4. Podmienky

video prezentácie

1. podmienený príkaz
2. podmienený cyklus

Podmienený príkaz

Pri programovaní často riešime situácie, keď sa má program na základe nejakej podmienky rozhodnúť medzi viacerými možnosťami. Napríklad, program má vypísať, či zadaný počet bodov stačí na známku z predmetu. Preto si najprv vyžiada číslo - získaný počet bodov, potom **porovná** túto hodnotu s požadovanou hranicou, napríklad 50 bodov a na základe toho vypíše, buď že je to dosť na známku, alebo nie je:

```
body = int(input('Zadaj získaný počet bodov: '))
if body >= 50:
    print(body, 'bodov je dostačujúci počet na známku')
else:
    print(body, 'bodov je málo na získanie známky')
```

Použili sme tu podmienený príkaz (príkaz vetvenia) `if`. Jeho zápis vyzerá takto:

```
if podmienka:    # ak podmienka platí, vykonaj 1. skupinu príkazov
    prikaz
    prikaz
    ...
else:            # ak podmienka neplatí, vykonaj 2. skupinu príkazov
    prikaz
    prikaz
    ...
```

V našom prvom príklade je v oboch skupinách príkazov len po jednom príkaze `print()`. Odsadenie skupiny príkazov (blok príkazov) má rovnaký význam ako vo for-cykle: budeme ich odsadzovať vždy presne o 4 medzery.

V pravidlách predmetu programovanie máme takéto kritériá na získanie známky:

- známka **A** aspoň 90 bodov
- známka **B** aspoň 80 bodov
- známka **C** aspoň 70 bodov
- známka **D** aspoň 60 bodov
- známka **E** aspoň 50 bodov
- známka **Fx** menej ako 50 bodov

Podmienka pre získanie známky **A**:

```
if body >= 90:
    print('za', body, 'bodov získavaš známku A')
```

```
else:
    ...
```

Ak je bodov menej ako 90, už to môže byť len horšia známka: dopíšeme testovanie aj známky **B**:

```
if body >= 90:
    print('za', body, 'bodov získavaš známku A')
else:
    if body >= 80:
        print('za', body, 'bodov získavaš známku B')
    else:
        ...
```

Všetky riadky v druhej skupine príkazov (za `else`) musia byť odsadené o 4 medzery, preto napríklad `print()`, ktorý vypisuje správu o známke **B** je odsunutý o 8 medzier. Podobným spôsobom zapíšeme všetky zvyšné podmienky:

```
body = int(input('Zadaj získaný počet bodov: '))
if body >= 90:
    print('za', body, 'bodov získavaš známku A')
else:
    if body >= 80:
        print('za', body, 'bodov získavaš známku B')
    else:
        if body >= 70:
            print('za', body, 'bodov získavaš známku C')
        else:
            if body >= 60:
                print('za', body, 'bodov získavaš známku D')
            else:
                if body >= 50:
                    print('za', body, 'bodov získavaš známku E')
                else:
                    print('za', body, 'bodov si nevyhovel a máš známku Fx')
```

Takéto odsadzovanie príkazov je v Pythone veľmi dôležité a musíme byť pritom veľmi presní. Príkaz `if`, ktorý sa nachádza vo vnútri niektorej vetvy iného `if`, sa nazýva **vnorený príkaz if**.

V Pythone existuje konštrukcia, ktorá uľahčuje takúto vnorenú sériu `if`-ov:

```
if podmienka_1:     # ak podmienka_1 platí, vykonaj 1. skupinu príkazov
    prikaz
    ...
elif podmienka_2:   # ak podmienka_1 neplatí, ale platí podmienka_2, ...
    prikaz
    ...
elif podmienka_3:   # ak ani podmienka_1 ani podmienka_2 neplatia, ale platí podmi
enka_3, ...
    prikaz
    ...
else:               # ak žiadna z podmienok neplatí, ...
    prikaz
    ...
```

Predchádzajúci program môžeme zapísať aj takto:

```
body = int(input('Zadaj získaný počet bodov: '))
if body >= 90:
    print('za', body, 'bodov získavaš známku A')
elif body >= 80:
    print('za', body, 'bodov získavaš známku B')
elif body >= 70:
    print('za', body, 'bodov získavaš známku C')
elif body >= 60:
    print('za', body, 'bodov získavaš známku D')
elif body >= 50:
    print('za', body, 'bodov získavaš známku E')
else:
    print('za', body, 'bodov si nevyhovel a máš známku Fx')
```

Ukážme ešte jedno riešenie tejto úlohy - jednotlivé podmienky zapíšeme ako intervaly:

```
body = int(input('Zadaj získaný počet bodov: '))
if body >= 90:
    print('za', body, 'bodov získavaš známku A')
if 80 <= body < 90:
    print('za', body, 'bodov získavaš známku B')
if 70 <= body < 80:
    print('za', body, 'bodov získavaš známku C')
if 60 <= body < 70:
    print('za', body, 'bodov získavaš známku D')
if 50 <= body < 60:
    print('za', body, 'bodov získavaš známku E')
if body < 50:
    print('za', body, 'bodov si nevyhovel a máš známku Fx')
```

V tomto riešení využívame to, že `else`-vetva v príkaze `if` môže chýbať, a teda pri neplatnej podmienke sa nevykoná nič:

```
if podmienka:        # ak podmienka platí, vykonaj skupinu príkazov
    prikaz
    prikaz
    ...
                     # ak podmienka neplatí, nevykonaj nič
```

Mohli by sme to zapísať aj takto:

```
if podmienka:        # ak podmienka platí, vykonaj skupinu príkazov
    prikaz
    prikaz
    ...
else:
    pass             # ak podmienka neplatí, nevykonaj nič
```

kde `pass` je tzv. **prázdny príkaz**, t.j. príkaz, ktorý nerobí nič. Môžeme ho použiť všade, kde chceme zapísať tzv. prázdny blok príkazov. Hoci je tento zápis správny, nezvykne sa takto používať.

Zrejme každý príkaz `if` po kontrole podmienky (a prípadnom výpise správy) pokračuje na ďalšom príkaze, ktorý nasleduje za ním (a má rovnaké odsadenie ako `if`). Okrem toho vidíme, že teraz sú niektoré podmienky trochu zložitejšie, lebo testujeme, či sa hodnota nachádza v nejakom intervale. (podmienku `80 <= body < 90` sme mohli zapísať aj takto `90 > body >= 80`)

V Pythone môžeme zapisovať podmienky podobne, ako je to bežné v matematike:

`body < 90`	je menšie ako
`body <= 50`	je menšie alebo rovné
`body == 50`	rovná sa
`body != 77`	nerovná sa
`body > 100`	je väčšie ako
`body >= 90`	je väčšie alebo rovné
`40 < body <= 50`	je väčšie ako … a zároveň menšie alebo rovné …
`a < b < c`	`a` je menšie ako `b` a zároveň je `b` menšie ako `c`

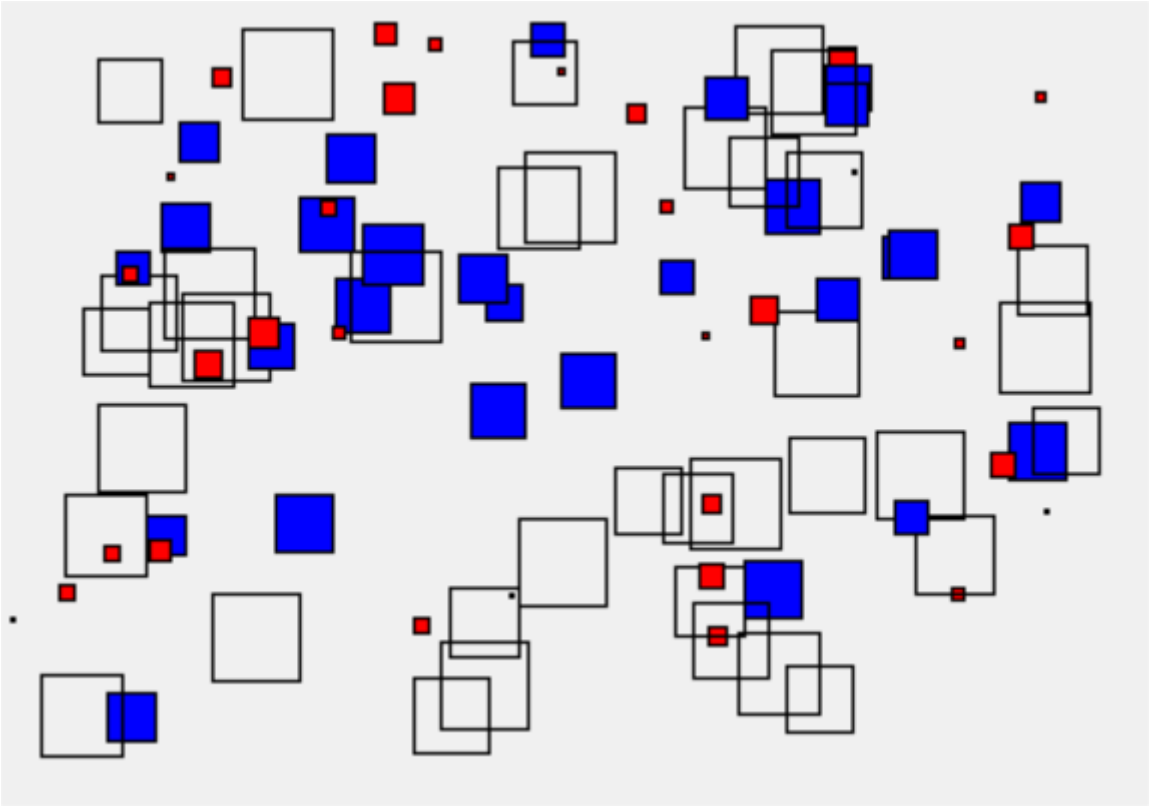
Ukážme použitie podmieneného príkazu aj v grafickom programe. Začneme s programom, ktorý na náhodné pozície nakreslí 100 rôzne veľkých štvorčekov. Aj veľkosť týchto štvorčekov bude náhodné číslo od 1 do 30. Štvorčeky bude zafarbovať podľa takéhoto pravidla:

- ak je ich veľkosť menšia alebo rovná 10, budú červené
- inak, ak ich veľkosť bude menšia alebo rovná 20, budú modré
- inak nebudú zafarbené, t.j. ich farba výplne bude `''`

Zrejme v poslednej skupine štvorcov budú len tie, ktoré sú väčšie ako 20. Zapíšme program:

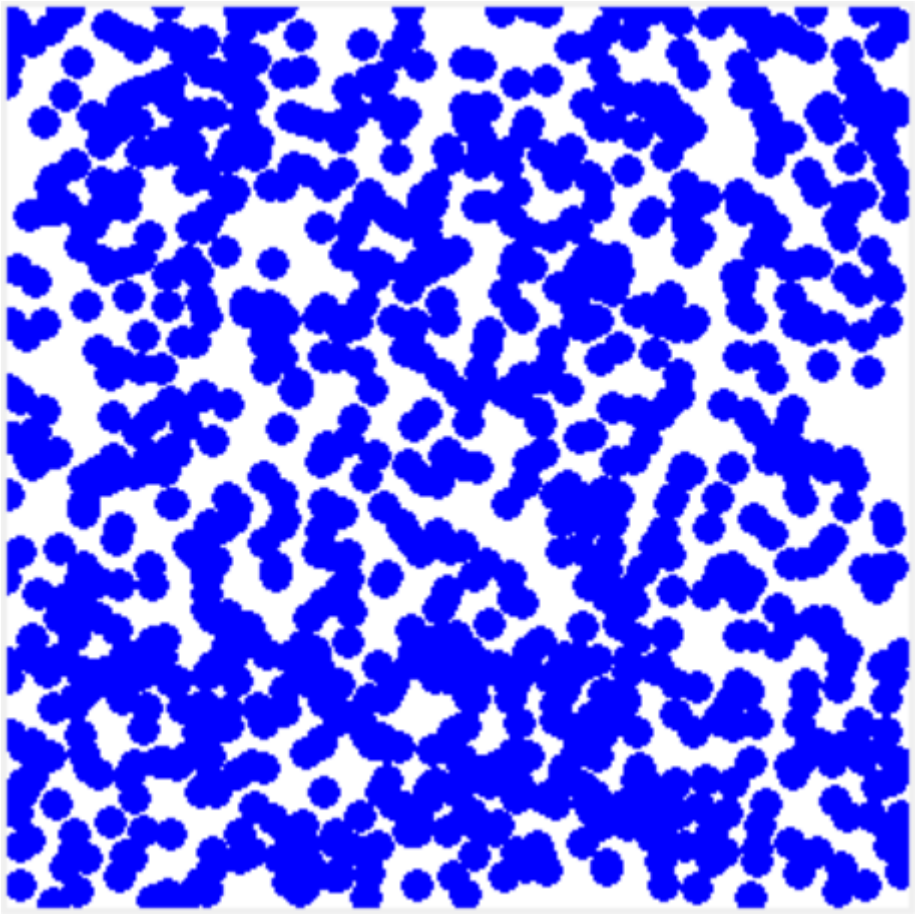
```
import tkinter
import random

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

for i in range(100):
    x = random.randint(1, 350)
    y = random.randint(1, 230)
    a = random.randint(1, 30)
    if a <= 10:
        farba = 'red'
    elif a <= 20:
        farba = 'blue'
    else:
        farba = ''
    canvas.create_rectangle(x, y, x+a, y+a, fill=farba)

tkinter.mainloop()
```

Aj v tomto programe využívame sériu `if` - `elif` - `else`, rovnako ako pri prepočítavaní bodov na známky v predchádzajúcom príklade. Dostávame



Pokračujme s grafickým programom, ktorý na náhodné pozície nakreslí 1000 malých krúžkov:

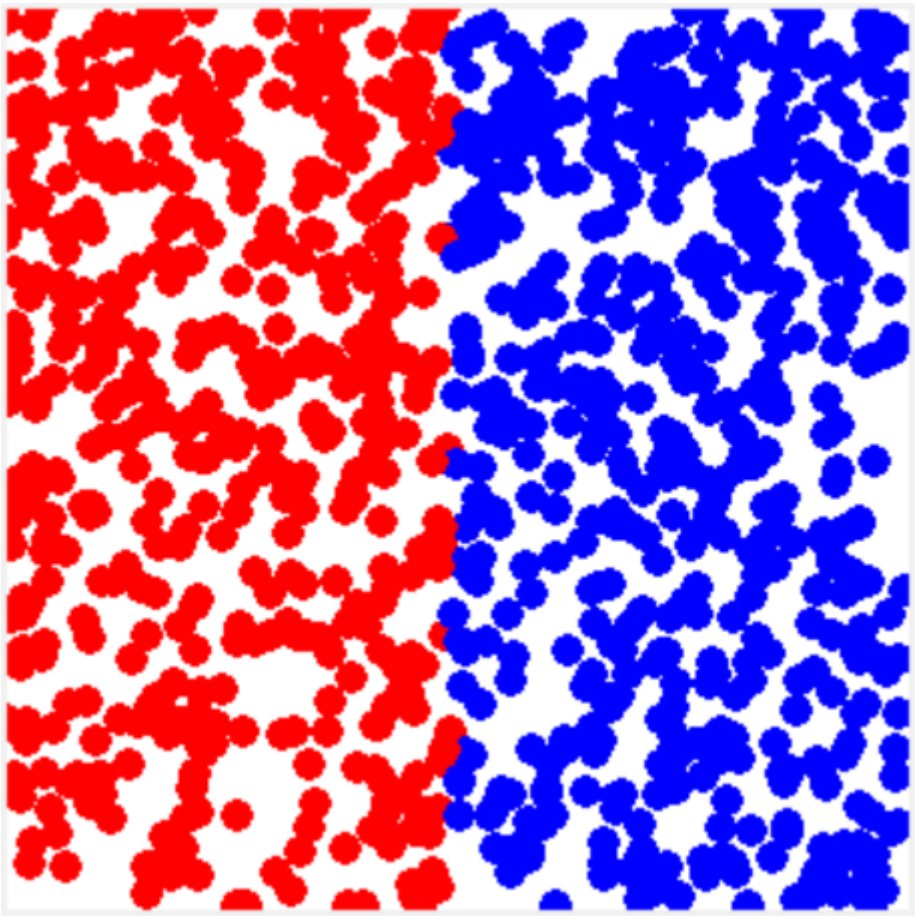
```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    farba = 'blue'
    canvas.create_oval(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



Všimnite si, že krúžky sú bez obrysu (doteraz mali kruhy čierny tenký obrys). Je to vďaka pomenovanému parametru `width=0`, ktorým určujeme hrúbku obrysu, teda teraz bude `0`.

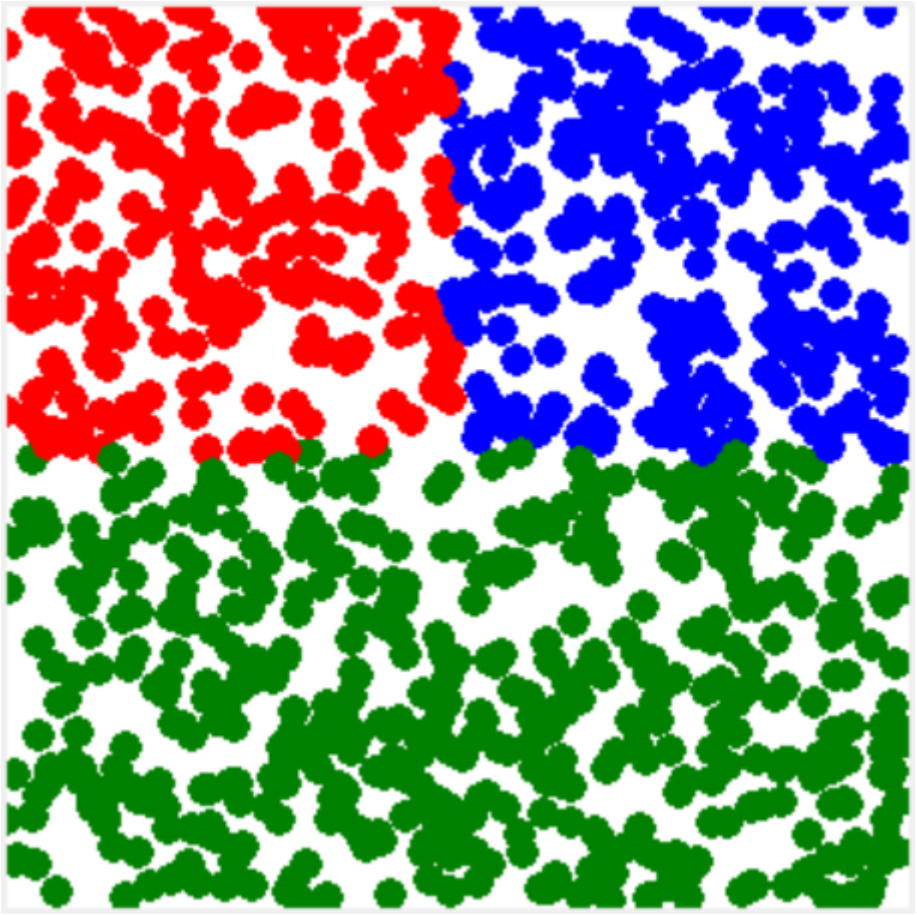
Niektoré z týchto krúžkov zafarbíme tak, že tie z nich, ktoré sú v ľavej polovici plochy budú červené a zvyšné v pravej polovici (teda `else` vetva) budú modré:

```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    if x < 150:
        farba = 'red'
    else:
        farba = 'blue'
    canvas.create_oval(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



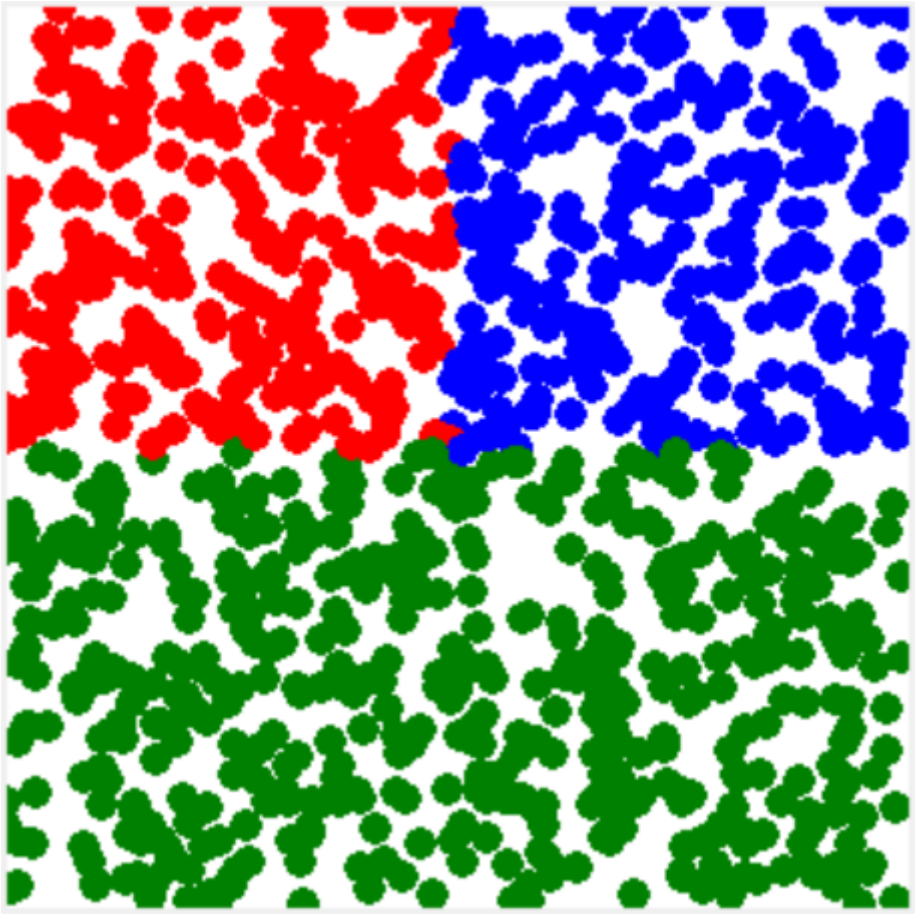
Skúsme pridať ešte jednu podmienku: všetky bodky v spodnej polovici (`y > 150`) budú zelené, takže rozdelenie na červené a modré bude len v hornej polovici. Jedno z možných riešení:

```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    if y < 150:
        if x < 150:
            farba = 'red'
        else:
            farba = 'blue'
    else:
        farba = 'green'
    canvas.create_oval(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



Podobne, ako sme to robili s intervalmi bodov pre rôzne známky, môžeme aj toto riešenie zapísať tak, že použijeme komplexnejšiu podmienku:

```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    if y < 150 and x < 150:
        farba = 'red'
    elif y < 150 and x >= 150:
        farba = 'blue'
    else:
        farba = 'green'
    canvas.create_oval(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



Podmienky v Pythone môžu obsahovať logické operácie a tieto majú obvyklý význam z matematiky:

- podmienka1 `and` podmienka2 … (a súčasne) znamená, že musia platiť obe podmienky
- podmienka1 `or` podmienka2 … (alebo) znamená, že musí platiť aspoň jedna z podmienok
- `not` podmienka … (neplatí) znamená, že daná podmienka neplatí

Otestovať rôzne kombinácie podmienok môžeme, napríklad takto:

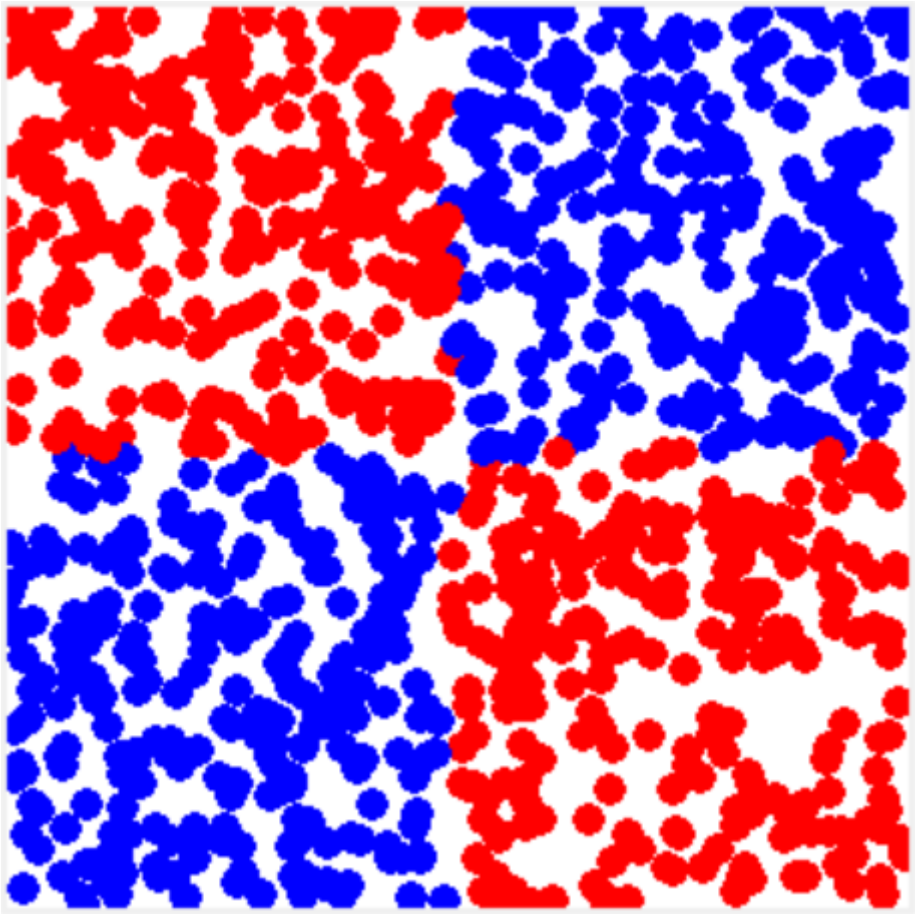
```
>>> a = 10
>>> b = 7
>>> a < b
False
>>> a >= b + 3
True
>>> b < a < 2 * b
True
>>> a != 7 and b == a - 3
True
>>> a == 7 or b == 10
False
>>> not a == b          # to isté ako  a != b
True
>>> 1 == '1'
False
>>> 1 < '2'             # nemôžeme takto porovnaváť čísla a znaky
...
TypeError: unorderable types: int() < str()
```

Všimnite si, že podmienky, ktoré platia, majú hodnotu `True` a ktoré neplatia, majú `False` - sú to dve špeciálne hodnoty, ktoré Python používa ako výsledky porovnávania - tzv. logických výrazov. Sú **logického typu**, tzv. `bool`. Môžeme to skontrolovať:

```
>>> type(1 + 2)
<class 'int'>
>>> type(1 / 2)
<class 'float'>
>>> type('12')
<class 'str'>
>>> type(1 < 2)
<class 'bool'>
```

Aj logické hodnoty môžeme medzi sebou porovnávať. Zamyslite sa nad takouto podmienkou v `if`:

```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    if (y < 150) == (x < 150):
        farba = 'red'
    else:
        farba = 'blue'
    canvas.create_oval(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



Logické operácie

Pozrime sa podrobnejšie na logické operácie `and`, `or` a `not`. Tieto operácie samozrejme fungujú pre logické hodnoty `True` a `False`.

Logický súčin **a súčasne**

A	B	A and B
`False`	`False`	`False`
`True`	`False`	`False`
`False`	`True`	`False`
`True`	`True`	`True`

Logický súčet **alebo**

A	B	A or B
`False`	`False`	`False`

Logický súčet **alebo**

A	B	A or B
`True`	`False`	`True`
`False`	`True`	`True`
`True`	`True`	`True`

Negácia **neplatí**

A	not A
`False`	`True`
`True`	`False`

Logické operácie fungujú nielen pre logický typ, ale aj pre skoro všetky ďalšie typy. V tomto prípade Python pre **každý typ** definuje prípady, ktoré on (Python) chápe ako `False` a zrejme všetky ostatné hodnoty tohto typu chápe ako `True`. Ukážme prípady pre doteraz známe typy, ktoré označujú logickú hodnotu `False`:

typ	`False`	`True`
`int`	`x == 0`	`x != 0`
`float`	`x == 0.0`	`x != 0.0`
`str`	`x == ''`	`x != ''`

Tejto tabuľke budeme rozumieť takto: keď bude Python niekde očakávať logickú hodnotu (napríklad v príkaze `if`) a bude tam namiesto toho celé číslo, tak, ak má hodnotu **0**, pochopí to ako `False` a ak má hodnotu rôznu od **0**, bude to pre neho znamenať `True`. Podobne aj s reťazcami: ak ako podmienku `if` uvedieme znakový reťazec, tento bude reprezentovať `False`, ak je prázdny a `True`, ak je neprázdny.

Napríklad:

```
pocet = int(input('zadaj: '))
if pocet:
    print('pocet je rôzny od 0')
else:
    print('pocet je 0')
meno = input('zadaj: ')
if meno:
    print('meno nie je prázdny reťazec')
else:
    print('meno je prázdny reťazec')
```

Logické operácie `and`, `or` a `not` majú v skutočnosti tiež trochu rozšírenú interpretáciu:

operácia: prvý `and` druhý

- ak **prvý** nie je `False`, tak
  - výsledkom je **druhý**
- inak (teda **prvý** je `False`)
  - výsledkom je **prvý**

Môžeme upraviť aj tabuľku pravdivostných hodnôt:

A	B	A and B
`False`	*hocičo*	**A**
`True`	*hocičo*	**B**

operácia: prvý `or` druhý

- ak **prvý** nie je `False`, tak
  - výsledkom je **prvý**
- inak (teda **prvý** je `False`)
  - výsledkom je **druhý**

Tabuľka:

A	B	A or B
`False`	*hocičo*	**B**
`True`	*hocičo*	**A**

operácia: `not` prvý

- ak **prvý** nie je `False`, tak
  - výsledkom je `False`
- inak
  - výsledkom je `True`

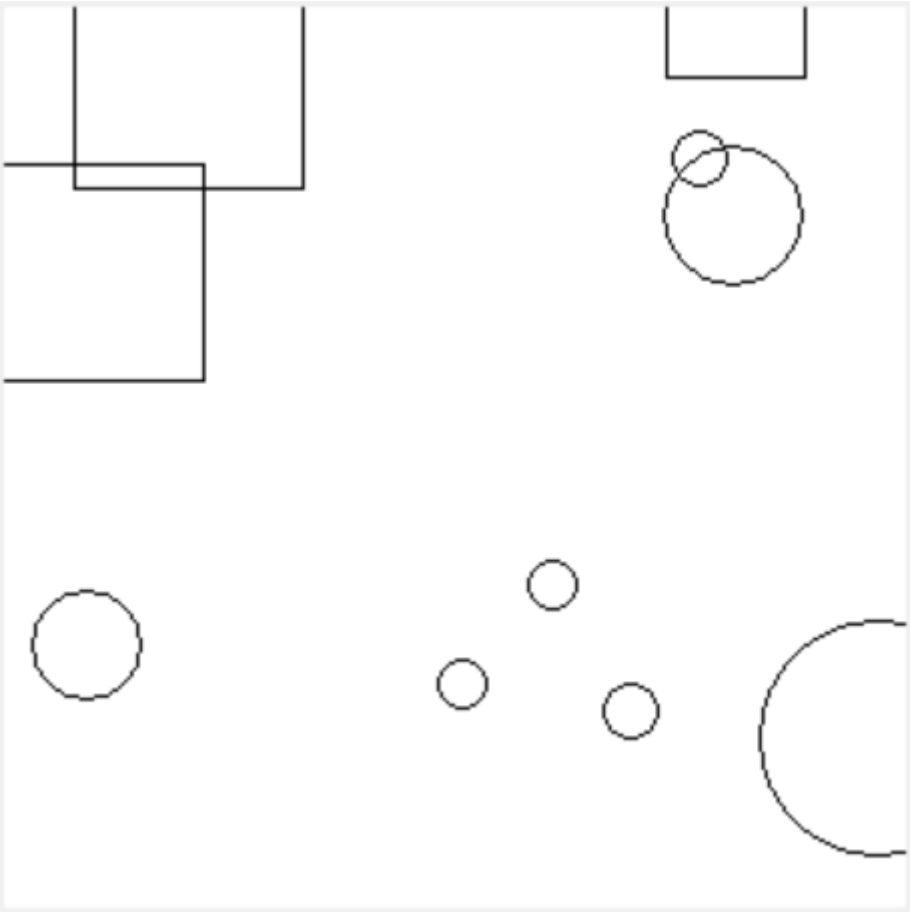
Napríklad:

```
>>> 1 + 2 and 3 + 4       # keďže 1+2 nie je False, výsledkom je 3+4
7
>>> 'ahoj' or 'Python'    # keďže 'ahoj' nie je False, výsledkom je 'ahoj'
'ahoj'
>>> '' or 'Python'        # keďže '' je False, výsledkom je 'Python'
'Python'
>>> 3 < 4 and 'kuk'       # keďže 3<4 nie je False, výsledkom je 'kuk'
'kuk'
>>> False or True         # keďže False je False, výsledkom je True
True
>>> 'False' or 'True'     # keďže 'False' nie je False, výsledkom je 'False'
'False'
```

Podmienený príkaz sa často používa pri náhodnom rozhodovaní. Napríklad, hádžeme mincou (náhodné hodnoty 0 a 1) a ak padne 1, kreslíme náhodnú kružnicu, inak nakreslíme náhodný štvorec. Toto opakujeme 10-krát:

```
import tkinter
import random
```

```
canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()

for i in range(10):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    a = random.randint(5, 50)

    if random.randrange(2):          # t.j. random.randrange(2) != 0
        canvas.create_oval(x-a, y-a, x+a, y+a)
    else:
        canvas.create_rectangle(x-a, y-a, x+a, y+a)

tkinter.mainloop()
```

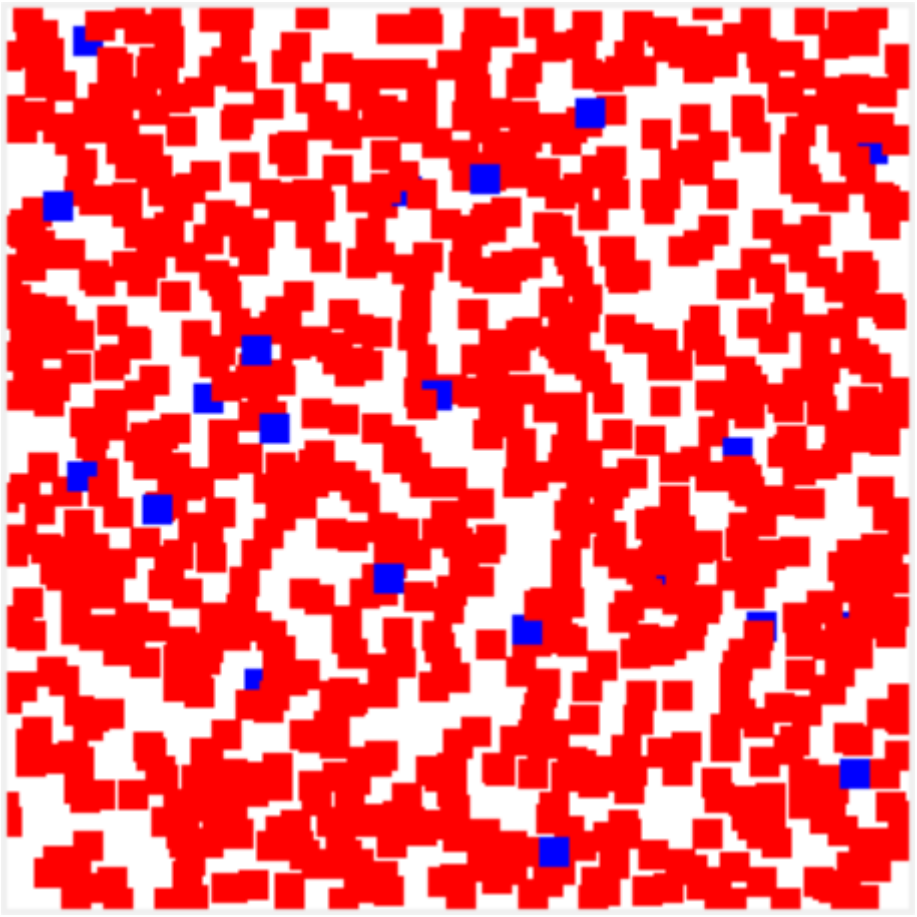


Približne rovnaké výsledky by sme dostali, ak by sme hádzali kockou so 6 možnosťami (`random.randrange(1, 7)`) a pre čísla 1, 2, 3 by sme kreslili kružnicu inak štvorec.

Túto ideu môžeme využiť aj pre takúto úlohu: vygenerujte 1000 farebných štvorčekov - modré a červené, pričom ich pomer je **1:50**, t.j. na 50 červených štvorčekov pripadne približne 1 modrý:

```
import tkinter
import random

canvas = tkinter.Canvas(bg='white', width=300, height=300)
canvas.pack()
```

```
for i in range(1000):
    x = random.randint(1, 300)
    y = random.randint(1, 300)
    if random.randrange(50):              # t.j. random.randrange(50) != 0
        farba = 'red'
    else:
        farba = 'blue'
    canvas.create_rectangle(x-5, y-5, x+5, y+5, fill=farba, width=0)

tkinter.mainloop()
```



Delitele čísel

Nasledovný príklad zisťuje, akých deliteľov má zadané číslo:

```
cislo = int(input('Zadaj číslo: '))
pocet = 0
print('delitele:', end=' ')
for delitel in range(1, cislo+1):
    if cislo % delitel == 0:        # mohli by sme zapísať aj  if not cislo % deli
tel:
        pocet += 1
        print(delitel, end=' ')
print()
print('počet deliteľov:', pocet)
```

Výstup môže byť napríklad takýto:

```
Zadaj číslo: 100
delitele: 1 2 4 5 10 20 25 50 100
počet deliteľov: 9
```

Malou modifikáciou tejto úlohy vieme vytvoriť ďalšie dva programy. Prvý zisťuje, či je zadané číslo prvočíslo:

```
cislo = int(input('Zadaj číslo: '))
pocet = 0
for delitel in range(1, cislo + 1):
    if cislo % delitel == 0:
        pocet += 1
if pocet == 2:
    print(cislo, 'je prvočíslo')
else:
    print(cislo, 'nie je prvočíslo')
```

Po spustení, napríklad:

```
Zadaj číslo: 101
101 je prvočíslo
```

Ďalší program zisťuje, či je nejaké číslo dokonalé, t.j. súčet všetkých deliteľov menších ako samotné číslo sa rovná samotnému číslu. Na základe tohto nájde (postupne preverí) všetky dokonalé čísla do 10000:

```
print('dokonalé čísla do 10000 sú', end=' ')
for cislo in range(1, 10001):
    sucet = 0
    for delitel in range(1, cislo):
        if cislo % delitel == 0:
            sucet += delitel
    if sucet == cislo:
        print(cislo, end=', ')
print()
print('=== viac ich už nie je ===')
```

Program vypíše:

```
dokonalé čísla do 10000 sú 6, 28, 496, 8128,
=== viac ich už nie je ===
```

Vráťme sa k programu, ktorý zisťuje, či je nejaké číslo prvočíslo:

```
cislo = int(input('Zadaj číslo: '))
pocet = 0
for delitel in range(1, cislo + 1):
    if cislo % delitel == 0:
        pocet += 1
if pocet == 2:
    print(cislo, 'je prvočíslo')
```

Tento for-cyklus prejde vždy `cislo`-krát prechodov - bude kontrolovať, či je `cislo` deliteľné premennou `delitel`. Asi by nám ale stačilo zistiť, či existuje aspoň jeden deliteľ, ktorý je väčší ako **1** a menší ako **cislo**. Ak taký nájdeme (t.j. platí `cislo % delitel == 0`), cyklus už ďalej nemusí preverovať zvyšné delitele. Predstavme si, že chceme zisťovať, či číslo **1000000** je prvočíslo. Tento náš program už pri treťom prechode cyklu (keď `delitel` má hodnotu 3) „vie“, že `pocet` je viac ako 2, a teda to určite nebude prvočíslo. Úplne zbytočne 999997-krát zisťuje, či nejaký deliteľ delí alebo nedelí naše `cislo`. V tomto prípade by sa **vykonávanie cyklu mohlo prerušiť** a dostali by sme oveľa rýchlejšie rovnako správny výsledok. Na prerušovanie for-cyklu využijeme špeciálny príkaz **break**, ktorý ale môžeme použiť len v tele cyklu.

Prerušenie cyklu pomocou break

Príkaz `break` v tele cyklu spôsobí ukončenie cyklu, t.j. vykonávanie príkazov cyklu sa v tomto momente preruší a pokračuje sa až za prvým príkazom za cyklom. Premenná cyklu bude mať potom hodnotu naposledy vykonávaného prechodu. Príkaz `break` sa musí použiť v nejakom vnorenom podmienenom príkaze `if`. Napríklad:

```
for premenná in ...:
    príkazy1
    if podmienka:
        break
    príkazy2
príkazy_za_cyklom
```

V každom prechode cyklu sa najprv vykonajú `príkazy1`. Potom odkontroluje `podmienka`: ak je pravdivá, vykonávanie cyklu končí a pokračuje sa na `príkazy_za_cyklom`. Ak je podmienka nepravdivá (neplatí), cyklus pokračuje vykonávaním `príkazy2` a prípadnými ďalšími prechodmi cyklu.

Vylepšime cyklus na zisťovanie prvočísel s prerušením pomocou `break`:

```
cislo = int(input('Zadaj číslo: '))
pocet = 0
for delitel in range(1, cislo + 1):
    if cislo % delitel == 0:
        pocet += 1
        if pocet > 2:
            break
if pocet == 2:
    print(cislo, 'je prvočíslo')
else:
    print(cislo, 'nie je prvočíslo')
```

Otestujte zisťovaním, či je `1000000000` prvočíslo. Pomocou predchádzajúcej verzie programu (bez `break`) by sme sa výsledku nedočkali, ale teraz sa dozvieme veľmi rýchlo, že:

```
Zadaj číslo: 1000000000
1000000000 nie je prvočíslo
```

Všimnite si, že v tomto prípade vôbec nepotrebujeme premennú `pocet`, ktorá počítala počet deliteľov. Podľa hodnoty premennej `delitel` po skončení cyklu, vieme presne povedať, či to bolo prvočíslo alebo nie. Upravme:

```
cislo = int(input('Zadaj číslo: '))
for delitel in range(2, cislo + 1):
    if cislo % delitel == 0:
        break
if cislo > 1 and delitel == cislo:
    print(cislo, 'je prvočíslo')
else:
    print(cislo, 'nie je prvočíslo')
```

Toto môžeme zapísať ešte krajšie pomocou pomocnej premennej `prvocislo`, v ktorej si budeme pamätať, či sme pre dané číslo ešte nenašli žiadneho deliteľa (`True`) alebo sme už jedného našli (`False`), a teda to určite prvočíslo nebude:

```
cislo = int(input('Zadaj číslo: '))
prvocislo = True
for delitel in range(2, cislo):
    if cislo % delitel == 0:
        prvocislo = False
        break
if prvocislo and cislo > 1:
    print(cislo, 'je prvočíslo')
else:
    print(cislo, 'nie je prvočíslo')
```

V tomto prípade ale for-cyklus nemôže ísť od `1` do `cislo`, ale pôjde od `2` do `cislo-1`.

Pripomeňme si ešte úlohu zo začiatku prednášky, v ktorej sa z počtu bodov zisťuje výsledné hodnotenie. Prepíšme teraz sériu príkazov `if` do for-cyklu, v ktorom bude jediný `if` a prerušenie cyklu pomocou `break`:

```
body = int(input('Zadaj získaný počet bodov: '))
hranica = 90
for znamka in 'ABCDEF':
    if body >= hranica:
        break
    hranica -= 10

if znamka != 'F':
    print('za', body, 'bodov získavaš známku', znamka)
else:
    print('za', body, 'bodov si nevyhovel a máš známku Fx')
```

V tomto riešení vidíme, že vo for-cykle sa postupne prechádzajú všetky hranice (hodnoty `90`, `80`, `70`, …) a pri prvej, ktorá vyhovuje (`body >= hranica`), sa z cyklu vyskakuje. V tom prípade bude v premennej cyklu `znamka` zodpovedajúce hodnotenie. Všimnite si, že v cykle pre známku `F` vyjde hranica `40`. To znamená, že pre `body` od `40` do `49` sa vyskočí z cyklu (pomocou `break`) s hodnotou `F` v premennej `znamka` a pre `body` menšie ako `40` cyklus normálne skončí a v premennej `znamka` bude stále hodnota `F`.

Podmienený cyklus

V Pythone existuje konštrukcia cyklu, ktorá opakuje vykonávanie postupnosti príkazov v závislosti od nejakej podmienky:

```
while podmienka:              # opakuj príkazy, kým platí podmienka
    prikaz
    prikaz
    ...
```

Vidíme podobnosť s podmieneným príkazom `if` - vetvením. Tento nový príkaz postupne:

- zistí hodnotu podmienky, ktorá je zapísaná za slovom `while`
- ak má táto podmienka hodnotu `False`, blok príkazov, ktorý je telom cyklu, sa preskočí a pokračuje sa na nasledovnom príkaze za celým while-cyklom (podobne ako v príkaze `if` bez vetvy `else`), hovoríme, že sa ukončilo vykonávanie cyklu
- ak má podmienka hodnotu `True`, **vykonajú sa všetky príkazy** v tele cyklu (v odsunutom bloku príkazov)
- a znovu sa testuje podmienka za slovom `while`, t.j. celé sa to opakuje

Najprv zapíšeme pomocou tohto cyklu, to čo už vieme pomocou for-cyklu:

```
for i in range(1, 21):
    print(i, i*i)
```

Vypíše tabuľku druhých mocnín čísel od 1 do 20. Prepis na cyklus `while` znamená, že zostavíme podmienku, ktorá bude testovať, napríklad premennú `i`: tá nesmie byť väčšia ako 20. Samozrejme, že už pred prvou kontrolou premennej `i` v podmienke cyklu `while`, musí mať nejakú hodnotu:

```
i = 1
while i < 21:
    print(i, i*i)
    i += 1
```

V cykle sa vykoná `print()` a zvýši sa hodnota premennej `i` o jedna.

`while`-cykly sa ale častejšie používajú vtedy, keď zápis pomocou `for`-cyklu je príliš komplikovaný, alebo sa ani urobiť nedá.

Ukážeme to na programe, ktorý bude do jedného radu tesne vedľa seba kresliť stále sa zväčšujúce štvorce postupne so stranami 10, 20, 30, … Pritom bude dávať pozor, aby naposledy nakreslený štvorec „nevypadol“ z plochy - teda chceme skončiť skôr, ako by sme nakreslili štvorec, ktorý sa už celý nezmestí do grafickej plochy. Štvorce so stranou `a` budeme kresliť takto:

```
canvas.create_rectangle(x, 200, x+a, 200-a)
```

vďaka čomu, všetky ležia na jednej priamke (`y = 200`). Keď teraz budeme posúvať x-ovú súradnicu vždy o veľkosť nakresleného štvorca, ďalší bude ležať tesne vedľa neho.

Program pomocou while-cyklu zapíšeme takto:

```
import tkinter

sirka = int(input('šírka plochy: '))
```

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

x = 5
a = 10
while x + a < sirka:
    canvas.create_rectangle(x, 200, x+a, 200-a, fill='white')
    x += a
    a += 10
# príkazy za cyklom

tkinter.mainloop()
```

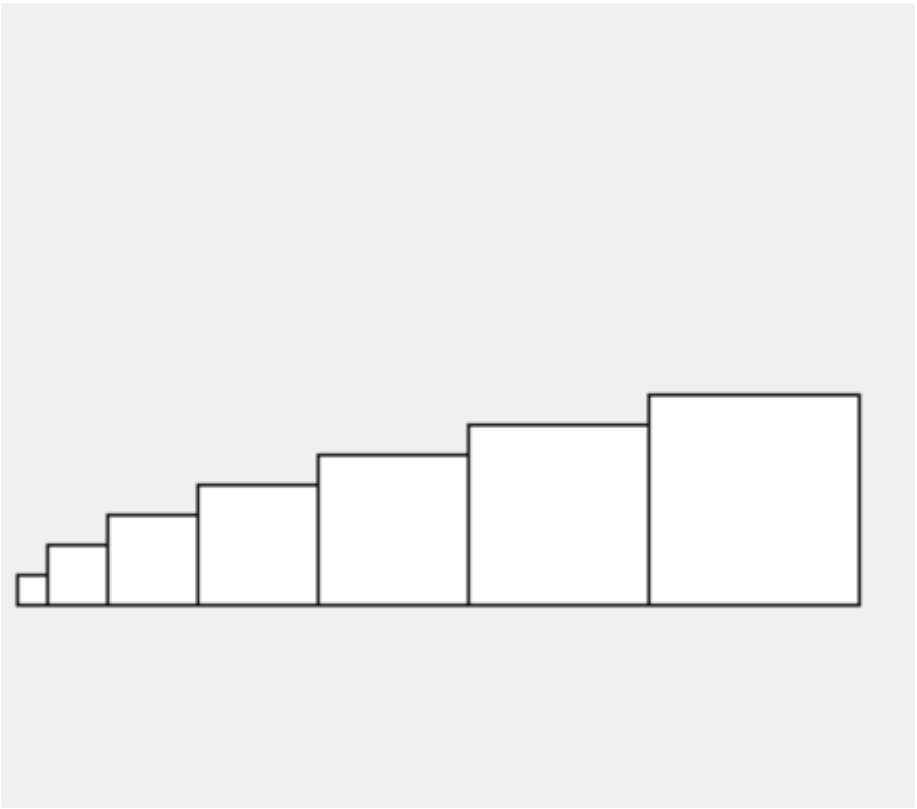
Program pracuje korektne pre rôzne šírky grafickej plochy. Ak zväčšovanie strany štvorca `a += 10` nahradíme `a = 2 * a`, program bude pracovať aj s takto zväčšovanými štvorcami (strany budú postupne 10, 20, 40, 80, …).

Zhrňme, ako funguje tento typ cyklu:

1. vyhodnotí sa podmienka `x + a < sirka`, t.j. pravý okraj štvorca, ktorý práve chceme nakresliť, sa ešte celý zmestí do grafickej plochy
2. ak je podmienka splnená (pravdivá), postupne sa vykonajú všetky príkazy, t.j. nakreslí sa ďalší štvorec so stranou `a` a potom sa posunie `x` (ľavý okraj budúceho štvorca) o veľkosť práve nakresleného štvorca `a` a tiež sa ešte zmení veľkosť budúceho štvorca `a` o 10
3. po vykonaní tela cyklu sa pokračuje v 1. kroku, t.j. opäť sa vyhodnotí podmienka
4. ak podmienka nie je splnená (nepravda), cyklus končí a ďalej sa pokračuje v príkazoch za cyklom

Uvedomte si, že podmienka nehovorí, kedy má cyklus skončiť, ale naopak - kým podmienka platí, vykonávajú sa všetky príkazy v tele cyklu.

Pre šírku grafickej plochy 300 dostávame takýto obrázok:



Konkrétne tento program s while-cyklom vieme jednoducho prepísať pomocou for-cyklu a `break`:

```
import tkinter

sirka = int(input('šírka plochy: '))

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

x = 5
for a in range(10, sirka, 10):
    if x + a >= sirka:
        break
    canvas.create_rectangle(x, 200, x+a, 200-a, fill='white')
    x += a

tkinter.mainloop()
```

Vyššie sme zostavili program, ktorý zisťoval, či je zadané číslo prvočíslo. Použili sme for-cyklus, v ktorom sme zadané číslo postupne delili všetkými číslami, ktoré sú menšie ako samotné číslo. Zistili sme, že na zisťovanie prvočísla nepotrebujeme skutočný počet deliteľov, ale malo by nám stačiť zistenie, či existuje aspoň jeden deliteľ. Keď sa vyskytne prvý deliteľ (t.j. platí `cislo % delitel != 0`), cyklus môžeme ukončiť a vyhlásiť, že číslo nie je prvočíslo. Ak ani jedno číslo nie je deliteľom nášho čísla, hodnota premennej `delitel` dosiahne `cislo` a to je situácia, keď cyklus tiež skončí (t.j. keď `delitel == cislo`, našli sme prvočíslo). Zapíšeme to while-cyklom:

```
cislo = int(input('Zadaj číslo: '))
delitel = 2
while delitel < cislo and cislo % delitel != 0:
```

```
    delitel = delitel + 1

if delitel == cislo:
    print(cislo, 'je prvočíslo')
else:
    print(cislo, 'nie je prvočíslo')
```

Do podmienky while-cyklu sme pridali novú časť. Operátor `and` tu má ten význam, že na to, aby sa cyklus opakoval, musia byť splnené obe časti. Uvedomte si, že **cyklus skončí vtedy**, keď prestane platiť zadaná podmienka, t.j. negácia podmienky (matematicky to upravíme):

- `not (delitel < cislo and cislo % delitel != 0)`
- `not delitel < cislo or not cislo % delitel != 0`
- `delitel >= cislo or cislo % delitel == 0`

while-cyklus teda skončí vtedy, keď `delitel >= cislo`, **alebo** `cislo % delitel == 0` (teda, že deliteľ by bol už zbytočne veľký alebo našli sme hodnotu, ktorá delí naše číslo).

Zisťovanie druhej odmocniny

Ukážeme, ako zistíme druhú odmocninu čísla aj bez volania funkcie `math.sqrt(x)`, resp. umocňovaním na jednu polovicu `x**0.5`.

Prvé riešenie:

```
cislo = float(input('zadaj číslo:'))

x = 0
while x**2 < cislo:
    x += 1

print('odmocnina', cislo, 'je', x)
```

Takto nájdené riešenie môže byť dosť nepresné, lebo `x` zvyšujeme o 1, takže, napríklad odmocninu z `26` vypočíta ako `6`. Skúsme zjemniť krok, o ktorý sa mení hľadané `x`:

```
cislo = float(input('zadaj číslo:'))

x = 0
while x**2 < cislo:
    x += 0.001

print('odmocnina', cislo, 'je', x)
```

Teraz dáva program lepšie výsledky, ale pre väčšiu zadanú hodnotu mu to trvá citeľne dlhšie - skúste zadať, napríklad `10000000`. Keďže mu vyšiel výsledok približne `3162.278` a dopracoval sa k nemu postupným pripočítavaním čísla `0.001` k štartovému 1, musel urobiť vyše 3 miliónov pripočítaní a tiež toľkokrát testov vo while-cykle (podmienky `x**2 < cislo`). Pre toto je takýto algoritmus nepoužiteľne pomalý.

Využijeme inú ideu:

- zvolíme si interval, v ktorom sa určite bude nachádzať hľadaný výsledok (hľadaná odmocnina), napríklad nech je to interval `<1, cislo>` (pre čísla väčšie ako 1 je aj odmocnina väčšia ako 1 a určite je menšia ako samotne `cislo`)
- ako `x` (prvý odhad našej hľadanej odmocniny) zvolíme stred tohto intervalu
- zistíme, či je druhá mocnina tohto `x` väčšia ako zadané `cislo` alebo menšia
- ak je väčšia (`x` je už zbytočne veľké), tak upravíme predpokladaný interval, tak že jeho hornú hranicu zmeníme na `x`
- ak je ale menšia, upravíme dolnú hranicu intervalu na `x`
- tým sa nám interval zmenšil na polovicu
- toto celé opakujeme, kým už nie je nájdené `x` dostatočne blízko k hľadanému výsledku, t.j. či sa nelíši od výsledku menej ako zvolený rozdiel (epsilon)

Zapíšme to:

```
cislo = float(input('zadaj číslo:'))

od = 0
do = cislo

x = (od + do) / 2

pocet = 0
while abs(x**2 - cislo) > 0.001:
    if x**2 > cislo:
        do = x
    else:
        od = x
    x = (od + do) / 2
    pocet += 1

print('druhá odmocnina', cislo, 'je', x)
print('počet prechodov while-cyklom bol', pocet)
```

Ak spustíme program pre `10000000` dostávame:

```
zadaj číslo:10000000
druhá odmocnina 10000000 je 3162.2776600670477
počet prechodov while-cyklom bol 41
```

čo je výrazné zlepšenie oproti predchádzajúcemu riešeniu, keď prechodov while-cyklom (hoci jednoduchších) bolo vyše 3 milióny.

Nekonečný cyklus

Cyklus s podmienkou, ktorá má stále hodnotu `True`, bude nekonečný. Napríklad:

```
i = 0
while i < 10:
    i -= 1
```

Nikdy neskončí, lebo premenná `i` bude stále menšia ako 10. Takéto výpočty môžeme prerušiť stlačením klávesov **Ctrl/C**.

Aj nasledovný cyklus je úmyselne nekonečný:

```
while 1:
    pass
```

Pripomíname, že príkaz `pass` je **prázdny príkaz**, ktorý nerobí nič. V tomto príklade `pass` označuje prázdne telo cyklu, a teda tento cyklus bude nikdy nekončiaci.

Už vieme, že príkaz `break` môžeme použiť v tele for-cyklu a vtedy sa zvyšok cyklu nevykoná. Nasledovný príklad ilustruje použitie `break` aj vo while-cykle:

```
sucet = 0
while True:
    retazec = input('zadaj číslo: ')
    if retazec == '':
        break
    sucet += int(retazec)
print('súčet prečítaných čísel =', sucet)
```

V tomto príklade sa čítajú čísla zo vstupu, kým nezadáme prázdny reťazec: vtedy cyklus končí a program vypíše súčet prečítaných čísel. Napríklad:

```
zadaj číslo: 5
zadaj číslo: 17
zadaj číslo: 2
zadaj číslo:
súčet prečítaných čísel = 24
```

Cvičenia

L.I.S.T.

- riešenia **aspoň 10 úloh** odovzdávaj na úlohový server https://list.fmph.uniba.sk/
- používaj len konštrukcie z doterajších prednášok (žiadne nové funkcie ani zoznamy)
- pozri si Riešenie úloh 4. cvičenia

1. Športovec prvý deň prebehol `x` kilometrov. Každý ďalší deň prebehol o 10% viac ako v predchádzajúci deň. Napíš program, ktorý pre dané `y` zistí, v ktorý deň športovec dokopy prebehne aspoň `y` kilometrov. Napríklad, po spustení môžeš dostať:

```
zadaj km pre prvý deň: 10
zadaj cieľové km: 20
na 9. deň prebehne 21.44 km

zadaj km pre prvý deň: 100
zadaj cieľové km: 121
na 3. deň prebehne 121.00 km
```

2. Budeme konštruovať takúto postupnosť celých čísel:

- začneme zadaným číslom `n`
- ak je párne, vydelíme ho `2`
- inak sa vynásobí `3` a pripočíta `1`
- toto sa opakuje, kým nedostaneme číslo `1`

Napíš program, ktorý pre dané štartové číslo vypíše takto skonštruovanú postupnosť. Napríklad:

```
zadaj číslo: 44
44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

3. Napíš program, ktorý číta desatinné čísla zo vstupu a keď zadáme `0`, čítanie čísel zo vstupu skončí a program vypíše súčet všetkých týchto čísel. Použi while-cyklus. Napríklad po spustení:

```
4. zadaj 1. číslo: 17
5. zadaj 2. číslo: 3.14
6. zadaj 3. číslo: -9.8
7. zadaj 4. číslo: 6
8. zadaj 5. číslo: 0
9. súčet všetkých prečítaných čísel je 16.34
```

4. Napíš program, ktorý zadané číslo **rozloží na prvočinitele** (vyjadrí ho ako súčin prvočísel). Tento rozklad zapíš v tvare rovnosti s násobením:

```
5. zadaj číslo: 60
6. 60 = 2 * 2 * 3 * 5

7. zadaj číslo: 1001
8. 1001 = 7 * 11 * 13

9. zadaj číslo: 37
10.37 = 37
```

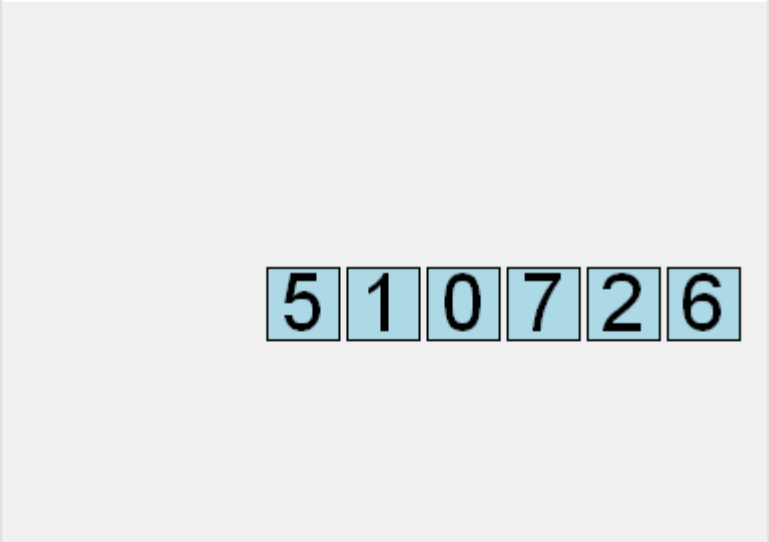
V programe bude while-cyklus, v ktorom budeš postupne skúšať deliť zadané číslo rôznymi deliteľmi: ak je číslo týmto deliteľom deliteľné, tak ho vypíšeš a samotné číslo vydelíš týmto deliteľom, inak deliteľ zvýšiš o 1 a celé sa to opakuje.

5. Napíš program, ktorý vypíše cifry zadaného čísla postupným delením desiatimi, teda vo while-cykle vypíšeš poslednú cifru (číslo % 10) a pritom ešte samotné číslo vydelíš 10. Súčasne každú túto cifru pripočítaš do počítadla `cs` (ciferný súčet). Môžeš dostať takýto výstup:

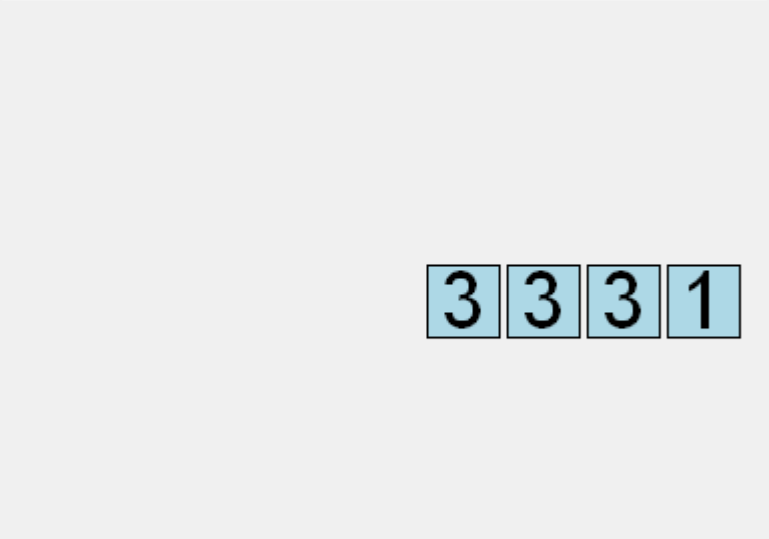
```
6. zadaj číslo: 4132
7. 2
8. 3
9. 1
10.4
11.ciferný súčet = 10
```

Všimni si, že cifry sú vypísané v opačnom poradí ako sú v zadanom čísle.

6. Využi while-cyklus z predchádzajúcej úlohy a vypíš cifry do grafickej plochy (zrejme sprava do ľava). Program jednotlivé cifry vypíše do farebných štvorcov. Napríklad, pre číslo `510726` by si mal dostať:



7. Uprav predchádzajúci program tak, aby sa číslo vypísalo v osmičkovej sústave (zrejme namiesto delenia `10` budeš deliť `8`). Pre číslo `1753` by si mal dostať:



Tento výsledok môžeš porovnať s volaním štandardnej funkcie `oct` (resp. pomocou formátovacej šablóny):

```
>>> oct(1753)
'0o3331'
>>> f'{1753:o}'
'3331'
```

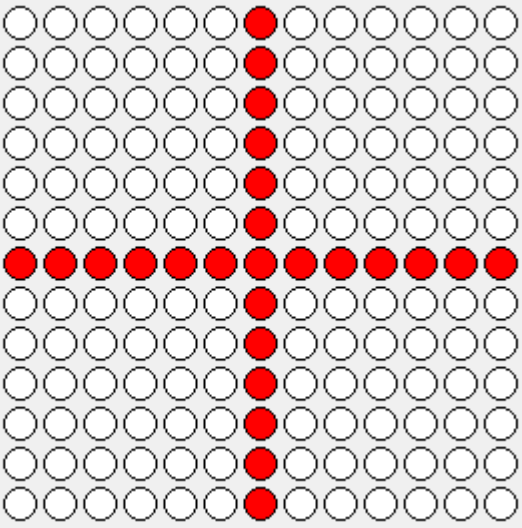
Vyskúšaj upraviť tento program tak, aby vypisoval cifry v dvojkovej sústave. Aj tu môžeš pre kontrolu využiť štandardnú funkciu `bin` (resp. formátovaciu šablónu):

```
>>> bin(479)
'0b111011111'
>>> f'{479:b}'
'111011111'
```

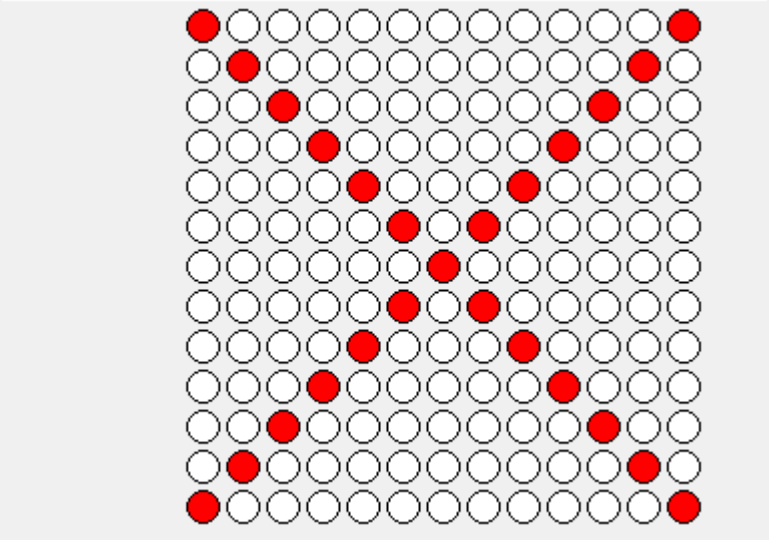
8. Nasledovný program vykresľuje farebné krúžky v štvorcovej sieti `n` x `n` a zafarbuje ich podľa podmienky v príkaze `if`:

```
9.  import tkinter
10.
11. canvas = tkinter.Canvas()
12. canvas.pack()
13.
14. n = 13
15. for i in range(n):
16.     for j in range(n):
17.         x = j * 20 + 100
18.         y = i * 20 + 12
19.         if i == 5:
20.             farba = 'red'
21.         else:
22.             farba = 'white'
23.         canvas.create_oval(x-8, y-8, x+8, y+8, fill=farba)
24.
25. tkinter.mainloop()
```

A. Zmeň túto podmienku tak, aby sa nakreslil obrázok, v ktorom sa zafarbí stredný rad a stredný stĺpec (v programe nemeň iné príkazy, nepridávaj ďalšie):



B. Zmeň túto podmienku tak, aby sa nakreslil obrázok, v ktorom sa zafarbia obe uhlopriečky:



Programy by mali fungovať správne aj pri zmenenom rozmere `n`. Vyskúšaj napríklad `n=10`.

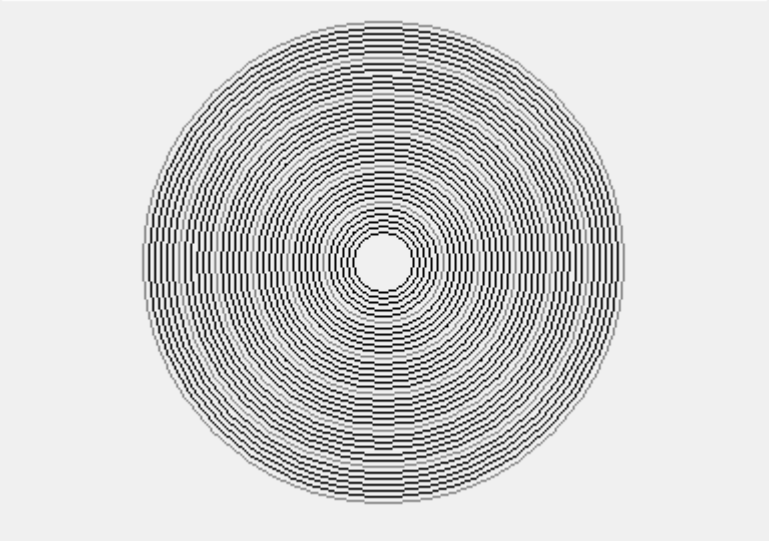
9. Žiaci sú v rade zoradení podľa veľkosti (od najmenšieho). Napíš program, ktorému najprv postupne oznamujeme (pomocou `input`) výšky žiakov a ten na záver (zadali sme „prázdnu“ výšku) vypíše, či boli zoradení správne. Použi príkaz `while` (dopredu nepoznáme počet žiakov), v ktorom bude príkaz `if`. Mali by fungovať takéto spustenia programu:

```
zadávaj výšky žiakov
   výška 1. žiaka: 100
   výška 2. žiaka: 110
   výška 3. žiaka: 110
   výška 4. žiaka: 120
   výška 5. žiaka:
všetci žiaci sú zoradení správne

zadávaj výšky žiakov
   výška 1. žiaka: 100
   výška 2. žiaka: 120
   výška 3. žiaka: 110
   výška 4. žiaka: 140
   výška 5. žiaka: 145
   výška 6. žiaka:
žiaci nie sú správne zoradení

zadávaj výšky žiakov
   výška 1. žiaka: 180
   výška 2. žiaka:
všetci žiaci sú zoradení správne
```

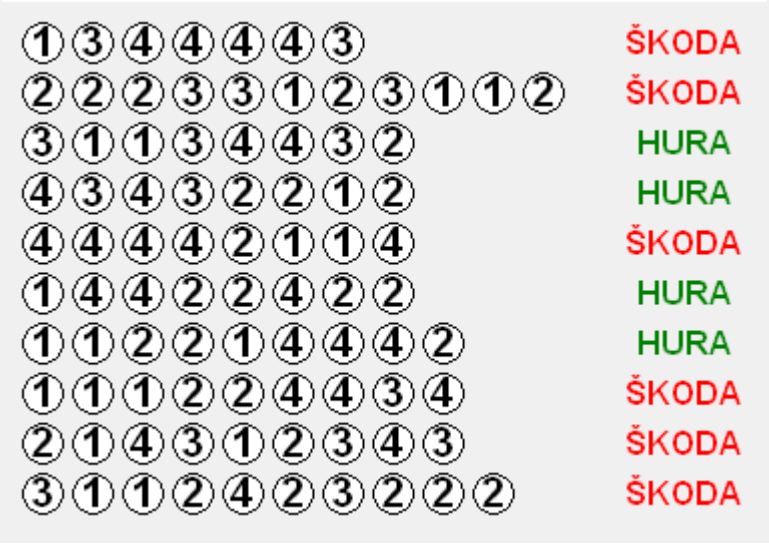
10. Napíš program, ktorý nakreslí gramofónovú LP platňu ako niekoľko sústredných kružníc. Najväčšia z nich má polomer `r` a každá ďalšia je o `3` menšia. Najmenšia kružnica by nemala mať menší polomer ako `15`. Každú `k`-tu kružnicu nakresli šedou farbou (začni od najväčšej). Napríklad pre premenné:

```
x, y = 190, 130
r = 120
k = 6
```

by si mal dostať takýto obrázok:

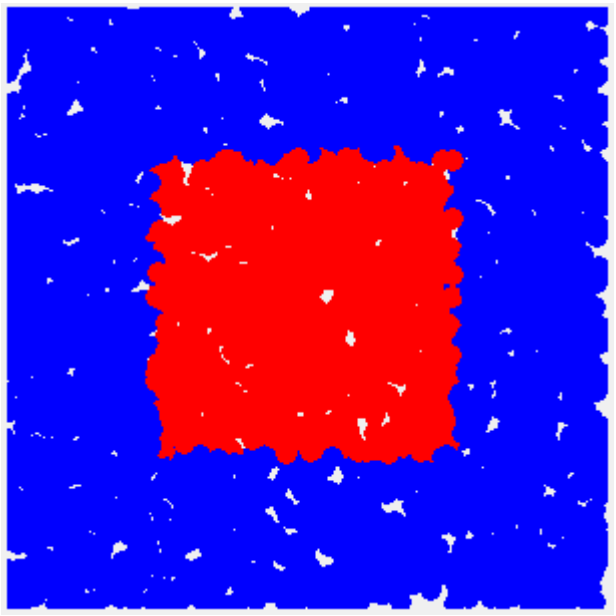


11. Budem hrať takúto hru: kladiem vedľa seba do radu mince s náhodnými hodnotami z `<1, 4>`; skončím, keď ich súčet bude väčší alebo rovný danému `k`. Ak skončil so súčtom, ktorý je rovný `k`, vypíše text `'HURÁ'` inak `'ŠKODA'`. Napíš program, ktorý túto hru odsimuluje 10-krát a zakreslí to pod seba, napríklad pre ¨k=21¨ môžeš dostať takýto obrázok:

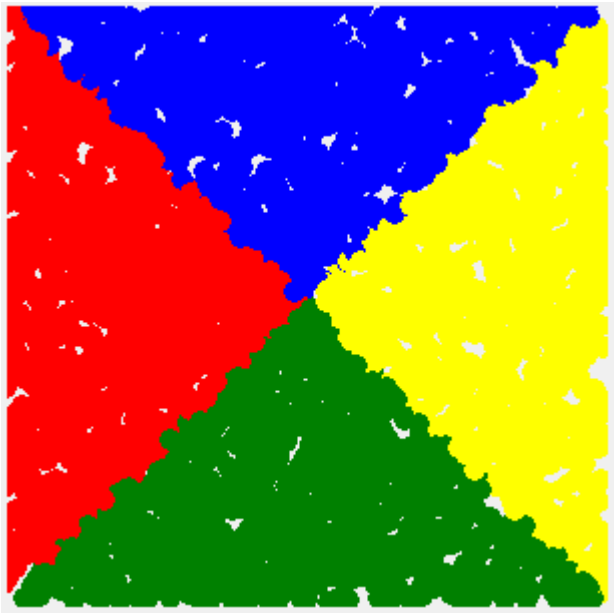


12. Využi program z prednášky, v ktorom sa vykresľovalo 1000 farebných bodiek (malé krúžky s polomerom 5 bez obrysu) podľa toho či mali `x`-ovú, alebo `y`-ovú súradnicu menšiu alebo väčšiu ako `150`. Veľkosť grafickej plochy bola 300x300. Zmeň v tomto

programe sériu príkazov `if` tak, aby kreslené bodky vytvorili vnútorný červený štvorec s rozmermi 150x150. Vykresli s `4000` farebnými bodkami:



13. Podobná úloha ako v predchádzajúcom príklade, len v tomto sa využívajú uhlopriečky štvorca. Asi tu budeš vedieť využiť podmienky `x < y` alebo `300 - x < y`:

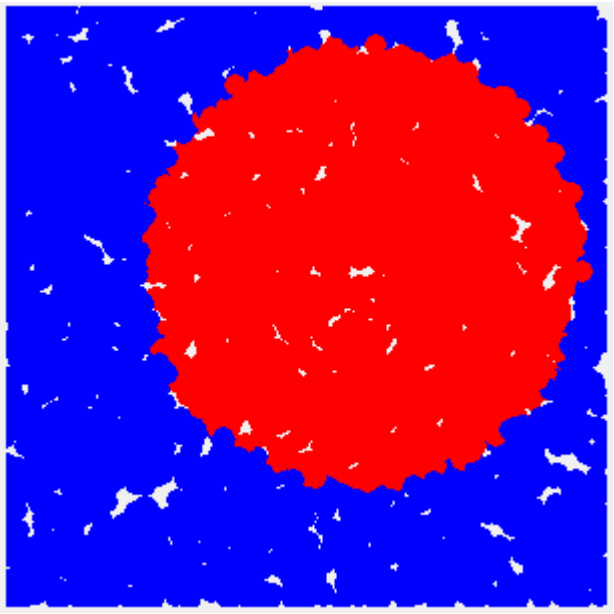


14. V ďalšej verzii bodkovacej úlohy vybodkuješ kruh. Program opäť nakreslí 4000 náhodných bodiek, ale tie z nich, ktoré majú vzdialenosť od `(x0, y0)` menšiu ako `r`, zafarbí na červeno, zvyšné na modro. Napríklad pre premenné:

```
15. x0, y0 = 180, 130
```

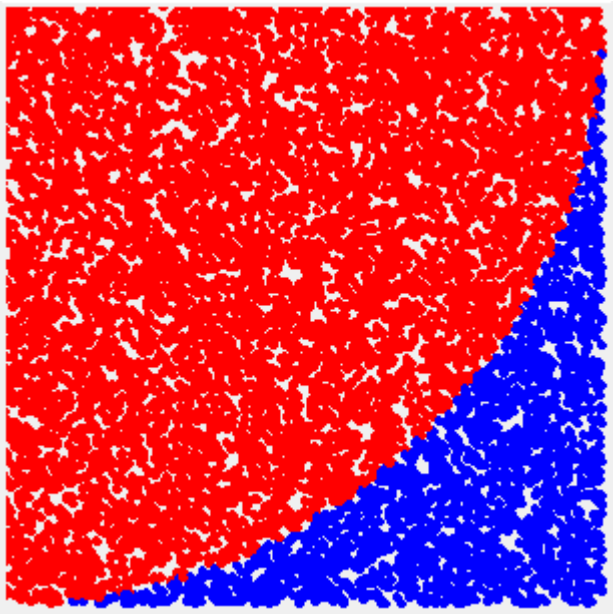
```
16.r = 110
```

by si mal dostať takýto obrázok:



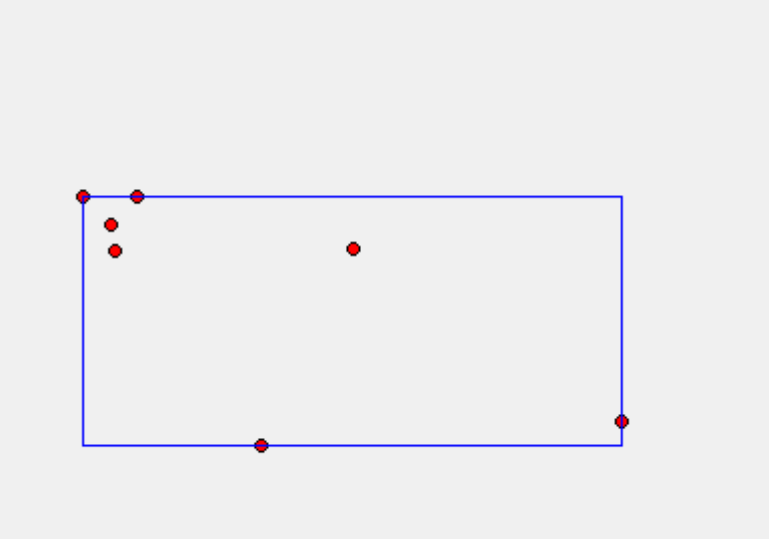
Vzdialenosť dvoch bodov v rovine môžeš počítať podľa vzorca `sqrt((x1-x2)**2 + (y1-y2)**2)`.

15. Aj ďalšia verzia programu generuje náhodné bodky v `300` x `300`. Bodiek teraz vygeneruj aspoň `10000` a zmenši ich na polomer `2`. Červené bodky budú tie, pre ktoré platí `x*x+y*y<=300*300`, zvyšné budú modré. Program ma záver vypíše podiel počtu červených ku všetkým bodkám krát `4`. Zamysli sa nad tým, prečo sa tento podiel blíži k číslu **pi**. Napríklad:



vyšiel tento podiel `3.1436`.

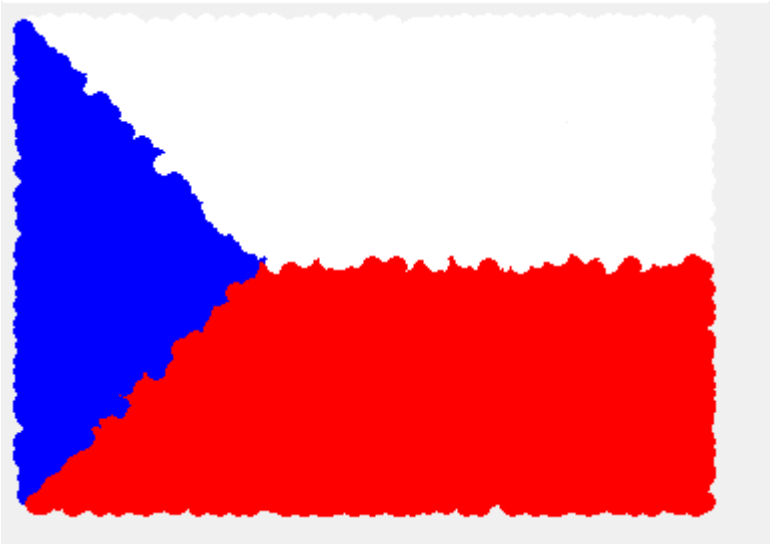
16. Napíš program, ktorý v grafickej ploche najprv vygeneruje `n` náhodných bodiek (malé kruhy). Potom nakreslí **čo najmenší** obdĺžnik tak, aby sa v ňom nachádzali všetky nakreslené body. Napríklad, pre `n=7` môžeš dostať takýto obrázok:



17. Vlajku Nemecka môžeš do štandardnej veľkosti grafickej plochy nakresliť tak, že pomocou cyklu vygeneruješ 10000 náhodných súradníc `x` z intervalu `<10, 350>` a `y` z intervalu `<10, 250>`. Na vygenerované súradnice nakreslíš farebnú bodku (kruh s polomerom `5` bez obrysu). Farbu bodky zvolíš podľa y-ovej súradnice:
    - ak je `y < 90` kresli čierny krúžok,
    - inak, ak je `y < 170` kresli červený krúžok,
    - inak nakresli žltý (môže byť `'gold'`) krúžok.

    Mal by si dostať takýto obrázok:

18. Vytvor program, ktorý podobne, ako v predchádzajúcej úlohe, nakreslí vlajku Českej republiky (bývalého Československa):



19. Hrací automat mi pri každom zatlačení (hodil som do neho 1 euro) dá 3 náhodné čísla z intervalu `<1, 20>`. Ak sú medzi týmito tromi číslami dve rovnaké, vyhrávam `5` euro, ak sú všetky tri rovnaké vyhrávam `100` euro. Začal som s nejakou sumou a hral som maximálne `1000` zatlačení, prípadne som skončil skôr, keď som všetko prehral. Napíš program, ktorý to všetko odsimuluje: pri každom zatlačení vypíše `'+5'` alebo `'+100'`, ak som niečo vyhral, alebo `'-1'`, ak som nič nevyhral. Napríklad:

```
začínam so sumou: 20
štart -1-1-1-1-1-1-1+5-1-1-1-1+5-1-1-1-1-1-1-1-1+5-1+5-1-1-1-1-1-1+5-1+5
-1-1-1-1-1-1-1-1-1-1+5+5-1-1-1-1+5-1-1-1+5-1-1-1+5+5-1-1+5+5-1-1+5-1-1-1-1
-1-1-1-1+5-1-1-1+5-1+5-1-1-1-1+5-1+5-1-1+5-1-1-1-1-1-1-1+5-1+5-1-1-1-1-1
+5-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1+5-1-1-1-1-1+5-1
-1-1-1+5-1-1+5-1-1-1-1-1-1-1-1-1-1-1-1
zostalo mi 0 euro
```

20. Máme mince a bankovky s hodnotami `1`, `2`, `5`, `10`, `20`, `50`, `100`. Napíš program, ktorý zistí, ako sa dá poskladať ľubovoľná suma peňazí **minimálnym počtom** kusov peňazí. Použi len jeden for-cyklus, v ktorom bude jeden `if`, nejaké priradenia a tiež `print`. Napríklad po spustení:

```
zadaj cislo: 346
3 krát hodnota 100
2 krát hodnota 20
1 krát hodnota 5
1 krát hodnota 1
```

21. Dvojkový logaritmus môžeme počítať pomocou funkcie `math.log2(cislo)`, ale vieme na to použiť aj algoritmus z prednášky, ktorý delením intervalu na polovice počítal

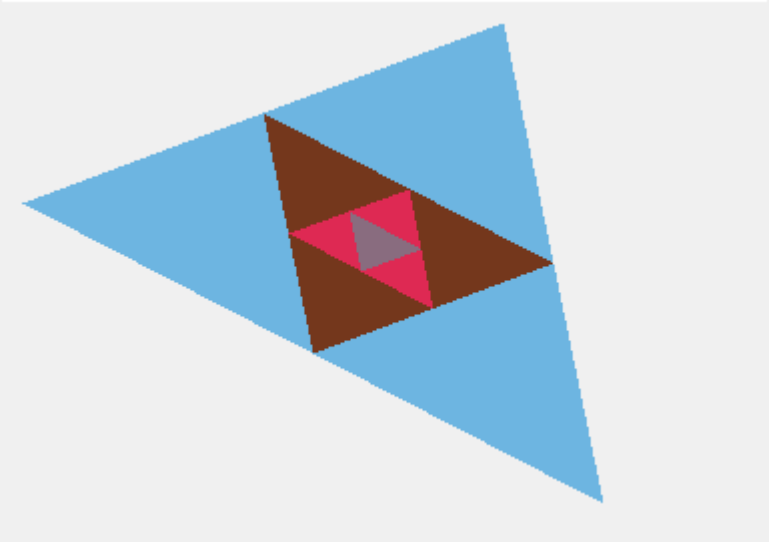
druhú odmocninu. Napíš program, ktorý to s nejakou presnosťou vypočíta takýmto algoritmom. Napríklad po spustení:

```
zadaj číslo: 1000
logaritmus 1000.0 je 9.965784382075071
>>> import math
>>> math.log2(1000)
9.965784284662087
```

22. Pomocou polygónu nakresli farebný trojuholník zadaný tromi jeho vrcholmi `a`, `b`, `c`. Potom do neho nakresli vpísaný trojuholník (jeho vrcholy sú v stredoch strán). Toto opakuj, kým sú všetky dĺžky strán trojuholníka aspoň 30. Trojuholníky vyfarbuj náhodnými farbami. Napríklad pre:

```
a = 10, 100
b = 250, 10
c = 300, 250
```

po spustení:



Pomôcka: ak mám dva vrcholy `(x1, y1)` a `(x2, y2)`, potom stred úsečky je `x3, y3 = (x1+x2)/2, (y1+y2)/2`

2. Týždenný projekt

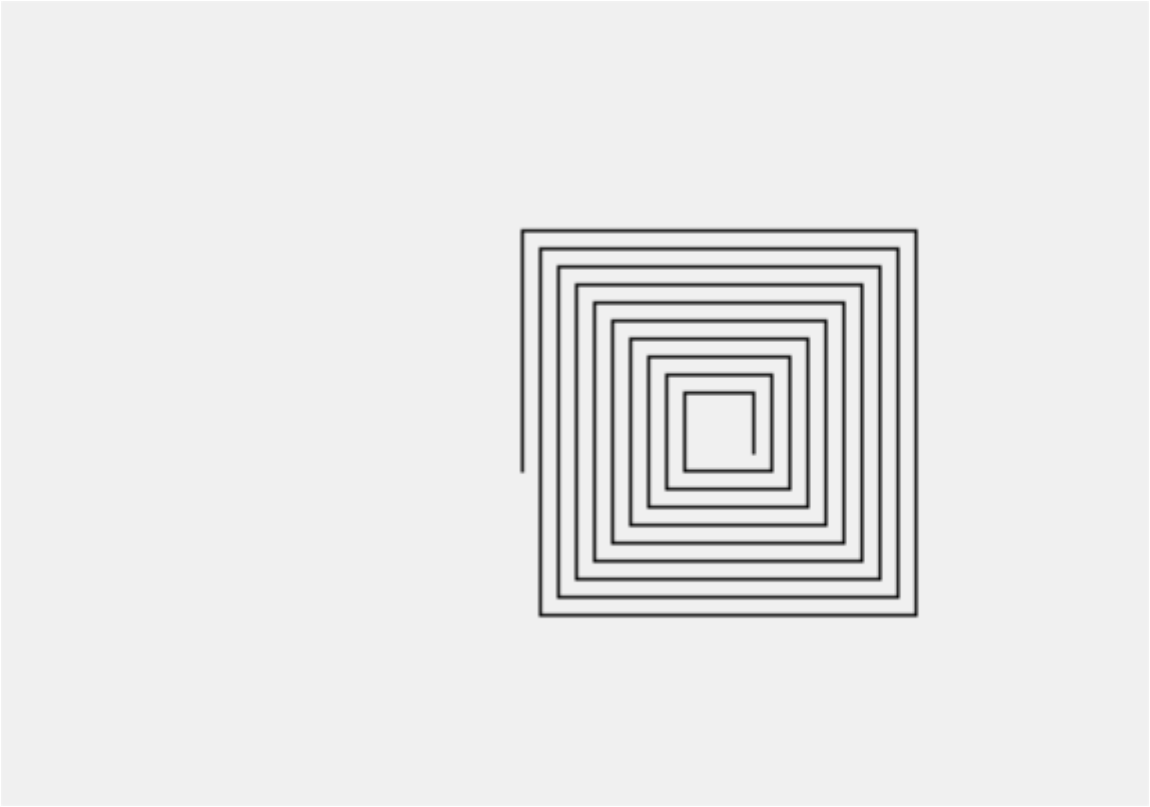
L.I.S.T.

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

Napíš pythonovský skript, ktorý najprv pomocou jedného volania príkazu `input()` prečíta tri celé čísla (oddelene medzerou): **štartová_dĺžka**, **inkrement** a **súčet_dĺžok**. Potom do grafickej plochy nakreslí štvorcovú špirálu, v ktorej prvá najkratšia úsečka má

dĺžku **štartová_dĺžka**, každá ďalšia úsečka špirály je o **inkrement** dlhšia. Kreslenie špirály končí vtedy, keď bude súčet všetkých úsečiek presne zadaný **súčet_dĺžok**. Posledná úsečka špirály môže byť aj kratšia, ale zrejme tak, aby súčet všetkých spolu bol presne zadané číslo (dokonca aj prvá úsečka môže mať kratšiu ako **štartová_dĺžka**, ak je zároveň poslednou úsečkou). Štvorcová špirála označuje, že susedné úsečky zvierajú 90 stupňov (je jedno, či smerom vľavo alebo vpravo). Špirálu môžeš začať kresliť na ľubovoľnej pozícii s ľubovoľným počiatočným natočením. Nemusí sa zmestiť do grafickej plochy.

Napríklad, ak zadáš v jednom `input()` tieto tri hodnoty: `20 3 2950`, môžeš dostať takýto obrázok:



V tomto riešení má špirála úsečky rovnobežné s osami. Pritom aj takéto riešenie:

je korektné.

V tvojom riešení nepoužívaj žiaden iný modul okrem `tkinter`. Z Pythonu používaj len príkazy z prvých štyroch prednášok. Do grafického plátna (`canvas`) kresli len pomocou `create_line`, do ktorého pošleš 4 čísla ako súradnice dvoch bodov.

Ak štvorcová špirála bude mať úsečky rovnobežné s osami, môžeš takto získať maximálne **75%** bodov z maximálneho počtu **3** body. **100%** bodov získaš len vtedy, keď bude špirála nejako natočená, napríklad ako na 2. obrázku.

Tvoj odovzdaný program s menom `riesenie.py` musí začínať tromi riadkami komentárov (zmeň meno a dátum odovzdania):

```
# 2. zadanie: spirala
# autor: Janko Hraško
# datum: 15.10.2021
```

Súbor `riesenie.py` odovzdávaj na úlohový server https://list.fmph.uniba.sk/ najneskôr do **23:00 15. októbra**, kde ho môžeš nechať otestovať. Odovzdať projekt a aj ho testovať môžeš ľubovoľný počet krát. Za riešenie tohto projektu môžeš získať **3 body**.