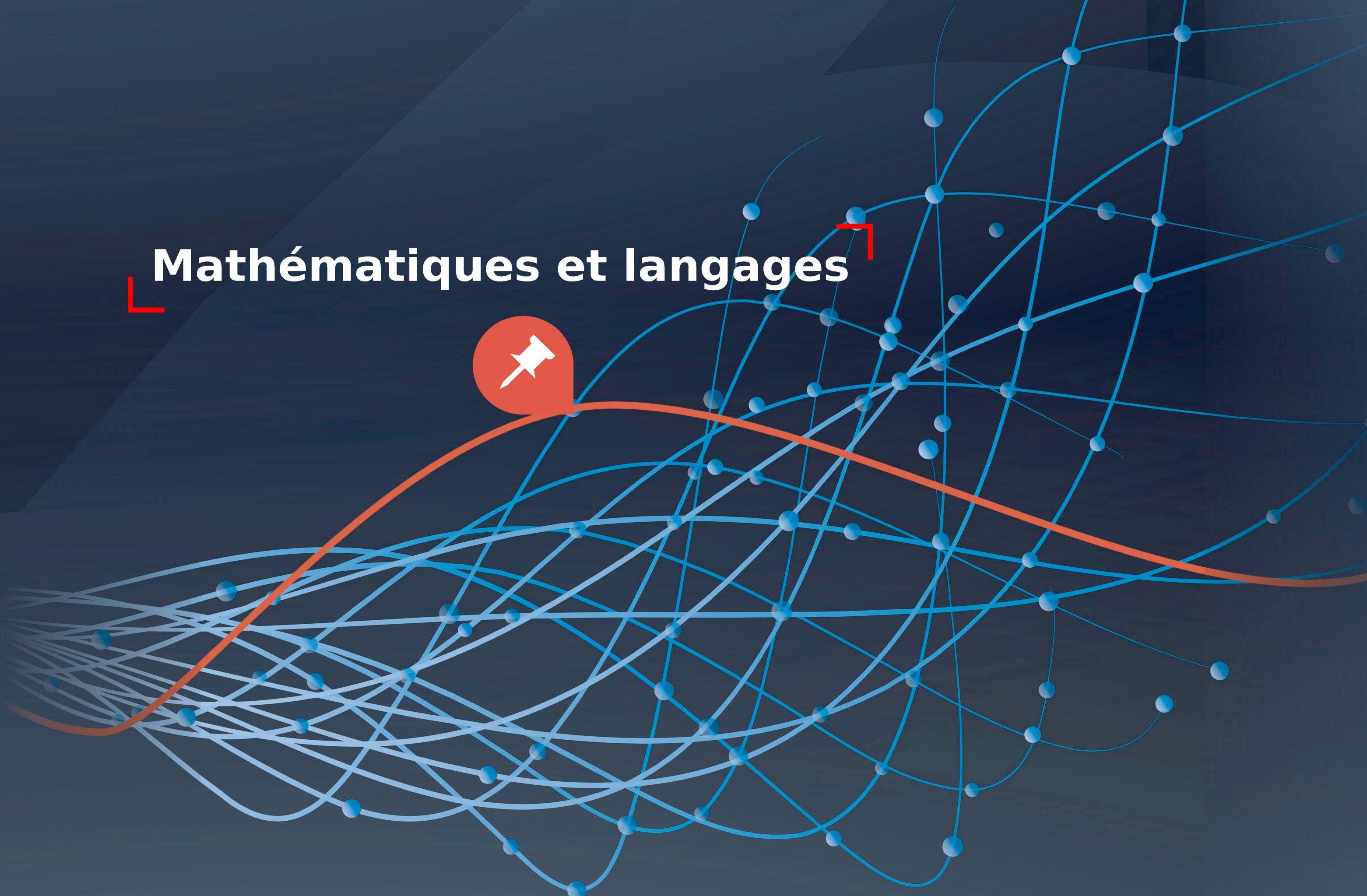
Le bulletin de l'APMEP - N° 528

AU FIL DES MATHS

de la maternelle à l'université. . .

Édition Avril, Mai, Juin 2018

Mathématiques et langages



APMEP

Association des Professeurs de Mathématiques de l'Enseignement Public

ASSOCIATION
DES PROFESSEURS DE MATHÉMATIQUES
DE L'ENSEIGNEMENT PUBLIC

26 rue Duméril, 75013 Paris
Tél. : 01 43 31 34 05 - Fax : 01 42 17 08 77
Courriel : secretariat-apmep@orange.fr - Site : https://www.apmep.fr
Présidente d'honneur : Christiane ZEHREN

Au fil des maths, c'est aussi une revue numérique augmentée :
https://afdm.apmep.fr
version réservée aux adhérents. Pour y accéder connectez-vous à votre compte *via* l'onglet *Au fil des maths* (page d'accueil du site) ou *via* le QRcode, ou suivez les logos.

Si vous désirez rejoindre l'équipe d'*Au fil des maths* ou bien proposer un article, écrivez à aufildesmaths@apmep.fr

Annonceurs : pour toute demande de publicité, contactez Valérie LAROSE vali.larose@gmail.com

AU FIL DES MATHS

ÉQUIPE DE RÉDACTION

Directeur de publication : *Alice ERNOULT*.

Responsable coordinateur de l'équipe : *Lise MALRIEU*.

Rédacteurs : *Marie-Astrid BÉZARD, Richard CABASSUT, Séverine CHASSAGNE-LAMBERT, Mireille GÉNIN, Cécile KERBOUL, Valérie LAROSE, Lise MALRIEU, Jean-Marie MARTIN, Pierre MONMARCHÉ, Vincent PANTALONI, Henry PLANE, Daniel VAGOST*.

« Fils rouges » numériques : *Paul ATLAN, Laure ÉTÉVEZ, Marianne FABRE, Adrien GUINEMER, Simon LE GAL, Julien MARCEAU, Harmia SOIHILI*.

Illustrateurs : *Pol LE GALL, Olivier LONGUET, Jean-Sébastien MASSET*.

Équipe TEXnique : *François COUTURIER, Isabelle FLAVIER, Anne HÉAM, François PÉTIARD, Olivier REBOUX, Guillaume SEGUIN, Sébastien SOUCAZE, Michel SUQUET*.

Relations avec le Bureau national : *Catherine CHABRIER*.

Votre adhésion à l'APMEP vous abonne automatiquement à *Au fil des maths*.

Pour les établissements, le prix de l'abonnement est de 60 € par an.

La revue peut être achetée au numéro au prix de 15 € sur la boutique en ligne de l'APMEP.

Mise en page : Olivier REBOUX
Dépôt légal : Juin 2018
Impression : Imprimerie Horizon P.A. de la plaine de Jouques 200 avenue de Coulin
13420 GEMENOS
ISSN : 2608-9297

De surprenantes arithmétiques (I)

André-Jean Glière a animé l'atelier « SurpreNantes arithmétiques » lors des journées de l'APMEP à Nantes (octobre 2017). Dans ce premier article, il évoque les nombres palindromes, les nombres de Lychrel et l'algorithme de Kaprekar.

André-Jean Glière

J'ai toujours pensé que la théorie des nombres était une science expérimentale et d'ailleurs, avant l'existence des ordinateurs, Gauss, Ramanujan et bien d'autres la considérèrent ainsi.

Richard Kenneth Guy [1]

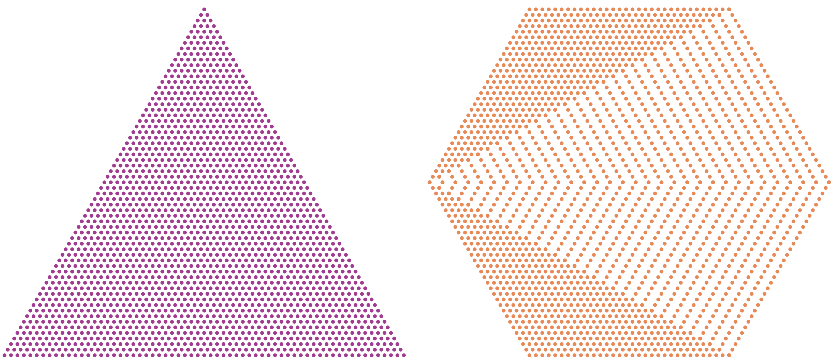
Introduction

L'origine de ce travail vient du sacro-saint rituel des bons vœux adressés le jour de la rentrée de janvier aux étudiants de la classe de MPSI de l'ÉSÉO [2] à laquelle j'enseigne les mathématiques et l'informatique. Depuis quelques années, j'ai pris l'habitude, avant de les traumatiser à vie avec la formule de Taylor avec reste intégral et de les mettre KO avec la démonstration du théorème de Bolzano-Weierstrass, de les échauffer pendant quelques minutes en manipulant les nombres entiers. La nouvelle année s'est révélée être chaque fois un bon prétexte. Par exemple, 2 016 a été l'occasion de jouer avec les nombres triangulaires et les nombres hexagonaux. Ce dernier est en effet le 63ᵉ nombre triangulaire :

$$2\,016 = 1 + 2 + 3 + \cdots + 63 = \frac{63 \times 64}{2}$$

et est également le 32ᵉ nombre hexagonal :

$$2\,016 = (2 \times 32 - 1) \times 32$$



2 016 est le 63ᵉ nombre triangulaire et le 32ᵉ nombre hexagonal.

1. Richard Kenneth Guy, né le 30 septembre 1916 à Nuneaton dans le Warwickshire, est un mathématicien britannique, professeur émérite de mathématiques à l'université de Calgary. Il est connu principalement pour son livre *Unsolved Problems in Number Theory* et pour avoir co-écrit *Winning Ways for your Mathematical Plays*. Il a également publié plus d'une centaine d'articles sur la théorie des jeux combinatoires, la théorie des nombres et la théorie des graphes. Guy est également une figure notable des études d'échecs.
2. École Supérieure d'Électronique de l'Ouest, située à Angers.

Quant à 2 017, outre le fait qu'il soit le 306e nombre premier (ce qui n'est pas rien et qui m'a donné prétexte à exposer les résultats d'Hadamard et de La Vallée Poussin, rien que ça !), il m'a fourni un palindrome sympathique dès le premier calcul :

$$2\,017 + 7\,102 = 9\,119.$$

Par suite, en consultant la littérature sur le sujet, en particulier les livres passionnants de Jean-Paul Delahaye et l'incontournable *Dictionnaire de (presque) tous les nombres entiers* de Daniel Lignon, et les pages de certains sites dédiés comme celles de Sayrac[3], j'ai découvert un bestiaire fabuleux. Des nombres de Harshad, divisibles par la somme de leurs chiffres, aux nombres de Carmichael, pseudo-premiers de Fermat, en passant par les nombres-pizzas dans le plan ou les nombres-cakes dans l'espace, et les nombres de Fibonacci, de Tribonacci, de Lucas ou de Padovan, la liste est inépuisable et on en a pour tous les goûts et tous les appétits.

Il a fallu donc choisir de parler de quelques-uns d'entre eux. Et puisque c'est une mathématique ludique dont il s'agit, il y a quelque temps on aurait dit récréative, adressée à des collégiens, à des lycéens et éventuellement à des étudiants qui veulent reposer leurs neurones trop sollicités par les définitions epsilonesques de Weierstrass (encore lui !), j'ai proposé durant l'atelier quatre processus itératifs simples (mais pas simplistes), quatre algorithmes que l'on peut aisément coder en langage Python, le langage informatique officiel de l'enseignement en classes préparatoires.

Cet article traite, entre autres, des nombres palindromes, des nombres de Lychrel, de la constante de Kaprekar et des nombres heureux. Je ne présente ici que trois algorithmes. Dans un second article, j'en exposerai un quatrième.

1. Nombres palindromes

Un nombre palindrome est un nombre qui peut se lire indifféremment de gauche à droite ou de droite à gauche. Il possède donc une « symétrie ».

Exemples :

a) Tous les nombres à un chiffre en base dix : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Il fallait les mentionner, mais cela n'est pas passionnant.

b) Les neuf nombres 11, 22, 33, 44, 55, 66, 77, 88, 99. C'est mieux, mais encore ?

c) Il existe quatre-vingt-dix palindromes à trois chiffres. Pourquoi ? Parce que... Parce que c'est évident : bien sûr, pour conserver la symétrie, il suffit d'intercaler entre les deux chiffres identiques des neuf palindromes précédents les dix chiffres précédemment cités : 101, 111, 121, ..., 191, 202, 212, 222, ..., 303, 313, 323, ..., 909, 919, 929, ..., 999.

d) De même, symétrie oblige, il existe exactement quatre-vingt-dix palindromes à quatre chiffres. Bizarre, non ? Pas du tout. Il suffit d'intercaler entre les deux chiffres identiques des neuf palindromes à deux chiffres identiques deux chiffres identiques. C'est du Raymond Devos ! 1 001, 1 111, 1 221, 1 331, 1 441, 1 551, 1 661, 1 771, 1 881, 1 991, ..., 9 009, 9 119, 9 229, ..., 9 999.

e) Dans le défi mathématique nº 1 du journal *Le Monde*, Cedric Villani demandait le nombre de palindromes à trois-cent-cinquante-et-un chiffres[4].

Programme Python nº 1

Il se compose de deux fonctions et d'une procédure.

La première fonction prend en paramètre une chaîne de caractères *chaine*, autrement dit un mot, et retourne la chaîne de caractères renversée.

Fonction renverse

```
def renverse(chaine):
 resultat = ''
 for x in range(len(chaine)):
 resultat += chaine[len(chaine) - 1 - x]
 return resultat
```

La seconde fonction prend en paramètre un nombre *nombre*, le transforme en chaîne de caractères, autrement dit en un mot et retourne un booléen : `Vrai` pour un palindrome, Faux sinon.

Fonction palindrome

```
def palindrome(nombre):
 chaine = str(nombre)
 return chaine == renverse(chaine)
```

La procédure prend en paramètre un entier maximum *nMax* et affiche chaque palindrome inférieur *nMax* ainsi que le nombre de palindromes inférieurs à *nMax*.

3. http://sayrac.rlc.free.fr
4. Défi nº 1 : . La réponse : .

Procédure ListeNomPalin

```
def ListeNomPalin(nMax):
 print("Liste des palindromes de 2 à {}"
 .format(nMax))
 nb=0
 for k in range(nMax+1):
 if(palindrome(k) == True):
 print(k,end = "; ")
 nb = nb+1
 print("Il y a ", nb,
 " palindromes inférieurs à ",nMax)
```

2. Algorithme de Lychrel

L'algorithme est le suivant :

> Pour un nombre entier naturel n, on calcule $L(n) = n + \tilde{n}$ où $\tilde{n}$ est le nombre dont les chiffres sont écrits dans l'ordre inverse de n. Puis on réitère cette opération jusqu'à obtenir un palindrome.

Depuis quelques années, avec mes étudiants, je teste la nouvelle année. Voyez-vous-même :

$$L(2\,018) = 2\,018 + 8\,102 = 10\,120$$
$$\text{et } L(10\,120) = 10\,120 + 2\,101 = 12\,221$$

On obtient là un palindrome en **deux** itérations. On est souvent encore plus chanceux :

$$\begin{aligned} L(2\,017) &= 2\,017 + 7\,102 = 9\,119, \\ L(2\,016) &= 2\,016 + 6\,102 = 8\,118, \\ L(2\,015) &= 2\,015 + 5\,102 = 7\,117, \\ L(2\,014) &= 2\,014 + 4\,102 = 6\,116, \\ &\text{etc...} \end{aligned}$$

on obtient un palindrome en **une seule** itération en remontant jusqu'en l'an 2000, sauf pour l'année 2002 qui en est déjà un (la chanceuse) et l'année 2008 qui nécessite elle aussi **deux** itérations (la pauvre) : $L(2\,008) = 2\,008 + 8\,002 = 10\,010$ et $L(10\,010) = 10\,010 + 1\,001 = 11\,011$.

Si vous remontez encore, l'année 1999 est plus coriace, car il ne faut pas moins de **sept** itérations pour obtenir le palindrome 712 217. Vérifiez-le.

En essayant un grand nombre de valeurs, on s'aperçoit qu'en général on obtient un palindrome en seulement quelques étapes. Je dis bien en général car bien sûr certains se font remarquer ! Pour les nombres inférieurs ou égaux à 100, on a besoin au maximum de **six** itérations sauf pour 89 et donc évidemment aussi pour 98. En effet, il leur faut **vingt-quatre** itérations pour atteindre un palindrome ! Vous pourrez le vérifier avec le code Python ci-dessous.

Programme Python n° 2

Il s'agit d'une procédure prenant en paramètre un nombre `nombre`, utilisant une boucle *tant que* et les deux fonctions précitées `renverse` et `palindrome`, et affichant le palindrome trouvé et le nombre d'itérations nécessaires.

Procédure testDeLychrel

```
def testDeLychrel(nombre, maxIterations):
 iterations=0
 while(iterations < maxIterations):
 if(palindrome(nombre)):
 print("On a trouvé un palindrome
 %d en %d itérations"
 %(nombre,iterations))
 return
 print(nombre," + ",
 int(renverse(str(nombre)))," = ",
 nombre + int(renverse(str(nombre))))
 nombre = nombre + int(renverse(str(nombre)))
 iterations += 1
 print("raté")
```

Notez que l'on se réserve le droit de ne pas faire tourner la machine trop longtemps en bornant le nombre d'itérations par $maxIterations$ et en se donnant la possibilité d'afficher « raté » au cas où. Pourquoi cette précaution ? Tout simplement parce que certains nombres échappent au test de Lychrel. Je veux dire que, malgré un très grand nombre d'itérations, mais alors un très très très grand nombre d'itérations, on est dans l'impossibilité d'obtenir un palindrome avec cet algorithme.

> De tels nombres sont appelés des nombres de Lychrel [5].

Le plus petit nombre de Lychrel potentiel connu à ce jour est 196. Il a atteint un nombre à un million de chiffres après... 2 415 836 itérations sans parvenir à un palindrome ! Il existe treize nombres inférieurs à 1 000, candidats à l'appellation nombre de Lychrel, c'est-à-dire pour lesquels on n'a pas trouvé pour le moment de palindrome :

196, 295, 394, 493, 592, 689, 691, 788, 790, 879, 887, 978, 986.

Il existerait deux-cent-quarante-six nombres de Lychrel suspectés inférieurs à 10 000.

Mais on doit rester très prudent. En 2005, Jason Doucette a en effet produit un palindrome à partir du nombre 1 186 060 307 891 929 990 en **deux-cent-soixante-et-une** itérations ! Vous pouvez essayer avec la procédure précédente. En quinze secondes, vous verrez défiler des dizaines de milliers de chiffres et **deux-cent-soixante-et-une** itérations. C'est beau et c'est indolore !

5. Le nom « Lychrel » a été donné à un tel nombre par Wade VanLandingham en l'honneur de sa fiancée, Cheryl. Jusqu'où va l'amour !

3. Algorithme de Kaprekar

Voici un deuxième algorithme, tout aussi séduisant que le précédent, en tout cas tout aussi simple. Car l'intérêt de ces mathématiques est leur simplicité. Elles sont compréhensibles par tout un chacun.

Il s'agit de l'algorithme de Kaprekar[6].

L'algorithme est le suivant :

Dattatreya Ramachandra KAPREKAR *(1905-1986).*

Soit un naturel n, on calcule $K(n) = M - m$ où m est le nombre dont les chiffres sont ceux de n écrits dans l'ordre croissant et M est le nombre dont les chiffres sont ceux de n écrits dans l'ordre décroissant. Puis on réitère le calcul $K(n)$, on construit ainsi la suite des itérés : $K(n)$, $K(K(n))$, $K(K(K(n)))$, ... jusqu'à obtenir... au fait quoi ?

Pour commencer, essayons :

- **des nombres à un chiffre** :
$$K(8) = 8 - 8 = 0.$$
- **des nombres à deux chiffres** :
$$K(54) = 54 - 45 = 9,$$
$$K(9) = 9 - 9 = 0.$$
$$K(75) = 75 - 57 = 18,$$
$$K(18) = 81 - 18 = 63,$$
$$K(63) = 63 - 36 = 27,$$
$$K(27) = 72 - 27 = 45,$$
$$K(45) = \cdots = 0.$$
À vous : calculez $K(13)$.
- **des nombres à trois chiffres** :
$$K(121) = 211 - 112 = 99,$$
$$K(99) = 99 - 99 = 0.$$
$$K(149) = 941 - 149 = 792,$$
$$K(792) = 972 - 279 = 693,$$
$$K(693) = 963 - 369 = 594,$$
$$K(594) = 954 - 459 = 495,$$
$$\boldsymbol{K}(\mathbf{495}) = 954 - 459 = \mathbf{495}.$$
À vous : calculez $K(231)$, $K(527)$, $K(619)$, $K(879)$, $K(999)$.
- **des nombres à quatre chiffres** :
$$K(1\,256) = 6\,521 - 1\,256 = 5\,265,$$
$$K(5\,265) = 6\,552 - 2\,556 = 3\,996,$$
$$K(3\,996) = 9\,963 - 3\,699 = 6\,264,$$
$$K(6\,264) = 6\,642 - 2\,466 = 4\,176,$$
$$K(4\,176) = 7\,641 - 1\,467 = 6\,174,$$
$$\boldsymbol{K}(\mathbf{6\,174}) = 7\,641 - 1\,467 = \mathbf{6\,174}.$$
$$K(3\,245) = 5\,432 - 2\,345 = 3\,087,$$
$$K(3\,087) = 8\,730 - 378 = 8\,352,$$
$$K(8\,352) = 8\,532 - 2\,358 = \mathbf{6\,174}.$$
$$K(4\,444) = 0.$$
À vous : calculez $K(7\,744)$, $K(7\,895)$.

Il est évident que, pour un nombre à un chiffre, l'algorithme mène à 0. C'est ce que l'on appelle **un cas dégénéré**.
Pour un nombre à deux chiffres, il semble en aller de même. Il est facile de le prouver. Considérons un entier naturel n dont les deux chiffres en base dix sont a et b. Comme $K(\overline{ab}) = K(\overline{ba})$, on peut supposer que $a \geqslant b$; il vient alors

$$K(n) = \overline{ab} - \overline{ba} = 10a + b - 10b - a = 9(a - b).$$

$K(n)$ est donc un multiple de 9. Par conséquent, il suffit de connaître l'aboutissement de l'algorithme de Kaprekar pour tous les multiples de 9. Nous avons déjà rencontré dans les exemples les nombres 9, 18, 27, 36, 45 et 99 : ils aboutissent à 0.

Les nombres « miroirs » 54, 63, 72 et 81 ont la même issue 0 (deux nombres miroirs ont la même image). Enfin, $K(90) = 81$, et il aboutit donc lui aussi à 0. CQFD.

Si les nombres à un ou à deux chiffres mènent tous à des cas dégénérés, il n'en va pas de même pour les nombres à trois et à quatre chiffres. Là, une surprise nous attend ! Au vu des quelques exemples donnés plus haut, nous pouvons conjecturer que, pour un nombre à trois chiffres, l'algorithme de Kaprekar aboutit à un cas dégénéré dans un petit nombre de cas et, dans tous les autres cas, au nombre 495, appelé **point fixe de la fonction K**.

Inutile de se priver du plaisir de faire de cette conjecture une propriété. La démonstration est à peine plus compliquée que la précédente.

6. Dattatreya Ramachandra Kaprekar (1905-1986) est un mathématicien indien passionné par les nombres. Il publie plusieurs livres qui restent peu connus du public. Martin Gardner, spécialiste des mathématiques récréatives, lui rend justice à partir de 1975. Outre la constante et les nombres qui portent son nom, il s'intéresse aux nombres Harshad, aux nombres de Demlo et aux auto-nombres.

On choisit un nombre n dont les trois chiffres en base dix sont pris a, b et c. On a bien compris que l'ordre des chiffres dans le nombre n ne change pas son image. Par exemple, $K(397) = K(973)$. On peut donc supposer que $a \geqslant b \geqslant c$. Il vient alors :

$$\begin{aligned} K(n) &= \overline{abc} - \overline{cba} \\ &= 100a + 10b + c - (100c + 10b + a) \\ &= 99(a - c). \end{aligned}$$

$K(n)$ est donc un multiple de 99. Par conséquent, comme tout à l'heure, il suffit de connaître l'aboutissement de l'algorithme de Kaprekar pour les nombres de la forme $99 \times k$ où k est un nombre entre 0 et 9, soit pour 891, 792, 693, 594, 495, 396, 297, 198, 99 et 0.

Les cinq premiers sont des images successives de l'algorithme : $K(891) = 792$, $K(792) = 693$, etc. Ils aboutissent tous à 495. Les trois suivants sont des miroirs des nombres que nous venons d'étudier. Ils aboutissent donc eux aussi à 495. Enfin, 99 et 0 sont dégénérés.

Point fixe de K pour les nombres à trois chiffres.

Conclusion : hormis les cas dégénérés, l'algorithme de Kaprekar se termine pour les nombres à trois chiffres par 495. On peut montrer qu'il le fait en au maximum **cinq itérations**.
Pour un nombre à quatre chiffres, on démontre de même que, hormis les cas dégénérés, l'algorithme de Kaprekar se termine par 6 174 en au maximum **sept itérations.** Le nombre 6 174 est appelée la **constante de Kaprekar.**

La constante de Kaprekar.

Résumons : les points fixes de K sont, pour un nombre à trois chiffres : 495 et, pour un nombre à quatre chiffres : 6 174.

Programme Python n° 3

Les deux premières fonctions permettent respectivement de transformer un nombre en la liste de ses chiffres et une liste de chiffres en un nombre en passant par l'intermédiaire d'une chaîne de caractères. Cela sera très utile pour pouvoir ordonner les chiffres du nombre du plus petit au plus grand et vice-versa (les listes sont faciles à ordonner). La première prend un nombre `nombre` en paramètre et retourne une liste `tab`, tandis que la deuxième prend une liste `liste` en paramètre et retourne un nombre `int(chaine)`.

Fonction nombreEnListe

```python
def nombreEnListe(nombre):
  chaine = str(nombre)
  tab = []
  for caract in chaine:
  tab = tab + [int(caract)]
  return tab
```

Fonction listeEnNombre

```python
def listeEnNombre(liste):
  chaine = ""
  for k in range(len(liste)):
  chaine = chaine+str(liste[k])
  return int(chaine)
```

La troisième fonction prend un nombre *nombre* en paramètre, appelle les deux fonctions précédentes et retourne $K(\textit{nombre}) = M - m$.

Fonction diffGrandPetit

```python
def diffGrandPetit(nombre):
  l = nombreEnListe(nombre)
  m = listeEnNombre(sorted(l))#ordre croissant
  M = listeEnNombre(sorted(l,reverse = True))
  #ordre décroissant
  print(M," − ",m," = ",M − m)
  #affichage de M − m
  return M − m
```

La procédure suivante prend en paramètre un nombre *nombre*, calcule $K(\textit{nombre})$ et réitère le calcul avec $\textit{nombre} = K(\textit{nombre})$ tant que *nombre* n'est pas un point fixe de la fonction K ou que *nombre* soit nul.

Procédure Kaprekar1234

```python
def Kaprekar1234(nombre):
  compteur = 0
  n1 = nombre
  n=diffGrandPetit(nombre)
  while(n != nombre):
  if(n == 0):
  break
  nombre = n
  n = diffGrandPetit(nombre)
  compteur = compteur+1
  if(n == 0):
  print("Cas dégénéré")
  else:
  print("Le nombre ", n1,
  " est associé à la constante ",
  "de Krapekar ",nombre, " en ",
  compteur, " itérations.")
```

Pour conclure et aller plus loin, on peut vérifier que :

- **Pour un nombre à cinq chiffres,** la fonction de Kaprekar n'a aucun point fixe, mais trois cycles, un de longueur 2 : (53 955, 59 994) et deux de longueur 4 : (61 974, 82 962, 75 933, 63 954) et (62 964, 71 973, 83 952, 74 943).
  Autrement dit, l'algorithme se termine soit sur un cas dégénéré, soit sur un cycle de longueur 2 ou un cycle de longueur 4.

- **Pour un nombre à six chiffres,** la fonction de Kaprekar a deux points fixes 549 945 et 631 764, et un cycle de longueur 7 :
  $(420\,876$, $851\,742$, $750\,843$, $840\,852$, $860\,832$, $862\,632$, $642\,654)$.
  Autrement dit, l'algorithme se termine soit sur un cas dégénéré, soit un point fixe, soit sur le cycle de longueur 7.

4. Nombres heureux, nombres malheureux

Nous arrivons maintenant à un troisième algorithme très abordable pour le néophyte. C'est par celui-ci que je commence en général la nouvelle année avec mes étudiants. Ils déchantent souvent parce que, depuis l'an 2000, les années se succèdent et se montrent désespérément malheureuses. Je m'explique :

L'algorithme est le suivant :

> Soit un entier naturel N écrit en base dix :
>
> $$N = \overline{a_n a_{n-1} \dots a_1 a_0}.$$
>
> On calcule la somme des carrés de ses chiffres : $S_2(N) = \sum_{k=0}^{n} a_k^2$.
> On réitère le calcul avec la somme obtenue, construisant ainsi la suite des itérés[7] : $S_2(N)$, $S_2(S_2(N))$, $S_2(S_2(S_2(N)))$, … jusqu'à ce qu'apparaisse un résultat remarquable. Et rassurez-vous, il apparaît rapidement.

Essayons 2 018, histoire de bien commencer l'année. La suite des itérés est : 2 018, 69, 117, 51, 26, 40, 16, 37, 58, 89, 145, 42, 20, 4, **16**. Stop ! Et alors ? Rien pour le moment. Si ce n'est qu'on tourne en rond.

Essayons encore 2 017 pour voir. La suite des itérés est cette fois : 2 017, 54, 41, 17, 50, 25, 29, 85, **89**. Et là, on a compris. On retourne au cycle vicieux précédent.

Essayons enfin 2 016 pour se souvenir :
$S_2(2\,016) = 41$ et on se souvient tout de suite que $S_2(2\,017) = 54$ et que $S_2(54) = 41$. Bingo !

Doit-on en conclure que l'algorithme aboutit toujours au cycle d'ordre 8 : $(4, 16, 37, 58, 89, 145, 42, 20)$? La conjecture est facile mais l'arithmétique est parfois difficile.

Reprenons depuis le début. Et le début pose tout de suite un sérieux problème.

$S_2(1)$ vaut 1 et reste 1 qu'on le veuille ou non.

$S_2(2)$ vaut 4 et le cycle d'ordre 8 est atteint.

La suite des itérées de 3 est : $(3, 9, 81, 65, 61)$. 16 ou 61, c'est du pareil au même : on rentre dans le cycle infernal précédent.

Doit-on calculer $S_2(4)$? Non, on connaît la réponse.

$S_2(5) = 25$, $S_2(25) = 29$, cela sent le déjà vu (voir 2 017).

$S_2(6) = 36$, $S_2(36) = 45$, 45 ou 54, du pareil au même (voir 2 017).

Le nombre 1 reste-t-il exceptionnel ? Continue-t-on jusqu'à 10 ? Courage…

Suite des itérés de 7 : $(7, 49, 97, 130, 10, 1)$. Gagné !

Sept.

Après ? Si on a envie d'expérimenter systématiquement, à la main, sans « *pythoner* », et avec un peu de temps, on peut suivre la vidéo à l'adresse : ▶.

Sinon, on passe. Dommage, car on y apprend qu'il existe vingt nombres inférieurs ou égaux à 100 qui aboutissent à 1 si on leur applique l'algorithme précédent, et que tous les autres aboutissent au cycle d'ordre 8.

Il est temps de définir deux types de nombres :

> **Définition**
>
> Soit un entier naturel N et soit la suite (u_n) définie de la manière suivante :
>
> $$\begin{cases} u_1 \text{ est la somme des carrés des chiffres de } N \\ \text{pour tout } n,\ u_{n+1} \text{ est la somme des carrés des chiffres de } u_n \end{cases}$$
>
> Le nombre N est dit **heureux** si, à partir d'un certain rang, la suite (u_n) est constante égale à 1 ; le nombre N est dit **malheureux** si, à partir d'un certain rang, u_n appartient à l'ensemble $\{4, 16, 37, 58, 89, 145, 42, 20\}$.

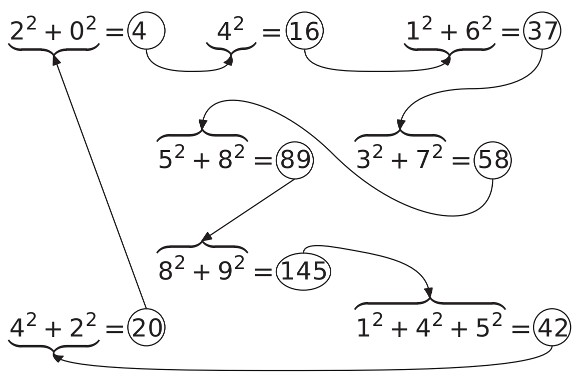
Un nombre peut-il être ni heureux, ni malheureux ? Autrement dit, peut-il être dans un état intermédiaire ? Eh bien… non, si l'on en croit la littérature, pardon, si l'on en croit les sites aux adresses suivantes : ▶ et ▶.

Conclusion : en itérant la fonction S_2, on démontre donc que l'on parvient :

- soit au point fixe 1 ;

7. Une telle suite, à savoir qu'un terme quelconque est la somme des carrés des chiffres du terme précédent, est appelée *joyeuse*.

- soit au cycle $(4, 16, 37, 58, 89, 145, 42, 20)$ d'ordre 8 :



Cycle des nombres malheureux.

Programme Python nº 4

La première fonction prend en paramètre un entier naturel n et retourne la somme des carrés de ses chiffres. On extrait les chiffres du nombre par une succession de divisions euclidiennes par 10.

Fonction somCarChif

```
def somCarChif(n):
 somme = (n%10)**2
 n = n//10
 while(n != 0):
 somme = somme + (n%10)**2
 n = n//10
 return somme
```

La deuxième fonction teste si une valeur appartient à une liste et retourne un booléen.

Fonction estPresent

```
def estPresent(valeur, liste):
 for k in range(len(liste)):
 if(liste[k] == valeur):
 return True
 return False
```

La procédure suivante affiche si un nombre n pris en paramètre est heureux ou malheureux et le nombre d'étapes pour le savoir. Elle fait appel aux deux fonctions précédentes.

Procédure NomHeureux

```
def NomHeureux(n):
 p = n
 s = somCarChif(p)
 print("La somme des carrés des chiffres",
 " de {} vaut {}".format(p,s))
 k=1
 l=[1,4,16,37,58,89,145,42,20]
 while(estPresent(s,l) == False):
 k = k+1
 p = s
 s = somCarChif(s)
 print("La somme des carrés des chiffres",
 " de {} vaut {}".format(p,s))
 if(s == 1):
 print("Le nombre {} est un nombre",
 " heureux d'ordre {}".format(n,k))
 else:
 print("Le nombre {} est un nombre ",
 "malheureux".format(n))
```

Pour pouvoir établir la liste des nombres heureux inférieurs à un entier $nMax$, on crée la procédure suivante qui emploie le même code que précédemment mais traduit en fonction. Elle retourne un booléen.

Fonction estNomHeureux

```
def estNomHeureux(n):
 resultat = True
 s = somCarChif(n)
 l = [1,4,16,37,58,89,145,42,20]
 while(estPresent(s,l) == False):
 s = somCarChif(s)
 if(s != 1):
 resultat = False
 return resultat
```

Procédure listenomHeureux

```
def listenomHeureux(nMax):
 print("Liste des nombres heureux de 1",
 " à {}".format(nMax))
 i = 0
 for k in range(1,nMax + 1):
 if(estNomHeureux(k) == True):
 i = i + 1
 print(k," est le nombre heureux n° ",i)
```

Utilisez ces scripts pour afficher par exemple les vingt nombres heureux inférieurs ou égaux à 100 : 1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100 ; ou bien montrez que $(1\,880, 1\,881, 1\,882)$ est un triplet de nombres heureux, ou bien que le *répunit*[8] 11 111 111 111 111 111 111 111 constitué de vingt-trois chiffres 1 est un nombre heureux d'ordre 4 ou enfin que le *nombre uniforme*[9] 99 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 constitué de cinquante chiffres 9 est un nombre malheureux !

8. Un *répunit* est un entier naturel dont l'écriture, dans une certaine base entière, ne comporte que des chiffres 1.
9. Un *nombre uniforme* (en anglais *repdigit* qui provient de *repeated digit*) est un entier naturel formé par la répétition d'un seul chiffre, le plus souvent dans le système de numération décimale.

Remarquez que la somme des carrés de ses chiffres n'est que 4 050 ce qui est somme toute raisonnable.

N'hésitez pas à chercher la prochaine année heureuse pour la souhaiter à vos élèves. Dépêchez-vous car elle est imminente !

Finalement, les nombres heureux sont nombreux[10].

Je suis heureux de vous apprendre qu'il en existe même **une infinité** : c'est une certitude. Quant à leur densité, elle serait d'environ 15 % : pour le moment, ce n'est qu'une conjecture.

Dans un second article, nous généraliserons cet algorithme à la somme des cubes des chiffres d'un nombre, puis aux sommes des puissances p des chiffres de ce nombre. Nous appellerons *agréables* ces transformations et nous verrons également les nombres *narcissiques*. Nous aborderons enfin les nombres *parfaits* et les nombres *amiables* qui ne sont que des cas particuliers de notions plus générales : *les suites aliquotes*.

Que du bonheur !

André-Jean Glière est professeur de mathématiques en classe préparatoire à l'ÉSÉO (École Supérieure d'Électronique de l'Ouest) à Angers.

10. Remarquons que tous les termes d'une suite joyeuse aboutissant à 1 sont heureux, par exemple 7, 49, 97, 130, 10, 1.

Sommaire du nº 528

Mathématiques et langages

APMEP

www.apmep.fr