8. Zoznamy

video prezentácia

zoznamy

Už poznáme tieto typy údajov:

- **jednoduché**:
  - číselné `int` a `float`
  - logické `bool`
- **postupnosti**:
  - postupnosti znakov `str`
  - postupnosti riadkov - otvorený textový súbor
  - postupnosti čísel - pomocou `range()`

Dátová štruktúra zoznam

- v Pythone sa tento typ volá **list**
- je to vlastne **postupnosť** hodnôt ľubovoľných typov (zvykne sa používať aj pojem **kolekcia**, po anglicky *collection*)
- hovoríme, že typ zoznam sa skladá z **prvkov**
- okrem názvu **zoznam**, môžeme používať aj názov **tabuľka** alebo **pole** (pole väčšinou pre zoznamy hodnôt rovnakého typu)
- tento typ sa podobá na **pole** v iných jazykoch (napríklad v Pascale je to dynamické pole, ktorého ale hodnoty musia byť rovnakého typu) - často im tak budeme hovoriť aj v Pythone

Zoznamy vytvárame vymenovaním prvkov v hranatých zátvorkách, v príklade ich hneď aj priradíme do rôznych premenných:

```
>>> teploty = [10, 13, 15, 18, 17, 12, 12]
>>> nakup = ['chlieb', 'mlieko', 'rozky', 'jablka']
>>> studenti = ['Juraj Janosik', 'Emma Drobna', 'Ludovit Stur', 'Pavol Habera', 'Margita Figuli
']
>>> zviera = ['pes', 'Dunco', 2011, 35.7, 'hneda']
>>> prazdny = []                                  # prázdny zoznam
>>> print(teploty)
    [10, 13, 15, 18, 17, 12, 12]
>>> type(zviera)
    <class 'list'>
```

Všimnite si, že niektoré z týchto zoznamov majú všetky prvky rovnakého typu (napríklad, všetky prvky sú celé čísla alebo všetky sú reťazce).

Operácie so zoznamami

Základné operácie so zoznamami fungujú skoro presne rovnako, ako ich vieme používať so znakovými reťazcami:

- **indexovanie** pomocou hranatých zátvoriek `[ ]` - je úplne rovnaké ako pri reťazcoch: indexom je celé číslo od 0 do počet prvkov zoznamu - 1, alebo je to záporné číslo, napríklad:

```
>>> zviera[0]
    'pes'
>>> nakup[1]
    'mlieko'
>>> studenti[-1]
    'Margita Figuli'
>>> ['red', 'blue', 'yellow', 'green'][1]
    'blue'
>>> ['red', 'blue', 'yellow'][2][4]
    'o'
```

- **zreťazenie** pomocou operácie `+` označuje, že vytvoríme nový väčší zoznam, ktorý bude obsahovať najprv prvky prvého zoznamu a za tým všetky prvky druhého zoznamu, napríklad:

```
>>> nakup2 = ['zosity', 'pero', 'vreckovky']
>>> nakup + nakup2
    ['chlieb', 'mlieko', 'rozky', 'jablka', 'zosity', 'pero', 'vreckovky']
>>> studenti = studenti + ['Karel Capek']
>>> studenti
    ['Juraj Janosik', 'Emma Drobna', 'Ludovit Stur', 'Pavol Habera', 'Margita Figuli', '
Karel Capek']
>>> prazdny + prazdny
    []
>>> [1] + [2] + [3, 4] + [] + [5]
    [1, 2, 3, 4, 5]
```

- **viacnásobné zreťazenie** pomocou operácie `*` označuje, že daný zoznam sa navzájom zreťazí určený počet krát, napríklad:

```
>>> jazyky = ['Python', 'Pascal', 'C++', 'Java', 'C#']
>>> vela = 3 * jazyky
>>> vela
    ['Python', 'Pascal', 'C++', 'Java', 'C#', 'Python', 'Pascal', 'C++', 'Java', 'C#', '
Python',
     'Pascal', 'C++', 'Java', 'C#']
>>> sto_krat_nic = 100 * [None]
>>> sto_krat_nic
    [None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None, None, None, None, None,
None,
     None, None, None, None, None, None, None, None, None, None]
>>> prazdny * 1000
    []
```

- **zisťovanie prvku** pomocou `in` označuje, či sa nejaká hodnota nachádza v danom zozname, napríklad:

```
>>> 'Pavol Habera' in studenti
    True
>>> 'pero' in nakup
```

```
    False
>>> 'pascal' in jazyky
    False
>>> prazdny in sto_krat_nic
    False
>>> teploty = [10, 13, 15, 18, 17, 12, 12]
>>> 18 in teploty
    True
>>> [18, 17] in teploty
    False
```

V poslednom príklade testujeme, či sa dvojprvkový zoznam `[18, 17]` nachádza niekde v zozname `teploty`. Lenže tento zoznam obsahuje len celé čísla a žiaden prvok nie je typu zoznam. Hoci pre znakové reťazce fungovalo hľadanie podreťazca, napríklad:

```
>>> 'th' in 'Python'
    True
```

pre zoznamy táto analógia nefunguje.

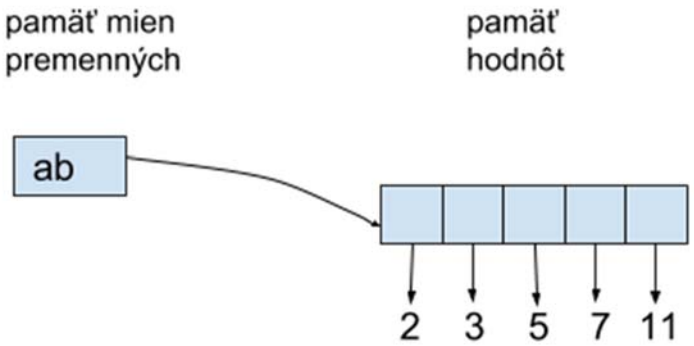
Ešte si pripomeňte zápis negácie takéhoto testu, ktorý analogicky funguje pre reťazce aj zoznamy:

```
>>> not 'Y' in 'Python'
    True
>>> 'Y' not in 'Python'
    True
>>> 'str' not in ['pon', 'uto', 'str', 'stv', 'pia', 'sob', 'ned']
    False
>>> 'štv' not in ['pon', 'uto', 'str', 'stv', 'pia', 'sob', 'ned']
    True
```

Pripomeňme si, ako vyzerajú premenné a ich hodnoty v pamäti Pythonu. Urobme toto priradenie:

```
ab = [2, 3, 5, 7, 11]
```

Do pamäti mien (globálny menný priestor) pribudne jeden identifikátor premennej `ab` a tiež referencia na päťprvkový zoznam `[2, 3, 5, 7, 11]`. Tento zoznam môžeme v pamäti hodnôt vizualizovať ako päť vedľa seba položených škatuliek, pričom v každej je referencia na príslušnú hodnotu:



Je dobre si uvedomiť, že momentálne máme v pamäti 6 premenných, jedna z nich je `ab` (je typu `list`) a zvyšných päť je `ab[0]`, `ab[1]`, `ab[2]`, `ab[3]` a `ab[4]` (všetky sú typu `int`).

Prechádzanie prvkov zoznamu

Tzv. **iterovanie** najčastejšie pomocou for-cyklu. Napríklad:

```
>>> teploty
    [10, 13, 15, 18, 17, 12, 12]
>>> for i in range(7):
        print(f'{i}. deň', teploty[i])
    0. deň 10
    1. deň 13
    2. deň 15
    3. deň 18
    4. deň 17
    5. deň 12
    6. deň 12
```

Využili sme indexovanie prvkov zoznamu indexmi od 0 do 6. Ak nepotrebujeme pracovať s indexmi, ale stačia nám samotné hodnoty, zapíšeme:

```
>>> for prvok in teploty:
        print(prvok, end=', ')
    10, 13, 15, 18, 17, 12, 12,
```

Prípadne, vieme využiť `enumerate`:

```
>>> for i, prvok in enumerate(teploty):
        print(f'{i+1}. deň', prvok)
    1. deň 10
    2. deň 13
    3. deň 15
    4. deň 18
    5. deň 17
    6. deň 12
    7. deň 12
```

Môžeme zapísať aj výpočet **priemernej** teploty:

```
teploty = [10, 13, 15, 18, 17, 12, 12]

sucet = 0
for prvok in teploty:
    sucet += prvok
priemer = sucet / 7
print(f'priemerná teplota je {priemer:.1f}')    # formátovanie desatinného čísla na jedno miest
o

priemerná teplota je 13.9
```

Podobne vieme zistiť, napríklad **maximálnu hodnotu** v zozname:

```
mx = teploty[0]
for prvok in teploty:
    if mx < prvok:
        mx = prvok
print('najteplejšie bolo', mx, 'stupňov')

najteplejšie bolo 18 stupňov
```

Alebo zistenie **počtu nadpriemerne** teplých dní:

```
teploty = [10, 13, 15, 18, 17, 12, 12]

sucet = 0
for prvok in teploty:
    sucet += prvok
```

```
priemer = sucet / 7
pocet = 0
for prvok in teploty:
    if prvok > priemer:
        pocet += 1
print('počet nadpriemerne teplých dní', pocet)
```

```
počet nadpriemerne teplých dní 3
```

Z týchto jednoduchých príkladov môžeme zapísať šablóny, ktoré niečo zisťujú o prvkoch zoznamu.

šablóna na zisťovanie hodnôt v zozname

Prvá jej verzia spočíta hodnoty prvkov (prvky by sme mohli, napríklad aj násobiť, alebo zreťazovať):

```
sucet = 0
for prvok in zoznam:
    sucet = sucet + prvok
print(sucet)
```

V druhej verzii šablóny budeme zisťovať nejakú informáciu o prvkoch zoznamu, napríklad minimálny prvok:

```
mn = zoznam[0]
for prvok in zoznam:
    if prvok < mn:
        mn = prvok
print(mn)
```

Zrejme podmienka aj priradenie budú závisieť od konkrétnej úlohy.

Napríklad, nasledovný program vypíše prvky zoznamu do jedného riadku:

```
zoznam = [47, 'ab', -13, 22, 9, 25]
print('prvky zoznamu:', end=' ')
for prvok in zoznam:
    print(prvok, end=' ')
print()
```

dostaneme výpis:

```
prvky zoznamu: 47 ab -13 22 9 25
```

Ale tento program:

```
for i in range(10):
    index = i % 4
    farba = ['red', 'blue', 'yellow', 'green'][index]
    print(farba)
```

pristupuje k niektorým prvkom aj viackrát a vypíše:

```
red
blue
yellow
green
red
blue
yellow
green
```

```
red
blue
```

Tento for-cyklus by sme mohli zapísať aj takto:

```
for i in range(10):
    print(['red', 'blue', 'yellow', 'green'][i%4])
```

Zmena hodnoty prvku zoznamu

Dátová štruktúra zoznam je **meniteľný** typ (tzv. **mutable**) - môžeme meniť hodnoty prvkov zoznamu, napríklad takto:

```
>>> studenti[3]
    'Pavol Habera'
>>> studenti[3] = 'Janko Hrasko'
>>> studenti[2] = 'Guido van Rossum'
>>> studenti
    ['Juraj Janosik', 'Emma Drobna', 'Guido van Rossum', 'Janko Hrasko', 'Margita Figuli', 'Kar
el Capek']
```

Môžeme zmeniť hodnoty prvkov zoznamu aj v cykle, ale k prvkom musíme pristupovať pomocou indexov, napríklad sme zistili, že náš teplomer ukazuje o 2 stupne menej ako je reálna teplota, preto opravíme všetky prvky zoznamu:

```
teploty = [10, 13, 15, 18, 17, 12, 12]
for i in range(7):
    teploty[i] = teploty[i] + 2
print(teploty)
```

```
[12, 15, 17, 20, 19, 14, 14]
```

Krajšie by sme to zapísali s využitím štandardnej funkcie `len()`, ktorá vráti počet prvkov zoznamu:

```
teploty = [10, 13, 15, 18, 17, 12, 12]
for i in range(len(teploty)):
    teploty[i] += 2
print(teploty)
```

```
[12, 15, 17, 20, 19, 14, 14]
```

Uvedomte si, že ak by sme prvky zoznamu neindexovali, ale prechádzali by sme ich priamo cez premennú cyklu `prvok`:

```
teploty = [10, 13, 15, 18, 17, 12, 12]
for prvok in teploty:
    prvok += 2
print(teploty)
```

nebude to fungovať:

```
[10, 13, 15, 18, 17, 12, 12]
```

Samotný zoznam sa tým nezmení: menili sme len obsah premennej cyklu `prvok`, ale tým sa nezmení obsah zoznamu.

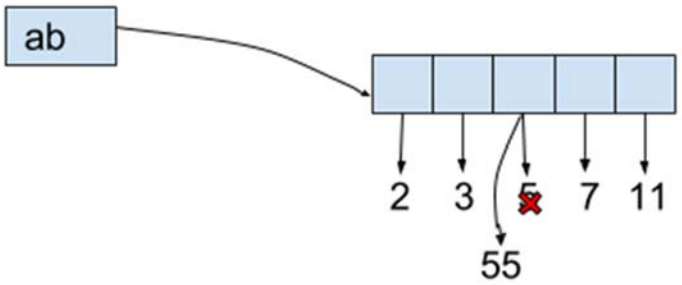
Je dobré si predstaviť, čo sa deje v pamäti pri zmene hodnoty prvku zoznamu. Zoberme si pôvodný päť prvkový zoznam prvočísel:

```
ab = [2, 3, 5, 7, 11]
```

Zmenou obsahu jedného prvku zoznamu sa zmení **jediná referencia**, všetko ostatné zostáva bez zmeny:

```
ab[2] = 55
```

dostávame:



Priraďovaním do jedného prvku zoznamu sa tento zoznam modifikuje, hovoríme, že priradenie do prvku je **mutable** operácia. Ukážme šablónu, pomocou ktorej vieme v cykle priradiť do rôznych prvkov rôzne hodnoty.

šablóna na vytváranie zoznamu

Ak potrebujeme v cykle priradiť do prvkov zoznamu rôzne hodnoty, príslušné indexy zoznamu už musia existovať. Najlepšie takýto zoznam pripravíme jedným priradením a viacnásobným zreťazením, napríklad pre `n` prvkov:

```
zoznam = [None] * n
for i in range(n):
    zoznam[i] = ... výpočet hodnoty
print(zoznam)
```

Napríklad pre vytvorenie zoznamu prvých `n` druhých mocnín čísel od `0` do `n-1`:

```
n = int(input('zadaj n: '))
mocniny = [None] * n
for i in range(n):
    mocniny[i] = i * i
print(mocniny)
```

```
zadaj n: 14
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169]
```

Štandardné funkcie so zoznamami

Nasledovné funkcie fungujú nielen so zoznamami, ale s ľubovoľnou postupnosťou hodnôt. V niektorých prípadoch však nemajú zmysel a vyhlásia chybu (napríklad číselný súčet prvkov znakového reťazca).

- funkcia `len(postupnosť)` -> vráti počet prvkov postupnosti
- funkcia `sum(postupnosť)` -> vypočíta číselný súčet prvkov postupnosti
- funkcia `max(postupnosť)` -> vráti maximálny prvok postupnosti (t.j. jeho hodnotu)
- funkcia `min(postupnosť)` -> vráti minimálny prvok postupnosti

Predchádzajúci príklad, v ktorom sme počítali priemernú, minimálnu aj maximálnu teplotu, prepíšeme:

```
teploty = [10, 13, 15, 18, 17, 12, 12]

sucet = sum(teploty)
maximum = max(teploty)
minimum = min(teploty)
priemer = sucet / len(teploty)
print('priemerná teplota je', f'{priemer:.1f}')
print('minimálna teplota je', minimum)
print('maximálna teplota je', maximum)
```

```
priemerná teplota je 13.9
minimálna teplota je 10
maximálna teplota je 18
```

Čo sa dá zapísať úspornejšie, ale menej čitateľne aj takto:

```
teploty = [10, 13, 15, 18, 17, 12, 12]

print('priemerná teplota je', f'{sum(teploty)/len(teploty):.1f}')
print('minimálna teplota je', min(teploty))
print('maximálna teplota je', max(teploty))
```

Funkcia list()

Už máme nejaké skúsenosti s tým, že v Pythone každý základný typ má definovanú svoju **konverznú funkciu**, pomocou ktorej sa dajú niektoré hodnoty rôznych typov prekonvertovať na daný typ. Napríklad

- `int(3.14)` -> vráti celé číslo `3`
- `int('37')` -> vráti celé číslo `37`
- `str(22 / 7)` -> vráti reťazec `'3.142857142857143'`
- `str(2 < 3)` -> vráti reťazec `'True'`

Podobne funguje aj funkcia `list(hodnota)`:

- parametrom musí byť **iterovateľná** hodnota, t.j. nejaká postupnosť, ktorá sa dá prechádzať (iterovať), napríklad for-cyklom
- funkcia `list()` túto postupnosť rozoberie na prvky a z týchto prvkov poskladá nový zoznam
- parameter môže chýbať, vtedy vygeneruje prázdny zoznam

Napríklad:

```
>>> list(zviera)                          # kópia existujúceho zoznamu
    ['pes', 'Dunco', 8, 35.7, 'hneda']
>>> list(range(5, 16))
    [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> list('Python')
    ['P', 'y', 't', 'h', 'o', 'n']
>>> list()                                # prázdny zoznam
    []
>>> list(3.14)
    ...
    TypeError: 'float' object is not iterable
```

Ak nejaký súbor obsahuje tieto riadky:

```
Aká
práca,

taká
pláca.
```

aj toto Python vidí ako **postupnosť** (otvorený súbor sa dá prechádzať for-cyklom ako postupnosť riadkov) a preto:

```
>>> list(open('subor.txt', encoding='utf-8'))
    ['Aká\n', 'práca,\n', '\n', 'taká\n', 'pláca.\n']
```

Rezy

Keď sme v znakovom reťazci potrebovali zmeniť nejaký znak, zakaždým sme museli vyrobiť kópiu reťazca, napríklad:

```
>>> retazec = 'Monty Python'
>>> retazec[4] = 'X'                    # takto sa to nedá
    ...
    TypeError: 'str' object does not support item assignment
>>> retazec = retazec[:4] + 'X' + retazec[5:]
>>> retazec
    'MontX Python'
```

Využili sme tu **rezy** (slice), t.j. získavanie podreťazcov. To isté sa dá použiť aj pri práci so zoznamami, lebo aj s nimi fungujú rezy, napríklad:

```
>>> jazyky
    ['Python', 'Pascal', 'C++', 'Java', 'C#']
>>> jazyky[1:3]
    ['Pascal', 'C++']
>>> jazyky[-3:]
    ['C++', 'Java', 'C#']
>>> jazyky[:-1]
    ['Python', 'Pascal', 'C++', 'Java']
```

Samozrejme, že pritom funguje aj určovanie kroku, napríklad:

```
>>> jazyky[1::2]
    ['Pascal', 'Java']
>>> jazyky[::-1]
    ['C#', 'Java', 'C++', 'Pascal', 'Python']
```

Uvedomte si, že takéto rezy nemenia obsah samotného zoznamu a preto hovoríme, že sú **immutable**.

Priraďovanie do rezu

Keď iba vyberáme nejaký podzoznam pomocou rezu, napríklad `zoznam[od:do:krok]`, takáto operácia s pôvodným zoznamom nič nerobí (len vyrobí úplne nový zoznam). Lenže my môžeme obsah zoznamu meniť aj takým spôsobom, že zmeníme len jeho nejakú časť. Takže rez zoznamu môže byť na ľavej strane priraďovacieho príkazu a potom na pravej strane priraďovacieho príkazu musí byť nejaká **postupnosť** (nemusí to byť zoznam). Priraďovací príkaz teraz túto postupnosť prejde, zostrojí z nej zoznam a ten vloží namiesto udaného rezu.

Preštudujte nasledovné príklady:

```
>>> zoz = list(range(0, 110, 10))
>>> zoz
    [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
>>> zoz[3:6] = ['begin', 'end']                 # tri prvky sa nahradili dvomi
>>> zoz
    [0, 10, 20, 'begin', 'end', 60, 70, 80, 90, 100]

>>> zoz[6:7] = [111, 222, 333]                  # jeden prvok sa nahradil tromi
>>> zoz
    [0, 10, 20, 'begin', 'end', 60, 111, 222, 333, 80, 90, 100]

>>> abc = list('Python')
>>> abc
    ['P', 'y', 't', 'h', 'o', 'n']
>>> abc[2:2]                                    # rez dĺžky 0
    []
>>> abc[2:2] = ['dve', 'slova']                 # rez dĺžky 0 sa nahradí dvomi prvkami
>>> abc
    ['P', 'y', 'dve', 'slova', 't', 'h', 'o', 'n']

>>> prvo = [2, 3, 5, 7, 11]
>>> prvo[1:-1]
    [3, 5, 7]
>>> prvo[1:-1] = []                             # rez dĺžky 3 sa nahradí žiadnymi prvkami
>>> prvo                                        # prvky sa takto vyhodili
    [2, 11]
```

Pozor! Všetky tieto príklady modifikujú pôvodný zoznam, teda priraďovanie do rezu je **mutable operácia**.

Porovnávanie zoznamov

Zoznamy môžeme navzájom porovnávať (na rovnosť, alebo menší/väčší). Funguje to na rovnakom princípe ako porovnávanie znakových reťazcov:

- postupne sa prechádzajú prvky jedného aj druhého zoznamu, kým sú rovnaké
- ak je jeden so zoznamov kratší, tak ten sa považuje za menší ako ten druhý
- ak pri postupnom porovnávaní prvkov nájde rôzne hodnoty, výsledok porovnania týchto dvoch rôznych hodnôt je výsledkom porovnania celých zoznamov
- každé dva porovnávané prvky musí Python vedieť porovnať: na rovnosť je to bez problémov, ale relačné operácie `<` a `>` nebudú fungovať napríklad pre porovnávanie čísel a reťazcov

Napríklad:

```
>>> [1, 2, 5, 3, 4] > [1, 2, 4, 8, 1000]
    True
>>> [1000, 2000, 3000] < [1000, 2000, 3000, 0, 0]
    True
>>> [1, 'ahoj'] == ['ahoj', 1]
    False
>>> [1, 'ahoj'] < ['ahoj', 1]
    ...
    TypeError: '<' not supported between instances of 'int' and 'str'
```

Zoznam ako parameter funkcie

Už predtým sme zisťovali priemernú teplotu zo zoznamu nameraných hodnôt. Teraz z toho vieme urobiť funkciu, napríklad:

```
def priemer(zoznam):
    sucet = 0
    pocet = 1
```

```
    for prvok in zoznam:
        sucet += prvok
        pocet += 1
    return sucet / pocet
```

Keďže pomocou štandardných funkcií `sum()` a `len()` vieme veľmi rýchlo zistiť súčet aj počet prvkov zoznamu, môžeme to zapísať elegantnejšie:

```
def priemer(zoznam):
    return sum(zoznam) / len(zoznam)
```

Ďalšia funkcia zisťuje počet výskytov nejakej konkrétnej hodnoty v zozname:

```
def pocet(zoznam, hodnota):
    vysl = 0
    for prvok in zoznam:
        if prvok == hodnota:
            vysl += 1
    return vysl
```

a naozaj funguje:

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

Zaujímavé je aj to, že funkcia funguje nielen pre zoznamy, ale pre ľubovoľnú postupnosť (iterovateľnú štruktúru), napríklad:

```
>>> pocet('bla-bla-bla', 'l')
    3
>>> pocet('bla-bla-bla', 'la')     # v postupnosti znakov sa 'la' nenachádza
    0
```

Pre znaky táto funkcia robí skoro to isté ako **metóda** `count()`, teda zápis:

```
>>> 'bla-bla-bla'.count('l')
    3
```

naša funguje úplne rovnako ako funkcia `pocet()`. Aj pre zoznamy existuje metóda `count`, ktorá robí presne to isté ako naša funkcia `pocet()`.

Metódy

Už vieme, že metódami voláme také funkcie, ktoré fungujú s nejakou hodnotou: za túto hodnotu dávame bodku (preto tzv. **bodková notácia**) a samotné meno funkcie s prípadnými parametrami. V prípade zoznamov to vyzerá takto:

```
zoznam.funkcia(parametre)
```

Pre zoznamy existujú tieto dve metódy, ktoré nemodifikujú ich obsah a preto vieme, že sú **immutable**.

metóda `count()`

Volanie metódy:

```
zoznam.count(hodnota)
```

vráti počet výskytov danej hodnoty v zozname. Táto metóda je **immutable** lebo nemení obsah zoznamu.

Metódu môžeme použiť nielen s premennou typu zoznam, napríklad:

```
>>> zoz = [1, 2, 3, 2, 1, 2]
>>> zoz.count(2)
    3
>>> zoz.count(4)
    0
```

Ale aj priamo s hodnotou zoznam, napríklad:

```
>>> [0, 1, 0, 0, 1, 0, 1, 0, 0].count(1)
    3
```

Alebo s výrazom, ktorý je typu zoznam:

```
>>> ([3, 7] * 100 + [7, 8] * 50).count(7)
    150
```

metóda `index()`

Volanie metódy:

```
zoznam.index(hodnota)
```

vráti index prvého výskytu danej hodnoty v zozname. Táto metóda je **immutable** lebo nemení obsah zoznamu. Funkcia spadne na chybe, ak sa daná hodnota v zozname nenachádza.

Aj túto metódu môžeme použiť rôznym spôsobom, napríklad s premennou typu zoznam:

```
>>> farby = ['red', 'blue', 'red', 'blue', 'yellow']
>>> farby.index('blue')
    1
>>> farby.index('green')
    ...
    ValueError: 'green' is not in list
```

Pri používaní tejto metódy musíme dávať pozor, aby nám program nepadal, keď sa daná hodnota v zozname nenachádza. Napríklad takto:

```
farby = ['red', 'blue', 'red', 'blue', 'yellow']
if 'green' in farby:
    index = farby.index('green')
else:
    print('"green" sa v zozname nenachádza')

'green' sa v zozname nenachádza
```

Všetky ďalšie metódy, ktoré tu uvedieme, sú **mutable**, teda budú modifikovať samotný zoznam.

metóda `append()`

Volanie metódy:

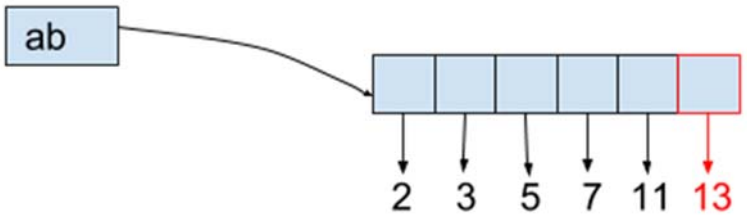
```
zoznam.append(hodnota)
```

pridá na koniec zoznamu nový prvok - zoznam sa takto predĺži o 1. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia nič nevracia, preto nemá zmysel priraďovať jej volanie do nejakej premennej (teda vracia hodnotu `None`).

Napríklad volanie:

```
>>> ab = [2, 3, 5, 7, 11]
>>> ab.append(13)
>>> ab
    [2, 3, 5, 7, 11, 13]
```

takto sa zmení vizualizácia pamäte pre premennú `ab`:



Zrejme nemá zmysel volať túto metódu s hodnotou typu zoznam namiesto premennej. Hoci to funguje dobre, nemáme šancu zistiť, ako vyzerá daný zoznam s pridaným prvkom:

```
>>> [1, 2, 3].append(4)
```

Niekde v pamäti hodnôt sa vyrobil zoznam `[1, 2, 3, 4]`, na ktorý ale nemáme žiadnu referenciu.

Vďaka tejto metóde sa oplatí naučiť aj druhý spôsob vytvárania zoznamu.

šablóna na vytváranie zoznamu pomocou append

Ak potrebujeme v cykle vytvárať nejaký zoznam rôznych hodnôt, pričom nemusíme poznať jeho výslednú dĺžku, môžeme takto elegantne využiť metódu `append`:

```
zoznam = []               # najprv je zoznam prázdny
for i in range(n):        # alebo for-cyklus pre inú postupnosť
    if ... nejaka_podmienka:
        nova_hodnota = ...
        zoznam.append(nova_hodnota)
print(zoznam)
```

Príklady:

- zoznam hodnôt postupnosti čísel `range(2, 100, 13)`:

```
zoznam = []
for i in range(2, 100, 13):
    zoznam.append(i)
print(zoznam)
```

```
[2, 15, 28, 41, 54, 67, 80, 93]
```

v tomto konkrétnom prípade sa to dalo zapísať aj takto výrazne jednoduchšie:

```
zoznam = list(range(2, 100, 13))
print(zoznam)
```

- zoznam reťazcov, ktoré sú vytvorené z mocnín `2` a obsahujú cifru `7`:

```
zoznam = []
for i in range(30):
    hodnota = str(2 ** i)
    if '7' in hodnota:
        zoznam.append(hodnota)
print(zoznam)
```

```
['32768', '131072', '1048576', '2097152', '16777216', '67108864', '134217728', '536870912']
```

metóda `pop()`

Volanie metódy:

```
zoznam.pop()
```

odoberie z konca zoznamu posledný prvok - zoznam sa takto skráti o 1. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia vracia hodnotu odobratého prvku. Ak bol zoznam prázdny, funkcia nič nevracia ale spadne na chybe.

Napríklad:

```
>>> abc = ['raz', 'dva', 'tri']
>>> for i in range(4):
        abc.pop()
    'tri'
    'dva'
    'raz'
    ...
    IndexError: pop from empty list
```

metóda `insert()`

Volanie metódy:

```
zoznam.insert(index, hodnota)
```

pridá na dané miesto zoznamu nový prvok - zoznam sa takto predĺži o 1. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia nič nevracia, preto nemá zmysel priraďovať jej volanie do nejakej premennej (teda vracia hodnotu `None`).

Napríklad:

```
>>> abc = ['raz', 'dva', 'tri']
>>> abc.insert(10, 'koniec')
>>> abc
    ['raz', 'dva', 'tri', 'koniec']
>>> abc.insert(2, 'stred')
>>> abc
    ['raz', 'dva', 'stred', 'tri', 'koniec']
>>> abc.insert(0, 'zaciatok')
>>> abc
    ['zaciatok', 'raz', 'dva', 'stred', 'tri', 'koniec']
>>> abc.insert(-1, 'predposledny')
>>> abc
    ['zaciatok', 'raz', 'dva', 'stred', 'tri', 'predposledny', 'koniec']
```

Uvedomte si, že `zoznam.insert(len(zoznam), hodnota)` pridáva vždy na koniec zoznamu, teda robí to isté ako `zoznam.append(hodnota)`.

metóda `pop()` s indexom

Volanie metódy:

```
zoznam.pop(index)
```

odoberie zo zoznamu príslušný prvok (daný indexom) - zoznam sa takto skráti o 1. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia vracia hodnotu odobratého prvku. Ak bol zoznam prázdny, funkcia nič nevracia ale spadne na chybe.

Napríklad:

```
>>> abc = ['raz', 'dva', 'tri', 'styri']
>>> abc.pop(7)
    ...
    IndexError: pop index out of range
>>> abc.pop(-1)                      # to isté ako abc.pop()
    'styri'
>>> abc.pop(0)                       # vyhadzuje prvý prvok
    'raz'
>>> abc
    ['dva', 'tri']
>>> abc.pop(1)
    'tri'
>>> abc.pop(0)
    'dva'
>>> abc.pop(0)
    ...
    IndexError: pop from empty list
>>> abc
    []
```

Posledné volanie metódy `pop()` sa snaží vybrať prvý prvok z prázdneho zoznamu - spôsobilo to vyvolanie správy o chybe.

metóda `remove()`

Volanie metódy:

```
zoznam.remove(hodnota)
```

odoberie zo zoznamu prvý výskyt prvku s danou hodnotou - zoznam sa takto skráti o 1. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia nič nevracia (teda vracia `None`). Ak sa daná hodnota v zozname nenachádza, funkcia spadne na chybe.

Napríklad:

```
>>> abc = ['raz', 'dva', 'tri', 'dva']
>>> abc.remove('dva')
>>> abc
    ['raz', 'tri', 'dva']
>>> abc.remove('styri')
    ...
    ValueError: list.remove(x): x not in list
```

metóda `sort()`

Volanie metódy:

```
zoznam.sort()
```

zmení poradie prvkov zoznamu tak, aby boli **usporiadané vzostupne** - zoznam takto nemení svoju dĺžku. Táto metóda je **mutable** lebo mení obsah zoznamu. Funkcia nič nevracia (teda vracia `None`). Ak sa prvky v zozname nedajú navzájom porovnávať (napríklad sú tam čísla aj reťazce), funkcia spadne na chybe.

Napríklad:

```
>>> abc = ['raz', 'dva', 'tri', 'styri']
>>> abc.sort()
>>> abc
    ['dva', 'raz', 'styri', 'tri']
>>> post = list(reversed(range(10)))
>>> post
    [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> post.sort()
>>> post
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Častou začiatočníckou chybou býva priradenie výsledku tejto metódy (teda `None`) do premennej, napríklad:

```
>>> abc = ['raz', 'dva', 'tri', 'styri']
>>> abc = abc.sort()                        # vždy vráti None
>>> print(abc)
    None
```

Takto si pokazíme referenciu na pekne utriedený zoznam.

Zoznam ako výsledok funkcie

Prvá funkcia vráti novo vytvorený zoznam rovnakých hodnôt:

```
def urob_zoznam(n, hodnota=0):
    return [hodnota] * n
```

Môžeme to použiť napríklad takto (premenné sme nazvali `pole`, lebo sa to trochu podobá na pascalovské polia):

```
>>> pole1 = urob_zoznam(30)
>>> pole2 = urob_zoznam(25, None)
>>> pole3 = urob_zoznam(3, 'Python')
>>> pole1
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> pole2
    [None, None, None, None, None, None, None, None, None, None,
     None, None, None, None, None, None, None, None, None, None,
     None, None, None, None, None]
>>> pole3
    ['Python', 'Python', 'Python']
```

Ďalšia funkcia vytvorí z danej postupnosti nový zoznam:

```
def zoznam(postupnost):
    vysl = []
    for prvok in postupnost:
        vysl.append(prvok)
    return vysl
```

Uvedomte si, že to robí skoro to isté ako volanie funkcie `list()`:

```
>>> py = zoznam('Python')
>>> py
    ['P', 'y', 't', 'h', 'o', 'n']
>>> cisla = zoznam(range(3, 25, 4))
>>> cisla
    [3, 7, 11, 15, 19, 23]
```

Funkcia `pridaj()` na základe nejakého zoznamu vytvorí nový zoznam, na koniec ktorého pridá nový prvok:

```
def pridaj(zoznam, hodnota):
    return zoznam + [hodnota]
```

Všimnite si, že pôvodný zoznam pritom ostal nezmenený:

```
>>> zoz = ['raz', 'dva', 'tri']
>>> novy = pridaj(zoz, 'styri')
>>> novy
    ['raz', 'dva', 'tri', 'styri']
>>> zoz
    ['raz', 'dva', 'tri']
```

Takejto funkcii budeme hovoriť, že je **immutable**, lebo nemení hodnotu žiadneho predtým existujúceho zoznamu.

Ak by sme túto funkciu zapísali takto:

```
def pridaj1(zoznam, hodnota):
    zoznam.append(hodnota)
    return zoznam
```

Volaním tejto funkcie by sme dostali veľmi podobné výsledky:

```
>>> zoz = ['raz', 'dva', 'tri']
>>> novy = pridaj1(zoz, 'styri')
>>> novy
    ['raz', 'dva', 'tri', 'styri']
>>> zoz
    ['raz', 'dva', 'tri', 'styri']
```

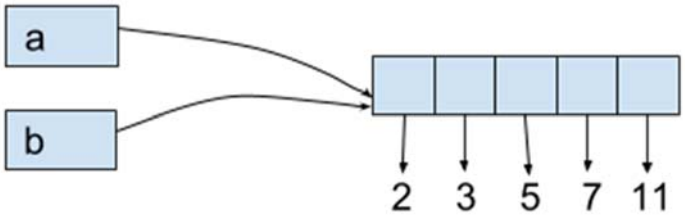
Ale zmenila sa pritom aj hodnota prvého parametra - premennej `zoz` typu zoznam. Táto funkcia je **mutable** - mení svoj parameter `zoz` typu `list`.

Dve premenné referencujú na ten istý zoznam

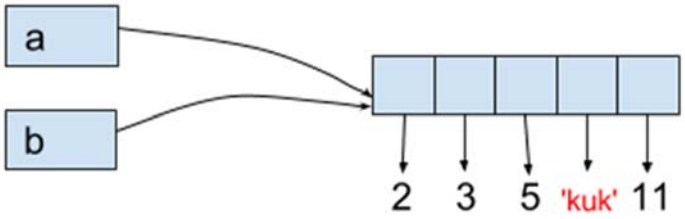
Už sme získali predstavu o tom, že priradenie zoznamu do premennej označuje, že sme v skutočnosti do premennej priradili referenciu na zoznam. Lenže na ten istý zoznam v pamäti môžeme mať viac referencií, napríklad:

```
>>> a = [2, 3, 5, 7, 11]
>>> b = a
>>> b[3] = 'kuk'
>>> a
    [2, 3, 5, 'kuk', 11]
```

Menili sme obsah premennej `b` (zmenili sme jej prvok s indexom 3), ale tým sa zmenil aj obsah premennej `a`. Totiž obe premenné referencujú na ten istý zoznam:



Keď teraz meníme obsah premennej `b` (ale len pomocou **mutable** operácií!), zmení sa aj obsah premennej `a`:



Zhrňme

Vkladanie do zoznamu

Videli sme viac rôznych spôsobov, ako môžeme pridať jednu hodnotu do zoznamu. Vkladanie nejakej `hodnoty` pred prvok s indexom `i`:

- pomocou rezu (**mutable**):

```
zoznam[i:i] = [hodnota]
```

- pomocou metódy `insert()` (**mutable**):

```
zoznam.insert(i, hodnota)
```

- ak `i` == `len(zoznam)`, pridávame na koniec (za posledný prvok), môžeme použiť metódu `append()` (**mutable**):

```
zoznam.append(hodnota)
```

to isté dosiahneme aj takto (**mutable**):

```
zoznam += [hodnota]
```

Vo vašich programoch použijete ten zápis, ktorý sa vám bude najlepšie hodiť, ale zápis s rezom `zoznam[i:i]` je najmenej čitateľný a používa sa veľmi zriedkavo.

- zrejme funguje aj (**immutable**):

```
zoznam = zoznam[:i] + [hodnota] + zoznam[i:]
```

toto priradenie nemodifikuje pôvodný zoznam, ale vytvára nový s pridanou hodnotou

Vyhadzovanie zo zoznamu

Aj vyhadzovanie prvku zo zoznamu môžeme robiť viacerými spôsobmi. Ak vyhadzujeme prvok na indexe `i`, môžeme zapísať:

- pomocou rezu (**mutable**):

- `zoznam[i:i+1] = []`

- pomocou príkazu `del` (**mutable**):

- `del zoznam[i]`

- pomocou metódy `pop()`, ktorá nám aj vráti vyhadzovanú hodnotu (**mutable**):

- `hodnota = zoznam.pop(i)`

- veľmi neefektívne pomocou metódy `remove()`, ktorá ako parameter očakáva nie index ale vyhadzovanú hodnotu (**mutable**):

- `zoznam.remove(zoznam[i])`

  tento spôsob je veľmi **neefektívny** (zbytočne sa hľadá prvok v zozname) a okrem toho niekedy môže vyhodiť nie `i`-ty prvok, ale prvok s rovnakou hodnotou, ktorý sa v zozname nachádza skôr, ako na indexe `i`.

- zrejme funguje aj (**immutable**):

- `zoznam = zoznam[:i] + zoznam[i+1:]`

  toto priradenie nemodifikuje pôvodný zoznam, ale vytvára nový bez prvku s daným indexom

Vyhodenie všetkých prvkov zo zoznamu

- najjednoduchší spôsob (**immutable**):

- `zoznam = []`

  môžeme použiť len vtedy, keď nepotrebujeme uchovať referenciu na zoznam - toto priradenie nahradí momentálnu referenciu na zoznam referenciou na úplne nový zoznam; ak to použijeme vo vnútri funkcie, stratí sa tým referencia na pôvodný zoznam

Ďalšie spôsoby uchovávajú referenciu na zoznam:

- všetky prvky zoznamu postupne vyhodíme pomocou while-cyklu (**mutable**):

```
while zoznam:
    zoznam.pop()
```

  toto je zbytočne veľmi neefektívne riešenie

- priradením do rezu (**mutable**):

- `zoznam[:] = []`

  je ťažšie čitateľné a menej pochopiteľné riešenie

- metódou `clear()` (**mutable**):

- `zoznam.clear()`

  je asi najčitateľnejší zápis

Vytvorenie kópie zoznamu

Ak potrebujeme vyrobiť kópiu celého zoznamu, dá sa to urobiť:

- pomocou cyklu:

```
kopia = []
for prvok in zoznam:
    kopia.append(prvok)
```

- môžeme využiť aj rez:

```
kopia = zoznam[:]
```

- keďže funguje funkcia `list()`, môžeme zapísať:

```
kopia = list(zoznam)
```

Cvičenia

L.I.S.T.

- riešenia **aspoň 12 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- používaj len konštrukcie z doterajších prednášok
- pozri si Riešenie úloh 8. cvičenia

1. Do zoznamových premenných `den` a `day` priraď 7 názvov dní v týždni v slovenčine a v angličtine (`'pondelok'`, …):

```
2. den = ...
3. day = ...
```

   Teraz napíš príkazy, ktoré vypíšu tieto dva zoznamy každý do jedného riadku (slová v riadku oddeľ medzerami):

```
pondelok utorok ...
Monday Tuesday ...
```

   Ďalej napíš for-cyklus, pomocou ktorého sa do 7 riadkov postupne vedľa seba vypíšu slovenský a anglický názov dňa v týždni. Použi obe premenné `den` a `day`:

```
pondelok Monday
utorok Tuesday
...
```

2. V nejakej obci je jediná ulica, na ktorej je `n` domov. Na miestnom úrade majú v zozname (typu `list`) pre každý dom zaznačený počet obyvateľov. Pomocou len štyroch `print` vypíš, koľko obyvateľov žije v celej obci, koľko domov je neobývaných, aký je maximálny počet obyvateľov v dome a v koľkých domoch býva tento maximálny počet. Napríklad, pre:

```
3. domy = [4, 2, 0, 5, 0, 1, 5, 4]
```

   vypíše:

```
počet obyvateľov je 21
neobývaných domov je 2
maximálny počet obyvateľov v dome je 5
počet maximálnych domov je 2
```

Teraz do premennej `domy` priraď nejaký iný zoznam aspoň 15 čísel z intervalu `<0, 10>` a opäť zisti všetky predchádzajúce informácie.

3. V premennej `recept` je zoznam, ktorého dĺžka je násobkom `3`. Napríklad:

```
recept = ['cukor', 100, 'g', 'vajíčka', 5, 'ks', 'mlieko', 2, 'dcl']
```

Napíš funkciu `vypis_recept(zoznam)`, ktorá takýto zoznam vypíše (pomocou `print`) do viacerých riadkov po troch, napríklad:

```
>>> vypis_recept(recept)
    cukor 100 g
    vajíčka 5 ks
    mlieko 2 dcl
```

Funkcia nemodifikuje vstupný `zoznam`.

Svoje riešenie zapíš do troch riadkov:

```
def vypis_recept(zoznam):
    for ...
        print(...
```

4. Napíš funkciu `vypis(zoznam, pocet)`, ktorá prvky daného zoznamu vypíše tak, že v každom riadku (možno okrem posledného) vypíše presne zadaný `pocet` prvkov zoznamu. Funkcia nemodifikuje vstupný `zoznam`. Napríklad:

```
>>> zoz = list(range(1, 19))
>>> vypis(zoz, 4)
    1 2 3 4
    5 6 7 8
    9 10 11 12
    13 14 15 16
    17 18
>>> zoz
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
>>> vypis(list('Python'), 2)
    P y
    t h
    o n
```

5. Napíš funkciu `najdlhsie(zoznam)`, ktorá zo zoznamu slov vráti (`return`) najdlhšie slovo. Funkcia nemodifikuje vstupný `zoznam` a nič nevypisuje. Napríklad:

```
>>> zoz = ['prvy', 'druhy', 'treti', 'stvrty', 'piaty']
>>> naj = najdlhsie(zoz)
>>> naj
    'stvrty'
>>> najdlhsie(den)          # den z prvej úlohy
    'pondelok'
```

6. Napíš funkciu `start(zoznam, n)`, ktorá z daného zoznamu znakových reťazcov vyrobí (a vráti) nový, v ktorom sa z každého reťazca ponechá len prvých `n` znakov. Funkcia nemodifikuje vstupný `zoznam` a nič nevypisuje. Napríklad:

```
>>> mesiace = ['januar', 'februar', 'marec', 'april', 'maj',
               'jun', 'jul', 'august', 'september',
               'oktober', 'november', 'december']
>>> zoz3 = start(mesiace, 3)
>>> zoz3
    ['jan', 'feb', 'mar', 'apr', 'maj', 'jun', 'jul', 'aug', 'sep', 'okt', 'nov', 'dec']
>>> start(mesiace, 1)
    ['j', 'f', 'm', 'a', 'm', 'j', 'j', 'a', 's', 'o', 'n', 'd']
>>> start(den, 2)             # den z prvej úlohy
    ['po', 'ut', 'st', 'št', 'pi', 'so', 'ne']
```

7. Napíš funkciu `cele(zoznam)`, ktorá z daného zoznamu vytvorí (a vráti) nový, v ktorom sa každý prvok previedol (pomocou funkcie `int()`) na celočíselnú hodnotu. Funkcia nemodifikuje vstupný `zoznam` a nič nevypisuje. Napríklad:

```
>>> x = cele([3.14, '14', 53])
>>> x
    [3, 14, 53]
>>> cele(list(str(2**20)))
    [1, 0, 4, 8, 5, 7, 6]
```

8. Napíš funkciu `vzostupne(zoznam)`, ktorá zistí, či sú prvky vstupného zoznamu usporiadané vzostupne. Funkcia vráti (`return`) `True` alebo `False`, nemodifikuje vstupný `zoznam` a nič nevypisuje. Nepoužívaj `sort` ani `sorted`. Napríklad:

```
>>> vzostupne([1, 5, 5, 8, 100])
    True
>>> vzostupne(['pyton', 'python', 'pytliak'])
    False
```

9. Napíš funkciu `rozdel(retazec)`, ktorá daný reťazec rozdelí na podreťazce a tieto vráti ako zoznam reťazcov. Podreťazce sú navzájom oddelené aspoň jednou medzerou alebo znakom `'\n'`. Funkcia nič nevypisuje. Nepoužívaj `split`. Napríklad:

```
>>> zoz = rozdel('  jeden   dva tri')
>>> zoz
    ['jeden', 'dva', 'tri']
>>> rozdel(input('zadaj cisla:'))
    zadaj cisla:11 234 5 5789
    ['11', '234', '5', '5789']
>>> rozdel('ah\noj\n')
    ['ah', 'oj']
```

10. Napíš funkciu `kresli_text(zoznam)`, v ktorej parameter `zoznam` je troj alebo štvor prvkový zoznam. Tento zoznam má na začiatku dve celé čísla (označujú súradnice `x` a `y`), tretím prvkom je znakový reťazec. Štvrtým prvkom zoznamu je reťazec, ktorý špecifikuje font. Keď tento prvok chýba, použi `'arial 20'`. Funkcia vypíše do grafickej plochy na zadané súradnice daný reťazec s daným fontom. Napríklad:

```
import tkinter

```

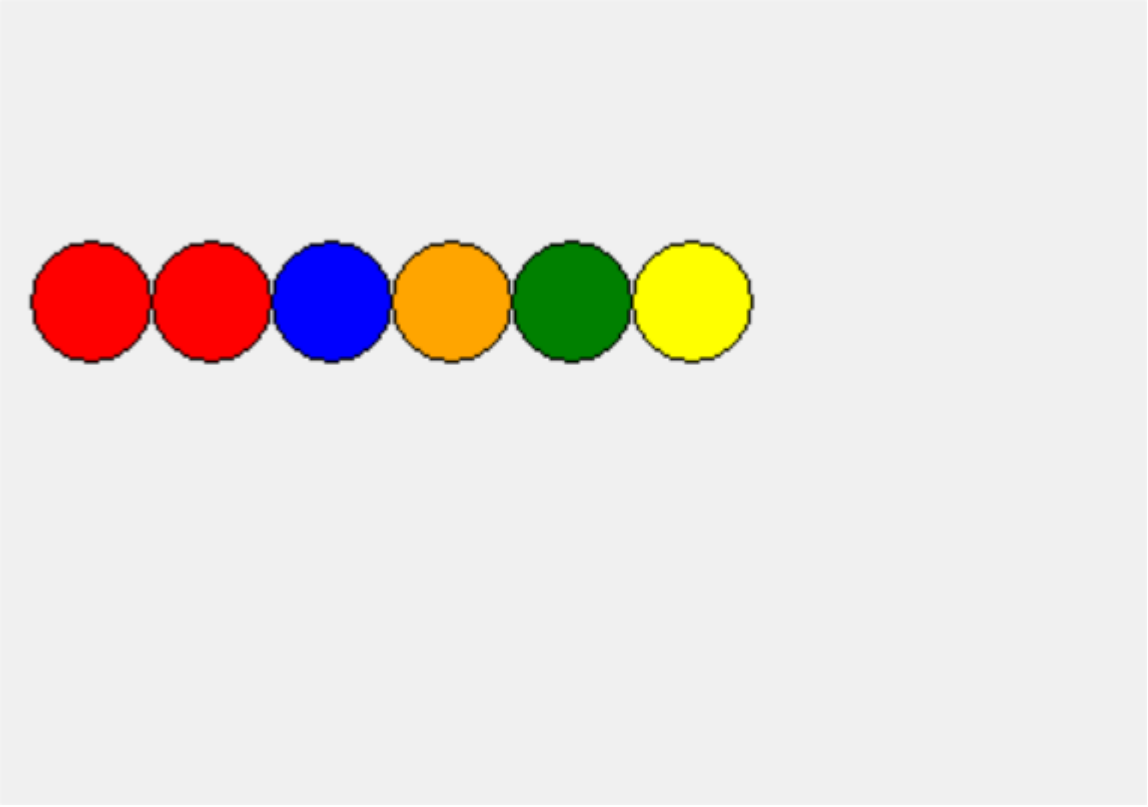
```
def kresli_text(zoznam):
    ...

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

zoz = [200, 100, 'PYTHON']
kresli_text(zoz)
kresli_text([200, 150, 'programovanie', 'consolas 35'])
kresli_text([200, 60, 'najlepšie je', 'tahoma 15'])

tkinter.mainloop()
```

vypíše tri texty: jeden na súradnice `(200, 100)`, druhý na `(200, 150)` a tretí `(200, 60)`:

11. Napíš funkciu `kresli(r, zoznam)`, ktorá z daného zoznamu farieb nakreslí rad farebných kruhov, Každý kruh má polomer `r`. Funkcia nemodifikuje vstupný `zoznam`. Napríklad:

```
import tkinter

def kresli(r, zoznam):
    ...

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

kresli(20, ['red', 'red', 'blue', 'orange', 'green', 'yellow'])

tkinter.mainloop()
```

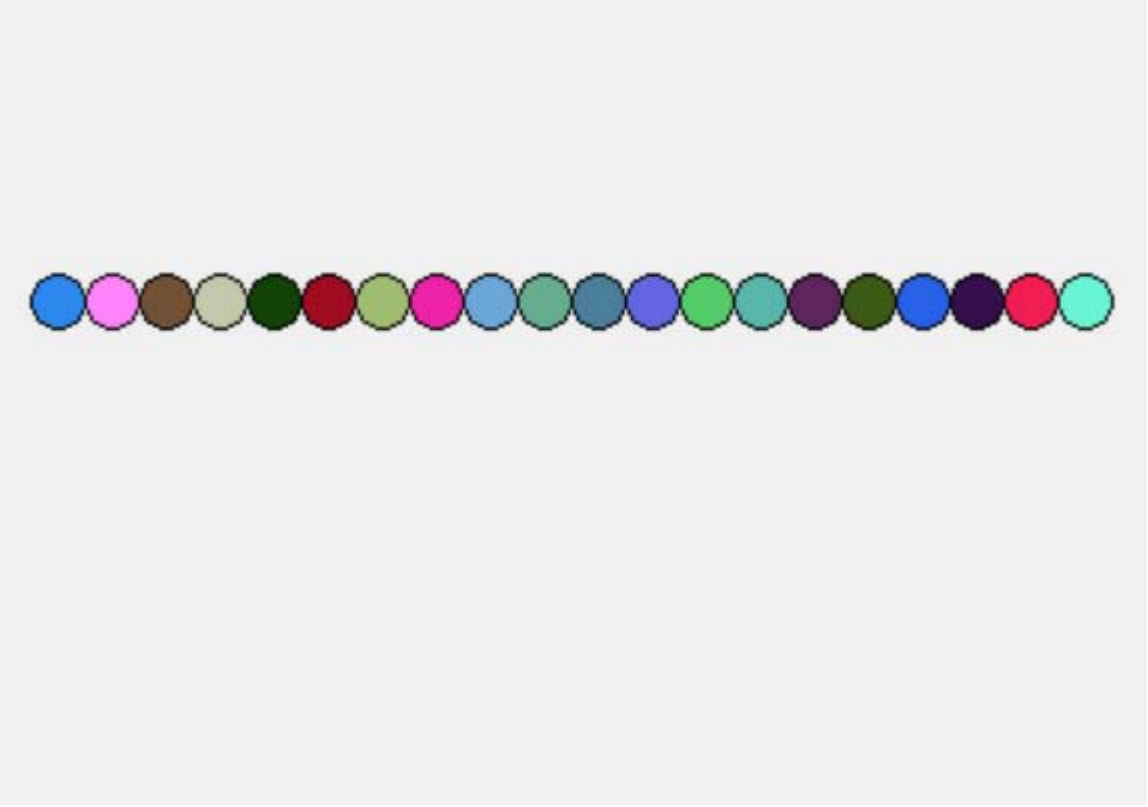
nakreslí:



12. Napíš funkciu `nahodne_farby(n)`, ktorá vygeneruje a vráti `n`-prvkový zoznam náhodných farieb. Tento zoznam môžeme poslať do funkcie `kresli` z predchádzajúcej úlohy. Napríklad:

```
import tkinter
import random

def nahodne_farby(n):
    ...

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

kresli(9, nahodne_farby(20))

tkinter.mainloop()
```

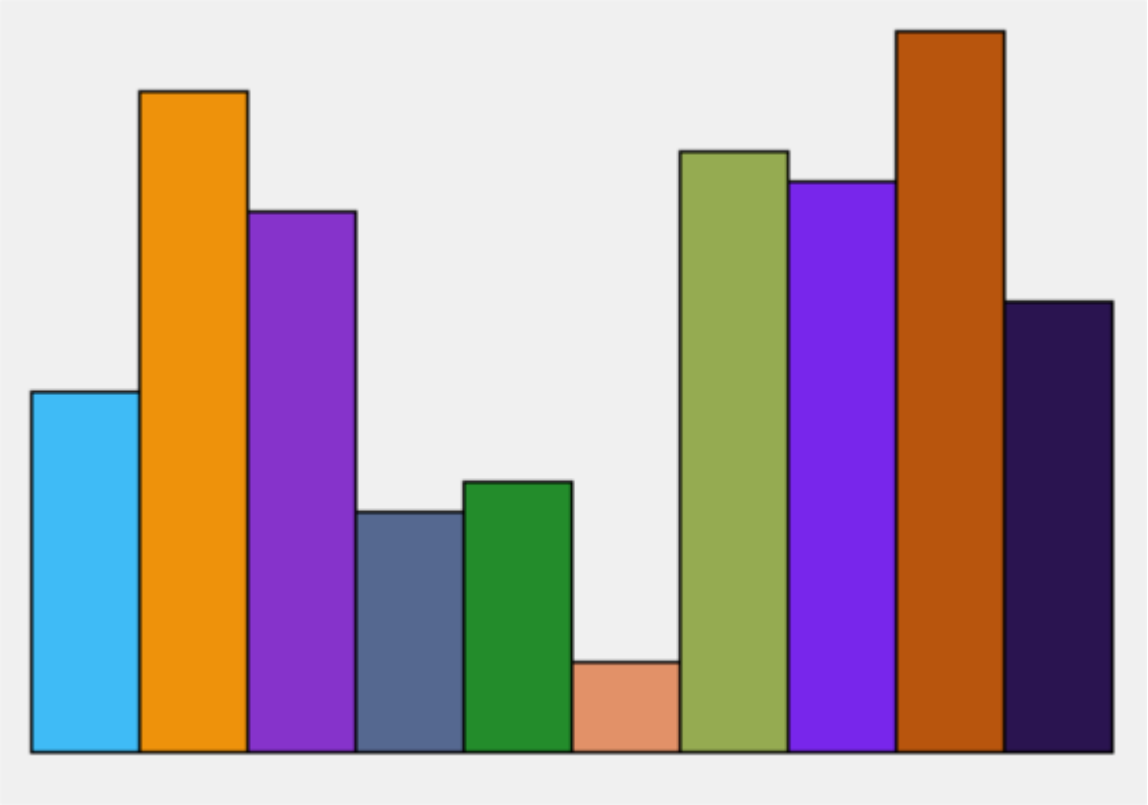
nakreslí:



13. Napíš funkciu `histogram(zoznam)`, ktorá z daného zoznamu čísel nakreslí stĺpcový diagram (obdĺžniky rovnakej šírky a výšky podľa číselnej hodnoty zo zoznamu). Obdĺžniky budú mať takú šírku, aby čo najlepšie vyplnili šírku plochy `360` a budú mať náhodne zafarbenú výplň. Funkcia nemodifikuje vstupný `zoznam`. Môžeš počítať s tým, že všetky hodnoty v zozname nie sú väčšie ako `240`. Napríklad:

```
14. import tkinter
15. import random
16. 
17. def histogram(zoznam):
18.     ...
19. 
20. canvas = tkinter.Canvas()
21. canvas.pack()
22. 
23. histogram([120, 220, 180, 80, 90, 30, 200, 190, 240, 150])
24. 
25. tkinter.mainloop()
```

nakreslí:



14. Napíš funkciu `zluc(zoz1, zoz2)`, ktorá z daných dvoch zoznamov `zoz1` a `zoz2` vyrobí (a vráti) nový, v ktorom budú prvky z pôvodných dvoch zoznamov na striedačku (najprv prvý prvok z prvého, potom prvý prvok z druhého, potom druhý z prvého, atď.). Funkcia nemodifikuje vstupné zoznamy a nič nevypisuje. Napríklad:

```
>>> zoz = zluc([1, 2, 3, 4, 5, 6, 7], [11, 22, 33])
>>> zoz
    [1, 11, 2, 22, 3, 33, 4, 5, 6, 7]
>>> zluc(list('Python'), list(range(10)))
    ['P', 0, 'y', 1, 't', 2, 'h', 3, 'o', 4, 'n', 5, 6, 7, 8, 9]
```

15. Napíš funkciu `spoj(zoznam)`, ktorá z daného zoznamu vyrobí (a vráti) znakový reťazec. V tomto reťazci budú všetky prvky zoznamu oddelené medzerami. Funkcia nemodifikuje vstupný `zoznam` a nič nevypisuje. Napríklad:

```
>>> ret = spoj(['vysledok', 23, '+', 321, 'je', 23+321])
>>> ret
    'vysledok 23 + 321 je 344'
>>> spoj(list(range(100, 200, 13)))
    '100 113 126 139 152 165 178 191'
```

16. Napíš funkciu `replace(zoznam, co, zaco)`, ktorá v danom zozname všetky výskyty hodnoty `co` nahradí hodnotou `zaco`. Funkcia nič nevracia ani nevypisuje, len modifikuje obsah zoznamu. Napríklad:

```
>>> zoz = [12, 13, 14, 13, 11, 14, 15, 13]
>>> replace(zoz, 13, 'x')
>>> zoz
    [12, 'x', 14, 'x', 11, 14, 15, 'x']
>>> zoz = [1, 2] * 10
>>> replace(zoz, 1, 9)
```

```
>>> zoz
    [9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2]
```

17. Napíš funkciu `dvojice(zoznam)`, ktorá v danom zozname spočíta (alebo zreťazí) dvojice prvkov. Teda spočíta/zreťazí (operácia `+`) prvý a druhý prvok a oba nahradí týmto súčtom, potom to isté s tretím a štvrtým, potom piatym a šiestym, atď. Funkcia nič nevracia ani nevypisuje, len modifikuje obsah zoznamu. Napríklad:

```
>>> z = list('programovanie')
>>> dvojice(z)
>>> z
    ['pr', 'og', 'ra', 'mo', 'va', 'ni', 'e']
>>> z = list(range(1, 11))
>>> z
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> dvojice(z)
>>> z
    [3, 7, 11, 15, 19]
>>> dvojice(z)
>>> z
    [10, 26, 19]
```

18. Napíš funkciu `fibonacci(n)`, ktorá vyrobí (vráti) `n`-prvkový zoznam prvých `n` prvkov **fibonacciho postupnosti**. Funkcia nič nevypisuje. Napríklad:

```
>>> fib = fibonacci(15)
>>> fib
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
>>> fibonacci(1)
    [0]
```

19. Napíš funkciu `rozklad(x)`, ktorá rozloží dané celé číslo `x` na prvočinitele (súčin prvočísel). Výsledkom funkcie bude zoznam týchto prvočísel (prvočiniteľov). Funkcia nič nevypisuje. Napríklad:

```
>>> r = rozklad(478632)
>>> r
    [2, 2, 2, 3, 7, 7, 11, 37]
>>> rozklad(43)
    [43]
```

20. Napíš funkciu `postupnost(x)`, ktorá vygeneruje (vráti ako výsledok) takúto postupnosť celých čísel: postupnosť začína hodnotou `x`; ďalší prvok sa vždy vypočíta podľa predchádzajúceho tak, že ak je párny, vydelí sa dvomi, inak sa vynásobí tromi a pripočíta jedna. Generovanie postupnosti končí, keď sa objaví číslo `1` (bude posledným prvkom postupnosti). Funkcia nič nevypisuje, len vráti (`return`) nejaký zoznam čísel. Napríklad:

```
>>> p = postupnost(3)
>>> p
    [3, 10, 5, 16, 8, 4, 2, 1]
>>> postupnost(1)
    [1]
>>> postupnost(7)
    [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

21. Napíš funkciu `vyhod_neparne(zoznam)`, ktorá z daného zoznamu celých čísel vyhodí tie prvky, ktoré sú nepárne. Funkcia nič nevracia ani nevypisuje, len modifikuje hodnotu zoznamu. Napríklad:

```
>>> x = [2, 3, 4, 5, 7, 8, 10, 11, 13, 15, 17]
>>> vyhod_neparne(x)
>>> x
    [2, 4, 8, 10]
>>> x = list(range(1, 100, 2))
>>> vyhod_neparne(x)
>>> x
    []
```

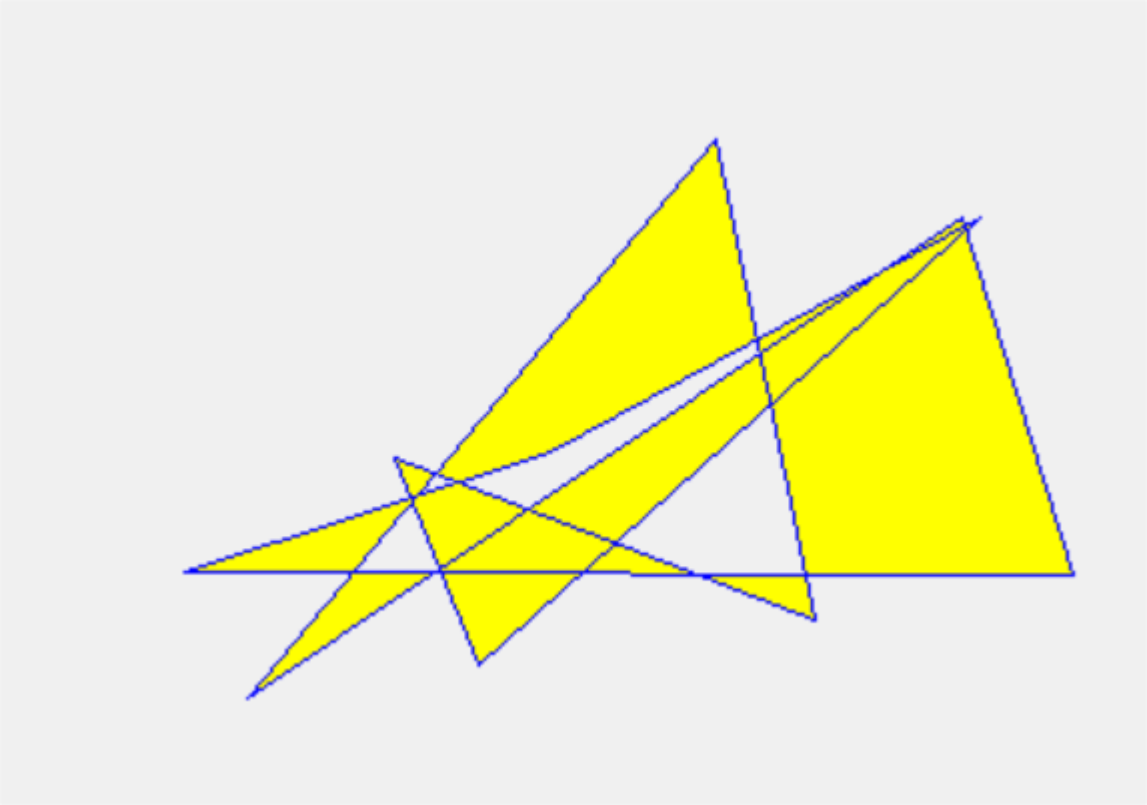
22. Napíš funkciu `vloz_za_kazde(zoznam, hodnota)`, ktorá do daného zoznamu vloží za každý prvok danú hodnotu. Funkcia nič nevracia ani nevypisuje, len modifikuje hodnotu zoznamu. Napríklad:

```
>>> z = list('Python')
>>> vloz_za_kazde(z, 99)
>>> z
    ['P', 99, 'y', 99, 't', 99, 'h', 99, 'o', 99, 'n', 99]
```

23. Napíš funkciu `nahodne_body(n)`, ktorá vygeneruje (vráti) zoznam čísel dĺžky `2*n` v tvare *[x, y, x, y, …]*. Tieto čísla sú súradnicami `n` náhodných vrcholov štvorca v grafickej ploche. `x`-ové súradnice generuj z intervalu `<10, 370>` a `y`-ové z intervalu `<10, 250>`. Ak by si tento zoznam poslal ako parameter do grafického príkazu `canvas.create_polygon`, dostaneš vyfarbený náhodný `n`-uholník. Napríklad:

```
import tkinter
import random

def nahodne_body(n):
    ...

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

canvas.create_polygon(nahodne_body(10), fill='yellow', outline='blue')

tkinter.mainloop()
```

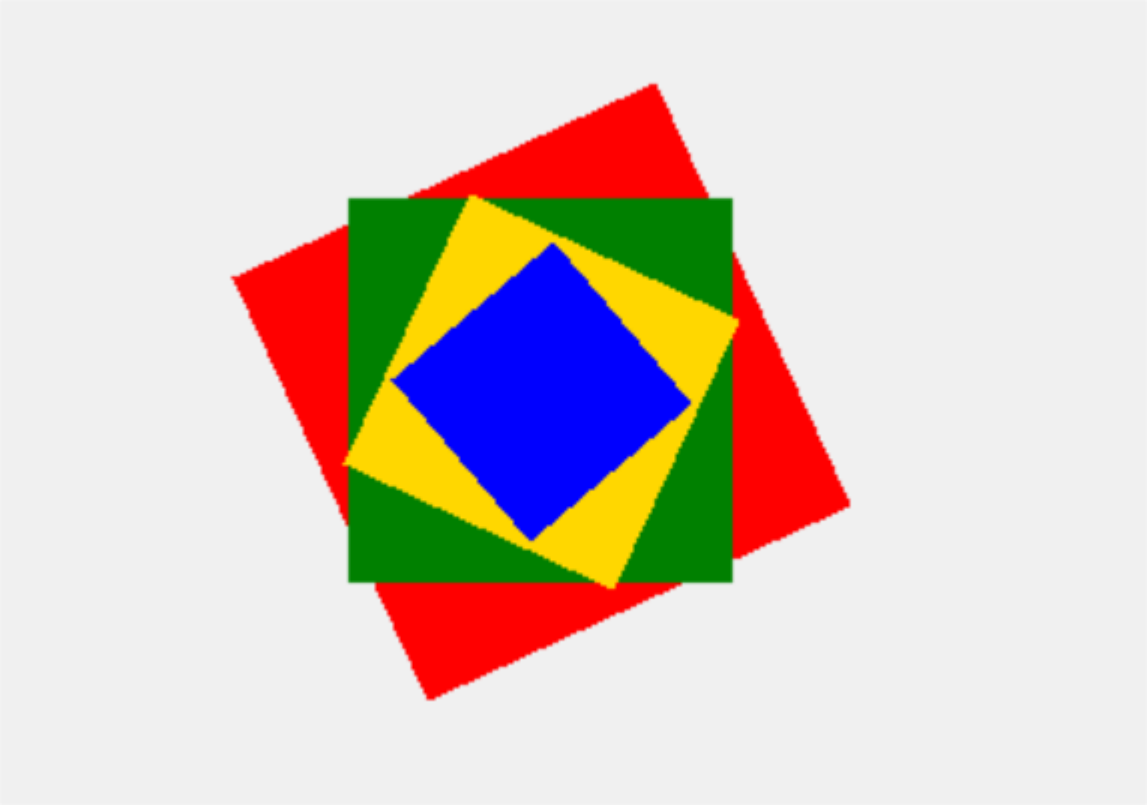
nakreslí:



24. Napíš funkciu `stvorec(x, y, r, uhol)`, ktorá vytvorí (vráti) 8-prvkový zoznam čísel v tvare *[x, y, x, y, x, y, x, y]*. Tieto čísla sú súradnicami štyroch vrcholov štvorca v grafickej ploche a tieto vrcholy ležia na kružnici so stredom `(x, y)` s polomerom `r`. Vrcholy štvorca budú na kružnici natočené o daný `uhol`. Ak by si tento zoznam poslal ako parameter do grafického príkazu `canvas.create_polygon`, dostaneš vyfarbený natočený štvorec. Napríklad:

```
import tkinter
from math import sin, cos, radians

def stvorec(x, y, r, uhol):
    ...

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

canvas.create_polygon(stvorec(180, 130, 110, 20), fill='red')
canvas.create_polygon(stvorec(180, 130, 90, 45), fill='green')
canvas.create_polygon(stvorec(180, 130, 70, 70), fill='gold')
canvas.create_polygon(stvorec(180, 130, 50, 95), fill='blue')

tkinter.mainloop()
```

nakreslí:



4. Týždenný projekt

L.I.S.T.

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

Napíš pythonovský skript, ktorý bude definovať túto funkciu:

```
vypis(meno_suboru, sirka, zarovnat=True, slovo='')
```

Funkcia `vypis()` dostáva ako prvý parameter meno textového súboru a druhým parametrom je celé číslo, ktoré udáva šírku výpisu. Funkcia tento súbor prečíta a celý ho vypíše do textovej plochy (do konzoly) tak, že bude zarovnaný na danú šírku.

Textový súbor sa skladá z **odsekov**, ktoré sa skladajú zo **slov**. Odseky sú navzájom oddelené aspoň jedným prázdnym riadkom. Slová v odseku sú navzájom oddelené aspoň jednou medzerou alebo koncom riadka.

Napríklad, `"subor1.txt"` sa skladá z týchto riadkov:

```
Ján Botto:
  Žltá ľalija


Stojí, stojí mohyla.

Na mohyle   zlá    chvíľa,
na mohyle tŕnie,   chrastie
 a v tom tŕní, chrastí rastie,
  rastie, kvety rozvíja
jedna   žltá   ľalija.
```

```
  Tá ľalija smutno vzdychá:

  Hlávku moju tŕnie pichá
 a  nožičky  oheň  páli –
pomôžte mi v mojom žiali!
```

Tento súbor obsahuje 5 „odsekov“, pričom najkratší je druhý a má 3 „slová“. Najdlhší je tretí odsek má 20 slov.

Volanie `vypis('subor1.txt', 20)` vypíše:

```
Ján    Botto:   Žltá
ľalija

Stojí, stojí mohyla.

Na     mohyle    zlá
chvíľa,   na  mohyle
tŕnie,  chrastie a v
tom   tŕní,  chrastí
rastie,      rastie,
kvety  rozvíja jedna
žltá ľalija.

Tá   ľalija   smutno
vzdychá:

Hlávku   moju  tŕnie
pichá a nožičky oheň
páli  – pomôžte mi v
mojom žiali!
```

Pričom `vypis('subor1.txt', 60)` vypíše:

```
Ján Botto: Žltá ľalija

Stojí, stojí mohyla.

Na  mohyle  zlá  chvíľa,  na  mohyle tŕnie, chrastie a v tom
tŕní,  chrastí  rastie,  rastie,  kvety  rozvíja  jedna žltá
ľalija.

Tá ľalija smutno vzdychá:

Hlávku  moju  tŕnie pichá a nožičky oheň páli – pomôžte mi v
mojom žiali!
```

Všimni si, že všetky riadky v odseku okrem posledného sú zarovnané vpravo na zadanú šírku, pričom, ak by bola dĺžka takéhoto riadka kratšia ako zadaná šírka, medzi slová sú rovnomerne vložené medzery. Ak nejaký riadok obsahuje len jedno slovo, tak ani tento sa nezarovnáva na pravý okraj. Ak je nejaké slovo dlhšie ako zadaná šírka, tak toto slovo sa nerozdeľuje do viacerých riadkov, ale bude v riadku výpisu jediné.

Funkcia `vypis()` má ešte ďalšie dva parametre, ktoré majú určené aj náhradné hodnoty:

- parameter `zarovnat` s hodnotou `True` pracuje tak, ako bolo popísané vyššie, teda všetky riadky všetkých odsekov (okrem posledných a jednoslovných) zarovnáva na pravý okraj.
  Hodnota `False` označuje, že odseky sa na pravý okraj nezarovnávajú, ale text ostáva „zubatý“.
  Napríklad, volanie `vypis('subor1.txt', 45, False)` vypíše:

```
Ján Botto: Žltá ľalija

Stojí, stojí mohyla.
```

```
Na mohyle zlá chvíľa, na mohyle tŕnie,
chrastie a v tom tŕní, chrastí rastie,
rastie, kvety rozvíja jedna žltá ľalija.

Tá ľalija smutno vzdychá:

Hlávku moju tŕnie pichá a nožičky oheň páli –
pomôžte mi v mojom žiali!
```

- ďalší parameter `slovo` s hodnotou `''` označuje, že treba spracovať (zarovnať, alebo nezarovnať) všetky odseky v súbore. Iná hodnota označuje, že chceme spracovávať len tie odseky, ktoré obsahujú toto konkrétne slovo. Napríklad, volanie `vypis('subor1.txt', 45, True, 'kvety')` vypíše len tento jeden odsek:

```
Na   mohyle   zlá   chvíľa,   na   mohyle   tŕnie,
chrastie   a   v   tom   tŕní,   chrastí   rastie,
rastie, kvety rozvíja jedna žltá ľalija.
```

Tvoj odovzdaný program s menom `riesenie.py` musí začínať tromi riadkami komentárov (s tvojim menom a dátumom):

```
# 4. zadanie: zarovnaj
# autor: Janko Hraško
# datum: 29.10.2021
```

Projekt `riesenie.py` odovzdávaj na úlohový server https://list.fmph.uniba.sk/ najneskôr do 23:00 **29. októbra**, kde ho môžeš nechať otestovať. Testovač bude spúšťať tvoju funkciu s rôznymi textovými súbormi, ktoré si môžeš stiahnuť z L.I.S.T.u. Odovzdať projekt aj ho testovať môžeš ľubovoľný počet krát. Môžeš zaň získať **5 bodov**.