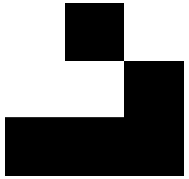


Life of Py: Conway’s Game of Life

Explanation of the assignment

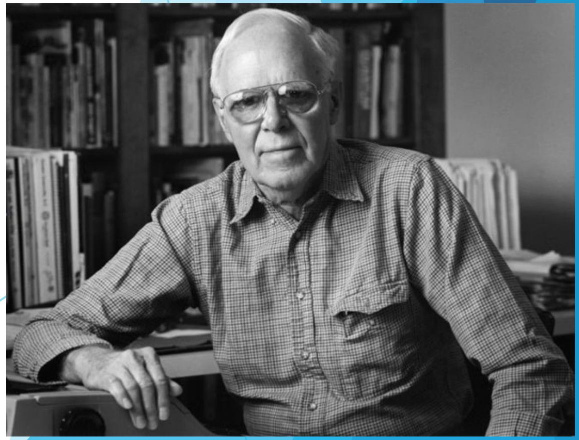
Recreational Mathematics

"I just play all the time and am fortunate enough to get paid for it."
– Martin Gardner, 1998

- Scientific American ran a monthly column by **Martin Gardner** from 1957 – 1980 (24 years!), called **"Mathematical Games"** (was then succeeded by Douglas Hofstadter with "Metamagical themes")
- Martin Gardners columns became legendary and have inspired millions (and continue to do so today.) Many mathematicians, programmers, scientists and engineers.
- In **October 1970** he published "The fantastic combinations of John Conway's new solitaire game 'life'", containing the following quote:

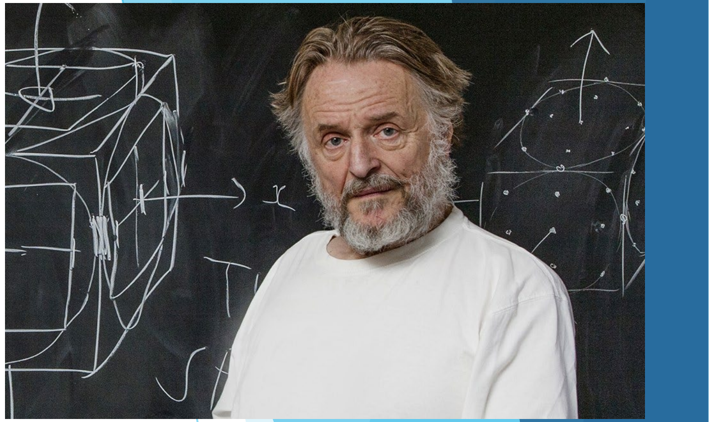
"Because of its analogies with the rise, fall and alterations of a society of living organisms, it belongs to a growing class of what are called 'simulation games' - games that resemble real-life processes.

To play Life without a computer you need a fairly large checkerboard and a plentiful supply of flat counters of two colors."



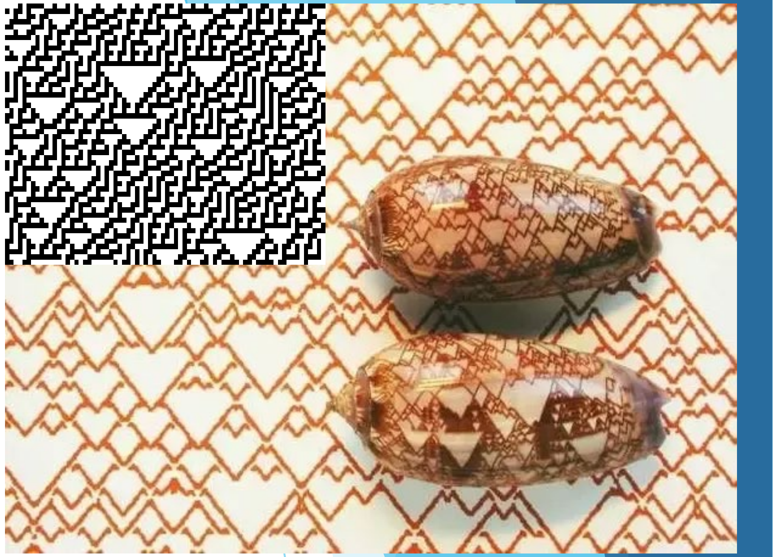
Martin Gardner (1914-2010)

John Horton Conway (born 1937)



- Studied Mathematics in Cambridge (UK), while being a fanatic backgammon player later worked at Princeton.
- Active in number theory, game theory, the theory of finite groups, knot theory and cryptography
- Responsible for the most widely read column by Martin Gardner on his Game of Life
- But there is more: e.g. Game of Sprouts (featured in Gardner column of 1970)

Life, Game of Life



- Not really a game, more a simulation of life or the universe. Life is an example of what is called **cellular automaton** or CA.
- Concept discovered by Stanislaw Ulam and John von Neumann in 1940. Other **cellular automata**, are in 1D: Rule 30, Rule 110, Rule 250 and in 2D: Wireworld**.**
- Stephen Wolfram has made systematic study of cellular automata
- Can be used as a model to study physics and philosophical concepts (metaphysical) like symmetry, algorithmic compressibility, but also ecology or biology as gene expression, growth patterns and populations follow similar rules
- Conway’s game of Life is **Turing-complete** (as is Wirewold)

Life or Game of Life: Concept

- Take a large board in which each field has two states: dead (empty) or alive (filled)

- To calculate the next state (***generation***), of the board for each field, sum the number of cells in the neighbouring eight cells

- Using this total, apply this rule to calculate the next state of the cell:
  - **0,1** : death
  - **2** : same as current state
  - **3** : birth, will be filled (alive) in next generation
  - **4 or more** : death

Game of Life example

		1		
		1		
		1		

0	0	0	0	0
0	1	1	1	0
0	2	1	2	0
0	3	2	3	0
0	2	1	2	0
0	1	1	1	0
0	0	0	0	0

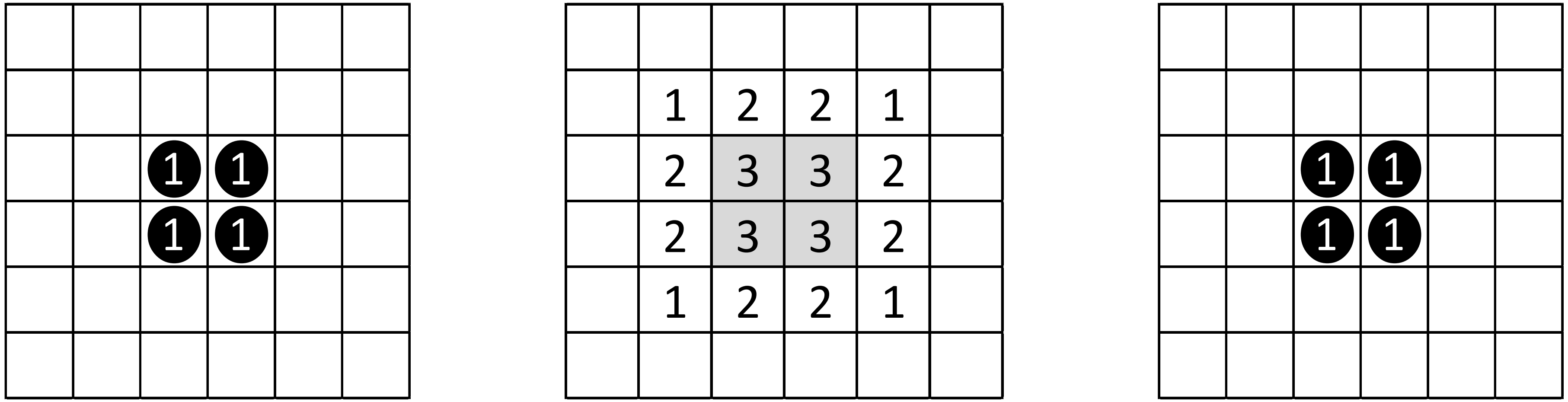
	1	1	1	

0	0	0	0	0
0	1	1	1	0
0	2	3	2	0
0	1	2	1	0
0	2	3	2	0
0	1	1	1	0
0	0	0	0	0

		1		
		1		
		1		

		■		
		■		
		■		

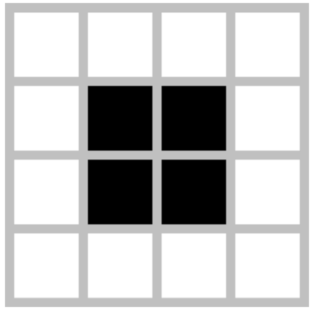
Stable pattern: block



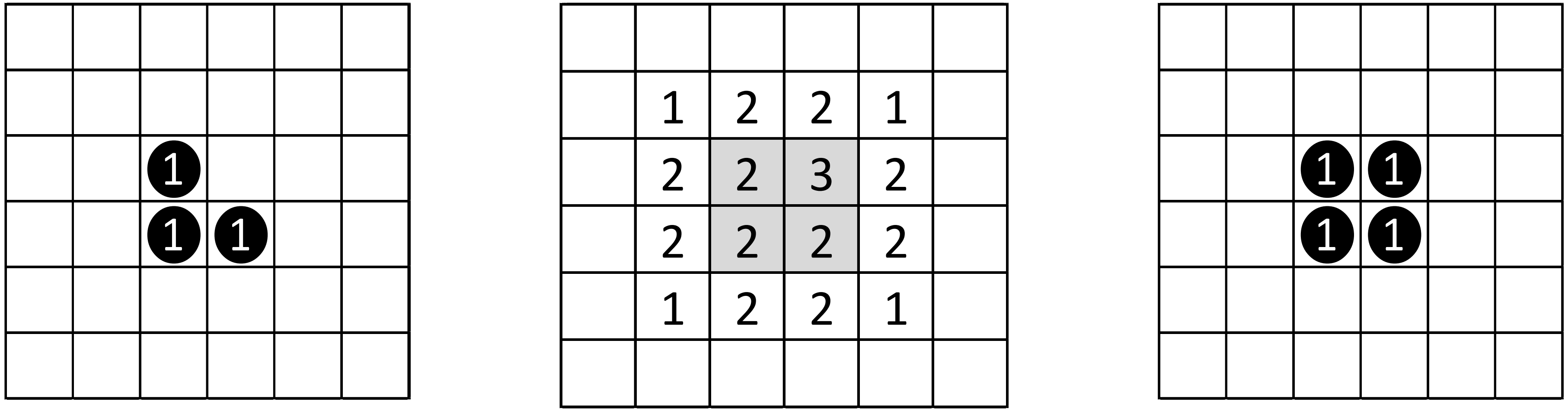
		❶	❶		
		❶	❶		

	1	2	2	1	
	2	3	3	2	
	2	3	3	2	
	1	2	2	1	

		❶	❶		
		❶	❶		



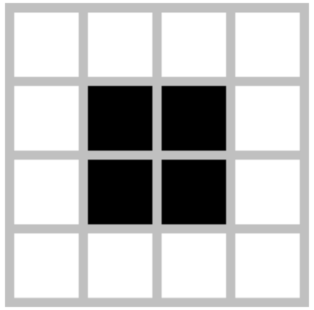
Stable pattern: block



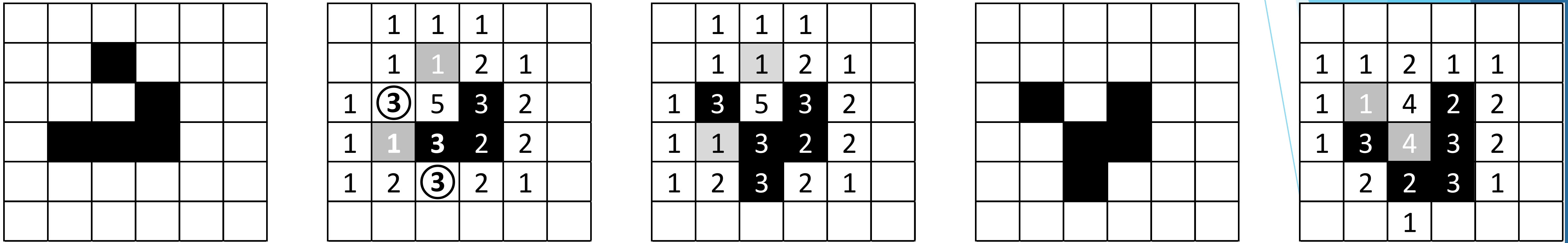
		1			
		1	1		

	1	2	2	1	
	2	2	3	2	
	2	2	2	2	
	1	2	2	1	

		1	1		
		1	1		



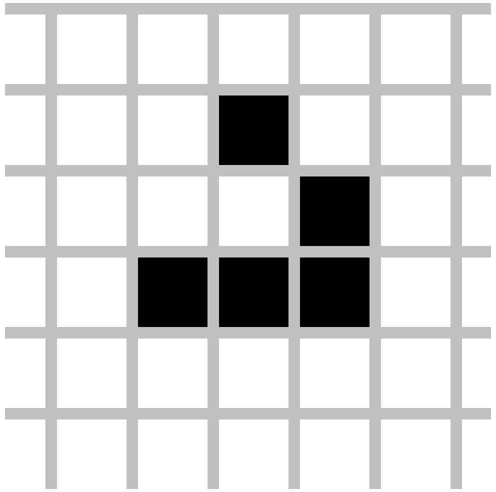
Glider/Floater



	1	1	1		
	1	1	2	1	
1	③	5	3	2	
1	1	3	2	2	
1	2	③	2	1	

	1	1	1		
	1	1	2	1	
1	3	5	3	2	
1	1	3	2	2	
1	2	3	2	1	

1	1	2	1	1	
1	1	4	2	2	
1	3	4	3	2	
	2	2	3	1	
		1			



Programming Life of Py

- Use numpy arrays (rows,columns), so board[y,x] is state of cell
- Use floats/integers: 1 = filled (alive) 0 = empty(dead)

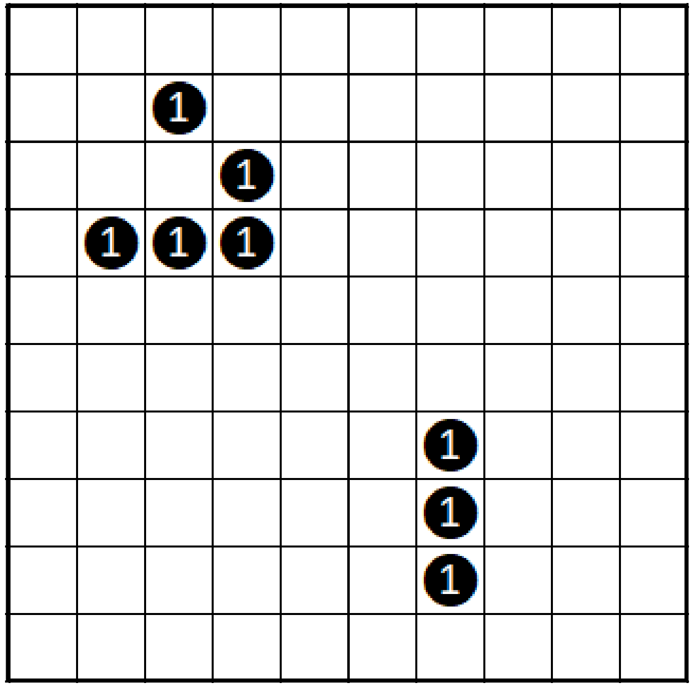
- Make a function in a separate script with two functions:
  - Create a window of a certain size
  - Draw the current board

- Fill with example patterns and/or random fill

- You can start the “naïve” way: Making two for-loops (one for rows and one for columns), then for each cell sum the surrounding eight cells

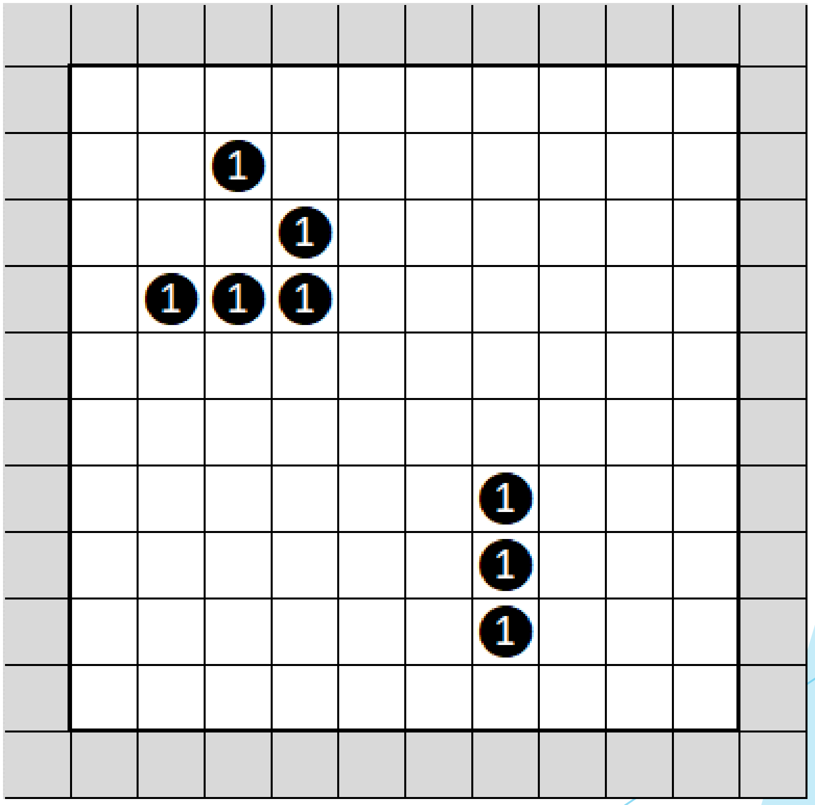
The power of numpy: vectorisation

ny x nx



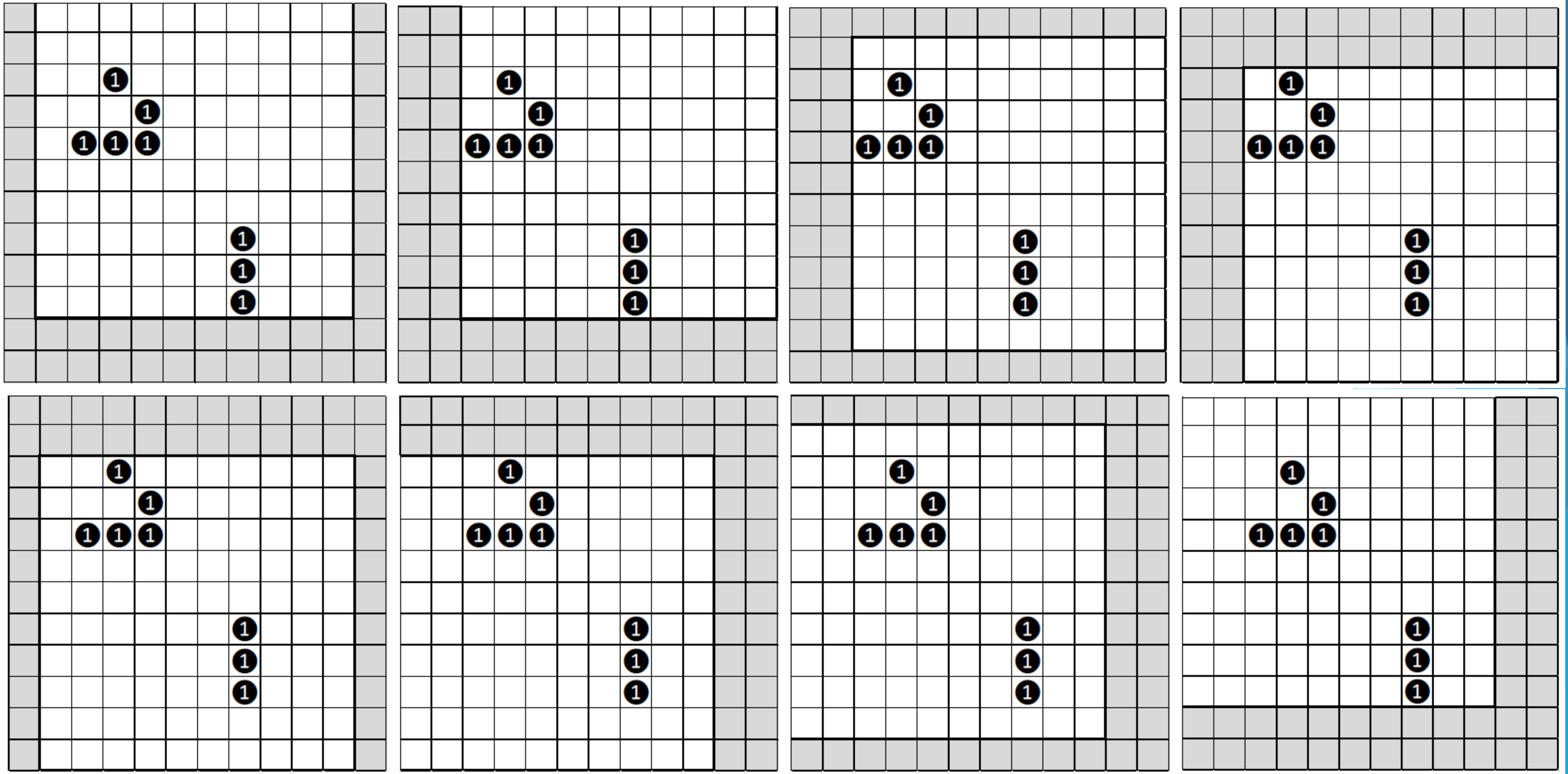
		1							
			1						
	1	1	1						
						1			
						1			
						1			

ny+2 x nx+2

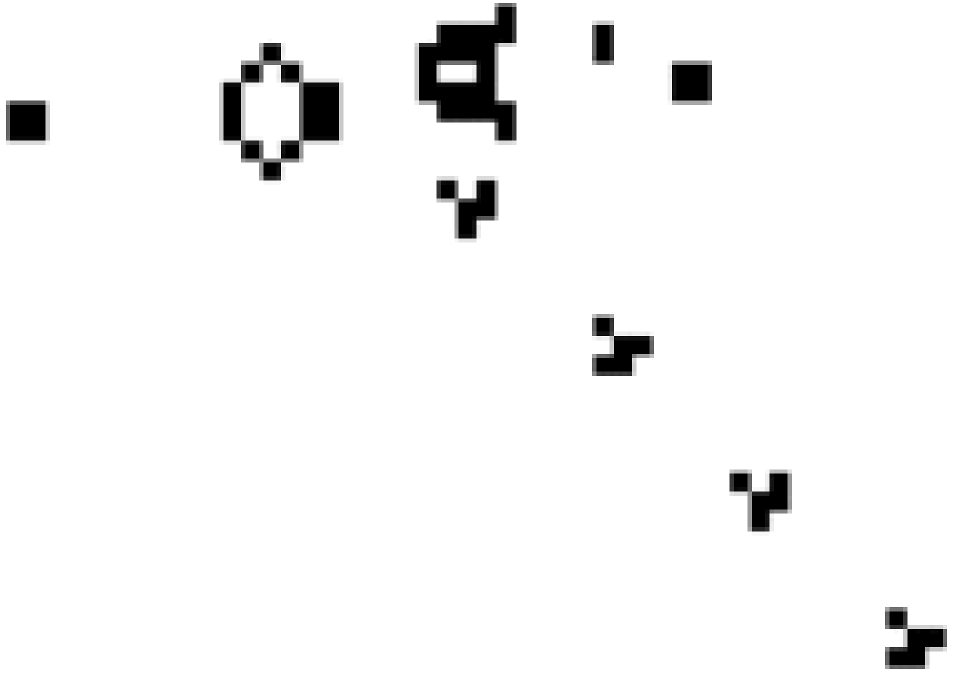


			1								
				1							
		1	1	1							
						1					
						1					
						1					

The power of numpy: vectorisation

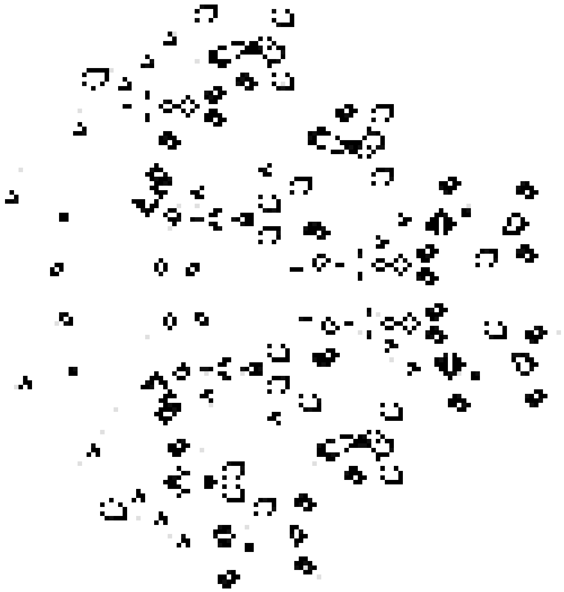


Many patterns have been discovered



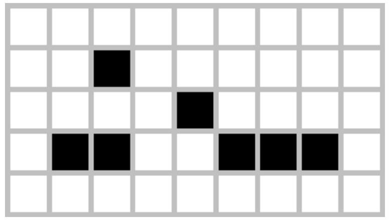
Glider Gun

Breeder

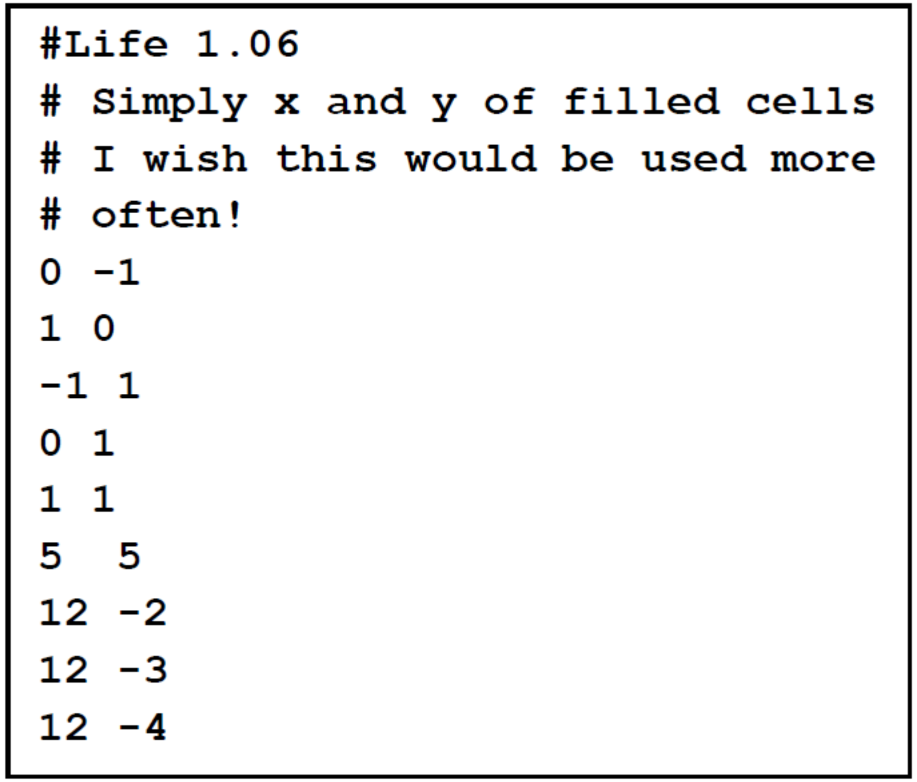


Acorn

- Becomes much bigger than you would expect



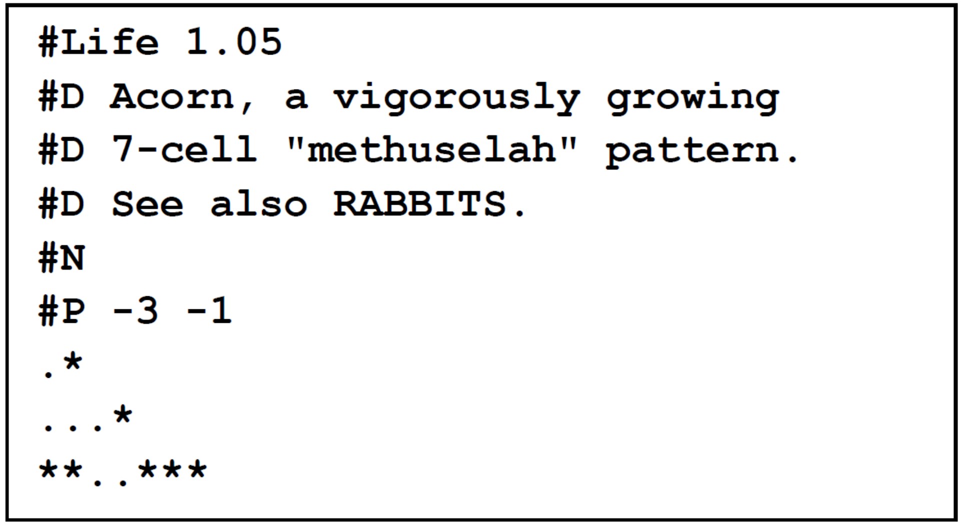
Load pattern files from internet:
(lines mentioning format often not presented)



```
#Life 1.06
# Simply x and y of filled cells
# I wish this would be used more
# often!
0 -1
1 0
-1 1
0 1
1 1
5  5
12 -2
12 -3
12 -4
```

- Your program has to be able to read this simple format
- There is a set available from BrightSpace (assignment section)
- However, on internet you will rarely find this as it is hard to read for humans. Most are RLE or Life 1.05! These are more easily edited manually.
- On BrightSpace and at the links, you will find a large collection in 1.05 and RLE format

Load pattern files from internet:
(lines mentioning format often not presented)

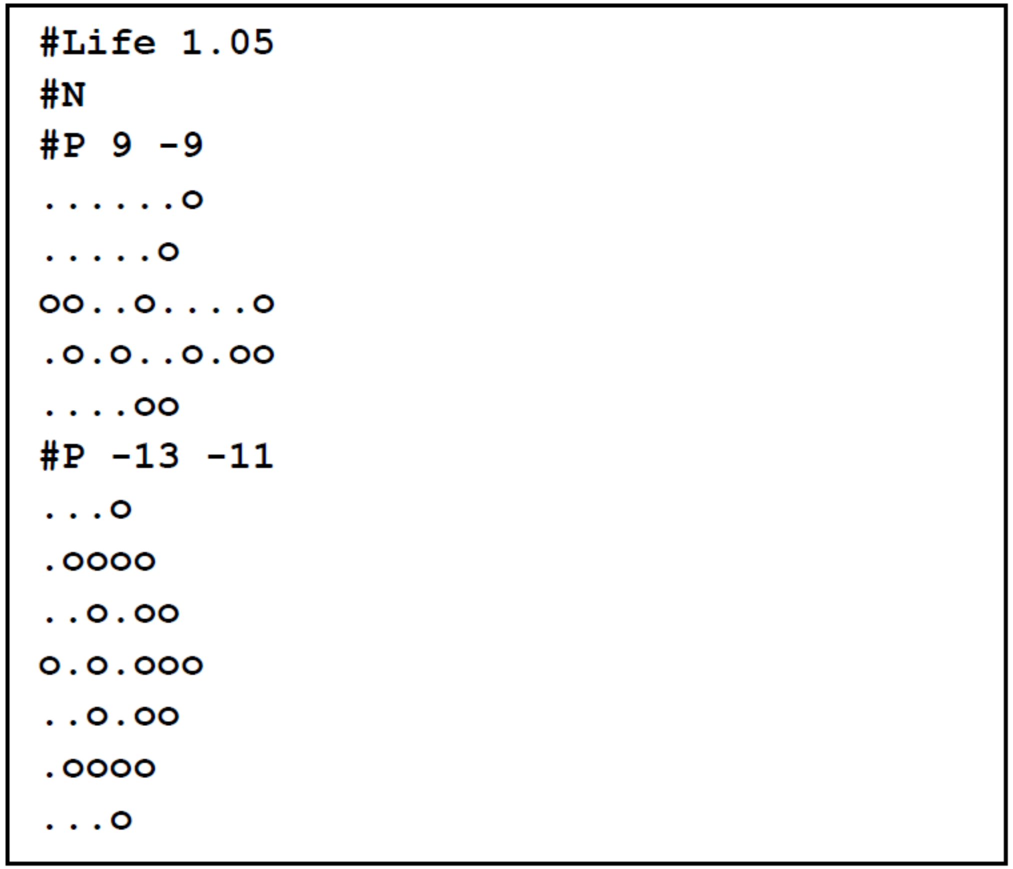


```
#Life 1.05
#D Acorn, a vigorously growing
#D 7-cell "methuselah" pattern.
#D See also RABBITS.
#N
#P -3 -1
.*
...*
**..***
```

#P x y means position: is a shift, gives you origin (xorig,yorig) of the pattern on the next lines

Text “Life 1.05” is often not there!

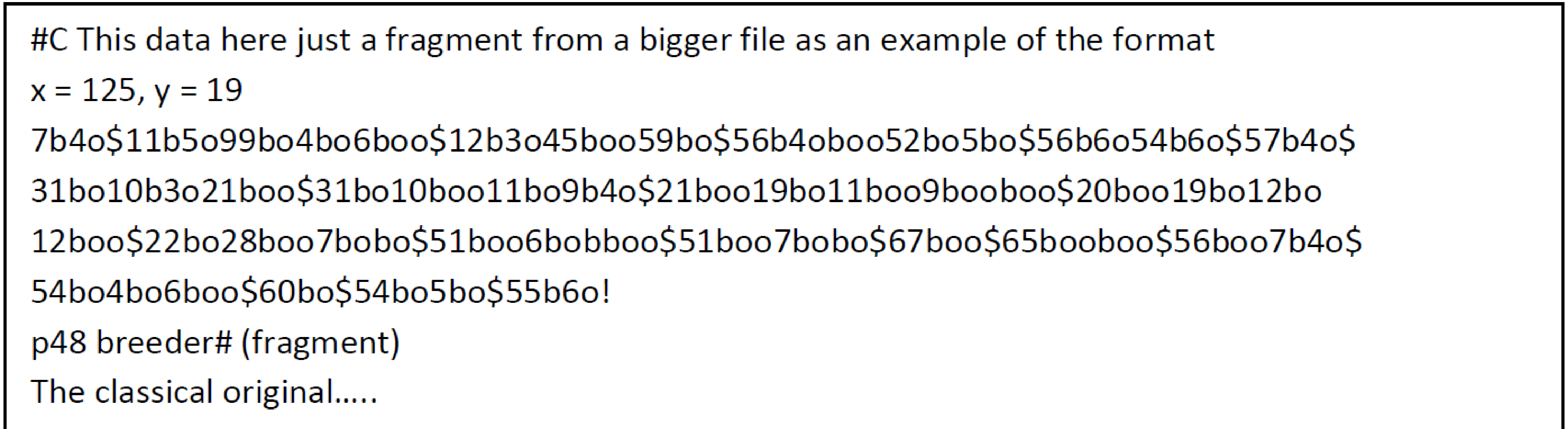
How do you recognize formats?



```
#Life 1.05
#N
#P 9 -9
......o
.....o
oo..o....o
.o.o..o.oo
....oo
#P -13 -11
...o
.oooo
..o.oo
o.o.ooo
..o.oo
.oooo
...o
```

Load pattern files from internet: RLE run length encoded

- Code: x=125, y = 19 => size, can be ignored!
- b = blank (empty) o = cell (filled)
  $ = new row (new line in file means nothing!) ! Means end of data
  #C = comment
- obbboo$bo = one cell, 3 blanks, 2 cells, then on next row: blank and cell
- 16b7obo$ = 16 blanks, 7 cells, 1 blank and an o



```
#C This data here just a fragment from a bigger file as an example of the format
x = 125, y = 19
7b4o$11b5o99bo4bo6boo$12b3o45boo59bo$56b4oboo52bo5bo$56b6o54b6o$57b4o$
31bo10b3o21boo$31bo10boo11bo9b4o$21boo19bo11boo9booboo$20boo19bo12bo
12boo$22bo28boo7bobo$51boo6bobboo$51boo7bobo$67boo$65booboo$56boo7b4o$
54bo4bo6boo$60bo$54bo5bo$55b6o!
p48 breeder# (fragment)
The classical original.....
```

Pro tip: Recognizing a format using sets

(saves a lot of programming)

- Strip() line (without arguments it removes leading & trailing spaces and newline)
- Check if line is data line (does it start with # or not?)
- Convert lines with list() to a list of chars, then convert with set() to a set!
- From this set subtract this set from a few sets for each format:

```
set(list(line.strip())) - charsetlife106
```

Use acorn.lif
as test case!
(see BrightSpace)

- When length of above set is zero, all chars in line fit in set, increase linescounter for this format
- Count number of lines which comply with characters in a data line for each format (nline105, nline106, nlinerle)

Pro tip: Recognizing a format using sets

(saves a lot of programming)

- Strip() line (without arguments it removes leading & trailing spaces and newline)
- Check if line is data line (does it start with # or not?)
- Convert lines with list() to a list of chars, then convert with set() to a set!
- From this set subtract this set from a few sets for each format:

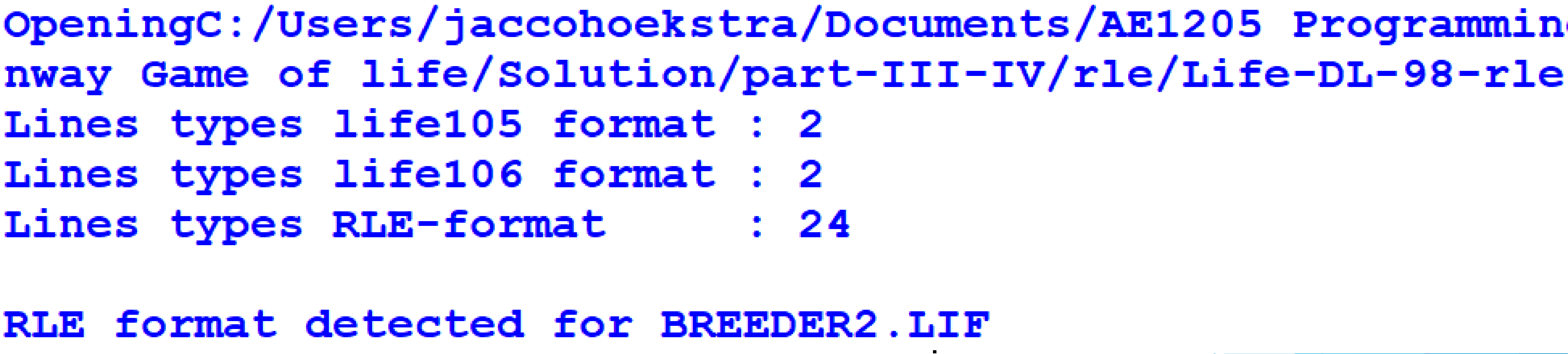
Use acorn.lif
as test case!
(see BrightSpace)

```
set(list("-20 , 3")) - set(["-",","," ","0","1",
"2","3","4","5","6","7","8","9"])
```

- When length of above set is zero, all chars in line fit in set, increase linescounter for this format

- Count number of lines which comply with characters in a data line for each format (nline105, nline106, nlinerle)

Example of output: detecting format



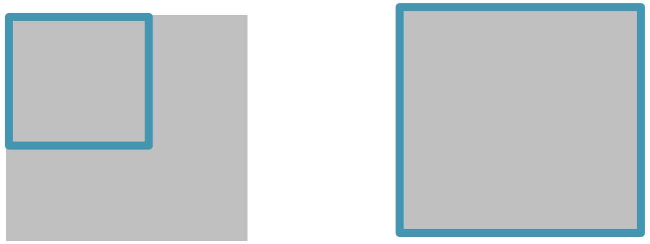
```
OpeningC:/Users/jaccohoekstra/Documents/AE1205 Programmin
nway Game of life/Solution/part-III-IV/rle/Life-DL-98-rle
Lines types life105 format : 2
Lines types life106 format : 2
Lines types RLE-format     : 24

RLE format detected for BREEDER2.LIF
```

For assignment:
At least reading of Life 1.06 required

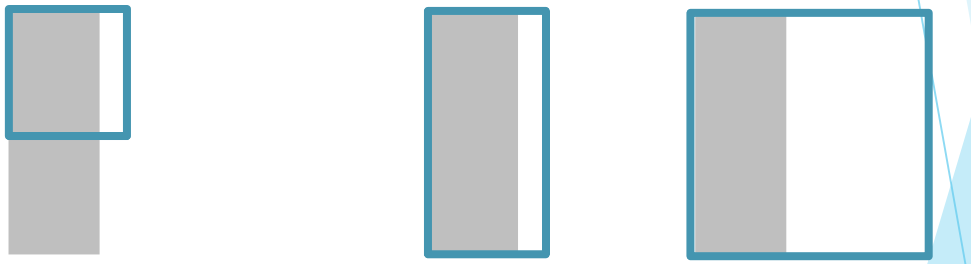
- Download all files from BrightSpace
- Implement at least function which reads Life 1.06
- But this is very rare format, hardly found anywhere
- So check all data lines for format that they could be, count them
- Assume the largest number of lines fitting will be the format

Placing loaded patterns on the board

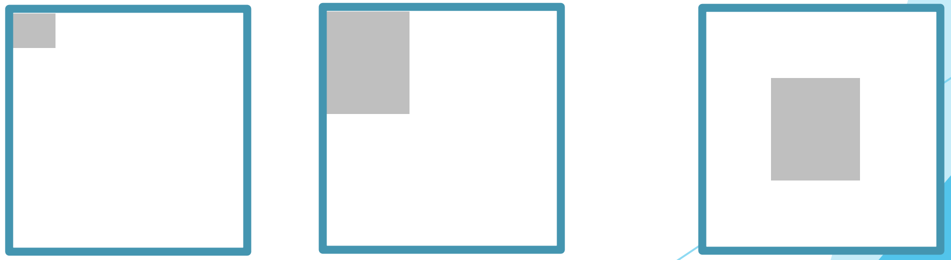
- When it is too large, increase size of board



- ***Adapt aspect ratio if necessary



- Smaller patterns: center on board



- ***Much smaller patterns: scale up board first

File open dialog box

- Download function from BrightSpace: it is in filedialogs.py
- Use this to retrieve file name for a user to open file
- Detect file type by analyzing the lines
- Return pattern
- In main program place on board and start simulation

Add user choice for random fill or loading a file

Required:

- Before start: option to choose a random fill (percentage and board size)

Optional:

- Add generation text and cell count

- Even make an editor? And Save function? (See filedialogs)

Life of Py: Required & Optional

- Required: Vectorizes version of Conway's Game of Life sing numpy arrays
- Required: Be able to open and convert life 1.06 files
- Required: Menu option random fill (percentage and board size)
- Optional: Recognize all files type based on number of data lines (use sets!)
- Optional: GUI options (generation number, cell count) Editor would be a lot of work.