Projet ISN : le chiffrement affine

L'objectif est de mettre en place un projet qui ne devrait pas poser de difficultés dans l'élaboration du programme. Ce projet permet de couvrir de nombreuses notions d'informatique.

La cryptographie est, en soi, une notion importante dans la transmission d'informations.

La partie théorique repose sur des notions élémentaires d'arithmétique (suffisamment simples pour ne pas nécessiter de rentrer dans des notions rapidement trop compliquées).

Ceci permet la mise en place et l'étude de certains algorithmes fondamentaux (liés à la notion de divisibilité) et nous permettra d'envisager la réalisation d'un programme simple en langage assembleur.

Enfin, il sera proposé deux challenges liés à la cryptographie qui permettront à l'élève de se confronter à certaines notions importantes de la cryptographie.

Sommaire

Cahier des charges

Production finale attendue

Un exécutable permettant de chiffrer un message à l'aide de la technique du chiffrement affine.

Caractéristiques de la production finale

- En entrée : le message à coder (chaîne de caractère en majuscule, sans espace et sans ponctuation), la clé de chiffrement (deux entiers *a* et *b*)
- En sortie : le message codé (chaîne de caractère en majuscule, sans espace et sans ponctuation) **et** la clé de déchiffrement (deux entiers *c* et *d*)

Contraintes à respecter

- Il faudra rendre un exécutable exploitable sur au moins deux OS (dont l'une sera pour Windows 32 bits)
- Rendre un dossier (au plus 10 pages) résumant le déroulement du projet.
  - Dans une partie commune : présentation du projet, démarche collaborative.
  - Dans une partie individuelle : chaque membre du groupe devra y détailler sa participation, son implication, ses difficultés, ses impressions.

Matériel et logiciel à mettre en œuvre

- Le langage de programmation est Python 3
- On utilisera cx_Freeze pour obtenir un exécutable

Composantes

Compétence	Place dans le projet

Dimension algorithmique	• Étude d'algorithmes
	  ◦ Quotient ou reste
	• Construction algorithme
	  ◦ Division euclidienne
	• Étude de la validité de l'algorithme
	• Complexité d'un algorithme (cf Challenge 2)

Éléments de programmation	• Typage des données (entiers et caractères) • Opérations élémentaires (+, *, /, %) • Fonctions : `input`, `print`, `ord`, `chr`, • Instructions : `if`, `for`, `while` • Étude du programme permettant d'effectuer le chiffrement de César • Élaboration de programmes : déchiffrement César, chiffrement affine, déchiffrement affine • Tests du programme principal (validité de la clé, fonctionnement du programme) • Mise en place de programmes nécessaires à la résolution des Challenges
Architecture	• Capacité des ordinateurs (cf Challenge 2) • Mise en place d'un programme en assembleur pour tester la divisibilité d'un nombre par un autre • Création d'un exécutable (architecture et portabilité)
Représentation de l'information	Codage des caractères (ASCII), cryptographie
Utilisation des réseaux	Protocoles de chiffrement (SSL/TLS)
Droits et responsabilité	Pourquoi est-ce important de crypter les données ? Quels-en sont les enjeux ?

Chronogramme

Déjà été vu :

- Notions d'algorithmique (au lycée et en ISN)
- Notions de programmation `input`, `print`, `if`, `for`, `while` (en cours d'ISN)
- types de données (entiers, caractères) (en cours d'ISN)
- codage ASCII (en cours d'ISN)
- opérations arithmétiques élémentaires (+, *, /, %) (en cours d'ISN)
- fonctionnement langage machine/assembleur (en cours d'ISN)

Séance	Information	Algorithmique	Programmation	Architecture	Travail à réaliser
Séance 1 cours	Éléments de cryptographie	Importance du modulo			Exposé : pourquoi faut-il crypter les informations ?
Séance 2 TP			- Notions nécessaires à l'élaboration du programme - Écriture du script du chiffrement de César		Exposé : présenter brièvement le protocole SSL
Séance 3 TP			Programme de chiffrement de César		Envoyer un message codé à un collègue avec la clé de chiffrement. Le collègue doit déchiffrer le message
Séance 4 cours	Déchiffrement (réversibilité du processus)	- Étude d'algorithmes quotient ou reste - Élaboration algorithme division euclidienne	Programme de déchiffrement de César		Résoudre le challenge 1
Séance 5 TP	Un message crypte peut-il toujours se déchiffrer ? Comment faire sans la clé (attaque)		Démarrer le projet en s'inspirant du chiffrement de Cesar, le script s élabore facilement		Élaborer un programme en assembleur pour tester la divisibilité
Séance 6 cours		Algorithme de divisibilité	- Continuation - Premiers tests (certaines valeurs de *a* ne conviennent pas)	Discussion autour du programme assembleur	Résoudre le challenge 2
Séance 7 TP	Le challenge permet de discuter de la solidité des méthodes de chiffrement.	Validité d'un algorithme (division euclidienne)	- Programme de déchiffrement (calculer l'inverse de *a* modulo 26) - conditions sur la clé		Avancer sur le projet
Séance 8 TP			Projet		Fournir un exécutable
Séance 9 TP			Projet	Création d'un exécutable, portabilité	Fournir un compte rendu
Séance 10 TP			Présentation du projet Bilan		

Chiffrement

Principe : on utilisera un chiffrement par substitution monoalphabetique.
Règle : Les messages seront écrits en majuscules, sans espace et sans ponctuation.
On va donc raisonner sur des entiers entre 65 et 90 (codage ASCII)

65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
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**

Chiffrement de César

Le chiffre de César ou code de César est un chiffrement par décalage. Elle consiste en une substitution de lettres par une autre plus loin dans l'alphabet.

Exemple :
Par exemple, si l'on utilise un décalage de 3, A serait remplacé par D, B deviendrait E, et ainsi de suite.

Lettre	**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**
nombre	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Chiffrement	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	***91***	***92***	***93***
modulo	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	***65***	***66***	***67***
Lettre cryptée	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**

Remarques :
3 est la **clé** du chiffrement
Lorsque la clé de codage vaut k, la fonction de codage est $f : [\![0;26]\!] \to [\![0;26]\!]$
$x \longmapsto (x+k) \mod 26$

Déchiffrement
Lorsqu'on connaît la clé k c'est très simple !
La fonction de décodage est $f : [\![0;26]\!] \to [\![0;26]\!]$
$x \longmapsto (x-k) \mod 26$

Chiffrement affine

Il s'agit ici de compliquer le chiffrement et le rendre donc plus sûr (le nombre de clé possible augmente et déchiffrement est plus complexe).
La fonction de codage affine est une fonction de chiffrement utilisant une clé composée de deux nombres (a ;b). La fonction de codage associée est alors :

$f : [\![0;26]\!] \to [\![0;26]\!]$
$x \longmapsto (a \times x+b) \mod 26$

Déchiffrement
C'est un peu plus délicat (il faut, en fait, que a soit inversible modulo 26).
Ce qui permet de réfléchir sur les techniques de chiffrement :

- il faut s'assurer que le processus soit réversible (ici si l'on choisit mal a, le message ne peut être décrypté !)
- La technique utilisée est-elle solide ?

Algorithmes

Autour de la division euclidienne.

Définition :

Soit a et b deux entiers naturels.
Il existe un unique couple $(q ; r)$ tel que $a = b \times q + r$ avec $0 \leqslant r < b$.

Quotient

Version multiplicative
Entrée : entiers a et b ($a > b$) *Sortie :* entier q (quotient de a par b)

`q ← 0`
`Tantque b x q <= a`
`    q ← q + 1`
`FinTantque`
`Affiche q - 1`

Reste

Utilisation du quotient	Par soustractions successives
Entrée : entiers a et b ($a > b$) *Sortie :* entier r (reste de a par b)	*Entrée :* entiers a et b ($a > b$) *Sortie :* entier r (reste de a par b)

`q ← 0`
`Tantque b x q <= a`
`    q ← q + 1`
`FinTantque`
`r ← a - b x (q - 1)`
`Affiche r`
	`r ← a`
	`Tantque r >= b`
	`    r ← r - b`
	`FinTantque`
	`Affiche r`

Quotient et reste

Entrée : entiers a et b ($a > b$) *Sortie :* entiers q et r (quotient de a par b)

`r ← a`
`q ← 0`
`Tantque r >= b`
`    r ← r - b`
`    q ← q + 1`
`FinTantque`
`Affiche q , r`

Étude de l'algorithme de la division euclidienne

Terminaison :
r est décrémenté à chaque itération.
Il deviendra donc inférieur à *b* (au bout de *b* itérations d'ailleurs).

Validité :
On veut établir que *a* est égal à $bq+r$ à chaque étape itération (invariant de la boucle)

- Avant la boucle $q=0$ et $r=a$ donc $a=bq+r$
- Supposons que $a=bq+r$ au début de l'itération, montrons que cette propriété reste vraie à la fin de chaque itération.
  Soit q' et r' les valeurs de q et r à la fin de l'itération.
  On a $q'=q+1$ et $r'=r-b$
  Donc $b\times q'+r'=b(q+1)+(r-b)=bq+b+r-b=bq+r$
  Et par hypothèse $a=bq+r$ donc $bq'+r'=a$
- La condition de sortie de boucle nous assure que $0\leqslant r<b$

Assembleur

Programme pour savoir si b divise a ($a \geq b$).

<table>
<tr><th>Algorithme</th><th>Programme assembleur</th></tr>
<tr><td>Entrée :
entiers a et b ($a \geq b$)
Sortie :
booléen divise
(Vrai si a divise b)</td><td>Situation initiale :
a est entré dans la case mémoire 100 et b dans la 101
Situation finale :
À la fin de l'exécution la cellule 102 contient la valeur 1 si b divise a et 0 sinon</td></tr>
<tr><td><pre>
c ← a-b
Si c==0 alors
    divise ← Vrai
FinSi
Tantque c > 0
    c ← c-b
    Si c==0 alors
        divise ← Vrai
    Sinon
        divise ← Faux
    FinSi
FinTantque

Affiche divise
</pre></td><td>
<table>
<tr><th>adresse</th><th>Instruction</th><th>Valeur</th><th>Commentaire</th></tr>
<tr><td>010</td><td>LDA</td><td>100</td><td>Charge la valeur de la case 100 (a) dans le registre</td></tr>
<tr><td>011</td><td>SUB</td><td>101</td><td>Soustrait la valeur de la case 101 (b) à la valeur du registre</td></tr>
<tr><td>012</td><td>BRZ</td><td>020</td><td>Si la valeur du registre vaut 0 va à l'adresse 20</td></tr>
<tr><td>013</td><td>BRP</td><td>011</td><td>Si la valeur du registre est strictement positive va à l'adresse 11</td></tr>
<tr><td>014</td><td>LDA</td><td># 0</td><td>Charge la valeur 0 dans le registre</td></tr>
<tr><td>015</td><td>STA</td><td>102</td><td>Stocke la valeur du registre à l'adresse 102</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>020</td><td>LDA</td><td># 1</td><td>Charge la valeur 1 dans le registre</td></tr>
<tr><td>021</td><td>STA</td><td>102</td><td>Stocke la valeur du registre à l'adresse 102</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>adresse</td><td>valeur</td><td></td><td></td></tr>
<tr><td>100</td><td>a</td><td></td><td>Contient la valeur de a</td></tr>
<tr><td>101</td><td>b</td><td></td><td>Contient la valeur de b</td></tr>
<tr><td>102</td><td></td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td></td><td>Valeur : Si # valeur qui suit sinon adresse de la valeur
LDA : charge le contenu dans le registre A
SUB : soustrait le nombre au registre A
BRP : saut conditionnel si valeur du registre A est positive saute à l'adresse indiqué
BRZ : saut conditionnel si valeur du registre A est nulle saute à l'adresse indiqué
STA : stocke le contenu du registre à l'adresse qui suit</td></tr>
</table>

Programmation : Chiffrement de César

Travail en commun sur le chiffrement de César.

Les fonctions *built-in* nécessaires

Type entier et caractère (pas de distinction entre chaîne et caractère pour Python)
Codage ASCII (Unicode pour Python mais ici sans importance)

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> type(68)    #renvoie le type de la donnée
<class 'int'>
>>> chr(68)     #Renvoie un caractère dont on connaît le code Unicode (ASCII)
'D'
>>> type(D)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    type(D)
NameError: name 'D' is not defined
>>> type("D")   #lors de l'utilisation de chaîne il faut utiliser les "_" ou '_'
<class 'str'>
>>> type('D')
<class 'str'>
>>> ord('D')    #Renvoie le code Unicode (ASCII) d'un caractère
68
>>> for i in range(5):
        print(chr(i+65))


A
B
C
D
E
>>> for i in range(26):    #affiche sans retour à la ligne
        print(chr(i+65),end="")


ABCDEFGHIJKLMNOPQRSTUVWXYZ
>>>
```

Les opérations

Reste (modulo) et concaténation

```
>>> 26%7
5
>>> 4%26
4
>>> (4+26)%26
4
>>> (4+59*26)%26
4
>>> for i in range(3):
        print((4+i*26)%26)


4
4
4
>>> mot=str()
>>> for i in range(26):    #on utilise la concatenation des chaines de caracteres
        mot=mot+chr(i+65)

>>> print(mot)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
>>>
```

Élaboration du script

```
mot=input("entrer le mot à coder : ")
cle=input("entrer la clé de codage : ")
k=int(cle)                              #par defaut le type entre est un str
for lettre in mot:                      #mot est un str et une liste (conteneur)
    nb=ord(lettre)-65                   #lettre devient nb
    nb_crypte=(nb+k)%26                 #chiffrement
    lettre_crypte=chr(nb_crypte+65)     #nombre devient lettre
    print(lettre_crypte,end="")         #affichage
```

```
>>> ================================ RESTART ================================
>>>
entrer le mot à coder : INFORMATIQUE
entrer la clé de codage : 4
MRJSVQEXMUYI
>>>
```

Fonction

Dans sa version fonction :

```
def cesar_chiffre_nb(x,k):                  #fonction qui effectue le chiffrement
    """Chiffrement monoalphabetique de x par y avec : y≡x+k[26]"""
    return (x+k)%26


def cesar_chiffre_mot(mot,k):
    """Chiffrement de Cesar. Le message est codé lettre par lettre"""
    mot_crypte=str()                        #creation d'une liste vide
    for lettre in mot:
        nb=ord(lettre)-65
        nb_crypte=cesar_chiffre_nb(nb,k)    #appel de la fonction définie precedemment
        lettre_crypte=chr(nb_crypte+65)
        mot_crypte=mot_crypte+lettre_crypte #utilisation de la concatenation
    return(mot_crypte)
```

```
>>> ================================ RESTART ================================
>>>
>>> cesar_chiffre_mot("INFORMATIQUE",4)
'MRJSVQEXMUYI'
>>>
```

Challenges : Attaque du chiffrement

Challenge 1
Déchiffrer le message suivant (il a été crypté à partir d'un chiffrement de César) :

PUMVYTHAPXBLLAZJPLUJLKBUBTLYPXBL

Solution
Idée : l'élève doit élaborer un programme qui teste avec 26 valeurs pour le décalage et observer le plus probable. Il s'agit ici d'une recherche exhaustive.

```
>>> attaque_cesar("PUMVYTHAPXBLLAZJPLUJLKBUBTLYPXBL")
PUMVYTHAPXBLLAZJPLUJLKBUBTLYPXBL
OTLUXSGZOWAKKZYIOKTIKJATASKXOWAK
NSKTWRFYNVZJJYXHNJSHJIZSZRJWNVZJ
MRJSVQEXMUYIIXWGMIRGIHYRYQIVMUYI
LQIRUPDWLTXHHWVFLHQFHGXQXPHULTXH
KPHQTOCVKSWGGVUEKGPEGFWPWOGTKSWG
JOGPSNBUJRVFFUTDJFODFEVOVNFSJRVF
INFORMATIQUEETSCIENCEDUNUMERIQUE
HMENQLZSHPTDDSRBHDMBDCTMTLDQHPTD
GLDMPKYRGOSCCRQAGCLACBSLSKCPGOSC
FKCLOJXQFNRBBQPZFBKZBARKRJBOFNRB
EJBKNIWPEMQAAPOYEAJYAZQJQIANEMQA
DIAJMHVODLPZZONXDZIXZYPIPHZMDLPZ
CHZILGUNCKOYYNMWCYHWYXOHOGYLCKOY
BGYHKFTMBJNXXMLVBXGVXWNGNFXKBJNX
AFXGJESLAIMWWLKUAWFUWVMFMEWJAIMW
ZEWFIDRKZHLVVKJTZVETVULELDVIZHLV
YDVEHCQJYGKUUJISYUDSUTKDKCUHYGKU
XCUDGBPIXFJTTIHRXTCRTSJCJBTGXFJT
WBTCFAOHWEISSHGQWSBQSRIBIASFWEIS
VASBEZNGVDHRRGFPVRAPRQHAHZREVDHR
UZRADYMFUCGQQFEOUQZOQPGZGYQDUCGQ
TYQZCXLETBFPPEDNTPYNPOFYFXPCTBFP
SXPYBWKDSAEOODCMSOXMONEXEWOBSAEO
RWOXAVJCRZDNNCBLRNWLNMDWDVNARZDN
QVNWZUIBQYCMMBAKQMVKMLCVCUMZQYCM
>>>
```

Challenge 2
Le nombre n suivant est le produit de deux nombres premiers $n = pq$ avec p plus petit que q

n=262158157939114458143411

Sachant que p et q sont assez proches l'un de l'autre, que vaut p ?

Solution
Idée : l'élève doit élaborer un programme (simple mais chronophage) qui teste la divisibilité de n

```
>>> facteur(262158157939114458143411)
490586350739
```

Remarques :

- Le programme tourne vraiment longtemps (quelques heures, voire toute la nuit).
- Ceci permet d'évoquer les notions de :
  - Complexité (temporelle) d'un algorithme : ici l'algorithme est très simple à mettre en place mais il faut un temps très long pour obtenir un résultat.
  - Comment rendre un algorithme plus performant ?
    Ici on peut partir de la racine carrée, on peut ne tester que sur les nombres impairs, ...
- Permet d'évoquer le chiffrement RSA.

Projet

L'objectif reste le programme de chiffrement affine. Il faut toutefois clairement se poser la question du déchiffrement. On peut envisager un programme plus complet (chiffrement et/ou déchiffrement et/ou attaque) selon les niveaux d'avancement dans le projet.

Élaboration du programme de chiffrement

Ceci devrait se faire rapidement à partir de ce qui a été vu sur le chiffrement de César.

Les entrées du programme

- On ne s'intéresse qu'aux messages en majuscules, sans espace et sans ponctuation (on peut envisager un test pour vérifier que le message respecte les contraintes et pour les plus avancés un petit programme permettant de transformer un texte : suppression des espaces, changement de la casse).
- La clé $(a\,;b)$: on commence à se poser le problème du déchiffrement (on peut envisager un test pour vérifier que la clé permet le déchiffrement).

Déchiffrement

- Conditions pour déchiffrer le message.
  Le programme de déchiffrement est le même. Il suffit d'utiliser la clé de déchiffrement.
- La clé de déchiffrement.
  Il n'est pas nécessaire de rentrer dans les détails (il s'agit du calcul d'un inverse modulo 26 qui nécessiterait d'utiliser le théorème de Bezout). L'idée sera ici de dire que ça ne marche pas toujours et lorsque l'inverse existe, on le trouve par une recherche exhaustive.

L'exécutable

- Utilisation du programme de son choix (ici j'ai utilisé cx_Freeze).
- Difficultés liées au logiciel, aux commandes, aux OS,...

Le script

```
"""
CHIFFREMENT AFFINE
Ce programme permet de crypter un message
Le message est en majuscule, sans espace et sans ponctuation
Chiffrement monoalphabetique de X par Y avec : Y≡a*X+b[26]
"""

mot=input("entrer le message a crypter : ")

erreur_typo=0                       #analyse des contraintes typographiques
for lettre in mot:
    if ord(lettre)<65 or ord(lettre)>90:
        erreur_typo+=1
if erreur_typo!=0:
    print("Le message doit être en majuscule, sans espace et sans ponctuation!")
else:

    #si la typographie du message permet le chiffrement
    cle_a=input("entrer la valeur de a : ")
    a=int(cle_a)
    if a%2==0 or a%13==0:           #test de la cle
        print("la valeur de a ne permet pas le dechiffrement!")

    #si la valeur de la cle permet le chiffrement
    else:
        cle_b=input("entrer la valeur de b : ")
        b=int(cle_b)

        mot_crypte=str()
        for lettre in mot:          #chiffrement du message
            nb=ord(lettre)-65
            nb_crypte=(a*nb+b)%26
            lettre_crypte=chr(nb_crypte+65)
            mot_crypte=mot_crypte+lettre_crypte

        for i in range(26):         #calcul de la cle de dechiffrement
            if (i*a)%26==1:
                a_inv=i%26
                b_inv=(-a_inv*b)%26

        #Affichage du resultat
        print("le message crypte est : ",mot_crypte,\
              "\n Secret : pour décrypter le message \n\
              utiliser le même programme avec la cle inverse : (",a_inv,";",b_inv,")")
```

IDLE Python

```
>>>
entrer le message a crypter : Science is what we understand well enough to explain to a
computer
Le message doit être en majuscule, sans espace et sans ponctuation!
>>>
================================ RESTART ================================
>>>
entrer le message a crypter : DONALDKNUTH
entrer la valeur de a : 8
la valeur de a ne permet pas le dechiffrement!
>>>
================================ RESTART ================================
>>> import Chiffrement_affine
entrer le message a crypter : SCIENCEISWHATWEUNDERSTANDWELLENOUGHTOEXPLAINTOACOMPUTER
entrer la valeur de a : 5
entrer la valeur de b : 1
le message crypte est :  NLPVOLVPNHKBSHVXOQVINSBOQHVEEVOTXFKSTVMYEBPOSTBLTJYXSVI
 Secret : pour décrypter le message
              utiliser le même programme avec la cle inverse : ( 21 ; 5 )
>>> help(Chiffrement_affine)
Help on module Chiffrement_affine:

NAME
    Chiffrement_affine

DESCRIPTION
    CHIFFREMENT AFFINE
    Ce programme permet de crypter un message
    Le message est en majuscule, sans espace et sans ponctuation
    Chiffrement monoalphabetique de X par Y avec : Y≡a*X+b[26]

DATA
    a = 5
    a_inv = 21
    b = 1
    b_inv = 5
    cle_a = '5'
    cle_b = '1'
    erreur_typo = 0
    i = 25
    lettre = 'R'
    lettre_crypte = 'I'
    mot = 'SCIENCEISWHATWEUNDERSTANDWELLENOUGHTOEXPLAINTOACOMPUTER'
    mot_crypte = 'NLPVOLVPNHKBSHVXOQVINSBOQHVEEVOTXFKSTVMYEBPOSTBLTJYXSVI'
    nb = 17
    nb_crypte = 8

FILE
    c:\python34\lib\idlelib\chiffrement_affine.py


>>>
```

Annexes

Création de l'exécutable pour Windows (32 ou 64 bits)

Utilisation de l'exécutable cxFreeze pour créer en .exe (à l'intérieur d'un dossier contenant le nécessaire) ou un .msi.
Deux possibilités :

- réaliser l'exécutable à partir du script Python : Chiffrement_affine.py
- construire l'exécutable par l'intermédiaire d'un script setup.py contenant des informations supplémentaires pour le programme

Fichier Setup.py

```
""" Fichier d'installation de notre script Chiffrement_affine.py """
from cx_Freeze import setup, Executable
# On appelle la fonction setup
setup (
        name = "Affine",
        version = "0.9",
        author = "Johann Dolivet",
        description = "Chiffrement affine",
        executables = [ Executable ("Chiffrement_affine.py")],)
```

Invite de commande windows pour un .exe

```
...
C:\Python34\Scripts>python setup.py build
running build
running build_exe
...

C:\Python34\Scripts>
```

Invite de commande windows pour un .msi

```
...
C:\Python34\Scripts>python setup.py bdist_msi
running bdist_msi
running build
running build_exe
...

C:\Python34\Scripts>
```

Exécution du programme :

```
entrer le message a crypter : INFORMATIQUE
entrer la valeur de a : 5
entrer la valeur de b : 7
le message crypte est :  VUGZOPHYVJDB
 Secret : pour décrypter le message
              utiliser le même programme avec la cle inverse : ( 21 ; 9 )
Appuyez sur une touche pour continuer...
```

Création de l'exécutable pour Linux (32 ou 64 bits)

L'exécutable a également été réalisé à partir de cx_Freeze (mais cette fois à partir de Python2).

Terminal pour la création de l'exécutable à partir de Chiffrement_affine.py (dans un dossier dist)

```
root@xubuntu:/home/xubuntu/Desktop/Scripts# cxfreeze Chiffrement_affine.py
creating directory /home/xubuntu/Desktop/Scripts3/dist
copying /usr/lib/pymodules/python2.7/cx_Freeze/bases/Console ->
/home/xubuntu/Desktop/Scripts3/dist/Chiffrement_affine
...
copying /usr/lib/python2.7/lib-dynload/bz2.x86_64-linux-gnu.so ->
/home/xubuntu/Desktop/Scripts/dist/bz2.x86_64-linux-gnu.so
root@xubuntu:/home/xubuntu/Desktop/Scripts#
```

Exécution du programme dans le terminal :

```
root@xubuntu:/home/xubuntu/Desktop/Scripts/dist# ./Chiffrement_affine
entrer le message a crypter : INFORMATIQUE
entrer la valeur de a : 5
entrer la valeur de b : 3
le message crypte est :  RQCVKLDURFZX
 Secret : pour decrypter le message
             utiliser le meme programme avec la cle inverse : ( 21 ; 15 )
root@xubuntu:/home/xubuntu/Desktop/Scripts/dist#
```

Commentaires généraux

Pour la partie théorique, c'est un thème que je connais. Cela fait 2 ans que j'enseigne la spécialité mathématiques en TS et j'ai eu l'occasion de bien étudié le thème de la cryptographie (chiffrement César, affine, Vigenère, Hill et RSA). Par ailleurs, j'ai suivi cette année un MOOC en relation avec le sujet qui m'a permis de créer des scripts et de me confronter aux problématiques liées à l'élaboration des scripts et à leurs contraintes. Ce MOOC m'a d'ailleurs donné l'idée des challenges (j'ai trouvé cette partie très stimulante).

Pour la partie algorithmique et assembleur, ce sont des algorithmes et programme que j'avais étudié ou réalisé cette année lors de la formation. Je cherchais un projet me permettant de traiter ces notions de base en mathématiques et surtout en informatique.

Pour la réalisation de l'exécutable j'ai essayé de réaliser des exécutables pour différentes architectures. Cette partie a été intéressante et m'a pris pas mal de temps. En effet, pour Windows ce fut rapide (mon ordinateur est un ordinateur 64 bits et fonctionne sous Windows et j'ai réalisé un exécutable en 32 bits sur un autre poste), mais je souhaitais vraiment créer un exécutable pour Linux (j'ai déjà eu l'occasion d'utiliser Linux). Je souhaite que la réalisation de l'exécutable soit un moment pour discuter de la portabilité des applications et des architectures 32 et 64 bits. J'ai donc essayé, à partir de machines virtuelles et de live-CD, de réaliser cet exécutable (toujours à partir de mon ordinateur). Ce qui m'a amené à un nouveau problème lié finalement à Python. Il est plus compliqué de construire un exécutable à partir d'un programme sur Python3 que sur Python2. Il a donc été plus simple pour moi d'adapter mon script (changement de `input` par `raw_input`, pas de parenthèses pour `print`, et j'ai enlevé tous les accents pour les problèmes d'encodage). Les exécutables pour Linux ont donc été fait à partir de scripts en Python2.