CORRIGE DS4 (MINES 2021)
MARCHONS

PARTIE 1

Q1

```
SELECT COUNT(*) FROM Participant WHERE ne BETWEEN 1999 AND 2003 ;
```

Q2

```
SELECT diff, AVG(duree)FROM Rando GROUP BY diff ;
```

Q3

```
SELECT pnom FROM Participant WHERE diff_max > (SELECT diff FROM Rando WHERE rid = 42) ;
```

Q4

```
SELECT DISTINCT rid FROM Rando WHERE rnom IN (SELECT rnom FROM Rando GROUP BY rnom
HAVING COUNT(*) > 1 ) ;
```

Q5

```
def importe_rando(nom_fichier):
    coords = []
    fichier = open(nom_fichier, "r")
    ligne = fichier.readline()
    ligne = fichier.readline()
    while ligne:
        valeurs = ligne.split(',')
        coords.append([float(v) for v in valeurs])
        ligne = fichier.readline()
    fichier.close()
    return coords
```

Q6

```
def plus_haut(coords):
    lat, lon = coords[0][0], coords[0][1]
    maxi = coords[0][2]
    for pt in coords[1:]:
        if pt[2] > maxi:
            lat, lon  = pt[0], pt[1]
            maxi = pt[2]
    return [lat, lon]
```

Q7

```
def deniveles(coords):
    positif, negatif = 0, 0
    for i in range(len(coords)-1):
        delta = coords[i+1][2] - coords[i][2]
        if delta > 0:
            positif += delta
        else:
            negatif -= delta
    return [positif, negatif]
```

Q8

```
def distance(c1, c2):
    Rt = 6371e3 + (c1[2] + c2[2]) /2
```

```
    phi_1, lambda_1 = radians(c1[0]), radians(c1[1])
    phi_2, lambda_2 = radians(c2[0]), radians(c2[1])

    d_phi = (phi_2 - phi_1) / 2
    d_lambda = (lambda_2 - lambda_1) /2

    racine = (sin(d_phi))**2 + cos(phi_1) * cos (phi_2) * (sin(d_lambda))**2
    d = 2 * Rt * asin(sqrt(racine))

    h = c1[2] - c2[2]

    return sqrt(d**2 + h**2)
```

Q9

```
def distance_totale(coords):
    d = 0
    for i in range(len(coords)-1):
        d += distance(coords[i], coords[i+1])
    return d
```

PARTIE 2

Q10

```
def vma(v1, a, v2):
    assert len(v1) == len(v2)
    v = []
    for i in range(len(v1)):
        v.append(v1[i] + a * v2[i])
    return v
```

Q11

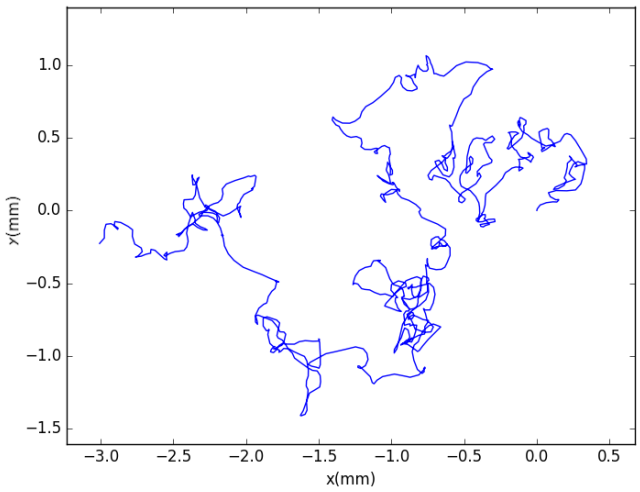
```
from random import uniform, gauss
from math import cos, sin, sqrt, pi

def derive(E):
    norme = gauss(MU, SIGMA)
    direction = uniform(0, 2*pi)
    fb = [norme * cos(direction) / M, norme * sin(direction) / M]
    E1 = E[2] / M , E[3] / M
    E2 = vma(fb, -ALPHA, E1)
    return E[2:4] + E2
```

Q12

```
def euler(E0, dt, n):
    Es = [E0]

    for i in range(n):
        E1 = derive(Es[-1])
        Es.append([Es[-1][0] + dt * E1[0],
                   Es[-1][1] + dt * E1[1],
                   Es[-1][2] + dt * E1[2],
                   Es[-1][3] + dt * E1[3],])
    return Es
```

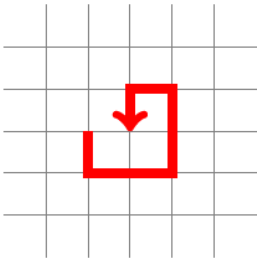


PARTIE 3

Q13

```
from random import randrange, choice
#positions auto-evitantes suivantes possibles
def position_possibles(p, atteints):
    possibles = []
    for dx, dy in ((-1,0), (1,0), (0,-1), (0,1)):
        if [p[0] + dx, p[1] + dy] not in atteints:
            possibles.append([p[0] + dx, p[1] + dy])
    return possibles
```

Q14



Q15

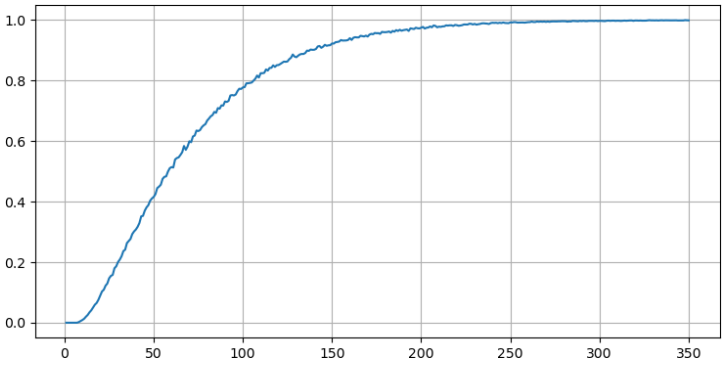
```
def genere_chemin_naif(n):
    chemin = [[0, 0]] # on part de l'origine
    echoue = False
    while len(chemin) < n and not echoue:
        possibles = position_possibles(chemin[-1] , chemin)
        if len(possibles) == 0:
            echoue = True
        else:
            chemin.append(choice(possibles))
    if echoue:
        return None
    return chemin
```

Q16

n appel de positions possibles, pour chacun 4 tests provoquant une recherche séquentielle.
Donc la complexité est en $O(n^2)$

Q17

Ce graphique représente la probabilité moyenne (avec 10000 essais) que la recherche d'un chemin échoue (la fonction position_possibles retourne None) en fonction de la longueur du chemin de 0 à 350 étapes.



Q18

La complexité de sorted est $O(n.\ \log\ n)$. Un algorithme utilisable est le tri fusion.

Q19

```
def est_CAE(chemin):
    ordone = sorted(chemin)
    auto_evitant = True
    i = 1
    while i < len(chemin) and auto_evitant:
        auto_evitant = (ordone[i] != ordone[i-1])
        i += 1
    return auto_evitant
```

Q20

```
def rot(p, q, a):
    dx = p[0] - q[0]
    dy = p[1] - q[1]
    if a==0: # pi
        return [p[0] + dx, p[1] + dy]
    elif a == 1: # +pi/2
        return [p[0] + dy, p[1] + dx]
    else: # # -pi/2
        return [p[0] - dy, p[1] - dx]
```

Q21

```
def rotation(chemin, i_piv, a):
    pivot = chemin[i_piv]
    for i in range(i_piv+1, len(chemin)):
        chemin[i] = rot(pivot, chemin[i], a)
    return chemin
```

Q22

```
def genere_chemin_pivot(n, n_rot):
    chemin = [[i, 0] for i in range(n)]
    for i in range(n_rot):
        auto_evitant = False
        while not auto_evitant:
            copie = chemin.copy()
            i_piv = randrange(1, n-1)
            a = randrange(0, 3)
            rotation(copie, i_piv, a)
            auto_evitant = est_CAE(copie)
        chemin = copie
    return chemin
```

Q23

Il faut tenir compte de l'orientation du dernier segment par rapport au précédent pour ne choisir que parmi les rotations possibles.