Sécurisation des communications

Chiffrement de Vigenère en Python

01

Blaise de Vigenère

Blaise de Vigenère (1523–1596) est un diplomate et cryptographe français de la Renaissance. Passionné par les mathématiques et les techniques de chiffrement, il publie en 1586 un traité consacré aux méthodes de codage des messages secrets. Son nom est aujourd'hui associé à l'un des systèmes de chiffrement les plus célèbres de l'histoire.

Le chiffre de Vigenère est un chiffrement par substitution dit polyalphabétique. Contrairement au chiffre de César, il utilise un mot-clé pour faire varier le décalage des lettres tout au long du message. Longtemps considéré comme incassable, il marque une étape importante dans l'évolution de la cryptographie avant l'ère moderne.

Partie 1 — Chiffrement

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Rappels

La fonction `ord` renvoie le code ASCII d'un caractère. Par exemple `ord('A')` vaut `65`, `ord('B')` vaut `66`, etc.
La fonction `chr` est l'inverse : `chr(65)` renvoie `'A'`.

Question 1 — `rang(lettre)`

Écrire une fonction `rang(lettre)` qui prend en paramètre une lettre majuscule et renvoie son rang dans l'alphabet (A→0, B→1, ..., Z→25).

Exemples

```
>>> rang('A')
0
>>> rang('C')
2
>>> rang('Z')
25
```

Correction

Correction

```
def rang(lettre):
    return ord(lettre) - ord('A')
```

Question 2 — `lettre(r)`

Écrire une fonction **`lettre(r)`** qui prend en paramètre un entier **`r`** (entre 0 et 25) et renvoie la lettre majuscule correspondante.

Exemples

```
>>> lettre(0)
'A'
>>> lettre(2)
'C'
```

Correction

Correction

```
def lettre(r):
    return chr(r + ord('A'))
```

Question 3 — `code_lettre(l1, l2)`

Le chiffrement de Vigenère repose sur l'**addition de lettres** modulo 26.

Exemples

```
>>> code_lettre('A', 'C')
'C'
>>> code_lettre('N', 'X')     # 13 + 23 = 36, 36 % 26 = 10 -> K
'K'
```

Correction

Correction

```
def code_lettre(l1, l2):
    r = (rang(l1) + rang(l2)) % 26
    return lettre(r)
```

Question 4 — `vigenere(texte, cle)`

Le chiffrement de Vigenère chiffre chaque lettre du texte en l'additionnant avec la lettre correspondante de la clé (répétée cycliquement).

Texte	B	O	N	J	O	U	R
Clé	C	L	E	C	L	E	C
Chiffré	D	Z	R	L	Z	Y	T

Exemples

```
>>> vigenere("NSI", "BAC")
'OSK'
>>> vigenere("BONJOUR", "CLE")
'DZRLZYT'
```

Correction

Correction

```
def vigenere(texte, cle):
    resultat = ""
    for i in range(len(texte)):
        resultat += code_lettre(texte[i], cle[i % len(cle)])
    return resultat
```

L'instruction `assert`

L'instruction **`assert`** vérifie une condition. Si elle est fausse, une erreur **`AssertionError`** est levée immédiatement.

Exemple

```
def diviser(a, b):
    assert b != 0       # lève une erreur si b vaut 0
    return a / b
```

Question 5 — Assertion dans `rang(lettre)`

Modifier la fonction **`rang`** pour ajouter une assertion vérifiant que **`lettre`** appartient à `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`.

Exemples

```
>>> rang('A')
0
>>> rang('a')    # lève AssertionError
>>> rang('3')    # lève AssertionError
```

Correction

Correction

```
def rang(lettre):
    assert lettre in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    return ord(lettre) - ord('A')
```

Partie 2 — Déchiffrement

Question 6 — `decode_lettre(l1, l2)`

Écrire la fonction inverse de `code_lettre` : on **soustrait** les rangs modulo 26.

Exemples

```
>>> decode_lettre('C', 'C')      # 2 - 2 = 0 -> A
'A'
>>> decode_lettre('K', 'X')      # 10 - 23 = -13, -13 % 26 = 13 -> N
'N'
```

Correction

Correction

```
def decode_lettre(l1, l2):
    r = (rang(l1) - rang(l2)) % 26
    return lettre(r)
```

Question 7 — `decodage_vigenere(texte, cle)`

Même logique que `vigenere`, mais avec `decode_lettre`.

Chiffré	D	Z	R	L	Z	Y	T
Clé	C	L	E	C	L	E	C
Texte	B	O	N	J	O	U	R

Exemples

```
>>> decodage_vigenere("OSK", "BAC")
'NSI'
>>> decodage_vigenere("DZRLZYT", "CLE")
'BONJOUR'
```

Correction

Correction

```
def decodage_vigenere(texte, cle):
    resultat = ""
    for i in range(len(texte)):
        resultat += decode_lettre(texte[i], cle[i % len(cle)])
    return resultat
```