=1    # on a trouvé la fin de la séquence
    else  : n+=1   # on augmente le rang du reste 1
if s==0:           # si le nombre est décimal
    print("Nombre décimal")
    print("Quotient :"+str(int(q))+","+t)
else:
    print("Période  :",p)
    print("Séquence :",t)
    if n==0 : print("Quotient :"+str(int(q))+","+t+t+t+"...") # la séquence commence après la virgule...
    else    : print("Quotient :"+str(q).replace(".",",")+t+t+t+"...") # la séquence commence plus loin
```

```
a=2018
b=19
Période  : 18
Séquence : 210526315789473684
Quotient :106,210526315789473684210526315789473684...
```

```
a=11
b=90
Période  : 1
Séquence : 2
Quotient :0,122...
```

```
a=1
b=4096
Nombre décimal
Quotient :0,0002441406250
```