TD: sécurisation des communications:

Exercice 1: Consulter la page d'accueil du site de la société informatique de France à l'adresse suivante: `https://www.societe-informatique-de-france.fr/`.

Préciser les informations disponibles sur le certificat de sécurité, quels sont les chiffrements utilisés, symétrique et asymétrique.

Exercice 2: code César:
Le code César, ou chiffrement par décalage, est une des plus anciennes et l'une des plus simples méthodes de chiffrement d'un message. Elle consiste à décaler chaque lettre du message d'un nombre de lettres constant (appelé clé de chiffrement) dans l'alphabet latin: 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.

Exemple: La lettre A est codée par la lettre C avec la clé 2, Z est codée par B avec la même clé. L'alphabet est repris du début quand c'est nécessaire.

1. **Code César pour un mot écrit en majuscules:**
   Écrire une fonction qui retourne un mot (donné en argument) chiffré selon le Code César avec la clé (donnée en argument). Le mot, donné en argument est tout en majuscule, sans accent ni caractère spécial.
   Tester votre fonction pour le mot 'EUROPE' avec la clé 5.

   NB: on pourra prendre comme alphabet 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

2. **Code César pour une phrase (sans accent ni ponctuation):**
   Écrire une fonction qui retourne une phrase (donnée en argument) chiffrée selon le Code César avec la clé (donnée en argument).
   Tester votre fonction pour la phrase 'Science sans conscience n est que ruine de l ame' avec la clé 3.

   NB: on pourra prendre comme alphabet 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz '

3. **Déchiffrement du Code César (clé connue):**
   Écrire une fonction qui déchiffre une phrase (donnée en argument) qui a été chiffrée par le programme précédent connaissant la clé (donnée en argument).
   Déchiffrer le code suivant ayant pour clé 12: 'FhflChvwCxqCphvvdjhCfkliiuhCpdlvCmhCvxlvCfdsdeohCghCohCghfkliiuhu'

4. **Déchiffrement du Code César (clé inconnue):**
   Écrire une fonction qui déchiffre une phrase (donnée en argument) qui a été chiffrée par le programme précédent sans connaître la clé qui a été utilisée. Pour réussir ce défi, voir l'article de Wikipédia sur l'analyse fréquentielle: `https://fr.wikipedia.org/wiki/Analyse_fr%C3%A9quentielle`
   Déchiffrer le code suivant: 'UhqghBCyrxvCgdqvCodCuxhCvhfuhwhCdxCghx lhphChwdjhCgxCedwlphqwCyhuw'

Exercice 3: code Vigenère:
Le code de Vigenère est un système de chiffrement par substitution polyalphabétique mais une même lettre du message clair peut, suivant sa position dans celui-ci, être remplacée par des lettres différentes, contrairement à un système de chiffrement mono alphabétique comme le code de César. On procède de la façon suivante pour coder un message avec la clé 'Musique':

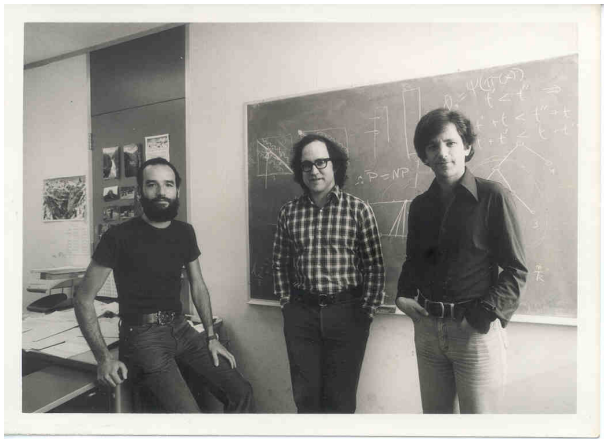
```
J adore ecouter la radio toute la journee
MusiqueMusiqueMusiqueMusiqueMusiqueMusiqu
```

La clé est répétée suffisamment de fois. On décalle ensuite chaque lettre du message de la valeur correspondante à la lettre de la clé juste ne dessous. Avec l'alphabet 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ', ceci donne:

- 'J' est décallée de 12,
- ' ' est décallé de 46,
- ...

1. Écrire une fonction qui crée une clé adaptée à la taille du message à coder.
2. Écrire une fonction qui permet de chiffrer un message avec le code Vigenère.
3. Écrire une fonction qui déchiffre un message, la clé étant connue. Déchiffrer alors le message suivant: 'kVddjWbfOmVzCYQWWoFeazUHbSKaIeQcUqmkbWHrjDHUd
   SYdByOUsZPNHzuHTQUsTtZYYUUxMOdbfGFqzaJdMcUU'

Exercice 4: système RSA simplifié:
Ce système a été inventé en 1977 par trois mathématiciens Ron Rivest, Adi Shamir et Len Adleman. La sécurité du système RSA repose sur la difficulté de factoriser un nombre en produit de deux facteurs premiers. Son principe utilise des théorèmes d'arithmétiques.



On considère un nombre $n = p.q$ où p et q sont des nombres premiers. On calcule le produit $f = (p-1)(q-1)$. On choisit un nombre pub premier et strictement inférieur à f. Le couple (pub, n) constitue la clé publique.

1. **Création de la clé privée:**

   1.1 On crée la clé privée $(priv, n)$ telle que le reste de la division euclidienne de $pub.priv$ par f soit égal à 1. On utilise le code suivant:

```
0 def cle(pub,f):
      priv = 1
2     while priv < f:
          if (pub*priv) % f == 1:
4             return priv
          priv += 1
```

   rsa_simple.py

   Expliquer le fonctionnement de la fonction *cle()*

   1.2 Expliquer pourquoi il est très difficile de créer la clé privée en ne connaissant que la clé publique.

   1.3 Écrire une fonction *creation_cles(p,q,pub)* qui calcule et affiche les couples (pub, n) et $(priv, n)$.

2. **Chiffrement d'un message:**

   2.1 On chiffre un message de la manière suivante:

   - on prend la première lettre du message,
   - on prend son code ascii (fonction *ord()*) que l'on notera m,
   - le code chiffré sera alors le reste de la division euclidienne de m^{pub} par n.
   - on met le code chiffré dans un tableau et on passe à la lettre suivante.

La fonction qui permet de chiffrer une lettre est la suivante. Expliquer son fonctionnement.

```
def rsa(ch, pub, n):
    m = ord(ch)
    chiffre = (m ** pub) % n
    return chiffre
```

rsa_simple.py

2.2 Écrire une fonction *chiffrement(message, pub, n)* qui permette de chiffrer un message complet et la tester.

3. **Déchiffrement d'un message:**

3.1 On déchiffre un message chiffré de la manière suivante:

- on prend la première valeur du tableau que l'on notera c,
- on calcule le reste de la division euclidienne de c^{priv} par n,
- il s'agit du code ascii de la première lettre du message qu'on transforme à l'aide de la fonction *char()*,
- on passe à la valeur suivante du tableau.

La fonction qui permet de déchiffrer une lettre est la suivante. Expliquer son fonctionnement.

```
def rsa_inv(c, priv, n):
    puissance = 1
    for i in range(1,priv + 1):
        puissance = (puissance * c) % n
    return puissance
```

rsa_simple.py

3.2 Écrire une fonction *dechiffrement(tab, priv, n)* qui permette de chiffrer un message complet et la tester.

4. **Application:**
Mes clées publiques et privées sont crées à partir des nombres suivants:

$$\begin{cases} p = 1439 \\ q = 2081 \\ pub = 883 \end{cases}$$

Déchiffrer alors le message suivant: [1100649, 1563587, 1110013, 2277490, 1100551, 1102888, 1843379, 2770101, 1563587, 1843379, 2919795, 1102888, 1443200, 1443200, 1945770, 1288017, 1102888, 1843379, 1110013, 2834302, 1862048, 792759, 792759, 1100551, 1920040, 2953766, 1843379, 1110013, 1102888, 1945034, 1945034, 1102888, 1843379, 792759, 2277490, 1862048, 1443200, 1843379, 880754, 1102888, 1843379, 2919795, 1945770, 1563587, 1862048, 1810588, 1100551, 1102888, 1843379, 1945770, 1443200, 529204, 2919795, 1920040, 1945034, 1100551, 1862048, 2883502, 2770101, 1102888, 1843379, 1385909, 1737432]

Exercice 5: factorisation ou comment casser le chiffrement RSA:

1. Ecrire une fonction qui prend en paramètre un entier $n > 1$ et renvoie un couple d'entiers (p, q) tel que:

   - si n est premier, alors la fonction renvoie le couple $(1, n)$,
   - sinon, la fonction renvoie (p, q) tel que $1 < p \leq q$ et $p.q = n$

   NB: si n n'est pas premier, la fonction renvoie un couple (p, q) parmi éventuellement plusieurs possibles.

2. Utiliser cette fonction dans le cas où $n = 99999640000243$. Commenter le temps d'exécution.

Exercice 6: chiffrement d'une image bitmap:
On souhaite chiffrer l'image lena.pbm en utilisant la bibliothèque pillow de Python.

Le format .pbm attribue une valeur 0 aux pixels blancs et 1 au pixels noirs. On peut manipuler les images avec pillow de la manière suivante:

```
0  from PIL import Image

2  #ouverture de la premiere image:
   image1 = Image.open('lena.pbm')
4  (L,H) = image1.size
   image1.show()
6
   #creation de la seconde image:
8  image2 = Image.new('1',(L,H))
   for x in range(L):
10     for y in range(H):
           p = image1.getpixel((x,y))
12         image2.putpixel((x,y),p)

14 image2.show()
   image2.save('lena_copie.pbm')
```

bitmap.py

1. Que permet de faire le programme bitmap.py?

2. A l'aide de pillow:

   2.1 Créer une image aléatoire de la même dimension que lena.pbm. Les pixels de cette image vaudront aléatoirement 0 ou 1.

Table de vérité de XOR (OU exclusif)		
a	**b**	**a XOR b**
0	0	0
0	1	1
1	0	1
1	1	0

2.2 Chiffrer l'image lena.pbm à l'aide de l'image aléatoire. Vous utiliserez pour ceci l'opérateur *ou exclusif* dont voici la table de vérité:

2.3 Dechiffrer alors l'image chiffrée afin d'obtenir à nouveau lena.pbm.

3. Pourquoi utiliser l'opérateur *ou exclusif*?

4. S'agit-il de chiffrement symétrique ou asymétrique?

5. Que dire de la qualité du chiffrement? Proposer une amélioration.