*Pour résoudre ces questions, j'ai utilisé Python pour afficher la réponse.*

*Les espaces et la ponctuation seront négligés.*

1. Afin de convertir les lettres en valeurs interprétables dans un programme, j'ai utilisé la norme ASCII, qui permet d'associer, dans notre cas, une valeur à chaque majuscule. Ainsi, selon cette norme, A = 65 ; B = 66 ; C = 67 ; … ; Z = 90.
   De ce fait, pour établir le dictionnaire associé à une clé donnée, il suffit de prendre la différence de la valeur ASCII de la clé et de 65, qui est en fait égal à la position de la lettre dans l'alphabet diminuée de 1
   Or, ici, la clé est T de code ASCII T = 84 ; donc, sa position dans l'alphabet est 19ème.
   Pour établir la liste complète des correspondances de la clé T ; nous allons créer des sous-listes : donc, cle_t = [ [ A ; T], [B ; S ], [C ; R]…]
   Dans une sous-liste, la première des deux valeurs correspond à la lettre codée, et la deuxième à la lettre non codée. Il est ici intéressant de remarquer que [A ; T] et [T ; A] sont tous deux inclus dans la liste cle_t.
   Nous remarquons que, dans les sous-listes, plus la première lettre se rapproche de Z, plus la deuxième se rapproche de A : ainsi, pour la première lettre U, la deuxième devra correspondre à Z, étant donné que l'alphabet est assimilé à un cercle.
   Ainsi, il faut découper la construction de notre liste de correspondances en deux temps : les valeurs linéaires, pour les lettres situées avant la clé (la boucle IF) et les valeurs situées après la clé, qui auront des correspondances « inversées », (la boucle ELSE).
   *Remarque : dans certains cas, on ajoute 1 à a pour annuler l'effet du 0 comme première valeur de la variable dans la boucle for.*

   Ensuite, on insère le message à coder dans le programme : ce dernier essaye alors, pour chaque lettre dans le message non codé de retrouver sa correspondance ; lorsqu'il l'identifie, il insère la correspondance dans une variable qui construit le message codé.
   Ainsi, lorsqu'on rentre : « Mon cher Bob je te souhaite bonne chance au concours de maths », tout en majuscules et sans espaces, le programme nous renvoie :

   HFGRMPCSFSKPAPBFZMTLAPSFGGPRMTGRPTZRFGRFZCBQPHTAMB

```
cle_t = [ ]
a = 19
for i in range (26):
    if i<(a+1):
        code = chr(65+i)
        non_code = chr(65+a-i)
        couple = [code,non_code]
        cle_t.append(couple)
    else:
        code = chr(65+i)
```

```
        non_code = chr(90+a+1-i)
        couple = [code,non_code]
        cle_t.append(couple)

message = input("Donnez le message codé:")
chaine = ""
for i in range(len(message)):
    for a in range (26):
        if cle_t[a][1] == message[i]:
            chaine = chaine + cle_t[a][0]
print(chaine)
```

2. Ici, la clé est H ; la valeur ASCII de H étant 72, a = 7.
   On remplace donc cela dans notre programme (on peut aussi changer le nom de la variable cle_h). La définition de la liste des correspondances est donc identique à la question précédente.
   Pour, cette fois-ci, on insère déjà un message codé : il faudra donc, pour chaque lettre du message, retrouver sa correspondance non codée : on suivra donc le processus contraire au programme précédent :

   En rentrant : « RNDWWDYTWZDODUOHOZMDONDPMQHZVDUOOQDPCTQO », on retrouve : QUELLEJOLIETENTATIVETUESVRAIMENTTRESFORT, ce que nous traduisons : « Quelle jolie tentative tu es vraiment très fort. »

   Remarque : on peut utiliser ce programme pour vérifier la question 1, en remettant a = 19 : notre réponse est ainsi validée.

```
cle_h = []
a = 7
for i in range (26):
    if i<(a+1):
        code = chr(65+i)
        non_code = chr(65+a-i)
        couple = [code,non_code]
        cle_h.append(couple)
    else:
        code = chr(65+i)
        non_code = chr(90+a+1-i)
        couple = [code,non_code]
        cle_h.append(couple)

message = input("Donnez le message codé:")
chaine = ""
```

```
for i in range(len(message)):
    for a in range (26):
        if cle_h[a][0] == message[i]:
            chaine = chaine + cle_h[a][1]
print(chaine)
```

3. Pour cette question, j'ai pris le message codé donnée dans l'énoncé et je l'ai décodé avec toutes les clés possible ; en employant la même stratégie que précédemment. Cette fois-ci, j'ai indiqué la clé et le message codé selon cette clé. J'ai ensuite cherché la clé qui formait un message compréhensible.

   Ainsi, pour le message :
   HIBEKETMTEYZUAYZMAETSMUVISUUEMJIKYJIVAYZTIPTIDIZIJYSTMEUXMUJITIUKMX
   MKETIUITJYZKDIZITMRMEUXMUIZRYOIBMKBI, la clé formant un message compréhensible est le M, qui écrit :
   « FELICITATIONSMONAMITUASREUSSIADECODERMONTEXTEJENEDOUTAISPASDETE SCAPACITESETDONCJENETAVAISPASENVOYELACLE », ce qui nous traduisons :
   « Félicitations mon ami tu as réussi à décoder mon texte je ne doutais pas de tes capacités et donc je ne t'avais pas envoyé la clé. »

```
message = input("Donnez votre message codé:")
for a in range(26):
    cle = []
    for i in range (26):
        if i<(a+1):
            code = chr(65+i)
            non_code = chr(65+a-i)
            couple = [code,non_code]
            cle.append(couple)
        else:
            code = chr(65+i)
            non_code = chr(90+a+1-i)
            couple = [code,non_code]
            cle.append(couple)
    chaine = ""
    for i in range(len(message)):
        for p in range (26):
            if cle[p][0] == message[i]:
                chaine = chaine + cle[p][1]
    print(chr(65+a),chaine)
```

4. Pour cette question, il a fallu construire toutes les listes de correspondance de toutes les clés, et pour chacune de ces listes, il fallait repérer les lettres qui étaient stables.

Or, dans ma réflexion, un lettre X stable aura une sous-liste de correspondance [X ;X] ; il fallait ainsi repérer ces lettres en montrant qu'elles sont doubles dans une même sous-liste. Le programme :

```
for a in range(26):
    cle = []
    for i in range (26):
        if i<(a+1):
            code = chr(65+i)
            non_code = chr(65+a-i)
            couple = [code,non_code]
            cle.append(couple)
        else:
            code = chr(65+i)
            non_code = chr(90+a+1-i)
            couple = [code,non_code]
            cle.append(couple)
    for p in range(26):
        if cle[p][0]==cle[p][1]:
            print("Clé:",chr(65+a),"Lettre:",chr(65+p))
```

Et son affichage : (Les lettres à droite sont stables pour la clé donnée)

Clé: A Lettre: A
Clé: A Lettre: N
Clé: C Lettre: B
Clé: C Lettre: O
Clé: E Lettre: C
Clé: E Lettre: P
Clé: G Lettre: D
Clé: G Lettre: Q
Clé: I Lettre: E
Clé: I Lettre: R
Clé: K Lettre: F
Clé: K Lettre: S
Clé: M Lettre: G
Clé: M Lettre: T
Clé: O Lettre: H
Clé: O Lettre: U
Clé: Q Lettre: I
Clé: Q Lettre: V
Clé: S Lettre: J
Clé: S Lettre: W
Clé: U Lettre: K

Clé: U Lettre: X
Clé: W Lettre: L
Clé: W Lettre: Y
Clé: Y Lettre: M
Clé: Y Lettre: Z

Nous remarquons que :

- ✓ Seules les lettres à valeur ASCII impaire pouvaient, en temps que clés, accepter des lettres stables ;
- ✓ Toutes les lettres de l'alphabet sont stables qu'une seule fois, pour une clé précise ;
- ✓ Sous une même clé, la différence de la valeur ASCII des deux lettres qui sont stables sous cette clé est constante et vaut 13. Ainsi, Si nous mettions l'alphabet sur un cercle, ces deux lettres stables seraient diamétralement opposées (26 lettres en tout, et 26/2 = 13)
- ✓ Entre deux clés de valeur ASCII impaire, la première lettre stable admise aura une valeur ASCII augmentée de 1 par rapport à la précédente première lettre stable. Ainsi, la première lettre stable sous la clé E est C, tandis que la première lettre stable admise sous C est B. Idem pour les deuxièmes lettres.

Sous ces conditions, une lettre peut admettre un codage par soi-même.