

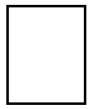
GOGOLE CRYPT

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
- 3) Mise en place et fonctionnement du projet p. 6
- 4) Démarche Collaborative et difficultés rencontrées p. 17
- 5) Bilan personnel et conclusion p. 19
- 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)

v 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 chiffrement par alphabet décalé.

v 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 :
 - i remplacer une lettre par la suivante dans l'alphabet
 - i remplacer une lettre par 5 lettres qui la suivent dans l'alphabet
 - i remplacer une lettre par son numéro $x2 + 3$
 - i 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.
 - i si le numéro de la lettre est paire, remplacer la lettre par son numéro divisé par 2. Sinon la remplacer par son numéro $x3$
 - i 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)

Options

Que dois-je faire?

Chiffre par substitution avec alphabet décalé

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

Options

Que dois-je faire?

Chiffre par substitution avec alphabet permuté

Texte avant transformation :

Clef : 314

Alphabet original : abcdefghijklmnopqrstuvwxyz
Alphabet permuté : upqgshvlkmrtifyowxceznjabd

Chiffrer Déchiffrer Echanger les textes

v 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 asci_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 asci_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]:  
                nvlcle = eatchpos  
  
                for eachPos in range(50):  
  
                    if lettres[loopi + nbcle * quotient].upper() == alphabet[eachPos]:  
                        print(alphabet[eachPos + nvlcle], 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 définit 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_impaire() :

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

```

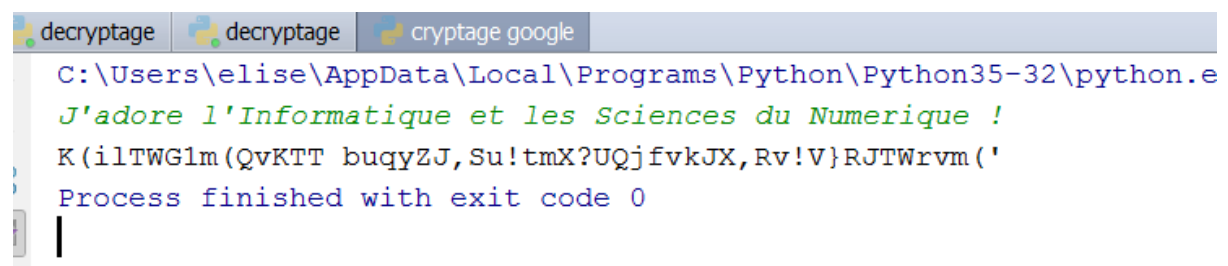
Au lieu d'ajouter 6 rangs au caractère, il faut maintenant lui enlever 6 rangs.

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 :

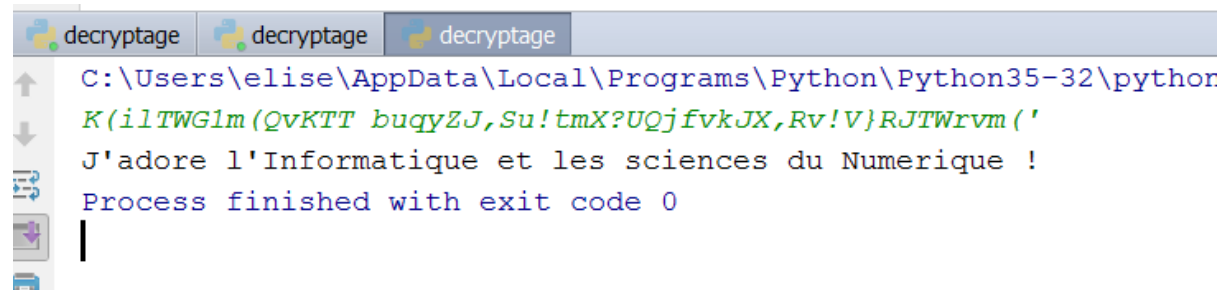


```

C:\Users\elise\AppData\Local\Programs\Python\Python35-32\python.e
J'adore l'Informatique et les Sciences du Numerique !
K(ilTWGlm(QvKTT buqyZJ,Su!tmX?UQjfvkJX,Rv!V}RJTWrvm('
Process finished with exit code 0

```

Et voici le décryptage :



```
C:\Users\elise\AppData\Local\Programs\Python\Python35-32\python
K(ilTWGlM(QvKTT buqyZJ,Su!tmX?UQjfvkJX,Rv!V}RJTWrvm('
J'adore l'Informatique et les sciences du Numerique !
Process finished with exit code 0
```

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", et 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 :

Gogole crypt

Gogole crypt

Testez notre nouvelle methode de cryptage revolutionnaire et presque indecryptable...

Attention, les accents ne sont pas lus par ce programme

Texte à crypter :

Crypter

Texte à décrypter :

Décrypter

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 a chaque fois qu'une lettre a été cryptée.

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

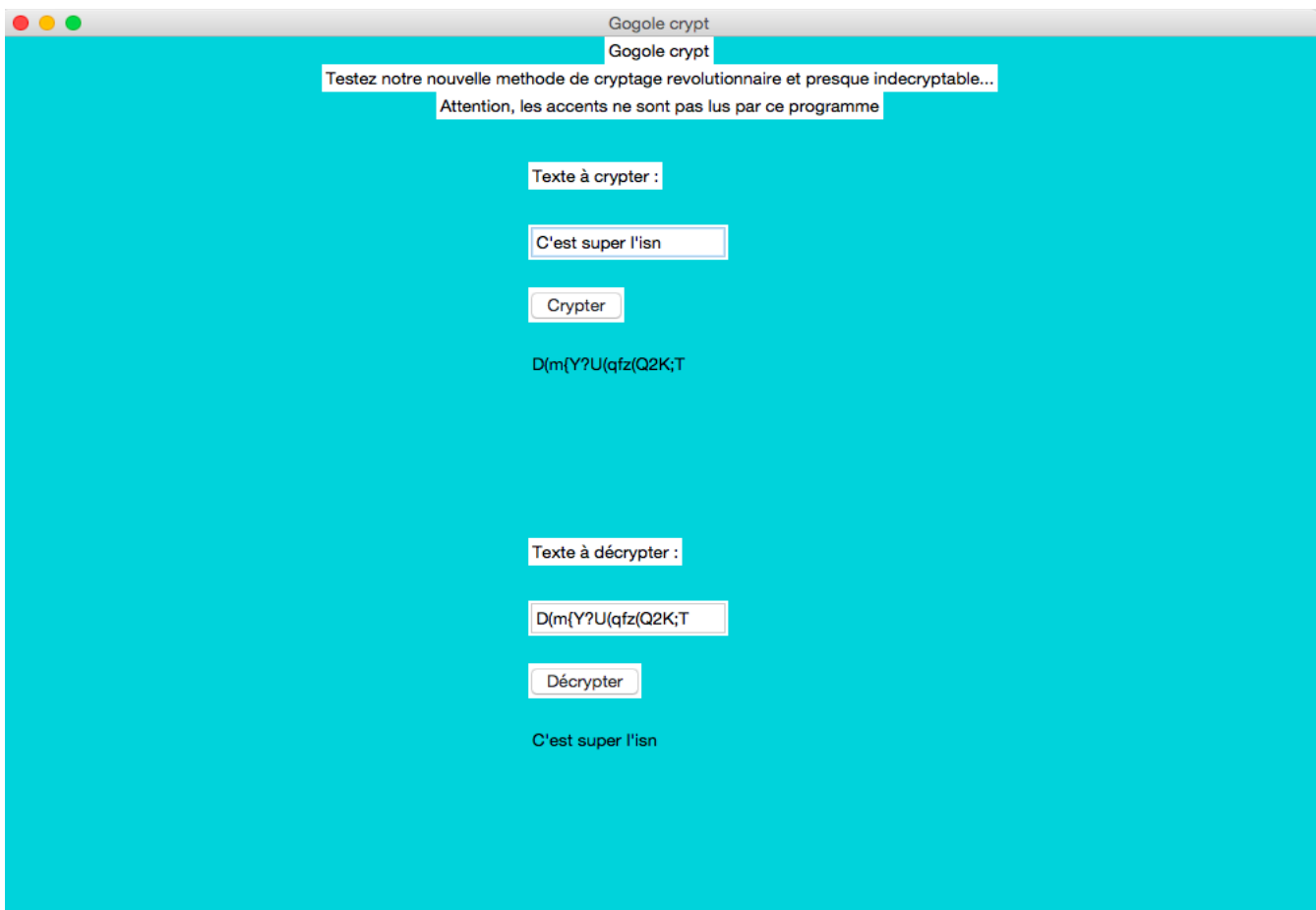
De manière à ce que l'addition a+1 ne continue pas lorsque le mot est terminé, nous avons créé une condition : cette addition se fait seulement si "a" est plus petit que le nombre de caractère entré - 1 (car le premier caractère vaut 0).

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)
label["text"] = mot

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

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



v 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/ecoledasource92/Gogole-crypt>

Partie personnelle de Violette

1. Le lancement du projet

Ma partie active dans le groupe a commencé au début du projet, lorsque nous avons entamé nos réflexions pour choisir quel pourrait être un cryptage simple mais difficilement décryptable. Grâce au projet que j'avais mené en seconde en MPS, je connaissais déjà différentes formes de cryptage comme le chiffre de César, Vigenère ... C'est en expliquant brièvement mes connaissances sur la cryptographie à Elise et Pauline que nous avons eu l'idée de toutes les mélanger, afin de créer un programme qui code deux lettres par technique de cryptage.

2. L'interface HTML

La partie sur laquelle j'ai le plus travaillé est celle de l'interface que j'ai du faire dans son intégralité. J'ai commencé lorsque nous avons fini de coder le cryptage : Pauline et Elise ont commencé le décryptage pendant que j'apprenais à coder en html afin de créer une interface graphique. J'ai utilisé le site open class room avec un cours d'environ 30 pages pour apprendre les bases du codage html et CSS. Le codage en html permet de créer des interfaces grâce à ce que l'on appelle des balises. Celles-ci permettent à l'ordinateur de comprendre ce qu'il doit afficher. Le code CSS permet d'ajouter la forme de cette interface, de choisir des polices, des couleurs ... J'ai décidé d'insérer directement le code CSS dans le code html. Il est situé dans l'en-tête appelé balise "head" du code.

Le code commence donc par l'initiation de la fenêtre et la création d'un en-tête, la balise "head" dans laquelle se trouve le titre de l'interface et le code CSS que je décrirais dans un second temps. Par la suite vient la balise "body", c'est à dire le corps de l'interface.

```
<body>
  <header>
     <!-- mettre une image gif en en-tete-->
    <h1><strong>Gogole crypt</strong></h1> <!-- Mettre gogole crypt en titre-->
    <p class="intro"> Testez notre nouvelle methode de cryptage revolutionnaire et presque indecryptable...</p>
    <p class="entreeacrypter">Texte à crypter </p>
    <textarea class="cadre1"> </textarea>
    <p class="sortiecrypté"> Texte crypté </p>
    <textarea class="cadre2"> </textarea>
  </body>
```

Ici, la balise "img" correspond une image gif que j'ai fait apparaitre sur la fenêtre. Puis la balise "h1" signifie "titre 1". Le titre de l'interface est "gogole crypt". La balise "strong" signifie que ce titre est important et le fait légèrement ressortir par rapport au reste du texte présent sur l'interface. On observe ensuite des balises "p", ce sont des paragraphes. Le premier présente notre méthode de cryptage et le second introduit l'entrée dans laquelle l'utilisateur doit saisir son texte à crypter. Cette entrée est créée par la balise "textarea". Cela se répète pour la sortie du texte crypté. On remarque que la plus part des balises sont "classées". Chaque "class=" présente après l'ouverture des balises donne un nom à celle ci, ce qui facilitera le codage de la forme en CSS.

Pour le codage en CSS, j'ai commencé par ouvrir une balise "style" dans laquelle j'ai mis les différentes classes de mon code html. Tout d'abord, j'ai choisis de mettre tous les paragraphes en police Georgia :

```
<style>
  p
  {
    font-family: Georgia
  }
```

Puis, pour la class intro, j'ai mis un bloc blanc autour du texte en choisissant sa largeur, ses dimensions, sa marge et une ombre.

```
.intro
{
  color: white; /* mettre tout le texte des paragraphes en blanc */
  width: 350px; /* On a indiqué une largeur */
  margin: auto; /* demander à ce que le bloc soit centré avec auto */
  border: 1px solid white;
  text-align: justify;
  padding: 12px;
  margin-bottom: 20px;
  box-shadow: 6px 6px 0px white;
}
```

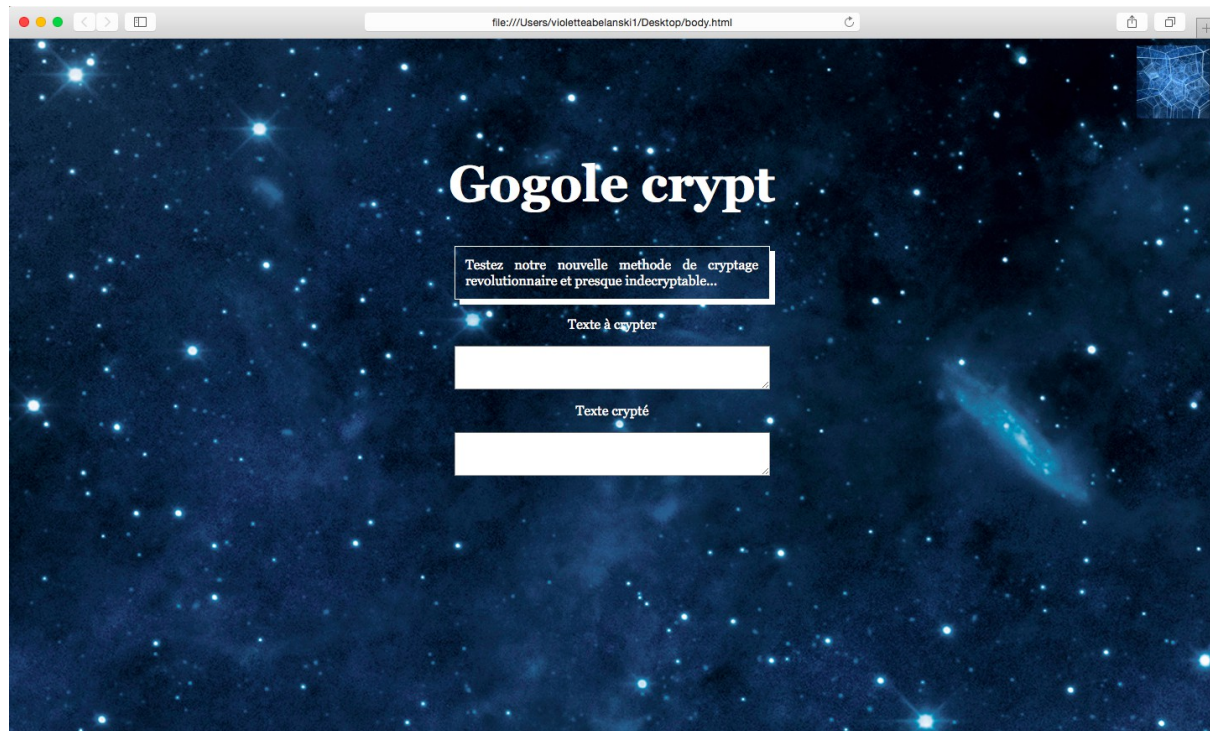
Ensuite, j'ai ajouté une image en fond de fenêtre, puis j'ai aligné tout le texte de l'en tête a droite, j'ai aussi grossi le titre, mis en police georgia, centré et en blanc.

```
body
{
    background-image: url("FOND-ETOILE-web-HD.jpg") /* mettre un fond étoilé*/
}
header
{
    text-align: right;
}
h1
{
    font-size: 60px; /* Titre h1 de 60 pixels */
    font-family: Georgia; /* le titre est en police Georgia */
    text-align: center; /* le titre est centré */
    color: white; /* le titre est en blanc */
}
```

Pour finir, j'ai mis le texte "texte à crypter" en blanc et centré, j'ai créé le cadre d'entrée 1 en blanc avec ses dimensions et sa marge puis le texte " texte crypté" ainsi que le cadre de sortie.

```
.entréeacrypter
{
    color:white;
    text-align: center;
}
.cadre1
{
    width: 350px; /* On a indiqué une largeur */
    padding: 12px;
    display: block;
    margin:auto;
}
.sortiecrypté
{
    color:white;
    text-align: center;
}
.cadre2
{
    width: 350px; /* On a indiqué une largeur */
    padding: 12px;
    display: block;
    margin: auto;
}
```

Voici le rendu final de cette interface en html :



3. L'interface en python

Finalement, le rendu physique terminé, j'ai dû trouver un moyen d'insérer notre code en python dans ce code en html. Après de nombreuses recherches et l'installation de quelques logiciels s'étant avérés trop compliqué pour mon niveau, j'ai décidé d'abandonner cette interface et d'en créer une en python. J'ai alors commencé à apprendre comment créer une interface grâce à tkinter avec une dizaine de vidéos Youtube. L'interface en python a été détaillée dans notre partie commune. J'ai d'abord créé la partie physique de l'interface pour créer les différentes zones de texte et entrées.

Afin que notre code fonctionne dans l'interface j'ai dû modifier quelques paramètres. Dans notre code initiale, chaque lettre modifiée se "print" à la fin de chaque fonction de cryptage. Cependant, pour que le mot ou la phrase crypté ou décrypté s'affiche sur l'interface, il a fallu trouver un moyen de rassembler chaque lettre à la fin de chaque fonction et de les "print" seulement à la fin de l'exécution du code. Pour cela, j'ai eu l'idée de créer un tableau :

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

Les lettres modifiées entrent dans ce tableau à la fin de chaque fonction de cryptage, lorsqu'on remplace la lettre initiale du mot entré par la lettre cryptée dans chaque case du tableau .

Après chaque modification de lettre, il suffit de faire + 1 à la variable "a", initialement à 0, pour que la prochaine lettre modifiée soit la suivante dans le texte entré. Cependant, cela ne suffisait pas. Il fallait que la variable "a" ne fasse + 1 seulement si il reste des caractères à coder. Puisque le premier caractère du tableau a pour numéro 0, on pose une condition : la variable "a" fait + 1 seulement si celle ci est inférieure au nombre de caractère dans l'entrée - 1.

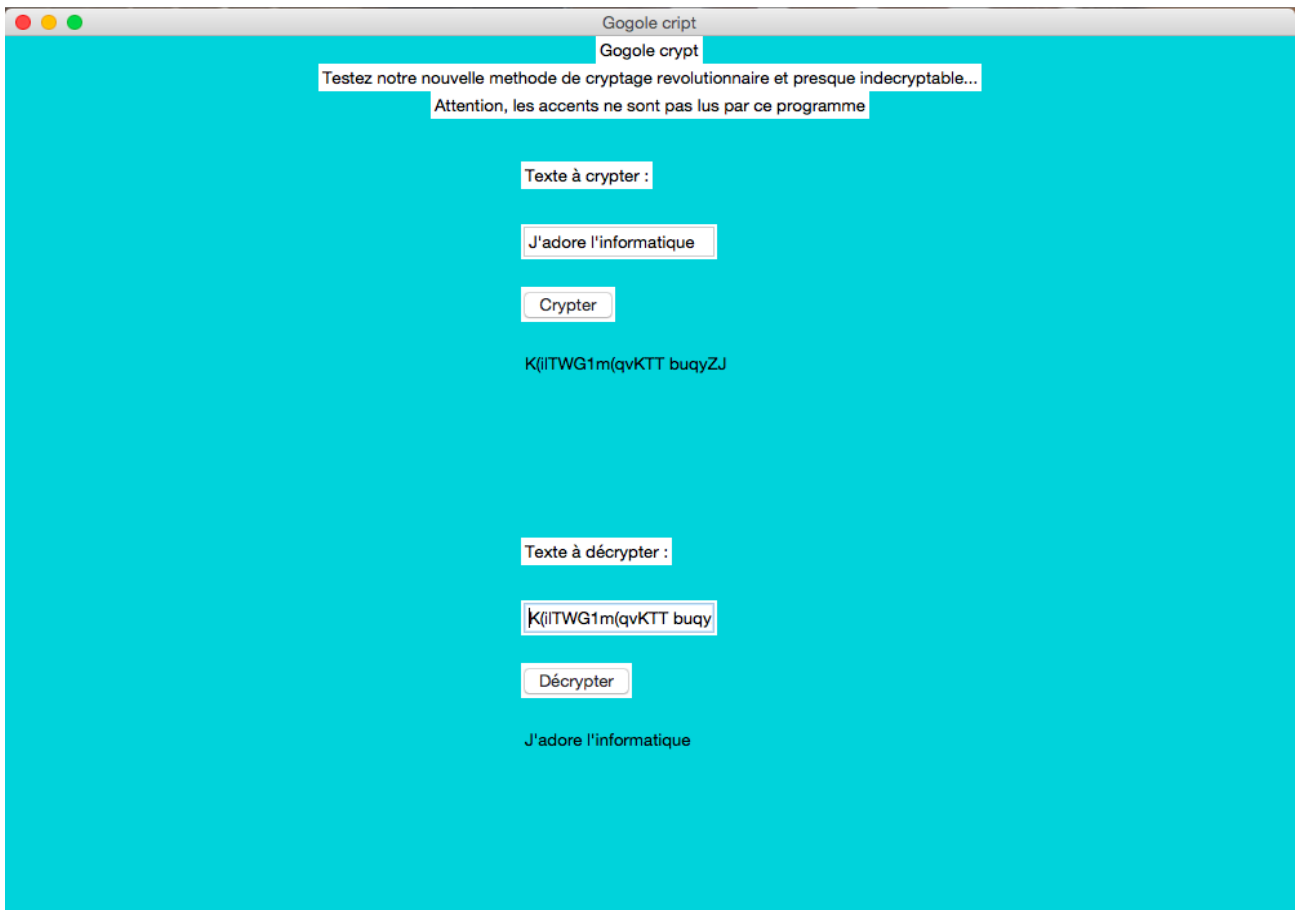
```
tableau[a] = newCaractere
if a < (nbtexte-1) :
    a = a + 1
```

A la fin du code, les éléments du tableaux se rassemblent grâce à "join" et se "print" sur l'interface à la place destinée à la sortie : "label" pour le cryptage et "lebel" pour le décryptage.

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

De ce fait, afin de permettre le bon fonctionnement de notre interface, j'ai dû comprendre précisément le déroulement du code de cryptage et de décryptage. C'est pour cela que même si j'ai beaucoup plus participé à la création de l'interface qu'à celle du code, ma compréhension de ce dernier n'est pas moindre.

Voici la preuve du bon fonctionnement de l'interface :



4. Le travail de groupe

Durant cette année, le travail de groupe avec Elise et Pauline s'est particulièrement bien déroulé. Nous nous sommes d'emblé accordé sur le choix du projet ainsi que la manière dont nous allions aborder les techniques de cryptage. Rapidement, nous nous sommes répartis les tâches afin de pouvoir avancer sur le projet durant la semaine puis regrouper nos travaux durant les cours.

Toujours sous une bonne entente, je me suis rapidement proposée pour la création de l'interface sans oublier d'expliquer au reste du groupe le fonctionnement de celle ci ainsi que les problèmes auxquels j'ai eu à faire. En effet, j'ai créé l'interface de toute part mais celle ci n'aurait pas vu le jour sans Elise et Pauline qui réglaient les problèmes de notre code en même temps, afin que celui ci soit prêt lorsque le côté physique de l'interface serait finit.

Toutes les décisions que nous avons dut prendre pour notre projet se sont faites ensemble, sous une très bonne entente, de l'abandon de l'interface html, à la couleur du fond de l'interface en python, en passant par le nombre de décalage du cryptage.

Notre projet n'est bien sure pas parfait, nous aurions pu créer un programme de cryptage bien plus performant et encore moins facile à décrypter mais malheureusement nous n'avions pas assez de temps pour approfondir le code. Pour l'interface, si j'avais eu plus de temps j'aurais éventuellement put trouver un moyen de rendre fonctionnelle l'interface html, ou d'améliorer le graphisme de l'interface tkinter. Cependant, nous sommes toutes les trois très fiers du rendu car nous avons travaillé dur pour y parvenir, nous avons du régler de nombreux problèmes et prendre de nombreuses décisions. Finalement, nous sommes parvenues à bout de notre idée initiale et nous somme très satisfaites de notre travail.

Ce travail m'a apporté beaucoup de connaissances notamment sur la création d'interface. J'ai pu comprendre la structure des sites internet, la manière dont ils sont créés ainsi que tout le travail qui se cache derrière une simple fenêtre. Ce travail de groupe a été une réussite pour nous toutes, du point de vu de la collaboration comme de l'aboutissement à un projet fonctionnel.