

SUPINFO Academic Dept.

**Arithmétique et cryptographie**

Mini - Projet

Cylindre de Jefferson

Version 1.0

Last update: 02/02/2017

Use: Students/Staff

Author: Laurent GODEFROY

SOMMAIRE

0 Préambule 3

1 Généralités sur le cylindre de Jefferson 4

2 Une version Console du cylindre de jefferson 4

3 Une version GUI du CYlindre de jefferson 7

4 Bonus 13

5 Barème indicatif 14

# Préambule

Cet examen est à réaliser par groupes de deux étudiants. Dans l’unique cas où le nombre d’étudiants de la promotion est impair, un et un seul groupe de trois est autorisé.

Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d’un 0, d’une mention « cheater », et le cas échéant d’un conseil de discipline.

Ce mini-projet donnera lieu à des soutenances. Vos horaires de passages vous seront communiqués par votre campus.

Les soutenances sont également par groupes de deux. Elles dureront **20 minutes** pendant lesquelles vous montrerez à votre examinateur le bon fonctionnement de votre programme en n’en faisant la démonstration. Si vous n’avez pas implémenté le projet dans sa totalité, vous exposerez les parties fonctionnelles.

Pour appuyer votre présentation, vous devrez préparer un fichier de type Powerpoint, dans lesquels vous expliquerez les points du code que vous jugez les plus importants et significatifs. Il n’est pas nécessaire d’envoyer ce fichier à votre examinateur, ce dernier le découvrira le jour de la soutenance. Une communication précisant tout cela vous sera envoyé courant février.

D’autre part, **n'hésitez pas à utiliser le forum pour échanger sur ce projet, j'ai ouvert un fil de discussion à cet usage.**

On pourra si on le souhaite ajouter des fonctions et procédures à celles demandées.

# Généralités sur le cylindre de Jefferson

**Remarque importante**: aucun code n’est demandé dans cette partie qui n’est qu’explicative.

Le but de ce mini-projet est d’implémenter en Python un programme simulant le célèbre **cylindre de Jefferson**.

Rappelons que cet appareil est constitué d’un certain nombre de disques, 36 pour l’original, pivotants autour d’un axe et sur lesquels sont inscrits des alphabets désordonnés. Les deux correspondants disposent des mêmes disques.

Chaque disque est identifiable par un numéro. La clé est l’ordre dans lequel les disques sont insérés sur l’axe, il s’agit donc d’une suite de numéros.

Pour chiffrer un message, l’expéditeur arrange les disques selon la clé, puis les fait pivoter de telle sorte que le message apparaisse sur une même ligne du cylindre. Le message chiffré sera alors le contenu de la sixième ligne suivante (ce dernier choix n’est que conventionnel, on aurait pu en effectuer un autre).

Pour déchiffrer un message, le destinataire arrange bien sûr lui aussi les disques selon la clé, puis les fait pivoter de telle sorte que le message chiffré apparaisse sur une même ligne du cylindre. Le message d’origine sera alors le contenu de la sixième ligne précédente.

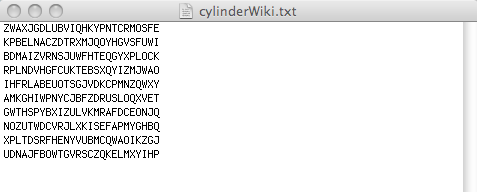
Voir <https://en.wikipedia.org/wiki/Jefferson_disk> pour des compléments et une photographie du cylindre original.

# Une version Console du cylindre de jefferson

**Il vous est fortement recommandé de lire l’intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.**

**Dans un fichier que l’on nommera « JeffersonShell.py », implémenter les sous-programmes suivants :**

* Une fonction « convertLetters(text) » où **text** est une chaîne de caractères, et qui la retournera après avoir supprimé les espaces, signes de ponctuations, lettres accentuées, etc.
* Une fonction « mix() » qui retourne une chaîne de caractère constituée des 26 lettres de l’alphabet en majuscules dans un ordre aléatoire.
* Une procédure « createCylinder(file,n) » où **file** est le nom d’un fichier texte et **n** un entier strictement positif. Elle ouvrira le fichier en écriture, puis y écrira **n** lignes, chacune d’elles étant générée à l’aide de la fonction précédente.
* Une fonction « loadCylinder(file) » où **file** est le nom d’un fichier texte dont chaque ligne contient une permutation des 26 lettres de l’alphabet en majuscules. Cette fonction retournera un dictionnaire dont les clés sont les entiers compris entre un **1** et le nombre de lignes du fichier, la valeur correspondante à une clé **i** étant la ième ligne du fichier.  
  **Exemple** : avec le fichier



cette fonction retournera ce dictionnaire :

|  |
| --- |
| {1: 'ZWAXJGDLUBVIQHKYPNTCRMOSFE', 2: 'KPBELNACZDTRXMJQOYHGVSFUWI', 3: 'BDMAIZVRNSJUWFHTEQGYXPLOCK', 4: 'RPLNDVHGFCUKTEBSXQYIZMJWAO', 5: 'IHFRLABEUOTSGJVDKCPMNZQWXY', 6: 'AMKGHIWPNYCJBFZDRUSLOQXVET', 7: 'GWTHSPYBXIZULVKMRAFDCEONJQ', 8: 'NOZUTWDCVRJLXKISEFAPMYGHBQ', 9: 'XPLTDSRFHENYVUBMCQWAOIKZGJ', 10: 'UDNAJFBOWTGVRSCZQKELMXYIHP'} |

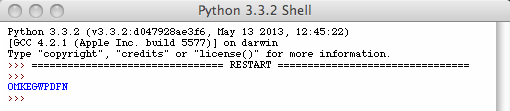
* Une fonction « keyOK(key,n) » où **key** est une liste d’entiers et **n** un entier strictement positif. Elle retournera **True** si la liste est une permutation de tous les entiers compris (au sens large) entre **1** et **n**.
* Une fonction « createKey(n) » où **n** est un entier strictement positif. Elle retournera une liste constituée des entiers compris (au sens large) entre **1** et **n** dans un ordre aléatoire.
* Une fonction « find(letter,alphabet) » où **letter** est une lettre majuscule de l’alphabet et **alphabet** une chaîne de caractères constituée des 26 lettres de l’alphabet en majuscules dans un certain ordre. Elle retournera l’indice de la lettre dans la chaîne. On codera soi-même cet algorithme de recherche sans utiliser une fonction toute faite du langage.
* Une fonction « shift(i) » où **i** est un entier compris (au sens large) entre 0 et 25. Elle retournera **i + 6** avec une congruence modulo 26.
* Une fonction « cipherLetter(letter,alphabet) » où **letter** est une lettre majuscule de l’alphabet et **alphabet** une chaîne de caractères constituée des 26 lettres de l’alphabet en majuscules dans un certain ordre. Elle retourne la lettre située 6 positions après la lettre passée en paramètre dans la chaîne passée elle aussi en paramètre. On suppose bien sûr que l’alphabet en question est circulaire.
* Une fonction « cipherText(cylinder,key,text) » où **cylinder** est un dictionnaire dont les clés sont les premiers entiers à partir de **1** et les valeurs sont des chaînes de caractères constituée des 26 lettres de l’alphabet en majuscules dans un certain ordre, **key** une liste d’entiers et **text** une chaîne de caractères. Elle vérifiera d’abord si la clé est valide par rapport au cylindre. Si c’est le cas, elle convertira la chaîne en lettres, puis retournera la chaîne chiffrée selon la méthode de Jefferson.
* Voir comment utiliser/modifier les fonctions précédentes pour procéder au déchiffrement d’un texte.

**Exemple de fonctionnement (repris du site wikipédia) de certains de ces sous-programmes :**

* Ce code :

|  |
| --- |
| cylinder = loadCylinder("cylinderWiki.txt")  key = [7,9,5,10,1,6,3,8,2,4]  print(jeffersonCipher(cylinder,key,"Retreat Now")) |

* Donne ce résultat :



**Utilisation de ces procédures et fonctions :**

* Déchiffrer le texte ‘GRMYSGBOAAMQGDPEYVWLDFDQQQZXXVMSZFS’ avec le cylindre ‘MP-1ARI.txt’ et la clé [12, 16, 29, 6, 33, 9, 22, 15, 20, 3, 1, 30, 32, 36, 19, 10, 35, 27, 25, 26, 2, 18, 31, 14, 34, 17, 23, 7, 8, 21, 4, 13, 11, 24, 28, 5].
* Chiffrer un texte de votre choix avec un cylindre et une clé de votre choix. Les inclure dans le projet.

**Trois petites questions qui attendent des réponses justifiées :**

* Que pensez-vous de la sécurité de cet algorithme ?
* Quels sont ses principales qualités et défauts ?
* Combien y-a-t-il de clés avec un cylindre de **n** disques ?

# Une version GUI du CYlindre de jefferson

**Il vous est fortement recommandé de lire l’intégralité de cette partie avant de commencer à coder. En particulier l’exemple final. Les travaux demandés sont mis en évidence avec une couleur bleue. On devra nécessairement utiliser la librairie graphique Pygame. L’usage d’une autre librairie ne sera pas pris en compte.**

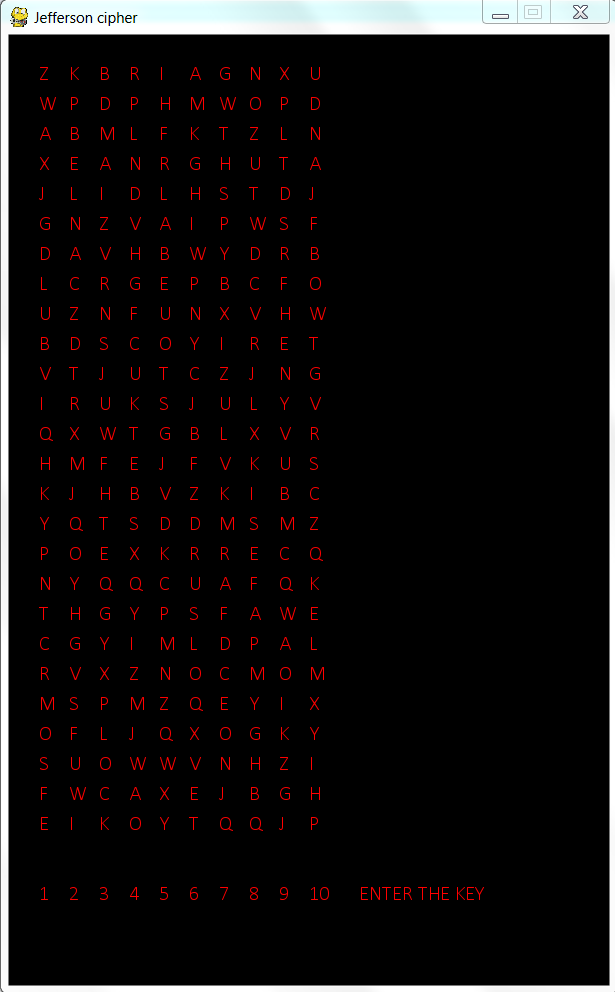
**Remarque :** vous êtes libres de désigner votre cylindre comme bon vous semble tant que les fonctionnalités sont présentes. Les captures d’écran de cette partie ne sont là que pour vous donner une idée.

**Dans un fichier que l’on nommera « JeffersonGUI.py », implémenter les sous-programmes suivants :**

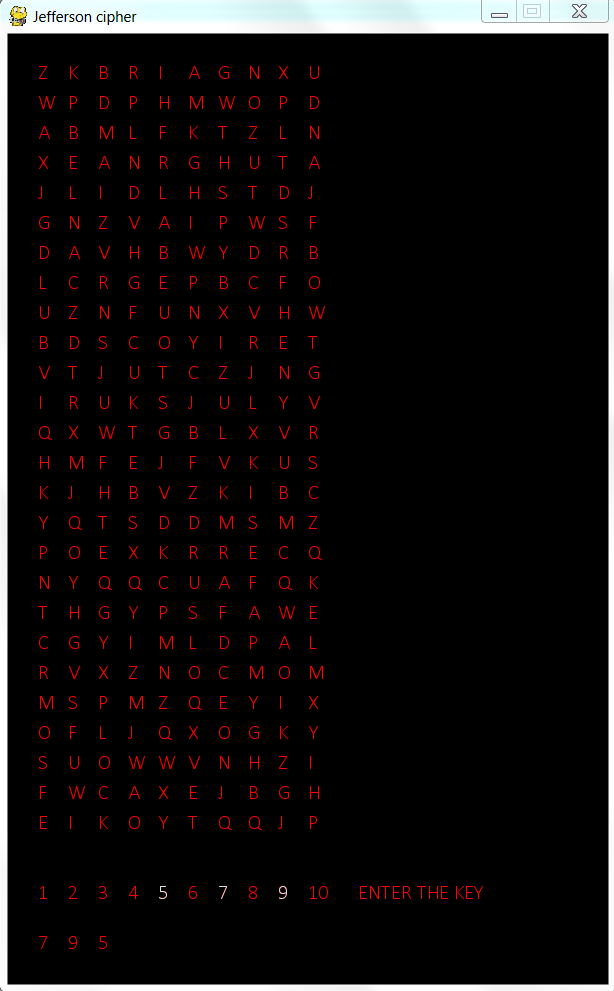
* Une procédure « displayCylinder(mySurface,cylinder,i) » où **mySurface** est la surface courante, **cylinder** un dictionnaire représentant un cylindre de Jefferson et **i** un numéro de disque. Elle affichera au “bon endroit“ sur la surface le **ième** disque du cylindre.
* Une procédure « displayCylinders(mySurface,cylinder) » où **mySurface** est la surface courante et **cylinder** un dictionnaire représentant un cylindre de Jefferson. Elle affichera sur la surface tous les disques du cylindre.
* Une fonction « enterKey(mySurface,n) » où où **mySurface** est la surface courante et **n** le nombre de disques du cylindre. Elle affichera en dessous du cylindre le numéro de chacun des disques. Elle attendra que l’utilisateur clique dans l’ordre qu’il souhaite sur chacun de ces numéros afin de constituer la clé. Dès qu’un numéro a été choisi il change de couleur et ne peut être sélectionné de nouveau. Sur une seconde ligne s’affiche progressivement la clé. Finalement la fonction retournera la clé qui est rappelons-le une liste de numéros.
* Une procédure « rotateCylinder(cylinder,i,up=True) » où **cylinder** est un dictionnaire représentant un cylindre de Jefferson, **i** un numéro de disque et **up** un booléen. Si **up** vaut **True** elle réalisera le décalage des lettres du ième disque “vers le haut“ et si **up** vaut **False** ce décalage se fera “vers le bas“. Cette procédure n’a aucun effet visuel sur la fenêtre graphique, la surface courante n’est d’ailleurs pas passée en paramètre. Elle modifie juste le dictionnaire représentant le cylindre.
* Une procédure « rotateCylinders(mySurface,cylinder) » où **mySurface** est la surface courante et **cylinder** un dictionnaire représentant un cylindre de Jefferson. L’utilisateur peut alors cliquer sur les flêches en dessous de chaque disque pour faire tourner ceux-ci dans un sens ou dans un autre. Ces rotations modifient la surface courante mais également le dictionnaire représentant le cylindre. Quand le texte qu’il souhaite chiffrer apparaît sur la ligne “CLEAR“ il clique alors sur “FINISH“ pour stopper le process. Le message chiffré apparaît alors sur la ligne “CIPHER“.
* Une procédure principale récupérant un cylindre dans un fichier texte (voir partie précédente), créant une surface de dimension adaptée au nombre de disques, puis utilisant les sous-programmes précédents pour permettre à l’utilisateur de saisir une clé et un message à chiffrer. Une fois chiffré le message est alors écrit dans un fichier texte.

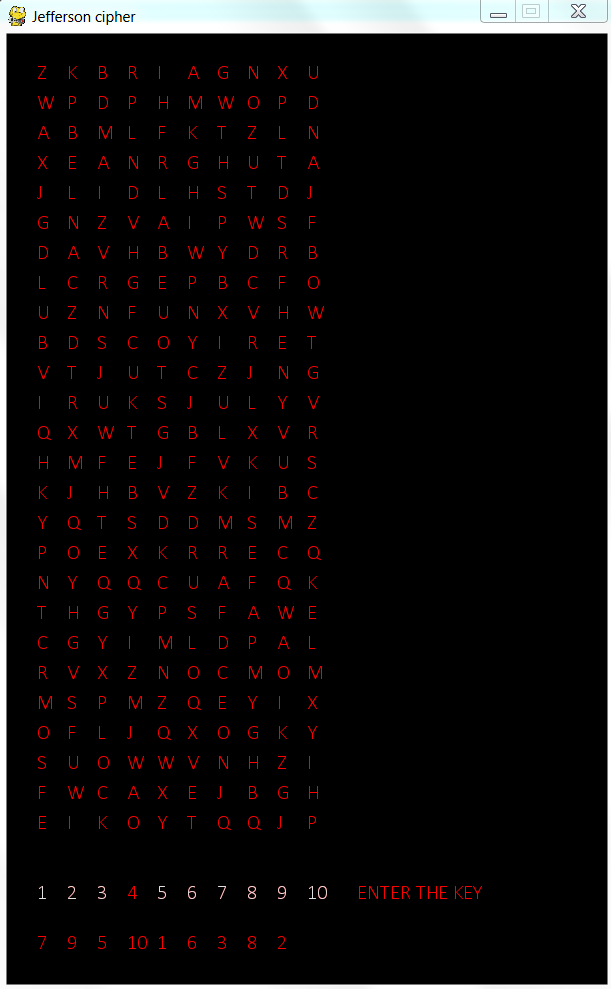
**Exemple de fonctionnement :**

* Affichage des disque provenant d’un fichier texte (le même fichier que dans la partie précédente) :

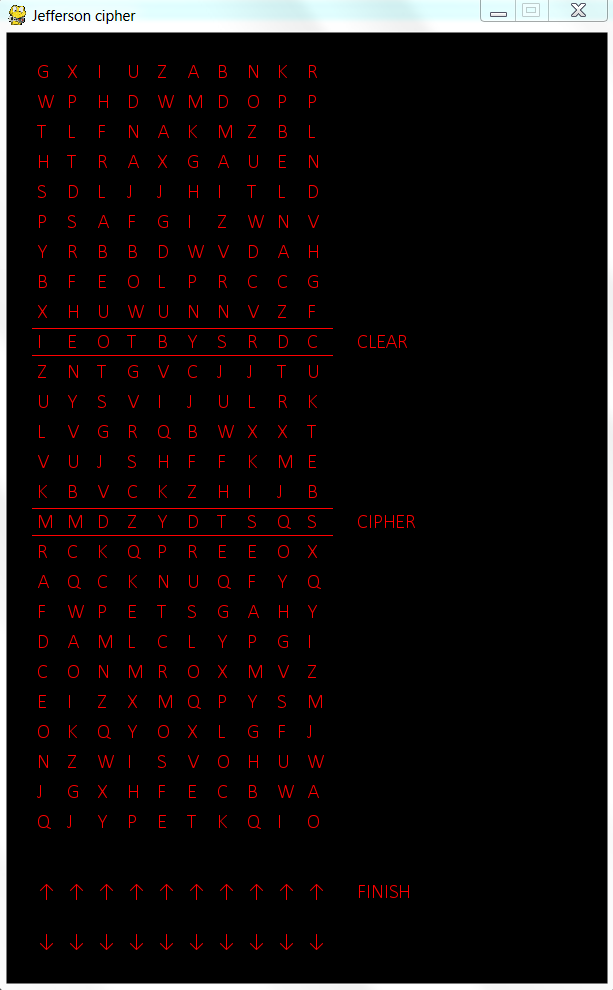


* Saisie d’une clé en cliquant sur les numéros des disques :

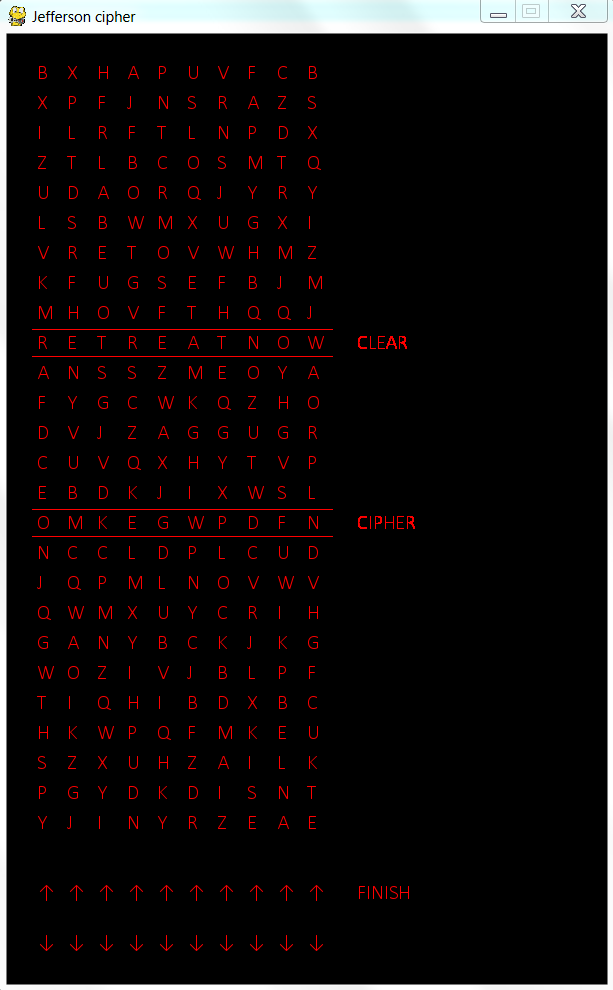




* Affichage des disques dans l’ordre de la clé :



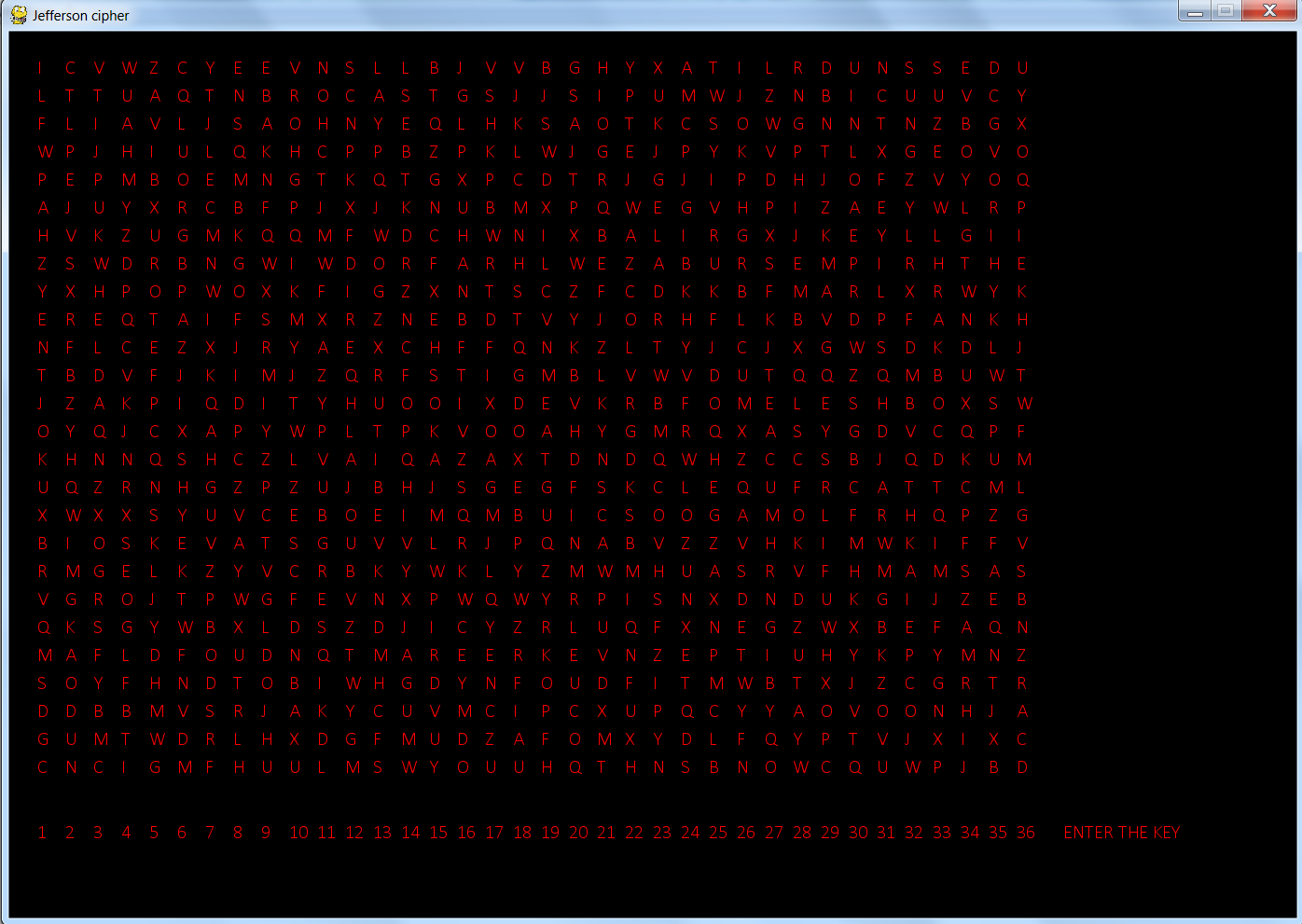
* Rotation des disques pour faire apparaître le message à chiffrer, et par conséquent le message chiffré :



* Ecriture du message chiffré dans un fichier texte :

express:Users:laurentgodefroy:Desktop:Image 8.png

Un exemple d’affichage avec un plus grand nombre de disques, ce qui montre le dimensionnement automatique de la surface :



# Bonus

Construire physiquement*, i.e*. avec des disques et un axe, un cylindre de Jefferson. De la qualité est attendue pour que ce bonus soit valorisé.

# Barème indicatif

Ce barème peut être amené à évoluer, il n’est donc qu’**indicatif**.

* Partie 2 : 15 points
* Partie 3 : 25 points
* Bonus : 10 points

Ce qui fait un total de 40 points, votre soutenance étant évaluée sur 20 points. La note totale sur 60 sera alors ramenée sur 20 par proportionnalité.