1 Exercice 1 (La distance de Hamming)

1) La distance de Hamming entre deux chaînes de caractères.

La distance de Hamming entre deux chaînes de caractères de mêmes longueurs est égale au nombre de caractères, à la même position, qui sont différents.

Exemples :

- La distance de Hamming entre "**sure**" et "**cure**" est 1.
- La distance de Hamming entre "**aabbcc**" et "**xaybzc**" est 3.

① Écrire une fonction **distanceH(S1, S2)** qui calcule et retourne la distance de Hamming entre $S1$ et $S2$ (les paramètres S1 et S2 sont deux chaînes de caractères de même longueur).

2) La distance de Hamming d'un langage.

On appellera langage, une liste des chaînes de caractères toutes de même longueur. La distance de Hamming d'un langage est égale au minimum des distances de Hamming entre deux chaînes de caractères de ce langage différentes deux à deux.

Exemple : Si langage =["aabb","xayy","tghy","xgyy"], sa distance de est égale à 1.

② Écrire une fonction **distanceH_langage(L)** qui calcule et retourne la distance de Hamming de son paramètre L.

3) La distance de Hamming entre 2 nombres entiers positifs.

La distance de Hamming entre 2 nombres entiers positifs est le nombre de bits distincts dans leurs représentations binaire.

Exemple : La distance de Hamming entre les nombres 7 et 4 est 2 (7 est représenté en binaire sur un octet (8bits) par $'00000111'$ et 4 est représenté en binaire par $'00000100'$

③ Écrire une fonction **binaire(N)** qui retourne une chaîne de caractères représentant la valeur binaire de N sur un octet (on suppose que $0 <= N < 256$).

④ Écrire une fonction **distanceNombre(A, B)** qui calcule la distance de Hamming entre les nombres A et B (on suppose que 0<=A<256 et 0<=B<256)

2 Exercice 2 (ADN et génôme)

On représente un brin d'ADN par une chaîne de caractères qui peut contenir quatre caractères différents : 'A' (Adénine), 'C' (Cytosine), 'G' (Guanine) et 'T' (Thymine).

① Écrire une fonction **estADN(ch)** qui prend en argument une chaîne de caractères et renvoie *True* si cette chaîne correspond à un brin d'ADN et *False* sinon.

```
>>> estADN("TTGAC")
True
>>> estADN("AMOG")
False
>>> estADN("CaTG")
False
```

② Écrire une fonction **masseMolaire(ch)** qui calcule la masse molaire d'une séquence ADN passée en argument. Chaque lettre a une masse donnée : 'A' (135 g/mol) ; 'T' (126 g/mol) ; 'G' (151 g/mol) ; 'C' (111 g/mol). Par exemple, masseMolaire("AGATC") =135 + 151 + 135 + 126 + 111)=658 g/mol.

Chaque base possède une base complémentaire avec laquelle elle peut s'associer : "A" et "T" sont complémentaires, et "C" et "G" sont complémentaires.

③ Écrire une fonction **brinComp(b)** qui étant donné un brin d'ADN b calcule et renvoie son brin complémentaire, c'est à dire le brin constitué des bases complémentaires de b.

```
>>> brinComp("AAGT")
'TTCA'
```

④ Écrire une fonction **sous_sequence(A, B)** qui prend en argument deux chaînes de caractères représentant des brins d'ADN et renvoie **True** si le premier brin est une sous-séquence du deuxième.

```
>>> sous_sequence("ATC", "GGTATCG")
True
>>> sous_sequence("GC", "AAT")
False
```

Un gène est une sous-chaîne d'un brin d'ADN qui commence après un triplet ATG et se termine avant un triplet TAG, TAA, ou TGA. De plus, la longueur d'une chaîne de gènes est un multiple de 3 et le gène ne contient aucun des triplets ATG, TAG, TAA, et TGA.

⑤ Écrire une fonction **TrouverGene(CH)** qui affiche tous les gènes dans le brin d'ADN CH.

```
>>> Entrer un génome: TTATGTTTTAAGGATGGGGCGTTAGTT
TTT
GGGCGT
>>> Entrer un génome: TGTGTGTATAT
Pas de gènes
```

3 Exercice 3

① Écrivez une fonction **freqChaine(ch)** qui prend une chaîne de caractères en paramètre et renvoie un dictionnaire contenant pour chaque lettre de la chaîne son nombre d'occurrences.

```
>>> freqChaine('abbabba!')
{ "a" : 3 , "b" : 4, "!": 1}
```

② Écrivez une fonction **maxDic(d)** qui prend en paramètre un dictionnaire (dont les valeurs sont des naturels) et renvoie la clé correspondant à la plus grande valeur.

```
>>> maxDic({'a': 3, 'b' : 5 , 'c' : 0, 'd' : 10})
'd'.
```

③ Écrivez une fonction **triCles(d)** qui prend un dictionnaire (dont les valeurs sont des naturels) et renvoie la liste des clés du dictionnaire par ordre de valeurs décroissantes dans le dictionnaire.

```
>>> triCles({'a': 3, 'b' : 5 , 'c' : 0, 'd': 10})
['d', 'b', 'a', 'c']
```

4 Exercice 4 (QCM)

Dans un premier temps, chaque question est représentée par un dictionnaire q contenant trois champs :

- q['question'] est une chaîne qui est la question à poser.
- q['correcte'] est la bonne réponse
- q['incorrecte'] est une liste contenant les mauvaises réponses

Un questionnaire est donc simplement une liste de questions, c'est à dire une liste de dictionnaires.

```
qs = [
       {'question': "Quelle est la couleur du cheval blanc d'Henry IV ?",
        'correcte': "blanc",
        'incorrecte': ["gris","noir","orange"]},

       {'question': "2 + 2 = ?",
        'correcte': "4",
        'incorrecte': ["42","44","0"]}
    ]
```

① Écrivez une fonction **question(q)** qui prend une question en argument, et qui : affiche la question, affiche les réponses possibles, lit la réponse de l'utilisateur, affiche "Bonne réponse !" ou "Mauvaise réponse !" et qui renvoie un "+1" ou "-1".

② Écrire une fonction **Quiz(q)** qui permet de poser toute les questions d'un questionnaire q et calcule le score et le renvoie, comme une note sur 100.

Toujours poser les questions dans le même ordre n'est pas une très bonne idée. Pour mélanger les questions, vous pouvez utiliser la méthode "**shuffle**" de la bibliothèque "**random**". Cette méthode permet de mélanger une liste existante :

```
>>> from random import shuffle
>>> t = [1,2,3,4,5,6]
>>> shuffle(t)
>>> t
[5, 6, 1, 4, 3, 2]
```

③ Modifier la fonction **Quiz(q)** pour qu'elle affiche les questions et les réponses dans le désordre.

5 Exercice 5

Un météorologiste souhaite améliorer l'efficacité du traitement des données qu'il utilise (tableau de température moyenne mensuelle ci-dessous). Son objectif étant de réaliser des traitements statistiques, il décide d'utiliser les structures des données du langage Python.

	Janvier	...	Décembre
France	06	...	08
...	...	...	...
Australie	35	...	30

On suppose avoir ces deux variables globales :

$$Temperatures = \{"France" : [06, 05, \ldots, 08], \ldots, "Australie" : [35, \ldots, 30]\}$$

$$Mois = ["Janvier", \ldots, "Dcembre"]$$

① Donner l'instruction permettant d'afficher en France la température du mois de février sous la forme *'France février 05'*.

② Écrire une fonction **AfficheMois(mois)**,ayant pour paramètre le nom d'un mois, et qui affiche pour tous les pays la température du mois comme précédemment.

Afin de pouvoir modifier ses données, le météorologiste souhaite utiliser deux fonctions que vous implanterez en python:

③ La Fonction **AjoutPays(pays, temps)** qui prend comme paramètre le nom d'un pays ainsi que la liste des températures associées à ce pays et l'ajoute au dictionnaire **Temperatures**.

```
>>> AjoutPays("Bresil",[25,....23]).
```

④ La Fonction **ModificationPaysMois(pays, mois, val)** qui prend comme paramètre d'entrée le nom du pays, le nom du mois, la valeur de la nouvelle température. Cette fonction doit modifier la valeur de la température du pays pour le mois indiqué.

```
>>> ModificationPaysMois("France","Janvier",08)
# doit modifier la valeur de la température en France au mois de Janvier
#(ici 08 remplaçant la valeur 06).
```

⑤ Donner la fonction **MoyennePays(pays)** qui pour un pays donné en paramètre d'entrée donne la moyenne annuelle des températures.

⑥ Donner la fonction **moyenneMois(mois)** qui pour un mois donné (par exemple Mars) donne la moyenne des températures de tous les pays.

⑦ Donner la fonction **MoyenneMax()** qui donne le pays dont la moyenne annuelle des températures est la plus élevée.