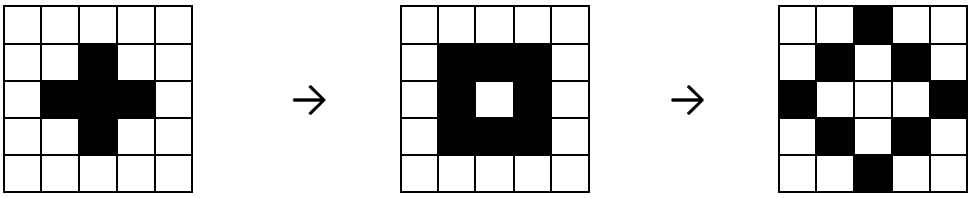
Conway's Game of Life

The Rules of the Game

1. Start with a square grid of "cells", each of which is "alive" (black) or "dead" (white).
2. At each step, we count the number of each cell's 8 immediate neighbours that are alive, and
    1. any live cell with fewer than 2 live neighbours dies
    2. any live cell with more than 3 live neighbours dies
    3. any live cell with 2 or 3 live neighbours lives on
    4. any dead cell with exactly three live neighbours becomes alive, and otherwise stays dead

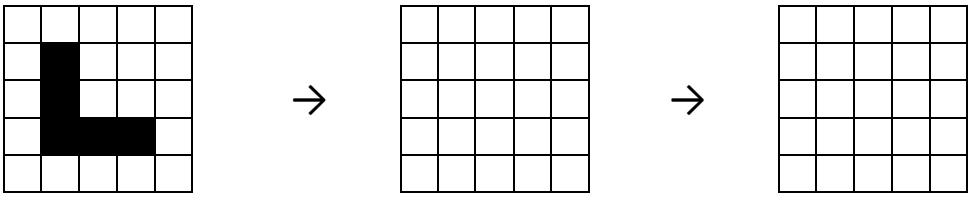
Example



Check that you understand how to apply the rules to the first pattern to reach the second and third patterns.

QUESTION 1

Fill in the patterns in the grids below.



Game of Life in Python

As you may have found, it is difficult to simulate the Game of Life using pen and paper. Fortunately, it is easily simulated on a computer.

The numbers in the square brackets near the top of the code correspond to the starting pattern for the simulation (1 = black, 0 = white):

```
x_init = [[0, 0, 0, 0, 0],
          [1, 0, 1, 1, 1],
          [0, 0, 0, 1, 1],
          [1, 1, 1, 1, 1],
          [0, 0, 0, 0, 0]]
```

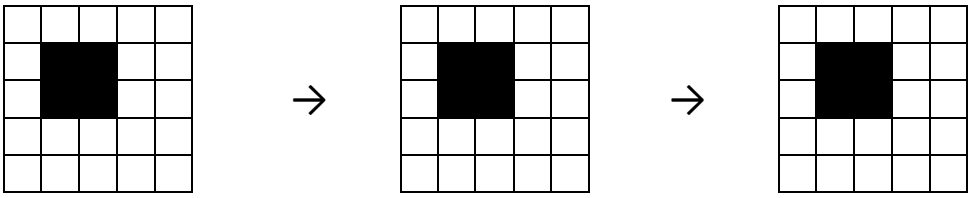
QUESTION 2

Run the simulation.

By changing the numbers in the square brackets, check your answers to Question 1.

Still Life

A “still life” is a pattern which doesn’t change from step to step.

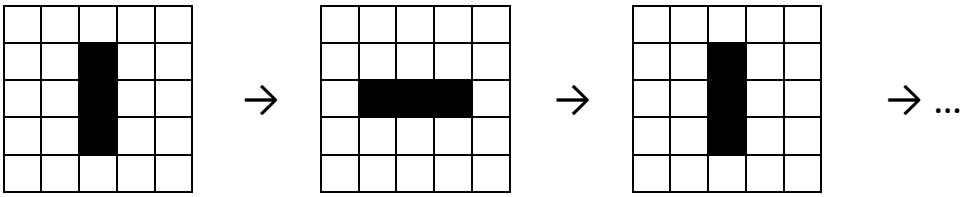


QUESTION 3

How many other still life patterns can you find?

Oscillator

An “oscillator” is a pattern which repeats after a fixed number of steps.

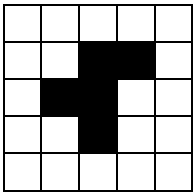


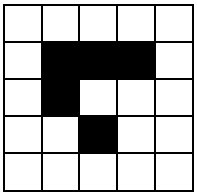
QUESTION 4

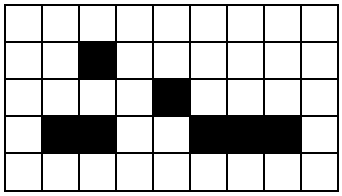
Can you find any other oscillator patterns?

Life

Why is it called ‘Game of Life’? Because there are some very simple starting patterns which lead to complicated behaviour reminiscent of living organisms.







QUESTION 5

Which of the above starting patterns result in 'life' ?

Which of the above patterns is a 'glider' ?

TIP: you can make the game board bigger by changing the value of d.

```
x_init = [[0, 0, 0, 0, 0],
          [1, 0, 1, 1, 1],
          [0, 0, 0, 1, 1],
          [1, 1, 1, 1, 1],
          [0, 0, 0, 0, 0]]

d = 30 ←
```

Experiment

QUESTION 6

Experiment with different starting patterns.

- Can you make a pattern that grows forever?

- What happens if you create several gliders pointing towards each other?

- What other interesting effects can you create…?

Compare you answers with your colleagues!

TIP: Increase the speed of the game by decreasing the value of `interval` (near the bottom of the notebook):

```
anim = animation.FuncAnimation(fig, updatefig, frames=N,
                               interval=1000, blit=True)
```

Challenge: Changing the Rules

There are many ways we can adapt the Game of Life. The rules of the game are encoded in the following line of code:

```
w = np.logical_or(np.logical_and(u == 1, np.logical_or(z == 2, z == 3)),
    np.logical_and(u == 0, z == 3))
```

Can you see how the game rules are encoded? **Hint**: `u` = central cell, and `z` = number of live surrounding cells.

QUESTION 7

Change the game rules so that a dead cell becomes alive if it has exactly 2 live neighbours (instead of 3). How does it change the result of the game?

Experiment with other rules!

Taking it Further

Game of Life Video

The Game of Life Wiki

http://jakevdp.github.io/blog/2013/08/07/conways-game-of-life/

Some more patterns to try:

```
diehard = [[0, 0, 0, 0, 0, 0, 1, 0],
           [1, 1, 0, 0, 0, 0, 0, 0],
           [0, 1, 0, 0, 0, 1, 1, 1]]

boat = [[1, 1, 0],
        [1, 0, 1],
        [0, 1, 0]]

r_pentomino = [[0, 1, 1],
               [1, 1, 0],
               [0, 1, 0]]

beacon = [[0, 0, 1, 1],
          [0, 0, 1, 1],
          [1, 1, 0, 0],
          [1, 1, 0, 0]]

acorn = [[0, 1, 0, 0, 0, 0, 0],
         [0, 0, 0, 1, 0, 0, 0],
         [1, 1, 0, 0, 1, 1, 1]]

spaceship = [[0, 0, 1, 1, 0],
             [1, 1, 0, 1, 1],
             [1, 1, 1, 1, 0],
             [0, 1, 1, 0, 0]]

block_switch_engine = [[0, 0, 0, 0, 0, 0, 1, 0],
                       [0, 0, 0, 0, 1, 0, 1, 1],
                       [0, 0, 0, 0, 1, 0, 1, 0],
                       [0, 0, 0, 0, 1, 0, 0, 0],
                       [0, 0, 1, 0, 0, 0, 0, 0],
                       [1, 0, 1, 0, 0, 0, 0, 0]]
```