The Game of Life is a well-known mathematical game that gives rise to amazingly complex behavior, although it can be specified by a few simple rules. (It is not actually a game in the traditional sense, with players competing for a win.) Here are the rules. The game is played on a rectangular grid. Each square on the grid can be either empty (represented by a 0 in our version) or occupied by a "living" cell (represented by a 1). At the beginning, you can specify empty and occupied cells in some way; then the game runs automatically. In each *generation*, the next generation is computed. A new cell is born on an empty square if it is surrounded by exactly three occupied neighbor cells. A cell dies of overcrowding if it is surrounded by four or more living neighbors, and it dies of loneliness if it is surrounded by zero or one living neighbor. A neighbor is an occupant of an adjacent square to the left, right, top, or bottom or in a diagonal direction. Figure 16 shows a cell and its neighbor cells.

Many configurations show interesting behavior when subjected to these rules. Figure 17 shows a *glider*, observed over five generations. Note how it moves. After four generations, it is transformed into the identical shape, but located one square to the right and below.

One of the more amazing configurations is the glider gun: a complex collection of cells that, after 30 moves, turns back into itself and a glider (see the large figure at the end).

Over the next week, you will use Python to program the game to eliminate the drudgery of computing successive generations by hand. Your homework assignment will come in two parts, also known as Programming Assignment 6 and Programming Assignment 7. The Wikipedia page on **The Game of Life** has a lot of useful information, as well as some real-time examples of program behavior. You should take a look at it.

Implementing this game requires a plan for what to do about the cells at the edges of the grid, since those cells don't have a full complement of eight neighbors. The Wikipedia entry does bring up this issue, and options range from simple to complex. For our purposes, it is sufficient to assume that every square outside of the grid is empty (i.e., 0) and will remain empty for the duration of the game.

(Much of the description above, and all the figures below, is borrowed from a textbook by Cay Horstmann.)

Programming Assignment 6 - Game of Life, Part 1

Your solution is to be written using Python 3. Make sure you provide comments including the file name, your name, and the date at the top of the file you submit. Also make sure to include appropriate docstrings for all functions.

The names of your functions must exactly match the names given in this assignment. The order of the parameters in your parameter list must exactly match the order given in this assignment.

For any given problem below, you may want to write additional functions other than those specified for your solution. That's fine with us.

The core problem to be solved in the implementation of the Game of Life is how to generate the next grid from the current grid. Your task is to write a function called `nextGen` which expects only one argument. That argument is a two-dimensional table (i.e., a list of lists) with m rows and n columns, representing the current grid. The elements of the table are either 0 (empty square) or 1 (occupied square). You may assume that all rows have the same number of elements.

Given the current grid, `nextGen` computes and returns (but does not print) a new next grid (without altering the current grid) by applying the simple rules provided above. For example, given this initial grid:

```
glider =   [[0,0,0,0,0,0,0],
            [0,0,1,0,0,0,0],
            [0,0,0,1,0,0,0],
            [0,1,1,1,0,0,0],
            [0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0]]
```

your function should work like this:

```
>>> x = nextGen(glider)
>>> x
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 1, 0, 0, 0], [0, 0, 1, 1, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
>>> y = nextGen(x)
>>> y
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0], [0, 1, 0, 1, 0, 0, 0],
[0, 0, 1, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
>>> z = nextGen(y)
>>> z
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 1, 0, 0],
[0, 0, 1, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
>>> q = nextGen(z)
>>> q
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0],
[0, 0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
```

If we formatted those results nicely, we'd see a sequence like this:

```
## generation 1

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

## generation 2

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

## generation 3

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

## generation 4

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

These grids correspond to the generations in Figure 17. Note that the grid passed to your function must remain unaltered:

```
>>> glider
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0], [0, 1, 1, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
```

Where to do the assignment

You can do this assignment on your own computer, or in the labs. In either case, use the IDLE development environment -- that's what we'll use when we grade your program. Put all the functions you created in a file called "prog67.py". Save that file until you've completed Programming Assignment 7.

Submitting the Assignment

Do not turn in this assignment by itself. Submit it with the work you do for Programming Assignment 7.

Saving your work

If you are working in the lab, you will need to copy your program to your own flash-drive. To save it on flash-drive, plug the flash-drive into the computer (your TA or the staff in the labs can help you figure out how), open the flash-drive, and copy your work to it by moving the folder with your files from the Desktop onto the flash-drive.

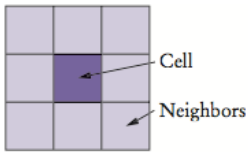


Figure 16
Neighborhood of a Cell

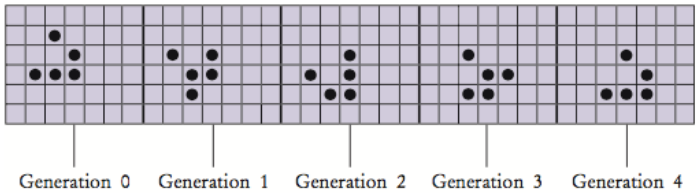
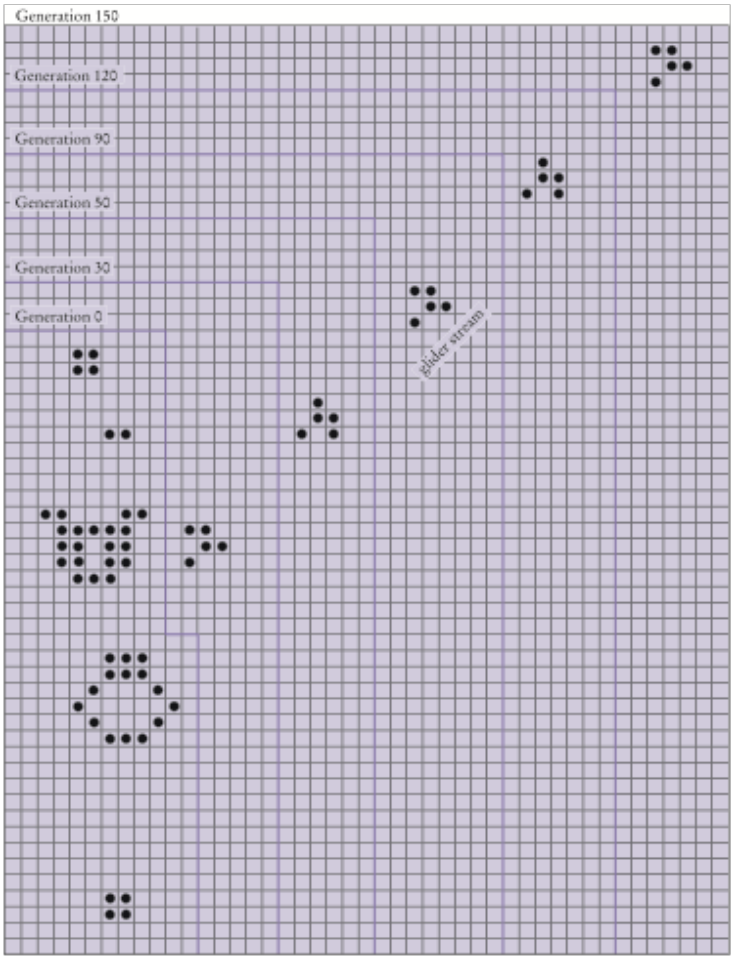


Figure 17
Glider


Generation 150
Generation 120
Generation 90
Generation 50
Generation 30
Generation 0
glider stream