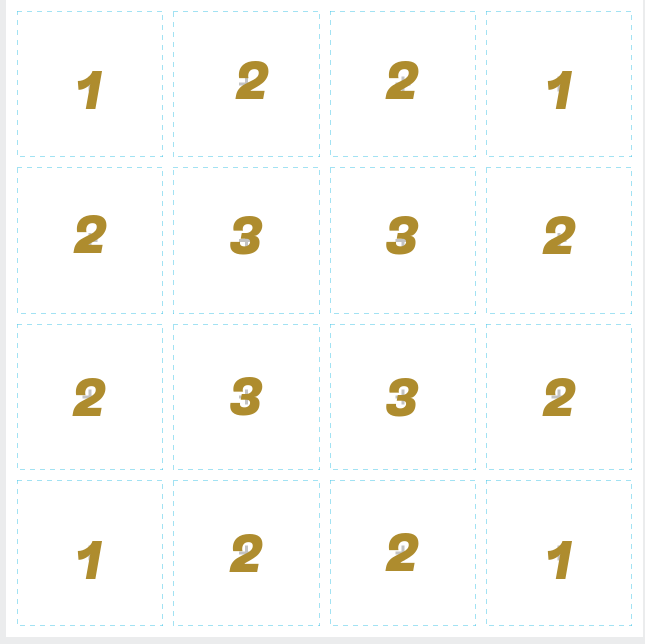
Game of Life Description

Question 1: Program for the Game of Life

Imagine every point in a grid is a life. For a point *A* in the grid, there are 8 points around it. If there is a black point, it means that the point is alive (otherwise it's dead). If 3 of the 8 points are alive, then the number of point *A* is 3 ("3" represents the number of living points within 8 surrounding nodes).

For example, Figure 1(a) is a grid with four rows and four columns. The corresponding number grid for Figure 1(a) is Figure 1(b).



1. Status for living and dead nodes
2. Living number of current status

Figure 1. Living number for every point In a grid

The living nodes in a grid changes generation by generation according to the number of neighbors that are ***alive***, as follows:

1. The neighbor of a given node are the eight nodes that touch it vertically, horizontally, or diagonally.
2. If a node is alive but either has no neighboring nodes alive or only one alive, then in the next generation the node dies of loneliness.
3. If a node is alive and has four or more neighboring nodes also alive, then in the next generation the node dies of overcrowding.
4. A living node with **either two or three** living neighbors remains alive in the next generation.
5. If a node is dead, then in the next generation it will become alive if it has **exactly three** neighboring nodes, no more, no fewer, that are already alive. All other dead nodes remain dead in the next generation.
6. All births and deaths take place at exactly the same time, so that dying nodes can help to give birth to another, but cannot prevent the death of others by reducing overcrowding; nor can nodes being born either preserve or kill nodes living in the previous generation.

An example is shown in Figure 2. Your goal is to write a python program that will show how an initial configuration will change from generation to generation



1. Initial status (b) after one generation (c)after two generations

Figure 2. Changes of generation

***Important: Sample input and Output***

The coordinates of one point in a grid is represented as follows:

1. The first row is identified by 0. Identification for a row increases.
2. The first column is identified by 0. Identification for a column increases.

As an example, in Figure 2(a), the coordinates for these three black points (living nodes) **from top to bottom** are:

The top node: (1,2)

The middle node: (2,2)

The bottom node: (3,2)

Your goal is to simulate the status for every node in this grid (no matter alive or dead). For the output, you have to output **1** if it's a living node, or **0** if it's dead. Your program should output ***every line*** of final generation as a ***list***.

To simplify the problem, **you only need to handle a board of 5 x 5**. What’s more, to avoid handling boundary cases, please make sure that all your living nodes’ coordinates ranges from row 1 to row 3, column 1 to column 3, which means that a node with either row 0 or column 0 are living node.

Use Figure 2(a) as an example to illustrate what is the input format for your program. If your initial grid is Figure 2(a), In this example, if you want to output the status after two generation, you input should be like:

5

3

2

1

2

2

2

3

2

t

The explanation for these inputs is:

5 --- it's a 5\*5 grid (**Your program should always use 5 as the first input**)

3 --- initial living node number (how much living node in an initial status)

2 --- generation number (how many generations have gone through)

1 --- The abscissa of the first living point

2 --- The ordinate of the first living point

2 --- The abscissa of the second living point

2 --- The ordinate of the second living point

3 --- The abscissa of the third living point

2 --- The ordinate of the third living point

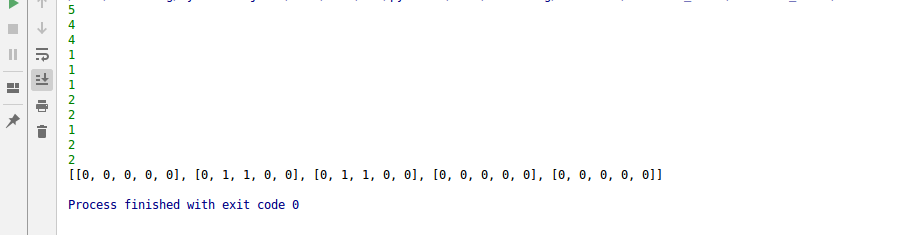
The output should be:

[[0, 0, 0, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 1, 0, 0], [0, 0, 0, 0, 0]]

This is because, after two generation, the grid changes to Figure 2(c). By using the output format (0 /1), as illustrated before, we get the above output.

***Notice that: The output is a list, elements are lists as well !***

To make sure you have understood the rule, another running process is shown in below, you can use this example to test your program.



Hint: In order to put a list as an element into another list, you can use ***list.append()*** function.