**Valeriy Lazarenko, Data Science Camp Test Tasks**

**Task 1:**

X = (0, -1, 1)

[Source code](#task1)

**Task 2:**

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

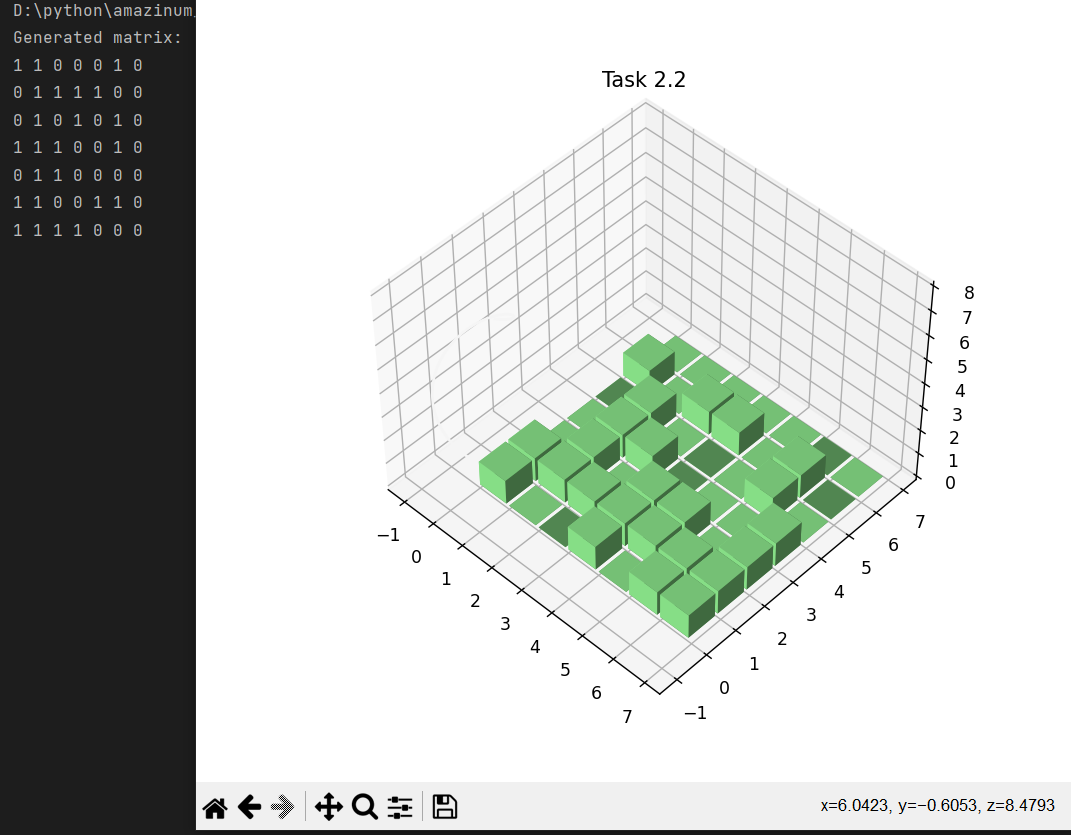
1 0 0 0 0 0 1

[Source code](#task2)

**Task 2.1:**

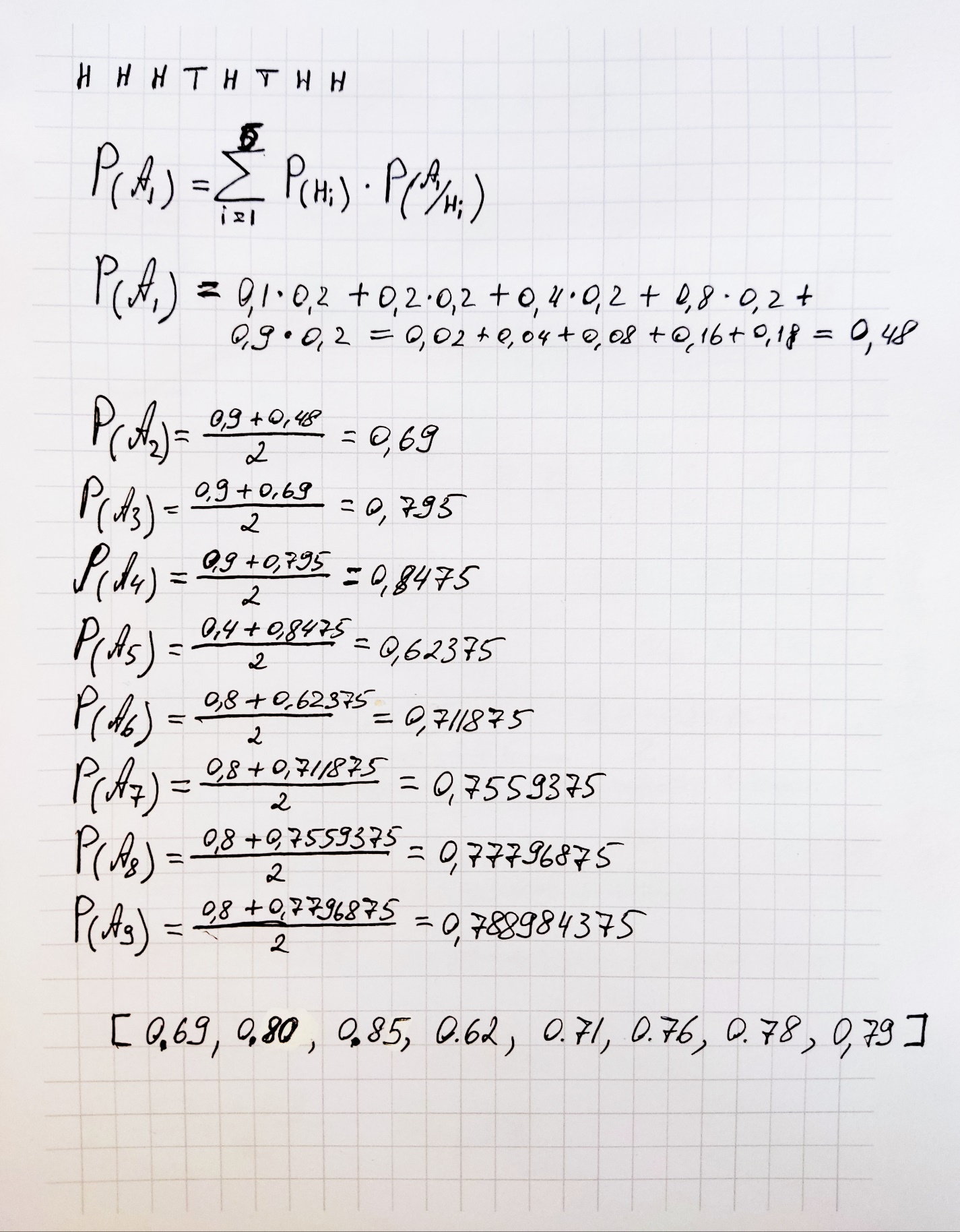
[Source code](#task21)

**Task 2.2:**

****

[Source code](#task22)

**Task 3:**



**Code of task 1:**

def input\_matrix():  
 A = [[], [], []]  
 for i in range(3):  
 print("Enter %d line of matrix A and divide nums by space: " % (i+1))  
 A[i] = list(map(int, input().split(" ")))  
  
 B = input("Enter matrix B and divide nums by space: ").split(" ")  
 B = list(map(int, B))  
  
 caltulate(A, B)  
  
  
def caltulate(A, B):  
 determinant = A[0][0] \* A[1][1] \* A[2][2] + A[0][1] \* A[1][2] \* A[2][0] + A[0][2] \* A[1][0] \* A[2][1] - \  
 A[0][2] \* A[1][1] \* A[2][0] - A[0][1] \* A[1][0] \* A[2][2] - A[0][0] \* A[1][2] \* A[2][1]  
  
 determinant1 = B[0] \* A[1][1] \* A[2][2] + B[1] \* A[0][2] \* A[2][1] + B[2] \* A[0][1] \* A[1][2] - \  
 B[2] \* A[1][1] \* A[0][2] - B[1] \* A[0][1] \* A[2][2] - B[0] \* A[1][2] \* A[2][1]  
  
 determinant2 = (A[0][0] \* B[1] \* A[2][2]) + (A[1][0] \* B[2] \* A[0][2]) + (A[2][0] \* B[0] \* A[1][2]) - \  
 (A[0][2] \* B[1] \* A[2][0]) - (A[1][0] \* B[0] \* A[2][2]) - (A[0][0] \* B[2] \* A[1][2])  
  
 determinant3 = A[0][0] \* A[1][1] \* B[2] + A[1][0] \* A[2][1] \* B[0] + A[2][0] \* A[0][1] \* B[1] - \  
 A[2][0] \* A[1][1] \* B[0] - A[0][1] \* A[1][0] \* B[2] - A[2][1] \* A[0][0] \* B[1]  
  
 if determinant != 0:  
 x = determinant1 / determinant  
 y = determinant2 / determinant  
 z = determinant3 / determinant  
 print("X = (%d, %d, %d)" % (x, y, z))  
 elif determinant == 0 and (determinant1 != 0 or  
 determinant2 != 0 or  
 determinant3!= 0):  
 print("Matrix don't have solution.")  
 elif determinant == 0 and determinant1 == 0 \  
 and determinant2 == 0 and determinant3 == 0:  
 print("Matrix have an infinite number of solutions.")  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 input\_matrix()

[Go back](#first)

**Code of task 2:**

def life(matrix, rows, cells, lifeCycle):  
 # Array for counting of cell's neighbours  
 neighbours = [[0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0]]  
  
 for iteration in range(lifeCycle):  
 # This cycle counts neighbours  
 for row in range(rows):  
 for cell in range(cells):  
 if matrix[row][cell] == 1:  
 if cell < 6:  
 neighbours[row][cell + 1] += 1  
 if cell > 0:  
 neighbours[row][cell - 1] += 1  
  
 for ncell in range(-1, 2):  
 position = cell + ncell  
 if row < 6 and 6 >= position >= 0:  
 neighbours[row + 1][position] += 1  
 if row > 0 and 6 >= position >= 0:  
 neighbours[row - 1][position] += 1  
  
 # This cycle check neighbours' amount and change cell's status  
 for row in range(rows):  
 for cell in range(cells):  
 if matrix[row][cell] == 1:  
 if neighbours[row][cell] <= 1 or neighbours[row][cell] >= 4:  
 matrix[row][cell] = 0  
 if matrix[row][cell] == 0 and neighbours[row][cell] == 3:  
 matrix[row][cell] = 1  
  
 # This cycle print matrix in console  
 for row in range(rows):  
 for cell in range(cells):  
 print(matrix[row][cell], end=' ')  
 print("")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 matrix = [[1, 0, 0, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 1, 1],  
 [1, 0, 0, 1, 0, 0, 1],  
 [0, 1, 1, 0, 1, 1, 0],  
 [1, 1, 1, 1, 0, 0, 1],  
 [1, 1, 1, 1, 1, 1, 1],  
 [1, 1, 0, 1, 1, 0, 1]]  
 # life(matrix\_name, rows in matrix, columns, iterations)  
 life(matrix, len(matrix), len(matrix[0]), 7)

[Go back](#first)

**Code of task 2.1:**

import random  
  
  
def show\_matrix(matrix):  
 for row in range(len(matrix)):  
 for cell in range(len(matrix)):  
 print(matrix[row][cell], end=' ')  
 print("")  
 print("")  
  
  
def generate\_matrix():  
  
 matrix = []  
  
 for i in range(7):  
 a = []  
 for j in range(7):  
 a.append(random.randint(0, 1))  
 matrix.append(a)  
  
 print("Generated matrix: ")  
 show\_matrix(matrix)  
  
 # life(matrix\_name, size, iterations)  
 life(matrix, 7)  
  
  
def life(matrix, size):  
 # Array for counting of cell's neighbours  
 neighbours = [[0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0]]  
  
 while True:  
 # This cycle counts neighbours  
 for row in range(size):  
 for cell in range(size):  
 if matrix[row][cell] == 1:  
 if cell < 6:  
 neighbours[row][cell + 1] += 1  
 if cell > 0:  
 neighbours[row][cell - 1] += 1  
  
 for ncell in range(-1, 2):  
 position = cell + ncell  
 if row < 6 and 6 >= position >= 0:  
 neighbours[row + 1][position] += 1  
 if row > 0 and 6 >= position >= 0:  
 neighbours[row - 1][position] += 1  
  
 # This cycle check neighbours' amount and change cell's status  
 for row in range(size):  
 for cell in range(size):  
 if matrix[row][cell] == 1:  
 if neighbours[row][cell] <= 1 or neighbours[row][cell] >= 4:  
 matrix[row][cell] = 0  
 if matrix[row][cell] == 0 and neighbours[row][cell] == 3:  
 matrix[row][cell] = 1  
  
 answer = input("Generate the next iteration? (y/n) ")  
 if answer == "y":  
 print("Changed matrix: ")  
 show\_matrix(matrix)  
  
 for var1 in range(len(neighbours)):  
 for var2 in range(len(neighbours[var1])):  
 neighbours[var1][var2] = 0  
 else:  
 quit()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 generate\_matrix()

[Go back](#first)

**Code of task 2.2:**

import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def generate\_matrix():  
 matrix = []  
  
 for i in range(7):  
 a = []  
 for j in range(7):  
 a.append(random.randint(0, 1))  
 matrix.append(a)  
  
 print("Generated matrix: ")  
 show\_matrix(matrix)  
 show\_matrix\_chart(matrix)  
  
 # life(matrix\_name, size, iterations)  
 life(matrix, 7, 1)  
  
  
def life(matrix, size, lifeCycle):  
 # Array for counting of cell's neighbours  
 neighbours = [[0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0]]  
  
 for iteration in range(lifeCycle):  
 # This cycle counts neighbours  
 for row in range(size):  
 for cell in range(size):  
 if matrix[row][cell] == 1:  
 if cell < 6:  
 neighbours[row][cell + 1] += 1  
 if cell > 0:  
 neighbours[row][cell - 1] += 1  
  
 for ncell in range(-1, 2):  
 position = cell + ncell  
 if row < 6 and 6 >= position >= 0:  
 neighbours[row + 1][position] += 1  
 if row > 0 and 6 >= position >= 0:  
 neighbours[row - 1][position] += 1  
  
 # This cycle check neighbours' amount and change cell's status  
 for row in range(size):  
 for cell in range(size):  
 if matrix[row][cell] == 1:  
 if neighbours[row][cell] <= 1 or neighbours[row][cell] >= 4:  
 matrix[row][cell] = 0  
 if matrix[row][cell] == 0 and neighbours[row][cell] == 3:  
 matrix[row][cell] = 1  
  
 print("Changed matrix: ")  
 show\_matrix(matrix)  
 show\_matrix\_chart(matrix)  
 ask\_user()  
  
  
def show\_matrix(matrix):  
 for row in range(len(matrix)):  
 for cell in range(len(matrix)):  
 print(matrix[row][cell], end=' ')  
 print("")  
 print("")  
  
  
def show\_matrix\_chart(matrix):  
 fig = plt.figure(figsize=(7, 7))  
 ax1 = fig.add\_subplot(projection='3d')  
  
 \_x = np.arange(7)  
 \_y = np.arange(7)  
 \_xx, \_yy = np.meshgrid(\_x, \_y)  
 x, y = \_xx.ravel(), \_yy.ravel()  
 \_tup = np.array(matrix)  
 top = \_tup[x, y]  
 bottom = np.zeros\_like(top)  
 width = depth = 0.9  
  
 z2 = np.linspace(0, 8)  
 x2 = np.sin(z2)  
 y2 = np.cos(z2)  
  
 ax1.bar3d(x, y, bottom, width, depth, top, color='#98FB98')  
 ax1.plot3D(x2, y2, z2, '#f5f5f5')  
 ax1.set\_title('Task 2.2')  
  
 plt.show()  
  
  
def ask\_user():  
 answer = input("Generate the next matrix? (y/n) ")  
 if answer == "y":  
 generate\_matrix()  
 else:  
 quit()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 generate\_matrix()

[Go back](#first)

**Code of task 3:**

def count\_full\_probability(coins1, tr):  
 probabilities = [0.48]  
  
 for i in range(len(tr)):  
 p = len(list(filter(lambda x: x == 1, tr[:i+1]))) / len(tr[:i+1])  
   
 for j in range(len(coins1) - 1):  
 if coins1[j] <= p <= coins1[j+1]:  
 a = coins1[j+1]  
 break  
 else:  
 a = coins1[-1]  
 prob = (probabilities[i] + a) / 2  
 probabilities.append(round(prob, 2))  
  
 print("Probabilities = ", probabilities)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 coins = [0.1, 0.2, 0.4, 0.8, 0.9]  
 tests\_res = [1, 1, 1, 0, 1, 0, 1, 1]  
 count\_full\_probability(coins, tests\_res)

[Go back](#first)