

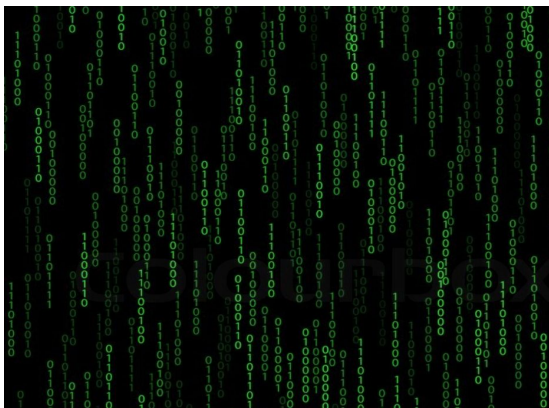
GOGOLE CRYPT



Spécialité ISN 2016-2017 : Informatique et Sciences du Numérique

Groupe : ABELANSKI Violette ; BUARD Elise ; VOISIN Pauline

Lien : <https://github.com/ecolelasource92/Gogole-crypt>



★ Sommaire :

- 1) Introduction p. 3
- 2) Présentation du projet p. 4-5
- 3) Mise en place et fonctionnement du projet p. 6-16
- 4) Démarche Collaborative et difficultés rencontrées p. 17-18
- 5) Bilan personnel et conclusion p. 19-25
- 6) Annexe (sur github sous forme de branche, une avec le programme de cryptage, une autre avec celui de décryptage et la dernière avec l'interface et le programme entier)

❖ I/ INTRODUCTION

Cette année de Terminale Scientifique 2016-2017, nous avons décidé de former un groupe de 3 personnes pour réaliser notre projet d'Informatique et Sciences du Numérique. Il se compose de Violette Abelanski, Elise Buard et de Pauline Voisin. Après concertation, nous avons eu l'idée de réaliser une plateforme de cryptage et de décryptage de messages.

- Contexte : Quelques temps après avoir entendu parler du C.E.S (Consumer Electronic Show) qui s'est tenu à Las Vegas pour sa session de 2017, nous nous sommes rendues compte que nous vivions dans un monde hyperconnecté dans lequel l'innovation, la créativité et les nouvelles technologies sont parties intégrantes. Malheureusement, à l'heure actuelle, les nouvelles technologies sont aussi à l'origine de nombreuses dérives : la surveillance internet en est un exemple frappant. Après un stage dans la police scientifique et des ateliers MPS, nous nous sommes interrogées aux modes de cryptage qui permettent de minimiser la grande surveillance accrue notamment par Google et les réseaux sociaux.
- Enjeu du Projet : À partir de ce constat, il s'agit de mettre en place une interface graphique à la façon de GOOGLE Traduction pour pouvoir traduire le langage courant en langage crypté et indéchiffrable pour une personne lambda. À cela s'ajoute une fonction décryptage qui permet l'action inverse. L'utilisateur rentre le texte qu'il souhaite coder, celui-ci est codé. Même chose pour l'utilisateur qui souhaite déchiffrer un message reçu sous forme de code. Cet interface ne comporte qu'une seule langue de traduction : le français. Notre code ne code que l'alphabet latin et les caractères spéciaux présent sur un clavier azerty et qwerty.
- But : Le but du projet est de créer un programme capable de crypter et de décrypter un message grâce à différentes méthodes de cryptage. Nous voulions utiliser une seule méthode de cryptage, se référant à une seule clé de code.. Nous voulions créer une interface simple et efficace.
- Problématique : Inspirées par les travaux d'Alan Turing, nous voulions nous confronter au monde complexe du codage. Nous voulions nous même expérimenter la création d'un code, si possible indéchiffrable, et voir ses possibles limites. Nous voulions comparer notre programme de codage aux méthodes déjà existantes. Nous nous sommes inspirées du code de César, du code de Vigenère, du chiffage par alphabet décalé.

❖ II/ Presentation du projet

Notre projet se nomme "GOOGLE CRYPT" en clin d'oeil à Google Traduction.

- Les Moyens mis en place :

- nos 3 ordinateurs portables
- l'espace de travail collaboratif GITHUB pour mettre en commun nos codes
- les modules Python Tutor, TextWrangler, Sublime Text et NotePad pour coder et créer l'interface graphique
- Google Drive pour échanger nos idées et avancer plus rapidement dans le projet
- Nous codons en PYTHON exclusivement, après avoir testé puis abandonné le code HTML

- Objectif initial :

- Pour que le message soit indécodable, l'idée est de changer de méthode de cryptage toutes les 5 lettres (puis cela sera 8 lettres finalement).
- chaque méthode de cryptage code pour deux caractères à la suite
- au début de notre projet nous avions des ébauches d'idées mais nous ne voulions codes que sur la base d'une seule méthode de cryptage en changeant toutes les lettres de conditions :
 - remplacer une lettre par la suivante dans l'alphabet
 - remplacer une lettre par 5 lettres qui la suivent dans l'alphabet
 - remplacer une lettre par son numéro $x2 + 3$
 - si le numéro de la lettre est inférieure ou égale à 13, il remplacer par 3 lettres avant, sinon par 7 lettres après dans l'alphabet.
 - si le numéro de la lettre est paire, remplacer la lettre par son numéro divisé par 2. Sinon la remplacer par son numéro $x3$
 - Chaque caractère espace sera codée par le caractère suivant l'espace dans le texte +1

Nous voulions nous inspirer de cette interface graphique de codage-décodage trouvée sur GITHUB :

Codes secrets (github, fiche signalétique)

Chiffre par subsitution avec alphabet décalé

Options

Que dois-je faire?

Texte avant transformation :

Clef : 3

Alphabet original : abcdefghijklmnopqrstuvwxyz

Alphabet décalé : xyzabcdefghijklmnopqrstuvw

Chiffrer

Déchiffrer

Echanger les textes

Texte après transformation :

Attaque par force brute :

Lancer

Arrêter

<

>

Attaque à texte clair connu :

Chercher

Chiffre par subsitution avec alphabet permuté

Options

Que dois-je faire?

Texte avant transformation :

Clef : 314

Alphabet original : abcdefghijklmnopqrstuvwxyz

Alphabet permuté : upqgshvlkmrtifyowxceznjabd

Chiffrer

Déchiffrer

Echanger les textes

❖ III/ Fonctionnement du Projet :

Nous voulions que notre programme de cryptage et décryptage soit à la fois simple et technique, en utilisant le plus possible les techniques et fonctions que nous avons appris depuis ce début d'année.

Après plusieurs séances, pour rendre notre travail plus professionnel et plus clair, nous avons décidé de mettre nos différentes méthodes de codage sous la forme de fonctions comme sur l'image ci-dessous :

```
def asci_lettres() :
```

Ainsi, il devenait plus simple de les “appeler” directement dans notre programme principal.

On commence tout d’abord à rentrer nos différentes variables et données de base :

```
texte = input()

nbtexte = len(texte)

alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
            "V", "W", "X", "Y", "Z", " ", ".", ",", ":", "!", "?", ";", "#", "(", ")", "'", "\"", "-", "1", "2", "3",
            "4", "5", "6", "7", "8", "9", "0", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
            "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

n = 0
cle = "CODE" # modifiable
nbcle = len(cle)
quotient = 2 // nbcle # penser a changer le 2 quand on veut faire prendre + de 2 lettres en vigenere = pour le reste
reste = 2 % nbcle
```

Le texte est à remplir par l'utilisateur. Cela est marqué par le terme “**input ()**”.

On rentre également un tableau qui est un alphabet un peu spécifique puisqu'il contient des lettres, des chiffres et des caractères spéciaux.

On rentre déjà les variables “**quotient**” et “**reste**” qui seront utiles pour la méthode de cryptage utilisant le code de Vigenère.

La variable **n**, représente une sorte de curseur pour déterminer quel caractère doit être codé pour quelle méthode, et va de deux lettres en deux lettres.

On incrémente **n** à 0, ce qui sera utile dans notre boucle “**While**”.

On rentre également les variables **nbtexte**, **clé** et **nbcle**.

On peut alors passer au traitement, c’est à dire aux fonctions et programmes de codage.

● Les Méthodes de cryptage :

1) ASCII lettre + 1 :

```
def ascii_lettres() :  
    for loop in range(2) :  
        newCaractere = chr(ord(lettres[loop]) + 1)  
        print(newCaractere, end="")
```

Par cette fonction, on souhaite transformer les deux premiers caractères du texte en nouvelles lettres ou caractères en passant par le tableau ASCII.

À-travers la boucle non conditionnelle **“for loop in range”**, on transforme le caractère. On emploie les fonctions **“ord”** et **“chr”** pour transformer le caractère en son chiffre complémentaire dans le tableau ASCII on fait + 1 puis on le convertit en lettre en se reportant toujours au tableau ASCII. Enfin, on choisit d’afficher le nouveau caractère sans retourner à la ligne avec la fonction **“end = “” ”**.

2) ASCII lettre + 8 :

```
def ascii_diminu() :  
    for loop in range(2) :  
        newCaractere = chr(ord(lettres[loop]) + 8)  
        print(newCaractere, end="")
```

Même chose pour ce programme qui code 2 nouveaux caractères avec un décalage de 8 rangs en plus (+8) dans le tableau ASCII.

3) Notre Alphabet :

```
def notre_alphabet() :  
    for loop in range(2) :  
        for eachPos in range(50) :  
            if lettres[loop].upper() == alphabet[eachPos]:  
                print(alphabet[eachPos + 5], end="")
```

Ici, on a employé deux boucles **“for loop in range”** : une pour coder 2 caractères et une pour parcourir notre tableau alphabétique spécial que nous avons définie au début de notre programme sous forme de tableau.

La fonction **“upper”** sert à transformer un caractère minuscule en majuscule. Cela à condition que le caractère soit similaire à un des caractères de notre tableau grâce à la fonction si **“if”**, alors on peut afficher le nouveau caractère, crypté par une lettre de 5 rang supérieur à celle initiale.

4) Le code de Vigenère

```
def vige_nere() :  
  
    for loopi in range(reste):  
  
        for eatchpos in range(50):  
  
            if cle[loopi] == alphabet[eatchpos]:  
                nvlc = eatchpos  
  
                for eachPos in range(50):  
  
                    if lettres[loopi + nbcle * quotient].upper() == alphabet[eachPos]:  
                        print(alphabet[eachPos + nvlc], end="")
```

Nous avons employé une méthode de cryptage déjà connue, celle du Code de Vigenère. La clé de notre méthode de cryptage de vigenère est le mot **“CODE”**. Cette clé va venir additionner sa place dans notre alphabet à celle de chaque lettre de notre texte que l’on transforme. Pour cela nous utiliserons successivement deux boucles la première pour faire tourner la clé et la faire reconnaître dans notre alphabet défini sous forme de tableau au début de notre programme. Puis la deuxième boucle pour chercher dans notre alphabet la lettre que l’on souhaite transformer + la lettre de la clé fois le quotient. On additionne donc après le lettre de notre texte que l’on souhaite coder à celle de la clé lui correspondant.

5) Pour un texte impair :

```
def texte_impair() :  
    dernierelettre = chr(ord(texte[nbtexte - 1]) + 6)  
    print(dernierelettre, end="")
```

Pour régler un des problèmes développé dans la partie personnelle d’Elise , nous avons créé cette méthode de cryptage en plus. Elle intervient lorsque le texte à coder est impair. Ainsi le dernier caractère par cette méthode spécifique va être codé et empêche que l’une des méthodes tourne dans le vide. Même démarche que précédemment. Le caractère codé sort en un nouveau caractère de 6 rangs supérieur. Quand le texte est pair il fait les méthodes de cryptage vu au dessus sans passer par cette fonction de décryptage.

6) Le Programme principal :

```
while n < (nbtexte - 1):

    lettres = texte[n:n + 2]

    if n % 8 == 0 or n % 8 == 1:
        asci_lettres()

    elif n % 8 == 2 or n % 8 == 3:
        asci_diminu()

    elif n % 8 == 4 or n % 8 == 5:
        notre_alphabet()

    elif n % 8 == 6 or n % 8 == 7:
        vige_nere()

    n = n + 2

if nbtexte % 2 == 1:
    texte_impaire()
```

Nous avons décidé de mettre notre programme principal sous forme de boucle conditionnelle “while” et de boucles “if”. Ainsi, tant que la variable n qui nous permet de crypter un à un les caractères de notre texte est inférieure à la longueur du texte (- 1) car le premier caractère est compté en tant que 0), chaque méthode de cryptage va coder deux caractères côte à côte ; grâce à “ *lettres = texte[n:n+2]* ” qui lui commande d’aller de deux en deux à chaque tour. De plus, en faisant la division euclidienne de n par 8, nous pouvons appeler les différentes méthodes de cryptage suivant la valeur de n. En effet n prend + 2 a chaque fin de méthode de cryptage, de cette façon le code passe aux deux lettres suivantes du texte à coder et ne code pas avec chaque méthode de cryptage les deux mêmes lettres.

De même, si le reste de la division euclidienne de la longueur du texte par 2 vaut 1, cela signifie que le texte est impair. On appelle donc la fonction “texte impair”.

• Les méthodes de décryptage :

1) ASCII lettres - 1 et ASCII lettres - 8 :

```
def asci_lettres() :  
  
    for loop in range(2) :  
  
        newCaractere = chr(ord(lettres[loop]) - 1)  
        print(newCaractere, end="")
```

```
def asci_diminue() :  
  
    for loop in range(2) :  
  
        newCaractere = chr(ord(lettres[loop]) - 8)  
        print(newCaractere, end="")
```

Les deux premières méthodes de décryptages sont sur le même principe que le cryptage sauf que cette fois-ci nous allons chercher à retrouver le texte original à partir du texte crypté. On fait donc l'inverse des opérations du cryptage ; en l'occurrence quand nous avons un + 1 en cryptage nous faisons un -1 en décryptage. De même pour la seconde méthode de cryptage utilisant un - 8 car nous avons fait un + 8 en cryptage.

2) Notre Alphabet et Vigenère :

```
def notre_alphabet() :  
  
    for loop in range(2) :  
  
        for eachPos in range(50) :  
  
            if lettres[loop].upper() == alphabet[eachPos]:  
                print(alphabet[eachPos - 5].lower(), end="")  
  
def vige_nere() :  
  
    for loopi in range(reste):  
        for eatchpos in range(50):  
            if cle[loopi] == alphabet[eatchpos]:  
                nvlcle = eatchpos  
                for eachPos in range(50):  
                    if lettres[loopi - nbcle * quotient] == alphabet[eachPos]:  
                        print(alphabet[eachPos - nvlcle].lower(), end="")
```

Même chose pour notre alphabet. Quand nous arrivions à identifier le caractère à coder grâce à notre alphabet situé en haut du code, nous faisons + 5 donc maintenant pour le décrypter il faut faire - 5. Il en va de même avec Vigenère ou il fallait faire [eachPos + nvlcle], il faut maintenant faire [eachPos - nvlcle].

3) Pour un texte impaire :

```
def texte_impair() :  
  
    dernierelettre = chr(ord(texte[nbtexte - 1]) - 6)  
    print(dernierelettre.lower(), end="")
```

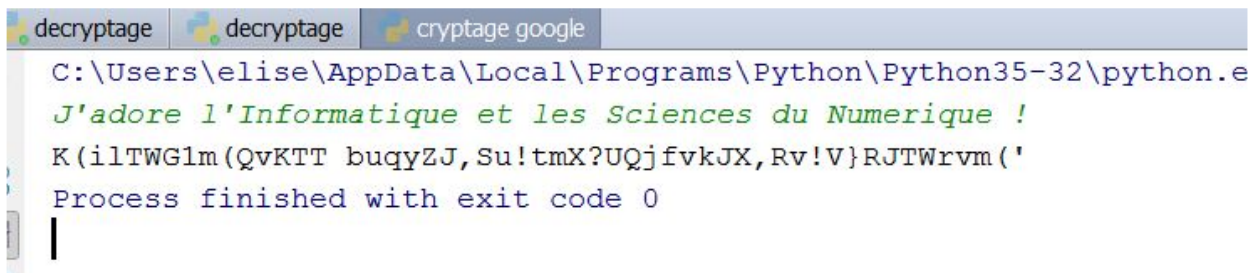
Au lieu d'ajouter 6 rangs au caractère, il faut maintenant lui enlever 6 rangs.

4) Programme principal :

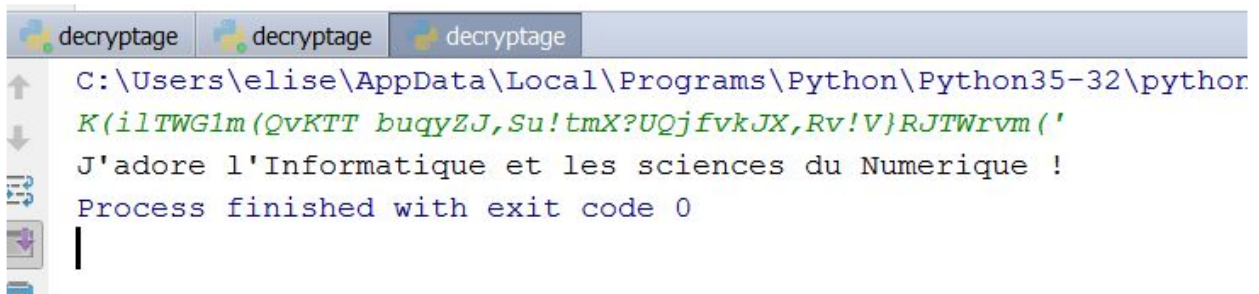
Le programme principal lui ne change pas il va donc appeler les différentes fonctions correspondants aux méthodes de décryptage.

5) Rendu final sous forme de code brut :

Voici le cryptage :



Et voici le décryptage :



Interface graphique

Notre interface graphique, codée en python grâce à **TKinter**, se divise en deux parties : la parties physique (la forme), et la partie fonctionnelle de codage (le fond).

- La forme

Tout commence par la mise en place d'une fenêtre, qui s'affiche après exécution du code de l'interface.

```
from tkinter import *
fenetre = Tk()
fenetre.geometry("1000x700+0+0")
fenetre.title("Gogole crypt")
fenetre["bg"] = "turquoise3"
```

On débute par importer tkinter. Par la suite, on crée une fenêtre, en choisissant ses dimensions (1000x700) ainsi que l'endroit où elle s'ouvrira sur l'écran (0+0). Le titre de cette fenêtre sera "Gogole crypt", est l'arrière plan sera turquoise.

Par la suite, nous créons les différents textes, entrées et boutons qui apparaîtront sur cette fenêtre.

```
texte1 = Label(fenetre, text="Gogole crypt").pack()
texte2 = Label(fenetre, text="Testez notre nouvelle methode de cryptage revolutionnaire et presque indecryptable...").pack()
texte3 = Label(fenetre, text="Texte à crypter :").place(x="400", y="100")
entree1 = Entry(fenetre, textvariable=texteacrypter).place(x="400", y="150")
label = Label(fenetre)
label["bg"] = "turquoise3"
label["fg"] = "black"
label.place(x="400", y="250")
bouton1 = Button(fenetre, command=crypte)
bouton1["text"] = "Crypter"
bouton1.place(x="400", y="200")
texte4 = Label(fenetre, text="Texte à décrypter :").place(x="400", y="400")
entree2 = Entry(fenetre, textvariable=texteadecrypter).place(x="400", y="450")
bouton2 = Button(fenetre, command=decrypte)
bouton2["text"] = "Décrypter"
bouton2.place(x="400", y="500")
lebel = Label(fenetre)
lebel["bg"] = "turquoise3"
lebel["fg"] = "black"
lebel.place(x="400", y="550")
texte5 = Label(fenetre, text="Attention, les accents ne sont pas lus par ce programme").pack()
```

Nous créons d'abord un premier label (texte1) pour le titre "Gogole crypt". Avec le mot "pack", ce titre sera centré et disposé en haut de la fenêtre.

On crée un deuxième label (texte2) présentant notre méthode de cryptage.

Par la suite, nous avons créé l'emplacement d'entrée du texte à crypter. Les indications x et y donnent l'emplacement de ce label sur la fenêtre.

En dessous du texte "texte à crypter" apparaît une zone d'entrée appelée entrée1. Le texte que l'utilisateur entrera ici s'appellera alors "texteacrypter". On définit "texteacrypter" grâce à "stringvar" :

```
| texteacrypter = StringVar()
```

Le but sera alors de saisir cette entrée "texteacrypter" afin de l'utiliser dans notre code en tant qu' input pour le crypter. Pour cela, nous créons un label à fond turquoise et à écriture blanche qui n'apparaîtra qu'après exécution du code.

Le code s'exécute lorsqu'on appuie sur le bouton 1 qui a pour fonction "crypt", c'est à dire la fonction de cryptage, et sur lequel apparaît un texte : "crypter".

Après cela, nous créons l'emplacement d'entrée du texte à décrypter. Nous commençons par une zone de texte (texte4) "texte à décrypter", puis une entrée (entrée2). L'entrée 2 aura pour nom "texteadecrypter". On définit "texteadecrypter" grâce à "stringvar":

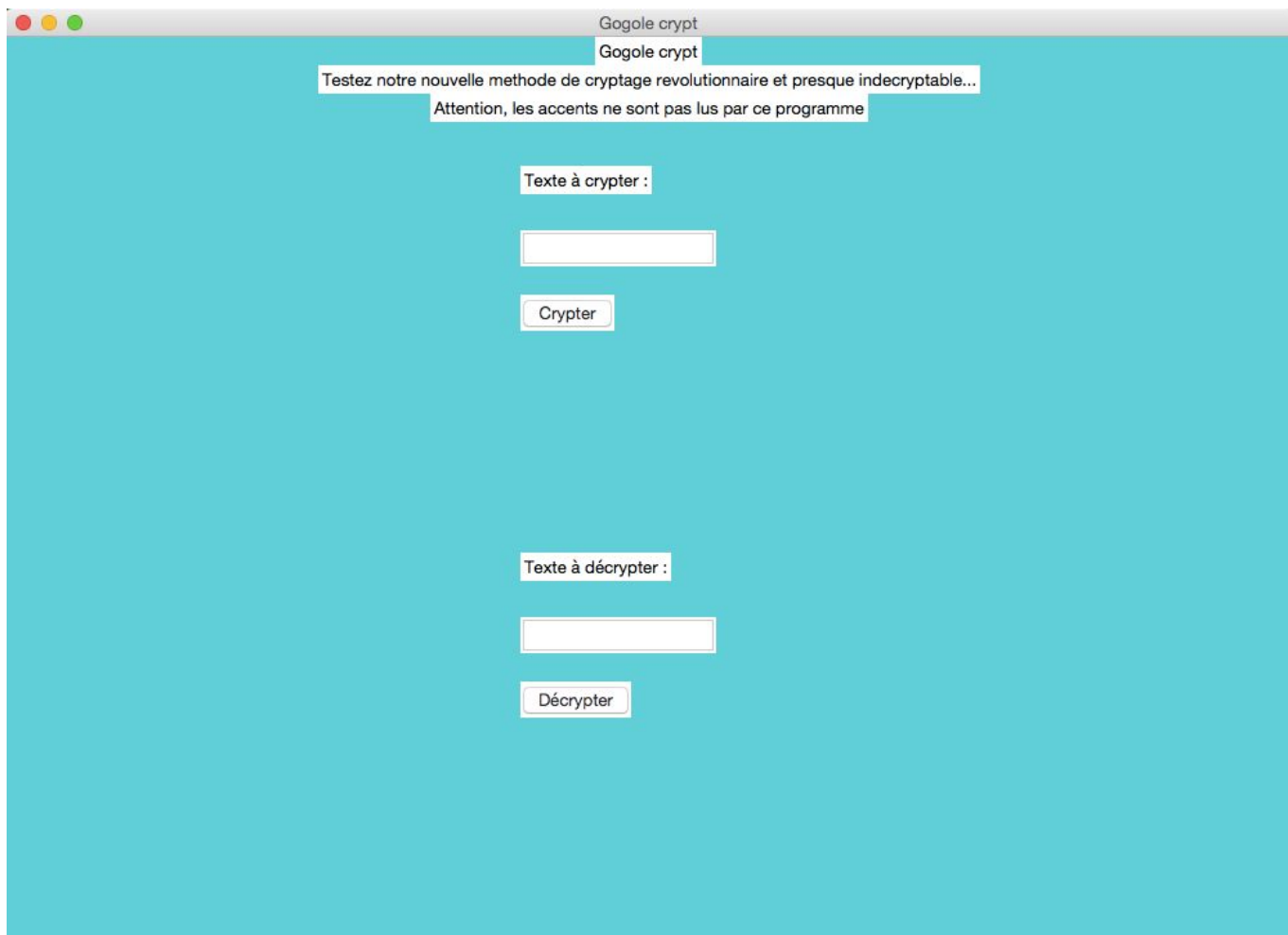
```
texteadecrypter = StringVar()
```

On crée alors un second bouton (bouton2) ayant pour fonction "decrypte", c'est à dire la fonction de décryptage.

Un second label est alors créé, appelé "lebel", qui lui aussi ne s'affichera qu'après exécution du code.

La dernière zone de texte (texte5) est simplement une indication faisant remarquer que les accents ne sont pas lus par le code. Celle ci sera centrée et placée en haut de la fenêtre.

Voici le rendu final de la forme de l'interface :



L'étape suivante de l'interface est donc d'y ajouter le fond.

- Le fond

La partie fonctionnelle de l'interface est en fait notre code, expliqué plus haut, avec cependant quelques modifications. Le code se trouve dans une fonction appelée "crypt" pour le cryptage et "decrypte" pour le décryptage.

```
def crypte() : def decrypte():
```

Le code en lui même nécessite des modifications. Il faut d'abord modifier le nom de l'entrée faite par l'utilisateur. On ne parlera plus de "input()" mais de "texteacrypter.get()" ou de "texteadecrypter.get()" afin que le code utilise les entrées présentent sur la fenêtre. Afin que la sortie de notre code s'affiche dans les différents label nommés "label" et "lebel", nous avons créé un tableau, que l'on a nommé "tableau" :

```
tableau = [0]*nbtexte
for i in range(nbtexte) :
    tableau[i] = texte[i]
```

Ce tableau contiendra le mot ou la phrase entrée par l'utilisateur, avec un caractère par case. Par la suite, à la fin de chaque fonction de cryptage ou de décryptage, on modifie la lettre initiale par la nouvelle lettre codée, puis on avance d'une case pour que la prochaine lettre modifiée soit la suivante. Pour cela on crée une variable nommée "a", qui prend a+1 à chaque fois qu'une lettre a été cryptée.

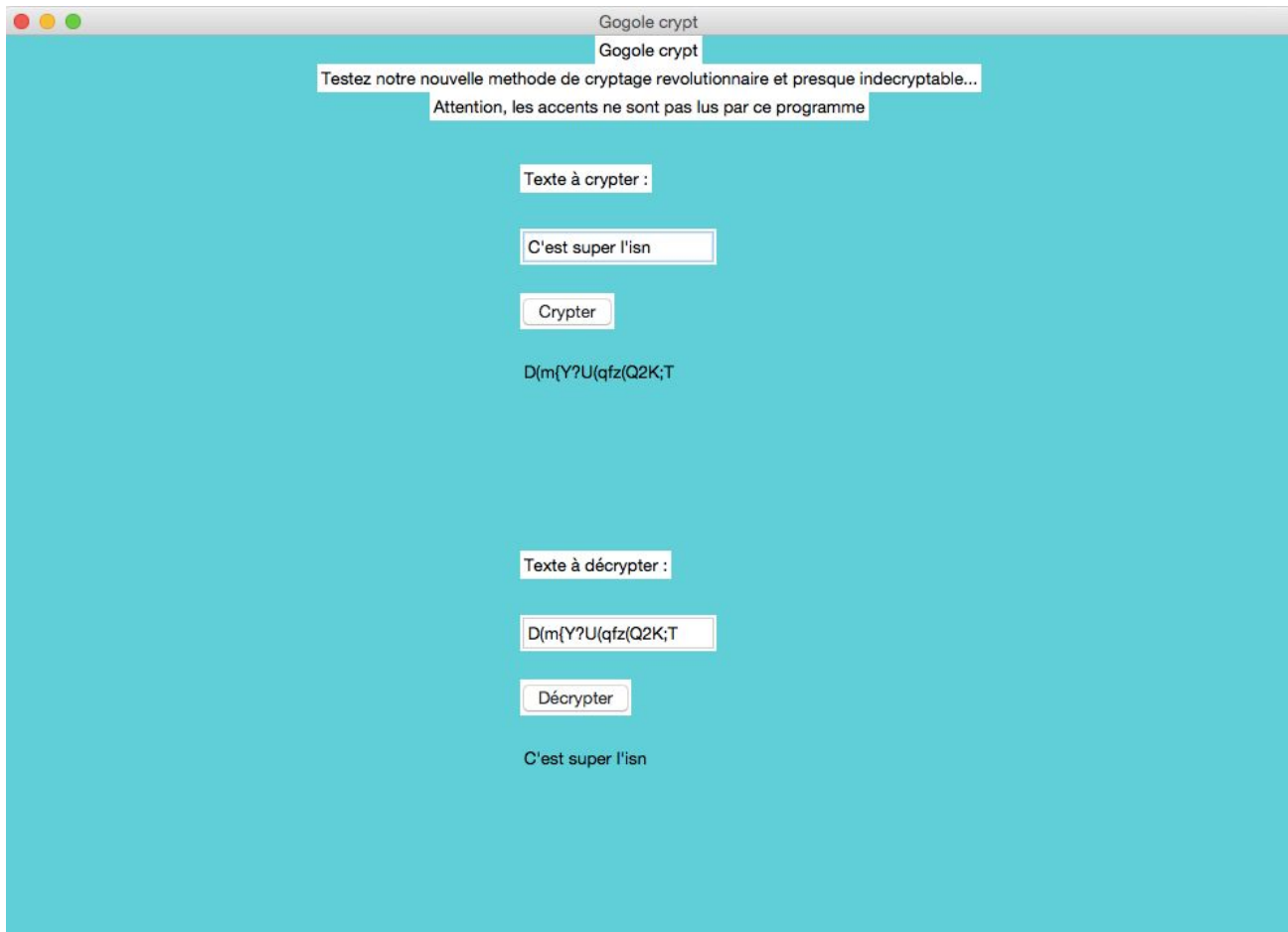
```
tableau[a] = newCaractere
if a < (nbtexte-1) :
    a = a + 1
```

De manière à ce que l'addition a+1 ne continue pas lorsque le mot est terminé, nous avons créé une condition : cette addition se fait seulement si "a" est plus petit que le nombre de caractère entré - 1 (car le premier caractère vaut 0).

A la fin de code, il nous suffira alors de regrouper les éléments du tableau et de les faire apparaître à l'emplacement "label" pour le cryptage et "lebel" pour le décryptage.

```
mot = "".join(tableau)    mot = "".join(tableau)
label["text"] = mot       lebel["text"] = mot
```


Voici la preuve du bon fonctionnement de l'interface graphique : Nous avons testé la phrase "C'est super l'isn".



❖ IV/ Repartition des Taches

Tâche	Nom
Création d'un cahier des charges sur Github : repository. Y ajouter notre cahier des charges déjà rédigé ultérieurement.	Tout le monde
Faire des recherches développées sur les différents types de cryptage.	Pauline
S'informer sur la création d'un site internet et ses difficultés.	Elise
Recherche sur le tableau ASCII	Violette
Création du code ASCII + 1 lettre	Elise
Création du code ASCII + 8 lettres	Pauline
Création du code Notre Alphabet + 5	Violette
Création du Code de Vigenère	Elise
Lier les codes en un seul	Elise
Decryptage	Elise et Pauline
Interface Graphique en Python	Violette
Légender le cryptage / decryptage sur Github	Pauline

- Difficultés et changements dans notre programme de cryptage et decryptage :

Nous avons choisi d'évoquer dans notre partie personnelle chacune les problèmes que nous avons rencontrés au fur et à mesure du code. Dans chaque parties personnelles chacune dira donc à quelles ont été les problèmes qu'elle a rencontrer.

- **Aller plus loin dans la démarche : ce que nous aurions pu améliorer**

Si nous avions eu plus de temps, c'est à dire si l'échéance de début juin avait été repoussée, nous aurions pu améliorer certains problèmes et aller encore plus loin dans notre démarche de cryptage et de décryptage :

- améliorer les méthodes de cryptage : aujourd'hui, bien que nous avons utilisé différentes méthodes de cryptages pour rendre le décryptage plus complexe, nous nous sommes arrêtées à des décalages de caractères. Nous aurions pu faire des permutations / substitutions de caractères par exemples.
- Aujourd'hui il est de plus en plus facile de "craquer" un code. Avec différents essais par ordinateur et par la méthode de la fréquence d'apparition des caractères, il est possible de décrypter notre code. Il ne semble malheureusement pas infaillible.
- Nous aurions pu vaincre les difficultés des caractères spéciaux (accents) pour que notre programmes soit encore meilleur.
- Nous aurions pu tenter de coder d'autres méthodes de cryptage déjà existantes : le chiffre de César, Enigma, le chiffre du CHE...etc
- Nous aurions pu ajouter d'autres caractères spéciaux pour crypter d'autres langues comme l'arabe ou le chinois.

Conclusion : Notre travail de groupe a été une réussite, nous nous sommes répartis équitablement les tâches, et ce travail a été effectué avec efficacité et dans la bonne humeur. Chacune de nous trois a su trouvé se dans quoi elle était meilleure et ce qu'elle a pu apporter au projet. Le fait de travailler en équipe nous a motivé et nous a fait découvrir de nouvelle façon de travailler. En effet il n'est pas facile de travailler en groupe chacun chez soit quand il s'agit d'informatique mais grâce au drive de google cela a été plus facile.

Lien contenant notre projet, celui-ci comporte une branche contenant le cryptage brut, une branche contenant le décryptage brut, et une branche contenant le code de l'interface.

→ <https://github.com/ecolelasource92/Gogole-crypt>

Partie Personnelle d'Elise Buard :

Cette idée de projet d'ISN de cryptage est décryptage découle du film *Imitation game* de Morten Tyldum. Ce film nous a inspiré pour ce projet d'ISN car dans ce film nous sommes plongé en plein dans le monde de l'informatique, du codage et de l'algorithmique. Il nous a paru intéressant de trouver un moyen de coder des conversations et de pouvoir les décoder. De plus j'ai lu "*Forteresse Digitale*" de Dan Brown, sur la NSA et sa section s'occupant de craquer les codes pour pouvoir espionner les messages venant du monde entier et démanteler les actions terroristes. Ce livre m'a donné plus qu'envie de moi aussi réussir à mettre en place un algorithme codant des messages pour ne pas être lu du grand public.

Je me suis personnellement occupé du cryptage et du décryptage :

1. cryptage

Premièrement nous avons créé les méthodes de cryptage séparément les unes des autres :

- La première et la deuxième utilisant le tableau ASCII, il a été facile de les mettre en place une fois que nous avons compris le fonctionnement de ce tableau. Ces méthodes transforment les caractères entrés par l'utilisateur par de nouvelles lettres ou caractères. Pour la première méthode en additionnant + 1 au rang du caractère entré par rapport au tableau ASCII et pour la deuxième méthode en additionnant + 8 au rang du caractère entré.

```
def asci_lettres() :  
  
    for loop in range(2) :  
  
        newCaractere = chr(ord(lettres[loop]) + 1)  
        print(newCaractere, end="")
```

Méthode de cryptage 1

→ Nous avons modifié une méthode de codage, en effet notre deuxième méthode utilisant le tableau ASCII devait sortir en cryptage des chiffres grâce à "ord" utilisé seul ("ord" fait correspondre une lettre à son chiffre dans le tableau ASCII). Mais quand nous

voulions décrypter il ne comprenait pas quand les chiffres étaient double (par exemple pour un “j” le cryptage était “45”). Le code de décryptage prenait les chiffres un par un et sortait donc plusieurs lettres pour un chiffre. Ne trouvant pas de solutions qui ne soit pas très complexe nous avons changer notre méthode de cryptage et l’avons fait coder en lettre comme la première mais avec une condition différente en ajoutant “chr” qui ramène le nombre correspondant dans le tableau ASCII à sa lettre.

```
def asci_diminu() :  
  
    for loop in range(2):  
  
        newCaractere = chr(ord(lettres[loop]) + 8)  
        print(newCaractere, end="")
```

Méthode de cryptage 2

- La troisième méthode de cryptage se basait sur un alphabet que nous avons choisit et définit par nous même et en nous aidant de France IOI il nous a été possible de le mettre en place sous forme de tableau. Cette méthode de cryptage part du même principe que celles utilisant le tableau ASCII mais tourne sur notre alphabet et fait + 5 au rang dans notre alphabet de la lettre entré.

```
def notre_alphabet() :  
    for loop in range(2):  
  
        for eachPos in range(50):  
  
            if lettres[loop].upper() == alphabet[eachPos]:  
                print(alphabet[eachPos + 5], end="")
```

Méthode de cryptage 3

→ Le problème principal rencontré pour cette méthode de cryptage a été de faire comprendre que nous voulons nous référer à notre propre tableau et cela a finalement été possible grâce aux “alphabet[eachPos]”. Le code va alors chercher dans les 50 lettres de notre alphabet la lettre correspondant à celle qui li est présenté par l'utilisateur du code.

Le problème d'utiliser notre propre alphabet était aussi qu'il y avait des caractères spéciaux à prendre en compte (nous avons mis ceux que l'on trouve sur un clavier lambda), et que les accents, notre code ne les prenait pas en compte et aller jusqu'à les oublier et à ne pas crypter de lettres à la place(plus développé dans la partie personnelle de Pauline). Nous avons donc banni les accents et affichés sur notre interface graphique de ne pas en mettre.

```
texte = input()
nbtexte = len(texte)

alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
            "V", "W", "X", "Y", "Z", " ", ".", ",", ":", "!", "?", ";", "#", "(", ")", "'", "\"", "-", "1", "2", "3",
            "4", "5", "6", "7", "8", "9", "0", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
            "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

n = 0
cle = "CODE" # modifiable
nbcle = len(cle)
quotient = 2 // nbcle # penser a changer le 2 quand on veut faire prendre + de 2 lettres en vigenere = pour le reste
reste = 2 % nbcle
```

- La dernière méthode de cryptage que nous avons utilisé et le fameux code de Vigenère, c'est celui dont je me suis occupé personnellement. Nous avons eu pas mal de problèmes sur le code Vigenère, en effet celui ci fonctionnant avec une clé il a été compliqué de lier une boucle clé + une boucle alphabet + une boucle générale. Cette méthode de cryptage est celle qui a mis le plus de temps à être mise en place. Surtout il fallait comprendre le fonctionnement de cette méthode et à mettre en place avec un algorithme. Pour crypter cette méthode additionne le rang dans notre alphabet de la lettre à crypter avec celui de la lettre de la clé qui lui correspond. Par exemple pour le premier caractère à crypter qui passe dans cette fonction on va additionner son rang à celui de la première lettre de la clé. Puis pour le deuxième caractère à crypter on va additionner son rang à celui de la deuxième lettre de la clé etc.. Notre clé ne comportant que 4 lettres "CODE" nous utilisons une fonction boucle pour la faire boucler.

```
def vige_nere() :

    for loopi in range(reste):

        for eatchpos in range(50):

            if cle[loopi] == alphabet[eatchpos]:
                nvlcle = eatchpos

            for eachPos in range(50):

                if lettres[loopi + nbcle * quotient].upper() == alphabet[eachPos]:
                    print(alphabet[eachPos + nvlcle], end="")
```

Méthode de cryptage 4

→ La difficulté qui nous a fait rester pendant minimum deux semaines sur cette méthode est le principe de quotient et reste. En effet sans le quotient ou le reste ce code ne tournait pas et nous ne trouvions pas l'équivalent dans l'alphabet (pour cette méthode aussi nous utilisons notre propre alphabet).

```
cle = "CODE" # modifiable

nbcle = len(cle)

quotient = 2 // nbcle #

reste = 2 % nbcle
```

2) Lier toutes les méthodes en un seul programme

Pour cela je me suis chargé de mettre toutes les méthodes de cryptage sous forme de fonctions grâce à la fonction "def" qui mise en début de chaque méthode permet de l'appeler autant de fois que l'on veut par la suite rendant le code plus lisible. Ex :

```
def asci_lettres() :

    for loop in range(2) :

        newCaractere = chr(ord(lettres[loop]) + 1)
        print(newCaractere, end="")
```

Le programme principal est présenté sous la forme suivante :

```

while n < (nbtexte - 1):

    lettres = texte[n:n + 2]

    if n % 8 == 0 or n % 8 == 1:
        |   asci_lettres()

    elif n % 8 == 2 or n % 8 == 3:
        |   asci_diminu()

    elif n % 8 == 4 or n % 8 == 5:
        |   notre_alphabet()

    elif n % 8 == 6 or n % 8 == 7:
        |   vige_nere()

    n = n + 2

if nbtexte % 2 == 1:
    texte_impair()

```

Chaque méthode de cryptage vu précédemment est appelée à tour de rôle et pour leur donner un ordre bien défini pour que cela soit facile à décrypter j'ai utilisé une variable n. Définit comme n = 0 au début du code et qui prend + 2 a chaque fois qu'une méthode a coder deux lettres.

Pourquoi deux lettres en même temps pour chaque méthode ?

C'est un choix tout à fait arbitraire, mais cela aurait pu être 3 ou 1 , nous voulions surtout qu'avec des mots court le code ne passe pas que dans une méthode mais dans plusieurs d'où le choix de 2 lettres en même temps.

Le code continue à crypter tant que n est inférieur au nombre de caractères du texte - 1 (-1 car au premier caractère est assigné le chiffre 0) car dès que n dépasse le nombre de caractères du texte - 1 cela veut dire qu'il n'y a plus de caractères à coder.

Le " lettres = texte[n:n+2] sert à que ce ne soit pas toujours les deux mêmes lettres du texte qui soit codés en boucle par toutes les méthodes de cryptage mais que cela saute de deux lettres en deux lettres. nb : on peut remarquer que ce " lettre" est utilisé dans chaque méthode de cryptage pour appeler les lettres a cryptées.

→ Mais le choix de coder pour chaque méthode des lettres par paire n'a pas été sans conséquences, en effet parfois, le programme ne tournait plus, aucun caractères ne sortaient et les modules sur internet indiquaient une erreur. C'était parce que la longueur du texte était impaire. Ainsi le code ne comprenait pas pourquoi à la fin du

texte il n'y avait qu'un seul caractère et non pas 2 à crypté. Il bugait donc au milieu d'une méthode de cryptage. Il a donc fallu à chaque fois vérifier que le texte soit impaire en demandant si le reste de la division est égal à 1. Dans ce cas là le caractère codé sort en un nouveau caractère de 6 rangs supérieur.

3) Décryptage

Le décryptage a été l'étape la plus simple du code il a suffi en effet de faire exactement pareil que le cryptage en transformant seulement les additions et soustractions. Ainsi quand le programme voit un caractère à décrypter il fait une soustraction du chiffre correspondant à sa méthode de cryptage et on retrouve donc le message entré au départ.

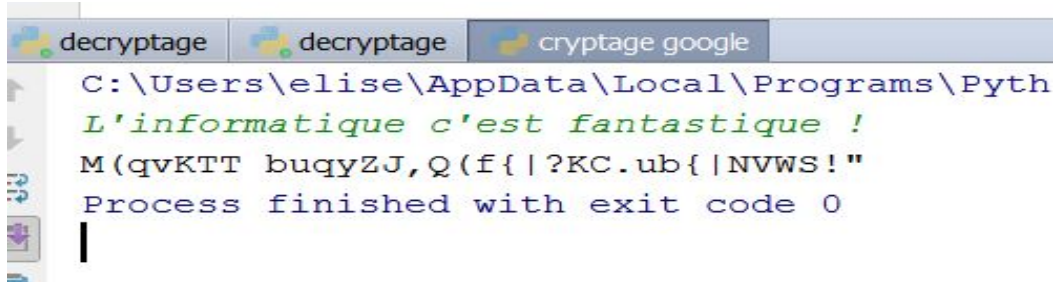
Exemple avec la méthode 1 ou quand au cryptage on faisait + 1 maintenant en décryptage on fait - 1 :

```
def asci_lettres() :  
    for loop in range(2) :  
        newCaractere = chr(ord(lettres[loop]) - 1)  
        print(newCaractere, end="")  
  
def asci_diminue() :  
    for loop in range(2) :  
        newCaractere = chr(ord(lettres[loop]) - 8)  
        print(newCaractere, end="")
```


4) Résultats

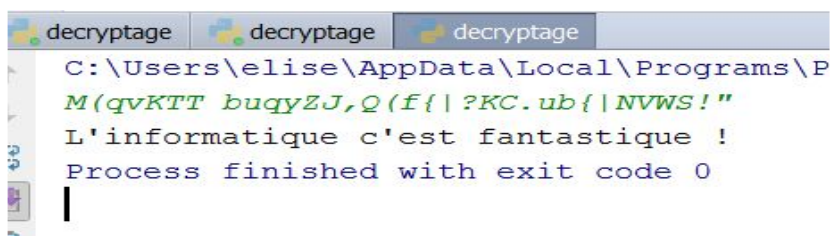
Voici la preuve du bon fonctionnement du code brut :

Voici le cryptage :



```
C:\Users\elise\AppData\Local\Programs\Python\Python39-64\Scripts\  
python cryptage google  
L'informatique c'est fantastique !  
M(qvKTT buqyZJ,Q(f{|?KC.ub{|NVWS!"  
Process finished with exit code 0
```

Et voici le décryptage :



```
C:\Users\elise\AppData\Local\Programs\Python\Python39-64\Scripts\  
python decryptage  
L'informatique c'est fantastique !  
M(qvKTT buqyZJ,Q(f{|?KC.ub{|NVWS!"  
Process finished with exit code 0
```

La démonstration du fonctionnement de ce programme avec le revêtement de l'interface est dans la partie de ma camarade Violette.

Pour conclure je dirais que ce projet était très innovant pour moi, il m'a appris beaucoup sur comment coder et comment appliquer concrètement les notions apprises dans France IOI. De plus il nous a permis d'approfondir toutes nos notions et en particulier les boucles et les tableaux. Nous avons conscience que notre algorithme est amateur et facile à craquer pour quelqu'un d'expérimenté mais nous sommes fière de notre rendu et du chemin que nous avons parcouru en 6 mois. En effet cela n'a pas toujours été facile et nous nous sommes parfois retrouvées dans des impasses heureusement résolues avec l'aide des gens de notre classe ou de notre professeur. De plus notre bonne cohésion de groupe nous a permis de produire en plus des algorithmes de cryptage et de décryptage que nous espérions, une interface graphique ! Chacun a trouvé sa place dans notre groupe de trois, en effet Violette s'occupait de l'interface graphique et de certaines recherches ; Pauline de faire du cryptage, beaucoup de recherches, et de se pencher sur les problèmes rencontrés ; et moi plus de créer le code brut en lui même. Ce fut un très beau projet et je j'ai donc la chance d'emporter avec moi de bagage informatique pour mes années dans le supérieur donc nous pouvons vraiment dire que l'informatique c'est fantastique !