intervalle de temps $[0, T_{max}]$:

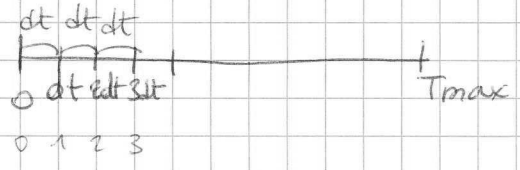
Q1: def init_T(Tmax, dt):

T = [0] if Tmax % dt == 0:

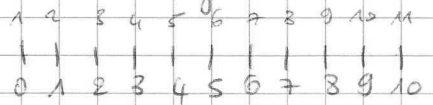
for i in range(1, ~~(Tmax//dt)+1~~ N): N = Tmax//dt + 1

T.append(i * dt) else: N = Tmax//dt + 2

return(T).



ex: On divise [0,10] en N intervalles réguliers de taille 1



→ soit 11 éléments

$\frac{10}{1} = 10$ N = 10 + 1

$\frac{T_{max}}{dt} = N$ on veut Tmax donc N' = N + 1

Si le dernier élément n'est pas Tmax alors N' = N+1 donne le dernier élément et N'' = N+2 donne l'élément suivant.

Q2: e(t) sera représenté sous forme de liste E[i] = e(ti)

def init_E(T, f):

N = len(T)

Emin = 1,3

Emax = 1,5

E = [] i = 0

while i < N and T[i] < 16 x 1/f :

E.append(Emin * sin(2π f T[i]))

i = i + 1

while i < N and T[i] < 2 x 16 x 1/f :

E.append(Emax * sin(2π f T[i]))

i = i + 1

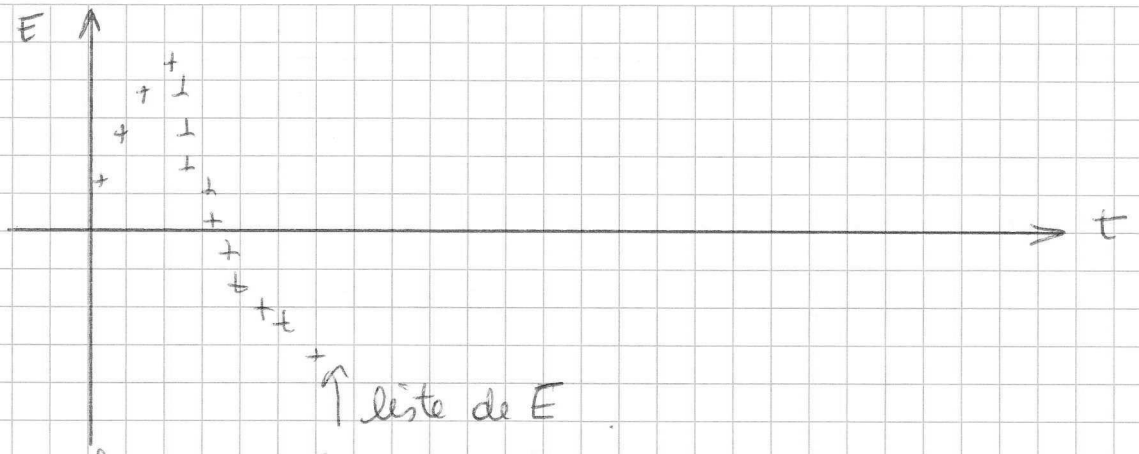
while i < N :

E.append(....) i = i + 1

return(E)

erreur du corrigé.

```
def init_E(T,f):
    N = len(T)
    E = []
    message = [0,1,0]
    for i in range(N):
        i = i//(16/f)
        val = message[i]
        if val == 0:
            E0 = 1,3
        else:
            E0 = 1,5
        E.append(E0 * sin(2π f T[i]))
    return(E)
```



il faut un détecteur d'enveloppe :

① diode bloquée $\frac{ds}{dt}(t) + \frac{1}{\tau} s(t) = 0$ et $s(t) > e(t)$

② diode passante $s(t) = e(t)$ et vérifie que $\frac{ds(t)}{dt} + \frac{1}{\tau} s(t) > 0$

$S[i] = s(t_i)$

Q3. $\frac{ds}{dt}(t_i) = \frac{s(t_{i+1}) - s(t_i)}{t_{i+1} - t_i}$

relation de récurrence quand la diode est bloquée à l'instant t_i

$$\frac{ds(t)}{dt} + \frac{1}{\tau} s(t) = 0$$

$$\frac{s(t_{i+1}) - s(t_i)}{\Delta t} + \frac{1}{\tau} s(t_i) = 0$$

OK

$$\boxed{s(t_{i+1}) = s(t_i)\left(1 - \frac{\Delta t}{\tau}\right)}$$

Q4 Diode passante en t_i, donc la relation est $s(t) = e(t)$

la valeur de $s(t_i) = e(t_i)$, en suivant cette démarche $s(t_{i+1}) = e(t_{i+1})$ et on ne vérifie pas que la relation est devenue nulle, il nous faudrait la valeur suivant t_{i+2}

erreur ds corrigé

$$\frac{ds(t_{i+1})}{dt} + \frac{1}{\tau} s(t_{i+1}) = 0$$

$$\frac{s(t_{i+1}) - s(t_i)}{\Delta t} + \frac{1}{\tau} s(t_{i+1}) = 0$$

Euler implicite $\frac{d(s(t_{i+1}))}{dt} = \frac{s(t_{i+1}) - s(t_i)}{\Delta t}$

$$s(t_{i+1})\left(\frac{1}{\Delta t} + \frac{1}{\tau}\right) = \frac{s(t_i)}{\Delta t}$$

$$\boxed{s(t_{i+1}) = \frac{\tau}{\Delta t + \tau} s(t_i)}$$

Q5: def sdve(T, E, tau):

S = [0] * len(T) delta_t = (T[1] - T[0])

diode = "passante"

~~if diode == "passante" :~~

for i in range(len(T)):

if diode == "passante" :

S[i] = E[i] , $S[i+1] = \frac{tau}{delta\_t + tau} S[i]$

~~S[i+1] = tau / (delta_t + tau) S[i] :~~

~~if S[i+1] < S[i] (1 - delta_t / tau) :~~ diode = "bloquee" S[i+1] >= E(i+1):

erreur du corrigé

else :

$S[i] = S[i-1] * (1 - \frac{delta\_t}{tau})$

$S[i+1] = S[i] * (1 - \frac{delta\_t}{tau})$

if S[i+1] <= E[i+1] :

diode = = "passante"

return (S)

Q6 : $\Delta t_1 = 100\,ns$ $\Delta t_1 = 0{,}1 . 10^{-6}$ donc 30 mesures

erreur du corrigé

$\Delta t_2 = 10\,ns$ $\Delta t_2 = 0{,}01 . 10^{-6}$ donc 300 mesures

$\Delta t_3 = 1\,ns$ $1\,ns = 1 . 10^{-9}\,s$ 3000 mesures

$T_{max} = 0{,}000003 = 3 . 10^{-6}\,s$

Q7 : cas $\Delta t_1 = 100\,ns$, euler explicite diverge .. resultats abherants.

cas $\Delta t_2 = 10\,ns$, une zone incertaine

cas $\Delta t_3 = 1\,ns$, répond exactement aux conditions.

Q8: résultat 1. inexploitable

résultat 2: ecart trop grand pour avoir 1,3V et 1,5V

résultat 3: On distingue clairement le maxi qui est tjs le même sur un intervalle

Δ 0,2V qui peut être vérifié.

variation de s(t) dans un intervalle de 0,1 V.

$\tau = 1\mu s$ $\quad \frac{\Delta t}{\tau}$ petit $\quad \frac{1ns}{1\mu s} = \frac{10^{-9}}{10^{-6}} = 10^{-3}$ facteur 1000

Signal RFID 13,56

2.1. bit de parité.

Q9 (erreur ds corrigé)

$(5)_{10} = (0101)_2 \Rightarrow$ bit de parité = 0

$(16)_{10} = (10000)_2 \Rightarrow$ bit de parité = 1

$(37)_{10} = (100101)_2 \Rightarrow$ ——— = 1

Q10:

```
def parite(bits):
    """ bits est une chaine de bits"""
    s=0
    for bit in bits:
        s=s+bit
    if s%2==0:
        return(0)
    else:
        return(1)
```

Q11: - permutation d'un 0 et un 1 (pas de détection)

- pas de correction possible car emplacement erroné non déterminé.

Code de Hamming :

code (7,4) ⇒ 3 bits de parité pour 4 bits de données

$(p_1, p_2, d_1, p_3, d_2, d_3, d_4)$

Q12 :

```
def encode_hamming(donnee):
    p1 = parite([donnee[0], donnee[1], donnee[3])
    p2 = parite([donnee[0], donnee[2], donnee[3])
    p3 = parite([donnee[1], donnee[2], donnee[3])
    return([p1, p2, donnee[0], p3, donnee[1], donnee[2],
            donnee[3]])
```

Contenu après réception :

$[c_1, c_2, c_3] = [0, 0, 0]$

Q13 :

```
def decode_hamming(message):
    c1 = parite([m4, m5, m6, m7]) = parite([p3, d2, d3, d4])
    c2 = parite([m2, m3, m6, m7]) = parite([p2, d1, d3, d4])
    c3 = parite([m1, m3, m5, m7]) = parite([p1, d1, d2, d4])

    if [c1, c2, c3] == [0, 0, 0]:
        return [d1, d2, d3, d4]
```

Si $\begin{matrix} c_1=0 \\ c_2=1 \\ c_3=1 \end{matrix}$ | pb sur d1

autre solution

position_erreur = c1*4 + c2*2 + c3

```
    else:
        if [c1, c2, c3] = [0, 1, 1]:
            print "erreur sur le bit d1"
            d1 = int((bin(d1) + bin('1'))[0])
        if [c1, c2, c3] = [1, 1, 0]:
            print "erreur sur le bit d3"
            d3 = int((bin(d3) + bin('1'))[0])
```

if $[c_1, c_2, c_3] = [1, 0, 1]$:
    print "erreur sur le bit d2"

if $(c_1, c_2, c_3) = (1, 1, 1)$:
    print "erreur sur le bit d_4"

return $[d_1, d_2, d_3, d_4]$

Q14 : donnée (1011)

encode_hamming(1011) = (0, 1, 1, 0, 0, 1, 1)

transmission ~~(0, 1, 0, 0, 1, 1, 1)~~

erreur de corrigé

$c_1 = 0$ $c_2 = 1$ $c_3 = 1$ (1, 0, 1, 0, 0, 1, 1)

la correction (faite dans le programme précédent

~~(0, 1, 0, 0, 1, 0, 1)~~ ↑

1 bit ~~de~~ est faux. (1, 0, 0, 0, 0, 1, 1)
alors qu'il n'y avait pas d'erreur
message (0, 0, 1, 1)

Q15 : bit de parité sur ~~les~~ bits de contrôle
non ! non, puisque inversion de p_1 et p_2

Utilisation des données de la puce pour autoriser ou non le passage

Q16 id_titre : entier [49987654]

zones : liste [1, 3]

date_fin : liste [année, mois, jour]
[2015, 08, 31]

Q17.

```
fichier = open('tableau.csv', 'w')
fichier.write('indice' + ; + '0' + ; + '1' + ... \n)
fichier.write('contenu' + ; +

for i in range(2,5):
    donnees = ligne[i].rstrip('\n').split(';')
    temps = donnees[0].split('-')
    fichier.write('contenu' + ; + temps[0] ... )
```

Q18.

```
def estAvant(date1, date2):
    Res = False
    if date1[0] < date2[0]:
        return(True)  Res = True
    if date1[0] == date2[0]:
        if date1[1] < date2[1]:
            return(True)  Res = True
        if date1[1] == date2[1]:
            if date1[2] < date2[2]:
                return(True)  Res = True
    return(Res)
```

autre

```
if date1[2] < date2[2]:
    if date1[1] <= date1[1]:
        if date1[0] <= date1[0]:
            return True
else:
    return False
```

~~else~~

~~return (False)~~

~~else:~~

~~return (False)~~

~~else:~~

~~return (False)~~

Q19 calcul de secondes.

Q20. def Ht_passage (~~date1, date2,~~ donnees_carte) :

    liste_noire
    (date1, date2)