# CONCOURS BLANC INFORMATIQUE
# AUTOUR DU SÉQUENÇAGE DU GÉNOME
# CORRIGE

*Q1.*

`seq[3]-> G ; seq[2 :6] -> CGTA`

*Q2.*

```
1 import random
2 def generation(n):
3     seq = ""                # chaine vide
4     for i in range(n):
5         a = random.randint(1,4)   # en fait randint(1,5) inclut 5. L'énoncé
6         if a == 1:                # se trompe, aucune pénalité pour vous
7             seq += 'A'
8         elif a == 2 :
9             seq += 'C'
10        elif a == 3 :
11            seq += 'G'
12        else:
13            seq += 'T'
14    return seq
```

*Q3. Une proposition simple …*

```
1 def pourcentage(seq):
2     a,b,c,d = 0,0,0,0   # compteurs des nombres de A,C,G,T dans seq
3     i = 0
4     while i < len(seq):
5         if seq[i]=='A':
6             a = a+1
7             i = i+1
8         elif seq[i]=='C':
9             b = b+1
10            i = i+1
11        elif seq[i]=='G':
12            c = c+1
13            i = i+1
14        else:
15            d = d+1
16            i = i+1
17    return [a*100/len(seq), b*100/len(seq), c*100/len(seq), d*100/len(seq)]
```

*Q4. Le nombre d'itérations est égal à len(seq)*

*Terminaison : nom de la variable : $i$ (si on note n la longueur de la séquence, n-i est un variant de boucle)*
*Justification : $i$ correspond à la position de la premeière lettre, à chaque tour dans la boucle la valeur de $i$ augmente de 1 jusqu'à dépasser n et dans ce cas on sort de la boucle while.*

*Q5.*

```
1 def recherche(M,T):
2     for i in range(len(T)):
3         j = 0
4         while (i+j < len(T)) and (j < len(M)) and (T[i+j] == M[j]):
5             j += 1
6         if j == len(M):  # càd si T[i+j]==M[j] pour j de 0 à len(M)-1
7             return i
8     return -1
```

*Q6. Nombre d'opérations pour chercher un motif de 50 caractères : $50 \times 3 \times 10^9 \times 7$ (varie selon votre réponse à Q5) = $1.05 * 10^{12}$*

*Temps mis par l'ordinateur : 1.05 seconde*

*Q7. Temps pour comparer deux séquences d'ADN : une séquence d'ADN contient $3 \times 10^9 / 50 = 0.6 \times 10^8$ morceaux de taille 50 : chacun prend 1.05 seconde pour une comparaison (par Q6). Ainsi, on mettra environ $50 \times 0.6 \times 10^8 = 3 \times 10^9$ secondes. Sachant qu'il faut environ $3 \times 10^3$ secondes pour faire une heure, on arrive à un temps d'environ un million d'heures.*

*Cette méthode prend beaucoup trop de temps*

*Q8. Préfixes de 'ACGTAC' : 'A', 'AC', 'ACG', 'ACGT', 'ACGTA'*

*Suffixes de 'ACGTAC' : 'C', 'AC', 'TAC', 'GTAC', 'CGTAC'*

*Q9. Plus grand préfixe de 'ACGTAC' qui soit aussi un suffixe : 'AC'*

*Plus grand préfixe de 'ACAACA' qui soit aussi un suffixe : 'ACA'*

*Q10. Type de la variable en sortie : liste d'entiers*
*Q11.*

```
1 def fonctionannexe(M):
2     F=[1]
3     i=1
4     j=0
5     m = len(M)  # on définit m pour la ligne suivante
6     while i < m :
7         if M[i]==M[j] : # un seul =
8             F.append(j+1)
9             i=i+1
10            j=j+1
11        else: # oubli du :
12            if j>0 :
13                j=F[j-1]
14            else:
15                F.append(0)
16                i=i+1
17    return F
```

*Q12.*

initialisation : i=1 ; j=0 ; F=[1]

Fin du premier passage dans la boucle while : i= 2 ; j= 0 ; F= [1,0] ;

Fin du deuxième passage dans la boucle while : i= 3 ; j= 1 ; F= [1,0,1] ;

Fin du troisième passage dans la boucle while : i= 3 ; j= 0 ; F= [1,0,1] ;

Fin du quatrième passage dans la boucle while : i= 4 ; j= 1 ; F= [1,0,1,1] ;

Fin du cinquième passage dans la boucle while : i= 5 ; j= 2 ; F= [1,0,1,1,2] ;

Fin du sixième passage dans la boucle while : i= 6 ; j= 3 ; F= [1,0,1,1,2,3] ;

(Suite page suivante)

*Q13.*

```
1 def recherchedichotomique(a,L):
2     g = 0             # g sera le premier indice de la sous-liste
3     d = len(L)-1      # d sera le dernier indice de la sous-liste
4     m = (g+d)//2      # m est un indice qui correspond quasiment au milieu
5     while g < d :
6         if L[m] == a :
7             return m
8         elif L[m] > a :  # a n'est pas dans L[m],...,L[d]
9             d = m-1      # donc on regarde la sous-liste L[g],...,L[m-1]
10        else :
11            g = m+1
12        m = (g+d)//2
13    return "l'element n'est pas dans la liste" # n'arrive pas par hypothèse
```

*Cet algorithme a une complexité logarithmique, i.e. en $O(\log_2(n))$. Il est donc plus rapide qu'un algorithme de recherche naïf qui a une complexité linéaire. Cependant, il suppose que la liste soit déjà triée.*

*Q14. h('CCC') : '111', $4^2 + 4 + 4^0 = 21$ donc h('CCC')=8* ***(modulo 13)***

*h('ACG') : '012', 4 + 2 = 6 donc h('ACG')=6*
*h('GAG') : '202', $2 * 4^2 + 0 + 2 * 1 = 34$ donc h('GAG')=8*

*Q15.*

```
1 def eval(P,b):
2     s=0
3     n=len(P)    # si P est de degré d, alors n=d+1
4     for k in range(n):
5         s = s + P[k]*b**(n-k)   # car P[0] est le coeff de plus haut degré
6     return(s)
```

*Q16.*

```
1 def hornerit(P,b):
2     s=P[0]
3     for k in range(1,len(P)):
4         s = s*b+P[k]
5     return s
```

*Q17.*

```
def echantillon(tab):
    g = open('echantillon.txt', 'w')
    g.write('ADN \t Genre \t \t Espèce \t Sous-espèce \n')
    for i in range(len(tab)):
        for j in range(len(tab[i])): #on peut le faire sans boucle avec les 4
            g.write(str(tab[i][j])+'\t')
        g.write('\n')
    g.close()
```

*Q18.*

```
f=open('sequence.txt', 'r')
tab2=[]
for i in f:
    l=i.strip().split('\t')
    tab2.append(l)
```