DS1 - INFORMATIQUE
AUTOUR DU SEQUENÇAGE DU GENOME
CORRIGE

Q1.

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

Q2.

```
1 def generation(n):
2     seq = str()
3     for i in range(n):
4         a = randint(1,4)
5     if a == 1:
6         seq += 'A'
7     elif a == 2 :
8         seq += 'C'
9     elif a == 3 :
10        seq += 'G'
11    else:
12        seq += 'T'
13    return(seq)
```

Q3. Une proposition simple …

```
1 def mystere(seq):
2     a,b,c,d = 0,0,0,0
3     i = len(seq)-1
4     while i>= 0:
5         if seq[i]=='A':
6             a += 1
7             i -= 1
8         elif seq[i]=='C':
9             b += 1
10            i -= 1
11        elif seq[i]=='G':
12            c += 1
13            i -= 1
14        else:
15            d += 1
16            i -= 1
17    return [a*100/len(seq),b*100/len(seq),c*100/len(seq),d*100/len(seq)]
```

Q4. Complexité linéaire (O(len(seq))

Terminaison : nom de la variable : i (c'est un variant de boucle)
justification : i correspond à la position de la dernière lettre, à chaque tour dans la boucle la valeur de i diminue de 1 jusqu'à être négatif et dans ce cas on sort de la boucle while.

Q5.

```
1 def recherche(M,T):
2     i = 0
3     while i < len(T) :
4         j = 0
5         while (i+j < len(T)) and (j < len(M)) and (T[i+j] == M[j]):
6             j += 1
7         if j == len(M):
8             return(i)
9        i += 1
10       return(-1)
```

Q6. Nombre d'opérations pour chercher un motif de 50 caractères : $50 \times 3 \times 10^9 = 1.5 * 10^{11}$

Temps mis par l'ordinateur : 0.15 secondes

Q7. Temps pour comparer deux séquences d'ADN : $10^8 \times 0.15 \sim 10^7$ secondes ~ 2777 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'entier

Q11.

```
1 def fonctionannexe(M):
2     F=[0]
3     i=1
4     j=0
5     while i < m : # m n'est pas défini: m=len(M)
6           if M[i]==M[j] : # un seul =
7               F.append(j+1)
8               i=i+1
9               j=j+1
10          else: #oubli du :
11              if j>0 :
12                  j=F[j-1]
13              else:
14                  F.append(0)
15                  i=i+1
16    return F
```

Q12.

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

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

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

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

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

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

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

Q13.

```
1 def KMP(M,T):
2     F=fonctionannexe(M)
3     i=0
4     j=0
5     while i < len(T) :
6         if T[i]==M[j]:
7             if j==len(M)-1:   return(i-j)
8             else:
9                 i=i+1
10                j=j+1
11        else:
12            if j > 0:
13                j=F[j-1]
14            else:
15                i=i+1
16     return -1
```

Q14.

```
def tri_selection(T):
    for i in range(len(T)-1):
        for j in range (i+1, len(T)):
            if T[i]>T[j] :
                T[i],T[j]=T[j],T[i]
    return T
```

Q15.

```
def recherchedichotomique(a,L):
    c = 0
    l = len(L)-1
    m = (c+l)//2
    while c < l :
        if L[m] == a :
            return m
        elif L[m] > a :
            l = m-1
        else :
            c = m+1
        m = (c+l)//2
    return c

il a une complexité logarithmique
```

Q16. 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

Q17.

```
def eval(P,b):
    s=0
    for k in range(len(P)):
        s+=P[k]*b**(n-k)
    return(s)
```

Q18.

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

Q19.

```
def hornerrec(P,b):
    if len(P)==1:
        return(P[0])
    else:
        s = P[len(P)-1]
        s1 = P[0:len(P)-1]
        return (hornerec(s1,b)*b+s)
```

Q20. But de la requête(1) : On compte le nombre de séquences étudiées le 01 mars 2018
Q21. Requête(2) = SELECT ADN FROM Sequence WHERE Gène=leuS
Q22. Requête(3) = SELECT Espèce FROM Echantillon Join Sequence on Numero=AND WHERE Employé='Martin' AND Date='01-03-2018'
Q23. Requête(4) = SELECT COUNT(Code), Employé FROM Sequence GROUP BY Employé
Q24. Requête(5)=SELECT MAX(c),Employé FROM (SELECT COUNT(Code) as c, Employé FROM Sequence GROUP BY Employé)

IV Exo supplémentaire

A.1) $z(t) = y'(t)$ donc en dérivant une nouvelle fois : $z'(t) = y''(t)$. On obtient donc le système d'équation différentielles du premier ordre suivant :

$$(S) \Leftrightarrow \begin{cases} y'(t) = z(t) \\ z'(t) = f\big(y(t)\big) \end{cases}$$

A.2) En intégrant chacune des équations du système (*S*) précédent entre l'instant t_i et t_{i+1} on obtient :

$$\int_{t_i}^{t_{i+1}} z(t)\,dt = \int_{t_i}^{t_{i+1}} y'(t)\,dt \quad \text{et} \quad \int_{t_i}^{t_{i+1}} z'(t)\,dt = \int_{t_i}^{t_{i+1}} f(y(t))\,dt$$

$$\int_{t_i}^{t_{i+1}} z(t)\,dt = y(t_{i+1}) - y(t_i) + \quad \text{et} \quad z(t_{i+1}) - z(t_i) = \int_{t_i}^{t_{i+1}} f(y(t))\,dt$$

$$Soit : \qquad y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\,dt \quad \text{et} \quad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\,dt$$

B.1) L'approximation d'Euler consiste à définir $y(t_i) \approx y_i$ et $y(t_{i+1}) \approx y_{i+1}$ et d'approximer la dérivée à l'instant t_i , ainsi le système *(S)* peut être approximé par :

$$(S) \Leftrightarrow \begin{cases} z_i = \dfrac{y_{i+1} - y_i}{h} \\ \dfrac{z_{i+1} - z_i}{h} = f(y_i) \end{cases}$$

Soit les 2 relations de récurrence :

$$\begin{cases} y_{i+1} = y_i + h\,z_i \\ z_{i+1} = z_i + h\,f(y_i) \end{cases}$$

B.2)

```
def euler(f,y0,z0,tmin,n,h):
    y=y0
    z=z0
    valeursy=[y0]
    valeursz=[z0]
    for k in range(n):
        y=y+h*valeursz[-1]
        z=z+h*f(valeursy[-1])
        valeursy.append(y)
        valeursz.append(z)
    return valeursy,valeursz
```

C.1)

```
def verlet(f,y0,z0,tmin,n,h):
    y = y0
    z = z0
    valeursy =[ y0 ]
    valeursz =[ z0 ]
    h2sur2=h**2/2.
    for k in range ( n ):
        fi=f(valeursy[-1])
        y = y + h * valeursz [ -1]+h2sur2*fi
        z = z + h/2. * (fi+f(y))
        valeursy . append ( y )
        valeursz . append ( z )
    return  valeursy , valeursz
```

D.1)

$$y''(t) + \omega^2 y = 0$$

a) Donc $f(x) = -\omega^2 x$ et comme $g' = -f$ une primitive est bien $g(x) = \omega^2 \frac{x^2}{2}$. Inversement, en posant $g(x) = \omega^2 \frac{x^2}{2}$ on obtient la dérivée $-f$.

b) Schéma d'Euler : $\begin{cases} y_{i+1} = y_i + h\, z_i \\ z_{i+1} = z_i + h\, f(y_i) = z_i - h\omega^2 y_i \end{cases}$

L'équation $\frac{1}{2}y'(t)^2 + g\big(y(t)\big) = E$ s'approxime donc $\frac{1}{2}{z_i}^2 + \omega^2 \frac{{y_i}^2}{2} = E_i$ or $z_i = \frac{y_{i+1} - y_i}{h}$ donc l'équation (II.2) s'approxime par :

$$\frac{1}{2}\Big(\frac{y_{i+1} - y_i}{h}\Big)^2 + \omega^2 \frac{{y_i}^2}{2} = E_i \quad puis \quad \frac{1}{2}{z_i}^2 + \omega^2 \frac{{y_i}^2}{2} = E_i$$

Ainsi

$$\frac{1}{2}(z_{i+1})^2 + \omega^2 \frac{{y_{i+1}}^2}{2} = E_{i+1}$$

Donc

$$E_{i+1} - E_i = \frac{1}{2}{z_{i+1}}^2 + \omega^2 \frac{{y_{i+1}}^2}{2} - \frac{1}{2}{z_i}^2 - \omega^2 \frac{{y_i}^2}{2}$$

On utilise alors les relations de récurrence :

$$E_{i+1} - E_i = \frac{1}{2}\big((z_i - h\omega^2 y_i)^2 + \omega^2 (y_i + h\, z_i)^2 - z_i^2 - \omega^2 y_i^2\big)$$

$$E_{i+1} - E_i = \frac{1}{2}\big((h\omega^2 y_i)^2 + \omega^2 (h\, z_i)^2\big) = h^2\omega^2 \frac{1}{2}\big(z_i^2 + \omega^2 y_i^2\big)$$

$$E_{i+1} - E_i = h^2\omega^2\, E_i$$

Schéma de Verlet : $\begin{cases} y_{i+1} = y_i + h\, z_i + \frac{h^2}{2} f_i = \quad y_i + h\, z_i - \frac{h^2}{2}\, \omega^2 y_i \\ z_{i+1} = z_i + \frac{h}{2}(f_i + f_{i+1}) = z_i - \frac{h}{2}(\omega^2 y_i + \omega^2 y_{i+1}) \end{cases}$

$$E_{i+1} - E_i = \frac{1}{2}(z_{i+1})^2 + \omega^2 \frac{{y_{i+1}}^2}{2} - \frac{1}{2}(z_i)^2 - \omega^2 \frac{{y_i}^2}{2}$$

Donc

$$E_{i+1} - E_i = \frac{1}{2}\left(\left(z_i - \frac{h}{2}\left(\omega^2 y_i + \omega^2\left(y_i + h\, z_i - \frac{h^2}{2}\, \omega^2 y_i\right)\right)\right)^2 + \omega^2\left(y_i + h\, z_i - \frac{h^2}{2}\, \omega^2 y_i\right)^2 - z_i^2 - \omega^2 y_i^2\right)$$

En développant, les termes sans h, en h et en h^2 se simplifient donc on a bien $E_{i+1} - E_i = O(h^3)$.

c) Sachant que *h est < 1* l'erreur sur E (qui est théoriquement constant) entre 2 instants consécutifs est donc plus faible pour Verlet que pour Euler qui est en $O(h^2)$. .

De plus, (II.2) est une équation cartésienne d'ellipse donc théoriquement le portrait de phase est une ellipse .. on peut accepter un cercle. C'est davantage le cas pour le schéma de Verlet que celui d'Euler.

Pour aller plus loin, la formule de récurrence sur z_{i+1} fait apparaitre une prise d'information au pas $i + 1$ pour f. On se rapproche donc des avantages d'un schéma implicite. Le schéma de Verlet semble donc plus stable