Sujet « zéro » de l'épreuve d'informatique TB

PRÉSENTATION

L'énoncé proposé ci-dessous est conçu pour l'épreuve d'informatique TB qui dure 45 minutes.

Vu le temps imparti, le choix a été d'une épreuve largement orientée QCM, avec néanmoins quelques questions « ouvertes » et à réponses courtes.

Les compétences visées dans ce texte sont les suivantes :

▷ Analyser et modéliser :
- Partie I. Q6 - Q7 -Q8 : analyser et modéliser un brin d'ADN
- Partie II. Q9 et suivantes : analyser et modéliser un arbre phylogénique

▷ Imaginer / Communiquer :
- Partie I. Q4 - Q8
- Partie II. Q 12

▷ Traduire un algorithme dans un langage de programmation : cette compétence est présente dans la quasi-totalité des questions (langage Python).

▷ Spécifier : cette compétence n'est pas directement évaluée dans ce sujet, sauf sous la forme « comprendre une spécification précise » :
- Partie I. Q6 - Q7 -Q8
- Partie II. Q14 - Q15

▷ Évaluer, contrôler, valider :
- Partie I. Q3 - Q4 -Q6
- Partie I. Q10 - Q11

PARTIE 1: TRAITEMENT DE SÉQUENCES

Dans cette partie, on s'intéresse au traitement automatique de séquences de caractères et en particulier de séquences d'ADN. En effet, un brin d'ADN peut être représenté par une séquence de caractères symbolisant les nucléotides qui se succèdent sur le brin d'ADN.

1.1 Traitement générique de chaînes de caractères

Considérons dans un premier temps des chaînes de caractères quelconques.

Question 1 (QCM)

Considérons les instructions suivantes :

```
a = "10"
b = "11"
c = a + b
print (c)
```

Qu'affiche le script ?

❑ "1011" ❑ 21 ❑ 1011 ❑ 1110 ❑ "1110"

Question 2 (QCM)

Parmi les expressions ci-dessous, lesquelles peuvent permettre d'obtenir le dernier caractère d'une chaîne non vide nommée maChaine ?

- ☐ maChaine[len(maChaine)]
- ☐ maChaine[len(maChaine) + 1]
- ☐ maChaine[len(maChaine - 1)]
- ☐ maChaine[len(maChaine) - 1]
- ☐ maChaine[-1]

Question 3 (QCM)

Considérons les instructions suivantes :

```
resultat = ""
for c in "Bonsoir" :
        resultat = resultat + c
print (resultat)
```

Qu'affiche le script ?

- ☐ B Bo Bon Bons Bonso Bonsoi Bonsoir *(sur plusieurs lignes)*
- ☐ riosnoB
- ☐ BonsoirBonsoirBonsoirBonsoirBonsoirBonsoirBonsoir
- ☐ Bonsoir

Question 4 (Réponse courte)

En changeant l'indentation d'une ligne dans le script de la question précédente, son exécution permet d'obtenir une des autres propositions faites ci-dessus. Quelle indentation faut-il modifier ? Pour quelle ligne ? Quel est alors le résultat obtenu ?

1.2 Cas particulier des séquences d'ADN

On cherche à mettre en place des traitements spécifiques aux chaines de caractères dans le but de symboliser et d'étudier des brins d'ADN.

1.2.1 Validité de la représentation

Question 5 (QCM)

Considérons la fonction suivante :

```
def fonction_question5(chaine, caractere_valide):
        resultat = True
        for caractere in chaine :
                if caractere != caractere_valide :
                        resultat = False
        return resultat
```

Que renverra chacune des interrogations interactives suivantes ?

Interrogation 1

```
>>> fonction_question5("aaaaaa","a")
```

☐ True ☐ False ☐ 6 ☐ 1

Interrogation 2

```
>>> fonction_question5("abaaaa","a")
```

☐ True ☐ False ☐ 5 ☐ 1

Interrogation 3

```
>>> fonction_question5("abaaaa","b")
```

☐ True ☐ False ☐ 1 ☐ 2

Interrogation 4

```
>>> fonction_question5("ababab","ab")
```

☐ True ☐ False ☐ 3 ☐ 1

Question 6 (QCM)

Parmi les scripts suivants lesquels permettent de déterminer si une chaine peut modéliser un brin d'ADN ? On rappelle qu'un brin d'ADN ne peut comporter que les caractères A, C, G et T.

☐

```
def fonction_question6(chaine):
    resultat = True
    for caractere in chaine :
        if caractere == "A" or caractere == "C" or caractere == "G" or caractere == "T" :
            resultat = False
    return resultat
```

☐

```
def fonction_question6(chaine):
    caracteres_valids = "ACGT"
    resultat = True
    for caractere in chaine :
        if caractere not in caracteres_valids :
            resultat = False
    return resultat
```

❑

```
def fonction_question6(chaine):
    caracteres_valids = "ACGT"
    resultat = True
    for caractere in chaine :
        temp = False
        for caractere_valid in caracteres_valids :
            if caractere == caractere_valid :
                temp = True
        if not temp :
            resultat = False
    return resultat
```

❑

```
def fonction_question6(chaine):
    resultat = False
    for caractere in chaine :
        if caractere == "A" or caractere == "C" or caractere == "G" or caractere == "T" :
            resultat = True
    return resultat
```

1.2.2 Lecture de brins d'ADN et mécanismes associés

On se propose d'écrire une fonction qui, étant donnée une chaîne de caractères représentant un brin d'ADN, fournit en sortie son complémentaire dans son sens de lecture, sachant que les liens se font ainsi : A avec T et C avec G. Considérons par exemple la portion d'ADN suivante :

```
->
A  T  G  C
|  |  |  |
T  A  C  G
         <-
```

Les flèches –> et <– matérialisent le sens de lecture sur chaque brin.
Ainsi, la fonction à proposer devra retourner GCAT en réponse à la chaine ACGT.

Dans ce but, on souhaite disposer d'une fonction nommée `fonction_question7` qui renvoie l'index de la première occurrence d'un caractère dans une chaîne s'il y est présent, -1 sinon.

Question 7 (QCM)

Parmi les scripts suivants lequel (ou lesquels) correspond (ou correspondent) à la fonction `fonction_question7`

❑

```
def fonction_question7 (caractere,chaine):
    i = -1
    k = 0
    while i == -1 and k < len(chaine) :
        if caractere == chaine[k] :
            i = k
            k = k + 1
    return i
```

❑

```
def fonction_question7 (caractere,chaine):
    i = -1
    k = 0
    while i == -1 or k < len(chaine) :
        if caractere == chaine[k] :
            i = k
        k = k + 1
    return i
```

❑

```
def fonction_question7 (caractere,chaine):
    i = -1
    k = 0
    while i == -1 or k >= len(chaine) :
        if caractere == chaine[k] :
            i = k
        k = k + 1
    return i
```

❑

```
def fonction_question7 (caractere,chaine):
    i = -1
    k = 0
    while i == -1 and k < len(chaine) :
        if caractere == chaine[k] :
            i = k
        k = k + 1
    return i
```

❑

```
def fonction_question7 (caractere,chaine):
    i = -1
    k = 0
    while i == -1 and k < len(chaine) :
        if caractere == chaine[k] :
            i = k
        i = i + 1
    return i
```

Question 8 (Réponse courte)

Proposez une fonction, qui, étant donnée une chaîne de caractères représentant un brin d'ADN, fournit en sortie son complémentaire dans son sens de lecture. On utilisera la fonction `fonction_question7`.

PARTIE 2: PHYLOGÉNIE - ÉVOLUTION DE CARACTÈRES ET LIENS ENTRE ESPÈCES

2.1 Contexte

Nous nous intéressons aux liens de parentés de m espèces, à partir de l'évolution de n caractères.

Deux espèces distinctes ont au moins un caractère différent.

Chaque caractère possède deux états : un état *originel* (noté 0) et un état *dérivé* (noté 1). L'évolution d'un caractère se fait toujours depuis l'état originel vers l'état dérivé. Nous supposons l'existence d'une espèce, appelée ancêtre commun, pour laquelle tous les caractères sont à l'état originel.

Voici un exemple avec 7 espèces nommées A,B,C,D,E,F,H et 6 caractères nommés c_0 , c_1 , c_2 , c_3 , c_4 , c_5. Le tableau donne pour chaque espèce l'état de chaque caractère.

	caractère					
espèce	c_0	c_1	c_2	c_3	c_4	c_5
A	0	1	0	0	0	1
B	0	0	0	1	0	1
C	1	0	0	0	0	0
D	0	0	0	0	0	1
E	0	0	0	0	0	0
F	0	0	1	0	0	1
H	0	0	0	1	1	1

Par exemple, pour l'espèce A les caractères c_1 et c_5 sont à l'état dérivé, les autres sont à l'état originel.
Pour l'espèce E, tous les caractères sont à l'état originel.

Nous représentons en Python chaque espèce par la liste des valeurs (0 ou 1) de ses caractères. L'ensemble des espèces est lui-même stocké dans une liste que l'on nomme `donnees_phylo`.

Question 9 (QCM)

En considérant uniquement les trois premières espèces (A,B,C) et les trois premiers caractères (c_0, c_1, c_2) de l'exemple ci-dessus, indiquez quelle instruction d'affectation de la liste `donnees_phylo` en donne une représentation exacte.

☐ `donnees_phylo = [[0,1,0],[0,0,0],[1,0,0]]` ☐ `donnees_phylo = [[0,0,1],[1,0,0],[0,0,0]]`

On considère la fonction

```
def add_caractere(donnees_phylo,nouv_caractere):
    if len(donnees_phylo)==len(nouv_caractere):
        for i in range(len(donnees_phylo)):
            donnees_phylo[i] = donnees_phylo[i] + [nouv_caractere[i]]
```

Question 10 (Réponse courte)

Dans la situation `donnees_Phylo = [[0,0,0,0],[0,0,0,0],[0,0,1,0]]` et `nouv_caractere = [1,1,0]`, quel est le contenu de `donnees_Phylo` après exécution de l'instruction `add_caractere (donnee_Phylo,nouv_caractere)` ?

Question 11 (Réponse courte)

Dans la situation `donnees_Phylo = [[0,0,0,0],[0,0,0,0],[0,0,1,0]]` et `nouv_caractere = [1,1,0,1]`, quel est le contenu de `donnees_Phylo` après exécution de l'instruction `add_caractere (donnees_Phylo,nouv_caractere)` ?

Question 12 (Réponse courte)

Proposez une fonction pour rajouter une nouvelle espèce à la liste `donnees_phylo` en complétant le script ci-dessous

```
def add_espece(donnees_Phylo,nouv_espece):
    if len(
```

Question 13 (QCM)

Étant donnée une telle représentation, quelle(s) instruction(s) permet(tent) d'accéder à la valeurs du troisième caractère de la dernière espèce ?

- ☐ `donnees_phylo[-1][3]`
- ☐ `donnees_phylo[-1][2]`
- ☐ `donnees_phylo[6][3]`
- ☐ `donnees_phylo[5][2]`

2.2 Relations entre espèces : ancêtre et descendants

Étant données deux espèces distinctes a et b :

1. L'espèce b est dite descendante de l'espèce a si et seulement si chaque caractère présent dans l'état originel pour l'espèce b est aussi présent dans l'état originel pour l'espèce a.
   On dit aussi que l'espèce a est ancêtre de l'espèce b.
2. L'espèce b est dite descendante immédiate de l'espèce a si et seulement si b est descendante de l'espèce a à la suite de la mutation d'un seul caractère.
   On dit aussi que l'espèce a est ancêtre immédiat de l'espèce b.

On suppose que chaque espèce, à l'exception de l'ancêtre commun, possède un unique ancêtre immédiat.

Pour l'exemple donné en début, H est descendante de D car les caractères présents à l'état originel pour l'espèce H (c_0, c_1 et c_2) sont à l'état originel pour l'espèce D.
Dans le but de tester si une espèce est ancêtre d'une autre, on propose les deux fonctions suivantes :

2.2.1 Première fonction

```
def est_ancetre1 (espece_1,espece_2):
        """
        renvoie True si espece_1 est ancêtre de espece_2, False sinon
        """
    resultat=True
    if len(espece_1) == len(espece_2):
        i = 0
        for i in range(len(espece_1)):
            if espece_2[i] == 0 and espece_1[i] == 1 :
                resultat=False
    return resultat
```

2.2.2 Seconde fonction

```
def est_ancetre2 (espece_1,espece_2):
    """
    renvoie True si espece_1 est ancêtre de espece_2, False sinon
    """
    resultat=True
    if len(espece_1) == len(espece_2):
        i = 0
        while i < len(espece_1) and resultat :
            if espece_2[i] == 0 and espece_1[i] == 1 :
                resultat=False
            i += 1
    return resultat
```

Question 14 (Réponse courte)

Comparer les deux versions lorsque `espece_1=[0,0,0,1,0,1]` et `espece_2=[0,0,1,0,0,1]` en précisant le nombre de passages dans la boucle.

2.2.3 Nombre de descendants d'une espèce

Question 15 (Compléter une fonction)

Complétez la fonction suivante en utilisant la fonction `est_ancetre2`.

```
def nombre_descendants (donnees_phylo,espece):
    """
    renvoie le nombre nb de descendants de espece
    compte tenu des données stockées dans donnees_phylo
    """









    return nb
```