Pascal Lafourcade, Guenaëlle De Julis et Malika More

15 ÉNIGMES LUDIQUES POUR SE PERFECTIONNER EN PROGRAMMATION PYTHON

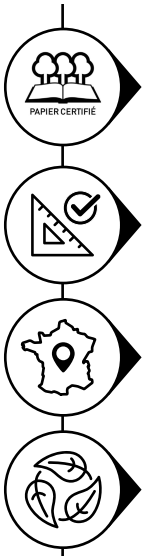
DUNOD

Découvrez aussi :

- P. Lafourcarde et M. More, *25 énigmes ludiques pour s'initier à la cryptographie*, Dunod, 2021.
- P. Lafourcarde et M. More, *15 énigmes ludiques pour s'initier à la programmation Python*, Dunod, 2022.
- J.-G. Dumas, P. Lafourcade, E. Roudeix, A. Tichit et S. Varrette, *Les NFT en 40 questions*, Dunod, 2022.
- J.-G. Dumas, P. Lafourcade, A. Tichit et S. Varrette, *Les blockchains en 50 questions*, 2e éd., Dunod, 2022.
- J.-G. Dumas, P. Lafourcade, P. Redon, *Architectures de sécurité pour Internet*, 2e éd., Dunod, 2020.
- J.-G. Dumas, J.-L. Roch, S. Varrette, E. Tannier, *Théorie des codes : Compression, cryptage, correction*, Dunod, 2018.
- D. Vergnaud, *Exercices et problèmes de cryptographie*, 4e éd. ? Dunod, 2023

Direction artistique : Nicolas Wiel

NOUS NOUS ENGAGEONS EN FAVEUR DE L'ENVIRONNEMENT :



Nos livres sont imprimés sur des papiers certifiés pour réduire notre impact sur l'environnement.

Le format de nos ouvrages est pensé afin d'optimiser l'utilisation du papier.

Depuis plus de 30 ans, nous imprimons 70 % de nos livres en France et 25 % en Europe et nous mettons tout en œuvre pour augmenter cet engagement auprès des imprimeurs français.

Nous limitons l'utilisation du plastique sur nos ouvrages (film sur les couvertures et les livres).

11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-085508-7

Sommaire

Avant-propos

Ces 15 énigmes ludiques vont vous faire découvrir des concepts importants de l'informatique. Elles nécessitent plus ou moins de logique, de réflexion et d'astuce, mais il est surtout indispensable de réaliser des programmes informatique pour les résoudre. Ce livre s'adresse à des lycéens scientifiques et étudiants en cursus informatique. L'objectif est de proposer des énigmes dont la solution peut être obtenue à l'aide d'un programme. Il fallait donc choisir un langage pour présenter les solutions, mais il est bien entendu possible de programmer dans votre langage de programmation favori pour résoudre ces énigmes. En début d'ouvrage, vous trouverez quelques pages de description des principales commandes en Python 3 utiles pour la résolution des énigmes.

Pour chaque énigme, trois niveaux progressifs d'indices se trouvent dans un chapitre séparé au milieu du livre. Ainsi, si après avoir commencé à réfléchir, vous êtes bloqué, vous trouverez grâce à ces indices une aide graduée pour vous donner un coup de pouce et vous mettre sur la piste de la solution.

Les niveaux de difficulté des énigmes sont indiqués par des étoiles. Le niveau le plus facile, concernant les énigmes qui vous permettront de vous échauffer, est représenté par ☆. Pour les résoudre, il faudra écrire des programmes Python 3 assez simples.

Le niveau intermédiaire, dont les énigmes reposent sur une réflexion et des calculs plus complexes, est représenté par ☆☆. Il arrive que la solution repose sur une astuce un peu moins évidente que dans le premier niveau, les programmes à écrire pour ces énigmes étant un peu plus complexes.

Le niveau ☆☆☆ est le niveau le plus difficile. Il comporte des énigmes qui nécessitent beaucoup de réflexion ou qui demandent des connaissances en mathématiques et en informatique un peu plus avancées. Il faudra faire preuve de ténacité pour concevoir ces programmes afin de trouver la solution.

Les thèmes des énigmes sont l'occasion de découvrir de nombreux concepts importants en informatique. La plus grande partie de cet ouvrage est constituée des solutions détaillées de toutes les énigmes. Chaque solution contient non seulement le programme Python 3 amenant à la résolution de l'énigme, mais aussi des explications détaillées sur la conception de l'algorithme corres-

pondant. En guise de clin d'œil, chaque solution est accompagnée d'une citation scientifique ou littéraire en rapport avec l'énigme ou sa solution.

L'aspect ludique de ce livre motivera certains lecteurs pour coder les solutions et les poussera à faire preuve de créativité pour écrire des programmes résolvant ces énigmes.

Cet ouvrage s'adresse à un public qui aime programmer pour découvrir et apprendre des concepts importants de l'informatique. Il s'adresse aussi, indirectement, à tous les enseignants d'informatique, car ces énigmes sont une banque d'exercices corrigés au même titre qu'un manuel de cours.

Les énigmes, les indices et les solutions contiennent de nombreux encarts : biographiques, historiques, techniques, mathématiques, Python 3, culturels, de solution de codes Python 3, d'objectifs pédagogiques pour les enseignants en rapport avec les concepts abordés ou encore des pistes pour aller plus loin. Ils sont représentés respectivement par ces icônes :








Les fichiers des énoncés des énigmes, des programmes des solutions et des compléments en Python 3 sont disponibles à l'adresse suivante :

http://dunod.link/kl9s5z5

Remerciements : Nous remercions Cédric Lauradoux de nous avoir montré la voie pour la création de ces énigmes. Nous adressons aussi nos remerciements à Flavien Binet, Emmanuel Delay, Diane Leblanc-Albarel, Christine Solnon, Thibault Ralet pour leurs contributions à l'élaboration du contenu de ce livre. Nous exprimons également notre gratitude à Matthieu Daniel et Anne Le Duc pour leurs commentaires et suggestions constructives, à la suite de leurs relectures assidues.

Clermont-Ferrand, le 12 septembre 2023.
Malika More, Guenaëlle De Julis, et Pascal Lafourcade *.

* Nous serons heureux de répondre à vos questions par email.

Chaque problème a sa solution, il y a donc des codes pour tous les goûts.

Une brève introduction à Python 3

Dans cet ouvrage, les solutions des énigmes sont données en Python 3 à titre d'illustration, un logiciel libre [#]. Il est bien entendu possible de rédiger les solutions dans le langage de programmation de votre choix, puisqu'un langage de programmation comprenant des instructions pour effectuer des calculs arithmétiques, des affectations de variables, des instructions conditionnelles et des boucles (`while`) permet de programmer n'importe quel algorithme.

1 JAN Origine de Python

Le langage Python a été créé par Guido van Rossum en février 1991. À cette époque, il lisait un recueil des sketchs de la série télévisée comique britannique *Monty Python's Flying Circus*, diffusée sur la BBC dans les années 1970. Il cherchait pour son langage de programmation un nom court, unique et légèrement mystérieux, et il choisit « Python », en l'honneur de cette émission dont il était fan.
Concernant les versions successives du langage Python, il est important de savoir que la version 2 n'est pas compatible avec la version 3, car de nombreux changements de syntaxe ont été faits. De plus, depuis 2020, la version Python 2 n'est plus mise à jour.

Figure 1 – Guido van Rossum.

Les commandes les plus utilisées dans les solutions des différentes énigmes sont présentées et, dans certains cas, des précisions supplémentaires sont données dans les énoncés ou dans les indices. Pour plus de détails sur Python 3, et pour apprendre à programmer en Python 3 de façon organisée, il existe de nombreux ouvrages et des sites de cours dédiés. Le site de référence (mais qui n'est pas le plus simple pour débuter), est le manuel officiel accessible en ligne `https://docs.python.org/fr/3/`. Si vous êtes déjà à l'aise avec Py-

[#] `https://www.python.org`

thon 3, vous n'aurez probablement pas besoin de vous référer à ce chapitre. Sinon, n'hésitez pas à le parcourir pour chercher des idées lorsque vous manquez d'outils pour programmer la solution d'une énigme.

Commenter tout ou partie d'une ligne

Tout d'abord, notez que le symbole # permet, en Python 3, de commenter le reste de la ligne, ce qui signifie que tout ce qui est écrit après ce symbole sur la même ligne n'est pas pris en compte lors de l'exécution du programme.

Syntaxe et bonnes pratiques

Des règles sont définies dans PEP [†] afin d'homogénéiser le code et d'appliquer des bonnes pratiques qui facilitent le travail sur des projets à plusieurs. Ces règles ne changent pas l'exécution d'un programme. Ces conventions de style, dont les principales sont présentées ci-dessous, sont définies dans PEP8.

Tout d'abord, il est essentiel de savoir que les indentations en Python 3 jouent un rôle crucial et que leur présence ou leur absence aux bons endroits peut modifier le comportement d'un programme, voire générer des erreurs. La proposition PEP8 recommande d'utiliser une indentation de quatre espaces.

Ensuite, les longues lignes doivent être évitées. La proposition suggère des lignes de maximum 80 caractères. Pour cela, il existe l'opérateur \ connu sous le nom de saut de ligne explicite : il permet de diviser une seule longue ligne continue en plusieurs lignes de code plus petites et plus faciles à lire.

```
1 string = "Cette" + "chaîne" + "est" + "bien" + "trop" + "longue" \
2  + "pour" + "tenir" + "sur" + "une" + "ligne."
```

Concernant le nommage des variables, des fonctions ou des constantes, là encore la proposition PEP8 indique que :

- les noms de variable ou de fonction doivent être en *snake case*, c'est-à-dire en minuscules, avec une séparation des mots par des «_».
  Par exemple : `une_variable`, `une_fonction(param1, param2)`.
- les constantes sont aussi en *snake case* mais en majuscules.
  Par exemple : `UNE_CONSTANTE`.

Par ailleurs, lorsqu'un programme grossit, il est habituel de le découper en plusieurs fonctions. Pour faciliter la lecture, il est recommandé (bien que non explicité dans PEP) d'attribuer des noms explicites aux fonctions avec, de préférence, un verbe d'action qui la décrit. Par exemple, une fonction qui lit une

[†] `https://peps.python.org/pep-0000/`

image peut s'appeler `lire_image()`, ou encore une fonction qui compte les voisins d'un nœud dans un graphe peut s'appeler `compter_voisins()`.

Enfin, pour une question de lisibilité, la proposition recommande de mettre une espace avant et après les opérateurs :

```
# incorrect
x=1
x = x+1
x = (a+b) * (a-b)
une_liste = [1,2,3]

# correct
x = 1
x = x + 1
x = (a + b) * (a - b)
une_liste = [1, 2, 3]
```

Affectation et opérations mathématiques simples

Le symbole `=` permet d'affecter la valeur d'une expression à une variable. Par exemple, les deux instructions `x=3` puis `x=x+1` affectent successivement la valeur `3` à la variable `x` puis la valeur `3+1`, soit `4`, à cette même variable.

Les opérations numériques (addition, soustraction, multiplication, division, puissance), ainsi que les opérations booléennes (négation, et, ou), se notent respectivement `a+b`, `a-b`, `a*b`, `a/b`, `a**b`, `not a`, `a and b`, `a or b`. Concernant la division euclidienne, si $a = b.q + r$ avec $0 \leq r < b$, alors q = `a//b`, et r = `a%b` ou q, r =`divmod(a, b)`.

Lorsque ces opérations sont possibles, pour convertir la variable `y` en entier, il faut écrire `x=int(y)`, et pour convertir la variable `x` en chaîne de caractères, il faut écrire `y=str(x)`.

Instruction conditionnelle : `if`

Pour exprimer une condition, les mots-clés `if`, `else` et `elif` sont utilisés. Le code suivant affecte la valeur `3` à la variable `res` si la valeur de la variable `x` est strictement plus grande que `3`. Sinon, si `x` vaut zéro, alors `res` reçoit la valeur `0`, et sinon `res` prend la valeur `1`.

```
if x > 3:
    res = 3
elif x == 0:
    res = 0
else:
    res = 1
```

Il est important de remarquer que `==` est utilisé dans les tests des conditions et que ce symbole est différent de celui utilisé pour symboliser l'affectation. Par ailleurs, le symbole : après la comparaison et une tabulation pour chacun des cas sont obligatoires. Les parties du code correspondant à `elif` et à `else` ne sont utilisées que si l'algorithme les rend nécessaires. Dans le cas contraire, il est possible de ne pas les écrire.

Les opérateurs de comparaison usuels se notent comme suit : `a<b`, `a<=b`, `a>b`, `a>=b`, `a==b`, et `a !=b` (pour `a` différent de `b`).

Boucles : `while` et `for`

Le programme suivant affiche les entiers de 0 à 9 grâce à une boucle `while`.

```
1 i = 0
2 while i < 10:
3     print(i)      #0 1 2 3 4 5 6 7 8 9
4     i = i + 1
```

Le programme suivant fait la même chose avec une boucle `for`. En Python 3, la fonction `range(i)` retourne l'ensemble des valeurs entières entre `0` et `i-1`.

```
1 for i in range(10):
2     print(i)      #0 1 2 3 4 5 6 7 8 9
```

Lecture de données dans un fichier

Dans de nombreuses énigmes, un fichier de données est accessible en ligne (lien en page VI), afin d'éviter de recopier l'énoncé. Il est important de savoir lire ces données pour pouvoir les manipuler dans un programme en Python 3.

Le programme suivant permet de lire ligne par ligne et un par un les mots du fichier `f.txt`

```
1 with open('./f.txt', mode='r', encoding='utf8') as f:
2     for ligne in f:
3         for mot in ligne.split():
4             print(mot)
```

L'instruction `with` garantit que le fichier est proprement fermé, même si une erreur se produit.

Le programme suivant, quant à lui, permet de lire caractère par caractère le contenu du fichier `f.txt`

```
with open('./f.txt', mode='r', encoding='utf8') as f:
    caractere = f.read(1)
    while caractere:
        print(caractere)
        caractere = fichier.read(1)
```

Opérations sur les chaînes de caractères

La chaîne de caractères vide est notée `""` (sans espace entre les guillemets). Pour concaténer deux chaînes de caractères, comme `"Bon"` et `"jour"`, il suffit d'utiliser le symbole d'addition `+`. Ainsi `"Bon" + "jour"` vaut `"Bonjour"`. Par ailleurs, en général, les guillemets simples (`'`) ou doubles (`"`) peuvent indifféremment être utilisés pour encadrer une chaîne de caractères.

Le code ASCII [#] comporte 128 caractères pour lesquels il est possible de passer de la valeur au symbole. Par exemple, le code `chr(97)` vaut `'a'` et le code `chr(122)` vaut `'z'`, mais le code `chr(123456789)` renvoie une erreur car le code ASCII est limité aux nombres entre 0 et 127. Réciproquement, la commande `ord('a')` renvoie `97`.

Opérations sur les listes

La liste vide est notée `[]`. Pour ajouter un élément `x` à la fin de la liste `L`, l'instruction `L.append(x)` est utilisée. Pour accéder au premier élément de la liste `L`, il faut écrire `L[0]` car les indices des listes commencent à `0` en Python 3. Pour obtenir le dernier élément de la liste, il est possible d'écrire `L[-1]`.

```
L=[]
L.append(4)
L.append(5)
L.append(6)
print(L)              # [4, 5, 6]
print(L[1])           # 5
print(len(L))         # affiche 3, le nombre d'éléments de la liste L
print(L[-1])          # affiche le dernier élément de la liste L
print(L[len(L) - 1])  # affiche le dernier élément de la liste L
```

Il est également possible d'accéder aux éléments d'une liste `L` par tranche en partant de l'indice `debut` jusqu'à l'indice `stop`, avec un pas `p` et la notation suivante `L[debut:stop:p]`. Cette opération est couramment appelée *slicing*.

```
b = "Bonjour!"
print(b[3:7])         # affiche la chaîne "jour"
print(b[2:8:2])       # affiche la chaîne "nor"
```

[#] American Standard Code for Information Interchange

Le pas est optionnel. Si le début n'est pas indiqué alors le début de la liste est considéré par défaut, de même pour la fin. Ainsi, le code `S == S[: :-1]` teste si `S` est un palindrome. De plus, la méthode `join()` permet d'assembler facilement des chaîne de caractères.

```
t = ["Bon", "jour", " !"]
print("".join(t))              # affiche "Bonjour!"
```

Types muables et immuables

En Python, les données sont stockées dans deux types de conteneur : muable et immuable. Dans un type muable, la valeur peut être modifiée, contrairement aux types immuables. Par exemple, une liste ou un dictionnaire sont muable, alors que les nombres, les *n*-uplets ou les chaînes de caractères sont immuables.

```
une_liste = [0, 1, 2]
une_liste[0] = 3
print(une_liste)
# affiche [3, 1, 2]

un_n_uplet = (0, 1, 2)
un_n_uplet[0] = 3
# génère une erreur :
# TypeError : 'tuple' object does not support item assignment

une_chaine = 'chaine de caractères'
une_chaine[0] = 'C'
# génère une erreur :
# TypeError : 'str ' object does not support item assignment
```

Ceci a notamment un impact sur les variables. En effet, leur assignement est une référence vers la valeur de l'objet. Or, la valeur d'un objet de type muable est une référence vers cet objet et non sa valeur.

```
# types immuables
a = 2              # une référence vers l'objet 2 est assignée à a
b = a              # une référence vers l'objet 2 est assignée à b
a = a + 1          # une référence vers l'objet 3 est assignée à a,
                   # mais l'objet référencé par b n'est pas impacté
print(a, b)        # affiche 3, 2

# types muables
a = [0, 1, 2] # une référence vers l'objet liste [0,1,2] est assignée à a
b = a          # la même référence est assignée à a
a[0] = 3       # une référence vers l'objet 3 est assignée au premier élément de l'objet liste
print(a, b)    # affiche [3, 1, 2] [3, 1, 2]
```

Fonctions et passage de paramètres

Pour définir une fonction en Python 3, il faut utiliser le mot-clé `def` suivi du nom de la fonction, par exemple `echanger()` dans l'exemple ci-dessous, puis des paramètres de la fonction, ici `x` et `y` entre parenthèses.

```
1  def echanger(x, y):
2      x = x + y
3      y = x - y
4      x = x - y
5      return (x, y)
6  a = 3
7  b = 4
8  print(echanger(a, b)) # affiche (4,3)
9  print(a, b)           # affiche 3 4
```

Les variables définies dans la fonction `echanger` sont des variables dites *locales* qui n'ont pas de lien avec les variables définies à l'extérieur de la fonction, bien qu'elles puissent porter le même nom. Les variables locales n'existent qu'à l'intérieur de la fonction et des valeurs leur sont affectées lors de l'appel de la fonction. Ici, `x` prend la valeur 3 et `y` la valeur du paramètre `y`, soit 4.

En Python 3, tous les arguments d'une fonction sont passés par *affectation*, comme pour assigner une valeur à une variable. Cela signifie que le paramètre devient une variable locale à laquelle est assignée une référence vers l'argument. Cependant, lorsque l'argument est un objet de type muable (liste, dictionnaire, etc.) sa valeur est une référence vers l'objet lui-même et non vers la valeur de cet objet. Cela induit que les modifications d'un argument muable ne sont pas locales à la fonction, comme l'illustre l'exemple ci-dessous.

```
1  def ajouter_dans_liste ( liste , x):
2      liste .append(x)
3
4  L = [1, 2, 3]
5  print(L)                    # affiche [1, 2, 3]
6  ajouter_dans_liste (L, 4)
7  print(L)                    # affiche [1, 2, 3, 4]
```

La modification faite sur la liste passée en paramètre est effective sur la liste définie en dehors de la fonction `ajouter_dans_liste()`.

Gestion des erreurs

En Python 3, il est possible de contrôler les messages d'erreurs et d'éviter ainsi qu'un programme plante sans en informer l'utilisateur. Par exemple, dans le code suivant, comme la variable `x` n'est pas définie, le programme ne va pas fonctionner et, grâce aux commandes `try` et `except`, le programme affiche :

`Une erreur s'est produite`. Si la variable `x` est définie avant le `try`, alors l'erreur `Division par zéro` est levée.

```
try :
        print(x / 0)
except ZeroDivisionError :
       print("Division par zéro")
except :
        print("Une erreur s'est produite")
```

Plusieurs types d'erreurs sont interceptées en Python 3, les plus communes étant : `NameError`, `ValueError`, `KeyError`.

Bibliothèques

Dans Python 3, il est possible d'inclure des *bibliothèques*, en général thématiques, composées de fonctions déjà programmées. Pour cela, il faut indiquer le nom de la bibliothèque au début du programme, précédé du mot-clé `import`. Par exemple, `import math` permet d'utiliser dans la suite du programme les fonctions mathématiques telles que `math.sqrt(x)` et `math.cos(x)`, pour calculer respectivement la racine carrée et le cosinus d'un nombre `x`.

D'autres bibliothèques sont accessibles comme les bibliothèques `random` ou `time`. La bibliothèque `random` permet de générer des nombres aléatoires. Avec la commande `time()` de la bibliothèque `time`, il est possible de connaître l'heure et ainsi de mesurer, par exemple, la durée d'exécution d'un programme.

```
import time
import random

debut = time()                  # donne l'heure courante
r = random.randint(10, 100)     # tire au hasard un entier entre 10 et 100
fin = time()
duree = fin - debut
print(duree)
```

Il est aussi possible d'ajouter des bibliothèques qui ne sont pas par défaut dans Python 3. Par exemple, la bibliothèque `numpy` peut être installée grâce à cette commande dans un terminal : `python3 -m pip install numpy`.

Par commodité, il est possible de renommer, dans un programme Python 3, une bibliothèque, à l'exemple de la bibliothèque `numpy` qui est souvent abrégée par `np` comme, dans le programme ci-dessous.

```
import numpy as np

zeros = np.zeros([10, 10])      # construit une matrice de taille 10 x 10 remplie de 0
uns = np.ones([10, 10])         # construit une matrice de taille 10 x 10 remplie de 1
t = np.linspace(3, 4, 11)       # construit une liste de 11 valeurs allant de 3 à 4
print(t)                        # [3. 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.]
```

1

Les énigmes à résoudre

1 Le Jeu de la vie ☆

Le jeu de la vie a été inventé en 1970 par le mathématicien britannique John Horton Conway. L'appellation de « jeu » est trompeuse, puisqu'il n'y a pas réellement de joueur, et qu'il ne se termine jamais. Il s'agit d'un système dynamique discret, c'est-à-dire qui évolue tout seul au cours du temps à partir d'une configuration initiale et d'un ensemble de règles, pendant une durée illimitée. Le jeu se déroule sur une grille bidimensionnelle infinie. Les cases de la grille sont appelées des *cellules*. Le but est de simuler le déroulement de la *vie* de ces cellules avec des règles simples. À un instant donné, chaque case de la grille est soit « vivante », soit « morte » (les cases vivantes sont souvent représentées en noir et les mortes en blanc). Chaque cellule a huit voisins :

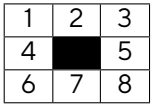


Figure 2 – Les huit cellules voisines de la cellule vivante centrale.

À partir d'une configuration de cellules vivantes et mortes l'état de chaque cellule évolue simultanément en fonction de ses voisines :

- une cellule morte devient vivante si exactement trois de ses cellules voisines sont vivantes, sinon elle reste morte ;
- une cellule vivante reste vivante si elle posséde deux ou trois cellules voisines vivantes, sinon elle meurt d'étouffement ou d'isolement.

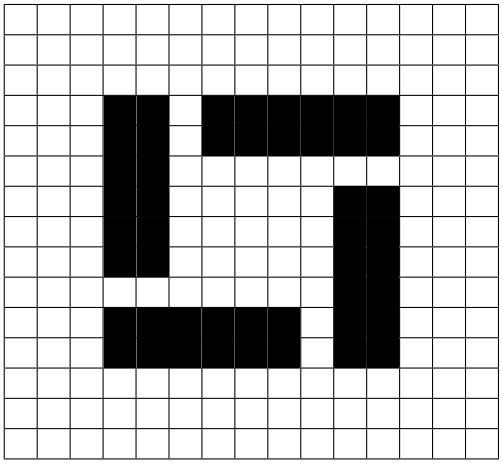
Cet *automate cellulaire* fascine à cause de la variété des configurations (parfois étonnantes et/ou esthétiques) et des comportements (stabilité, périodicité, déplacement, etc.) qui apparaissent pendant le calcul, en fonction de la configuration initiale, malgré la très grande simplicité des règles d'évolution.

Énigme 1 : *Saurez-vous écrire un programme Python 3 qui affiche le dessin produit après la 200^{e} itération à partir de la grille de la figure 3 ?*

Solution page 89.

Pour simplifier les aspects graphiques du programme, il est suggéré de travailler en ASCII art, par exemple en utilisant le caractère « . » pour les cellules mortes, et le caractère « # » pour les cellules vivantes. La grille initiale de l'énigme est alors composée de 15 lignes de 15 caractères chacune.

Il est possible de modéliser cette grille à l'aide d'une liste de listes en Python 3, ce qui permet d'accéder à chaque cellule par ses coordonnées.



```
. . . . . .
. . . . . .
. . . . . .
. . .##.######. . .
. . .##.######. . .
. . .##. . . . . .
. . .##. . . . .##. . .
. . .##. . . . .##. . .
. . .##. . . . .##. . .
. . . . . .##. . .
. . .######.##. . .
. . .######.##. . .
. . . . . .
. . . . . .
. . . . . .
```

Figure 3 – Grille de départ.

John Horton Conway (26/12/1937 - 11/04/2020)

Mathématicien anglais, il a étudié l'algèbre, la géometrie, la théorie des nombres, la théorie des nœuds et la théorie des jeux combinatoires. En 1970, il a inventé le jeu de la vie. Il a obtenu le prix Berwick en 1971 et il a été le premier lauréat du prix Pólya en 1987.

En 1967, avec Michael Paterson, il a conçu *Sprouts*, un jeu de réflexion à deux joueurs qui se joue avec un stylo et un papier. Au départ, il y a *n* points dessinés sur la feuille de papier. À tour de rôle, chaque joueur relie deux points (pas forcément différents) avec une ligne droite ou courbe et ajoute un point n'importe où sur cette ligne. Les lignes ne doivent ni se croiser entre elles, ni se croiser elles-mêmes et un point ne peut être relié à plus de trois lignes. Une ligne reliant un point à lui-même compte pour