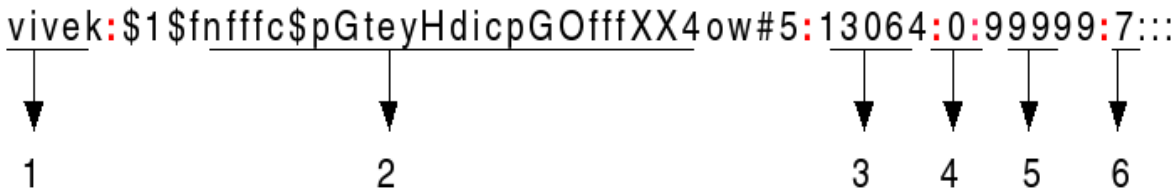
Les consignes :

3.1 – Etude de contenu du fichier shadow.txt :

3.1.1 - Rappels Théoriques:

Le fichier etc / shadow utilisé avec les systèmes de type Linux ou unix, stocke en toute sécurité les mots de passe hachés (une forme d'écriture) à partir d'un compte d'utilisateur. Pour chaque utilisateur répertorié, le fichier contient une seule entrée à chaque ligne. Chaque champ trouvé dans le mot de passe saisi est séparé par deux points ":".



1. ***Nom d'utilisateur**: C'est votre nom d'utilisateur.*
2. ***Mot de passe** : Le mot de passe crypté. Le mot de passe doit comporter entre 8 et 12 caractères, y compris les caractères spéciaux, les chiffres, les caractères alphabétiques en minuscules, etc. Habituellement, le format du mot de passe est défini sur $ id $ salt $ hashed, L'identifiant $ id est l'algorithme utilisé sur GNU / Linux comme suit:*

1. ***1** est MD5*
2. ***$2a$** est Blowfish*
3. ***$2y$** est Blowfish*
4. ***5** est SHA-256*
5. ***6** est SHA-512*

3. ***Dernier changement de mot de passe (dernier changement):*** *Days since Jan 1, 1970 that password was last changed*
4. ***Minimum*** *: Le nombre minimal de jours requis entre les changements de mot de passe, c'est-à-dire le nombre de jours restants avant que l'utilisateur ne soit autorisé à changer son mot de passe*
5. ***Maximum*** *: Le nombre maximal de jours de validité du mot de passe.*
6. ***Warn*** *: Le nombre de jours avant que le mot de passe expire, l'utilisateur est averti que son mot de passe doit être change.*
7. ***Inactive*** *: Le nombre de jours après l'expiration du mot de passe, ce compte est désactivé.*
8. ***Expire*** *: jours depuis le 1er janvier 1970, ce compte est désactivé, c'est-à-dire une date absolue spécifiant quand le login ne peut plus être utilisé*

Le mot de passe crypté comprend de 13 à 24 caractères de l'alphabet de 64 caractères a à z, A à Z, de 0 à 9, \. et /. En option, il peut commencer avec un caractère "$". Cela signifie que le mot de passe crypté a été généré en utilisant un autre algorithme (pas DES). Par exemple, s'il commence par "$ 1 $", cela signifie que l'algorithme basé sur MD5 a été utilisé. Veuillez noter qu'un champ de mot de passe qui commence par un point d'exclamation (!) Signifie que le mot de passe est verrouillé. Les caractères restants sur la ligne représentent le champ de mot de passe avant que le mot de passe ne soit verrouillé.

3.2- Mise en œuvre d'un script d'attaque par brute force.

3.2.1 – Rappels Théoriques

Une attaque par force brute est une méthode d'essai et d'erreur utilisée pour obtenir des informations telles qu'un mot de passe d'utilisateur ou un numéro d'identification personnel (PIN). Dans une attaque par force brute, un

logiciel automatisé est utilisé pour générer un grand nombre de suppositions consécutives quant à la valeur des données désirées. Les attaques par force brute peuvent être utilisées par des criminels pour déchiffrer des données cryptées, ou par des analystes de sécurité pour tester la sécurité réseau d'une organisation.

Une attaque par force brute est également connue sous le nom de crack de force brute ou simplement force brute.

Un exemple d'un type d'attaque par force brute est connu sous le nom d'attaque par dictionnaire, qui peut essayer tous les mots d'un dictionnaire. D'autres formes d'attaque par force brute peuvent essayer des mots de passe couramment utilisés ou des combinaisons de lettres et de chiffres.

Une attaque de cette nature peut prendre beaucoup de temps et de ressources. D'où le nom "attaque par force brute"; le succès est généralement basé sur la puissance de calcul et le nombre de combinaisons essayées plutôt que sur un algorithme ingénieux.

Les mesures suivantes peuvent être utilisées pour se défendre contre les attaques par force brute:

- *Obliger les utilisateurs à créer des mots de passe complexes*
- *Limitation du nombre de tentatives de connexion infructueuses de connexion d'un utilisateur*
- *Verrouillage temporaire des utilisateurs qui dépassent le nombre maximal de tentatives de connexion infructueuses.*

Qualités:

Le nombre de tentatives est limité par la longueur maximale et le nombre de caractères à essayer par position (ou octet si l'on considère les mots de passe Unicode)

Le temps pour terminer est plus grand, mais il y a une plus grande couverture de la valeur probable du texte en clair

(toutes les possibilités seulement si elles sont définies sur la longueur maximale et tous les caractères possibles sont pris en compte dans chaque position)

Le nombre de tentatives est limité par la longueur maximale et le nombre de caractères à essayer par position (ou octet si l'on considère les mots de passe Unicode)

Le temps pour terminer est plus grand, mais il y a une plus grande couverture de la valeur probable du texte en clair (toutes les possibilités seulement si elles sont définies sur la longueur maximale et tous les caractères possibles sont pris en compte dans chaque position).

3.2.1 - Mise en œuvre du script :

Le script se trouve dans le dépôt github du lien fourni dans l'e-mail.

3.3- Mise en œuvre d'un script d'attaque par dictionnaire :

3.3.1 – Rappels Théoriques

Une attaque par dictionnaire est une technique ou une méthode utilisée pour enfreindre la sécurité informatique d'une machine ou d'un serveur protégé par mot de passe. Une attaque de dictionnaire tente de vaincre un mécanisme d'authentification en entrant systématiquement chaque mot dans un dictionnaire comme un mot de passe ou en essayant de

déterminer la clé de déchiffrement d'un message ou d'un document chiffré.

Les attaques de dictionnaire réussissent souvent car de nombreux utilisateurs et entreprises utilisent des mots ordinaires comme mots de passe. Ces mots ordinaires sont facilement trouvés dans un dictionnaire, tel qu'un dictionnaire anglais.

Qualités:

Le dictionnaire ou les combinaisons possibles sont basées sur des valeurs probables et tendent à exclure les possibilités éloignées. Il peut être basé sur la connaissance d'informations clés sur une cible particulière (noms des membres de la famille, anniversaire, etc.). Le dictionnaire peut être basé sur des motifs observés sur un grand nombre d'utilisateurs et de mots de passe connus (par exemple, quelles sont les réponses les plus probables au niveau mondial). Le dictionnaire est plus susceptible d'inclure des mots réels que des chaînes de caractères aléatoires.

Le temps d'exécution de l'attaque par dictionnaire est réduit car le nombre de combinaisons est limité à celles de la liste des dictionnaires

Il y a moins de couverture et un mot de passe particulièrement bon peut ne pas figurer sur la liste et sera donc manqué.

Le principal compromis entre les deux attaques est la couverture contre le temps à compléter. Si vous avez une réflexion raisonnable sur ce que sera le mot de passe, vous pouvez ignorer les réponses improbables et obtenir une réponse plus rapidement. Ceci est important parce que les

mots de passe sont souvent sujets à changement et parce que plus la longueur du mot de passe augmente, plus le temps de les deviner augmente vraiment vite.

Essai de compréhension pour le script d'attaque du dictionnaire.

Part 1

Le code source se trouve dans le lien du référentiel GitHub envoyé avec l'email.

Nous commençons par importation de ces deux modules:

Hashlib:- *nous aidera à hacher et à saler sur un système d'exploitation Windows.*

Time:- *nous permettra de surveiller le temps nécessaire à l'exécution du programme.*

```
import hashlib
import time
```

Nous affectons ensuite nos fichiers texte aux variables que nous allons injecter dans le code. Cela permet de gagner du temps lors de la modification des fichiers que nous voulons déchiffrer. Les valeurs peuvent simplement être modifiées en haut du fichier sans avoir besoin de changer le corps principal du code.

shadowFile:- contient les mots de passe cryptés.

DictionaryFile:- contient une liste de mots de passe qui doivent être chiffrés et comparés au chiffrement shadowFile.

outputFile:- détient les résultats des mots de passe décryptés trouvés.

```
shadowFile = 'shadow.txt'
DictionaryFile = 'dico_mini_fr.txt'
outputFile = 'decrypted_passwords.txt'
```

Nous allons maintenant stocker 3 types de fonctions de cryptage fournies par le module hashlib, chacune représentée par une propriété numérique qui se trouve également dans certains des mots de passe cryptés dans shadow.txt. Si le nombre est 1, 5 ou 6, nous utiliserons sa valeur de cryptage correspondante pour hacher le code respectif en question. La définition d'un dictionnaire avec ces paires de clés et de valeurs en haut du code nous permet de modifier ou d'ajouter facilement au dictionnaire si nécessaire. Cela nous aide également à être dynamiques dans nos codes par les référençant des variables plutôt que des noms de fichiers exacts.

```
type_crypt = {   1:hashlib.md5, 5:hashlib.sha256,
6:hashlib.sha512 }
```

Nous voulons être lire le fichier shadowFile dans le dossier du projet et pour ce faire, nous commente et définissons une fonction appelée get_passwords.

```
def get_passwords():
```

Définir une variable de dictionnaire qui doit être utilisée pour collecter tout le code renvoyé par le programme.

```
  userDictionary = {}
```

Ensuite, nous allons utiliser un bloc try et placer notre code à l'intérieur. Ceci est utilisé pour assurer le traitement des exceptions d'erreur.

```
try:
```

Pour commencer le processus de lecture, nous définissons une variable qui contient la méthode open (). Cette méthode nécessite deux paramètres: le nom du fichier (transmis en tant que variable) à lire et la commande "r" qui indique au lecteur de fichiers de lire le contenu du fichier.

```
pass_file = open(shadowFile, "r")
```

Maintenant que le programme est conscient d'un besoin de lire dans le fichier shadow.txt, nous lui dit des comment lire le fichier, en déclarant une nouvelle variable lines, nous avons stocker à l'intérieur la variable pass_file que nous avons déclarée précédemment, puis enchaîner la méthode readlines (). Cela permet au programme de lire le fichier shadow.txt ligne par ligne et les séparant dans une chaîne de caractères située dans une liste.

```
lines = pass_file.readlines()
```

Suite à cela, nous devons maintenant utiliser la liste générée et conservée dans la variable de lines, dans l'utilisation dans une boucle pour. Pour chaque ligne trouvée dans les variables lines, nous convenons un certain traitement contre les données.

```
for line in lines:
```

Nous savons que nous devons diviser chaque line à chaque intersection où un deux-points est trouvé. Nous pouvons facilement le faire en déclarant une variable appelée tab et en demandant au programme de diviser chaque line avec la méthode split (). Nous entrons dans le colon entre les points d'exclamation en tant que paramètre, ce qui divise chaque ligne en une chaîne de caractères où chaque colon a été trouvé. Nous avons maintenant chaque ligne et sa chaîne de caractères correspondante dans sa propre liste [] et imprimée sur sa propre nouvelle ligne.

```
tab = line.split(':')
```

Utilisation du bloc de code if pour implémenter certaines conditions. Le besoin de ces conditions vient du fait qu'un mot de passe contenant les caractères "!" et "" dans*

les caractères suivants un nom d'utilisateur et précédant le hachage ou simplement de position [1], sont considérés indéchiffrables. Nous voulons également nous assurer qu'aucune liste vierge n'est pas qualifié pour la prochaine partie du programme. Si la longueur (nous utilisons la méthode len(tab) pour entourer tab variable contenant la line.split (‘ : ’)) à la variable tab est supérieure à un et les caractères dans la position tab[1] à la liste tab[] ne contient pas le caractère "!" ou "" pour que la ligne évalue à vrai et se qualifie pour l'étape de traitement suivant dans le programme.*

```
if len(tab) > 1 and tab[1] not in ['!','*']:
```

De ce point, nous nous retrouvons avec les mots de passe qui correspondent à nos conditions et ils devraient ressembler à ce qui suit: -

['root', '$1$934b4a210c17493f68bf6bfe74bff77a', '16749', '0', '99999', '7', '', '', '\n'].

Nous accédons à chaque chaîne de caractères en utilisant tab[0], tab[1], etc. L'indexation est pertinente pour sa position dans la liste. La première chaîne étant tab[0] et la dernière étant tab[8]. Nous avons seulement besoin des deux premiers indices pour compléter notre tâche. Nous commençons notre traitement des données ci-dessus en plaçant le nom d'utilisateur situé à tab[0] dans une variable nommée correctement l'utilisateur.

```
user = tab[0]
```

Nous n'avons pas encore complètement séparé le hachage correctement comme montré ici: '1 934b4a210c17493f68bf6bfe74bff77a'.
Si nous regardons de près la chaîne de caractères située à tab[1], nous trouvons $ 1 $ précédant la chaîne. Ceci est corrélé au dictionnaire type_crypt déclaré et situé en haut de notre code. Ce nombre placé entre les signes $ est le type de cryptage utilisé pour le hachage du mot de passe. Nous devrons séparer notre nombre de ces signes $ afin que notre programme puisse lire quel mode de cryptage est nécessaire.

Nous utilisons le méthode split encore en l'enchaînant à tab[1] index, la position où il se trouve dans la variable tab[]. Nous passons la méthode le symbole $ en paramètre.

```
crypt = tab[1].split('$')
```

Maintenant, notre type de hachage et de mot de passe haché est déplacé vers la variable crypt qui est elle-même une liste de chaînes de caractères. Il devrait apparaître comme tel:
['', '1', '934b4a210c17493f68bf6bfe74bff77a'].
La liste a trois indices et la première crypt[0] est une chaîne vide résultant de la méthode .split ('$'). Il enlève le caractère mais l'espace sur lequel il a existé continuera à exister. Nous n'aurons besoin que des deux derniers indices de la liste crypt: crypt[1] et crypt[2].

En continuant, nous utilisons la variable appelée userDictionary que nous avons définie au début de la fonction. Il doit prendre la variable utilisateur comme clé et les indices crypt[1] et crypt[2] placés dans une liste comme valeur pour chaque utilisateur. La syntaxe utilisée pour y parvenir est la suivante:

```
userDictionary[user] = [crypt[1], crypt[2]]
```

Nous allons maintenant utiliser la méthode d'impression pour imprimer sur la console. A l'intérieur de la parenthèse, nous utilisons des objets vides pour prendre
avantage de l'interpolation en combinaison avec la méthode .format (). Nous plaçons simplement les variables que nous souhaitons utiliser dans l'ordre dans lequel nous voulons qu'elles apparaissent et elles seront interpolées dans les objets vides précédant la méthode .format ().
Le code ci-dessous va imprimer: - [*] Mot de passe de Cracking For = root: 1,934b4a210c17493f68bf6bfe74bff77a

```
print('[*] Cracking Password For =  {}: {},{}'.format(user,
crypt[1], crypt[2]))
```

Nous pouvons maintenant retourner userDictionary en écrivant le code suivant. Si nous imprimions userDictionary, il retournait à la console le premier userDictionary puis, en raison de la boucle for, itérait sur le prochain userDictionary en l'ajoutant au premier, en continuant de cette manière jusqu'à ce que tous les userDictionary aient été itérés. Nous somme laissant avec: -

```
{'root': ['1', '934b4a210c17493f68bf6bfe74bff77a']
}
{'root': ['1', '934b4a210c17493f68bf6bfe74bff77a'],
'fred': ['1', '9ebf8e708dcb3f28cb43d5d52655ab14']
}
{'root': ['1', '934b4a210c17493f68bf6bfe74bff77a'],
'fred': ['1', '9ebf8e708dcb3f28cb43d5d52655ab14'],
'giselle': ['1', '6e5fa4d9c48ca921c0a2ce1e64c9ae6f']
}
```

```
      return (userDictionary)
```

Ce dernier bloc de code complète la première fonction dont nous avons besoin.
Il est implémenté en même temps et fait en effet partie du bloc de code try. Il traite des exceptions d'erreur dans python et note simplement le type d'erreur à attendre et quoi imprimer si ce type d'erreur est trouvé.

```
except FileNotFoundError:
      print("Error: Can't open file")
```

Part 2

Nous sommes maintenant arrivés au point où nous décrypterons réellement les mots de passe. La fonction précédente traitait de la mise en forme des résultats dans un dictionnaire python qui nous permettrait de gérer le décryptage avec une relative facilite.

Premièrement, nous définissons la fonction decrypt_shadow () dans python en utilisant le mot-clé def. Après, nous passons userDictionary en tant que paramètre. N'oubliez pas userDictionary est le résultat de la fonction get_passwords () et contient toutes les données nécessaires pour trouver les mots de passe.

```
def decrypt_shadow(userDictionary):
```

Comme dans la première fonction que nous avons créée, nous devons déclarer le passwordDictionary qui doit être utilisé pour stocker le résultat final de notre fonction.

```
   passwordDictionary = {}
```

Immédiatement après, nous implémentons un bloc try avec notre exception d'erreur située plus bas.

```
try:
```

Une fois de plus le programme est alerté de la tâche d'ouverture d'un fichier avec la méthode open (). Cette fois, nous passons la variable DictionaryFile qui correspond à 'dico_mini_fr.txt' comme indiqué en haut du code. Nous allons de nouveau, utiliser l'option "r" pour alerter le programme que nous avons l'intention de lire le texte dans le fichier indiqué. Tout ceci doit être placé dans la variable dictionary_file.

```
      dictionary_file = open(DictionaryFile, "r")
```

Nous allons maintenant utiliser le module de time que nous avons importé au tout début de notre code. Ici, nous demandons au module time d'utiliser sa méthode time () pour simplement commencer à garder l'heure et la placer dans la variable timeStart.

```
timeStart = time.time()
```

Maintenant, nous allons construire un bloc de code for-loop demandant au programme de lire chaque ligne située dans le fichier dictionary_file.

```
for line in dictionary_file:
```

Cette partie suivante du code est importante car elle nous permet d'avoir chaque mot de passe situé dans le dictionary_file imprimé sur une nouvelle ligne. Nous y parvenons en utilisant la méthode replace() chaînée à la variable line et en lui passant deux paramètres. Le premier étant la chaîne de caractères que nous souhaitons remplacer et ensuite avec ce que nous aimerions le remplacer avec. Nous passons le caractère "\ n" newline et souhaitons qu'il soit remplacé par une chaîne vide. Cela imprimera alors une nouvelle ligne pour chaque ligne lue sans le "\ n" placé à la fin, ce qui provoquerait sans aucun doute l'échec de notre décryptage.

```
password = line.replace("\n", "")
```

En allant de l'avant, nous allons maintenant incorporer une autre boucle for pour commencer à parcourir les éléments du userDictionary. Nous voulons accéder à la fois à la propriété de l'utilisateur et aux valeurs d'ombre contenues dans userDictionary. Nous utiliserons donc les variables user et shadow comme des itérations dans la boucle for. Enfin, nous chaînons la méthode .items() à la fin de userDictionary pour pouvoir itérer sur la clé et les valeurs de userDictionary.

```
for user, shadow in userDictionary.items():
```

De plus, nous allons ajouter quelques conditions à la boucle for avec un bloc de code if. La condition indique que si l'int (shadow [0]), signifiant la clé dans le shadow trouvée au premier index avec un cast de type int changeant la valeur d'une chaîne de caractérs en un integer est trouvé dans type_crypt.keys(), alors le la condition est vraie et les données peuvent continuer à être traitées. La méthode .keys() permet d'itérer sur les propriétés des variables clés, ayant défini type_crypt au tout début de notre code, type_crypt.keys() nous permettra de savoir si le type de cryptage présent dans la variable shadow est présent dans le dictionnaire type_crypt lui-même. Cela permettra à notre ombre d'être décryptée.

```
if int(shadow[0]) in type_crypt.keys():
```

Now that we know our data has indeed a matching value at both the **shadow[0]** *index and* **type_crypt.keys()** *property values we assign* **type_crypt[int(shadow[0])]** *to the variable* **crypt.**
The key value of **type_crypt[int(shadow[0])]** *is now equal to* **int(shadow[0])** *and the built in function* **md5** *given to us by the module* **hashlib** *and used in the* **type_crypt** *at the beginning of our code. The code below gives us the type of cryptage needed for each iteration of the* **DictionaryFile.**

```
crypt = type_crypt[int(shadow[0])]
```

Maintenant que nous savons que nos données ont en effet une valeur correspondante à la fois pour les valeurs de propriété shadow[0] et type_crypt.keys(), nous assignons type_crypt [int (shadow[0])] à la variable crypt.
La valeur clé de type_crypt [int (shadow[0])] est maintenant égale à int (shadow[0]) et la fonction intégrée md5 qui nous avons est donnée par le module hashlib et utilisée dans le type_crypt au début de notre code . Le code ci-dessous nous donne le type de cryptage nécessaire pour chaque itération du DictionaryFile.

```
crypt_password = crypt(password.encode('utf- 8')).hexdigest()
```

Nous allons maintenant entrer notre condition if final, qui stipule que si le hash crypté de crypt_password est égal à celui trouvé dans l'index shadow[1] est vrai, nous continuons avec les données à traiter.

```
if (crypt_password == shadow[1]):
```

Ici, nous commençons par dire au module de temps que nous avons importé d'arrêter le chronométrage.

```
timeEnd = time.time()
```

Ici, nous déclarons que la propriété user dans passwordDictionary est égale au mot de passe trouvé et aux valeurs de début timeEnd - time. Cette section du code nous donnera le mot de passe et le temps nécessaire pour trouver le mot de passe dans une liste python, pour chaque user dans le passwordDictionary.

```
passwordDictionary[user] = [password ,   float(timeEnd - timeStart)]
```

*Its now time to print out our data to the console. As in the function **get_passwords** we will use the **print()** method and the **.format()** to take advantage of the interpolation capabilities within python. The empty objects found in the **print()** method Will be populated by in order of appearance by the variables in the **format()** method.*
Il est maintenant temps d'imprimer nos données sur la console. Comme dans la fonction get_passwords, nous utiliserons la méthodes print() et .format() pour tirer des capacités d'interpolation dans python. Les objets vides trouvés dans la méthode print(),sera rempli par ordre d'apparition des variables de la méthode format ().

```
print ("Found password: \"{}\" for user {}".format(password, user))
```

Ici, nous retournons simplement le passwordDictionary qui contient les utilisateurs, leurs mots de passe trouvés et le temps nécessaire pour trouver les mots de passe.

```
        return (passwordDictionary)
```

C'est le gestionnaire d'exception d'erreur implémenté lorsque nous avons d'abord construit le bloc try pour cette fonction. Il nous informe du type d'exception attendu et de ce qu'il faut imprimer si l'exception est présente.

```
        except FileNotFoundError:
              print("Error: Can't open file")
```

Part 3

Enfin, nous arrivons à la dernière étape de notre code. Ici nous allons définir une fonction file_write qui prendra la variable passwordDictionary générée par la fonction get_passwords que nous venons de compléter en paramètre.

```
        def file_write(passwordDictionary):
```

Tout d'abord, nous implémentons un bloc try et son exception error située un peu plus bas dans le code.

```
try:
```

En définissant une variable nommée output, nous utiliserons à nouveau la méthode open () pour créer et ouvrir le fichier outputFile. En passant le "w" comme deuxième paramètre, nous alertons le programme que nous souhaitons sur la variable de sortie.

```
            output = open(outputFile, "w")
```

Nous passons maintenant à créer une autre for-loop qui doit prendre deux variables d'itérateur utilisateur et mot de passe. Nous utilisons le passwordDictionary avec la méthode items () chaînée, nous permettant de parcourir les paires clé-valeur de chaque utilisateur et mot de passe trouvé dans userDictionary.

```
        for user ,password in passwordDictionary.items():
```

En continuant avec notre code, nous mettons maintenant à l'intérieur de la variable out, la chaîne que nous souhaitons imprimer. Nous utilisons les capacités d'interpolation de la méthode .format() pour remplir les objets vides de la chaîne par l'ordre trouvé dans la méthode .format(). Nous passons l'utilisateur, le mot de passe, et le temps a fallu trouver le mot de password[1] à notre chaîne.

```
            out = "user: {}, password: {}, time:
{}'s".format(user, password[0], round(password[1], 4))
```

Here we shall use the **write()** *method chained to the output variable to write the contents of the* ***out*** *variable to the* **outputFile** *with a linebreak at the end of each string formulated from* ***out****.*

Ici, nous allons utiliser la méthode write () chaînée à la variable de sortie pour écrire le contenu de la variable out dans le fichier outputFile avec un saut de ligne à la fin de chaque chaîne formulée à partir d'out.

```
                output.write(out + '\n')
```

C'est l'exception d'erreur qui indique l'erreur attendue et ce qu'il faut imprimer si l'erreur est détectée.

```
    except FileNotFoundError:
        print("Error: Can't write file")
```