Domáca úloha 1

O game of life som už počul, doteraz som ale nechápal ako je to spravené.
Keď som bol malý a začínal s PC tak popri Powder Toy bolo Conway's game of life – prišlo mi to ako zázrak, ako môže niekto simulovať život? Ale to bola naivná predstava 10 ročného ja.

Začal som s počiatočným kódom od LLM, pretože python som už dlho nepísal:

```
def step(row):
    new_row = [0] * len(row)  # Create a new row of the same size, initialized with 0s

    for i in range(len(row)):
        left = row[i - 1] if i > 0 else 0  # Left neighbor (0 if at the beginning)
        right = row[i + 1] if i < len(row) - 1 else 0  # Right neighbor (0 if at the end)

        # Rule: If a cell has exactly 1 neighbor alive, it becomes alive
        if row[i] == 1 and (left == 1 or right == 1):
            new_row[i] = 1
        elif row[i] == 0 and (left + right == 1):
            new_row[i] = 1

    return new_row

def printrow(row):
    output = ''.join(['█' if cell == 1 else ' ' for cell in row])  # Use █ for alive, space for dead
    print(output)

row = [0, 1, 0, 1, 1, 0, 0, 1]
printrow(row)  # It will print a row of spaces and █ characters
```

Spustil som program a dostal som tento výstup:
`█ ███ ` `█` - toto sa mi nepáčilo, keďže mi to príde trocha neintuitívne, tak som to prepísal:

```
def step(row):
    new_row = [0] * len(row)  # Create a new row of the same size, initialized with 0s

    for i in range(len(row)):
        left = row[i - 1] if i > 0 else 0  # Left neighbor (0 if at the beginning)
        right = row[i + 1] if i < len(row) - 1 else 0  # Right neighbor (0 if at the end)
```

```
        # Rule: If a cell has exactly 1 neighbor alive, it becomes alive
        if row[i] == 1 and (left == 1 or right == 1):
            new_row[i] = 1
        elif row[i] == 0 and (left + right == 1):
            new_row[i] = 1

    return new_row
```

Následne som si preštudoval syntax, a uvedomil že definícia left a right je vlastne skvelá, pretože nebudem musieť robiť mentálnu aritmetiku na poliach.

Pridal som pravidlo – ak má bunka dvoch živých susdeov tak zomrie (teraz už ja, nie LLM :d).

```
        # pravidlo: ak ma bunka dvoch zivych susedov, zomrie
        if row[i] == 1 and (left == 1 and right == 1):
            new_row[i] = 0
```

Následne som napísal prvotnú podobu funkcie printRow, ktorá vypísala živé bunky ako ‚Y' a mŕtve ako ‚x'.

```
def printrow(row):
output = str(genNumber) + ' ' + ''.join(['Y' if cell == 1 else 'x' for cell in row])
row = [1, 0, 0, 1, 1, 0, 0, 0]

printrow(step(row))  # spusti algoritmus
```

Potom som pridal iterovanie voči generáciám.

```
def printrow(row):
    global generations
    global genNumber

    for _ in range(generations):  # opakuj dany pocet generacii
        output = str(genNumber) + ' ' + ''.join(['Y' if cell == 1 else 'x' for cell in row])
        print(output)

        row = step(row)  # vygeneruj dalsiu generaciu
        genNumber += 1  # zvys pocet generacii

printrow(row)  # spusti algoritmus
```

Zároveň aj premennú genNumber aby som mal jednoducho a čítateľne prehľad o tom, ktorý riadok je ktorá generácia.

Príklad vstup/výstup:

Konkrétne sa jednalo o 5 generácií:

Input

generations = 5 # pocet generacii na vykonanie
row = [1, 0, 0, 1, 1, 0, 0, 0] # vstupny array prvkov

Output
1 YxxYYxxx 2 xYYYYYxx 3 YYxxxYYx 4 YYYxYYYY 5 YxYxYxxY

Je tam veľmi pekne vidieť, že ak sú tri ‚Y' vedľa seba, tak zanikne presne člen v strede, zároveň špeciálny prípad 5x Y vedľa seba, stredné členy vymreli.

Input

generations = 5 # pocet generacii na vykonanie
row = [0, 0, 0, 0, 0, 0, 0, 0] # vstupny array prvkov

Output
1 xxxxxxxx 2 xxxxxxxx 3 xxxxxxxx 4 xxxxxxxx 5 xxxxxxxx
Input

generations = 5 # pocet generacii na vykonanie
row = [1, 1, 1, 1, 1, 1, 1, 1] # vstupny array prvkov

Output
1 YYYYYYYY 2 YxxxxxxY 3 xYxxxxYx 4 YxYxxYxY 5 xxxYYxxx