The Riddler

Ou, en bon Voltairien moyen, « l'homme mystère ». Le personnage est apparu dans la série Batman, en tant qu'ennemi juré du Dark Knight (Bruce Wayne). Le personnage est assez amusant surtout interprété au cinéma par Jim Carrey. Un jeu de mots (private joke) lui donne son nom : Edouard Nygma devient E-Nygma (énigme). Toutefois, aujourd'hui, nous ne parlerons pas de cinéma, nous allons continuer à approfondir nos connaissances en cryptographie, (et accessoirement en python) en regardant le codage judicieusement nommé « enigma ». Merci FREELOG de me rendre chaque mois plus intelligent ;-).

De l'histoire historique.
L'histoire mondiale connaît bien le nom d'enigma et lui donnera un autre sens : la terrible « machine enigma ». Pour bien comprendre ce qu'évoque cette notion, faisons un tour dans le passé et dans l'histoire de la seconde guerre mondiale. A cette époque, les services du chiffre allemand mettent au point une machine redoutable qui permet de crypter les communications. D'après son concepteur, elle est inviolable. Autrement dit, on ne peut pas percer le code d'un message crypté par cette machine et transmis aux soldats sur le terrain. Pour les militaires, pouvoir communiquer sans que l'ennemi ne sache ce qui est transmis (même s'il peut capter le message chiffré) est vital (ou à tout le moins, décisif).

A quoi ressemble la bête ?
C'est une sorte de machine à écrire, surmontée d'un système d'engrenages, sur laquelle un opérateur tape un texte en clair, et lit sur un système d'affichage lumineux le texte codé. Le système est réversible (c'est à dire que la frappe du texte codé redonne le texte en clair : méthode simple et efficace adaptée au terrain particulier de l'action). Le codage est réalisé par une succession de roues dentées (3 dans la version initiale). Les anglais réussiront à casser ce code au moyens de « cribles » et de « bombes ». Les cribles étant des tables de lectures permettant d'identifier les redondances et les bombes des embryons d'ordinateurs. Lorsque les allemands se rendront compte que leur code a été cassé, il mettront en service le dernier engrenage, inutilisé jusque là. A noter que les U-boote (untersee-boote : sous-marins) utiliseront toujours 1 engrenage de plus que les autres corps d'armée (4 roues dentées pour tout le monde; 5 roues pour eux). Les anglais réussiront de nouveau à casser le code, notamment grâce à l'aide américaine, qui perfectionnera les bombes d'après les travaux d'Alan Turing, en connectera plusieurs (les premiers réseaux) pour diminuer les temps de traitements, et cela aboutira finalement à la création de « colossus » le premier engin que l'on puisse qualifier d'ordinateur. Grâce à cela des milliers de vies furent sauvées et il est probable qu'ainsi le sort de la guerre évolua favorablement en faveur de la démocratie.

Le principe
L'opérateur radio reçoit un message avec le mot de passe du jour, et le texte crypté. Il positionne les engrenages selon le mot de passe fourni (mise des engrenages en position initiale), et saisit ensuite le texte crypté. Il lit en temps réel, la traduction de ses instructions sur l'ancêtre de l'afficheur à diodes. Ce système put être « cassé » grâce aux travaux d'Alan Turing et à l'Intelligence Service anglaise, car il repose simplement sur le très grand nombre de combinaisons rendues possibles par la multiplication des engrenages. La machine offrait aussi la possibilité de compliquer les choses avec l'emploi d'un système avec fiche électrique et un codage linéaire en retour par le train d'engrenages (une sorte de surchiffrement). Quasiment tous les articles scientifiques qui traitent de cryptographie mentionnent ce type de codage. Des films et des documentaires existent sur ce sujet, ainsi que des livres, voire des romans Je vous invite à vous y reporter pour trouver des renseignements plus précis sur ce sujet.

Notre réalisation
Nous allons nous aussi utiliser un système similaire pour procéder au codage de textes divers et

variés. Heureusement, python est notre ami, car il possède une bibliothèque spécialement dédiée à cet usage. Si vous avez bien suivi jusqu'à maintenant, vous savez que la complexité du code généré s'exprime en fonction du nombre d'engrenages utilisés pour le codage. Plus il y en a, et plus le codage est difficile à contourner. Bien sûr, cela n'arrêtera pas les spécialistes du décryptage, mais peut offrir une sécurité suffisante pour vous e-mails, lorsque vous souhaitez transmettre des informations un peu sensibles (Eusébio, tu es beau;-)). Pour les puristes, il convient de préciser que python n'émule pas scientifiquement un machine énigma. Il se contente de retenir le principe du codage sur n roues dentées avec une position initiale constituée par la clé. Encore une fois, n'oubliez jamais que la longueur de la clé est vitale pour assurer une protection maximale, et dans le cas d'enigma, augmenter le nombre d'engrenages utilisés complique la tâche de « cassage » de code.

Kékonfé ?
Une fois de plus, nous allons sacrifier sur l'autel de la connaissance de python, en parcourant le code pas à pas. Comme toujours, nous allons commencer par les imports. Comme Python possède une machine enigma virtuelle, en plus du classique Tkinter, nous allons invoquer le module rotor qui est en fait, cette fameuse machine virtuelle.

```
from Tkinter import *
import rotor
```

Il aurait été possible de créer une machine enigma plus simple, avec un codage ne prenant pas en compte l'aspect graphique de l'application. Dans ce cas, le codage se fait sur quelques lignes. Toutefois, dans ce cas précis, il est nettement plus interessant de travailler avec une interface graphique. Outre le confort visuel (toujours sommaire dans nos réalisations, mais on est pas là pour faire beau, mais pour faire efficace ;-)))), il ne faut pas oublier que cette méthode permet de réutiliser la phrase codée par un simple copier-coller pour la placer dans un e-mail. Inversement, on pourra, toujours par copier-coller, récupérer la phrase cryptée sur le mail pour la coller à sa place avant le décodage. Donc, on importe « Tkinter » pour l'interface graphique, et on importe « rotor » pour implémenter la machine enigma. Rien de plus simple.

Ensuite, nous allons passer à la phase « initialisations ». En effet, nous aurons besoin de 2 variables fondamentales, que nous devrons pouvoir utiliser à tous les niveaux d'imbrication de notre réalisation : le mot de passe et le texte à traiter, soit en codage, soit en décodage. Donc, il convient de créer 2 variables globales recpsaisiemdp et recupsaisietxt.

```
global recupsaisiemdp
recupsaisiemdp='toto'
global recupsaisietxt
recupsaisietxt='nibe'
```

Bien entendu, il faut les initialiser. Dans ce cas précis, comme d'habitude, nous procédons avec finesse et originalité ;-).

Au coeur du problème
Avec Python, on peut travailler vite et efficacement. Il va de soi que nous aurons 2 procédures : l'une pour le codage, l'autre pour le décodage. M. de La Palice n'aurait pas fait mieux. Examinons en détail la partie codage. Comme toujours, nous commençons par la définition de la fonction.

```
def Codage():
  motdepasse=mot_de_passe.get()
  rt = rotor.newrotor(motdepasse, 12)
  message_cyphe=rt.encrypt(texte_a_travailler.get())
```

```
  txt.insert(END,message_cyphe)
  del rt
```

Ce qui suit la définition de fonction peut être codé immédiatement, sans même attendre de connaître précisément la partie interface graphique. Nous savons que nous devons récupérer les 2 variables que nous avons déclarées globales. Nous créons donc les variables en local avec des noms explicites : motdepasse et message_cyphe. A titre d'anecdote, on devrait dire cyphé au lieu de crypté, car ce dernier terme signifie « caché ». Or le texte n'est pas caché. Seul son sens n'est pas apparent. Fin du quart d'heure culturel ;-). Comme on sait que l'on va récupérer les chaines de caractères adéquates dans l'interface graphique, on utilise la méthode .get( ) qui nous permettra de lier chaîne saisie et nom de variable. A l'issue, la variable locale motdepasse contiendra la chaîne de caractère située dans la zone adéquate de saisie. C'était le plus difficile à faire. Ensuite, il suffit de créer une instance de l'objet rotor (par convention, elle s'appelle « rt ») et l'on indique que ce nouveau rotor a comme clé motdepasse et sera codé sur 12 roues : rotor.newrotor(motdepasse, 12). Ben voilà, c'est quasiment fini. Comme pour le mot de passe, on récupère le texte à travailler par .get( ) et on l'affecte à « message_cyphe »,; non sans avoir effectué dans le même temps, le codage proprement dit par la méthode encrypt de l'instance rt de l'objet rotor (qui est notre enigma). Il ne reste plus qu'à procéder à un affichage en bonne et due forme. C'est le rôle de txt.insert, qui affichera le texte cyphe dans une zone adéquate. Pour terminer proprement, on détruit l'instance rt, puisqu'elle ne nous sert plus (pour cette fois. Mais comme l'on crée à chaque fois une nouvelle instance, cela n'a pas d'importance : on crée, on utilise et on détruit).

Ya toujours du monde ?
Si vous lisez ces lignes, soyez en remerciés, car vous venez de faire un bel effort. Mais l'important, c'est que vous ayez compris le principe. D'ailleurs, on vous laisse faire vous même le commentaire du morceau concernant le décodage. Vous remarquerez que le code est identique à l'exception de la méthode decrypt.

```
def Decodage():
  motdepasse=mot_de_passe.get()
  rt = rotor.newrotor(motdepasse, 12)
  message_clair=rt.decrypt(texte_a_travailler.get())
  txt.insert(END,message_clair)
  del rt
```

Encore une fois, on détruit l'instance crée. Puis, on envisage que l'utilisateur puisse cliquer pour quitter définitivement notre application (Nooooon ! Faites pas ça les copains ;-)))). Il convient donc d'écrire la procédure afférente, qui ne posera aucun problème de compréhension.

```
def quitte():
  fen1.destroy()
```

Je suis encore là
Alors, vous êtes maso ;-)) ? Bon, dans ce chapitre, on va articuler tout ça autour de notre sublime interface. On connaît ses classiques, donc, comme à chaque fois, on a commencé par les imports. Maintenant, on va créer les instances pour notre interface graphique. Toujours sans surprise, si vous êtes familier de nos ateliers.

```
fen1=Tk()
fen1.title('By the Blue Velvet League')
```

On crée la fenêtre globale, et on lui donne un titre, c'est plus sympa. Le rôle de notre interface est de permettre la saisie de la clé et du texte à coder, la sortie se faisant dans une zone texte, avec des

ascenseurs. Pour ce faire, voici le code adéquat :

```
Message_cle=Label(fen1,fg='blue',text='Votre mot de passe ').pack(fill=X)
mot_de_passe=Entry(fen1)
mot_de_passe.bind("<Return>",recupsaisiemdp)
mot_de_passe.pack(fill=X)
```

On pose un Label pour identifier la zone qui est juxtaposée, et qui est une zone de saisie Entry. Pour valider, il faudra appuyer sur la touche « return » en fin de saisie pour valider votre frappe. A noter pour ceux qui nous rejoignent les méthodes bind et pack qui font partie de la trousse de survie du programmeur python. Bind, permet d'intercepter un événement et pack permet l'affichage du widget. Pour Bind, la méthode utilisée est notée entre crochets : c'est l'action sur la touche return ou entrée pour les non anglophones qui déclenche l'action. Quant à l'indispensable méthode pack, nous lui passons en paramètres, fill=X, qui imposera au bouton de remplir tout l'espace de la fenêtre. C'est un luxe de présentation, mais même si nous avons toujours prêché pour l'efficacité au détriment de la beauté, un peu de soin dans l'interface ne fait que renforcer la lisibilité de l'ensemble. A noter le recupsaisiemdp qui ne sert pas vraiment dans ce cas. Faites l'effort de lire la documentation en ligne pour mieux apprécier bind. Nous verrons plus loin pourquoi nous avons opté pour cette méthode. Comme de juste, la deuxième partie qui concerne le texte à coder ressemble comme deux gouttes d'eau à ce que nous venons de voir.

```
Message_texte=Label(fen1,fg='blue',text='Votre texte ').pack(fill=X)
texte_a_travailler=Entry(fen1)
texte_a_travailler.bind("<Return>",recupsaisietxt)
texte_a_travailler.pack(fill=X)
```

C'est exactement le même code, aux variables près. Maintenant, nous allons codifier nos boutons pour agir sur le codage et le décodage. C'est ça, la programmation évènementielle.

```
bouton_code=Button(fen1,text='Coder',command=Codage).pack(fill=X)
bouton_decode=Button(fen1,text='Decoder',command=Decodage).pack(fill=X)
```

On crée une instance de bouton, et on lui passe les paramètres text (texte à afficher dns le bouton) et on indique par command, la procédure à lancer lors de l'appui sur le bouton. Cette fois, nous n'avons pas fait d'effort au niveau présentation, mais si vous le souhaitez, sur le même modèle que les labels, vous pouvez vous amuser avec les paramètres bg (background = couleur de fond d'écran du bouton) et fg (foreground = couleur du texte). Essayez de leur affecter les valeur 'Red', 'Yellow' et autres... Toutefois n'oubliez pas les virgules de séparation.

L'affichage
C'est par ici la sortie. Maintenant, nous allons coder une zone dans laquelle nous allons écrire. Chaque nouveau texte codé ou décodé viendra s'ajouter à la file des autres. D'où la présence des ascenseurs. Ici, aucune subtilité de codage, n'est utilisée. Le code n'appelle donc aucun commentaire particulier. Tout d'abord, on definit les 2 ascenseurs, horizontal et vertical.

```
ascenceur=Scrollbar(fen1,orient=VERTICAL)
ascenceurh=Scrollbar(fen1,orient=HORIZONTAL)
```

On travaille toujours sur notre instance fen1. Et on ajoute la zone de texte en la dimensionnant grâce à height (hauteur) et width (ben, oui, largeur...) et en y ajoutant les deux scrollbars précédemment définies.

```
txt=Text(fen1,height=10,width=50,yscrollcommand=ascenceur.set,xscrollcommand=ascenceurh.set
)
ascenceur.config(command=txt.yview)
ascenceurh.config(command=txt.xview)
```

Toujours pour le fun, on conclut cette zone par les méthodes pack des 2 ascenseurs et de la zone texte. Encore une subtilité, on fixe la barre de l'ascenseur vertical du coté droit, et celle de l'ascenseur horizontal au fond (bottom). Amusez vous à la fixer du coté gauche pour voir...

```
ascenceur.pack(side=RIGHT,fill=Y)
ascenceurh.pack(side=BOTTOM,fill=X)
txt.pack()
```

Bon, on n'est plus très loin de la touche finale. Il ne nous manque que le bouton pour quitter, et on y place tout ce que l'on vient de voir, comme paramètres.

```
boutonQ=Button(fen1,fg='red',text='Quitter',command=quitte).pack(fill=X,side=BOTTOM)
```

Et enfin la commande magique pour ajouter la touche finale :

```
fen1.mainloop()
```

qui autorisera la scrutation des événements pour l'interactivité tant prisée de nos jours.

Pas malheureux d'être à la fin
Cette fois encore, nous avons fait de la cryptographie, et accessoirement de la dactylographie. Au moment du bilan de notre réalisation, il nous reste à expliquer pourquoi nous avons opté pour l'action return. Tout simplement pour un contrôle global de la saisie avant tout clic sur un des boutons. A notre humble avis, cela permet d'éviter toute incohérence de saisie. Vous pouvez jongler avec d'autres choix. Pour le moment, vous disposez d'un instrument performant destiné au codage que vous pouvez inclure dans vos mails (et récupérer) par un simple copier/ coller. Si vous voulez faire tendance, vous préciserez bien à votre correspondant que le mot de passe doit s'échanger par un autre canal que celui par lequel transitera le message (bonjour le téléphone), ce qui évitera de donner des informations sur la clé. Comme d'habitude, notre CD n'attend que vos réalisations. Donc, n'hésitez pas à bidouiller, améliorer, trafiquer ce code pour donner à la communauté des lecteurs d'autres outils pour communiquer en toute confidentialité, et surtout n'oubliez pas que python est votre ami et FREELOG aussi ;-))).