



PROJET NSI 1ère: CRYPTAGES

Fonctionnement du code

TABLE DES MATIÈRES

Explication du projet

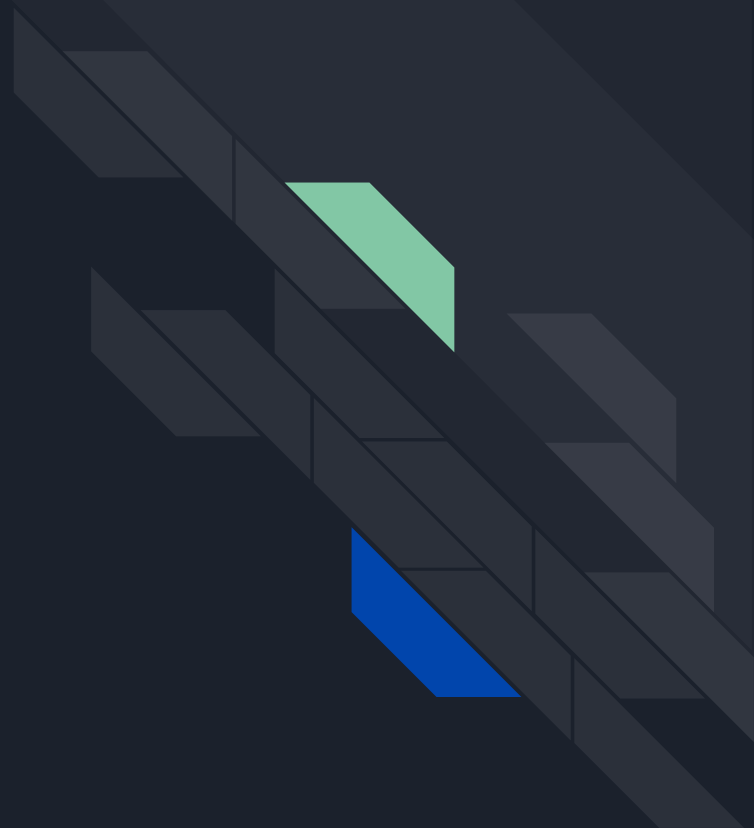
Schéma des fichiers

Module algorithmes

Module interface

Fichier principal

Résultat





Explication du projet

Dans ce projet, je vais travailler sur un logiciel de cryptage et décryptage de messages grâce à quatre célèbres méthodes de cryptage basique. Le logiciel permettra à l'utilisateur d'introduire un message alphanumérique puis de choisir la méthode de codage sur une interface graphique, ainsi que le décoder en connaissant la méthode de codage et la clé de déchiffrement.

Une autre caractéristique disponible sera la sauvegarde d'un fichier texte avec les données et métadonnées de l'opération.



Schéma des fichiers

Pour utilisation: output/
 cryptages /
 ... // contiendra les fichiers.txt des métadonnées
 cryptage.exe // reste des fichiers accompagnant l'exécutable
 icone.ico // fichier exécutable
 ... // icône du logiciel

Pour développement: src/
 modules/
 algorithmes/
 __init__.py
 cesar.py
 polybe.py
 rot13.py
 vigenere.py
 interface/
 __init__.py
 interface.py
 cryptage.py



Module algorithmes

- Chaque sous-module correspond à un algorithme de cryptage, pour lequel il présente une fonction d'encodage, et une autre de décodage
- Les fonctions peuvent être importées depuis un autre script, mais ne seront pas appelées lors d'une éventuelle exécution du module

Module algorithmes - César (codage)

- Prend un message clair et un décalage en argument
- Vérifie que le décalage est numérique
- Pour chaque caractère
 - Si c'est un chiffre ou une lettre: applique le décalage sur les 10 chiffres ou les 26 lettres
 - Si c'est un caractère spécial: le laisse inchangé
- Retourne le résultat

```
cesar.py X
NSI_FILES_2122 > projet > src > modules > algorithmes > cesar.py > ...
4 def cesar_c(message_clair, decalage=3):
5
6     """
7     Retourne un message reçu en paramètre codé selon la méthode du code de César.
8     """
9
10    # convertit decalage en entier et si ce n'est pas possible car il s'agit d'un caractère, affiche une erreur
11    try:
12        decalage = int(decalage)
13    except ValueError:
14        raise ValueError("La clé avec César doit être un décalage numérique")
15
16    message_code = ""
17
18    for caractere in message_clair.upper():
19        if caractere in alphabet: # vérifie si le caractère est une lettre pour la coder, ou le laisser inchangé si ce ne l'est pas
20            i = alphabet.index(caractere)
21            i_codage = (i+decalage)%26 # codage de la lettre avec le décalage indiqué
22
23            message_code += alphabet[i_codage]
24
25        elif caractere in chiffres: # vérifie si le caractère est un chiffre pour le coder, ou le laisser inchangé si ce ne l'est pas
26            i = chiffres.index(caractere)
27            i_codage = (i+decalage)%10 # codage d'un chiffre avec le décalage indiqué
28
29            message_code += chiffres[i_codage]
30        else:
31            message_code += caractere
32
33    # retourne un dictionnaire contenant les caractéristiques du codage et son résultat pour les montrer sur l'interface
34    return {
35        "methode": "César",
36        "alphabet_base": alphabet,
37        "chiffres_base": chiffres,
38        "cle": decalage,
39        "message_clair": message_clair,
40        "message_code": message_code
41    }
```

Module algorithmes - César (décodage)

- Prend un message clair et un décalage en argument
- Vérifie que le décalage est numérique
- Pour chaque caractère
 - Si c'est un chiffre ou une lettre: applique le décalage inverse sur les 10 chiffres ou les 26 lettres
 - Si c'est un caractère spécial: le laisse inchangé
- Retourne le résultat

```
43 def césar_d(message_code, decalage=3):
44     """
45     Retourne décodé un message préalablement codé selon la méthode de César. Le message est reçu en paramètre.
46     """
47
48     # convertit decalage en entier et si ce n'est pas possible car il s'agit d'un caractère, affiche une erreur
49     try:
50         decalage = int(decalage)
51     except ValueError:
52         raise ValueError("La clé avec César doit être un décalage numérique")
53
54     message_decode = ""
55
56     for caractere in message_code.upper():
57         if caractere in alphabet: # vérifie si le caractère est une lettre pour la décoder
58             i = alphabet.index(caractere)
59             i_decodage = i - decalage
60
61             # vérification: le codage se réalise pour des indices de 0 à 25 (26 lettres dans l'alphabet)
62             if i_decodage < 0:
63                 i_decodage += 26
64
65             message_decode += alphabet[i_decodage]
66
67         elif caractere in chiffres: #vérifie si le caractère est un chiffre pour le décoder
68             i = chiffres.index(caractere)
69             i_decodage = i - decalage
70
71             # vérification: le codage se réalise pour des indices de 0 à 9 (10 chiffres de base)
72             if i_decodage < 0:
73                 i_decodage += 10
74
75             message_decode += chiffres[i_decodage]
76         else:
77             message_decode += caractere
78
79     #retourne un dictionnaire contenant les caractéristiques du décodage et son résultat pour les montrer sur l'interface
```

Module algorithmes - ROT 13

- Prend un message clair en argument
- Appelle la fonction de codage ou décodage de César avec un décalage de 13

```
© NSI_FILES_2122 > projet > src > modules > algorithmes > rot13.py > ...
1  from .cesar import cesar_c, cesar_d
2
3  def rot13_c(message_clair):
4      """
5      Retourne un message reçu en paramètre codé selon la méthode ROT-13.
6      """
7
8      # rot13 est le code de césar avec un décalage de 13
9      resultats = cesar_c(message_clair, 13)
10
11     # retourne un dictionnaire contenant les caractéristiques du codage et son résultat pour les montrer sur l'interface
12     return {
13         "methode": "ROT13",
14         "alphabet_base": resultats["alphabet_base"],
15         "chiffres_base": resultats["chiffres_base"],
16         "cle": None,
17         "message_clair": message_clair,
18         "message_code": resultats["message_code"]
19     }
20
21 def rot13_d(message_code):
22     """
23     Retourne décodé un message préalablement codé selon la méthode ROT-13.
24     Le message est reçu en paramètre.
25     """
26
27     # rot13 est le code de césar avec un décalage de 13
28     resultats = cesar_d(message_code, 13)
29
30     # retourne un dictionnaire contenant les caractéristiques du décodage et son résultat pour les montrer sur l'interface
31     return {
32         "methode": "ROT13",
33         "alphabet_base": resultats["alphabet_base"],
34         "chiffres_base": resultats["chiffres_base"],
35         "cle": None,
36         "message_code": message_code,
37         "message_decode": resultats["message_decode"]
38     }
```


Module algorithmes - Vigenère (codage)

- Prend un message clair, une clé et un alphabet en argument
- Vérifie que les caractères de la clé appartiennent à l'alphabet
- Ajuste la longueur de la clé en la répétant et supprime les espaces
- Pour chaque caractère
 - Si c'est une lettre de l'alphabet: applique le décalage en additionnant l'indice de la lettre et de la position de clé correspondante
 - Si c'est un caractère spécial ou un chiffre: le laisse inchangé
- Retourne le résultat

```
vigenere.py X
@ NSI_FILES_2122 > projet > src > modules > algorithmes > vigenere.py > vigenere_c
1 def vigenere_c(message_clair, cle_orig, alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
2     """
3     Code un message selon le chiffrement de Vigenère
4     """
5
6     # vérifie que la clé ne contient que des lettres
7     for caractere in cle_orig.upper():
8         if not caractere in alphabet:
9             raise ValueError("Un caractère de la clé n'appartient pas à l'alphabet. Rappel: la clé ne peut contenir que des lettres non accentuées avec Vigenère. Exceptio
10
11     message_code = ""
12
13     cle = cle_orig.replace(" ", "") # supprime les espaces dans la clé
14
15     # s'assure que la clé est assez longue pour coder le message en la répétant autant de fois que nécessaire
16     while len(cle) < len(message_clair):
17         cle += cle
18
19     k = 0 # variable qui va contenir l'indice du caractère travaillé de la clé
20
21     for caractere in message_clair.upper():
22         if caractere in alphabet:
23             # somme des indices du caractère obtenu dans l'alphabet et du caractère de la clé correspondant, modulo len(alphabet) pour obtenir son indice dans l'alphabet
24             i = ( alphabet.index(caractere) + alphabet.index(cle.upper()[k]) ) % len(alphabet)
25             k += 1
26
27             message_code += alphabet[i]
28         # code uniquement les caractères présents dans l'alphabet indiqué en paramètre
29         else:
30             message_code += caractere
31
32     # retourne un dictionnaire contenant les caractéristiques du codage et son résultat pour les montrer sur l'interface
33     return {
34         "methode": "Vigenère",
35         "alphabet_base": alphabet,
36         "chiffres_base": None,
37         "cle": cle_orig,
38         "message_clair": message_clair,
39         "message_code": message_code
40     }
```

Module algorithmes - Vigenère (décodage)

- Prend un message clair, une clé et un alphabet en argument
- Vérifie que les caractères de la clé appartiennent à l'alphabet
- Ajuste la longueur de la clé en la répétant et supprime les espaces
- Pour chaque caractère
 - Si c'est une lettre de l'alphabet: applique le décalage inverse en soustrayant l'indice de la lettre et de la position de clé correspondante
 - Si c'est un caractère spécial ou un chiffre: le laisse inchangé
- Retourne le résultat

```
40
41
42 def vigenere_d(message_code, cle_orig, alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
43     """
44     Décode un message selon le chiffrement de Vigenère
45     """
46
47     # vérifie que la clé ne contient que des lettres
48     for caractere in cle_orig.upper():
49         if not caractere in alphabet:
50             raise ValueError("Un caractère de la clé n'appartient pas à l'alphabet. Rappel: la clé ne peut contenir que des lettres non accentuées avec Vigenère.")
51
52     message_decode = ""
53
54     cle = cle_orig.replace(" ", "") # supprime les espaces dans la clé
55
56     # s'assure que la clé est assez longue pour décoder le message en la répétant autant de fois que nécessaire
57     while len(cle) < len(message_code):
58         cle += cle
59
60     k = 0
61
62     for caractere in message_code.upper():
63         if caractere in alphabet:
64             # différence des indices du caractère obtenu dans l'alphabet et du caractère de la clé correspondant
65             i = (alphabet.index(caractere) - alphabet.index(cle.upper()[k]))
66             # vérification: l'indice obtenu correspond à une position dans l'alphabet de len(alphabet) lettres
67             if i < 0:
68                 i += len(alphabet)
69             k += 1
70
71             message_decode += alphabet[i]
72
73         # décode uniquement les caractères présents dans l'alphabet indiqué en paramètre
74         else:
75             message_decode += caractere
76
77     # retourne un dictionnaire contenant les caractéristiques du décodage et son résultat pour les montrer sur l'interface
78     return {
79         "methode": "Vigenère",
```



Module algorithmes - Polybe (grille)

- Fonction partagée par les fonctions de codage et de décodage
- Crée la grille nécessaire pour appliquer l'algorithme de Polybe à partir d'un alphabet de 25 caractères

```
def creer_grille(alphabet):  
    # vérifie que l'alphabet introduit a bien une longueur de 25, arrete l'exécution sdu programme et affiche une erreur sinon  
    if len(alphabet) != 25:  
        raise ValueError("L'alphabet doit comporter 25 lettres")  
  
    # construit la grille de codage (matrice) a partir de l'alphabet de 25 lettres donné  
    grille = list()  
    for i in range(1, 6):  
        ligne = alphabet[5*(i-1) : (5*i)]  
        grille.append(list(ligne))  
    return grille
```

Module algorithmes - Polybe (codage)

- Crée la grille
- Pour chaque caractère
 - Si c'est une lettre de l'alphabet: ajoute au message décodé les coordonnées de la lettre dans la grille
 - si c'est un chiffre: génère une erreur, les chiffres ne peuvent pas être codés avec Polybe
 - Si c'est un caractère spécial: le laisse inchangé
- Retourne le résultat

```
def polybe_c(message_clair, alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
    """
    Code un message selon le chiffrement de Polybe
    """

    grille = creer_grille(alphabet)

    message_code = ""

    for caractere in message_clair.upper():
        if caractere in alphabet:
            codage = ""
            # cherche dans quelle ligne de la matrice de codage se trouve le caractère à coder, joint sa ligne et colonne au message
            for ligne in grille:
                if caractere in ligne:
                    codage += str(grille.index(ligne) + 1)
                    codage += str(ligne.index(caractere) + 1)
                    message_code += codage
        elif caractere in chiffres:
            # affiche une erreur si le caractère à coder est un chiffre
            raise ValueError("Les chiffres ne peuvent pas être encodés avec Polybe")
        else:
            message_code += caractere
```

Module algorithmes - Polybe (décodage)

- Vérifie que le message ne contient pas de lettres
- Crée la grille
- Pour chaque caractère
 - Si c'est un chiffre: ajoute au message décodé la lettre correspondant aux coordonnées dans la grille
 - Si c'est un caractère spécial: le laisse inchangé
- Retourne le résultat

```
def polybe_d(message_code, alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"):
    """
    Décode un message selon le chiffrement de Polybe
    """

    for caractere in message_code.upper():
        print(caractere)
        if caractere in alphabet:
            raise ValueError("Les lettres ne peuvent pas être décodées avec Polybe")

    grille = creer_grille(alphabet)
    message_decode = ""

    # la variable i sera incrémentée de 2 si sa position dans le message contient un chiffre
    # (message_code[i]-1 et message_code[i+1]-1 seront les indices sur la matrice d'une lettre)
    # et sera incrémentée de 1 si le caractère trouvé est différent (espace, ponctuation)
    i = 0

    while i < len(message_code):
        if message_code[i] in chiffres:
            # ajoute au message décodé le caractère trouvé sur la grille en fonction de ses coordonnées (indiquées sur le message codé)
            x = int(message_code[i])-1
            y = int(message_code[i+1])-1
            message_decode += str(grille[x][y])
            i+=2
        else:
            message_decode += message_code[i]
            i+=1

    # retourne un dictionnaire contenant les caractéristiques du codage et son résultat pour les montrer sur l'interface
    return {
        "methode": "Polybe",
        "alphabet_base": alphabet,
        "chiffres_base": chiffres,
        "cle": None,
        "message_code": message_code,
```



Module interface

- Contient la classe Interface, dont le constructeur crée l'interface graphique avec ses objets; et les méthodes permettent de modifier ses attributs
- Utilise le module Tkinter
- Si elle il est exécuté, le module affiche la phrase: `"Vous avez exécuté le module Interface, qui contient la classe Interface"`
- Le code est ci-après

NSI_FILES_2122 > projet > src > modules > interface > interface.py > Interface > __init__

```

1  # importation du module externe Tkinter qui nous aide à contruire une interface graphique
2  from tkinter import (CENTER, RIGHT, Button, Label, OptionMenu,
3  | | | | | Radiobutton, StringVar, Text, Entry, messagebox)
4
5  import os
6  from pathlib import Path
7
8  class Interface:
9      """
10     Classe qui définit l'aspect et la fonctionnalité de l'interface. Elle prend un objet de type Tk en argument, ainsi qu'une fonction de codage/décodage et une fonction pou
11     Certains attributs de la classe commencent par "_". Ils sont supposés être privés et ne doivent pas être accédés en dehors de la classe, or le concept de règles de visib
12     """
13
14     def __str__(self):
15         return "Interface basée sur un objet Tk"
16
17     def __init__(self, fenetre, codage, sauvegarder):
18
19         """
20         | Constructeur de la classe, exécuté lorsque une instance est créée. Initialise les composants de l'interface.
21         """
22
23         # initialisation de la fenêtre
24         self._fenetre = fenetre
25         self._fenetre.title("Encodeur - Décodeur")
26         self._fenetre.iconbitmap(f'{Path(os.getcwd()).parent.absolute()}\cryptage\icone.ico') # l'icone, lors de d'installation, se trouve dans le même directoire 'cryptage' que
27
28         ## COMPOSANTS DE L'INTERFACE
29
30         # crée une étiquette conteant les instructions
31         self._message_instructions = Label(self._fenetre, width=50, height=2, font=("Helvetica", 12), background="#f0f0ed", text="Choisissez la méthode, le mode et la clé de coda
32
33         # place le texte à un endroit précis de la fenêtre avec la méthode grid()
34         self._message_instructions.grid(row=0, column=0, columnspan=4, padx=10, pady=10)
35
36         # variable qui va contenir le mode de codage sélectionnée, 1 (Encoder) est la valeur par défaut
37         self._selection_mode = StringVar(self._fenetre, "1")
38
39         # dictionaire contenant le texte et le numéro des boutons à générer pour les différents modes disponibles

```



```

38 # dictionnaire contenant le texte et le numéro des boutons à générer pour les différents modes disponibles
39 modes = {
40     "Encoder": "1",
41     "Décoder": "2"
42 }
43
44 # boucle créant les boutons indiqués dans le dictionnaire
45 ligne = 1
46 for (texte, valeur) in modes.items():
47     Radiobutton(self._fenetre, text = texte, variable = self._selection_mode, value = valeur, command=self.changer_selon_mode).grid(row = ligne, column=0, columnspan=4,
48     ligne += 1
49
50 # variable qui va contenir la méthode de codage sélectionnée, ROT13 est la valeur par défaut
51 self._selection_methode = StringVar(self._fenetre, "ROT13")
52
53 # tuple contenant les différentes méthodes disponibles et liste affichée, de type OptionMenu
54 methodes = ("ROT13", "CODE DE CÉSAR", "CODE DE VIGENÈRE", "CARRÉ DE POLYBE")
55 self._liste_de_methodes = OptionMenu(self._fenetre, self._selection_methode, *methodes, command=self.changer_selon_methode)
56
57 # affiche l'outil de sélection de la méthode de codage
58 self._liste_de_methodes.grid(row = 3, column=0, columnspan=4, padx=10, pady=10)
59
60 # étiquette et champ de texte pour entrer le message à encoder/décoder
61 self._label_entree = Label(self._fenetre, width=50, height=2, font=("Helvetica", 12), background="#f0f0ed", text="Entrez le message:
62 self._label_entree.grid(row = 4, column=0, columnspan=4, pady=5, sticky="W")
63
64 self._champ_entree = Text(self._fenetre, font=('Helvetica', 12), height=3, padx=10, pady=10)
65 self._champ_entree.grid(row = 5, column=0, columnspan=4, padx=10, pady=2)
66
67 # variable qui contiendra la valeur de la clé
68 self._v_cle = StringVar(self._fenetre, "")
69
70 # étiquette et champ de texte pour entrer la clé de codage
71 # elles ne sont pas montrées par défaut car la méthode par défaut, rot13, ne nécessite pas de clé de codage
72 self._label_cle = Label(self._fenetre, width=50, height=2, font=("Helvetica", 12), background="#f0f0ed", text="Entrez la clé de codage:")
73
74 self._entree_cle = Entry(self._fenetre, textvariable=self._v_cle)

```



```

75     self._entree_cle = Entry(self._fenetre, textvariable=self._v_cle)
76
77     # étiquette et champ de texte où le message encodé/décodé sera affiché
78     self._label_sortie = Label(self._fenetre, width=50, height=2, font=("Helvetica", 12), background="#f0f0ed", text="Le résultat est:
79     self._label_sortie.grid(row = 9, column=0, columnspan=4, pady=5, sticky="W")
80
81     self._champ_sortie = Text(self._fenetre, font=('Helvetica', 12), height=3, padx=10, pady=10, state="disabled")
82     self._champ_sortie.grid(row = 12, column=0, columnspan=4, padx=10, pady=2)
83
84     # bouton qui, actionné, appellera la fonction de codage
85     self._bouton_coder = Button(self._fenetre, justify=CENTER, text = "Coder", command=codage)
86     self._bouton_coder.grid(row = 13, column=0, columnspan=4, pady=10)
87
88     # bouton qui, actionné, sauvegardera le message introduit et le résultat de son encodage/décodage
89     self._bouton_sauvegarder = Button(self._fenetre, justify=RIGHT, text = "Sauvegarder", command=sauvegarder)
90     self._bouton_sauvegarder.grid(row = 13, column=3, columnspan=4, pady=10)
91
92 # getters pour le message entré, la sortie, le mode, la méthode, et la clé
93 def get_entree(self):
94     """
95     |   Retourne le message entré
96     |   """
97     return self._champ_entree.get("1.0", "end-1c")
98
99 def get_sortie(self):
100     """
101     |   Retourne le message à la sortie
102     |   """
103     return self._champ_sortie.get("1.0", "end-1c")
104
105 def get_mode(self):
106     """
107     |   Retourne le mode de codage
108     |   """
109     return self._selection_mode.get()
110
111 def get_methode(self):
112     """
113     |   Retourne la méthode codage

```

```

109         return self._selection_mode.get()
110
111     def get_methode(self):
112         """
113         | Retourne la méthode codage
114         """
115         return self._selection_methode.get()
116
117     def get_cle(self):
118         """
119         | Retourne la clé de codage
120         """
121         return self._v_cle.get()
122
123
124     # méthodes de la classe
125     def effacer_sortie(self):
126         """
127         | Fonction qui efface le champ de sortie
128         """
129         self._champ_sortie.configure(state="normal")
130         self._champ_sortie.delete("1.0", "end-1c")
131         self._champ_sortie.configure(state="disabled")
132
133
134     def afficher_sortie(self, message):
135         """
136         | Fonction qui affiche le message à la sortie
137         """
138         self._champ_sortie.configure(state="normal")
139         self._champ_sortie.insert("1.0", message)
140         self._champ_sortie.configure(state="disabled")
141
142
143     def afficher_alerte(self, alerte):
144         """
145         | Fonction qui affiche une alerte avec un message de texte donné
146         """
147         messagebox.showerror("Erreur", alerte)

```

```

148
149 def changer_selon_mode(self):
150     """
151     Fonction qui change l'interface en fonction du mode de codage sélectionné
152     """
153     if self.get_mode() == "1":
154         self._message_instructions.config(text="Choisissez la méthode, le mode et la clé de codage")
155         self._label_cle.config(text="Entrez la clé de codage:")
156         self._bouton_coder.config(text="Coder")
157     elif self.get_mode() == "2":
158         self._message_instructions.config(text="Choisissez la méthode, le mode et la clé de décodage")
159         self._label_cle.config(text="Entrez la clé de décodage:")
160         self._bouton_coder.config(text="Décoder")
161
162 def changer_selon_methode(self, *args):
163     """
164     Fonction qui change l'interface en fonction de la méthode de codage sélectionnée
165     """
166     if self.get_methode() == "ROT13" or self.get_methode() == "CARRÉ DE POLYBE":
167         self._label_cle.grid_forget()
168         self._entree_cle.grid_forget()
169     elif self.get_methode() == "CODE DE CÉSAR":
170         self._label_cle.config(text="Entrez le décalage:")
171         self._label_cle.grid(row = 8, column=0, columnspan=4, pady=5, sticky="W")
172         self._entree_cle.grid(row = 8, column=1, columnspan=4, pady=5)
173     elif self.get_methode() == "CODE DE VIGENÈRE":
174         self._label_cle.config(text="Entrez la clé de codage:")
175         self._label_cle.grid(row = 8, column=0, columnspan=4, pady=5, sticky="W")
176         self._entree_cle.grid(row = 8, column=1, columnspan=4, pady=5)
177
178 # si le module n'est pas importé, mais exécuté
179 if __name__ == "__main__":
180     print("Vous avez exécuté le module Interface, qui contient la classe Interface")

```

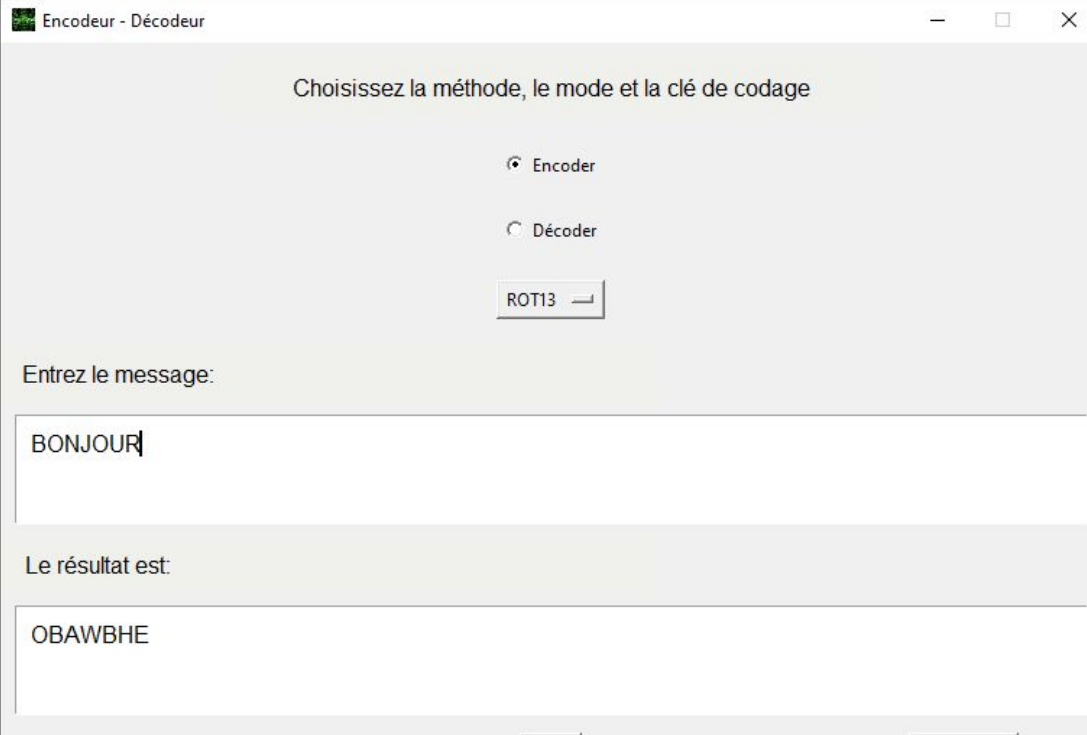


Fichier principal

- Contient une fonction “codage()” qui appliquera les différentes méthodes de codage selon ce que l'utilisateur a sélectionné, et affichera le résultat dans l'interface
- Contient une fonction “sauvegarder()” qui crée un fichier.txt dans le repertoire “output” avec les métadonnées de l'opération si l'utilisateur le demande
- Contient la boucle principale du programme, qui crée une instance de la classe Interface à partir d'une instance Tk et affiche l'interface

```
100
101 # création d'un objet Tk et de l'interface (objet), il sera impossible de changer les dimensions de l'interface
102 fenetre_principale = Tk()
103 fenetre_principale.resizable(False, False)
104 interface = Interface(fenetre_principale, codage, sauvegarder)
105
106 # le script entre dans une boucle infinie en attendant qu'un évènement se produise
107 fenetre_principale.mainloop()
```

RÉSULTAT FINAL



Encodeur - Décodeur

Choisissez la méthode, le mode et la clé de codage

☒ Encoder

☐ Décodeur

ROT13

Entrez le message:

BONJOUR

Le résultat est:

OBAWBHE