```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 19 08:36:20 2017

@author: stephane
"""

from random import randint
import matplotlib.pyplot as pypl
from math import sqrt, exp, sqrt, pi
import numpy as np
from scipy.integrate import quad


"""
PREMIÈRE PARTIE
"""


def tirage_de():
    return randint(1, 6)


def serie(n):
    return sum(tirage_de() for _ in range(n))/float(n)

"""
>>> serie(1000)
3.518
>>> serie(1000)
3.44
"""

plein_de_series = [serie(10**2) for _ in range(10**2)]


print sum(plein_de_series)/len(plein_de_series),\
          max(plein_de_series), min(plein_de_series),\
          len([_ for m in plein_de_series if m <= 3.4]),\
          len([_ for m in plein_de_series if m >= 3.6]),\
          len([_ for m in plein_de_series if m <= 3]),\
          len([_ for m in plein_de_series if m >= 4])

"""
10**3 -> 3.49024   3.99 3.0    308 271    1   0
10**4 -> 3.500459  4.12 2.88  2854 2903   19  14
10**5 -> 3.5008688 4.22 2.69 28695 29215 183 176
"""

"""
DEUXIÈME PARTIE
"""


def ab_series(a, b):
    res = [0]
    for i in range(a):
        res.append((i*res[-1] + serie(b)) / (i+1))
    return res

"""
>>> ab_series(5, 200)
```



```
[0, 3.425, 3.4725, 3.5383333333333336, 3.51875, 3.5089999999999995]
>>> ab_serie(5, 200)
[0, 3.4, 3.48, 3.4599999999999995, 3.47875, 3.501]

>>> ab_series(10, 10**3)
[0, 3.553, 3.4835000000000003, 3.4790000000000005, 3.45575, 3.4631999999999996,
3.449666666666667, 3.461, 3.463375, 3.4623333333333335, 3.4707999999999997]
>>> ab_series(10, 10**4)
[0, 3.4867, 3.48735, 3.4921666666666664, 3.491225, 3.4900800000000003,
3.493466666666667, 3.502457142857143, 3.5039125, 3.5057111111111112,
3.5082700000000004]
>>> ab_series(10, 10**5)
[0, 3.49999, 3.4999700000000002, 3.4999633333333335, 3.5016825000000003,
3.5016240000000005, 3.5021850000000003, 3.5027257142857144, 3.50256375,
3.5021500000000003, 3.502085]
>>> ab_series(10, 10**6)
[0, 3.499063, 3.497649, 3.4979443333333333, 3.49911025, 3.4993698,
3.4992121666666667, 3.499449, 3.499578125, 3.499527555555556,
3.4998343000000007]
"""

sigma = sqrt(35./12)

"""
a = 10**2
b = 10**5

x = range(b, (1+a)*b, b)    # On ne prend pas en compte n=0 (ben oui...)
pypl.plot(x, ab_series(a, b)[1:])  # Même chose
pypl.plot([b, (1+a)*b], [3.5, 3.5])
pypl.plot(x, [3.5+sigma/sqrt(i) for i in x])
pypl.plot(x, [3.5−sigma/sqrt(i) for i in x])

pypl.grid()
pypl.xlim([0, (1+a)*b])

pypl.savefig('moyenne'+str(a*b)+'.pdf')
pypl.show()
pypl.clf()
"""
"""
TROISIÈME PARTIE
"""



n = 100
nb_series = 10**2

plein_de_series = [serie(n) for _ in range(nb_series)]

x = [i/float(n) for i in range(250, 451)]  # On s'intéresse à la bande centrale
y = [plein_de_series.count(i) for i in x]
my = float(max(y))

pypl.plot(x, [j/my for j in y])

x = np.arange(2, 5, 0.01)
y = np.exp(-n*(x-3.5)**2/(2*sigma**2))
"""
Si vous n'avez pas tiqué, c'est que vous avez mal lu ! Relisez ce qui précède !
Strange, non ? C'est la magie des fonctions vectorialisées de numpy.
"""
```

```
pypl.plot(x, y)

pypl.savefig('tcl' + str(nb_series) + '.pdf')
pypl.clf()


def f(x):
    return exp(-x**2/2)/sqrt(2*pi)

print quad(f, -10, (3.4-3.5)*10/sigma)[0]

# 0.279092324711  C'est la proportion limite de séries avec une moyenne <3.4

print quad(f, -10, (3-3.5)*10/sigma)[0], quad(f, (4-3.5)*10/sigma, 10)[0]
# 0.00170739558906 0.00170739558906
#
# Relisez les résultats de la fin de la première partie

"""
CINQUIÈME PARTIE
"""


def arrets(p, n):  # p étages n personnes
    stop = [False] * p
    for _ in range(n):
        stop[randint(0, p-1)] = True
    return sum([1 for x in stop if x])
    # Oui, bon... une boucle ou un count feraient aussi bien l'affaire


def simulation(p, n, nbexp):
    return sum(arrets(p, n) for _ in range(nbexp)) / float(nbexp)


def espe_theo(p, n):
    return p*(1-(1-1/float(p))**n)

"""
>>> simulation(15, 10, 10**4), espe_theo(15, 10)
(7.4923, 7.475822621159781)
>>> simulation(15, 10, 10**4)
7.47449
>>> simulation(30, 10, 10**4), espe_theo(30, 10)
(8.6211, 8.625858187257986)
>>> simulation(100, 10, 10**4), espe_theo(100, 10)
(9.5657, 9.561792499119559)
"""
```