```
#%% TP 13

import random as rd

#%% exo 1

# q1
def lancer_de():
    return rd.randint(1,6)

# q2
def tirer():
    urne = ['R','R','R','B','B','V']
    return rd.choice(urne)

# q3
def tirer_suc(n):
    return [tirer() for k in range(n)]

# q4
def tirer_sim(n):
    urne = ['R','R','R','B','B','V']
    return rd.sample(urne,n)

# q5
def hello(p):
    t = rd.random()
    if t<p :
        return "bonjour"
    else :
        return "au revoir"

# q6
def jour():
    t = rd.random()
    if t<0.2 :
        return "lundi"
    elif t<0.9 :
        return "mardi"
    else :
        return "mercredi"

#%% exo 2

# q1
def repete_de(N):
    X = 0
    for k in range(N):
        res = lancer_de()
        if res == 6:
            X = X+1
    return X

# q2
def frequence(N):
    return repete_de(N)/N

# q3
# frequence(10000) renvoie environ 1/6

# q4
absi = [N for N in range(1,2000)]
ordo = [frequence(N) for N in absi]
import matplotlib.pyplot as plt
plt.plot(absi,ordo)
plt.show()
```

```
#%%
# q5
X = 0
N = 10000
for k in range(N):
    if jour()=="lundi":
        X = X+1
print(X/N)

#%%
# q6
def nb_pair(n,d):
    urne = [k for k in range(1,n+1)]
    liste_tires = rd.sample(urne,d)
    Y = 0
    for x in liste_tires:
        if x%2==0:
            Y = Y+1
    return Y

# q7
X = 0
N = 10000
for k in range(N):
    if nb_pair(10,2)==0:
        X = X+1
print(X/N)

#%% exo 3

# q1

# q2
def souris(n):
    pos = "G"
    for k in range(n):
        if pos == "S":
            pos = "S"
        elif pos == "G":
            pos = rd.choice(["G","G","S","C"])
        elif pos == "C":
            pos = rd.choice(["C","C","S","G"])
    return pos

# q3
def proba(n,N):
    X_G,X_C,X_S = 0,0,0
    for k in range(N):
        pos = souris(n)
        if pos == "G":
            X_G = X_G+1
        if pos == "S":
            X_S = X_S+1
        if pos == "C":
            X_C = X_C + 1
    return X_G/N, X_C/N, X_S/N

# q4
# g(n+1) = 0,5*g(n) + 0,25*c(n)
# c(n+1) = 0,5*c(n) + 0,25*g(n)
# s(n+1) = s(n) + 0,25*c(n) + 0,25*g(n)

# q5
def proba_exacte(n):
    g,c,s = 1,0,0
    for k in range(n):
        new_g = g/2 + c/4
```

```
        new_c = c/2 + g/4
        new_s = s + g/4 + c/4
        g,c,s = new_g,new_c,new_s
    return g,c,s

# on constate numériquement que plus N est grand plus proba(6,N) est proche de
# la valeur exacte proba_exacte(6)

#%% exo 4

# q1
def pas():
    t = rd.random()
    if t<0.15:
        return 1
    elif t<0.4:
        return 2
    else :
        return 3

# q2
def position():
    x = 0
    for k in range(10):
        x = x + pas()
    return x

# q3
X = 0
N = 100000
for k in range(N):
    if position()==25:
        X = X+1
print(X/N)
# on trouve environ 0,166

# q4
# on fait la moyenne des positions
pos = 0
N = 100000
for k in range(N):
    pos = pos + position()
print(pos/N)
# on trouve environ 24,5

#%% exo 5

# q1
# det(A)=ad-bc
# A est inversible si et seulement si det(A) != 0

# q2
import numpy as np
def inversible(A) :
    det = A[0,0]*A[1,1] - A[0,1]*A[1,0]
    return det != 0
# à tester avec A = np.array([[1,2],[3,4]])

# q3
def matrice_alea():
    a = rd.random()
    b = rd.random()
    c = rd.random()
    d = rd.random()
    return np.array([[a,b],[c,d]])

# q4
```

```
X = 0
N = 10000
for k in range(N):
    A = matrice_alea()
    if inversible(A):
        X = X+1
print(X/N)
# on obtient une probabilité de 1 :
# chacune des 10 000 matrices aléatoires testées est
# inversible ! En même temps, en prenant a,b,c,d au hasard
# on a peu de chance que a*d = b*c. Tellement peu de chance
# que la probabilité que cela arrive est nulle.
# Cependant, cela ne signifie pas que toutes les matrices
# sont inversibles ; mais seulement que la probabilité
# de trouver une matrice non inversible au hasard est nulle.

#%% exo 6

# q1
def position(nb_pas) :

        x = [0]
        y = [0]
        choix = [(0,1), (0,-1), (1, 0), (-1, 0)]

        for i in range(1,nb_pas):
            pos = rd.choice(choix)
            x.append(x[i-1] + pos[0])
            y.append(y[i-1] + pos[1])

        return (x,y)


def trajet(nb_pas, N) :

    for i in range(N) :
           (x,y) = position(nb_pas)
           plt.plot(x,y)
    plt.show()

nb_pas=500
N=100
trajet(nb_pas, N)

# on remarque que la trajectoire de l'ivrogne semble bornée
# en fait la probabilité que l'ivrogne aille "loin" est "faible"

def retour(nb_pas):
    x = 0
    y = 0
    choix = [(0,1), (0,-1), (1, 0), (-1, 0)]
    k = 0
    while k<nb_pas :
        k += 1
        deplacement = rd.choice(choix)
        x += deplacement[0]
        y += deplacement[1]
        if x==0 and y==0 :
            return k
    return nb_pas

def moyenne(nb_pas_max,K):
    som = 0
    for k in range(K):
        som += retour(nb_pas_max)
    return som/K
```

```
# il faut prendre nb_pas_max très grand pour approcher
# le vrai temps de retour moyen à l'origine.

# En fait, la variable aléatoire T égal au temps de premier
# retour à l'origine n'est pas une variable aléatoire finie :
# T est à valeurs dans N (l'ensemble des entiers naturels)
# tout entier. L'étude de son espérance dépasse donc en fait
# le cadre mathématique de la première année de BCPST
# (mais pas celui de la deuxième !)

#%% exo 7

# q1
# pi

# q2
# P(M appartient à D) = aire(D)/aire(carré) = pi/4

# q3
# on simule l'expérience consistant à placer M au hasard
# dans le carré (aka : tirer une fléchette) et on estime
# la probabilité que M appartienne à D (aka : la probabilité
# d'atteindre la cible). Cette probabilité est une approximation
# de pi/4 donc en multipliant par 4 on a une approximation de pi

def tir_flechette():
    x = rd.uniform(-1,1)
    y = rd.uniform(-1,1)
    return x,y

def cible_atteinte():
    x,y = tir_flechette()
    return x**2 + y**2 < 1

X = 0
N = 10000
for k in range(N):
    if cible_atteinte():
        X = X+1
print(4*X/N)
```