UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE

FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

PROGRAMOVANIE V JAZYKU PYTHON

Vybrané kapitoly v praktických úlohách

Viera Michaličková

NITRA 2023

PROGRAMOVANIE V JAZYKU PYTHON: VYBRANÉ KAPITOLY V PRAKTICKÝCH ÚLOHÁCH

VIERA MICHALIČKOVÁ

2023

Programovanie v jazyku Python: Vybrané kapitoly v praktických úlohách

Edícia Prírodovedec č. 822

Autor:

PaedDr. Viera Michaličková, PhD.

Recenzent:

doc. RNDr. Gabriela Lovászová, PhD.

Publikácie je podporená z projektu 001UKF-2-1/2022 Zvyšovanie kvality prípravy budúcich učiteľov matematiky, fyziky, chémie, informatiky, anglického jazyka, slovenského jazyka a techniky formou doplňujúceho pedagogického štúdia a rozširujúceho štúdia na UKF v Nitre.

ISBN 978-80-558-2056-9

OBSAH

ÚVOD

Toto skriptum je primárne určené študentom rozširujúceho štúdia informatiky a študentom bakalárskeho štúdia učiteľstva informatiky. Obsahuje vybrané, pokročilé témy z úvodného kurzu programovania v jazyku Python spracované vo forme sérií praktických úloh:

- *Funkcie,*
- *Znakové reťazce,*
- *Chyby v programe, výnimky,*
- *Zoznamy, n-tice* a
- *Textové súbory*.

Každá z tém začína stručným, motivačným úvodom do problematiky. Nasledujú pracovné listy s úlohami. Téma je ukončená zhrnutím podstatných poznatkov, súvislostí, zápisov a postupov, ktoré vyplývajú z riešenia úloh z pracovných listov. Dôležitou súčasťou publikácie je aj kľúč s riešeniami úloh a doplňujúcimi vysvetleniami, vďaka ktorému môžu študenti tento materiál využívať aj pre samoštúdium, resp. v domácej príprave na vyučovanie.

Naším hlavným cieľom je, aby sa študenti učili programovanie aktívne, riešením zmysluplných problémov, vytváraním vlastných programov. Veľký dôraz kladieme na experimentovanie s príkazmi jazyka Python v interaktívnom móde, porovnávanie rôznych vzorových riešení, skutočné porozumenie programátorským konceptom a efektívne používanie formálnych zápisov programovacieho jazyka. Študentov, budúcich učiteľov informatiky a programovania, tiež chceme viesť k správnemu a uvedomelému používaniu odbornej terminológie.

Úlohy pripravené v pracovných listoch vytvárajú dostatočný priestor na získanie vedomostí a zručností na úrovni postačujúcej na vyučovanie týchto tém na strednej škole.

TÉMA 1: FUNKCIE

Pri programovaní v jazyku Python sa nezaobídeme bez funkcií. Mnohé funkcie riešiace typické a často sa opakujúce praktické problémy sú k dispozícii v štandardnej knižnici. Naprogramoval ich niekto skúsenejší za nás a môžeme ich hneď alebo po importovaní príslušného modulu používať (volať) vo svojom programe. Často však budeme potrebovať napísať vlastnú funkciu. Napr. keď si uvedomíme, že na viacerých miestach programu je nutné zabezpečiť vykonanie úplne rovnakej postupnosti príkazov. Nebudeme predsa písať viackrát to isté. Okrem toho, prípadnú chybu by bolo potrebné opravovať na viacerých miestach. Funkcie môžu mať parametre, vďaka čomu sa dá algoritmus v nich naprogramovaný aplikovať na rôzne vstupné údaje. Obvykle je tiež rozumné rozložiť problém na menšie, jednoduchšie čiastkové problémy (podproblémy) a ich riešenia naprogramovať v samostatných funkciách (podprogramoch). Takýto program bude prehľadnejší, zrozumiteľnejší, ľahšie udržiavateľný.

PRACOVNÝ LIST Č. 1

Úloha 1: Vysvetlite, čo je výsledkom volania nasledujúcich funkcií. Ktoré z nich vracajú ako výsledok výpočtu nejakú hodnotu? Akého je táto hodnota typu? Ktoré z nich majú vstupné parametre? Akého typu sú tieto parametre?

```
import random
meno = input('Tvoje meno:')
d = len(meno)
print('Tvoje meno ma', d,' pismen.')
print('Prazdniny su za dverami, hura!')
print()
x = 123.456789
print(round(x, 2))
a = random.randint(10, 20)
print(a)
b = int(input('b: '))
print(abs(b))
```

Úloha 2: V uvedených príkladoch funkcií označte: *hlavičku funkcie, názov funkcie, parametre funkcie, telo funkcie, miesto volania funkcie.*

```
def pozdrav(m):
  print('Ahoj ' + m + '!', 'Poviem Ti basnicku:')

def basnicka():
  print('Poslali ma nasi k vasim,')
  print('aby prisli vasi k nasim.')
  print('Ak nepridu vasi k nasim,')
  print('Tak nepridu nasi k vasim!')

def koniec():
  print('Prajem Ti pekny den!')

meno = input('Tvoje meno: ')
pozdrav(meno)
basnicka()
koniec()
```

Úloha 3: Opravte chybu v nasledujúcej funkcii a zavolajte ju v hlavnom programe:

```
def vtip
  print('Informatik je človek, ktorý sa v potravinách čuduje,')
  print('prečo nemá kilo mäsa 1024 gramov.')

# hlavný program
```

Úloha 4: Opravte chybu v nasledujúcich funkciách a zavolajte ich v hlavnom programe:

```
def stredny_rod():
print('mesto')
print('srdce')
print('vysvedčenie')
print('dievča')

def muzsky_rod():
  print('chlap')
  print('hrdina')
  print('dub')
  print('stroj')

  def zensky_rod():
    print('žena')
    print('ulica')
    print('dlaň')
    print('kosť')

# hlavný program
print('Poviem Ti vzory podstatných mien.')
rod = input('Vyber si rod (z/m/s): ')
if rod == 'z':
  zensky_rod()
elif rod == 'm':
  muzsky_rod()
elif rod == 's':
  stredny_rod()
else:
  print('chyba na vstupe')
```

Úloha 5: Vo všetkých nasledujúcich programoch očakávame výstup: ABC. Zistite, ktoré z nich naozaj tento výstup vypíšu. Programy, ktoré nevypíšu očakávaný výstup, opravte (nemeňte telá funkcií):

```
# Program 1
def prva():
  print('A')

def druha():
  print('B')

def tretia():
  print('C')


prva()
druha()
tretia()
```

```
# Program 2
def prva():
  print('A')

prva()

def druha():
  print('B')

def tretia():
  print('C')

druha()
tretia()
```

```
# Program 3
def prva():
  print('A')

def druha():
  prva()
  print('B')
  tretia()

def tretia():
  print('C')


tretia()
```

Úloha 6: Upravte program tak, aby pre zadané vstupné hodnoty vypísal výsledky všetkých aritmetických operácií:

```
def aritmeticke_operacie(x, y):
  print('sucet:')
  print('sucin:')
  print('rozdiel:')
  print('podiel:')

# hlavný program
a = int(input('a = '))
b = int(input('b = '))
aritmeticke_operacie()
```

Úloha 7: Porovnajte výsledky jednotlivých volaní funkcie *print()*. Označte všetky *pozičné parametre* a *pomenované parametre*. Ktoré z volaní funkcie *print()* skončí vznikom chyby?

```
X, y = 3, 4
print(x, y)
print(y, x)
print(x, y, end='')
print(x, y, sep='*')
print(sep='*', x, y)
print(x, y, end ='.', sep='+')
```

Úloha 8: Otestujte uvedené programy a porovnajte výsledky získané volaním funkcie *zaramuj()*:

```
# Program 1
def zaramuj(sprava, znak='*'):
  ciara = znak * (len(sprava)+2)
  print(ciara)
  print(znak + sprava + znak)
  print(ciara)

# hlavný program
zaramuj('REKLAMA','o')
#zaramuj('REKLAMA')
#zaramuj()
```

```
# Program 2
def zaramuj(sprava, znak='*', medzera=False):
  ciara = znak * (len(sprava)+2)
  if medzera:
    ciara = ciara + znak*2
    sprava = ' ' + sprava + ' '
  print(ciara)
  print(znak + sprava + znak)
  print(ciara)

# hlavný program
zaramuj('REKLAMA','o')
#zaramuj('REKLAMA')
#zaramuj('REKLAMA', 'x', True)
#zaramuj('REKLAMA', medzera=True, znak='x')
#zaramuj(medzera=True, znak='x', 'REKLAMA')
#zaramuj(medzera=True, znak='x', sprava='REKLAMA')
```

Úloha 1: Nasledujúce funkcie vracajú ako výsledok svojho výpočtu obvod a obsah obdĺžnika s danými dĺžkami strán. Použite tieto funkcie v hlavnom programe a vypíšte získané výsledky na obrazovku:

```
def obvod(a, b):
  return 2 * (a + b)

def obsah(a, b):
  return a * b

# hlavný program
sirka = float(input('zadaj sirku obdlznika:'))
dlzka = float(input('zadaj dlzku obdlznika:'))
```

Úloha 2: Napíšte funkciu, ktorá prevedie výšku zadanú v metroch a centimetroch na centimetre.

```
# ...
print(vyska_v_cm(1,72), 'cm')    # program vypíše 172 cm
print(vyska_v_cm(2,5), 'cm')     # program vypíše 205 cm
print(vyska_v_cm(0,85), 'cm')    # program vypíše 85 cm
```

Úloha 3: Napíšte funkciu *nacitaj_cislo()*, ktorá načíta od používateľa číselnú hodnotu typu *int* a vráti ju ako výsledok volania.

```
# ...
x = nacitaj_cislo()
y = nacitaj_cislo()
print(max(x, y))      # program vypíše väčšie z čísel x, y
```

Úloha 4: Ktorá z uvedených verzií funkcie vracia pre párne čísla *True* a pre nepárne *False*?

```
def je_parne(x):
  if x % 2 == 0:
    return True
  else:
    return False

def je_parne(x):
  return x % 2 == 0
```

Úloha 5: Dni v týždni označme celými číslami 1-7. Logická hodnota *True* znamená, že sme práve na dovolenke. Aké hodnoty vracia funkcia pre jednotlivé prípady vstupných parametrov? Svoju odpoveď overte spustením programu.

```
def budik(den, dovolenka=False):
  if dovolenka:
    if 1 <= den <= 5:
      return '9:00'
    else:
      return 'off'
  if 1 <= den <= 5:
    return '6:30'
  else:
    return '8:00'
```

```
# hlavný program
print(budik(1,True))
print(budik(3))
print(budik(6))
print(budik(7,True))
```

Úloha 6: V nasledujúcom programe potrebujeme pre zadané kladné celé číslo vypočítať ciferný súčet a ciferný súčin. Jedna z funkcií je už naprogramovaná, druhá zatiaľ nie. Je možné program spustiť? Aký význam má príkaz *pass*?

```
def ciferny_sucet(cislo):
  sucet = 0
  while cislo != 0:
    sucet += cislo % 10
    cislo //= 10
  return sucet

def ciferny_sucin(cislo):
  pass

# hlavný program
print(ciferny_sucet(123))
print(ciferny_sucet(55))
```

Úloha 7: Ak je súčet všetkých deliteľov nejakého prirodzeného čísla rovný práve jeho dvojnásobku, nazývame ho dokonalé. Napr. číslo 6 je dokonalé, lebo má delitele 1, 2, 3, 6. Ich súčet je 12 = 2 * 6. Dokončite nasledujúci program tak, aby vypísal všetky dokonalé čísla od 1 po zadanú hranicu *n*:

```
def sucet_delitelov(c):
  s = 1
  if c != 1:
    s += c
  for i in range(2,c//2 + 1):
    if c % i == 0:
      s += i
  return s

def je_dokonale(c):
  pass

# hlavný program
n = int(input('n = '))

for cislo in range(1, n+1):
  if je_dokonale(cislo):
    print(cislo)
```

Úloha 8: V predchádzajúcej úlohe vidíme premenné dvojakého typu. Premenná *n* sa nazýva *globálna* a existuje od momentu vytvorenia až do skončenia programu. Premenné *s*, *i* a *c* vo funkcii *sucet_delitelov()* voláme *lokálne*. Keď funkcia skončí (v tomto prípade vykonaním príkazu *return*), lokálne premenné zaniknú. V nasledujúcom programe otestujte postupne všetky štyri funkcie. Aké hodnoty sa vypíšu na obrazovku? Aký význam má kľúčové slovo *global*?

```
x = 10

def fun1():
  print(x)
```

```
def fun2():
  x = 0
  print(x)

def fun3():
  x = x + 1
  print(x)

def fun4():
  global x
  x = x + 1
  print(x)

# hlavný program
print('pred:', x)
fun1()
#fun2()
#fun3()
#fun4()
print('po:', x)
```

ZHRNUTIE

- Funkcie významne zjednodušujú prácu programátora. Problém si môže rozdeliť na menšie podproblémy a tie vyriešiť v samostatných funkciách. Funkcie navyše možno používať (zavolať) opakovane s rôznymi vstupnými hodnotami.
- Keď chceme definovať funkciu, napíšeme najprv jej hlavičku, potom naprogramujeme jej telo. Hlavička funkcie začína kľúčovým slovom *def*, nasleduje meno funkcie a okrúhle zátvorky *()*. Hlavička funkcie končí dvojbodkou. Príkazy v tele funkcie musia byť odsadené zľava (tvoria jeden blok).
- Ak má funkcia parametre, uvádzame ich v hlavičke v zátvorkách (), oddeľujú sa čiarkami. Parametre uvedené v hlavičke funkcie voláme *formálne*. Zastupujú *skutočné* parametre (konkrétne vstupné hodnoty), ktoré do funkcie posielame pri jej volaní.
- V zápise volania funkcie bez parametrov musíme uvádzať prázdne zátvorky ().
- Niektoré parametre funkcie môžu mať východiskovú (náhradnú) hodnotu. Vo volaní funkcie ich potom nemusíme uvádzať.
- Skutočné parametre vo volaní funkcie uvádzame v poradí, v akom sú uvedené v hlavičke funkcie, alebo použijeme mená parametrov (vtedy na poradí nezáleží).
- Definícia funkcie musí predchádzať jej volaniu. V tele jednej funkcie môžeme zavolať inú funkciu.
- Funkcie môžu vracať ako výsledok svojho volania hodnotu rôzneho typu. Vrátenú hodnotu môžeme vypísať na obrazovku, uložiť do premennej alebo použiť vo výraze.
- Niektoré funkcie nevracajú žiadnu hodnotu, ich volanie predstavuje príkaz.
- Príkaz *return* slúži na vrátenie hodnoty z funkcie. Vo funkcii sa môže nachádzať aj viackrát. Vykonaním príkazu *return* sa výpočet vo funkcii ukončí a spracovanie programu pokračuje za miestom volania funkcie.
- Pomocné premenné používané vo funkciách sa volajú *lokálne*. Existujú len v priebehu vykonávania funkcie.
- Premenné vytvorené mimo tiel funkcií sa nazývajú *globálne*. Od momentu vytvorenia ich majú k dispozícii všetky funkcie. V pamäti existujú až do skončenia programu.

- Ak má lokálna premenná vo funkcii rovnaké meno ako niektorá globálna premenná, vo funkcii túto globálnu premennú zatieni.
- Ak chceme vo funkcii meniť hodnotu premennej, ktorá je globálna, musí byť označená kľúčovým slovom *global* (inak bude považovaná za lokálnu premennú).
- Príkazom *pass* môžeme nahradiť dočasne chýbajúce telo funkcie.

TÉMA 2: ZNAKOVÉ REŤAZCE

Znakový reťazec (typ *str,* z angl. *string*) patrí k štruktúrovaným údajovým typom. Nejde totiž o jedinú, ďalej nedeliteľnú hodnotu (ako je to v prípade hodnôt typu *int*, *float* alebo *bool*), ale o postupnosť znakov, z ktorých je reťazec zložený. Ku znakom reťazca môžeme pristupovať prostredníctvom indexov (poradových čísel vyjadrujúcich pozíciu znaku v reťazci). S reťazcami je možné vykonávať rôzne operácie, používať špeciálne funkcie a metódy. Keďže dáta sú v praxi veľmi často uložené v textovom formáte, programátori musia vedieť so znakovými reťazcami zručne pracovať.

PRACOVNÝ LIST Č. 1

Úloha 1: Reťazec je postupnosť znakov uzavretá v úvodzovkách alebo v apostrofoch. Pre niektoré špeciálne znaky používame zápis začínajúci spätnou lomkou. Aký majú tieto znaky význam?

```
print('Janko sa rozhodol ist na nakup.')

print('Mama povedala: "Janko, kup rozky!"')
print("I'm happy!")

print("Slnko svieti,\nvoda láka,\npostavme si snehuliaka.\n")

print('Slovensko:\tBratislava')
print('Rakúsko:\tViedeň')
print('Poľsko:\t\tVaršava')
print('Česko:\t\tPraha')
print('Ukrajina:\tKyjev')

print('\\')
```

Úloha 2: Ktoré operácie s reťazcami už poznáte? Napíšte program, v ktorom ich použijete.

Úloha 3: Reťazec je zložený zo znakov. K jednotlivým znakom sa dá pristupovať prostredníctvom indexov. Indexuje sa celými číslami. Prvý znak reťazca má index 0. Ktoré znaky sa vypíšu na obrazovku v nasledujúcom programe?

```
s = 'programovanie'
print(s[0])
print(s[1])
print(s[8])
print(s[10])
print(s[12])
print(s[13])
```

Úloha 4: A ktoré znaky sa vypíšu teraz?

```
s = 'programovanie'
i = 2
print(s[i])
print(s[2*i])
print(s[2*i+4])
print(s[i-1])
print(s[i+1])
```

Úloha 5: V predchádzajúcej úlohe sme videli, že index znaku, s ktorým chceme pracovať, je možné získať aj ako výsledok vyhodnotenia výrazu. Aký výsledok uvidíme na obrazovke po spustení nasledujúceho programu?

```
from random import randrange

s = 'programovanie'
print(s[randrange(13)])
print(s[randrange(13)])
print(s[randrange(13)])
```

Úloha 6: Pri práci s reťazcami je v jazyku Python možné používať aj záporné indexy. Nakreslite obrázok s vyznačením indexov znakov reťazca pomocou nezáporných aj záporných čísel. Čo sa v nasledujúcom programe vypíše na obrazovku?

```
s = 'programovanie'
print(s[-1])
print(s[-2])
print(s[12])
print(s[5]==s[-5])
print(s[-13]==s[0])
print(s[13])
```

Úloha 7: Reťazec často potrebujeme spracovať po znakoch. Otestujte uvedený program. Upravte ho tak, aby vyhláskoval meno zadané používateľom:

```
for znak in 'Nitra':
  print(znak)
```

Úloha 8: Doplňte uvedený program tak, aby vypísal, koľkokrát sa v ňom nachádza medzera:

```
def pocet_medzier(s):
  n = 0

  return n

text = input()
print(pocet_medzier(text))
```

Úloha 9: Koľko znakov majú uvedené reťazce?

```
s1 = 'Everest'
s2 = 'x'
s3 = ''
s4 = 'Tento text je dost dlhy na to, aby sa nam nechcelo pocitat jeho
znaky rucne. Pouzime funkciu len.'
```

Úloha 10: Ak vieme dĺžku reťazca, vieme aj index jeho posledného znaku. Vypíšte posledný znak reťazca bez použitia záporného indexu.

```
text = input()
print(len(text))
print(text[     ])
```

Úloha 11: Po znakoch reťazca sa vieme a niekedy aj musíme prejsť pomocou indexovania. Aký výstup očakávate v nasledujúcom programe? Overte svoju predpoveď.

```
text = input()
for i in range(len(text)):
  print(str(i) + '. znak je ' + text[i])
```

Úloha 12: Upravte program z predchádzajúcej úlohy tak, aby sa na obrazovku vypísali znaky pôvodného reťazca v obrátenom poradí.

PRACOVNÝ LIST Č. 2

Úloha 1: Dokončite funkciu, ktorá zistí, či zadaný reťazec obsahuje hľadaný znak:

```
def obsahuje_znak(retazec, znak):
  pass

text = 'Univerzita Konstantina Filozofa v Nitre'
hladany = 'f'
if obsahuje_znak(text, hladany):
  print('je tam')
else:
  print('nie je tam')
```

Úloha 2: Napíšte funkciu, ktorá zo vstupného reťazca odstráni všetky medzery.

```
def bez_medzier(retazec):
  pass

text = input()
print('retazec zbaveny medzier:', bez_medzier(text))
```

Úloha 3: Napíšte funkciu, ktorá vráti náhodný binárny reťazec (postupnosť núl a jednotiek) požadovanej dĺžky:

```
def nahodny_binarny(dlzka):
  pass

# vypíšeme ich na obrazovku viacej
for i in range(10):
  print(nahodny_binarny(8))
```

Úloha 4: Dokončite funkciu, ktorá z rodného čísla zistí, či je osoba muž alebo žena.

```
def zisti_pohlavie(rc):
  if :
    return 'zena'
  return 'muz'

vstup = input('Tvoje rodne cislo:')
print('Si ' + zisti_pohlavie(vstup) + '.')
```

Úloha 5: Každý znak má v znakovej tabuľke (napr. ASCII alebo Unicode) svoj číselný kód. Hovoríme mu poradové, t. j. *ordinálne číslo*. Aký kód majú iniciály vášho mena? Pri hľadaní odpovede využite pripravenú funkciu:

```
def vypis_kodov(retazec):
  for znak in retazec:
    print('znak', znak,'ma kod', ord(znak))
```

Úloha 6: Niekedy vieme číselný kód znaku a chceme vidieť jeho znakovú reprezentáciu. Otestujte uvedenú funkciu s inými vstupnými parametrami:

```
def vypis_znakov(start, stop):
  for kod in range(start, stop+1):
    print(kod,'>> ', chr(kod))

vypis_znakov(32, 127)
```

Úloha 7: Potrebujeme funkciu, ktorá vstupný reťazec zašifruje tak, že každý jeho znak nahradí znakom, ktorý je v ASCII tabuľke o požadovaný počet pozícií ďalej. Opravte nesprávne riešenie uvedené nižšie:

```
def sifruj(sprava, posun=1):
  vysledok = ''
  for znak in sprava:
    novy_znak = znak + posun
    vysledok = vysledok + novy_znak
  return vysledok

print(sifruj('ABC'))
print(sifruj('ABC',3))
print(sifruj('Python je super',5))
print(sifruj('mama',2))
```

Úloha 8: Keďže znaky sú v tabuľke usporiadané, vieme ich porovnávať. Vieme tiež, že malé písmená, veľké písmená a znaky číslic nasledujú v tabuľke za sebou. Doplňte do programu funkcie na zisťovanie počtu veľkých písmen a číslic. Všetky funkcie otestujte na vstupe získanom od používateľa:

```
def pocet_malych(slovo):
  pocet = 0
  for znak in slovo:
    if 'a'<= znak <= 'z':
      pocet += 1
  return pocet

vstup = input()
print('pocet malych:', pocet_malych(vstup))
```

Úloha 9: V prvej úlohe ste napísali funkciu na zisťovanie, či sa nejaký znak nachádza v reťazci. Dá sa to aj inak, pomocou operátora *in*. Aké hodnoty uvidíme na obrazovke?

```
sprava = input()
print('a' in sprava)
print('x' in sprava)
print('x' not in sprava)
```

Úloha 10: S využitím operátora *in* naprogramujte funkciu, ktorá zistí, koľko samohlások sa nachádza vo vstupnom reťazci. Dokončite rozpracované riešenie:

```
def pocet_samohlasok(text):
  pocet = 0

  return pocet

slovo = 'jahoda'
print(f'pocet samohlasok: {pocet_samohlasok(slovo)}')
```

Úloha 1: Reťazce sú *nemeniteľné hodnoty*. Znaky reťazca nie je možné zmeniť na iné. Ak potrebujeme reťazec upraviť (nejaké znaky z neho vynechať, či naopak niektoré znaky nahradiť inými) musíme vždy vyrobiť nový reťazec. Vyskúšajte:

```
s = 'programovanie'
s[3] = 'X'
```

Úloha 2: Z reťazca sa dajú ľahko získavať jeho časti (podreťazce). Hovoríme im *rezy*. Všimnite si zápis, ktorým z pôvodného reťazca získame rezy (nové reťazce). Vysvetlite význam jednotlivých zápisov. Nakreslite obrázok, v ktorom vyznačíte znaky vytvoreného rezu (rovnako aj v ďalších úlohách o rezoch).

```
s = 'programovanie'
print(s[1:5])
print(s[2:6])
print(s[3:4])
```

Úloha 3: Čo sa stane, ak niektorú z častí pred alebo za dvojbodkou vynecháme?

```
s = 'programovanie'
print(s[2:])
print(s[:7])
```

Úloha 4: A čo keď zápis pri indexovaní rezu rozšírime o ďalšiu dvojbodku a číslo?

```
s = 'programovanie'
print(s[2:10:2])
print(s[10:2:-1])
```

Úloha 5: Posledná záhada na preskúmanie. Aké rezy získame z pôvodného reťazca?

```
s = 'programovanie'
print(s[::3])
print(s[::-1])
```

Úloha 6: Napíšte funkciu, ktorá overí, či je reťazec zadaný na vstupe palindrom. Využite rez.

```
def je_palindrom(retazec):
  pass

s = 'jelenovipivonelej'
if je_palindrom(s):
  print(s, 'je palindrom')
else:
  print(s, 'nie je palindrom')
```

Úloha 7: Uvedená funkcia spracúva číslo zadané na vstupe vo forme reťazca. Vysvetlite, čo presne na jednotlivých riadkoch robí. Potom ju vhodne premenujte a otestujte v hlavnom programe na niekoľkých vstupoch:

```
def funkcia(zapis_cisla):
  if '.' not in zapis_cisla:
    return zapis_cisla
  poz = 0
  while zapis_cisla[poz] != '.':
    poz = poz + 1
```

```
  pred = zapis_cisla[:poz]
  za = zapis_cisla[poz + 1:]
  return za + '.' + pred
```

Úloha 8: Niekedy aj čísla spracúvame ako reťazce znakov číslic. Nasledujúca funkcia vráti, koľko párnych cifier obsahuje číslo zadané ako vstupný parameter. Prečo bolo potrebné v tele funkcie použiť aj funkciu *int()* ?

```
def pocet_parnych(cislo):
  n = 0
  for znak in cislo:
    c = int(znak)
    if c%2 == 0:
      n += 1
  return n

print(pocet_parnych('12345678'))
print(pocet_parnych('13545134694132100 39'))
```

Pri práci s reťazcami môžeme používať aj mnohé užitočné funkcie, ktoré sú už v Pythone pre reťazce naprogramované. Môžeme napr. ľahko overiť, či reťazec tvoria len malé, veľké písmená, číslice alebo biele znaky. Alebo môžeme reťazec skonvertovať na taký, v ktorom budú všetky malé písmená nahradené veľkými (resp. naopak). Často tiež potrebujeme zisťovať, či reťazec obsahuje nejaký znak alebo podreťazec alebo nahradiť v reťazci výskyty nejakého znaku či podreťazca niečím iným.

Úloha 9: Aký význam majú funkcie použité v nasledujúcom programe? Svoje odpovede overte na viacerých vstupoch. Uveďte príklady vstupov, pre ktoré posledné tri funkcie vrátia hodnotu *True* alebo *False*:

```
text = input('zadaj text: ')
print(text.lower())
print(text.upper())
print()
print(text.islower())
print(text.isupper())
print(text.isdigit())
print(text.isspace())
```

Úloha 10: Všimnime si zápisy v predošlom programe. Najprv sme napísali reťazec, potom bodku a potom volanie funkcie, ktorú chceme pre tento reťazec použiť. Takého funkcie voláme *metódy*. Pracujú s tým reťazcom, pre ktorý ich zavoláme. Vyskúšajme ešte niektoré ďalšie metódy. Na čo slúžia? Čo vracajú?

```
s = 'jablko je zdrave, hruska tiez, ale viac mi chuti jablko'

print(s.count('jablko'))
print(s.count('banan'))

print(s.find('hruska'))
print(s.find('slivka'))

s = s.replace('jablko', 'ovocie')
s = s.replace('hruska', 'zelenina')
s = s.replace('u','')
print(s)
```

ZHRNUTIE

- Reťazec je postupnosť znakov uzavretá v úvodzovkách alebo apostrofoch. Reťazec obsahujúci úvodzovky musí byť ohraničený apostrofmi a naopak.
- Zápisy \n, \t, \\ sú príklady tzv. únikových (v angl. *escape*) sekvencií. Spätná lomka uvedená pred znakom mení v priebehu spracovania reťazca jeho štandardný význam. Takto môžeme do reťazca vložiť znak konca riadku, tabulátor či samotný znak reprezentujúci spätnú lomku.
- Reťazce môžeme spájať pomocou operátora +. Napr. *'ano' + 'nie' == 'anonie'*.
- Reťazec môžeme vynásobiť kladným celým číslom. Ide o operáciu opakovaného spájania. Napr. *'ano' * 3 == 'anoanoano'*.
- Funkcia *len()* vracia počet znakov reťazca (jeho dĺžku). Prázdny reťazec '' má dĺžku 0.
- K znakom reťazca pristupujeme prostredníctvom celočíselných indexov. Prvý znak zľava má index *0*, posledný znak má index o *1* menej ako celková dĺžka reťazca. Index zapisujeme do hranatých zátvoriek. Ak *s* je reťazec, *s[i]* je znak reťazca na pozícii *i*.
- Na indexovanie znakov sa dajú využiť aj záporné čísla. Posledný znak reťazca má index *-1*, predposledný *-2* atď.
- K požadovanému znaku reťazca sa často dostaneme na základe vyhodnotenia *indexového výrazu* napísaného v hranatých zátvorkách, ktorý môže obsahovať premenné, konštanty, celočíselné operácie aj volania funkcií. Napr. zápisom *s[random.randrange(len(s))]* získame náhodne zvolený znak reťazca *s*.
- Reťazec často spracúvame po znakoch v cykle *for*.
- Funkcie môžu mať parameter typu *str*, môžu tiež vracať reťazec ako výsledok výpočtu.
- Reťazce sú *nemeniteľné* (v angl. *immutable*). Po vytvorení reťazca už nie je možné meniť jeho znaky priraďovacím príkazom. Ak chceme reťazec upraviť (napr. vynechať z neho nejaké znaky alebo nahradiť niektoré znaky inými), musíme vytvoriť nový reťazec.
- Znaky v znakových tabuľkách sú usporiadané, môžeme ich preto porovnávať. Každému znaku zodpovedá číselný kód (ide o jeho poradové číslo v tabuľke znakov). Funkcia *ord()* vracia číselný kód znaku. Funkcia *chr()* vracia znak zodpovedajúci zadanému číselnému kódu.
- Pomocou operátora *in* zisťujeme, či reťazec obsahuje konkrétny znak alebo podreťazec.
- Z reťazca je možné vybrať (vyrezať) jeho časť. Vznikne tak nový reťazec, ktorý nazývame *rez* (v angl. *slice*). Ak *s* je reťazec, tak *s[start:stop]* je nový reťazec obsahujúci znaky pôvodného reťazca z rozsahu *start..stop-1*.
- Zápisom *s[start:stop:krok]* získame do rezu znaky na pozícii *start, start+krok, start+2*krok, ... start + n*krok*, pričom *start + n*krok < stop*.
- Ak chceme vytvárať rez od začiatku alebo po koniec pôvodného reťazca, môžeme hranice *start* a *stop* v zápise indexovania vynechať (napr. *s[:3], s[1:]* alebo *s[::2]*). Ak potrebujeme rez vytvoriť zo znakov pôvodného reťazca vyberaných v poradí sprava doľava, používame zápornú hodnotu kroku. Napr. *s[::-1]* predstavuje rez obsahujúci všetky znaky pôvodného reťazca v obrátenom poradí.
- V štandardnej knižnici jazyka Python nájdeme veľa užitočných metód pre prácu s reťazcami, napr. *lower(), upper(), islower(), isupper(), isdigit(), isspace(), count(), find(), replace()*. Význam vstupných parametrov a návratovej hodnoty jednotlivých funkcií je uvedený v online dokumentácii jazyka Python. Metódou nazývame funkciu, ktorú voláme pre konkrétny objekt, v tomto prípade pre reťazec. Za reťazcom napíšeme bodku a potom názov metódy. Napr. volaním *s.count('*')* získame počet výskytov znaku *'*'* v reťazci *s*.

TÉMA 3: CHYBY V PROGRAME, VÝNIMKY

Dôležitou súčasťou práce programátora je aj objavovanie a odstraňovanie chýb v programoch. Niektoré chyby sú očividné a opravia sa veľmi ľahko, pretože nás na ne upozornia chybové hlásenia v čase, keď svoj program vytvárame a priebežne testujeme. Keď nedodržíme pravidlá pre písanie programov (syntax jazyka Python), interpreter nášmu zápisu nerozumie. Takéto chyby voláme **syntaktické** (pri písaní zdrojového kódu urobíme preklep, zabudneme napísať apostrof, dvojbodku či zátvorku, neodsadíme blok kódu zľava, ako meno premennej či funkcie zvolíme niektoré z vyhradených slov jazyka a pod.).

Na rozdiel od syntaktických chýb, **logické chyby** sa v programe odhaľujú ťažšie. Program na prvý pohľad vyzerá v poriadku, pracuje, no výsledky, ktoré dáva, nie sú vždy správne alebo aj úplne nezmyselné. Takáto situácia si vyžaduje opätovnú analýzu celého riešenia. Príklady logických chýb: nevhodne inicializovaná premenná, zle zapísaná podmienka, nevhodne zvolený počet opakovaní, chyba v použitom vzorci a pod.

Programátor musí byť predvídavý a uvažovať aj o takých chybách, ktoré by v priebehu vykonávania programu nemuseli, ale mohli vzniknúť. Chyby, ktoré vznikajú počas behu programu, nazývame **behové chyby** (v angl. *runtime errors*). Ide o také situácie, keď interpreter Pythonu síce príkazu rozumie, ale nie je schopný ho vykonať (používateľ zadá hodnotu v nesprávnom formáte, vstupná hodnota je z neprípustného rozsahu, pri vyhodnocovaní výrazu vznikne požiadavka deliť nulou, chceme načítať údaje zo súboru, ktorý na disku neexistuje a pod.). O behových chybách sa v programe dozvedáme prostredníctvom **výnimiek**. Python nám umožňuje na ne vhodne reagovať.

PRACOVNÝ LIST Č. 1

Úloha 1: V zápise programu je syntaktická chyba. Aká? Program spustite a pozorujte chybový výpis na konzole.

```
for i in range(10)
  print('ahoj')

C:\Users\...\Python38\python.exe "D:/.../test.py"
 File "D:/.../test.py", line 1
  for i in range(10)
                   ^
SyntaxError: invalid syntax
```

Úloha 2: Na ktorých miestach programu sme urobili uvedené syntaktické chyby?

```
for i range(10):
  print('ahoj')

fro i in range(10):
  print('ahoj')

for i in range(10):
  print('ahoj)

for i in range(10):
  print('ahoj'
```

```
for i in range(10):
print('ahoj')
```

A. Blok s telom cyklu sme neodsadili zľava.
B. V zápise príkazu *for* chýba kľúčové slovo *in*.
C. Zabudli sme ukončiť reťazec.
D. V zápise príkazu *for* je preklep.
E. Chýba zátvorka.

Úloha 3: Je v uvedenom programe nejaká syntaktická chyba?

```
pocet = 10
1kus = 5
print(f'celkova suma: {1kus * pocet}')
```

Úloha 4: V nasledujúcom programe je logická chyba. Pre aké vstupné hodnoty sa prejaví?

```
print('Pocitam aritmeticky priemer zadanych hodnot.')
x = float(input('zadaj x: '))
y = float(input('zadaj y: '))
priemer =  x + y / 2
print('aritmeticky priemer: ', priemer)
```

Úloha 5: Náš program má vypísať všetky násobky prirodzeného čísla *x*, ktoré sú menšie nanajvýš rovné hodnote *h*. Ktoré z uvedených riešení neobsahuje logickú chybu?

```
# Riešenie A
x = int(input())
h = int(input())
for i in range(x,h,x):
  print(i, end=' ')

# Riešenie B
x = int(input())
h = int(input())
for i in range(x,h+1,x):
  print(i, end=' ')

# Riešenie C
x = int(input())
h = int(input())
for i in range(x,x,h+1):
  print(i, end=' ')
```

PRACOVNÝ LIST Č. 2

Úloha 1: Spustime nasledujúci program a na vstupe zadajme nečíselnú hodnotu. Pozorujme výstup:

```
vek = int(input('Tvoj vek: '))
if vek <= 12:
  print('Vstup zamietnuty.')
else:
  print('Vstup povoleny.')
```

Na obrazovke uvidíme:

```
Tvoj vek: osem
Traceback (most recent call last):
 File "D:/.../test.py", line 1, in <module>
   vek = int(input('Tvoj vek: '))
ValueError: invalid literal for int() with base 10: 'osem'
```

Vykonávanie programu sa zastavilo. Z výpisu je zrejmé, že počas behu programu vznikla chyba. Nastal výnimočný stav, niečo, s čím si interpreter nevedel poradiť.

O behovej chybe nás interpreter informoval vytvorením tzv. **výnimky** (v angl. *exception*). V našom príklade vznikla výnimka typu *ValueError*. Na vstupe sme zadali reťazec znakov, ktorý funkcia *int()* nedokáže skonvertovať na celé číslo.

Úloha 2: V konzole Pythonu sme vyskúšali rôzne príkazy. Zámerne sme vyvolali vznik rôznych výnimiek. K jednotlivým situáciám doplňte správne vysvetlenia:

```
>>> 123 + 'x'
...
TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> 5 / 0
...
ZeroDivisionError: division by zero

>>> 'hallo'[10]
...
IndexError: string index out of range

>>> subor = open('data.txt', 'r')
...
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'

>>> import math
>>> math.sin()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: sin() takes exactly one argument (0 given)

>>> pocet = pocet + 1
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'pocet' is not defined
```

Úloha 3: Otestujme uvedený program. Na vstupe najprv zadáme celé číslo. Potom sa zámerne pomýlime a zadáme nečíselný vstup:

```
pocet_deti = int(input('pocet deti: '))
pocet_jablk = int(input('pocet jablk: '))
print(f'Kazde dieta dostane {pocet_jablk // pocet_deti} jablk.')
```

Prvé spustenie programu:

```
pocet osob: 3
pocet jablk: 15
Kazde dieta dostane 5 jablk.
```

Druhé spustenie programu:

```
pocet osob: 10
pocet jablk: 12*
Traceback (most recent call last):
 File "D:/.../test.py", line 2, in <module>
   pocet_jablk = int(input('pocet jablk: '))
ValueError: invalid literal for int() with base 10: '12*'
```

V druhom prípade sa na obrazovke objavilo chybové hlásenie. Vznikla výnimka typu *ValueError*. Vo vykonávaní programu nebolo možné ďalej pokračovať, pretože nečíselný vstup sa nedá

skonvertovať na hodnotu typu *int*. Neformálne povieme, že funkcia *int()* vyhodila výnimku typu *ValueError*.

Dalo by sa vznikaniu výnimiek nejako zabrániť? Vo všeobecnosti, áno. Museli by sme však vždy vopred testovať, či je hodnota, pre ktorú chceme príkaz vykonať, v prípustnom formáte, resp. z dovoleného rozsahu. V našej vzorovej úlohe o rozdeľovaní jabĺk by sme ešte pred volaním funkcie *int()* museli overiť, či bolo na vstupe zadané celé číslo, napr. takto:

```
vstup = input('pocet deti: ')
if vstup.strip().isdigit():
  pocet_deti = int(vstup)
  # atď.
else:
  print('na vstupe nebolo zadane cele cislo')
```

Takýto prístup je však veľmi nepraktický a neprehľadný. Behovým chybám preto nebudeme predchádzať, ale v prípade, že nastanú, **zachytíme ich a ošetríme**.

Úloha 4: Preštudujte si nasledujúci program. Odhadnite význam nových kľúčových slov:

```
try:
  pocet_deti = int(input('pocet deti: '))
  pocet_jablk = int(input('pocet jablk: '))
  odmena = pocet_jablk // pocet_deti
except ValueError:
  print('Chyba! Na vstupe musi byt cele cislo!')
else:
  print(f'Kazde dieta dostane {odmena} jablk.')
```

Namiesto toho, aby sme najprv kontrolovali, či reťazec zadaný používateľom skutočne obsahuje len znaky tvoriace zápis celého čísla, necháme program, aby sa o prevod reťazca na celé číslo pokúsil. Ak sa mu to nepodarí, vznikne výnimka, my ju zachytíme a spracujeme. V tomto prípade nám postačí oznam pre používateľa upozorňujúci na nesprávne zadané vstupné hodnoty. Do bloku *try* píšeme časť programu, ktorá by mohla byť zdrojom výnimky. Ak v niektorom riadku v bloku *try* vznikne výnimka, program okamžite pokračuje vykonávaním príkazov v zodpovedajúcom bloku *except*. Ak žiadna výnimka v bloku *try* nevznikne, vykonajú sa príkazy v bloku *else.* Blok *else* je nepovinný, ak ho nepotrebujeme, nemusíme ho uviesť.

Úloha 5: Niektorí programátori nešpecifikujú typ výnimky, ktorú chcú v bloku *except* ošetriť. Ak v niektorom riadku v bloku *try* nastane výnimka (ľubovoľná), zachytíme ju v bloku *except.* Overte toto tvrdenie opakovaným spustením programu:

```
try:
  pocet_deti = int(input('pocet deti: '))
  pocet_jablk = int(input('pocet jablk: '))
  odmena = pocet_jablk // pocet_deti
except:
  print('nastala nejaka vynimka')
else:
  print(f'Kazde dieta dostane {odmena} jablk.')
```

Úloha 6: Ak chceme zistiť, o akú konkrétnu výnimku sa jedná, môžeme k nej pristúpiť pomocou premennej a vypísať potrebné informácie na obrazovku. Všimnite si nový zápis:

```
try:
  pocet_deti = int(input('pocet deti: '))
```

```
  pocet_jablk = int(input('pocet jablk: '))
  odmena = pocet_jablk // pocet_deti
except Exception as chyba:
  print('typ chyby: ', type(chyba))
  print('popis chyby: ', chyba)
else:
  print(f'Kazde dieta dostane {odmena} jablk.')
```

Typ *Exception* reprezentuje akúkoľvek výnimku, ktorá by mohla nastať. My ju pomenujeme *chyba* a pomocou tejto premennej sa o nej dozvieme viac. Zistite, čo uvidíme, keď na vstupe zadáme nulový počet detí.

Úloha 7: Dokončite program tak, aby v prípade vzniku nejakej výnimky, vypísal informáciu o nej na obrazovku. Potom program upravte tak, aby ste rôzne výnimky ošetrili v osobitných blokoch *except*.

```
   :
  a = int(input())
  b = int(input())
  print(f'podiel: {a // b}, zvysok: {a % b}')
       Exception as      :
  print(type(chyba))
  print(chyba)
```

Úloha 8: Napíšte program, ktorý pre dve celé čísla *x* a *y* (základ a exponent) vypočíta príslušnú mocninu x^y. Možné chybové stavy ošetrite s využitím mechanizmu výnimiek: ak používateľ nezadá celé číslo, vypíšte reťazec *'ValueError'*.

Úloha 9: Napíšte program, v ktorom od používateľa načítate textovú správu (ľubovoľný reťazec) a index požadovaného znaku (celé číslo). S využitím mechanizmu výnimiek ošetrite možné behové chyby. V prípade, že pre index používateľ zadá nečíselný vstup, vypíšte reťazec *'ValueError'*. Ak používateľ nezadá index z prípustného rozsahu, vypíšte *'IndexError'*.

Úloha 10: Funkcia *max()* použitá v nasledujúcom programe vracia najväčší prvok zo zoznamu, ktorý jej pošleme ako vstupný parameter. Nedokáže si však poradiť s ľubovoľným vstupom. Inicializujte zoznam *data* tak, aby ste vyvolali vznik jednotlivých výnimiek. Uveďte tiež príklady rôznych vstupov, pre ktoré výnimky nevzniknú.

```
data = ?
try:
  x = max(data)
except TypeError:
  print('Chyba: hodnoty roznych typov nemozno porovnavat!')
except ValueError:
  print('Chyba: na vstupe je prazdny zoznam!')
else:
  print('najvacsi prvok:', x)
```

Úloha 1: Okrem blokov *try, except* a *else*, súvisí so spracovaním výnimiek aj blok *finally*. Do bloku *finally* umiestňujeme tie príkazy, ktoré sa majú vykonať vždy, bez ohľadu na to, či v bloku *try* vznikla alebo nevznikla výnimka. Preštudujte si program:

```
try:
  pocet_deti = int(input('pocet deti: '))
  pocet_jablk = int(input('pocet jablk: '))
  odmena = pocet_jablk // pocet_deti
except ValueError:
  print('Chyba! Na vstupe musi byt cele cislo!')
except ZeroDivisionError:
  print('Chyba! Pokus o delenie nulou!')
else:
  print(f'Kazde dieta dostane {odmena} jablk.')
finally:
  print('dovidenia')
```

Vyvolajte vznik niektorej z výnimiek a zadajte aj korektné vstupné hodnoty. Kedy sa na obrazovke objaví reťazec *'dovidenia'* ?

Úloha 2: K blokom používaným pri spracovaní výnimiek priraďte ich význam:

```
try:   except:  else:  finally:
```

A. Sem sa píše kód, ktorý sa vykoná vždy.
B. Sem sa píše kód ošetrujúci zachytenú výnimku.
C. Sem sa píše kód, ktorý by mohol spôsobiť problém.
D. Sem sa píše kód, ktorý sa vykoná, keď žiadna výnimka nenastane.

Úloha 3: Pri spracovaní čísel a reťazcov sme už v programe veľakrát použili rôzne funkcie. Ak niektorá z funkcií nebola schopná zrealizovať svoj výpočet, oznámila to na miesto volania (do volajúcej funkcie) prostredníctvom výnimky. Uvažujme napr. program, v ktorom používateľ zadáva hodnotu úroku, na základe ktorej sa zvýši suma na bankovom účte. Funkcia *float()* pre nečíselný vstup vyhodí výnimku typu *ValueError*. Zareagujeme na ňu tak, že premennú *urok* nastavíme na východiskovú hodnotou *1%.* Zabezpečíme tým, že program bude ďalej pokračovať zmysluplným spôsobom. Otestujte program pre rôzne vstupné hodnoty:

```
try:
  urok = float(input('urok = '))
except ValueError:
  urok = 0.01

# pokračovanie programu
print(urok)
```

Úloha 4: Niekedy sa tiež môže stať, že funkcia aj pre korektné vstupy nevie vrátiť zmysluplný výsledok. Preštudujte nasledujúci program:

```
text = input('Tvoj text: ')
hladany = 'kuk'
print(text.index(hladany))
```

Otestujete program s 3 rôznymi vstupmi, napr.:

```
Tvoj text: Kukulienka zakukaj.
13
Tvoj text: Nezakukam.
4
Tvoj text: Mozno inokedy.
Traceback (most recent call last):
 File "D:/../test.py", line 11, in <module>
   print(text.index(hladany))
ValueError: substring not found
Process finished with exit code 1
```

Na výstupe vidíme index prvého výskytu hľadaného podreťazca.

V poslednom prípade sa hľadaný podreťazec v zadanom texte nenachádza, reťazcová metóda *index()* túto skutočnosť oznámila vytvorením výnimky typu *ValueError*.

Prečo metóda *index()* vyhodila výnimku a nevrátila napr. hodnotou *-1*? Hodnota *-1* je index posledného znaku reťazca. Nedá sa preto využiť pre účely rozpoznania neúspechu operácie.

Úloha 5: Aj pri písaní vlastných funkcií by sme mali mať istotu, že vstupy, ktoré má funkcia spracovať, sú korektné a že funkcia vracia správny, zmysluplný výsledok. Preštudujte nasledujúcu funkciu na výpočet obsahu kruhu so zadaným polomerom a testovací program, v ktorom túto funkciu zavoláme:

```
import math

def obsah_kruhu(polomer):
  return math.pi * polomer ** 2

print(obsah_kruhu(1))      # 3.141592653589793
print(obsah_kruhu(-2))     # 12.566370614359172
```

V obidvoch prípadoch sa zrealizoval výpočet a výsledok sa vypísal na obrazovku. Ktorý z nich je nesprávny a prečo?

Úloha 6: Funkcia *obsah_kruhu()* mala volajúcu funkciu (v našom prípade hlavný program) prostredníctvom výnimky informovať, že nie je schopná vrátiť správny výsledok. Na vytvorenie výnimky požadovaného typu slúži príkaz *raise*. Opravte program podľa vzoru a otestujte ho pre rôzne hodnoty polomerov:

```
import math

def obsah_kruhu(polomer):
  if polomer < 0:
    raise ValueError('Polomer kruhu nemôže byť záporné číslo.')
  return math.pi * polomer ** 2

try:
  print(obsah_kruhu(1))
  print(obsah_kruhu(-2))
except ValueError as chyba:
  print(chyba)
```

Obvykle pri vytváraní výnimky uvádzame v zátvorkách aj reťazec opisujúci chybu (no nie je to nevyhnutné), napr.:

```
raise ValueError('na vstupe musí byť celé číslo')
raise ZeroDivisionError('menovateľ zlomku je nulový')
raise TypeError('zlý operand')
raise ValueError
```

V poslednom prípade sme pri vytvorení výnimky typu *ValueError* nepoužili žiadnu vlastnú správu.

Úloha 7: Príkaz *raise* môže stáť aj samostatne, bez špecifikovania typu výnimky:

```
raise
```

Tento zápis používame na propagovanie (šírenie, vyhodenie) výnimky ďalej, t. j. o úroveň vyššie do volajúcej funkcie. Volajúca funkcia môže výnimku vhodne ošetriť (ak obsahuje zodpovedajúci blok *except*) alebo výnimku posunie ďalej. Ak sa o existujúcu výnimku „nikto nepostará“, zachytí ju napokon interpreter Pythonu a program sa ukončí.

V nasledujúcom príklade sa od používateľa snažíme načítať celé číslo. Používateľ má dovolené urobiť 3 chyby. Ak nastane chyba typu *ValueError*, zachytíme ju a používateľa informujeme vypísaním správy. Po treťom neúspešnom pokuse zareagujeme „vyhodením“ výnimky na miesto volania. Príkaz *print()* sa už po tretí raz nevykoná. Otestujte program pre rôzne vstupné hodnoty:

```
def citaj_cislo():
  n = 0
  while True:
    try:
      return int(input('Tvoje číslo: '))
    except ValueError:
      n += 1
      if n >= 3:
        raise
      print('Zadaj celé číslo!')

# hlavný program
x = citaj_cislo()
```

Úloha 8: Naprogramujte funkciu na výpočet obsahu štvorca tak, aby v prípade zápornej dĺžky strany vyhodila výnimku typu *ValueError* s vlastným slovným opisom chyby. V hlavnom programe vypíšte informáciu o vzniknutej chybe a otestujte program pre rôzne vstupy:

```
def obsah_stvorca(a):
  return a * a

# hlavný program
try:
  strana = int(input('zadaj dlzku strany:'))
  print(obsah_stvorca(strana))
except ValueError as chyba:
  print(chyba)
```

ZHRNUTIE

- Pri písaní programu robíme rôzne chyby. Na *syntaktické chyby* nás upozorní interpreter chybovým hlásením (nebude nášmu zápisu rozumieť). *Logické chyby* musíme v programe lokalizovať a opraviť sami (program sa nezastaví, výsledky však budú pre niektoré vstupy nesprávne).

- Po spustení programu môžu vznikať aj tzv. *behové chyby* (v angl. *runtime errors*), ktoré by mal alebo dokonca musí programátor predvídať a vopred zabezpečiť, aby program na ne reagoval očakávaným a bezpečným spôsobom.
- Ak v priebehu vykonávania programu vznikne behová chyba (pokúšame sa o delenie nulou, chceme použiť funkciu pre nevyhovujúce vstupné dáta, index je mimo rozsahu prípustných hodnôt a pod.), Python o tomto výnimočnom stave informuje vytvorením výnimky (objektu s informáciou o chybe).
- Príklady výnimiek: *ValueError, TypeError, IndexError, ZeroDivisionError, FileNotFoundError*.
- Pri práci s výnimkami používame bloky *try, except, else* a *finally*.
- Do bloku *try* píšeme príkazy, ktoré môžu byť zdrojom výnimky.
- Ak výnimka nastane, vykonávanie príkazov v bloku *try* už nepokračuje. Výnimku zachytíme v bloku *except*. Blokov *except* môže byť viac, vďaka čomu sa dá na rôzne typy výnimiek reagovať rôznym spôsobom.
- Výnimku je možné pomenovať a prostredníctvom príslušnej premennej s ňou ďalej pracovať (napr. vypísať informáciu o chybe, ktorú objekt výnimky obsahuje).
- Typ výnimky nemusíme v bloku *except* špecifikovať. V takom prípade sa v ňom zachytí akákoľvek výnimka, ktorá vznikne v príslušnom bloku *try*.
- V bloku *else* uvádzame, čo sa má stať, ak v bloku *try* výnimka nenastane.
- Blok *finally* je podobne ako blok *else* nepovinný. Zíde sa nám neskôr pri komplexnejších projektoch, zvyčajne pre uvoľnenie rôznych prostriedkov, ktoré program používa (napr. uzatvorenie otvorených súborov, sieťových spojení a pod.).
- Program po vzniku výnimky nemusí nevyhnutne skončiť, po obnove korektného stavu (napr. získaní vstupnej hodnoty v správnom formáte alebo nastavení požadovanej premennej na vhodnú náhradnú hodnotu) môže pokračovať vo výpočte.
- Pri písaní vlastných funkcií je rozumné overovať, či sú hodnoty vstupných parametrov korektné a v prípade potreby „vyhodiť“ zodpovedajúcu výnimku na miesto volania.
- Výnimku vygenerujeme príkazom *raise*. Môžeme, ale nemusíme špecifikovať typ výnimky a podrobnejší opis chyby.
- Ak sa na mieste volania funkcie, ktorá „vyhodila“ výnimku, o túto výnimku nezaujímame (nereagujeme na ňu), propaguje sa o úroveň vyššie. Ak sa na výnimku nikde v zdrojovom kóde nereaguje, vykonávanie programu ukončí napokon interpreter Pythonu a na konzole uvidíme správu o chybe.

TÉMA 4: ZOZNAMY, N-TICE

Programy väčšinou spracúvajú veľa údajov, ktoré musíme vedieť efektívne organizovať (chceme v nich predsa vyhľadávať alebo realizovať rôzne iné výpočty, samozrejme čo najrýchlejšie). Najjednoduchšou a v praxi najčastejšie používanou údajovou štruktúrou je *zoznam* prvkov (údajový typ *list*). K prvkom zoznamu sa pristupuje prostredníctvom celočíselných indexov, ktoré predstavujú pozície prvkov v zozname. Zoznam je *meniteľná údajová štruktúra* – v priebehu vykonávania programu sa môže počet prvkov i prvky, ktoré obsahuje, meniť. Veľa operácií, ktoré poznáme zo spracovania reťazcov, môžeme analogicky aplikovať aj na zoznamy (reťazce sú postupnosti znakov). Existujú aj *nemeniteľné postupnosti prvkov*, ktoré voláme n-tice (údajový typ *tuple*). Tie je výhodné použiť v prípadoch, keď sa uložená postupnosť hodnôt nemení. Okrem toho, funkcie môžu vracať v *n*-tici viac hodnôt súčasne. Prvky zoznamu môžu byť jednoduchého (*int, float, bool*), ale aj zloženého typu (reťazce, zoznamy či n-tice). Zoznamami zoznamov, resp. zoznamami *n*-tíc sa v tejto publikácii nezaoberáme.

PRACOVNÝ LIST Č. 1

Úloha 1: Prvky zoznamu vymenúvame v hranatých zátvorkách, oddeľujeme ich čiarkami. K prvkom zoznamu budeme podobne ako pri znakoch v reťazci pristupovať prostredníctvom celočíselných indexov. Prvý prvok zoznamu má index 0, druhý prvok 1 atď. Uvedené zoznamy znázornite graficky. Vyznačte tiež *indexy* jednotlivých prvkov. Čo by mohli tieto údajové štruktúry reprezentovať?

```
a = [95, 80, 77, 55, 85, 90, 92, 88, 12, 49]
b = [6.5, 6.32, 5.78, 7.12, 7.88, 6.14]
c = [True, True, False, True, False, False, True, True]
d = ['Zuzana', 'Jozef', 'Mária', 'Peter']
e = []
f = list()
```

Úloha 2: Často je nutné zistiť, koľko prvkov zoznam obsahuje. Počet prvkov vstupného zoznamu vráti funkcia *len()*. Aké výsledky uvidíme na obrazovke pre zoznamy z predchádzajúcej úlohy?

```
print(len(a), len(b), len(c), len(d), len(e), len(f))
```

Úloha 3: Môže zoznam obsahovať aj prvky rôznych typov?

Úloha 4: Zoznamy môžeme podobne ako znakové reťazce spájať pomocou operátora *+*. Aký bude stav jednotlivých zoznamov po vykonaní nasledujúcich príkazov?

```
a = [1, 2, 3, 4, 5]
b = [0, 0, 0]
c = a + b
d = b + a
e = []
e = e + [100]    # alebo e += [100]
e = e + [200]    # alebo e += [200]
e = e + [300]    # alebo e += [300]
print(a, b, c, d, e)
```

Úloha 5: Zoznam *data* je na začiatku prázdny. Naplňte ho postupne *n* celočíselnými hodnotami zadávanými na klávesnici:

```
n = int(input('pocet prvkov:'))
data = []
# dokončite riešenie
print(data)
```

Úloha 6: Uvedený program otestujte pre rôzne vstupné reťazce. Vysvetlite, na čo slúži reťazcová metóda *split().* Akú hodnotu vracia?

```
vstup1 = input()            # napr. 'Danka Janka Petko Palko'
vstup2 = input()            # napr. '0,2000,2500,1800,1950,2100'
vstup3 = input()            # napr. '1+2+3'
data1 = vstup1.split()
data2 = vstup2.split(',')
data3 = vstup3.split('+')
print(data1, data2, data3)
```

Úloha 7: Zoznam reťazcov získaný ako výsledok volania metódy *split()* je často potrebné kvôli ďalšiemu spracovaniu transformovať na zoznam číselných hodnôt. Zadajte na vstupe v jednom riadku známky žiakov, oddeľte ich čiarkou. Otestujte program pre rôzne vstupy:

```
vstup = input().split(',')
print(vstup)                # napr. '1,2,3,4,5,2,3,1,1,3,1,2,1,2,1,1,4'
znamky = []
for x in vstup:
  znamky += [int(x)]
print(znamky)
```

Úloha 8: Zoznamy môžeme opakovane spájať tak, že zoznam *vynásobíme kladným celým číslom*. Takýto zápis je výhodný napr. vtedy, keď potrebujeme zoznam inicializovať väčším počtom rovnakých prvkov. Dá sa rovnaký zoznam vytvoriť inak?

```
>>> data = [0] * 10
>>> data
[0,0,0,0,0,0,0,0,0,0]
```

Úloha 9: Potrebujeme zoznam 20 náhodne zvolených jednociferných čísel. Aký zoznam vznikne v nasledujúcom programe?

```
import random
cisla = [random.randrange(10)] * 20
print(cisla)
```

Úloha 10: *Generátorová notácia* zoznamu (v angl. aj *list comprehension*) predstavuje veľmi praktický a kompaktný spôsob *vytvárania nových zoznamov*. Celý návod na vytvorenie zoznamu uvedieme v hranatých zátvorkách. Napíšeme *výraz*, vyhodnotením ktorého *vznikne nový prvok,* a ten sa pridá na koniec generovaného zoznamu. Určíme tiež, pre aké *vstupné hodnoty*, resp. koľkokrát sa má tento *výraz* vyhodnocovať (v klauzule *for*). Môžeme tiež povedať, že zo vstupných hodnôt treba pri generovaní nových prvkov použiť len tie, ktoré vyhovujú konkrétnej podmienke (v klauzule *if*). Spustite uvedený program. Aké zoznamy sme vytvorili? Koľko prvkov obsahujú? Ako sa tieto prvky vypočítali?

```
from random import randint
from math import pi

nuly = [0 for i in range(100)]

a = [i for i in range(10)]
```

```
b = [2 * i + 1 for i in range(10)]
c = [i ** 2 for i in range(5)]
d = [randint(100, 999) for i in range(10)]
roky = [1812, 1867, 1945, 1969, 1971, 1980, 1984, 2012, 2015]
e = [rok % 100 for rok in roky if rok > 1900]

slova= ['toto', 'je', 'zoznam', 'slov']
f = [slovo[0].upper() for slovo in slova]

veci = ['ceruzka', 'pero', 'guma', 'papier']
g = [vec for vec in veci if len(vec) > 5]

h = [str(round(pi, i)) for i in range(1, 6)]

nasobky = [x * y for x in range(1, 4) for y in range(1, 4)]
cisla = [int(x) for x in input().split()]
```

Úloha 11: Už sme videli, že pomocou konštruktora zoznamu – funkcie *list()*, môžeme vytvoriť prázdny zoznam. Ak zavoláme túto funkciu so vstupným parametrom, ktorý predstavuje nejakú postupnosť prvkov, vytvoríme nový zoznam obsahujúci príslušné prvky. Preštudujte si uvedený program. Potom pomocou funkcie *list()* vytvorte zoznam obsahujúci znaky vášho mena v obrátenom poradí, všetky písmená nech sú veľké:

```
>>> zoznam = list('ahoj')
>>> zoznam
['a', 'h', 'o', 'j']
>>> zoznam = list(range(10))
>>> zoznam
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> zoznam2 = list(zoznam)
>>> zoznam2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Úloha 12: Znázornite priebeh vykonávania uvedeného programu graficky. Čo sa vypíše na obrazovku?

```
a = [1, 2, 3]
b = [4, 5, 6, 7, 8]
a = b
print(a)
print(b)
```

PRACOVNÝ LIST Č. 2

Úloha 1: Prvky zoznamu majú indexy zodpovedajúce ich pozícii v zozname. Podobne ako v prípade reťazcov, aj zo zoznamov môžeme vytvárať rezy (nové zoznamy tvorené prvkami vybranými z pôvodného zoznamu). Aké prvky bude obsahovať zoznam *data* po vykonaní uvedených príkazov?

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
x = a[0] + a[1]
y = a[len(a)//2]
z = a[-1]
data = [x, y, z] * 3
data = data[1::2]
print(data)
```

Úloha 2: Vymyslite úlohu analogickú k predchádzajúcej pre svoju spolužiačku/spolužiaka. Správnu odpoveď overte spustením programu.

Úloha 3: Zoznam je *meniteľná štruktúra* (v angl. *mutable structure*). Môžeme doň prvky pridávať, odstraňovať (teda meniť jeho dĺžku), ale i meniť hodnoty prvkov. Prvok zo zoznamu odstránime príkazom *del*. Otestujte uvedený program. Po každej zmene zoznamu vypíšte jeho aktuálny stav na obrazovku:

```
ziaci = ['Janko', 'Jurko', 'Katka', 'Hanka', 'Petko']
del ziaci[1]
del ziaci[-1]
ziaci[0] = 'Vierka'
ziaci[1] = 'Julia'
```

Úloha 4: Zoznam potrebujeme obvykle spracúvať prvok po prvku. Porovnajte výstupy uvedených programov:

```
cisla = [1, 2, 3, 4, 5]

# Program 1
for x in cisla:
  print(x, end=' ')
print()

# Program 2
for i in range(len(cisla)):
  print(cisla[i], end=' ')
print()

# Program 3
for x in cisla[::-1]:
  print(x, end=' ')
print()

# Program 4
for i in range(len(cisla) - 1, -1, -1):
  print(cisla[i], end=' ')
print()

# Program 5
for i, x in enumerate(cisla):
  print(f'a[{i}] = {x}')
```

Úloha 5: Vytvorte program, v ktorom zadáte vstupný zoznam s celými číslami a hranice intervalu. Potom zistite, koľko prvkov zoznamu je zo zadaného intervalu.

Úloha 6: Vytvorte program, v ktorom od používateľa načítate známky študentov zo skúšky. Potom vypočítajte aritmetický priemer známok a vypíšte známky tých študentov, ktorí dosiahli nadpriemerné výsledky (majú známku lepšiu ako priemer).

Úloha 7: V priebehu týždňa sa merali teploty vzduchu. Sú uložené v zozname. V ktorý deň sledovaného obdobia bolo najteplejšie? Aká bola vtedy teplota? Vytvorte program, ktorý spracuje vstupný zoznam teplôt a vypíše požadované informácie.

Úloha 8: Vytvorte program, ktorý bude generovať *n* náhodných oznamovacích viet z reťazcov pripravených v zoznamoch. Do prvého zoznamu uložte niekoľko mien, do druhého niekoľko činností,

do tretieho niekoľko miest. Počty reťazcov v jednotlivých zoznamoch nemusia byť rovnaké. Čím viac prvkov budú mať, tým bude výstup zaujímavejší.

Príklad vstupu:

```
kto = ['Matus', 'Filip', 'Dano', 'Alex', 'Maxim', 'Emil', 'Tibor']
corobi = ['tancuje', 'spieva', 'recituje', 'sa uci', 'spi', 'strikuje']
kde = ['v komore', 'na stanici', 'vo vani', 'v posteli', 'na ihrisku']
n = int(input('pocet viet:'))
```

Príklad výstupu:

```
Matus spieva vo vani.
Emil strikuje v komore.
Tibor recituje na stanici.
```

Úloha 9: Naprogramujte simuláciu *n* hodov hracou kockou. Vypíšte, koľkokrát padli jednotlivé hodnoty 1..6 (absolútne aj relatívne početnosti). Počty padnutí jednotlivých hodnôt bude rozumné evidovať v šesťprvkovom zozname, pomenujme ho napr. *p*:

```
import random
n = int(input('pocet hodov: '))
p = [0, 0, 0, 0, 0, 0]             # alebo p = [0] * 6
```

Úloha 10: Konali sa voľby. Do zoznamu uložte čísla reprezentujúce hlasy odovzdané jednotlivým kandidátom (uvažujme, že kandidátov je napr. *10* a majú čísla *0, 1, ..., 9*). Ak je na vstupe zoznam hodnôt napr. *1 1 2 1 3 1 4*, znamená to, že hlasovalo spolu 7 voličov. Štyria volili kandidáta s číslom 1, ďalší volili kandidátov *2, 3* a *4*. Nás zaujíma, kto voľby vyhral a aký bol rozdiel medzi víťazom a posledným v poradí. Vytvorte program, ktorý spracuje výsledky volieb.

Úloha 11: Na ulici stoja jeden vedľa druhého domy s rôznymi počtami poschodí. Vytvorte program, ktorý zistí, koľko existuje na ulici takých domov, že majú vľavo aj vpravo od seba dom s menším počtom poschodí (budeme ich volať domy s výhľadom). Informáciu o domoch na ulici načítajte od používateľa a uložte do zoznamu.

Úloha 12: V programe vygenerujte náhodne *n*-prvkový zoznam obsahujúci nuly a jednotky. Potom od používateľa načítajte hodnotu *k* a upravte pôvodný zoznam (t. j. nevytvárajte nový zoznam) tak, že každý jeho prvok posuniete o *k* miest vpravo. O zozname uvažujeme ako o logickom kruhu. Prvky z pravého konca sa posúvajú na začiatok zoznamu. Vypíšte obsah zoznamu pred aj po úprave.

PRACOVNÝ LIST Č. 3

Úloha 1: Zoznam je objekt a pri práci s ním môžeme používať rôzne metódy. Experimentujte s uvedeným programom a vysvetlite význam jednotlivých zoznamových metód. Všetky metódy, ktoré v tomto programe pre zoznam voláme, *menia jeho obsah*:

```
a = []
for i in range(5):
  x = input(f'a[{i}] = ')
  a.append(int(x))

a.append(100)
a.append(200)
a.insert(1, 0)
print(a)
```

```
x = a.pop()
print(x)
print(a)

x = a.pop(0)
print(x)
print(a)

a.remove(100)
print(a)
a.clear()
print(a)
```

Úloha 2: K dispozícii máme aj 2 také metódy, ktoré *nemenia obsah zoznamu*. Volaním *a.count(x)* získame počet výskytov prvku *x* v zozname *a*. Volanie *a.index(x)* vracia pozíciu prvého výskytu prvku *x* v zozname *a.* V prípade, že taký prvok v zozname nie je, vznikne výnimka *ValueError*. Ak chceme vyhľadávať iba v časti zoznamu začínajúcej na indexe *start* a končiacej na indexe *koniec-1* zavoláme rovnakú metódu s troma parametrami: *a.index(x, start, koniec).* Vytvorte krátky program a demonštrujte v ňom použitie týchto metód.

Úloha 3: Už vieme, že pri indexovaní rezov zo zoznamu (podzoznamov) vždy vznikne nový zoznam, teda pôvodný zoznam sa nezmení. Zápis používaný pri rezaní sa môže objaviť aj na ľavej strane príkazu priradenia. V takom prípade musí byť na pravej strane príkazu priradenia výraz, ktorého výsledkom je nejaká sekvencia prvkov (nemusí ísť o zoznam). Takéto *priradenie do rezu* sa vykoná nasledovne:

- výraz na pravej strane sa vyhodnotí a vytvorí sa nový zoznam,
- nový zoznam nahradí podzoznam definovaný rezom.

Priradením do rezu sa teda mení obsah zoznamu. Vyskúšajte si priraďovanie do rezu vo vlastnom programe podobnom vzorovému:

```
>>> mena = ['Matej', 'Marek', 'Lukas', 'Jan', 'Frantisek']
>>> mena[1:4] = ['Pavol', 'Peter', 'Tomas']
>>> mena
['Matej', 'Pavol', 'Peter', 'Tomas', 'Frantisek']
```

Tri prvky s indexom *1, 2, 3* sa nahradili novými prvkami.

Nahradiť môžeme aj nerovnaký počet prvkov. Napr.:

```
>>> mena = ['Matej', 'Marek', 'Lukas', 'Jan', 'Frantisek']
>>> mena[1:4] = ['Pavol', 'Peter']
>>> mena
['Matej', 'Pavol', 'Peter', 'Frantisek']
```

Tri prvky sa nahradili dvoma.

```
>>> mena = ['Matej', 'Pavol', 'Peter', 'Frantisek']
>>> mena[-2:] = ['Andrej', 'Jakub', 'Filip']
>>> mena
['Matej', 'Pavol', 'Andrej', 'Jakub', 'Filip']
```

Posledné dva prvky sa nahradili troma.

```
>>> mena = ['Matej', 'Pavol', 'Andrej', 'Jakub', 'Filip']
>>> mena[:2] = []
>>> mena
```

```
['Andrej', 'Jakub', 'Filip']
```

Prvé dva prvky pôvodného zoznamu boli odstránené a nahradené prázdnym zoznamom.

```
>>> mena = ['Andrej', 'Jakub', 'Filip']
>>> mena[:] = []
>>> mena
[]
```

Obsah pôvodného zoznamu sme nahradili prázdnym zoznamom (všetky prvky sa odstránili).

Úloha 4: Zoznam môže byť vstupným parametrom funkcie. Napíšte funkciu, ktorá vráti počet záporných čísel vo vstupnom zozname.

```
def pocet_zapornych(data):
  pass
```

Úloha 5: V zozname sú uložené výšky rastlín v centimetroch. Okrem tohto zoznamu používateľ zadá kladné alebo záporné desatinné číslo, ktoré znamená, koľkokrát sa majú rastliny zväčšiť alebo zmenšiť. Napr. koeficient *0.5* znamená, že každá z rastlín sa zmenší na polovicu. Koeficient *2.0* znamená, že každá rastlina sa zväčší dvojnásobne. Napíšte program, ktorý vypíše výšky rastlín po zväčšení alebo zmenšení. V riešení použite vlastnú funkciu s názvom *uprav_vysky()* s dvoma parametrami – zoznamom výšok a koeficientom zväčšenia, resp. zmenšenia.

Úloha 6: Programovací jazyk Python ponúka viacero štandardných funkcií, ktoré zjednodušujú prácu so zoznamom, prvú z nich už poznáme:

```
a = [10, 20, 30, 40, 50]
print(len(a))    # vráti počet prvkov zoznamu (jeho dĺžku)
print(sum(a))    # vráti súčet prvkov uložených v zozname
print(max(a))    # vráti prvok s maximálnou hodnotou
print(min(a))    # vráti prvok s minimálnou hodnotou
```

Uvedené funkcie však nie je možné aplikovať vždy. Nemá zmysel sčítavať prvky nečíselného typu alebo hľadať maximum v zozname s prvkami rôznych typov. Zavolanie príslušných funkcií by skončilo v takýchto prípadoch chybou.

Sformulujte problém, ktorý by ste mohli vyriešiť s využitím vyššie uvedených funkcií. Vytvorte príslušný program.

Úloha 7: Prítomnosť prvku v zozname môžeme podobne ako pri reťazcoch overiť pomocou operátora *in*. Napíšte funkcie s troma parametrami, ktoré vrátia *True* alebo *False*, podľa toho, či sa prvok *x* nachádza v oboch zoznamoch *a, b*, resp. nenachádza v ani jednom z nich:

```
def je_v_oboch(a, b, x):
  pass

def nie_je_v_ziadnom(a, b, x):
  pass
```

Úloha 8: Napíšte funkciu, ktorá upraví vstupný zoznam reťazcov tak, že každý výskyt zadaného reťazca nahradí prázdnym reťazcom.

Úloha 9: Napíšte funkciu, ktorá vráti *n*-prvkový zoznam s prvkami inicializovanými na východiskovú hodnotu určenú parametrom funkcie.

Úloha 10: Napíšte funkciu, ktorá vráti *n*-prvkový zoznam celých čísel naplnený náhodnými číslami z intervalu definovaného parametrami funkcie.

Úloha 11: Vytvorte program, v ktorom používateľ zadá dva vzostupne usporiadané zoznamy celých čísel. Následne program vytvorí a vypíše na konzolu nový, vzostupne usporiadaný zoznam obsahujúci všetky hodnoty z oboch vstupných zoznamov. Zlúčenie zoznamov zrealizujte vo funkcii.

Príklad vstupu:

```
prvy = [1, 5, 7, 9]
druhy = [2, 3, 8, 10, 11, 12]
```

Príklad výstupu:

```
[1, 2, 3, 5, 7, 8, 9, 10, 11, 12]
```

Úloha 12: Uveďte príklady funkcií s parametrom typu zoznam, ktoré menia stav skutočného parametra a ktorého ho nemenia.

Úloha 13: Údaje, ktoré máme uložené v zozname, je často potrebné usporiadať. Usporadúvať možno čísla aj reťazce, v prípade reťazcov sa porovnáva lexikograficky (t. j. „podľa abecedy“). V Pythone máme na výber dve možnosti: funkciu *sorted()* s parametrom typu zoznam a zoznamovú metódu *sort()* bez parametrov, ktorú voláme pre konkrétny zoznam. Vytvorte program, v ktorom zavoláte obe tieto funkcie.

Úloha 14: Pre metódu *sort()* a pre funkciu *sorted()* môžeme použiť voliteľný parameter *reverse* a dosiahnuť tak zostupné usporiadanie prvkov:

```
>>> data = [5, 3, 8, 6, 1, 2]
>>> data.sort(reverse=True)
>>> data
[8, 6, 5, 3, 2, 1]
```

Usporiadajte zoznam mien žiakov podľa abecedy. Najprv vzostupne, potom zostupne.

Úloha 15: Už sme videli, že zadávanie viacerých hodnôt oddelených oddeľovačom, napr. čiarkou, si môžeme zjednodušiť napr. s využitím generátorovej notácie takto:

```
cisla = [int(x) for x in input().split(',')]
print(cisla)
```

Príklad vstupu a výstupu

```
10,20,30,40,50
[10, 20, 30, 40, 50]
```

Rovnaký výsledok môžeme dosiahnuť aj pomocou funkcie *map()*. Na každý prvok zoznamu reťazcov, ktorý vráti metóda *split()*, sa aplikuje funkcia *int()*. Vzniknutú postupnosť celočíselných hodnôt je ešte potrebné skonvertovať na zoznam.

```
cisla = list( map(int, input().split(',')) )
print(cisla)
```

Mnohí programátori preferujú pri načítavaní štruktúrovaného vstupu používanie funkcie *map()*. Mapovacia funkcia je príkladom toho, že v Pythone je možné funkcie posielať ako parametre do iných funkcií. Vyskúšajte si jej použitie pri načítavaní postupnosti viacerých desatinných čísel oddelených medzerou.

Úloha 16: Rovnako ako reťazce, aj zoznamy je možné medzi sebou porovnávať pomocou štandardných relačných operátorov. Preskúmajme niekoľko príkladov:

```
>>> a = list(range(1,5))    #  [1, 2, 3, 4]
```

```
>>> b = list(range(1,5))   #  [1, 2, 3, 4]
>>> a == b
True
>>> a != b
False
>>> a[1] = 0
>>> a == b     # [1, 0, 3, 4] == [1, 2, 3, 4]
False
>>> a < b      # [1, 0, 3, 4] < [1, 2, 3, 4]
True
```

Skúste zrealizovať aj iné experimenty, porovnávajte napr. zoznamy rôznych dĺžok. Sformulujte na záver pravidlá, ktorými sa riadi vyhodnocovanie relačných operátorov pre zoznamy.

PRACOVNÝ LIST Č. 4

Úloha 1: Niekedy v programoch pracujeme s takými postupnosťami prvkov, ktoré po vytvorení už nemeníme. Nepridávame ani neodstraňujeme prvky a ani nemeníme ich hodnoty. V takýchto prípadoch je vhodnejšie (z pohľadu spracovania v pamäti efektívnejšie) namiesto zoznamu (údajového typu *list*) používať n-ticu (údajový typ *tuple*):

Vytvorme napr. n-ticu so 7 prvkami – reťazcami predstavujúcimi názvy dní týždňa:

```
>>> dni_tyzdna = ('pondelok', 'utorok', 'streda', 'stvrtok', 'piatok',
'sobota', 'nedela')
>>> dni_tyzdna
('pondelok', 'utorok', 'streda', 'stvrtok', 'piatok', 'sobota', 'nedela')
```

V zápisoch vyššie vidíme, že na rozdiel od zoznamu, n-ticu zapisujeme do okrúhlych zátvoriek. Prvky n-tice oddeľujeme čiarkami.

Vytvorme teraz n-ticu s dvoma prvkami reprezentujúcu bod v rovine. Prvkami tejto dvojice sú súradnice *x* a *y*:

```
>>> bod = (1, 4)
>>> bod
(1, 4)
```

Prvky tvoriace n-tice môžu byť rovnakého typu (n-tice čísel, n-tice reťazcov a pod.). Ale podobne ako zoznamy, môžu byť aj rôzneho typu:

```
>>> student = ('Jan', 'Smutny', 22)
>>> student
('Jan', 'Smutny', 22)
```

K prvkom n-tice pristupujeme podobne ako pri zoznamoch prostredníctvom indexov. Vypíšte na obrazovku prvé a posledné prvky vyššie uvedených n-tíc.

Úloha 2: S n-ticami môžeme vykonávať aj všetky ostatné operácie, ktoré sme si vyskúšali pri zoznamoch (spájanie, rezanie, porovnávanie, používanie rôznych funkcií – napr. *len()*, *max()*, *min()*, *sum()*). Ide však o nemennú údajovú štruktúru (v angl. *immutable structure*), jej prvky už po vytvorení nie je možné zmeniť, odstraňovať ich, ani do nej pridávať či vkladať ďalšie prvky. Vyskúšajte uvedené príkazy. Čo sa stane?

```
>>> dni_tyzdna = ('pondelok', 'utorok', 'streda', 'stvrtok', 'piatok',
'sobota', 'nedela')
```

```
>>> len(dni_tyzdna)
7
>>> vikend = dni_tyzdna[5:]
>>> vikend
('sobota', 'nedela')
>>> mesto = ('Nitra', 48.30763, 18.08453)
>>> mesto[0] = 'Bratislava'
```

Úloha 3: Máme k dispozícii premennú *data*, o ktorej vieme, že ide o referenciu na postupnosť prvkov. Ako zistíme, či je to zoznam alebo n-tica?

Úloha 4: Doplňte výsledky vykonania nasledovných príkazov:

```
>>> tuple('Python')

>>> tuple(range(1,11))

>>> tuple(['a', 'b', 'c'])

>>> list((0,-1, 1))

>>> tuple()

>>> list()
```

Úloha 5: Osobitnú pozornosť je potrebné venovať n-tici pozostávajúcej iba z jediného prvku. Vo svojom zápise musí obsahovať čiarku:

```
>>> melodia = ('C',)
>>> melodia
('C',)
>>> type(melodia)
<class 'tuple'>
```

V prípade, že by zápis čiarku neobsahoval, vytvorila by sa hodnota úplne iného typu:

```
>>> melodia = ('C')
>>> melodia
'C'
>>> type(melodia)
<class 'str'>
```

Vytvorte n-ticu ***melodia*** s jedným, dvoma a viacerými prvkami.

Úloha 6: Pri práci s n-ticami je potrebná opatrnosť, zmiasť môže aj obyčajný preklep. V uvedenom príklade je prvý zápis s desatinnou bodkou, to znamená, že sa vytvorí desatinné číslo. V druhom zápise je použitá čiarka, ktorá však slúži na oddelenie dvoch prvkov n-tice:

```
>>> (3.14)
3.14
>>> (3,14)
(3, 14)
```

Úloha 7: N-tice sú iterovateľné, teda je ich možné prehľadávať v cykle *for:*

```
prvocisla = (2, 3, 5, 7, 11, 13)
for i in prvocisla:
  print(i, end=' ')      # 2 3 5 7 11 13
```

Pre použitie n-tice vo *for* cykle nie je potrebné vymenované prvky uzavrieť do zátvoriek:

```
for i in 2, 3, 5, 7, 11, 13:
  print(i, end=' ')
```

N-tice sa používajú aj pri viacnásobnom priradení, napr.:

```
x, y, z = 10, 20, 30
print(x, y, z)            # 10, 20, 30
```

Hodnoty na pravej strane príkazu priradenia tvoria n-ticu s 3 prvkami.

Vysvetlite vykonanie uvedeného príkazu priradenia:

```
>>> zaznam = ('Nitra', 48.30763, 18.08453)
>>> mesto, sirka, dlzka = zaznam
```

Úloha 8: V jazyku Python môže byť výsledkom funkcie viac ako len jedna hodnota. V takomto prípade sa všetky vrátené hodnoty zabalia do jednej n-tice. Návratové hodnoty musia byť oddelené čiarkami, okrúhle zátvorky nie sú povinné.

Príklad:

```
def fun(a, b):
  x = random.randrange(a, b)
  y = random.randrange(a, b)
  return x, y
```

Uvedená funkcia vrátila 2 náhodné čísla. N-ticu je teraz možné použiť ako n-ticu alebo ju možno priradiť („rozbaliť“) do zodpovedajúceho počtu premenných:

```
# hlavný program
print(fun(1, 10))       # napr. (7, 4)
m, n = fun(1, 10)       # napr. m, n = (2, 5)
print(m, n)             # 2 5
```

Napíšte a otestujte funkciu, ktorá vráti podiel a zvyšok po celočíselnom delení dvoch celých čísel. V hlavnom programe osobitne vypíšte podiel a osobitne zvyšok po delení.

Úloha 9: Naprogramovali sme funkciu, ktorá nájde všetky delitele čísla *x* a vráti ich ako jednu n-ticu (hodnotu typu *tuple*). Využite túto funkciu v programe, ktorý overí, či je zadané číslo prvočíslo. Môže táto funkcia vrátiť aj prázdnu n-ticu?

```
def delitele(x):
  d = ()
  for i in range(1, x + 1):
    if x % i == 0:
      d = d + (i,)        # Všimnime si!
  return d
```

V označenom riadku vidíme, že k pôvodne prázdnej n-tici postupne pripájame *jednoprvkové n-tice* s práve objaveným deliteľom. Vznikne nová, dlhšia n-tica, ktorú si zapamätáme pomocou tej istej premennej.

Úloha 10: Napíšte program, v ktorom pre číslo zadané používateľom zistíte, koľko párnych a koľko nepárnych cifier obsahuje. Riešenie realizujte ako funkciu s jedným parametrom (zadaným číslom). Výsledok musí funkcia vrátiť vo forme n-tice. Prvým prvkom nech je počet párnych cifier, druhým prvkom počet nepárnych cifier.

Úloha 11: Opravte funkciu tak, aby vrátila cifry vstupného parametra ako n-ticu. Cifry nech sú v n-tici uvedené v pôvodnom poradí.

```
def rozober_na_cifry(cislo):
  vysledok = ()
  cislo = str(cislo)
  for znak in cislo:
    vysledok += (int(znak))
  return vysledok
```

Hlavný program doplňte tak, aby vypísal ciferný súčet, najväčšiu a najmenšiu cifru.

```
# hlavný program
x = int(input())
cifry = rozober_na_cifry(x)
print('cifry:', cifry)
```

Úloha 12: Robot sa pohybuje po štvorcovej sieti podľa plánu, ktorý je zakódovaný vo vstupnom reťazci. Vždy začína na pozícii so súradnicami 0, 0. Vstupný reťazec obsahuje postupnosť inštrukcií pohnúť sa v niektorom zo 4 smerov (S = sever, J = juh, V = východ, Z = západ). Dĺžka každého kroku je 1. Vytvorte program na vypísanie súradníc bodu, do ktorého sa robot dostane. Naprogramujte funkciu s jedným parametrom (reťazcom s návodom), ktorá vráti n-ticu (súradnice bodu, do ktorého sa robot presunul).

ZHRNUTIE

- Postupnosti hodnôt uchovávame najčastejšie v údajovej štruktúre zoznam (*list*). Prvky zoznamu sú obvykle rovnakého typu, môžu byť aj rôzneho typu. Vymenúvajú sa v hranatých zátvorkách oddelené čiarkami.
- K prvkom zoznamu máme priamy prístup prostredníctvom celočíselných indexov, podobne ako ku znakom reťazca.
- Funkcia *list()* slúži na vytvorenie nového, prázdneho zoznamu. Ak bude mať v parametri objekt predstavujúci nejakú postupnosť prvkov, budú tieto prvky pridané do vytvoreného zoznamu. Pomocou tejto funkcie môžeme vytvárať aj kópie existujúcich zoznamov.
- Ak *a, b* sú zoznamy, priradením *a = b* dosiahneme, že obe premenné budú odkazovať na ten istý zoznam v pamäti.
- Zoznamy môžeme spájať pomocou operátora + a opakovane spájať pomocou operátora *. Pomocou relačných operátorov ==, !=, <, <=, >=, > vieme zoznamy porovnávať.
- Generátorová notácia umožňuje veľmi stručným zápisom vytvárať nové zoznamy na základe špecifikovaného návodu na generovanie prvkov.
- Pomocou operátora *in* overujeme prítomnosť prvku v zozname.
- Metódou *split()* získame z postupnosti hodnôt zadanú na vstupe vo forme jedného reťazca zoznam reťazcov. Ak ide o postupnosť číselných hodnôt, zo zoznamu reťazcov obvykle následne vytvárame zoznam hodnôt typu *int* alebo *float*.
- Podobne ako na reťazce, aj na zoznamy môžeme aplikovať operáciu rezu. Každý rez je nový zoznam, ktorý vnikne z prvkov pôvodného zoznamu.
- Zoznam je meniteľná (v angl. *mutable*) údajová štruktúra. V priebehu vykonávania programu doň môžeme pridávať nové prvky (na koniec, na začiatok, dovnútra – pomocou operátora + alebo metód *append()*, *insert()*), alebo ich odstraňovať (pomocou operátora *del*

alebo metód *pop()*, *remove()*, *clear()*). Aj prvky na jednotlivých pozíciách môžeme kedykoľvek zmeniť príkazom priradenia.

- Špeciálnou operáciou je ***priradenie do rezu***. Ak je rez uvedený na ľavej strane príkazu priradenia, na pravej strane musí byť nejaká postupnosť prvkov. Časť zoznamu špecifikovaná rezom sa v takom prípade nahradí prvkami z priraďovanej postupnosti. Napr. príkazom *a[3:5] = range(10)* nahradíme dvojicu prvkov *a[3]*, *a[4]* desiatimi prvkami *0..9*. Príkazom *a[:] = []* odstránime zo zoznamu všetky prvky.
- Aktuálny počet prvkov zoznamu zistíme pomocou funkcie *len()*. Pri spracovaní prvkov zoznamu sa nám zídu aj funkcie *min()*, *max()*, *sum()*, resp. metódy *index()*, *find()*, *count()*, *replace()* a pod.
- Prvky zoznamu najčastejšie spracúvame postupne v poradí zľava doprava v cykle *for*.
- Zoznam môže byť vstupným parametrom funkcie, funkcia tiež môže vracať zoznam ako výsledok svojho výpočtu. V prípade vstupného parametra typu zoznam vo funkcii pracujeme priamo so skutočným parametrom. Do funkcie sa odovzdáva referencia na skutočný parameter. Musíme preto myslieť na to, či chceme pôvodný zoznam zmeniť alebo nie.
- Prvky v zozname môžeme usporiadať pomocou funkcie *sorted()* alebo metódy *sort()*. Funkcia *sorted()* vytvorí nový usporiadaný zoznam z prvkov vstupného zoznamu. Metóda *sort()* usporiada prvky v pôvodnom zozname, pre ktorý ju zavoláme.
- Funkcia *map()* sa dá využiť v prípade, že na každý prvok zoznamu chceme aplikovať rovnakú funkciu. Zápisom *a = list(map(int, input().split())* získame zoznam celočíselných hodnôt. V prvom parametri sme uviedli funkciu *int.* Použije sa na každý prvok zoznamu reťazcov, ktorý je uvedený druhom parametri. Objekt vrátený funkciou *map()* je potrebné na záver pretypovať na zoznam.
- Ak vieme, že postupnosť hodnôt, s ktorou v programe pracujeme, sa nebude po vytvorení meniť, použijeme údajový typ *tuple*, teda n-ticu.
- N-tica je nemeniteľná (v angl. *immutable*). Počet prvkov ani ich hodnoty nie je možné po vytvorení n-tice zmeniť.
- S n-ticami vieme vykonávať rovnaké operácie ako s inými postupnosťami prvkov (indexovanie, rezanie, spájanie, porovnávanie, štandardné funkcie) s výnimkou pridávania, odstraňovania prvkov a priraďovanie nových hodnôt prvkom n-tice.
- Prvky v n-tici môžu byť rovnakého, ale aj rôzneho typu. Často v n-tici spájame údaje, ktoré logicky patria k sebe a reprezentujú nejaký skutočný objekt (napr. súradnice bodu, údaje o jednom študentovi a pod.)
- Prvky n-tice sa uvádzajú v okrúhlych zátvorkách. Prázdnu n-ticu zapisujeme: *()*, jednoprvkovú n-ticu s povinnou čiarkou za prvým prvkom: *(10,)* .
- Funkcia môže v n-tici vrátiť ako výsledok viacero hodnôt naraz. Za príkazom *return* stačí hodnoty vymenovať oddelené čiarkami, okrúhle zátvorky nie sú povinné.
- Hodnoty uložené v n-tici (teda aj n-ticu vrátenú funkciou) je možné príkazom priradenia rozbaliť do príslušného počtu premenných.
- Údajový typ akejkoľvek hodnoty či premennej vieme ľahko overiť pomocou funkcie *type()*.

TÉMA 5: TEXTOVÉ SÚBORY

Bežne používané aplikácie spracúvajú (čítajú a vytvárajú) rôzne typy údajov, ktoré sú uložené v externých súboroch. V mnohých prípadoch ide o textový súbor, napr. vo formáte *.txt*, *.csv, .html.* Textový súbor je postupnosť znakov (vrátane znakov konca riadku *'\n'*). Možno naň nahliadať aj ako na postupnosť riadkov. Každý riadok je postupnosť znakov, ktorá je ukončená znakom konca riadku '\n'. Textový súbor môže obsahovať aj prázdne riadky. Také riadky obsahujú len znak konca riadku. Vo všeobecnosti zahŕňa práca so súborom 3 kroky: *otvorenie súboru, čítanie zo súboru* alebo *zapisovanie do súboru* a *zatvorenie súboru.*

PRACOVNÝ LIST Č. 1

Úloha 1: Obsah vstupného súboru vypíšeme na obrazovku. V priečinku s programom (súborom *.py*), pripravte textový súbor *vstup.txt*. Program spustite a otestujte viackrát pre rôzny obsah vstupného súboru (editujte ho). Vysvetlite význam príkazov v jednotlivých riadkoch programu:

```
try:
  f = open('vstup.txt', 'r')
  obsah = f.read()
  print(obsah)
  print(len(obsah))
  f.close()
except FileNotFoundError:
  print('subor neexistuje')
```

Úloha 2: Upravte nasledujúci program tak, aby na obrazovku vypísal počet číslic nachádzajúcich sa vo vstupnom súbore:

```
try:
  f = open('vstup.txt', 'r')
  obsah = read()

  for znak in obsah:
    # spracuj aktuálny znak

  print('v subore je', pocet, 'cislic')
  f.close()
except FileNotFoundError:
  print('subor neexistuje')
```

Úloha 3: Vo vstupnom súbore je na každom riadku jedno celé číslo. Vypočíta nasledujúci program ich súčet? Ak nie, prečo?

```
f = open('vstup.txt', 'r')
sucet = 0
for riadok in f:
  sucet += riadok
print(sucet)
f.close()
```

Poznámka: V ďalších úlohách vynecháme ošetrovanie výnimky *FileNotFoundError*. Budeme predpokladať, že vstupný súbor vždy existuje a je uložený v správnom priečinku.

Úloha 4: Predchádzajúcu úlohu sme mohli vyriešiť aj inak. Porovnajte nižšie uvedené riešenie s predchádzajúcim:

```
f = open('vstup.txt', 'r')
sucet = 0
riadky = f.readlines()
for riadok in riadky:
  sucet += int(riadok)
print(sucet)
f.close()
```

Úloha 5: Porovnajte uvedené dva programy. Spustite ich pre rovnaký textový súbor *vstup.txt*:

```
# Program 1
f = open('vstup.txt', 'r')
for riadok in f:
  print(riadok)
f.close()

# Program 2
f = open('vstup.txt', 'r')
for riadok in f:
  print(repr(riadok))
f.close()
```

Príklad vstupného súboru *vstup.txt*:

```
V tomto texte sa     nachadzaju aj medzery,
  aj znaky konca riadkov. A tiez prazdne riadky.

A aj tabulatory:      Koniec.
```

Úloha 6: Do vstupného súboru uložte niekoľko veršov básne. Napíšte program, ktorý vypíše prvých 5 znakov básne na obrazovku, každý znak na samostatný riadok. Využite vzorový program, v ktorom zo vstupného súboru čítame jeden znak:

```
f = open('basnicka.txt', 'r')
print(f.read(1))
f.close()
```

Úloha 7: Z textového súboru môžeme čítať rôznym spôsobom – prečítať jeho obsah naraz ako jeden znakový reťazec, postupne čítať riadky súboru alebo spracovávať súbor znak po znaku. Doteraz sme operáciu postupného čítania realizovali s využitím príkazu *for*. Ak čítame riadky či znaky zo súboru v cykle *while*, musíme vedieť rozlíšiť situáciu, keď sme už všetky riadky, resp. znaky súboru prečítali. Porovnajte uvedené štyri programy. Doplňte do nich výpisy poradových čísel znakov, resp. riadkov.

```
# Program 1
f = open('vstup.txt', 'r')
znak = f.read(1)
while znak != '':
  print(znak)
  znak = f.read(1)
f.close()

# Program 2
f = open('vstup.txt', 'r')
riadok = f.readline()
```

```
while riadok != '':
  print(riadok)
  riadok = f.readline()
f.close()

# Program 3
f = open('vstup.txt', 'r')
for riadok in f:
  for znak in riadok:
    print(znak)
f.close()

# Program 4
f = open('vstup.txt', 'r')
for riadok in f:
  print(riadok)
f.close()
```

Úloha 8: Napíšte program, ktorý zistí, či je v súbore znak ' ' (medzera). Ak áno, vypíšte, koľký znak v poradí to je. Ak nie, vypíšte správu *'ziadna medzera'***.**

Úloha 9: Textová správa (SMS) môže obsahovať najviac 160 znakov. Vstupný súbor obsahuje text. Cena jednej SMS správy je 0.10 Eur. Napíšte program, ktorý vypočíta a vypíše, koľko bude stáť odoslanie textu uloženého vo vstupnom súbore. Keď je text dlhší ako 160 znakov, rozdelí sa do viacerých SMS správ.

Úloha 10: Vo vstupnom textovom súbore sú viaceré riadky. Na začiatku alebo na konci niektorých riadkov boli pri editovaní omylom vložené nežiadúce medzery a tabulátory. Pri spracovaní riadkov by nám prekážali. Môžeme ich odstrániť pomocou metódy *strip()*. Pripravte si vstupný súbor a otestujte uvedený program:

```
f = open('vstup.txt', 'r')
for riadok in f:
  print(riadok)
  print(repr(riadok))
  print(riadok.strip())
f.close()
```

Úloha 11: Vo vstupnom textovom súbore sú viaceré riadky s údajmi o kúpenom tovare, jednotkovej cene a počte kusov. Názov tovaru, cena a počet kusov sú oddelené čiarkami. Doplňte program tak, aby vypočítal celkovú sumu zaplatenú za nákup. Vysvetlite význam reťazcovej metódy *split()*. Hodnotu akého typu vracia?

```
f = open('nakup.txt', 'r')
for riadok in f:
  casti = riadok.split(',')
  print(casti)
f.close()
```

Príklad vstupného súboru *nakup.txt*:

```
mys,25.50,5
podlozka,3.00,5
sluchadla,120.00,1
tablet,578.00,2
```

Úloha 12: Vo vstupnom súbore *narodeniny.txt* sú uvedené mená a dátumy narodenia. Zistite, koľkí sa narodili v zadanom mesiaci. Príklad vstupného súboru:

```
Janko;17.2.1989
Matusko;7.2.1990
Magda;13.2.1985
Jozko;27.5.1987
Alenka;8.12.1989
Hanka;1.5.1984
Jurko;22.11.1981
```

PRACOVNÝ LIST Č. 2

Úloha 1: Doplňte do programu príkazy tak, aby ste do výstupného súboru zapísali viaceré príslovia a porekadlá. Ako Python vie, že do súboru chcete zapisovať?

```
f = open('vystup.txt', 'w')
f.write('Kto chce psa bit, palicu si najde.')
f.write('\n')
f.close()
```

Úloha 2: Nasledujúci program skopíruje obsah vstupného súboru do výstupného súboru (vytvorí jeho kópiu). Upravte ho tak, že z pôvodného súboru vynecháte všetky prázdne riadky.

```
fr = open('vstup.txt', 'r')
fw = open('vystup.txt', 'w')
for riadok in fr:
  fw.write(riadok)
fr.close()
fw.close()
```

Úloha 3: Opravte program tak, aby z pôvodného vstupného súboru vynechal všetky medzery a všetky ostatné znaky skopíroval do výstupného súboru. Obsah pôvodného súboru vypíšte na obrazovku.

```
fr = open('vstup.txt', 'r')
text = fr.read()
for znak in text:
  f.write(znak)
f.close()
```

Úloha 4: Porovnajte nasledujúce programy. Vysvetlite, v čom sa líšia:

```
# Program 1
fr = open('in.txt','r')
fw = open('out.txt','w')
r = fr.readline()
fw.write(r)
fw.close()

# Program 2
fr = open('in.txt','r')
fw = open('out.txt','w')
r = fr.readline()
print(r, end='')
fw.close()
```

```
# Program 3
fr = open('in.txt', 'r')
fw = open('out.txt', 'w')
r = fr.readline()
print(r, end='', file=fw)
fw.close()
```

Úloha 5: Pri práci so súbormi môžeme používať príkaz *with*. Odkaz na otváraný súbor za priradí do premennej uvedenej za kľúčovým slovom *as*. Otvorené súbory nemusíme zatvárať, zatvoria sa automaticky na konci bloku *with*. Porovnajte nasledujúce tri použitia konštrukcie *with* v praxi:

```
# Príklad 1
with open('vstup.txt','r') as subor:
  for riadok in subor:
    print(riadok, end='')

# Príklad 2
with open('original.txt', 'r') as fr:
  with open('kopia.txt', 'w') as fw:
     fw.write(fr.read())

# Príklad 3
with open('original.txt', 'r') as fr, open('kopia.txt', 'w') as fw:
  fw.write(fr.read())
```

Úloha 6: Vo vstupnom súbore je na každom riadku meno zamestnanca a jeho plat. Oddeľovačom mena a sumy je čiarka. Uvedený program zisťuje, kto má najnižší plat a vytvorí výstupný textový súbor s informáciou o jeho 10 % zvýšení. Prepíšte zdrojový kód programu s využitím príkazu *with*.

```
fr = open('platy.txt', 'r')
fw = open('rozhodnutie.txt', 'w')
meno = ''
najnizsi = 10000
for riadok in f:
  casti = riadok.split(',')
  plat = float(casti[1])
  if plat < najnizsi:
    najnizsi = plat
    meno = casti[0]
print('Sprava pre mzdove oddelenie:', file=fw)
print(f'Zamestnanec {meno} bude mať plat {najnizsii*1.1} Eur.', file=fw)
fr.close()
fw.close()
```

Úloha 7: Napíšte program, ktorý do výstupného súbor *data.txt* zapíše požadovaný počet náhodne zvolených celých čísel zo zadaného intervalu. Čísla nech sú oddelené medzerami. Takýto súbor s náhodne vygenerovanými dátami by mohol poslúžiť na testovacie účely pri riešení iných problémov.

Úloha 8: Vymyslite zadanie zaujímavého, netriviálneho problému o spracovaní údajov uložených vo vstupnom textovom súbore a vytvorte program, ktorý ho vyrieši. Pripravte vstupný textový súbor (údaje v riadkoch nech sú oddelené napr. čiarkami). Výsledky spracovania vypíšte na obrazovku a zapíšte do výstupného textového súboru.

ZHRNUTIE

- Veľa súborov, s ktorými sa v praxi stretávame, má textový formát. Textový súbor sa ľahko edituje, pre používateľa je jeho obsah zrozumiteľný. Z programátorského hľadiska je textový súbor postupnosť znakov. Textové súbory často spracúvame po riadkoch. Každý riadok je ukončený znakom konca riadku *'\n'*.
- Súbor, s ktorým chceme pracovať, musíme najprv otvoriť pomocou funkcie *open()*. Prvým parametrom tejto funkcie je reťazec reprezentujúci názov súboru (vrátane cesty k nemu), v druhom parametri (tiež ide o reťazec) špecifikujeme, či súbor otvárame na čítanie, na zápis alebo na pripisovanie na konci súboru (*'r','w','a'* z angl. *read*, *write*, *append*).
- Ak súbor, ktorý otvárame na čítanie neexistuje, vznikne výnimka *FileNotFoundError*.
- Súbor, ktorý otvárame na zápis, nemusí na disku existovať. Ak neexistuje, vytvorí sa nový súbor. Ak existuje, prepíše sa (jeho pôvodný obsah stratíme).
- Textový súbor môžeme prečítať celý naraz ako jeden znakový reťazec – pomocou funkcie *read()*.
- Súbor obvykle čítame po riadkoch alebo po znakoch – pomocou funkcií *readline()* a *read(pocet_znakov)*, napr. *read(1*) v prípade čítania jedného znaku.
- Po prečítaní riadku alebo znaku sa ukazovateľ aktuálnej pozície v súbore presunie za práve prečítaný riadok, resp. znak. Na konci súboru vracajú funkcie *readline()* a *read()* prázdny reťazec.
- Funkcia *readlines()* vráti všetky riadky textového súboru ako zoznam reťazcov. Tento môžeme následne spracovať.
- Do súboru môžeme zapisovať pomocou funkcií *write()* aj *print()*. Funkcia *write()* zapisuje do súboru, pre ktorý ju zavoláme. Jej parametrom je zapisovaný reťazec. Ak chceme zabezpečiť odriadkovanie, musíme zapísať aj znak *'\n'*.
- Funkcia *print()* sa používa rovnako ako pri výpise na obrazovku. Jej voliteľný parameter *file* však musíme nastaviť tak, aby ukazoval na výstupný súbor.
- Po skončení práce so súborom je potrebné súbor zatvoriť – pomocou funkcie *close()*. Zatvorením súboru uvoľňujeme tento zdroj pre používanie iným programom. Tiež máme istotu, že obsah zapisovaný v programe sa do súboru na disk naozaj dostane.
- Príkaz *with* zjednodušuje zápis otvárania súborov. Súbory používané v bloku *with* nie je potrebné zatvárať, ich uzatvorenie prebehne automaticky.
- Pri spracúvaní textového súboru po riadkoch je obvykle potrebné analyzovať obsah prečítaného riadku a získať z neho údaje dôležité pre výpočet. Funkcia *split()* vracia zoznam s časťami riadku, ktoré oddeľuje znak uvedený ako parameter (napr. čiarka, bodkočiarka). Ak funkciu *split()* zavoláme bez parametrov, predpokladá sa, že oddeľovačom údajov v riadku je medzera.
- Ak prečítaný riadok obsahuje prebytočné biele znaky na začiatku alebo na konci, odstránime ich pomocou funkcie *strip()*. O skutočnom obsahu prečítaného reťazca (riadku) vrátane medzier, tabulátorov a znakov konca riadku sa ľahko presvedčíme pomocou funkcie *repr()*.

RIEŠENIA ÚLOH

TÉMA 1: FUNKCIE

Pracovný list č. 1

1. Niektoré funkcie nemajú parametre, niektoré majú 1 a viac parametrov. Niektoré funkcie nevracajú ako výsledok žiadnu hodnotu. Niektoré funkcie vracajú hodnotu nejakého typu:

```
import random
# funkcia input má 1 parameter – reťazec reprezentujúci správu
# pre používateľa, vracia reťazec zadaný používateľom
meno = input('Tvoje meno:')
```

```
# funkcia len má 1 parameter – reťazec, vracia celé číslo – počet znakov
vstupného reťazca
d = len(meno)
```

```
# funkcia print má 3 parametre, ktoré vypíše na obrazovku, nevracia ako
# výsledok žiadnu hodnotu
print('Tvoje meno ma', d,' pismen.')
```

```
# v tomto volaní funkcie print je len jediný vstupný parameter
print('Prazdniny su za dverami, hura!')
# v tomto prípade voláme funkciu print bez parametrov - odriadkujeme
print()
```

```
x = 123.456789
# funkcia round má 2 parametre – číslo a počet miest, na
# ktoré chceme zaokrúhľovať, na miesto volania vracia zaokrúhlené číslo
print(round(x, 2))
```

```
# funkcia randint má 2 celočíselné parametre, vracia náhodne zvolenú
# hodnotu z rozsahu 10..20
a = random.randint(10, 20)
print(a)
```

```
b = int(input('b: '))
# funkcia abs má 1 parameter (číslo), vracia jeho absolútnu hodnotu
# (číslo)
print(abs(b))
```

2. Hlavičky funkcií sme zvýraznili tučným, telá funkcií šikmým písmom. Názvy (mená) funkcií sú podfarbené žltou, parameter zelenou a miesta volania funkcií modrou farbou:

```
def pozdrav(m):
  print('Ahoj ' + m + '!', 'Poviem Ti basnicku:')

def basnicka():
  print('Poslali ma nasi k vasim,')
  print('aby prisli vasi k nasim.')
  print('Ak nepridu vasi k nasim,')
  print('Tak nepridu nasi k vasim!')
```

```
def koniec():
  print('Prajem Ti pekny den!')

meno = input('Tvoje meno: ')
pozdrav(meno)
basnicka()
koniec()
```

3. V hlavičke funkcie *vtip()* chýbali prázdne zátvorky (funkcia nemá parametre) a dvojbodka na konci hlavičky. Ak funkcia nemá parametre, v zápise jej volania tiež uvádzame prázdne zátvorky:

```
def vtip():
  print('Informatik je človek, ktorý sa v potravinách čuduje,')
  print('prečo nemá kilo mäsa 1024 gramov.')

# hlavný program
vtip()
```

4. Príkazy, ktoré tvoria telo funkcie, musia tvoriť blok odsedený zľava. V programoch sa osadzuje zľava jednotne o 2 alebo 4 znaky (napr. východisková hodnota v IDLE je 4).

```
def stredny_rod():
  print('mesto')
  print('srdce')
  print('vysvedčenie')
  print('dievča')

def muzsky_rod():
  print('chlap')
  print('hrdina')
  print('dub')
  print('stroj')

def zensky_rod():
  print('žena')
  print('ulica')
  print('dlaň')
  print('kosť')
```

5. Aby sme mohli funkciu zavolať (použiť), musí byť definovaná. V *Programe 1* sme najprv uviedli definície všetkých funkcií a potom sme ich zavolali v hlavnom programe (toto je obvyklý postup). V *Programe 2* volaniam jednotlivých funkcií predchádzajú ich definície, výpis na obrazovku bude v poriadku. V *Programe 3* stačí definíciu funkcie *tretia()* presunúť tak, aby predchádzala funkcii *druha()*, v ktorej tele ju voláme. Posledný príkaz bude volanie funkcie *druha()*:

```
def prva():
  print('A')

def tretia():
  print('C')

def druha():
  prva()
  print('B')
  tretia()

druha()
```

6. Vo volaní funkcie v hlavnom programe posielame do funkcie skutočné parametre (hodnoty premenných *a, b*). Premenné *x, y* sú formálne parametre, ktoré používame vo funkcii. Pri volaní funkcie nadobudnú hodnoty skutočných parametrov:

```
def aritmeticke_operacie(x, y):
  print('sucet:', x + y)
  print('sucin:', x * y)
  print('rozdiel:',x - y)
  if y != 0:
    print('podiel:', x // y)
  else!
    print('podiel: delenie nulou!')

# hlavný program
a = int(input('a = '))
b = int(input('b = '))
aritmeticke_operacie(a, b)
```

7. Pomenované parametre sme zvýraznili žltou farbou. Ostatné sú pozičné. Pri volaní funkcie musí programátor uvádzať pozičné parametre v správnom poradí. Pri pomenovaných parametroch na poradí nezáleží. Najprv sa však vždy uvádzajú pozičné parametre. V zakomentovanom riadku je syntaktická chyba, interpreter ju oznámi hlásením: *SyntaxError: positional argument follows keyword argument.*

```
x = 3, y = 4
print(x, y)
print(y, x)
print(x, y, end='')
print(x, y, sep='*')
# print(sep='*', x, y)          syntaktická chyba
print(x, y, end ='.', sep='+')
```

8. V *Programe 1* má funkcia *zaramuj()* 2 parametre. Druhý má definovanú východiskovú (v angl. *default*) hodnotu. Preto môžeme túto funkciu volať aj s jedným skutočným parametrom. V takom prípade sa pri orámovaní správy použije znak *'*'*. Funkciu *zaramuj()* nemôžeme volať bez parametrov, prvý pozičný parameter je povinný:

```
zaramuj('REKLAMA','o')
oooooooooo
oREKLAMAo
oooooooooo
```

```
zaramuj('REKLAMA')
*********
*REKLAMA*
*********
```

```
zaramuj()
TypeError: zaramuj() missing 1 required positional argument: 'sprava'
```

V *Programe 2* budú mať prvé dve volania rovnaký výsledok ako v *Programe 1*. Parametre *znak* a *medzera* majú definovanú východiskovú hodnotu. Ak bude mať tretí parameter hodnotu *True*, pred a za text správy sa pridá medzera:

```
zaramuj('REKLAMA', 'x', True)
xxxxxxxxxxx
x REKLAMA x
xxxxxxxxxxx
```

Ak do funkcie odovzdávame vstupné parametre s využitím ich mien, tak na poradí, v akom ich uvádzame, nezáleží. Z ostatných volaní funkcie je problematické len to, v ktorom sme pozičný parameter uviedli až za pomenovanými:

```
zaramuj(medzera=True, znak='x', 'REKLAMA')
SyntaxError: positional argument follows keyword argument.
```

Pracovný list č. 2

1.

```
def obvod(a, b):
  return 2 * (a + b)

def obsah(a, b):
  return a * b

# hlavný program
sirka = float(input('zadaj sirku obdlznika:'))
dlzka = float(input('zadaj dlzku obdlznika:'))
print('o =', obvod(sirka, dlzka))
print('S =', obsah(sirka, dlzka))
```

2.

```
def vyska_v_cm(m, cm):
  return 100 * m + cm
```

3.

```
def nacitaj_cislo():
  vstup = input('zadaj cele cislo:')
  return int(vstup)
```

4. Obe funkcie vracajú pre párne číslo hodnotu *True*. Druhý zápis je efektívnejší. Výraz *x%2 == 0* sa vyhodnotí. Výsledkom vyhodnotenia je logická hodnota, ktorú funkcia rovno vráti ako výsledok.

5. Počas dovolenky cez pracovné dni vstávame o deviatej, cez víkend budík nezvoní. Ak nie sme na dovolenke, budík je nastavený cez pracovné dni na 6:30 a cez víkend na 8:00:

```
print(budik(1,True))  # 9:00
print(budik(3))       # 6:30
print(budik(6))       # 8:00
print(budik(7,True))  # off
```

6. Áno, program môžeme spustiť a otestovať funkciu, ktorú už máme naprogramovanú. Príkazom *pass* nahradzujeme zatiaľ chýbajúce telo druhej funkcie.

7. Funkcia, ktorá overuje dokonalosť čísla, využíva pomocnú funkciu na výpočet súčtu deliteľov:

```
def je_dokonale(c):
  return sucet_delitelov(c)== 2*c
```

8. Globálna premenná *x* má na začiatku programu hodnotu *10*. V prvom prípade vidíme, že k nej máme prístup aj vo funkcii *fun1()*. Pre volaním funkcie, vo funkcii aj po skončení funkcie sme vypísali rovnakú hodnotu tej istej globálnej premennej:

```
fun1()
pred: 10
10
po: 10
```

Vo funkcii *fun2()* priraďujeme premennej *x* hodnotu *0*. Nezmeníme tak tým hodnotu globálnej premennej. Vytvorí sa lokálna premenná rovnakého mena, ktorá po skončení funkcie zanikne. Podľa výpisu hodnoty premennej *x* pred a po volaní funkcie vidíme, že vo funkcii sme s ňou nepracovali:

```
fun2()
pred: 10
0
po: 10
```

Vo funkcii *fun3()* chceme hodnotu premennej *x* zvýšiť o *1*. Vznikne však výnimka. Premenná x vo funkcii sa považuje za lokálnu. V momente vyhodnotenia výrazu na pravej strane príkazu priradenia ešte v pamäti neexistuje:

```
fun3()
pred: 10
Traceback (most recent call last):
  File "C:\..\ test.py", line 21, in <module>
    fun3()
  File "C:\..\test.py", line 11, in fun3
    x = x + 1
UnboundLocalError: cannot access local variable 'x' where it is not
associated with a value
```

Ak chceme vo funkcii pracovať s globálnou premennou, musíme to oznámiť uvedením kľúčového slova *global*. Podľa výpisu vidíme, že sme vo funkcii naozaj zmenili hodnotu globálnej premennej *x*:

```
fun4()
pred: 10
11
po: 11
```

Pracovný list č. 1

1. '\n' – znak konca riadku, '\t' – tabulátor, '\\' - spätná lomka

2. K základným operáciám s reťazcami patrí spájanie reťazcov pomocou operátora + a opakované spájanie pomocou operátora *:

```
>>> s1 = 'maxipes'
>>> s2 = 'fik'
>>> s1 + s2
'maxipesfik'
>>> s1 + ' ' + s2
'maxipes fik'
>>> s2 * 3
'fikfikfik'
>>> 'kukuk' * 10
'kukukkukukkukukkukukkukukkukukkukukkukukkukukkukuk'
>>> s = ''
>>> s += 'a'
>>> s += 'b'
>>> s
'ab'
```

3. Reťazec má len 13 znakov, posledný znak má index 12. Pokus o zobrazenie znaku na pozícii 13 skončí chybou (index je mimo prípustného rozsahu):

```
p
r
v
n
e
Traceback (most recent call last):
  File "C:\Users\...\test.py", line 7, in <module>
    print(s[13])
IndexError: string index out of range
```

4. V tejto úlohe vidíme, že pozíciu znaku môžeme vyjadriť aj celočíselným výrazom, tzv. *indexovým výrazom*, ktorý obsahuje premenné, operátory, zátvorky alebo aj volania funkcií:

```
o
r
v
r
g
```

5. Napr. *mog*, alebo ľubovoľné iné 3 náhodne zvolené znaky z reťazca *s*. Indexy znakov generujeme náhodne pomocou funkcie *randrange()* z modulu *random*. Každé volanie tejto funkcie vráti celé číslo z rozsahu *0..12*.

6.

	0	1	2	3	4	5	6	7	8	9	10	11	12
s	'p'	'r'	'o'	'g'	'r'	'a'	'm'	'o'	'v'	'a'	'n'	'i'	'e'

	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
s	'p'	'r'	'o'	'g'	'r'	'a'	'm'	'o'	'v'	'a'	'n'	'i'	'e'

```
e
i
e
False
True
Traceback (most recent call last):
  File "C:\Users\..\test.py", line 7, in <module>
    print(s[13])
IndexError: string index out of range
```

7.

```
meno = input('Tvoje meno:')
for znak in meno:
  print(znak)
```

8.

```
def pocet_medzier(s):
  n = 0
  for znak in s:
    if znak == ' ':
      n += 1
  return n

text = input()
print(pocet_medzier(text))
```

9. Dĺžku reťazca zistíme pomocou funkcie *len().* Reťazec *s3* má dĺžku *0*, neobsahuje žiadny znak:

```
s1 = 'Everest'
s2 = 'x'
s3 = ''
s4 = 'Tento text je dost dlhy na to, aby sa nam nechcelo pocitat jeho
znaky rucne. Pouzime funkciu len.'
print(len(s1))  # 7
print(len(s2))  # 1
print(len(s3))  # 0
print(len(s4))  # 97
```

10. Index posledného znaku je o 1 menší ako dĺžka reťazca:

```
text = input()
print(len(text))
print(text[len(text)-1])
```

11. Príklad vstupu a výstupu:

```
nitra
0. znak je n
1. znak je i
2. znak je t
3. znak je r
4. znak je a
```

12.

```
text = input()
```

```
for i in range(len(text)-1, -1, -1):
  print(str(i) + '. znak je ' + text[i])
```

Pracovný list č. 2

1. V uvedenom riešení overujeme postupne každý znak vstupného reťazca. Neskôr uvidíme aj iné, stručnejšie riešenie:

```
def obsahuje_znak(retazec, znak):
  for z in retazec:
    if z == znak:
      return True
  return False

text = 'Univerzita Konstantina Filozofa v Nitre'
hladany = 'f'
if obsahuje_znak(text, hladany):
  print('je tam')
else:
  print('nie je tam')
```

2. Reťazec *vysledok* je na začiatku funkcie prázdny. Postupne k nemu pripájame všetky znaky vstupného reťazca, ktoré sú rôzne od medzery:

```
def bez_medzier(retazec):
  vysledok = ''
  for znak in retazec:
    if znak != ' ':
      vysledok += znak
  return vysledok

text = input()
print('retazec zbaveny medzier:', bez_medzier(text))
```

3.

```
from random import randrange

def nahodny_binarny(dlzka):
  vysledok = ''
  for i in range(dlzka):
    vysledok += str(randrange(2))
  return vysledok

# vypíšeme ich na obrazovku viacej
for i in range(10):
  print(nahodny_binarny(8))
```

4. Ženy majú na pozícii 2 v rodnom čísle znak '5' alebo '6'. K mesiacu z dátumu narodenia sa totiž pripočítava hodnota 50:

```
def zisti_pohlavie(rc):
  if rc[2]=='5' or rc[2]=='6':
    return 'zena'
  return 'muz'

vstup = input('Tvoje rodne cislo:')
print('Si ' + zisti_pohlavie(vstup) + '.')
```

5.

```
vypis_kodov('VM')
znak V ma kod 86
znak M ma kod 77
```

6.

```
>>> vypis_znakov(9820, 9830)
9820 >>  ♜
9821 >>  ♝
9822 >>  ♞
9823 >>  ♟
9824 >>  ♠
9825 >>  ♡
9826 >>  ♢
9827 >>  ♣
9828 >>  ♔
9829 >>  ♥
9830 >>  ♦
```

7. Najprv je potrebné vypočítať ordinálne číslo znaku po posune. Číselný kód následne prevedieme opäť na znak:

```
def sifruj(sprava, posun=1):
  vysledok = ''
  for znak in sprava:
    novy_znak = chr(ord(znak) + posun)
    vysledok = vysledok + novy_znak
  return vysledok

print(sifruj('ABC'))                    # BCD
print(sifruj('ABC',3))                  # DEF
print(sifruj('Python je super',5))      # U~ymts%oj%xzujw
print(sifruj('mama',2))                 # ococ
```

8. Uvedieme len chýbajúce funkcie:

```
def pocet_velkych(slovo):
  pocet = 0
  for znak in slovo:
    if 'A'<= znak <= 'Z':
      pocet += 1
  return pocet

def pocet_cislic(slovo):
  pocet = 0
  for znak in slovo:
    if '0'<= znak <= '9':
      pocet += 1
  return pocet


vstup = input()
print('pocet malych:', pocet_malych(vstup))
print('pocet velkych:', pocet_velkych(vstup))
print('pocet cislic:', pocet_cislic(vstup))
```

9. Príklad vstupu a výstupu:

```
nitra
True
False
True
```

10.

```
def pocet_samohlasok(text):
  pocet = 0
  samohlasky = 'aeiouy'
  for znak in text:
    if znak in samohlasky:
      pocet += 1
  return pocet

slovo = 'jahoda'
print(f'pocet samohlasok: {pocet_samohlasok(slovo)}')
```

Pracovný list č. 3

1. Pri pokuse o zmenu znaku príkazom priradenia vznikne chyba *TypeError* – objekt typu str nepodporuje priradenie prvku:

```
>>> s = 'programovanie'
>>> s[3] = 'X'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[3] = 'X'
TypeError: 'str' object does not support item assignment
```

2. Žltou farbou sú vyznačené znaky reťazca tvoriace rez. Z výsledkov experimentov vidíme, že zápisom *s[od : do]* indexujeme rez, ktorý obsahuje znaky pôvodného reťazca na pozíciách *od, od+1, od+2, ... do-1*.

```
s = 'programovanie'
print(s[1:5])  # 'rogr'     'programovanie'
print(s[2:6])  # 'ogra'     'programovanie'
print(s[3:4])  # 'g'        'programovanie'
```

3. Ak vynecháme číslo za dvojbodkou, myslíme tým do konca reťazca. Ak vynecháme číslo pred dvojbodkou, myslíme tým od začiatku reťazca:

```
s = 'programovanie'
print(s[2:])        # 'ogramovanie'    'programovanie'
print(s[:7])        # 'program'        'programovanie'
```

4. Číslo za druhou dvojbodkou predstavuje *krok*. V prvom prípade sa do rezu vyberie znak na pozícii *2*, potom *4, 6* a *8* (keďže horná hranica je *10*, posledný znak rezu musí byť na nižšej pozícii). V druhom prípade za pri konštrukcii rezu začína na indexe *10* a postupne sa doň pridávajú znaky na pozíciách *9, 8, 7, 6, 5, 4* a *3* (znak s indexom *2* už nie je súčasťou rezu). Hodnota *-1* znamená, že postupujeme s krokom 1 sprava doľava.

```
s = 'programovanie'
print(s[2:10:2])    # 'ormv'           'programovanie'
print(s[10:2:-1])   # 'navomarg'       'programovanie'
```

5. V prvom prípade sa do rezu dostane každý tretí znak počnúc indexom *0* až po koniec reťazca. V druhom prípade získame reťazec obsahujúci všetky znaky pôvodného, ale v obrátenom poradí. Do rezu totiž pridávame všetky znaky od posledného až po nultý. Hodnotou kroku -1 sme zabezpečili, že postupujeme sprava doľava:

```
s = 'programovanie'
print(s[::3])       # 'pgmae'
print(s[::-1])      # 'einavomargorp'
```

6.

```
def je_palindrom(retazec):
  return retazec == retazec[::-1]

s = 'jelenovipivonelej'
if je_palindrom(s):
  print(s, 'je palindrom')
else:
  print(s, 'nie je palindrom')
```

7. Funkcia spracuje vstupný parameter – reťazec reprezentujúci zápis celého alebo desatinného čísla nasledovne: Ak ide o zápis čísla s desatinnou bodkou, zistí, na ktorej je pozícii. Následne urobí rezy z pôvodného reťazca – časť pred bodkou a časť za bodkou. Ako výsledok vráti reťazec, v ktorom sú tieto časti vymenené. Pre vstupný parameter reprezentujúci celé číslo bez desatinnej bodky funkcia vráti ten istý reťazec. Funkcia sa môže volať napr. *vymen_casti()*:

```
def vymen_casti(zapis_cisla):
  if '.' not in zapis_cisla:
    return zapis_cisla
  poz = 0
  while zapis_cisla[poz] != '.':
    poz = poz + 1
  pred = zapis_cisla[:poz]
  za = zapis_cisla[poz + 1:]
  return za + '.' + pred

cislo = input('cislo: ')
print(vymen_casti(cislo))
```

Príklady vstupu a výstupu:

```
cislo: 123
123
cislo: 3.14
14.3
cislo: 2023.007
007.2023
```

8. Znaky reťazca musíme najprv pretypovať na celé číslo, až potom je možné s nimi robiť celočíselné operácie. V tomto prípade chceme o každej číslici čísla zisťovať, či je deliteľná dvoma bezo zvyšku (teda či je párna).

9.

```
text = input('zadaj text: ')
print(text.lower())  # veľké písmená sa prevedú malé
print(text.upper())  # malé písmená sa prevedú na veľké
print()
```

```
print(text.islower())  # True, ak reťazec obsahuje len malé písmená
print(text.isupper())  # True, ak reťazec obsahuje len veľké písmená
print(text.isdigit())  # True, ak reťazec obsahuje iba číslice
print(text.isspace())  # True, ak reťazec obsahuje iba biele znaky
```

Príklad vstupu a výstupu:

```
zadaj text: NiTra
nitra
NITRA

False
False
False
False
```

10.

```
s = 'jablko je zdrave, hruska tiez, ale viac mi chuti jablko'

print(s.count('jablko'))  # vráti počet výskytov vstupného podreťazca
print(s.count('banan'))

print(s.find('hruska'))   # vráti pozíciu výskytu vstupného podreťazca
print(s.find('slivka'))   # alebo -1, ak sa v reťazci nenachádza

s = s.replace('jablko', 'ovocie')   # nahradí výskyty prvého druhým
s = s.replace('hruska', 'zelenina') # podreťazcom
s = s.replace('u','')
print(s)
```

Výstup programu:

```
2
0
18
-1
ovocie je zdrave, zelenina tiez, ale viac mi chti ovocie
```

Posledným volaním *s.replace()* sme všetky výskyty znaku *'u'* nahradili prázdnym reťazcom '' (inak povedané: odstránili sme všetky znaky *'u'*).

TÉMA 3: CHYBY V PROGRAME, VÝNIMKY

Pracovný list č. 1

1. Miesto syntaktickej chyby označil vo hlásení o chybe aj interpreter Pythonu. V zápise príkazu *for* sme zabudli na dvojbodku.

2. Správne poradie odpovedí: B, D, C, E, A

3. Áno. Meno premennej nemôže v Pythone začínať číslicou.

4. V programe chceme vypočítať aritmetický priemer z dvoch hodnôt zadaných na vstupe. Logickú chybu obsahuje príkaz priradenia:

```
priemer =  x + y / 2
```

Pre vstupné hodnoty *x == 1* a *y == 2* program vypíše výsledok *2.0*, hoci správne má byť *1.5*. Operácia delenia má však pri spracovaní výrazu vyššiu prioritu. Zabudli sme použiť zátvorky. Príkaz priradenia je potrebné opraviť takto:

```
priemer =  (x + y) / 2
```

5. Požiadavke v zadaní vyhovuje len *Riešenie B*. V *Riešení A* je horná hranica intervalu v *range()* o 1 menej ako požadovaná. V *Riešení C* je vymenené poradie parametrov v zápise *range()*.

Pracovný list č. 2

2.

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Nepodporované typy operandov (celé čísla a reťazce sa nedajú sčitovať).

```
ZeroDivisionError: division by zero
```

Delenie nulou.

```
IndexError: string index out of range
```

Index znaku v reťazci je mimo prípustného rozsahu.

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

Súbor, ktorý sme chceli otvoriť, sa na disku nenašiel.

```
TypeError: sin() takes exactly one argument (0 given)
```

Funkcia *sin()* má presne jeden vstupný parameter (nezadali sme žiadny).

```
NameError: name 'pocet' is not defined
```

Meno *pocet* nebolo definované (hodnotu premennej *pocet* sme v príklade chceli zvyšovať o 1).

6. Výstup programu pre nulový počet detí:

```
pocet deti: 0
pocet jablk: 10
typ chyby:  <class 'ZeroDivisionError'>
popis chyby:  integer division or modulo by zero
```

7.

```
# Riešenie a)
try:
  a = int(input())
  b = int(input())
  print(f'podiel: {a // b}, zvysok: {a % b}')
except Exception as chyba:
  print(type(chyba))
  print(chyba)

# Riešenie b)
try:
  a = int(input())
  b = int(input())
  print(f'podiel: {a // b}, zvysok: {a % b}')
except ValueError:
  print('Chyba: nebolo zadane cele cislo')
except ZeroDivisionError:
  print('Chyba: delenie nulou')
```

8. V druhom z riešení sme použili blok *else*:

```
# Riešenie 1
try:
  x = int(input())
  y = int(input())
  print(x**y)
except ValueError:
  print('ValueError')

# Riešenie 2
try:
  x = int(input())
  y = int(input())
except ValueError:
  print('ValueError')
else:
  print(x**y)
```

9.

```
sprava = input()
try:
  index = int(input())
  print(sprava[index])
except ValueError:
  print('ValueError')
except IndexError:
  print('IndexError')
```

10.

```
data = []
```

Pre prázdny zoznam vznikne výnimka *ValueError*.

```
data = [10, 20, 30, 'kuk']
```

Ak sú v zozname číselné a nečíselné údaje, vznikne výnimka *TypeError*. Pri hľadaní najväčšieho prvku je nutné hodnoty porovnávať. Celé číslo a reťazec nie je možné porovnať.

```
data = ['adam', 'matus', 'filip']
```

Na obrazovku sa vypíše výsledok. Najväčším prvkom je reťazec *'matus'* (porovnáva sa v abecednom poradí, t. j. lexikograficky).

```
data = [327, 63, 246, 128]
```

Na obrazovku sa vypíše výsledok. Najväčším prvkom je hodnota *327.*

```
data = [True, 10, 20]
```

Ak sú v zozname celé čísla a logické hodnoty, výnimka nevznikne. Hodnota *True* sa totiž dá skonvertovať na 1, hodnota *False* na 0.

Pracovný list č. 3

1. Pre nečíselný vstup vznikne výnimka *ValueError*, keďže funkcia *int()* nie je schopná takýto vstup skonvertovať na celé číslo. Pre nulový počet detí vznikne výnimka *ZeroDivisionError*. Reťazec *'dovidenia'* uvidíme na výstupe vždy, bez ohľadu na to, či vznikla alebo nevznikla nejaká výnimka.

2. *try* – C, *except* – B, *else* – D, *finally* – A

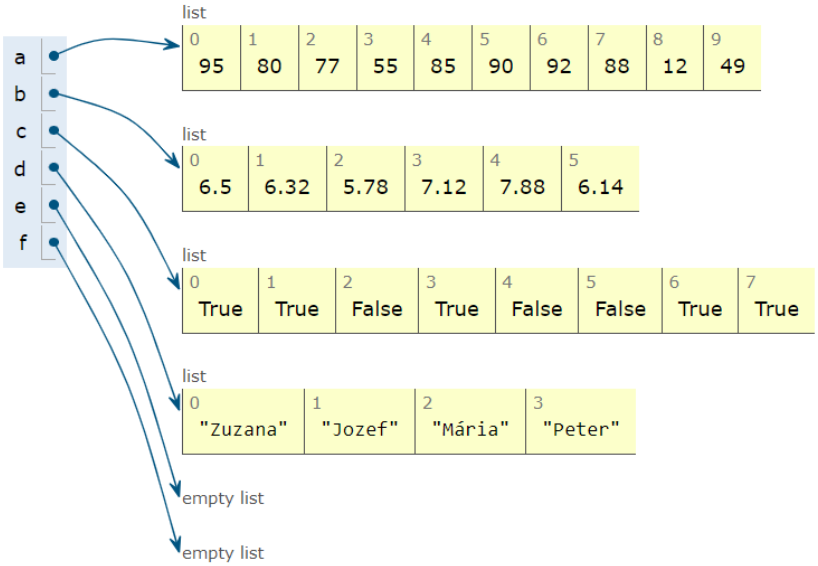
5. Vzorec na výpočet obsahu kruhu je v poriadku. Hoci aj pre zápornú hodnotu vstupného parametra *polomer* získame výsledok, ide o chybu. Hodnota polomeru nemôže byť záporná.

8. Zdrojom výnimky *ValueError* môže byť funkcia *int()* v hlavnom programe, ale aj funkcia *obsah_stvorca()*:

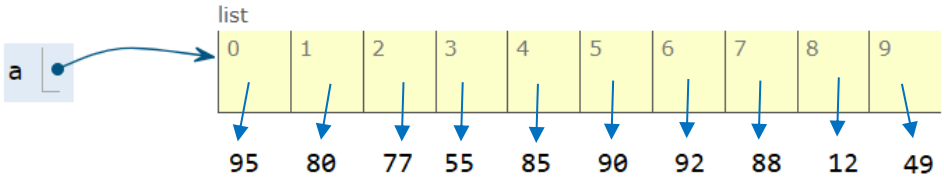
```
def obsah_stvorca(a):
  if a < 0:
    raise ValueError('Chyba: Dlzka strany nesmie byt zaporne cislo!')
  return a * a

# hlavný program
try:
  strana = int(input('zadaj dlzku strany:'))
  print(obsah_stvorca(strana))
except ValueError as chyba:
  print(chyba)
```

Pracovný list č. 1

1. Nasledujúca vizualizácia zoznamov bola vytvorená pomocou nástroja Python Tutor: https://pythontutor.com/. Premenné *a, b, c, d, e, f* sú odkazy (referencie) na jednotlivé zoznamy celých čísel, desatinných čísel, logických hodnôt a reťazcov. Posledné dva zoznamy sú prázdne.



Pozície prvkov sú vyznačené celočíselnými indexami. Prvý prvok v poradí má index *0*. V skutočnosti by sme mali aj jednotlivé prvky zoznamu znázorňovať ako referencie na príslušné hodnoty (v pamäti nie sú hodnoty prvkov umiestnené za sebou, v prvkoch zoznamu sú uložené odkazy na miesta v pamäti s príslušnými hodnotami), napr.:



Zoznam *a* by mohol uchovávať napr. body žiakov, ktoré dosiahli na záverečnom teste. V zozname *b* by mohlo ísť o dĺžky skokov v súťaži skokanov do diaľky. V zozname *c* by sme mohli mať uloženú informáciu o tom, ktoré vrcholy grafu sme už pri prehľadávaní navštívili a ktoré nie. Zoznam *d* môže obsahovať napr. mená prítomných žiakov.

2. Výstup programu:

```
10 6 8 4 0 0
```

3. Prvky zoznamu nemusia byť rovnakého typu. Môžu predstavovať napr. riadky tabuľky, v ktorej stĺpcoch chceme uchovávať rôzne údaje, napr.:

```
osoba = ['Zuzana', 'Mudra', 50, 172.5, True]
```

4. Výstup programu:

```
[1, 2, 3, 4, 5] [0, 0, 0] [1, 2, 3, 4, 5, 0, 0, 0] [0, 0, 0, 1, 2, 3, 4, 5] [100, 200, 300]
```

Príkazom *e = e + [100]* predĺžime zoznam o nový prvok s hodnotou *100*. Ide o spájanie existujúceho zoznamu *e* s jednoprvkovým zoznamom *[100].*

5.

```
n = int(input('pocet prvkov:'))
data = []
# do prázdneho zoznamu postupne pridáme celé čísla
for i in range(n):
  x = int(input('zadaj cislo:'))
  data += [x]                    # pozor, nie data += x !!!
print(data)
```

Príklad vstupu a výstupu:

```
pocet prvkov:5
zadaj cislo:1
zadaj cislo:2
zadaj cislo:3
zadaj cislo:4
zadaj cislo:5
[1, 2, 3, 4, 5]
```

6. Výstup programu pre vzorové vstupy:

```
['Danka', 'Janka', 'Petko', 'Palko'] ['0', '2000', '2500', '1800',
'1950', '2100'] ['1', '2', '3']
```

Metóda *split()* rozdelí reťazec na časti podľa znaku oddeľovača určeného v parametri. Vráti zoznam reťazcov. Ak ju zavoláme bez parametrov, za oddeľovač sa považuje znak medzery.

7. Výstup programu pre vzorový vstup:

```
['1', '2', '3', '4', '5', '2', '3', '1', '1', '3', '1', '2', '1', '2',
'1', '1', '4']
[1, 2, 3, 4, 5, 2, 3, 1, 1, 3, 1, 2, 1, 2, 1, 1, 4]
```

Zoznam reťazcov *vstup* vrátený metódou *split()* sme spracovali v cykle *for*. Do zoznamu *znamky* sme postupne pridávali prvky zoznamu *vstup* skonvertované na celé čísla.

8. Prvky zoznamu by sme mohli vymenovať alebo postupne pridať do prázdneho zoznamu v cykle. Prvý zo zápisov je však najpraktickejší:

```
data = [0] * 10

data = [0,0,0,0,0,0,0,0,0,0]

data = []
for i in range(10):
  data += [0]
```

9. Jeden z možných výstupov programu (vždy pôjde o zoznam, v ktorom budú mať všetky prvky rovnakú hodnotu):

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

Výraz *[random.randrange(10)] * 20* sa vyhodnotí nasledovne: funkcia *randrange()* vráti nejakú náhodnú hodnotu, napr. *5*. Jednoprvkový zoznam *[5]* následne vynásobíme číslom *20. Výsledkom* vyhodnotenia *[5] * 20* je *20*-prvkový zoznam pätiek.

Ak potrebujeme zoznam *20* prvkov s rôznymi náhodnými hodnotami, musíme každú hodnotu generovať osobitne:

```
import random
cisla = []
for i in range(20):
  cisla += [random.randrange(10)]
print(cisla)
```

Príklad výstupu:

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

10.

100 prvkový zoznam núl:

```
nuly = [0 for i in range (100)]
```

10 prvkový zoznam s prvkami *0, 1, 2, 3, ..., 9*:

```
a = [i for i in range (10)]
```

10 prvkový zoznam s prvkami *1, 3, 5, ..., 19* (nepárne čísla):

```
b = [2 * i + 1 for i in range(10)]
```

5 prvkový zoznam s prvkami *0, 1, 4, 9, 16* (druhé mocniny):

```
c = [i ** 2 for i in range(5)]
```

10 prvkový zoznam náhodných trojciferných čísel:

```
d = [randint(100, 999) for i in range(10)]
```

V zoznamu *e* budú posledné dvojčíslia rokov väčších ako 1900:

```
roky = [1812, 1867, 1945, 1969, 1971, 1980, 1984, 2012, 2015]
e = [rok % 100 for rok in roky if rok > 1900]
```

V zozname *f* budú prvé písmená slov (prevedené na veľké):

```
slova= ['toto', 'je', 'zoznam', 'slov']
f = [slovo[0].upper() for slovo in slova]
```

V zozname *g* budú len reťazce zo zoznamu *veci* s dĺžkou viac ako *5*:

```
veci = ['ceruzka', 'pero', 'guma', 'papier']
g = [vec for vec in veci if len(vec) > 5]
```

V zozname *h* bude 5 reťazcov reprezentujúcich hodnotu Ludolfovho čísla na 1..5 desatinných miest:

```
h = [str(round(pi, i)) for i in range(1, 6)]
```

Zoznam bude obsahovať všetky násobky *x * y* pre hodnoty *x, y* z určených intervalov:

```
nasobky = [x * y for x in range(1, 4) for y in range(1, 4)]
```

Vstupný reťazec obsahujúci čísla oddelené medzerou sa rozdelí metódou *split()* na časti, reťazce v získanom zozname sa pri generovaní nového zoznamu *cisla* konvertujú na celočíselné hodnoty:

```
cisla = [int(x) for x in input().split()]
```

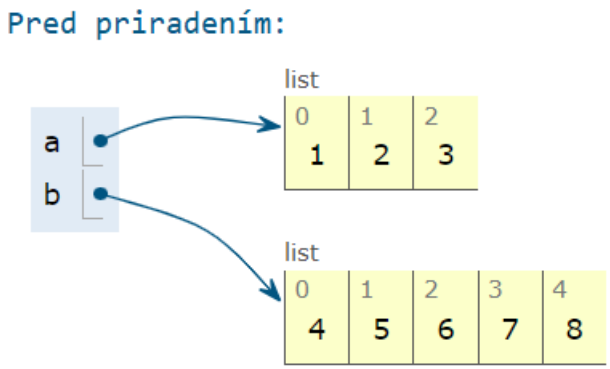
11. Príkazom *zoznam2 = list(zoznam)* sa v pamäti vytvoril nový, nezávislý zoznam s rovnakými prvkami. Riešenie poslednej časti úlohy:

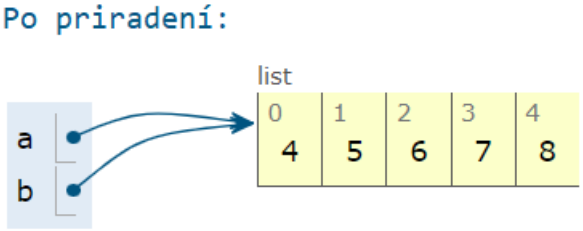
```
meno = input('Tvoje meno:')
pismena = list(meno[::-1].upper())
print(pismena)
```

Príklad vstupu a výstupu:

```
Tvoje meno:Simona
['A', 'N', 'O', 'M', 'I', 'S']
```

12. Priradením *a = b* dosiahneme to, že premenná *a* bude ukazovať na rovnaký zoznam ako premenná *b*. K pôvodnému zoznamu *[1, 2, 3]* sme stratili prístup. Ak naň neukazuje žiadna iná premenná, bude z pamäte automaticky odstránený:





Výstup programu:

```
[4, 5, 6, 7, 8]
[4, 5, 6, 7, 8]
```

Pracovný list č. 2

1. Výstup programu:

```
[60, 30, 100, 60]
```

3. Príkazom *del* odstránime prvok na príslušnej pozícii.

```
ziaci = ['Janko', 'Jurko', 'Katka', 'Hanka', 'Petko']
print(ziaci)
del ziaci[1]
print(ziaci)
del ziaci[-1]
print(ziaci)
ziaci[0] = 'Vierka'
print(ziaci)
```

```
ziaci[1] = 'Julia'
print(ziaci)
```

Výstup programu:

```
['Janko', 'Jurko', 'Katka', 'Hanka', 'Petko']
['Janko', 'Katka', 'Hanka', 'Petko']
['Janko', 'Katka', 'Hanka']
['Vierka', 'Katka', 'Hanka']
['Vierka', 'Julia', 'Hanka']
```

4. V prvých dvoch programoch vypisujeme prvky zľava doprava, v *Programoch 3* a *4* sprava doľava (teda odzadu). Ak pri prehľadávaní zoznamu nepotrebujeme pracovať s indexami prvkov, praktickejšie bude použiť zápis:

```
for x in zoznam:
  # spracovanie prvku s hodnotou x
```

V zápise s využitím indexovania nesmieme zabudnúť, že posledný prvok má index o 1 menej ako je dĺžka zoznamu:

```
for index in range(len(zoznam)):
  # spracovanie prvku zoznam[index]
```

Výstup vzorových programov:

```
1 2 3 4 5
1 2 3 4 5
5 4 3 2 1
5 4 3 2 1
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
```

V poslednej ukážke sme použili funkciu *enumerate()*, vďaka ktorej máme k dispozícii hodnotu aj pozíciu prvku súčasne.

5.

```
cisla = [int(x) for x in input('cisla:').split()]

a = int(input('a = '))
b = int(input('b = '))
pocet = 0
for x in cisla:
  if a <= x <= b:
    pocet += 1
print(pocet)
```

Príklad vstupu a výstupu:

```
cisla:5 10 15 20 25 30
a = 7
b = 22
3
```

6.

```
pocet_ziakov = int(input('pocet ziakov: '))
```

```
sucet = 0
znamky = []
for i in range(pocet_ziakov):
  znamka = int(input('znamka: '))
  znamky += [znamka]
  sucet += znamka

priemer = sucet / pocet_ziakov
print(f'priemer: {priemer}')
print('vsetky znamky: ', znamky)

print('nadpriemerni maju tieto znamky: ')
for znamka in znamky:
  if znamka < priemer:
    print(znamka)
```

Príklad vstupu a výstupu:

```
pocet ziakov: 5
znamka: 1
znamka: 2
znamka: 3
znamka: 4
znamka: 5
priemer: 3.0
vsetky znamky:  [1, 2, 3, 4, 5]
nadpriemerni maju tieto znamky:
1
2
```

7. Porovnajte 3 vzorové riešenia:

V prvom riešení inicializujeme pomocnú premennú zámerne dostatočne nízkou, v kontexte riešenej úlohy neplatnou teplotou. Po skončení cyklu *for* bude premenná *maxt* obsahovať najvyššiu teplotu:

```
# 1. riešenie
teploty = [10, 12, 13, 15, 16, 9, 9]
maxt = -1000
for t in teploty:
  if t > maxt:
    maxt = t
print(maxt)
```

V druhom riešení inicializujeme premennú *maxt* nultým prvkom zoznamu teplôt a následne kontrolujeme ostatné prvky zoznamu. Priebežne si pamätáme aj index zatiaľ najväčšieho prvku. Z tretieho riešenia je vidno, že nie je potrebné si osobitne pamätať hodnotu priebežného maxima. Stačí poznať jeho pozíciu:

```
# 2. riešenie
maxt = teploty[0]
maxi = 0
for i in range(1, len(teploty)):
  if teploty[i] > maxt:
    maxt = teploty[i]
    maxi = i
print(f'maximalna teplota: {maxt} °C')
print(f'bola v {maxi + 1}. den')
```

```
# 3. riešenie:
maxi = 0
for i in range(1, len(teploty)):
  if teploty[i] > teploty[maxi]:
    maxi = i
print(f'maximalna teplota: {teploty[maxi]} °C')
print(f'bola v {maxi + 1}. den')
```

8. Pre každý zoznam vyberieme 1 náhodný prvok (vygenerujeme náhodne index z prípustného rozsahu). Získané reťazce vypíšeme v tvare oznamovacej vety:

```
from random import randrange

kto = ['Matus', 'Filip', 'Dano', 'Alex', 'Maxim', 'Emil', 'Tibor']
corobi = ['tancuje', 'spieva', 'recituje', 'sa uci', 'spi', 'strikuje']
kde = ['v komore', 'na stanici', 'vo vani', 'v posteli', 'na ihrisku']
n = int(input('pocet viet:'))
for i in range(n):
  meno = kto[randrange(len(kto))]
  cinnost = corobi[randrange(len(corobi))]
  miesto = kde[randrange(len(kde))]
  print(f'{meno} {cinnost} {miesto}.')
```

V riešení sme mohli využiť aj funkciu *choice()* z modulu *random*, ktorá vráti náhodne zvolený prvok zo vstupného zoznamu:

```
>>> import random
>>> kto = ['Janko', 'Marienka', 'Petko', 'Palko', 'Misko']
>>> random.choice(kto)
'Janko'
>>> random.choice(kto)
'Palko'
```

9. Ak padne na kocke hodnota *h*, zvýšime hodnotu *p[h]* o 1. V programe generujeme hodnoty z rozsahu 0..5 (keďže prvky v zozname počítadiel indexujeme od nuly). Pre používateľa programu to nie je dôležité, stačí, že rozlišujeme 6 možných výsledkov náhodného pokusu:

```
import random

n = int(input('pocet hodov: '))

# počítadlá pre jednotlivé výsledky (može padnúť niektorá zo 6 hodnôt)
p = [0, 0, 0, 0, 0, 0]
# alebo p = [0] * 6

# simulácia
for i in range(n):
  h = random.randrange(6)
  p[h] += 1

# výpis výsledkov
for i in range(6):
  print(f'hodnota {i + 1} padla {p[i]} krat ->  {p[i]*100/n:.2f}%')
```

Príklad výstupu:

```
pocet hodov: 1000
```

```
hodnota 1 padla 184 krat ->  18.40%
hodnota 2 padla 179 krat ->  17.90%
hodnota 3 padla 157 krat ->  15.70%
hodnota 4 padla 147 krat ->  14.70%
hodnota 5 padla 156 krat ->  15.60%
hodnota 6 padla 177 krat ->  17.70%
```

10. Uvedené vzorové riešenie predpokladá, že najvyšší počet hlasov môže mať len jeden kandidát. V prípade viacerých víťazov s rovnakým počtom hlasov by bolo potrebné voľby opakovať:

```
# uvažujeme, ze kandidáti majú čísla 0, 1, 2, ... 9
kandidati = [0] * 10  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
hlasy = [int(x) for x in input('hlasy volicov:').split()]
for hlas in hlasy:
  kandidati[hlas] += 1
print('vysledky kandidátov:',kandidati)

# hľadanie víťaza
maxi, mini = 0, 0
for i in range(1, len(kandidati)):
  if kandidati[i] > kandidati[maxi]:
    maxi = i
  if kandidati[i] < kandidati[mini]:
    mini = i

rozdiel = kandidati[maxi] - kandidati[mini]
print(f'vitazom je kandidat c. {maxi}')
print(f'rozdiel medzi prvym a poslednym je {rozdiel} hlasov')
```

Príklad vstupu a výstupu:

```
hlasy volicov:1 1 2 1 2 1 2 1 1 1 1 2 3 4 5 8 4 4 4
vysledky kandidátov: [0, 8, 4, 1, 4, 1, 0, 0, 1, 0]
vitazom je kandidat c. 1
rozdiel medzi prvym a poslednym je 8 hlasov
```

11. Dom na pozícii *i* má ľavého suseda na pozícii *i-1* a pravého suseda na pozícii *i+1*. Domy na okraji ulice majú len jediného suseda, kontrolujeme ich preto osobitne:

```
ulica = [8, 1, 2, 12, 2, 1, 3, 4, 1, 1, 2, 2, 3]
kolko = 0
if ulica[0] > ulica[1]:
  kolko += 1

for i in range(1, len(ulica) - 1):
  if ulica[i] > ulica[i - 1] and ulica[i] > ulica[i + 1]:
    kolko += 1

if ulica[-1] > ulica[-2]:
  kolko += 1

print('Pocet domov s vyhladom:', kolko)
```

Výstup pre vzorový vstup (domy s výhľadom sú zvýraznené):

```
Pocet domov s vyhladom: 4
```

12. Vo vzorovom riešení *k* krát posunieme obsah zoznamu vpravo. Vždy si najprv zapamätáme hodnotou posledného prvku, ktorý sa po spracovaní ostatných prvkov presunie na 0. pozíciu (teda na začiatok zoznamu):

```
import random
a = [random.randrange(2) for i in range(10)]
k = int(input('posun:'))
print('pred posunom:',a)

for i in range(k):
  pom = a[-1]
  for j in range(len(a) - 1, 0, -1):
    a[j] = a[j - 1]
  a[0] = pom

print('po posune:', a)
```

Príklad vstupu a výstupu:

```
posun:2
pred posunom: [0, 1, 1, 1, 1, 0, 1, 1, 0, 1]
po posune: [0, 1, 0, 1, 1, 1, 1, 0, 1, 1]
```

Pracovný list č. 3

1. Metódou *append()* pridávame nový prvok na koniec zoznamu, metódou *insert()* môžeme vložiť nový prvok na konkrétnu pozíciu. Metóda *pop()* slúži na odstránenie prvku z konca zoznamu alebo nachádzajúceho sa na konkrétnej pozícii. Metódou *remove()* môžeme zo zoznamu odstrániť prvok s konkrétnou hodnotou. Metódy *pop()* a *remove()* vygenerujú v prípade neúspechu výnimky *IndexError*, resp. *ValueError*. Všetky prvky zo zoznamu odstráni metóda *clear()*:

Príklad vstupu a výstupu:

```
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
[1, 0, 2, 3, 4, 5, 100, 200]
200
[1, 0, 2, 3, 4, 5, 100]
1
[0, 2, 3, 4, 5, 100]
[0, 2, 3, 4, 5]
[]
```

2. Ukážka použitia metódy na vyhľadávanie v zozname:

```
a = list(range(10))
print(a)
x = int(input('x:'))
try:
  i = a.index(x)
  print('Hladany prvok je na pozicii:', i)
except ValueError:
  print('Hladany prvok sa v zozname nenachadza.')
```

Príklad vstupu a výstupu:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
x:12
Hladany prvok sa v zozname nenachadza.
```

4. Do funkcie sa odovzdáva referencia na skutočný parameter *a*, teda premenná *data* ukazuje na rovnaký zoznam ako premenná *a*. V tejto funkcii len čítame hodnoty prvkov a overujeme, či spĺňajú podmienku, obsah vstupného zoznamu *a* sa nemení:

```
def pocet_zapornych(data):
  pocet = 0
  for x in data:
    if x < 0:
      pocet += 1
  return pocet

# hlavný program
a = [5, -7, -10, 14, 33, 43, -45]
print(pocet_zapornych(a))
```

5.

```
def uprav_vysky(rastliny, koef):
  for i in range(len(rastliny)):
    rastliny[i] *= koef

# hlavný program
r = [1.5, 7.1, 4.25, 3.0, 5.5]
uprav_vysky(r, 2.0)
print(r)
uprav_vysky(r, 0.25)
print(r)
```

Výstup programu pre vzorový vstup:

```
[3.0, 14.2, 8.5, 6.0, 11.0]
[0.75, 3.55, 2.125, 1.5, 2.75]
```

V nižšie uvedenom nesprávnom riešení len meníme hodnotu pomocnej premennej *x*, ktorá pri prechode zoznamu nadobúda hodnoty prvkov zoznamu. Nemeníme obsah pôvodného zoznamu!

```
def uprav_vysky(rastliny, koef):
  for x in rastliny:
    x *= koef
```

6. Funkcie *sum()* by sme mohli využiť pri výpočte aritmetického priemeru zo známok. Funkciu *max()* či *min()* pri hľadaní najvyššej či najnižšej teploty. Obe tieto úlohy sme už riešili v predchádzajúcich pracovných listoch. Príslušné programy budú vďaka funkciám kratšie.

7.

```
def je_v_oboch(a, b, x):
  return (x in a) and (x in b)

def nie_je_v_ziadnom(a, b, x):
  return (x not in a) and (x not in b)

# hlavný program
a = [1, 2, 3]
```

```
b = [3, 4, 5, 6]
print(je_v_oboch(a, b, 3))
print(je_v_oboch(a, b, 1))
print(nie_je_v_ziadnom(a, b, 3))
print(nie_je_v_ziadnom(a, b, 99))
```

Výstup programu pre vzorový vstup:

```
True
False
False
True
```

8. Funkcia vo vzorovom riešení má 3 parametre, vďaka čomu je univerzálnejšia. Ak chceme, môžeme zvoliť aj iný reťazec, ktorým sa majú výskyty konkrétneho reťazca nahradiť:

```
def nahrad(data, co, cim = ''):
  for i in range(len(data)):
    if data[i] == co:
      data[i] = cim

# hlavný program
retazce = ['aaa', 'bbb', 'aaa', 'ccc', 'ddd']
nahrad(retazce, 'aaa')
print(retazce)
```

Výstup programu pre vzorový vstup:

```
['', 'bbb', '', 'ccc', 'ddd']
```

9.

```
def vytvor_zoznam(n, h=0):
  return [h] * n

# hlavný program
data = vytvor_zoznam(10)
print(data)
data = vytvor_zoznam(3, 1)
print(data)
```

Výstup programu:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1]
```

10.

```
import random

def vytvor_nahodny_zoznam(n, a, b):
  return [random.randint(a, b) for i in range(n)]

# hlavný program
data = vytvor_nahodny_zoznam(10, 100, 999)
print(data)
```

Výstup programu:

```
[578, 812, 701, 123, 853, 107, 677, 326, 984, 331]
```

11. Vo funkcii *zluc()* najprv vytvoríme nový prázdny zoznam, do ktorého postupne pridávame prvky zo zoznamov *a, b*. Pred pridaním každého ďalšieho prvku musíme overiť, ktorý z aktuálnych prvkov zo zoznamu *a* alebo *b* je menší. Po skončení prvého cyklu *while* je ešte potrebné spracovať prvky toho zoznamu, ktorý zostal neprázdny. Pomocné indexy *i, j* reprezentujú pozície aktuálne porovnávaných prvkov zo zoznamov *a, b*.

```
def zluc(a, b):
  i = j = 0
  c = []
  while i < len(a) and j < len(b):
    if a[i] <= b[j]:
      c += [a[i]]    # alebo c.append(a[i])
      i += 1
    else:
      c += [b[j]]
      j += 1
  while i < len(a):
    c += [a[i]]
    i += 1
  while j < len(b):
    c += [b[j]]
    j += 1
  return c


# hlavný program
prvy = [1, 5, 7, 9]
druhy = [2, 3, 8, 10, 11, 12]
print(zluc(prvy, druhy))
```

Výstup programu pre vzorový vstup:

```
[1, 2, 3, 5, 7, 8, 9, 10, 11, 12]
```

12. Stav zoznamu menila napr. funkcia *uprav_vysky()* z *Úlohy 5*. Stav zoznamu nemenila napr. funkcia *pocet_zapornych()* z *Úlohy 4*. Uvedieme ešte príklad dvoch ďalších funkcií, ktoré obracajú poradie prvkov vo vstupnom zozname. Z výstupu programu je vidno, že prvá funkcia vracia nový zoznam, s pôvodným nepracuje. Druhá funkcia vymení poradie prvkov „na mieste“, teda priamo v tom zozname, ktorý jej posielame ako vstupný parameter:

```
def obrat1(zoznam):
  return zoznam[::-1]

def obrat2(zoznam):
  n = len(zoznam)
  for i in range(n//2):
    zoznam[i], zoznam[n-i-1] = zoznam[n-i-1], zoznam[i]

# hlavný program
a = [1,2,3,4,5]
b = obrat1(a)
print('obrat1(a)')
print('a:', a)
print('b:', b)
print('obrat2(a)')
obrat2(a)
```

```
print('a:', a)
```

Výstup programu pre vzorový vstup:

```
obrat1(a)
a: [1, 2, 3, 4, 5]
b: [5, 4, 3, 2, 1]
obrat2(a)
a: [5, 4, 3, 2, 1]
```

13. Usporiadajme čísla uložené v zozname *data*:

```
>>> data = [5, 3, 8, 6, 1, 2]
>>> data.sort()
>>> data
[1, 2, 3, 5, 6, 8]
>>> data = [5, 3, 8, 6, 1, 2]
>>> sorted(data)
[1, 2, 3, 5, 6, 8]
>>> data
[5, 3, 8, 6, 1, 2]
```

Z experimentu vyššie vidno, že: Metóda *sort()* realizuje usporadúvanie „na mieste“, t. j. modifikuje zoznam, pre ktorý je volaná. Funkcia *sorted()* vytvorí nový zoznam, v ktorom budú prvky usporiadané vzostupne, a vráti ho ako výsledok.

14.

```
>>> mena = ['Barbora', 'Albert', 'Zuzana', 'Cyril', 'Jan', 'Jozef']
>>> mena.sort()
>>> mena
['Albert', 'Barbora', 'Cyril', 'Jan', 'Jozef', 'Zuzana']
>>> mena.sort(reverse=True)
>>> mena
['Zuzana', 'Jozef', 'Jan', 'Cyril', 'Barbora', 'Albert']
```

15.

```
cisla = list( map(float, input().split()) )
print(cisla)
```

Príklad vstupu a výstupu:

```
1.5 3.12 123.77 0.001
[1.5, 3.12, 123.77, 0.001]
```

16. Zoznamy sa porovnávajú takto:

- zodpovedajúce si prvky zo zoznamov sa postupne porovnávajú, až pokým nenastane prvá nezhoda,
- výsledok porovnania týchto prvkov je tiež výsledkom porovnania celých zoznamov,
- v prípade, že sú zoznamy rôzne dlhé, kratší zoznam sa považuje za menší,
- porovnávať možno len porovnateľné prvky (napr. operátory > a < sa nedajú zmysluplne použiť na porovnanie čísla a reťazca, operácia == alebo != by sa vyhodnotila správne)

Pracovný list č. 4

1. Vo vzorovom riešení vidíme 3 rôzne spôsoby ako indexovať posledný prvok zvolenej n-tice:

```
>>> dni_tyzdna[0]
'pondelok'
```

```
>>> dni_tyzdna[-1]
'nedela'
>>> dni_tyzdna[6]
'nedela'
>>> dni_tyzdna[len(dni_tyzdna)-1]
'nedela'
```

2. V ukážke nižšie vidíme, že pokus o priradenie novej hodnoty prvku n-tice viedol k vzniku výnimky *TypeError* ('tuple' objekt nepodporuje priradenie prvku)..

```
>>> mesto = ('Nitra', 48.30763, 18.08453)
>>> mesto[0]='Bratislava'
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    mesto[0]='Bratislava'
TypeError: 'tuple' object does not support item assignment
```

V n-tici sa zvyknú spájať údaje, ktoré patria logicky k sebe a nie je žiaduce ich neskôr prepísať inými hodnotami. V tomto prípade sme chceli zmeniť údaj o meste s príslušnými geografickými súradnicami. Celá n-tica by však už nedávala zmysel.

3. Môžeme použiť funkciu *type()* alebo vypísať hodnotu premennej na obrazovku (v okrúhlych zátvorkách je určite n-tica):

```
>>> type(data)
<class 'tuple'>
>>> data
(100, 200, 300)
```

4. V riešení nižšie je vidno, že funkcia *tuple()* môže mať ako vstupný parameter „hocičo“ tvorené prvkami, vrátane zoznamu. Prázdna n-tica nemá žiadny prvok, zapisujeme ju len prázdnou dvojicou okrúhlych zátvoriek.

```
>>> tuple('Python')
('P', 'y', 't', 'h', 'o', 'n')
>>> tuple(range(1,11))
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> tuple(['a', 'b', 'c'])
('a', 'b', 'c')
>>> list((0,-1, 1))
[0, -1, 1]
>>> tuple()
()
>>> list()
[]
```

5.

```
>>> melodia = ('C',)
>>> melodia
('C',)
>>> melodia = ('C', 'D')
>>> melodia
('C', 'D')
>>> melodia = ('C', 'D', 'E', 'F', 'G')
>>> melodia
('C', 'D', 'E', 'F', 'G')
```

7. Do prvej premennej sa priradila hodnota prvého prvku n-tice, do druhej premennej sa priradila hodnota druhého prvku n-tice a do tretej premennej sa priradila hodnota tretieho prvku n-tice. Hovoríme, že n-tica sa rozbalila do premenných.

8. Funkcia *div_mod()* vráti 2 hodnoty zabalené v n-tici a to celočíselný podiel a zvyšok po delení.

```
def div_mod(a, b):
  return a // b, a % b
```

Otestujme ju:

```
vysledok = div_mod(7, 2)
print(vysledok)                 # (3, 1)
print('podiel:', vysledok[0])   # podiel: 3
print('zvysok:', vysledok[1])   # zvysok: 1
```

Niekedy je potrebné pracovať iba s niektorými hodnotami vrátenej n-tice. V takom prípade sa využívajú indexy.

9. V praxi niektoré funkcie vracajú pre niektoré špeciálne situácie aj prázdnu n-ticu. Táto konkrétna funkcia však nemôže nikdy vrátiť prázdnu n-ticu. Pre vstupnú hodnotu 1 vráti jednoprvkovú n-ticu s deliteľom 1 (uvažujeme len korektné hodnoty vstupných parametrov, teda prirodzené čísla).

Hlavný program:

```
cislo = int(input('cislo:'))
d = delitele(cislo)
print('delitele:', d)
if len(d) == 2:
  print(f'{cislo} je prvocislo')
else:
  print(f'{cislo} nie je prvocislo')
```

Príklad vstupu a výstupu:

```
cislo:28
delitele: (1, 2, 4, 7, 14, 28)
28 nie je prvocislo
```

10.

```
def parne_neparne(cislo):
  p = n = 0
  cislo = str(cislo)
  for cifra in cislo:
    if int(cifra) % 2 == 0:
      p += 1
    else:
      n += 1
  return p, n

# hlavný program
x = int(input('cislo:'))
print(parne_neparne(x))
```

Príklad vstupu a výstupu:

```
cislo:2023
(3, 1)
```

11. V zápise jednoprvkovej n-tice chýbala čiarka. Vo funkcii celočíselný parameter skonvertujeme najprv na reťazec. Zjednoduší nám to prácu s jednotlivými ciframi čísla.

```
def rozober_na_cifry(cislo):
  vysledok = ()
  cislo = str(cislo)
  for znak in cislo:
    vysledok += (int(znak),)
  return vysledok

# hlavný program
x = int(input('cislo:'))
cifry = rozober_na_cifry(x)
print('cifry:', cifry)
print('najvacsia cifra:', max(cifry))
print('najmensia cifra:', min(cifry))
print('ciferny sucet:', sum(cifry))
```

Príklad vstupu a výstupu:

```
cislo:2023
cifry: (2, 0, 2, 3)
najvacsia cifra: 3
najmensia cifra: 0
ciferny sucet: 7
```

12.

```
def chod_podla_navodu(navod):
  x = y = 0
  for kam in navod:
    if kam == 'S':
      y += 1
    elif kam == 'J':
      y -= 1
    elif kam == 'V':
      x += 1
    else:
      x -= 1
  return x, y

# hlavný program
kod = input()
bod = chod_podla_navodu(kod)
print(f'{bod[0]},{bod[1]}')
```

Príklad vstupu a výstupu:

```
JJJVVVVVZSS
4,-1
```

Pracovný list č. 1

1.

```
try:                          # pri otváraní súboru môže vzniknúť chyba
  f = open('vstup.txt', 'r') # súbor otvoríme na čítanie, pomenujeme ho f
  obsah = f.read()            # celý obsah súboru získame ako 1 reťazec
  print(obsah)                # vypíšeme ho na obrazovku
  print(len(obsah))           # zistíme počet znakov v súbore
  f.close()                   # súbor uzavrieme
except FileNotFoundError:     # na zachytenú výnimku zareagujeme výpisom
  print('subor neexistuje')
```

2.

```
try:
  f = open('vstup.txt', 'r')
  obsah = f.read()
  cislice = '0123456789'
  pocet = 0
  for znak in obsah:
    if znak in cislice:
      pocet += 1
  print('v subore je', pocet, 'cislic')
  f.close()
except FileNotFoundError:
  print('subor neexistuje')
```

3. Premenná *riadok* je typu *str* (reťazec). Pred pripočítaním do celkového súčtu je potrebné tento reťazec skonvertovať na celé číslo: *int(riadok)*.

4. V riešení v Úlohe 3 sme spracúvali riadky postupne čítané zo súboru v cykle *for*. V riešení v Úlohe 4 sme najprv získali všetky riadky vstupného súboru ako zoznam reťazcov. V cykle *for* sme spracúvali prvky tohto zoznamu**.**

5. Vo výstupe *Programu 2* uvidíme obsah jednotlivých riadkov vrátane prítomných bielych znakov:

```
'V tomto texte sa     nachadzaju aj medzery,   \n'
'  aj znaky konca riadkov. A tiez prazdne riadky.\n'
'\n'
'A aj tabulatory:\t\tKoniec.\n'
```

6. K riešeniu úlohy môžeme pristúpiť dvojako: čítať 5 znakov jeden po druhom alebo prečítať naraz reťazec obsahujúci prvých 5 znakov. Predpokladáme, že vstupný súbor má viac ako 5 znakov:

```
# Riešenie 1
f = open('basnicka.txt', 'r')
for i in range(5):
  print(f.read(1))
f.close()

# Riešenie 2
f = open('basnicka.txt', 'r')
znaky = f.read(5)
for znak in znaky:
  print(znak)
f.close()
```

7. Vo výpise pre väčšiu názornosť využijeme funkciu *repr()*:

```
# Program 1
f = open('vstup.txt', 'r')
znak = f.read(1)
i = 0
while znak != '':
  i += 1
  print(f'{i}. {repr(znak)}')
  znak = f.read(1)
f.close()

# Program 2
f = open('vstup.txt', 'r')
riadok = f.readline()
i = 0
while riadok != '':
  i += 1
  print(f'{i}. {repr(riadok)}')
  riadok = f.readline()
f.close()

# Program 3
f = open('vstup.txt', 'r')
i = 0
for riadok in f:
  for znak in riadok:
    i += 1
    print(f'{i}. {repr(znak)}')
f.close()

# Program 4
f = open('vstup.txt', 'r')
i = 0
for riadok in f:
  i += 1
  print(f'{i}. {repr(riadok)}')
f.close()
```

8.

```
f = open('vstup.txt', 'r')
znak = f.read(1)
n = 1
while znak !='' and znak !=' ':
  n += 1
  znak = f.read(1)
if znak==' ':
  print(n)
else:
  print('ziadna medzera')
f.close()
```

9.

```
f = open('vstup.txt', 'r')
text =  f.read()
pocet_sprav = len(text) // 160
```

```
if len(text) % 160 > 0:
  pocet_sprav += 1
print(f'zaplatime: {pocet_sprav*0.10:.2f} Eur')
f.close()
```

10. V tele cyklu *for* vypisujeme reťazec najprv štandardným spôsobom. Pomocou funkcie *repr()* dosiahneme zobrazenie bielych znakov *'\t', '\n'*. Reťazec získaný zavolaním metódy *strip()* bude zbavený bielych znakov nachádzajúcich sa na začiatku alebo konci reťazca *riadok*.

11. Metóda *split()* vracia zoznam reťazcov, ktoré sú v tomto prípade vo vstupnom reťazci oddelené čiarkou.

```
f = open('nakup.txt', 'r')
suma = 0.0
for riadok in f:
  casti = riadok.split(',')
  #print(casti)
  jednotkova_cena = float(casti[1])
  pocet_kusov = int(casti[2])
  suma += jednotkova_cena * pocet_kusov
print(f'Cena nakupu: {suma:.2f} Eur')
f.close()
```

12.

```
mesiac = int(input('zadaj mesiac: '))
kolko = 0
f = open('narodeniny.txt', 'r'):
riadky = f.readlines()
for riadok in riadky:
  data = riadok.strip().split(';')
  # print(data)
  # print(data[0])
  # print(data[1])
  datum = data[1].split('.')   # z dátumu nás zaujíma údaj o mesiaci
  if int(datum[1])==mesiac:
    kolko += 1
print(kolko)
```

Pracovný list č. 2

1. Súbor otvoríme na zápis vtedy, keď v druhom parametri funkcie *open()* uvedieme reťazec *'w'*. Znak konca riadku musíme do výstupného súboru zapísať „ručne“:

```
f = open('vystup.txt', 'w')
f.write('Kto chce psa bit, palicu si najde.')
f.write('\n')
f.write('Nie je vsetko zlato, co sa blysti.\n')
f.write('Nekric hop, kym si nepreskocil.\n')
f.close()
```

2. Za prázdny riadok sa považuje aj taký, v ktorom boli len samé biele znaky, napr. medzery. Metóda *strip()* zbaví reťazec predstavujúci aktuálny riadok bielych znakov na začiatku a konci reťazca, vrátane znaku konca riadka.

```
fr = open('vstup.txt', 'r')
fw = open('vystup.txt', 'w')
for riadok in fr:
  if riadok.strip()!='':
```

```
    fw.write(riadok)
fr.close()
fw.close()
```

3.

```
fr = open('vstup.txt', 'r')
text = fr.read()
for znak in text:
  print(znak)
  if znak != ' ':
    f.write(znak)
f.close()
```

4. Programy sa líšia predposledným riadkom:

```
# do súboru fw sa zapíše reťazec r
fw.write(r)
# na obrazovku sa vypíše reťazec r, odriadkovať netreba, znak konca
# riadku je súčasťou reťazca r
print(r, end='')
# reťazec r sa zapíše do súboru fw (presmerovali sme výstupný prúd
# z obrazovky do súboru)
print(r, end='', file=fw)
```

5. V prvom programe sme otvorili jeden súbor na čítanie a jeho obsah po riadkoch opisujeme na obrazovku. V druhom príklade z jedného súboru čítame a do druhého zapisujeme. Tentokrát celý súbor naraz. Tretí program len ukazuje, že otvorenie viacerých súborov môžeme zapísať aj v jednom riadku. Súbory nebolo potrebné zatvárať, zatvorili sa automaticky.

6.

```
with open('platy.txt', 'r') as fr, open('rozhodnutie.txt', 'w') as fw:
  meno = ''
  najnizsi = 10000
  for riadok in fr:
    casti = riadok.split(',')
    plat = float(casti[1])
    if plat < najnizsi:
      najnizsi = plat
      meno = casti[0]
  print('Sprava pre mzdove oddelenie:', file=fw)
  print(f'{meno} bude mat odteraz plat {najnizsi*1.1:.2f} Eur.',file=fw)
```

7.

```
import random

def zapis_cisla(n, a, b, subor='data.txt'):
  with open(subor, 'w') as f:
    for i in range(n):
      cislo = random.randint(a, b)
      print(f'{cislo} ', end='', file=f)
      # alebo takto:
      # f.write(str(cislo)+' ')

# hlavný program
zapis_cisla(100,100,999)
zapis_cisla(10,0,1,'nuly_jednotky.txt')
```

8. Zadanie v tejto úlohe je otvorené, uvedieme jeden príklad možného správneho riešenia:

Zadanie problému:

Vo vstupnom textovom súbore si zberateľ zaznamenáva informácie o svojej zbierke euromincí. Pre každú z krajín si zapíše každú novú mincu, ktorú získa. Každú z nominálnych hodnôt eviduje len raz. Vytvorte program, ktorý do výstupného súboru zapíše na samostatné riadky tie štáty, z ktorých má zberateľ kompletnú zbierku.

Príklad vstupného súboru *euromince.txt*:

```
Slovensko;0.01;0.02;0.05;0.1;0.2;0.5;1;2
Nemecko;1;2
Francuzko;2;0.5;0.2
Spanielsko;0.02;0.05;0.1;0.2;0.5
Taliansko;1;2;0.01;0.02;0.05;0.1;0.2;0.5
```

Program riešiaci problém:

```
with open('euromince.txt', 'r') as fr, open('vystup.txt', 'w') as fw:
  riadky = fr.readlines()
  for riadok in riadky:
    data = riadok.strip().split(';')
    if len(data) == 9:
      fw.write(data[0] + '\n')
      print(data[0])
```

LITERATÚRA

Blaho, Andrej. 2023. Programovanie v Pythone 2022/2023. [Online] [Dátum: 29. 6. 2023] https://python.input.sk/.

Guniš, Ján a kol. 2020. Riešenie problémov a programovanie. Bratislava : Centrum vedecko-technických informácií SR, 2020. ISBN 978-80-89965-62-5.

Lovászová, Gabriela – Michaličková, Viera – Kvaššayová, Nika. 2022. Programovanie v jazyku Python : Vybrané kapitoly pre učiteľov informatiky. Nitra : UKF v Nitre, Prírodovedec č. 797, 2022. 80 s. ISBN 978-80-558-1951-8

Michaličková, Viera a kol. 2021. Python Intermediate. Nitra : UKF v Nitre, 2021. 167 s. ISBN 978-80-558-1784-2.

Michaličková, Viera. 2021. Python pre teenagerov – údajové štruktúry : Edícia Prírodovedec č. 733. Nitra : UKF, 2021. 156 s. ISBN 978-80-558-1641-8.

Python Software Foundation. 2023. Python 3.11 documentation. [Online] [Dátum: 29. 6. 2023] https://docs.python.org/3.11/.

Sedgewick, Robert – Wayne, Kevin – Dondero, Robert. 2015. Introduction to Programming in Python: An Interdisciplinary Approach 1st Edition. s.l. : Addison-Wesley Professional, 2015. ISBN-13: 9780134076430.

Vorderman, Carol. 2022. Programovanie pre deti. Slovart, 2022. 224 s. ISBN 9788055656434.

Názov:	**Programovanie v jazyku Python**
Podnázov:	Vybrané kapitoly v praktických úlohách
Autor:	Viera Michaličková

Vydavateľ:	Univerzita Konštantína Filozofa v Nitre
Edícia:	Prírodovedec č. 822
Formát:	A4
Rok vydania:	2023
Miesto vydania:	Nitra
Počet strán:	83

ISBN 978-80-558-2056-9

ISBN 978-80-558-2056-9