

ENSAE TD noté, mardi 8 novembre 2023

Toutes les questions valent 2 points. Il est conseillé de lire l'intégralité de l'énoncé avant de commencer.

1

Une langue étrangère s'écrit avec 10 lettres **ABCDEFGHIJ**. Chacune est représentée par 4 bits :

```
A 0000
B 0001
C 0010
D 0011
E 0100
F 0101
G 0110
H 0111
I 1000
J 1001
```

Avec cette représentation, **00000110** signifie **AG**.

1) Ecrire une fonction qui code une séquence de lettres en une séquence de 0 et 1.

```
def code(text):
    # ....
    return ...

assert code("AG") == "00000110"
```

2) On s'intéresse au décodage d'un message. Première étape : écrire une fonction qui retourne la première lettre correspondant au premier code qui commence un message codé.

```
def first_letter(chaine):
    # ....

assert first_letter("10010001") == "J"
```

3) Ecrire une fonction qui reçoit une séquence de 0 et de 1 et retourne la séquence de lettres correspondante.

```
def decode(chaine):
    # ....
    return ...

assert decode("00000110") == "AG"
```

4) On forme une classe avec les fonctions précédentes. Il faut compléter le code suivant.

```
class Coding:
    def __init__(self):
```

```

        self.mapping = {'A': '0000', 'B': '0001', 'C': '0010', 'D': '0011',
                       'E': '0100', 'F': '0101', 'G': '0110', 'H': '0111',
                       'I': '1000', 'J': '1001' }

    def first_letter(self, chaine):
        # ....

    def code(self, text):
        # ...

    def decode(self, chaine):
        # ...

c1 = Coding()
assert c1.code("AG") == "00000110"
assert c1.first_letter("00000110") == "A"
assert c1.decode("00000110") == "AG"

```

5) On veut réduire la taille du message codé. Les lettres de A à G sont maintenant codées sur 3 bits et les suivantes sur 5.

```

A 000
B 001
C 010
D 011
E 100
F 101
G 110
H 11100
I 11101
J 11110

```

On crée une nouvelle classe **Coding35** qui hérite de la classe **Coding**.

```

class Coding35(Coding):
    def __init__(self):
        # ....

c1 = Coding35()
assert c1.code("AH") == "00011100"
assert c1.decode("00011100") == "AH"

```

6) Que fait la fonction suivante ? Que suppose-t-elle sur la méthode **decode** pour qu'elle fonctionne. *Il n'est pas demandé de modifier votre code pour qu'elle fonctionne.*

```

def which_coding(text, codings):
    return [c for c in codings if c.decode(text) is not None]

codings = [Coding(), Coding35()]
assert which_coding("0000", codings) == codings[:1]

```

7) Dans ce langage, les lettres sont toutes équiprobables. Quel code produit le texte codé le plus court pour un texte aléatoire très grand et quantifier le gain ? Que se passe-t-il si la lettre J a une probabilité de 0.5 et toutes les autres lettres ont la même probabilité d'apparition ? Que suggérez-vous pour optimiser le Coding en terme de longueur ? *Aucun code n'est demandé.*

8) On change le Coding des lettres A et B : **A 00** et **B 01**. Il faut créer une troisième classe héritant de la première. Que valent **c.code("BGBB")** et **c.code("DEF")** ? Que retourne votre méthode **decode** ?

```
class Coding235(Coding):
    def __init__(self):
        # ...

c = Coding235()
assert c.code("BGBB") == "011100101"
assert c.code("DEF") == ...
```

9) Dans le cas précédent, la première lettre peut être soit **B** soit **D**. Ecrire une méthode qui retourne toutes les options pour la première lettre d'un message codé.

```
class Coding235(Coding):
    # ...
    def first_letters(self, chaine):
        # ...

c = Coding235()
assert c.first_letters("011100101") == {"B", "D"}
```

10) Ecrire une méthode **decode** qui retourne toutes les solutions par récurrence.

```
class Coding235(Coding):
    # ...
    def decode(self, chaine):
        # ...

c = Coding235()
assert c.decode("011100101") == {"BGBB", "DEF"}
```