13. Dvojrozmerné tabuľky

video prezentácia

dvojrozmerné tabuľky

Pythonovský zoznam `list` (blízky jednorozmerným poliam v iných programovacích jazykoch) slúži hlavne na uchovávanie nejakej postupnosti alebo skupiny údajov. V takejto štruktúre sa dajú uchovávať aj dvojrozmerné tabuľky ako zoznam zoznamov (opäť je to analógia k dvojrozmerným poliam). Dvojrozmerné údaje sa často vyskytujú, napríklad ako rôzne hracie plochy (štvorčekový papier pre piškvorky, šachovnica pre doskové hry, rôzne typy labyrintov), ale napríklad aj rastrové obrázky sú často uchovávané v dvojrozmernej tabuľke. Aj v matematike sa niekedy pracuje s dvojrozmernými tabuľkami čísel (tzv. matice).

Už vieme, že prvkami zoznamu môžu byť opäť postupnosti (zoznamy alebo n-tice). Práve táto vlastnosť nám poslúži pri reprezentácii dvojrozmerných tabuliek. Napríklad, takúto tabuľku (matematickú maticu 3x3):

```
1 2 3
4 5 6
7 8 9
```

môžeme v Pythone zapísať:

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Zoznam `m` má tri prvky: sú to tri riadky (zoznamy čísel - teda jednorozmerné polia). Našou prvou úlohou bude vypísať takýto zoznam do riadkov. Ale takýto jednoduchý výpis sa nám nie vždy bude hodiť:

```
>>> for riadok in m:
        print(riadok)
    [1, 2, 3]
    [4, 5, 6]
    [7, 8, 9]
```

Častejšie to budeme robiť dvoma vnorenými cyklami. Zadefinujme funkciu `vypis()` s jedným parametrom dvojrozmernou tabuľkou:

```
def vypis(tab):
    for riadok in tab:
        for prvok in riadok:
            print(prvok, end=' ')
        print()
```

To isté vieme zapísať aj pomocou indexovania:

```
def vypis(tab):
    for i in range(len(tab)):
        for j in range(len(tab[i])):
            print(tab[i][j], end=' ')
        print()
```

Obe tieto funkcie sú veľmi častými šablónami pri práci s dvojrozmernými tabuľkami. Teraz výpis tabuľky vyzerá takto:

```
>>> vypis(m)
    1 2 3
    4 5 6
    7 8 9
```

Vytváranie dvojrozmerných tabuliek

Pozrime, ako môžeme vytvárať nové dvojrozmerné tabuľky. Okrem priameho priradenia, napríklad:

```
>>> matica = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

ich môžeme poskladať z jednotlivých riadkov, napríklad:

```
>>> riadok1 = [1, 2, 3]
>>> riadok2 = [4, 5, 6]
>>> riadok3 = [7, 8, 9]
>>> matica = [riadok1, riadok2, riadok3]
```

Častejšie to ale bude pomocou nejakých cyklov. Závisí to od toho, či sa vo výslednej tabuľke niečo opakuje. Vytvorme dvojrozmernú tabuľku veľkosti 3x3, ktorá obsahuje samé `0`:

```
>>> matica = []
>>> for i in range(3):
        matica.append([0, 0, 0])
>>> matica
    [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> vypis(matica)
    0 0 0
    0 0 0
    0 0 0
```

Využili sme tu štandardný spôsob vytvárania jednorozmerného zoznamu pomocou metódy `append()`. Obsah ľubovoľného prvku matice môžeme zmeniť obyčajným priradením:

```
>>> matica[0][1] = 9
>>> matica[1][2] += 1
>>> vypis(matica)
    0 9 0
    0 0 1
    0 0 0
```

Prvý index v `[]` zátvorkách väčšinou bude pre nás označovať poradové číslo riadka, v druhých zátvorkách je poradové číslo stĺpca. Už sme si zvykli, že riadky aj stĺpce sú číslované od 0.

Hoci pri definovaní matice sa zdá, že sa 3-krát opakuje to isté. Zapíšme to pomocou viacnásobného zreťazenia (operácia `*`) zoznamov:

```
>>> matica1 = [[0, 0, 0]] * 3
```

Opäť sa potvrdzuje, že je to veľmi nesprávny spôsob vytvárania prvkov zoznamu:
zápis `[0, 0, 0]` označuje **referenciu** na trojprvkový zoznam, potom `[[0, 0, 0]]*3` rozkopíruje túto jednu referenciu trikrát. Teda vytvorili sme zoznam, ktorý trikrát obsahuje referenciu na ten istý riadok.
Presvedčíme sa o tom priradením do niektorých prvkov takéhoto zoznamu:

```
>>> matica1[0][1] = 9
```

```
>>> matica1[1][2] += 1
>>> vypis(matica1)
    0 9 1
    0 9 1
    0 9 1
```

Uvedomte si, že zápis:

```
>>> matica1 = [[0, 0, 0]] * 3
```

v skutočnosti znamená:

```
>>> riadok = [0, 0, 0]
>>> matica1 = [riadok, riadok, riadok]
```

Zapamätajte si! Dvojrozmerné štruktúry **nikdy** nevytvárame tak, že viacnásobne zreťazujeme (násobíme) jeden riadok viackrát. Pritom:

```
>>> matica2 = [[0] * 3, [0] * 3, [0] * 3]
```

je už v poriadku, lebo v tomto zozname sme vytvorili tri rôzne riadky.

Niekedy sa na vytvorenie „prázdnej“ dvojrozmernej tabuľky definuje funkcia:

```
def vyrob(pocet_riadkov, pocet_stlpcov, hodnota=0):
    vysl = []
    for i in range(pocet_riadkov):
        vysl.append([hodnota] * pocet_stlpcov)
    return vysl
```

Otestujme:

```
>>> a = vyrob(3, 5)
>>> vypis(a)
    0 0 0 0 0
    0 0 0 0 0
    0 0 0 0 0
>>> b = vyrob(2, 6, '*')
>>> vypis(b)
    * * * * * *
    * * * * * *
```

Iný tvar zápisu tejto funkcie:

```
def vyrob(pocet_riadkov, pocet_stlpcov, hodnota=0):
    vysl = [None] * pocet_riadkov          # None alebo ľubovoľná iná hodnota
    for i in range(pocet_riadkov):
        vysl[i] = [hodnota] * pocet_stlpcov
    return vysl
```

Je na programátorovi, ktorú formu zápisu tejto funkcie použije.

Niekedy potrebujeme do takto pripravenej tabuľky priradiť nejaké hodnoty, napríklad postupným zvyšovaním nejakého počítadla:

```
def ocisluj(tab):
    poc = 0
    for i in range(len(tab)):
        for j in range(len(tab[i])):
```

```
            tab[i][j] = poc
            poc += 1
```

Všimnite si, že táto funkcia vychádza z druhej funkcie (šablóny) pre vypisovanie dvojrozmernej tabuľky: namiesto výpisu prvku (`print()`) sme do neho niečo priradili. Táto funkcia `ocisluj()` nič nevypisuje ani nevracia žiadnu hodnotu „len“ modifikuje obsah tabuľky, ktorá je parametrom tejto funkcie.

```
>>> a = vyrob(3, 5)
>>> ocisluj(a)
>>> vypis(a)
    0 1 2 3 4
    5 6 7 8 9
    10 11 12 13 14
```

Funkciu `ocisluj` môžeme zapísať aj takto jednoduchšie:

```
def ocisluj(tab):
    poc = 0
    for riadok in tab:
        for j in range(len(riadok)):
            riadok[j] = poc
            poc += 1
```

Zobrazenie obsahu dvojrozmernej tabuľky v grafickej ploche

Zapíšme takúto funkciu na výpis obsahu dvojrozmernej tabuľky:

```
import tkinter

def kresli_text(tab):
    d = 20
    for r, riadok in enumerate(tab):
        for s, prvok in enumerate(riadok):
            canvas.create_text(s*d + 10, r*d + 10, text=prvok)
```

Otestujeme:

```
canvas = tkinter.Canvas()
canvas.pack()

t = vyrob(7, 11)
ocisluj(t)
kresli_text(t)

tkinter.mainloop()
```

Dostávame takýto výpis:

```
 0  1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65
66 67 68 69 70 71 72 73 74 75 76
```

Zaujímavejší výstup v grafickej ploche dostaneme, keď namiesto čísel budeme vykresľovať farebné štvorčeky. Predpokladajme, že čísla v tabuľke sú len z nejakého malého intervalu, napríklad sú to čísla z intervalu `<0, 3>`, potom môžeme každú hodnotu v tabuľke zakresliť jednou zo štyroch farieb, napríklad 0 = `'white'`, 1 = `'black'`, 2 = `'red'`, 3 = `'blue'`. Pre istotu pri priradení farby pre každý štvorček vypočítame zvyšok po delení počtom farieb, teda číslom `4`:

```
import tkinter

def kresli(tab, d=20):
    farby = ('white', 'black', 'red', 'blue')
    for r, riadok in enumerate(tab):
        for s, prvok in enumerate(riadok):
            x, y = s*d + 5, r*d + 5
            farba = farby[prvok % len(farby)]
            canvas.create_rectangle(x, y, x+d, y+d,
                                    fill=farba, outline='light gray')
```

Keď teraz vykreslíme obsah tabuľky:

```
canvas = tkinter.Canvas()
canvas.pack()

t = vyrob(7, 11)
ocisluj(t)
kresli(t)

tkinter.mainloop()
```

Dostávame takýto výpis:

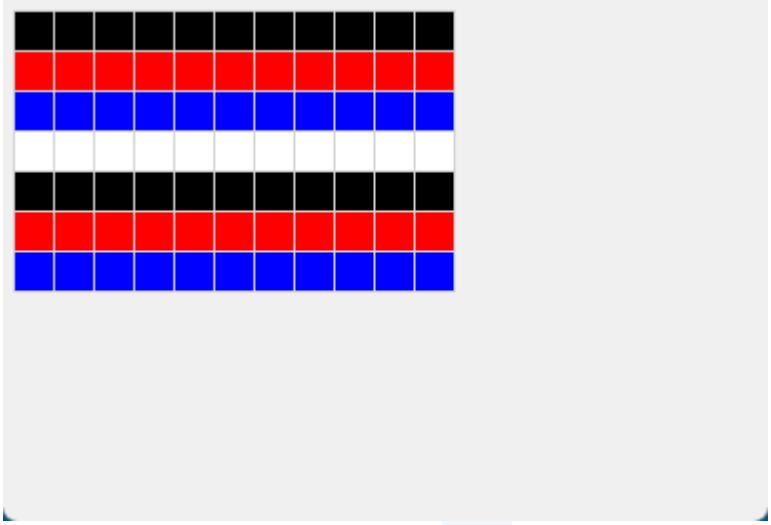
Aby sme takýto výpis mohli zavolať viackrát za sebou a zakaždým sa zobrazil aktuálny obsah tabuľky, trochu vylepšíme funkciu:

```
import tkinter

def kresli(tab, d=20, farby=('white', 'black', 'red', 'blue')):
    canvas.delete('all')
    for r, riadok in enumerate(tab):
        for s, prvok in enumerate(riadok):
            x, y = s*d + 5, r*d + 5
            farba = farby[prvok % len(farby)]
            canvas.create_rectangle(x, y, x+d, y+d,
                                    fill=farba, outline='light gray')
    canvas.update()
```

Otestujeme:

```
canvas = tkinter.Canvas()
canvas.pack()

t = vyrob(7, 11)
ocisluj(t)
kresli(t)
canvas.after(1000)

for i, riadok in enumerate(t):
    for j in range(len(riadok)):
        riadok[j] = i+1
kresli(t)

tkinter.mainloop()
```

Najprv sa vykreslí predchádzajúci obrázok a po sekunde v textovom okne a obrázok prekreslí:



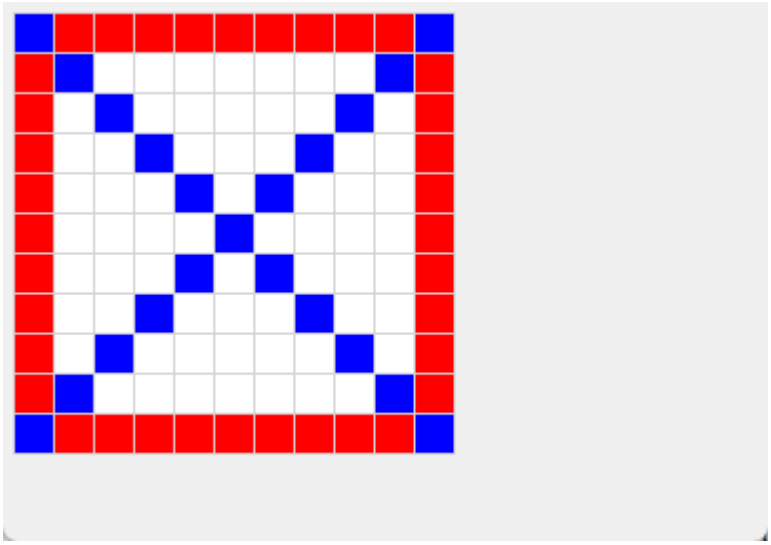
Všimnite si, že lokálnu premennú `farby` sme presunuli medzi parametre. Vďaka tomuto, budeme môcť túto funkciu zavolať aj takto:

```
kresli(t, 15, ('white', 'black', 'red', 'blue', 'yellow'))
```

Ďalším testom najprv vytvoríme tabuľku veľkosti 11x11 s hodnotou `0` (zobrazila by sa ako biela tabuľka). Potom vyrobíme červený rámik, t.j. po obvode zapíšeme hodnoty `2` a na záver na obe uhlopriečky zapíšeme hodnoty `3`, t.j. po vykreslení budú modré:

```
canvas = tkinter.Canvas()
canvas.pack()

n = 11
t = vyrob(n, n)
for i in range(n):
    for j in range(n):
        if i == 0 or i == n-1 or j == 0 or j == n-1:
            t[i][j] = 2
    t[i][i] = t[i][n-1-i] = 3
kresli(t)

tkinter.mainloop()
```

Nakreslí:



Niekoľko príkladov práce s dvojrozmernými tabuľkami

1. zvýšime obsah všetkých prvkov o 1:

```
2. def zvys_o_1(tab):
3.     for riadok in tab:
4.         for i in range(len(riadok)):
5.             riadok[i] += 1
```

Zrejme všetky prvky tejto tabuľky musia byť nejaké čísla, inak by funkcia spadla na chybe.

```
>>> p = [[5, 6, 7], [0, 0, 0], [-3, -2, -1]]
>>> zvys_o_1(p)
>>> p
    [[6, 7, 8], [1, 1, 1], [-2, -1, 0]]
```

6. podobný cieľ má aj druhá funkcia: hoci nemení samotnú tabuľku, vytvorí novú, ktorej prvky sú o jedna väčšie ako v pôvodnej tabuľke:

```
7. def o_1_viac(tab):
8.     nova_tab = []
9.     for riadok in tab:
10.        novy_riadok = [0] * len(riadok)
11.        for i in range(len(riadok)):
12.            novy_riadok[i] = riadok[i] + 1
13.        nova_tab.append(novy_riadok)
14.    return nova_tab
```

To isté trochu inak:

```
def o_1_viac(tab):
    nova_tab = []
    for riadok in tab:
        novy_riadok = list(riadok)          # kópia pôvodného riadka
        for i in range(len(novy_riadok)):
            novy_riadok[i] += 1
        nova_tab.append(novy_riadok)
    return nova_tab
```

15. kópia dvojrozmernej tabuľky:

```
16. def kopia(tab):
17.     nova_tab = []
18.     for riadok in tab:
19.         nova_tab.append(list(riadok))
20.     return nova_tab
```

21. číslovanie prvkov tabuľky inak ako to robila funkcia `cisluj()`: nie po riadkoch ale po stĺpcoch. Predpokladáme, že všetky riadky sú rovnako dlhé:

```
22. def ocisluj_po_stlpcoch(tab):
23.     poc = 0
24.     for j in range(len(tab[0])):
25.         for i in range(len(tab)):
26.             tab[i][j] = poc
27.             poc += 1
```

Všimnite si, že táto funkcia má oproti pôvodnému `ocisluj()` vymenené dva riadky for-cyklov.

```
>>> a = vyrob(3, 5)
```

```
>>> ocisluj_po_stlpcoch(a)
>>> vypis(a)
    0 3 6 9 12
    1 4 7 10 13
    2 5 8 11 14
```

28. spočítame počet výskytov nejakej hodnoty:

```
29. def pocet(tab, hodnota):
30.     vysl = 0
31.     for riadok in tab:
32.         for prvok in riadok:
33.             if prvok == hodnota:
34.                 vysl += 1
35.     return vysl
```

Využili sme tu prvú verziu funkcie (šablóny) pre výpis dvojrozmernej tabuľky. Ak si ale pripomenieme, že niečo podobné robí štandardná metóda `count()`, ale táto funguje len pre jednorozmerné zoznamy, môžeme našu funkciu vylepšiť:

```
def pocet(tab, hodnota):
    vysl = 0
    for riadok in tab:
        vysl += riadok.count(hodnota)
    return vysl
```

Otestujeme:

```
>>> a = [[1, 2, 1, 2], [4, 3, 2, 1], [2, 1, 3, 1]]
>>> pocet(a, 1)
    5
>>> pocet(a, 4)
    1
>>> pocet(a, 5)
    0
```

36. funkcia zistí, či je nejaká matica (dvojrozmerný zoznam) symetrická, t. j. či sú prvky pod a nad hlavnou uhlopriečkou rovnaké, čo znamená, že má platiť `matica[i][j] == matica[j][i]` pre každé `i` a `j`:

```
37. def symetricka(matica):
38.     vysl = True
39.     for i in range(len(matica)):
40.         for j in range(len(matica[i])):
41.             if matica[i][j] != matica[j][i]:
42.                 vysl = False
43.     return vysl
```

Hoci je toto riešenie korektné, má niekoľko nedostatkov:

- funkcia zbytočne testuje každú dvojicu prvkov `matica[i][j]` a `matica[j][i]` dvakrát, napríklad či `matica[0][2] == matica[2][0]` aj `matica[2][0] == matica[0][2]`, tiež zrejme netreba kontrolovať prvky na hlavnej uhlopriečke, či `matica[i][i] == matica[i][i]`
- keď sa vo vnútornom cykle zistí, že sme našli dvojicu `matica[i][j]` a `matica[j][i]`, ktoré sú navzájom rôzne, hoci sa zapamätá, že výsledok funkcie bude `False`, ďalej sa pokračuje prehľadávať zvyšok matice - toto je zrejme zbytočné, lebo výsledok je už známy - asi by sme mali vyskočiť z týchto cyklov; POZOR! príkaz `break` ale neurobí to, čo by sa nám tu hodilo:

```
def symetricka(matica):
```

```
    vysl = True
    for i in range(len(matica)):
        for j in range(len(matica[i])):
            if matica[i][j] != matica[j][i]:
                vysl = False
                break                    # vyskočí z cyklu
    return vysl
```

Takéto vyskočenie z cyklu nám veľmi nepomôže, lebo vyskakuje sa len z vnútorného a ďalej sa pokračuje vo vonkajšom. Našťastie my tu nepotrebujeme vyskakovať z cyklu, ale môžeme priamo ukončiť celú funkciu aj s návratovou hodnotou `False`.

Prepíšme funkciu tak, aby zbytočne dvakrát nekontrolovala každú dvojicu prvkov a aby sa korektne ukončila, keď nájde nerovnakú dvojicu:

```
def symetricka(matica):
    for i in range(1, len(matica)):
        for j in range(i):
            if matica[i][j] != matica[j][i]:
                return False
    return True
```

44. funkcia vráti pozíciu prvého výskytu nejakej hodnoty, teda dvojicu `(riadok, stĺpec)`. Keďže budeme potrebovať poznať indexy konkrétnych prvkov zoznamu, použijeme šablónu s indexmi:

```
45. def index(tab, hodnota):
46.     for i in range(len(tab)):
47.         for j in range(len(tab[i])):
48.             if tab[i][j] == hodnota:
49.                 return i, j
```

Funkcia skončí, keď nájde prvý výskyt hľadanej hodnoty (prechádza po riadkoch zľava doprava):

```
>>> a = [[1, 2, 1, 2], [1, 2, 3, 4], [2, 1, 3, 1]]
>>> index(a, 3)
    (1, 2)
>>> index(a, 5)
```

Na tomto poslednom príklade vidíme, že naša funkcia `index()` v nejakom prípade nevrátila „nič“. My už vieme, že vrátila špeciálnu hodnotu `None`, ktorá sa ale v príkazovom režime nevypíše. Ak by sme výsledok volania funkcie vypísali príkazom `print()`, dozvieme sa:

```
>>> print(index(a, 5))
    None
```

hodnota `None`

Táto špeciálna hodnota je výsledkom všetkých funkcií, ktoré nevracajú žiadnu hodnotu pomocou `return`. To znamená, že každá funkcia ukončená bez `return` v skutočnosti vracia `None` ako keby posledným príkazom funkcie bol:

```
return None
```

Túto hodnotu môžeme často využívať v situáciách, keď chceme nejako oznámiť, že napríklad výsledok hľadania bol neúspešný. Tak ako to bolo v prípade našej funkcie `index()`, ktorá v prípade, že sa v tabuľke hľadaná hodnota nenašla, vrátila `None`. Je zvykom takýto výsledok testovať takto:

```
vysledok = index(tab, hodnota)
if vysledok is None:
```

```
    print('nenasiel')
else:
    riadok, stlpec = vysledok
```

Teda namiesto testu `premenna == None` alebo `premenna != None` radšej používame `premenna is None` alebo `premenna is not None`.

Tabuľky s rôzne dlhými riadkami

Doteraz sme predpokladali, že všetky riadky dvojrozmernej štruktúry majú rovnakú dĺžku. Niekedy sa ale stretáme so situáciou, keď riadky budú rôzne dlhé. Napríklad:

```
>>> pt = [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
>>> vypis(pt)
    1
    1 1
    1 2 1
    1 3 3 1
    1 4 6 4 1
    1 5 10 10 5 1
```

Tento zoznam obsahuje prvých niekoľko riadkov Pascalovho trojuholníka. Našťastie funkciu `vypis()` (obe verzie) sme napísali tak, že správne vypíšu aj zoznamy s rôzne dlhými riadkami.

Niektoré tabuľky nemusia mať takto pravidelný tvar, napríklad:

```
>>> delitele = [[6, 2, 3], [13, 13], [280, 2, 2, 2, 5, 7], [1]]
>>> vypis(delitele)
    6 2 3
    13 13
    280 2 2 2 5 7
    1
```

Zoznam `delitele` má v každom riadku rozklad nejakého čísla (prvý prvok) na prvočinitele (súčin zvyšných prvkov).

Preto už pri zostavovaní funkcií musíme niekedy myslieť na to, že parametrom môže byť aj zoznam s rôznou dĺžkou riadkov. Zapíšme funkciu, ktorá nám vráti zoznam všetkých dĺžok riadkov danej dvojrozmernej štruktúry:

```
def dlzky(tab):
    vysl = []
    for riadok in tab:
        vysl.append(len(riadok))
    return vysl
```

Pre naše dva príklady zoznamov dostávame:

```
>>> dlzky(pt)
    [1, 2, 3, 4, 5, 6]
>>> dlzky(delitele)
    [3, 2, 6, 1]
```

Podobným spôsobom môžeme generovať nové dvojrozmerné štruktúry s rôznou dĺžkou riadkov, pre ktoré poznáme práve len tieto dĺžky:

```
def vyrob_d(dlzky, hodnota=0):
```

```
    vysl = []
    for dlzka in dlzky:
        vysl.append([hodnota] * dlzka)
    return vysl
```

Otestujeme:

```
>>> m1 = vyrob_d([3, 0, 1])
>>> m1
    [[0, 0, 0], [], [0]]
>>> m2 = vyrob_d(dlzky(delitele), 1)
>>> vypis(m2)
    1 1 1
    1 1
    1 1 1 1 1 1
    1
```

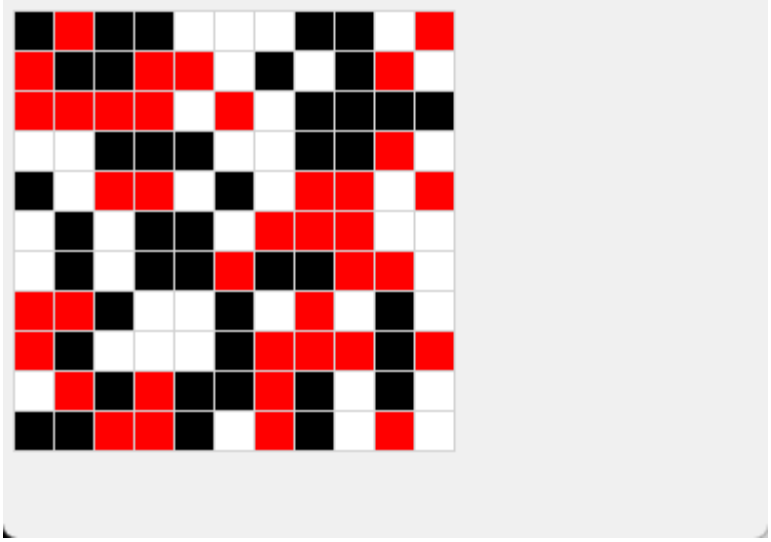
Zamyslite sa, ako budú vyzerať tieto zoznamy:

```
>>> n = 7
>>> m3 = vyrob_d([n] * n)
>>> m4 = vyrob_d(range(n))
>>> m5 = vyrob_d(range(n, 0, -2))
```

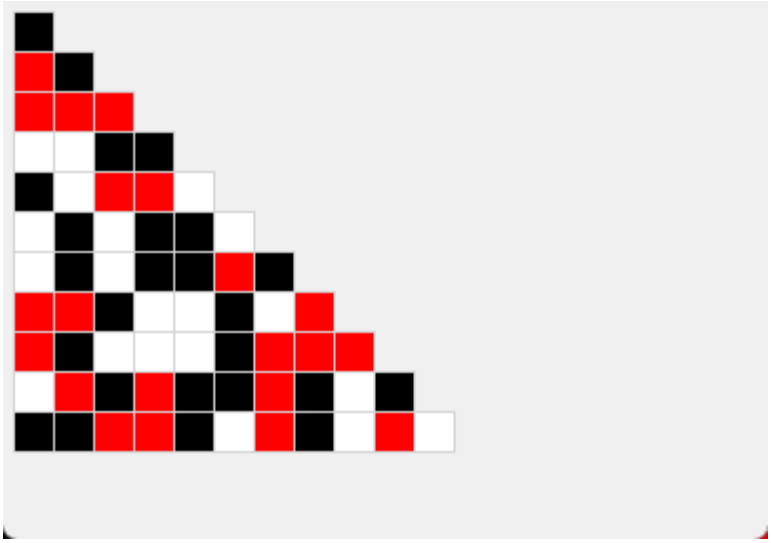
Ukážme, ako vyzerá dvojrozmerná tabuľka s rôzne dlhými riadkami v grafickej ploche. Najprv vygenerujeme náhodný obsah tabuľky a potom každý riadok skrátime:

```
import random

canvas = tkinter.Canvas()
canvas.pack()

n = 11
t = vyrob(n, n)                          # tabuľka n x n samých 0
for riadok in t:
    for i in range(n):
        riadok[i] = random.randint(0, 2)  # všetky prvky sú náhodné z <0, 2>
kresli(t)
canvas.after(1000)
for i in range(n):
    t[i] = t[i][:i + 1]                  # každý i-ty riadok sa skráti na i+1 prvkov
kresli(t)

tkinter.mainloop()
```

Najprv bude:



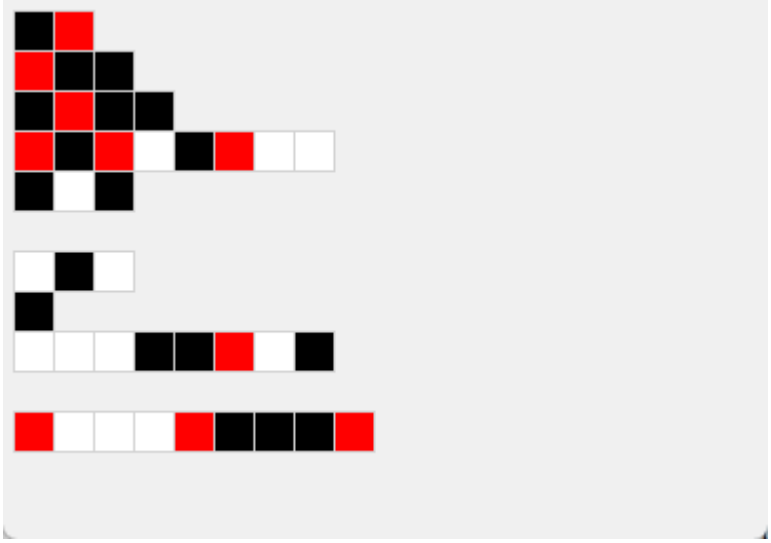
po sekunde:



Hoci, keby sme riadky tabuľky skracovali namiesto pôvodného cyklu takto:

```
for i in range(n):
    t[i] = t[i][:random.randrange(n)]
```

Mohli by sme dostať takéto zobrazenie (niektoré riadky tabuľky môžu byť teraz prázdne):



Hra LIFE

Informácie k tejto informatickej simulačnej hre nájdete na wikipedii

Pravidlá:

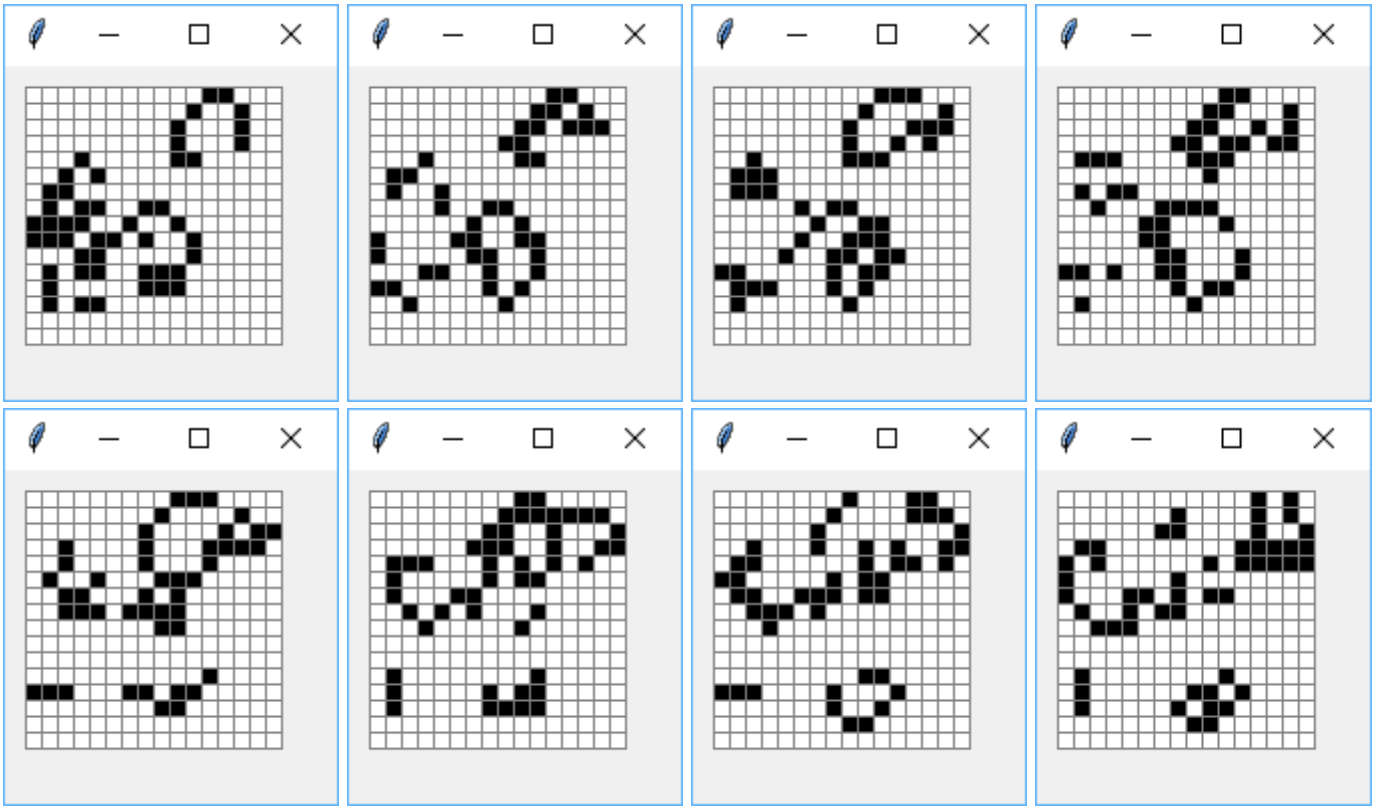
- v nekonečnej štvorcovej sieti žijú bunky, ktoré sa rôzne rozmnožujú, resp. umierajú
- v každom políčku siete je buď živá bunka, alebo je políčko prázdne (budeme označovať ako **1** a **0**)
- každé políčko má 8 susedov (vodorovne, zvislo aj po uhlopriečke)
- v každej generácii sa s každým jedným políčkom urobí:
    - ak je na políčku bunka a má práve 2 alebo 3 susedov, tak táto bunka prežije aj do ďalšej generácie
    - ak je na políčku bunka a má buď 0 alebo 1 suseda, alebo viac ako 3 susedov, tak bunka na tomto políčku do ďalšej generácie neprežije (umiera)
    - ak má prázdne políčko presne na troch susediacich políčkach živé bunky, tak sa tu v ďalšej generácii narodí nová bunka

Štvorcovú sieť s `0` a `1` budeme ukladať v dvojrozmernej tabuľke veľkosti `n` x `n`. V tejto tabuľke je momentálna generácia bunkových živočíchov. Na to, aby sme vyrobili novú generáciu, si pripravíme pomocnú tabuľku rovnakej veľkosti a do nej budeme postupne zapisovať bunky novej generácie. Keď už bude celá táto pomocná tabuľka hotová, prekopírujeme ju do pôvodnej tabuľky. Dvojrozmernú tabuľku budeme vykresľovať do grafickej plochy.

```
import tkinter
import random

def nahodne(n):
    vysl = []
    for i in range(n):
        vysl.append([])
        for j in range(n):
            vysl[-1].append(random.randrange(2))
##    vysl[3][1] = vysl[3][2] = vysl[3][3] = vysl[2][3] = vysl[1][2] = 1
##    vysl[3][28] = vysl[3][27] = vysl[3][26] = vysl[2][26] = vysl[1][27] = 1
    return vysl

def kresli(tab, d=8):
    canvas.delete('all')
    for r, riadok in enumerate(tab):
        for s, prvok in enumerate(riadok):
            x, y = s*d + 5, r*d + 5
            farba = ('white', 'black')[prvok]
            canvas.create_rectangle(x, y, x+d, y+d, fill=farba, outline='lightgray')
    canvas.update()

def nova_generacia(p):
    nova = []
    for r in range(len(p)):
        nova.append([0] * len(p[r]))
    for r in range(1, len(p)-1):
        for s in range(1, len(p[r])-1):
            ps = (p[r-1][s-1] + p[r-1][s] + p[r-1][s+1] +
                  p[r][s-1]   +             p[r][s+1] +
                  p[r+1][s-1] + p[r+1][s] + p[r+1][s+1])
            if ps==3 or ps==2 and p[r][s]:
                nova[r][s] = 1
    return nova

canvas = tkinter.Canvas(width=600, height=600)
canvas.pack()

plocha = nahodne(50)
kresli(plocha)
```

```
for i in range(1000):
    plocha = nova_generacia(plocha)
    kresli(plocha)

tkinter.mainloop()
```

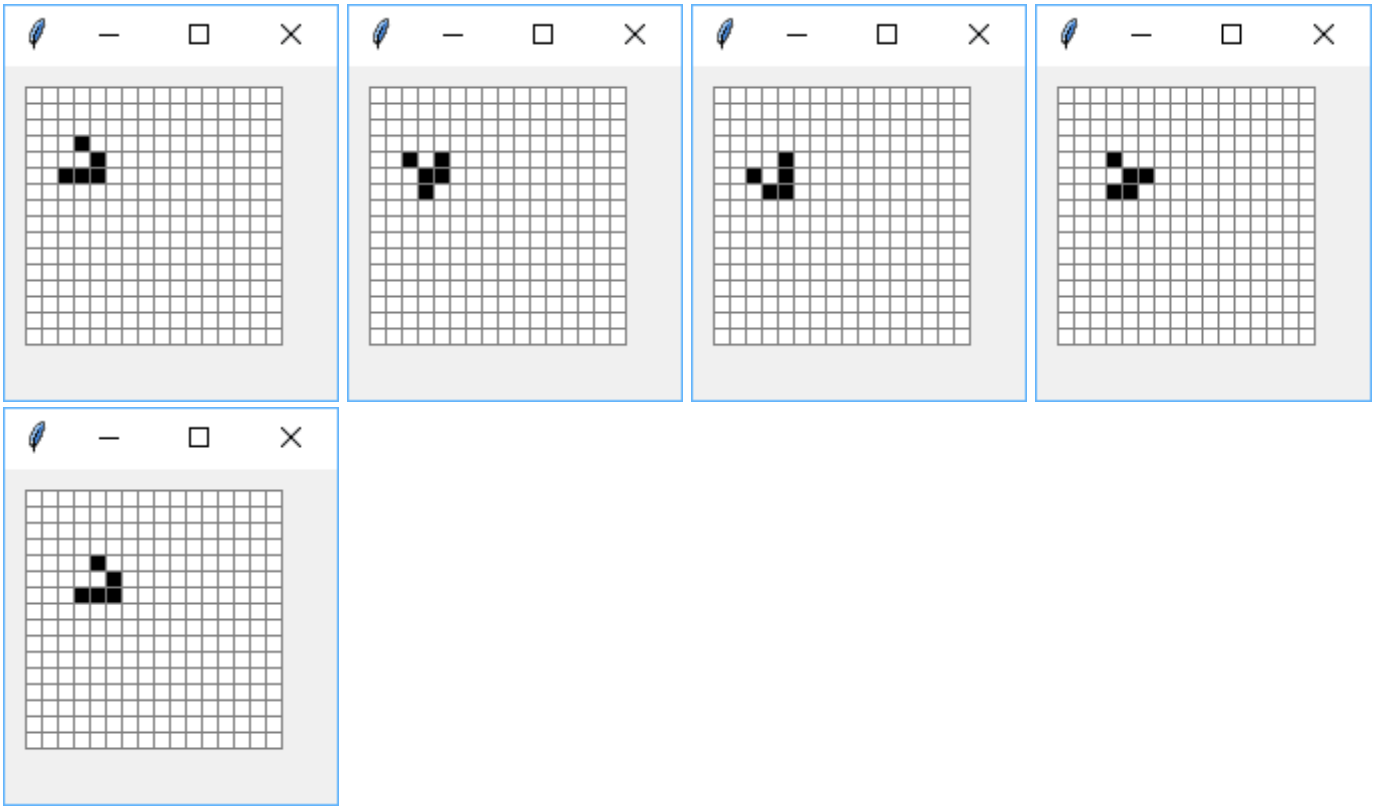
Na tejto sérii obrázkov môžete sledovať, ako sa s nejakej náhodnej pozície postupne generujú ďalšie generácie:



Namiesto náhodného obsahu môžeme vytvoriť prázdnu (vynulovanú) sieť, do ktorej priradíme:

```
tab[5][2] = tab[5][3] = tab[5][4] = tab[4][4] = tab[3][3] = 1
```

Dostávame takýto klzák (glider), ktorý sa pohybuje po ploche nejakým smerom:



Všimnite si, že po 4 generáciách má rovnaký tvar, ale je posunutý o 1 políčko dole a vpravo.

Cvičenia

L.I.S.T.

- riešenia **aspoň 12 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- pozri si Riešenie úloh 13. cvičenia

1. Modifikuj funkciu `vypis(tab, sirka=4)`, ktorá vypisuje dvojrozmernú tabuľku do riadkov, pričom každý prvok je formátovaný na zadanú šírku, napríklad pre `sirka` = `5` takto `f'{repr(prvok):>5}'`. Otestuj:

```
>>> vypis([[1, 6, 3.14], [0.5, 1.5, 2.5]], 5)
    1    6  3.14
  0.5  1.5   2.5
>>> vypis([[1, 2, 3], [None, None], ['4', '5', '6'], ['Python', 3.9]])
    1    2    3
 None None
  '4'  '5'  '6'
 'Python'  3.9
```

2. Zadefinuj funkcie `max2(tab)`, `min2(tab)` a `sum2(tab)`, ktoré zistia najväčší prvok, najmenší prvok a súčet všetkých prvkov dvojrozmernej tabuľke. Využi štandardné funkcie `max()`, `min()` a `sum()`. Napríklad:

```
>>> p = [[1, 6, 3.14], [0.5, 1.5, 2.5]]
>>> max2(p)
6
>>> min2(p)
0.5
```

```
>>> sum2(p)
    14.64
>>> r = [[-1, -2], [-3, -4]]
>>> max2(r)
    -1
>>> min2(r)
    -4
```

3. Napíš funkciu `zoznam_suctov(tab)` počíta súčty prvkov v jednotlivých riadkoch tabuľky a ukladá ich do výsledného zoznamu. Všetky hodnoty v tabuľke sú celé čísla. Napríklad:

```
>>> suc = zoznam_suctov([[1, 2, 3], [4], [], [5, 6]])
>>> suc
    [6, 4, 0, 11]
```

4. Vylepši funkciu `zoznam_suctov(tab)` tak, aby fungovala nielen pre celočíselné hodnoty, ale pre ľubovoľný typ, v ktorom funguje operácia `+`. Pre prázdny riadok tabuľky, funkcia spočíta `None`. Napríklad:

```
>>> zoznam_suctov([['1', 'x', '2'], [], [5, 6], [3.1, 4], [(5, 6), (7,)]])
    ['1x2', None, 11, 7.1, (5, 6, 7)]
```

5. Napíš funkciu `pridaj_sucty(tab)`, ktorá podobne ako v predchádzajúcej úlohe počíta súčty prvkov po riadkoch, ale ich ukladá na koniec každého riadka tabuľky. Funkcia nič nevracia ani nevypisuje. Namiesto toho modifikuje vstupnú tabuľku. Napríklad:

```
>>> a = [[1, 2, 3], [4], [5, 6]]
>>> pridaj_sucty(a)
>>> vypis(a)
      1   2   3   6
      4   4
      5   6  11
>>> t = [['1', 'x', '2'], [], [5, 6], [3.1, 4], [(5, 6), (7,)]]
>>> pridaj_sucty(t)
>>> vypis(t, 7)
       '1'     'x'     '2'   '1x2'
      None
         5       6      11
       3.1       4     7.1
    (5, 6)    (7,) (5, 6, 7)
```

6. Napíš funkciu `preklop(tab)`, ktorá vyrobí novú dvojrozmernú tabuľku, v ktorej bude pôvodná tabuľka preklopená okolo hlavnej uhlopriečky (vymenené riadky a stĺpce). Predpokladaj, že všetky riadky majú rovnakú dĺžku. Napríklad:

```
>>> p = [[1, 2], [5, 6], [3, 4]]
>>> vypis(preklop(p), 2)
    1  5  3
    2  6  4
>>> vypis(p, 2)
    1  2
    5  6
    3  4
```

7. Zadefinuj funkciu `ocisluj2(tab)`, ktorá zmení všetky prvky dvojrozmernej tabuľky tak, že ich postupne prechádza po stĺpcoch a čísluje ich celými číslami od `0`. Riadky tabuľky nemusia mať rovnakú dĺžku. Napríklad:

```
>>> ab = [[1, 1], [], [1, 1, 1], [1], [1, 1, 1, 1]]
>>> ocisluj2(ab)
>>> vypis(ab)
      0    4

      1    5    7
      2
      3    6    8    9
```

8. Zadefinuj funkciu `pascalov_trojuholnik(n)`, ktorá vygeneruje prvých `n` riadkov pascalovho trojuholníka a uloží ich do dvojrozmernej tabuľky. Pri vytváraní každého nasledovného riadka tabuľky využi predchádzajúci riadok (každý prvok v ňom je súčtom dvoch susedných). Napríklad:

```
>>> pt = pascalov_trojuholnik(5)
>>> vypis(pt)
      1
      1    1
      1    2    1
      1    3    3    1
      1    4    6    4    1
```

9. Textový súbor v každom riadku obsahuje niekoľko slov, oddelených medzerou (riadok môže byť aj prázdny). Napíš funkciu `citaj(meno_suboru)`, ktorá prečíta tento súbor a vyrobí z neho dvojrozmernú tabuľku slov: každý riadok tabuľky zodpovedá jednému riadku súboru. Napríklad, ak súbor `'text.txt'` obsahuje:

```
Anicka dusicka
kde si bola
ked si si cizmicky
zarosila
```

potom

```
>>> x = citaj('text.txt')
>>> x
   [['Anička', 'dušička'], ['kde', 'si', 'bola'], ['keď', 'si', 'si', 'čižmičky'], ['za
rosila']]
```

10. Funkcia `zapis(tab, meno_suboru)` je opačná k funkcii `citaj` v predchádzajúcej úlohe: zapíše danú dvojrozmernú tabuľku slov do súboru. Napríklad:

```
>>> x = [['Anička', 'dušička'], ['kde', 'si', 'bola'], ['keď', 'si', 'si', 'čižmičky'],
['zarosila']]
>>> zapis(x, 'text1.txt')
```

vytvorí rovnaký súbor ako bol `'text.txt'`.

Uvedom si, že ak by vstupná dvojrozmerná tabuľka obsahovala čísla, táto funkcia vytvorí korektný súbor čísel, napríklad:

```
>>> zapis([[1, 11, 21], [345], [-5, 10]], 'cisla.txt')
```

vytvorí súbor `'cisla.txt'`:

```
1 11 21
345
-5 10
```

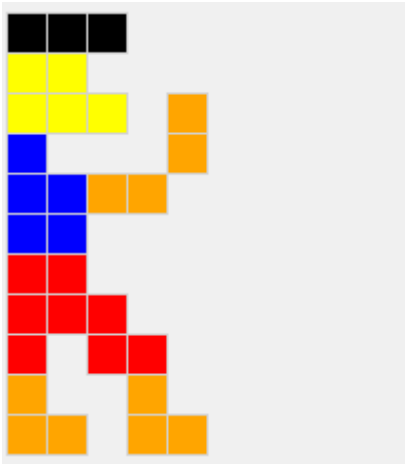
11. Funkcia `citaj_cisla(meno_suboru)` bude podobná funkcii `citaj(meno_suboru)` z (9) úlohy, len táto predpokladá, že vstupný súbor obsahuje len celé čísla. Funkcia vráti dvojrozmernú tabuľku čísel. Napríklad, pre textový súbor `'cisla.txt'` z (10) úlohy:

```
>>> tab = citaj_cisla('cisla.txt')
>>> tab
    [[1, 11, 21], [345], [-5, 10]]
```

12. Z prednášky uprav funkciu `kresli`:

```
def kresli(tab, d=20, farby=('black', 'yellow', 'orange', 'blue', 'red', 'white')):
    canvas.delete('all')
    for r, riadok in enumerate(tab):
        for s, prvok in enumerate(riadok):
            x, y = s*d + 5, r*d + 5
            farba = farby[prvok]
            canvas.create_rectangle(x, y, x+d, y+d,
                                    fill=farba, outline='light gray')
    canvas.update()
```

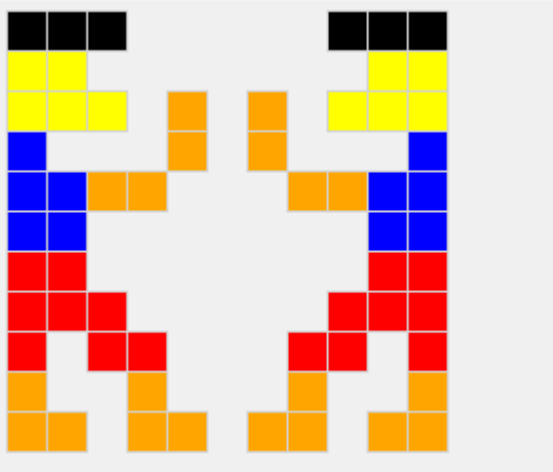
tak, aby sa prvky tabuľky, ktoré majú hodnotu `None`, nekreslili (ostalo po nich prázdne miesto). Teraz vytvor dvojrozmernú tabuľku `p` (všetky riadky majú rovnakú dĺžku `5`), po vykreslení ktorej dostávaš takýto obrázok:



13. Napíš funkciu `zrkadlo(obr)`, ktorá z dvojrozmernej tabuľky `obr` vyrobí novú tak, že každému riadku pridá `None` a zrkadlový obsah riadka. Napríklad pre obrázok z (12) úlohy:

```
kresli(zrkadlo(p))
```

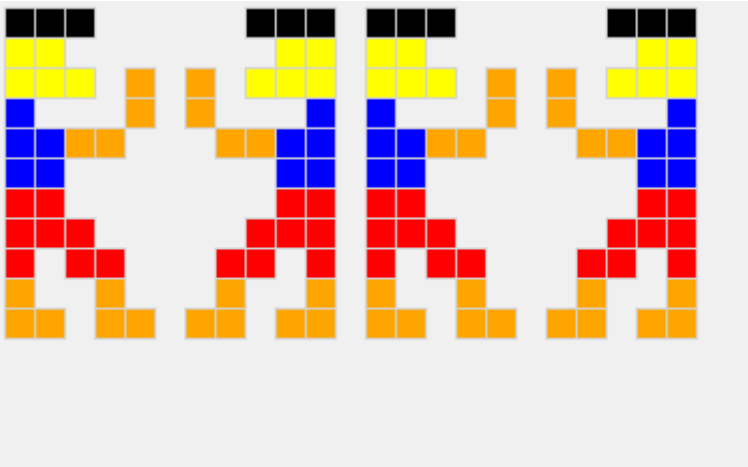
nakreslí:



a potom aj:

```
kresli(zrkadlo(zrkadlo(p)), 15)
```

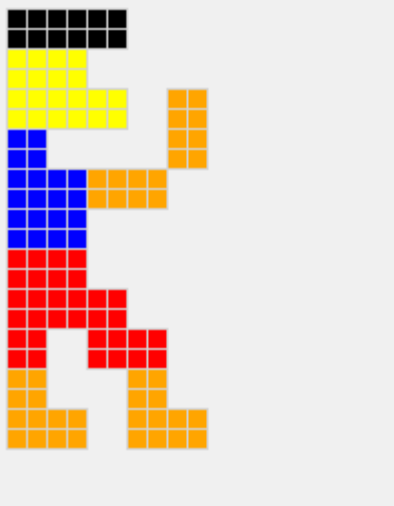
nakreslí:



14. Napíš funkciu `zvacsi(obr)`, ktorá z dvojrozmernej tabuľky `obr` vyrobí novú tak, že sa dvojnásobne zväčší: nová tabuľka má dvojnásobný počet riadkov aj stĺpcov a každý pôvodný prvok tu teraz bude 4-krát. Funkcia nezmení pôvodnú tabuľku `obr`. Napríklad pre obrázok z (12) úlohy:

```
15. kresli(zvacsi(p), 10)
```

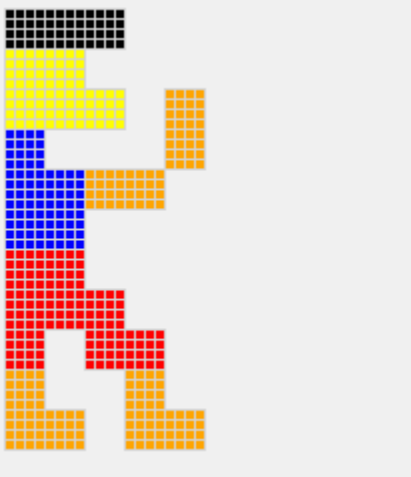
nakreslí:



a potom aj:

```
kresli(zvacsi(zvacsi(p)), 5)
```

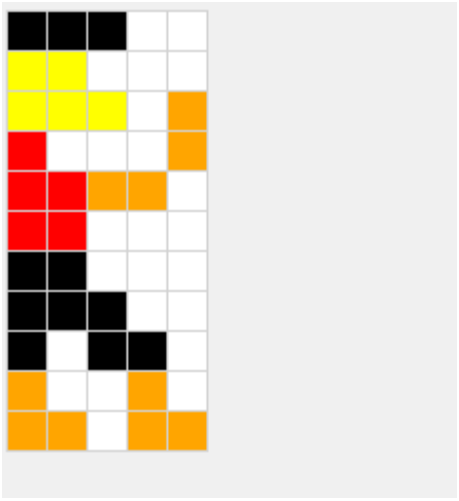
nakreslí:



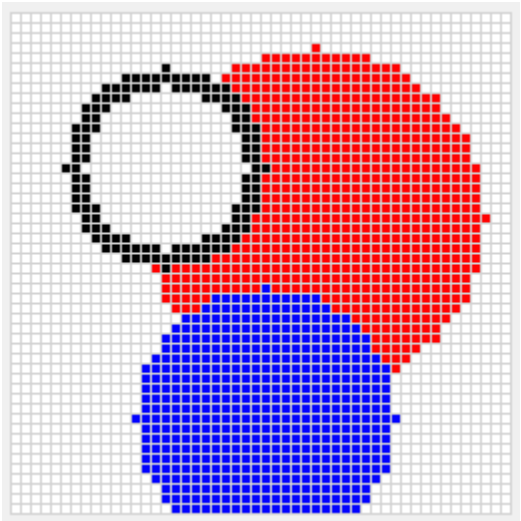
15. Napíš funkciu `nahrad(obr, post)`, ktorá z dvojrozmernej tabuľky `obr` vyrobí novú tak, že zoberie postupnosť dvojíc `post`, každá dvojica obsahuje dve hodnoty (čo nahradiť, čím nahradiť) a postupne všetky prvky pôvodnej tabuľky nahradí podľa tohto zoznamu. Napríklad pre obrázok z (12) úlohy:

```
kresli(nahrad(p, ((3, 4), (4, 0), (None, 5))))
```

nakreslí:



16. Napíš funkciu `kruh(tab, r, r1, s1, hodnota)`, ktorá bude vypĺňať nejakú oblasť dvojrozmernej tabuľky zadanou hodnotou `hodnota`. Touto oblasťou bude kruh s polomerom `r` a so stredom `r1`, `s1` (riadok, stĺpec). Funkcia by mohla fungovať tak, že postupne skontroluje všetky prvky tabuľky, či ich „vzdialenosť“ od stredu je menšia alebo rovná `r` a vtedy im zmení hodnoty. Funkcia nič nevracia ani nevypisuje. Funkcia modifikuje obsah tabuľky. Napríklad po nakreslení štyroch kruhov v tabuľke 50x50 môžeš dostať:



17. Napíš funkciu `do_radu(tab)`, ktorá vráti vytvorenú jednorozmernú tabuľku (zoznam `list`) z riadkov dvojrozmernej tabuľky `tab`. Použi na to jeden for-cyklus a metódu `extend` - štandardná funkcia, pomocou ktorej môžeme k nejakému zoznamu prilepiť na koniec naraz viac prvkov (na rozdiel od `append`, ktorý vie prilepiť len jeden prvok). Napríklad:

```
>>> tab1 = [[1], [2, 3, 4], [5, 6], [7]]
>>> zoz = do_radu(tab1)
>>> zoz
    [1, 2, 3, 4, 5, 6, 7]
>>> do_radu([['prvy'], [], ['druhy', 'treti']])
    ['prvy', 'druhy', 'treti']
```

18. Napíš funkciu `do_dvojrozmernej(postupnost, sirka)`, ktorá bude v istom zmysle fungovať naopak ako funkcia `do_radu` z predchádzajúceho príkladu: funkcia dostáva postupnosť nejakých hodnôt

(napríklad jednorozmerný zoznam) a vyrobí z nej dvojrozmernú tabuľku, v ktorej okrem posledného riadku majú všetky zadanú šírku. Posledný riadok môže byť kratší. Napríklad:

```
>>> t1 = do_dvojrozmernej(range(10), 3)
>>> vypis(t1)
   0  1  2
   3  4  5
   6  7  8
   9
>>> t2 = do_dvojrozmernej(do_radu(t1), 5)
>>> vypis(t2)
   0  1  2  3  4
   5  6  7  8  9
>>> vypis(do_dvojrozmernej('programovanie', 5))
  'p'  'r'  'o'  'g'  'r'
  'a'  'm'  'o'  'v'  'a'
  'n'  'i'  'e'
```

7. Týždenný projekt

L.I.S.T.

- riešenie odovzdaj na úlohový server https://list.fmph.uniba.sk/

Napíš pythonovský skript, v ktorom zadefinuješ päť funkcií na prácu so zoznamami:

- `number_of_lists(zoznam)` … vráti `int`
- `get_elements(zoznam)` … vráti `tuple`
- `flat_list(zoznam)` … mení `zoznam`, teda vráti `None`
- `nested_replace(zoznam, hodnota1, hodnota2)` … vráti `list`
- `change_values(zoznam, hodnota1, hodnota2)` … mení `zoznam`, teda vráti `None`

Vstupom pre tieto funkcie môže byť zoznam, ktorý môže obsahovať nielen čísla a reťazce, ale aj ďalšie podzoznamy. Tieto podzoznamy môžu opäť obsahovať ďalšie podzoznamy, atď. Napríklad aj takýto zoznam `['a', ['dom', [2], 3], [], [[[2]]], 'b']` môže byť vstupom do tvojich funkcií.

number_of_lists()

Funkcia `number_of_lists(zoznam)` vráti počet zoznamov, ktoré sa nachádzajú v danom vstupnom parametri. Napríklad:

```
>>> number_of_lists([1, 'a', 2])
    1
>>> number_of_lists([[], 1, 'a', [3], 2])
    3
>>> number_of_lists(['a', ['dom', [2], 3], [], [[[2]]], 'b'])
    7
>>> number_of_lists([1, '[]', 2])
    1
>>> number_of_lists((1, 2))
    0
```

get_elements()

Funkcia `get_elements(zoznam)` vráti sploštený zoznam prvkov daného zoznamu (v tvare n-tice), teda taký, ktorý už neobsahuje žiadne podzoznamy. Funkcia vráti hodnotu v tvare `tuple`. Napríklad:

```
>>> print(get_elements([1, 2, 3, [4, 5], 6, [[[7]]], [], 8]))
    (1, 2, 3, 4, 5, 6, 7, 8)
>>> print(get_elements(['a', ['dom', [2], 3], [], [[[2]]], 'b']))
    ('a', 'dom', 2, 3, 2, 'b')
>>> print(get_elements([[], [[[]]], []]))
    ()
>>> zoz = [[[7]], 8]
>>> print(get_elements(zoz))
    (7, 8)
>>> zoz
    [[[7]], 8]
```

Pôvodný zoznam pri tom ostane bez zmeny.

flat_list()

Funkcia `flat_list(zoznam)` sploští daný vstupný zoznam. Na rozdiel od predchádzajúcej funkcie táto nič nevracia, len modifikuje vstupný zoznam. Napríklad:

```
>>> zoz = [[[7]], 8]
>>> print(flat_list(zoz))
    None
>>> zoz
    [7, 8]
>>> p = [1, 2, 3, [4, 5], 6, [[[7]]], [], 8]
>>> flat_list(p)
>>> p
    [1, 2, 3, 4, 5, 6, 7, 8]
```

nested_replace()

Funkcia `nested_replace(zoznam, hodnota1, hodnota2)` vráti kópiu pôvodného zoznamu, v ktorom budú všetky prvky vstupného zoznamu s hodnotou `hodnota1` change_valuesené hodnotou `hodnota2`. Ak sú nejakými prvkami opäť zoznamy, tak bude `hodnota1` change_valuesená `hodnotou2` aj v týchto zoznamoch, aj ich podzoznamoch atď. Napríklad:

```
>>> zoz = [[[7]], 8]
>>> print(nested_replace(zoz, 7, 'a'))
    [[['a']], 8]
>>> zoz
    [[[7]], 8]
>>> print(nested_replace([1, 2, 3, [1, 2], 3, [[[1]]], [], 2], 1, 'x'))
    ['x', 2, 3, ['x', 2], 3, [[['x']]], [], 2]
>>> print(nested_replace([3, [33, [333, [13], 13]], 36], 3, 'q'))
    ['q', [33, [333, [13], 13]], 36]
>>> print(nested_replace([3, [33, [333, [13], 13]], 36], [13], 'm'))
    [3, [33, [333, 'm', 13]], 36]
```

change_values()

Funkcia `change_values(zoznam, hodnota1, hodnota2)` change_valuesí v danom zozname `zoznam` všetky prvky s hodnotou `hodnota1` hodnotou `hodnota2`. Ak sú nejakými prvkami opäť zoznamy, tak nahrádza aj v týchto zoznamoch, aj v ich podzoznamoch atď. Funkcia nič nevracia, len modifikuje vstupný zoznam. Napríklad:

```
>>> zoz = [[[7]], 8]
>>> print(change_values(zoz, 7, 'a'))
    None
>>> zoz
```

```
    [[['a']], 8]
>>> p = [1, 2, 3, [1, 2], 3, [[[1]]], [], 2]
>>> change_values(p, 1, 'x')
>>> p
    ['x', 2, 3, ['x', 2], 3, [[['x']]], [], 2]
>>> p = [1, 2, 3, [1, 2], 3, [[[1]]], [], 2]
>>> change_values(p, 4, 'z')
>>> p
    [1, 2, 3, [1, 2], 3, [[[1]]], [], 2]
>>> p = ['a', ['dom', [2], 3], [], [[[2]]], 'b']
>>> change_values(p, 2, 'abc')
>>> p
    ['a', ['dom', ['abc'], 3], [], [[['abc']]], 'b']
```

Nemeň mená funkcií. Zrejme využiješ rekurziu.

Tvoj odovzdaný program s menom `riesenie.py` musí začínať tromi riadkami komentárov:

```
# 7. zadanie: zoznamy
# autor: Janko Hraško
# datum: 19.11.2021
```

Projekt `riesenie.py` odovzdaj na úlohový server https://list.fmph.uniba.sk/ najneskôr do 23:00 **26. novembra**, kde ho môžeš nechať otestovať. Môžeš zaň získať **5 bodov**.