17. Výnimky

video prezentácia

výnimky

Zapíšme funkciu, ktorá prečíta celé číslo zo vstupu:

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

>>> cislo()
    zadaj cislo: 234
    234
```

Toto funguje, len ak zadáme korektný reťazec celého čísla. Spadne to s chybovou správou pri zle zadanom vstupe:

```
>>> cislo()
    zadaj cislo: 234a
    ...
    ValueError: invalid literal for int() with base 10: '234a'
```

Aby takýto prípad nenastal, vložíme pred volanie funkcie `int()` test, napríklad takto:

```
def test_cele_cislo(retazec):
    for znak in retazec:
        if znak not in '0123456789':
            return False
    return True

def cislo():
    vstup = input('zadaj cislo: ')
    if test_cele_cislo(vstup):
        return int(vstup)
    print('chybne zadane cele cislo')
```

Pripadne s opakovaným vstupom pri chybne zadanom čísle (hoci aj tento test `test_cele_cislo()` nie je dokonalý a niekedy prejde, aj keď nemá):

```
def cislo():
    while True:
        vstup = input('zadaj cislo: ')
        if test_cele_cislo(vstup):
            return int(vstup)
        print('*** chybne zadane cele cislo ***')
```

Takto ošetrený vstup už nespadne, ale oznámi chybu a pýta si nový vstup, napríklad:

```
>>> cislo()
    zadaj cislo: 234a
    *** chybne zadane cele cislo ***
    zadaj cislo: 234 a
    *** chybne zadane cele cislo ***
    zadaj cislo: 234
    234
>>>
```

try - except

Python umožňuje aj iný spôsob riešenia takýchto situácií: chybe sa nebudeme snažiť predísť, ale keď vznikne, tak ju „ošetríme“. Využijeme novú programovú konštrukciu `try` - `except`. Jej základný tvar je:

```
try:
    '''blok príkazov'''
except MenoChyby:
    '''ošetrenie chyby'''
```

Konštrukcia sa skladá z dvoch častí:

- príkazy medzi `try` a `except`
- príkazmi za `except`

Blok príkazov medzi `try` a `except` bude teraz Python spúšťať „opatrnejšie“, t.j. ak pri ich vykonávaní nastane uvedená chyba (meno chyby za `except`), vykonávanie bloku príkazov sa okamžite ukončí a pokračuje sa príkazmi za `except`, pritom Python zruší chybový stav, v ktorom sa práve nachádzal. Ďalej sa pokračuje v príkazoch za touto konštrukciou.

Ak pri opatrnejšom vykonávaní bloku príkazov uvedená chyba nenastane, tak príkazy za `except` sa preskočia a normálne sa pokračuje v príkazoch za konštrukciou.

Ak pri opatrnejšom vykonávaní bloku príkazov nastane iná chyba ako je uvedená v riadku `except`, tak táto konštrukcia túto chybu nespracuje, ale pokračuje sa tak, ako sme boli zvyknutí doteraz, t.j. celý program spadne a IDLE o tom vypíše chybovú správu.

Ukážme to na predchádzajúcom príklade (pomocnú funkciu `test_cele_cislo()` teraz už nepotrebujeme):

```
def cislo():
    while True:
        vstup = input('zadaj cislo: ')
        try:
            return int(vstup)
        except ValueError:
            print('*** chybne zadane cele cislo ***')
```

To isté by sa dalo zapísať aj niekoľkými inými spôsobmi, napríklad:

```
def cislo():
    while True:
        try:
```

```
            return int(input('zadaj cislo: '))
        except ValueError:
            print('*** chybne zadane cele cislo ***')

def cislo():
    while True:
        try:
            vysledok = int(input('zadaj cislo: '))
            break
        except ValueError:
            print('*** chybne zadane cele cislo ***')
    return vysledok

def cislo():
    ok = False
    while not ok:
        try:
            vysledok = int(input('zadaj cislo: '))
            ok = True
        except ValueError:
            print('*** chybne zadane cele cislo ***')
    return vysledok
```

Na chybové prípady odteraz nemusíme pozerať ako na niečo zlé, ale ako na výnimočné prípady (výnimky, t.j. **exception**), ktoré vieme veľmi šikovne vyriešiť. Len neošetrená výnimka spôsobí spadnutie nášho programu. Často to bude znamenať, že sme niečo zle naprogramovali, alebo sme nedomysleli nejaký špeciálny prípad.

Na predchádzajúcom príklade sme videli, že odteraz bude pre nás dôležité meno chyby (napríklad ako v predchádzajúcom príklade **ValueError**). Mená chýb nám prezradí Python, keď vyskúšame niektoré konkrétne situácie, napríklad:

```
>>> 1+'2'
    ...
    TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 12/0
    ...
    ZeroDivisionError: division by zero
>>> x+1
    ...
    NameError: name 'x' is not defined
>>> open('')
    ...
    FileNotFoundError: [Errno 2] No such file or directory: ''
>>> [1,2,3][10]
    ...
    IndexError: list index out of range
>>> 5()
    ...
    TypeError: 'int' object is not callable
>>> ''.x
    ...
    AttributeError: 'str' object has no attribute 'x'
>>> 2**10000/1.
    ...
    OverflowError: int too large to convert to float
>>> import x
    ...
```

```
    ImportError: No module named 'x'
>>> def t(): x += 1
>>> t()
    ...
    UnboundLocalError: local variable 'x' referenced before assignment
```

Všimnite si, že Python za meno chyby vypisuje aj nejaký komentár, ktorý nám môže pomôcť pri pochopení dôvodu chyby, resp. pri ladení. Tento text je ale mimo mena chyby, v `except` riadku ho nepíšeme. Takže, ak chceme odchytiť a spracovať nejakú konkrétnu chybu, jej meno si najjednoduchšie zistíme v dialógovom režime v IDLE.

Spracovanie viacerých výnimiek

Pomocou `try` a `except` môžeme zachytiť aj viac chýb ako jednu. V ďalšom príklade si funkcia vyžiada celé číslo, ktoré bude indexom do nejakého zoznamu. Funkcia potom vypíše hodnotu prvku s týmto indexom. Môžu tu nastať dve rôzne výnimky:

- **ValueError** pre zle zadané celé číslo indexu
- **IndexError** pre index mimo rozsah zoznamu

Zapíšme funkciu:

```
def zisti(zoznam):
    while True:
        try:
            vstup = input('zadaj index: ')
            index = int(vstup)
            print('prvok zoznamu =', zoznam[index])
            break
        except ValueError:
            print('*** chybne zadane cele cislo ***')
        except IndexError:
            print('*** index mimo rozsah zoznamu ***')
```

otestujeme:

```
>>> zisti(['prvy', 'druhy', 'treti', 'stvrty'])
    zadaj index: 22
    *** index mimo rozsah zoznamu ***
    zadaj index: 2.
    *** chybne zadane cele cislo ***
    zadaj index: 2
    prvok zoznamu = treti
>>>
```

To isté by sme dosiahli aj vtedy, keby sme to zapísali pomocou dvoch vnorených príkazov `try`:

```
def zisti(zoznam):
    while True:
        try:
            try:
                vstup = input('zadaj index: ')
                index = int(vstup)
```

```
                print('prvok zoznamu =', zoznam[index])
                break
            except ValueError:
                print('*** chybne zadane cele cislo ***')
        except IndexError:
            print('*** index mimo rozsah zoznamu ***')
```

Zlúčenie výnimiek

Niekedy sa môže hodiť, keď máme pre rôzne výnimky spoločný kód. Za `except` môžeme uviesť aj viac rôznych mien výnimiek, ale musíme ich uzavrieť do zátvoriek (urobiť z nich `tuple`), napríklad:

```
def zisti(zoznam):
    while True:
        try:
            print('prvok zoznamu =', zoznam[int(input('zadaj index: '))])
            break
        except (ValueError, IndexError):
            print('*** chybne zadany index zoznamu ***')
```

```
>>> zisti(['prvy', 'druhy', 'treti', 'stvrty'])
    zadaj index: 22
    *** chybne zadany index zoznamu ***
    zadaj index: 2.
    *** chybne zadany index zoznamu ***
    zadaj index: 2
    prvok zoznamu = treti
>>>
```

Uvedomte si, že pri takomto zlučovaní výnimiek môžeme stratiť detailnejšiu informáciu o tom, čo sa v skutočnosti udialo.

Príkaz `try` - `except` môžeme použiť aj bez uvedenia mena chyby: vtedy to označuje zachytenie všetkých typov chýb, napríklad:

```
def zisti(zoznam):
    while True:
        try:
            print('prvok zoznamu =', zoznam[int(input('zadaj index: '))])
            break
        except:
            print('*** chybne zadany index zoznamu ***')
```

Takýto spôsob použitia `try` - `except` sa ale **neodporúča**, skúste sa ho vyvarovať! Jeho používaním môžete ušetriť zopár minút pri hľadaní všetkých typov chýb, ktoré môžu vzniknúť pri vykonávaní daného bloku príkazov. Ale skúsenosti ukazujú, že môžete zase stratiť niekoľko hodín pri hľadaní chýb v takýchto programoch. Veľmi často sú takéto bloky `try` - `except` bez uvedenia výnimky zdrojom veľmi nečakaných chýb.

Ako funguje mechanizmus výnimiek

Kým sme nepoznali výnimky, ak nastala nejaká chyba v našej funkcii, dostali sme nejaký takýto výpis:

```
>>> fun1()
    Traceback (most recent call last):
      File "<pyshell#9>", line 1, in <module>
        fun1()
      File "p.py", line 13, in fun1
        fun2()
      File "p.py", line 16, in fun2
        fun3()
      File "p.py", line 19, in fun3
        fun4()
      File "p.py", line 22, in fun4
        int('x')
    ValueError: invalid literal for int() with base 10: 'x'
```

Python pri výskyte chyby (t.j. nejakej výnimky) hneď túto chybu nevypisuje, ale zisťuje, či sa nevyskytla v bloku príkazu `try` - `except`. Ak áno a meno chyby zodpovedá menu v `except`, vykoná definované príkazy pre túto výnimku.

Ak na danom mieste neexistuje obsluha tejto výnimky, vyskočí z momentálnej funkcie a zisťuje, či jej volanie v nadradenej funkcii nebolo chránené príkazom `try` - `except`. Ak áno, vykoná čo treba a na tomto mieste pokračuje ďalej, akoby sa žiadna chyba nevyskytla.

Ak ale ani v tejto funkcii nie je kontrola pomocou `try` - `except`, vyskakuje o úroveň vyššie a vyššie, až kým nepríde na najvyššiu úroveň, teda do dialógu IDLE. Keďže nik doteraz nezachytil túto výnimku, IDLE ju vypíše v nám známom tvare. V tomto výpise vidíme, ako sa Python „vynáral“ vyššie a vyššie.

Práca so súbormi

Pri práci so súbormi výnimky vznikajú veľmi často a nie je jednoduché ošetriť všetky situácie pomocou podmienených príkazov. Najčastejšou chybou je neexistujúci súbor:

```
try:
    with open('x.txt') as subor:
        cislo = int(subor.readline())
except FileNotFoundError:
    print('*** neexistujuci subor ***')
    cislo = 0
except (ValueError, TypeError):
    print('*** prvy riadok suboru neobsahuje cele cislo ***')
    cislo = 10
```

Prípadne sa môže hodiť pomocná funkcia, ktorá zistí, či súbor s daným menom existuje:

```
def existuje(meno_suboru):
    try:
        with open(meno_suboru):
            return True
```

```
    except (TypeError, OSError, FileNotFoundError):
        return False
```

Uvedomte si ale, že táto funkcia, aby zistila, či súbor existuje, ho otvorí a okamžite aj zatvorí. Z tohto dôvodu nasledovný kód nie je veľmi efektívny:

```
if existuje('x.txt'):
    with open('x.txt') as t:
        obsah = t.read()
else:
    obsah = ''
```

Vyvolanie výnimky

V niektorých situáciách sa nám môže hodiť vyvolanie vzniknutej chyby aj napriek tomu, že ju vieme zachytiť príkazom `try` - `except`. Slúži na to nový príkaz `raise`, ktorý má niekoľko variantov. Prvý z nich môžete vidieť v upravenej verzii funkcie `cislo()`. Funkcia sa najprv 3-krát pokúsi prečítať číslo zo vstupu, a ak sa jej to napriek tomu nepodarí, rezignuje a vyvolá známu chybu `ValueError`:

```
def cislo():
    pokus = 0
    while True:
        try:
            return int(input('zadaj cislo: '))
        except ValueError:
            pokus += 1
            if pokus >= 3:
                raise
            print('*** chybne zadane cele cislo ***')

>>> cislo()
    zadaj cislo: jeden
    *** chybne zadane cele cislo ***
    zadaj cislo: dva
    *** chybne zadane cele cislo ***
    zadaj cislo: tri
    ...
    ValueError: invalid literal for int() with base 10: 'tri'
```

Pomocou príkazu `raise` môžeme vyvolať nielen práve zachytenú výnimku, ale môžeme vyvolať ľubovoľnú inú chybu aj s vlastným komentárom, ktorý sa pri nej vypíše, napríklad:

```
raise ValueError('chybne zadane cele cislo')
raise ZeroDivisionError('delenie nulou')
raise TypeError('dnes sa ti vobec nedari')
```

Príklad s metódou index()

Poznáme už metódu `index()`, ktorá v zozname (typ `list`, `tuple` alebo `str`) nájde prvý výskyt nejakej hodnoty. Metóda je zaujímavá tým, že vyvolá výnimku `ValueError`, ak sa táto hodnota v zozname nenachádza. Kým sme nepoznali odchytávanie výnimiek, väčšinou sme museli najprv kontrolovať, či sa tam príslušný prvok nachádza a až potom, keď sa nachádza, volali sme `index()`, napríklad:

```
def nahrad(zoznam, h1, h2):
    if h1 in zoznam:
        i = zoznam.index(h1)
        zoznam[i] = h2
```

Pomocou `try` - `except` to vyriešime výrazne efektívnejšie:

```
def nahrad(zoznam, h1, h2):
    try:
        i = zoznam.index(h1)
        zoznam[i] = h2
    except ValueError:
        pass
```

Táto metóda `index()`, ktorá funguje pre jednorozmerné zoznamy, nás môže inšpirovať aj na úlohu, v ktorej budeme hľadať indexy do dvojrozmernej tabuľky. Napíšme funkciu `hladaj(zoznam, hodnota)`, ktorá hľadá prvý výskyt danej hodnoty v dvojrozmernom zozname a ak taký prvok nájde, vráti jeho číslo riadku a číslo stĺpca. Ak sa tam taký prvok nenachádza, funkcia by mala vyvolať rovnakú výnimku, ako to robila pôvodná metóda `index()`, t.j. `ValueError: hodnota is not in list`. Zapíšme riešenie dvoma vnorenými cyklami:

```
def hladaj(zoznam, hodnota):
    for r in range(len(zoznam)):
        for s in range(len(zoznam[r])):
            if zoznam[r][s] == hodnota:
                return r, s
    raise ValueError(f'{hodnota!r} is not in list')
```

alebo to isté zápis pomocou štandardnej funkcie `enumerate()`:

```
def hladaj(zoznam, hodnota):
    for r, riadok in enumerate(zoznam):
        for s, prvok in enumerate(riadok):
            if prvok == hodnota:
                return r, s
    raise ValueError(f'{hodnota!r} is not in list')
```

Hoci je toto správne riešenie, vieme ho zapísať aj efektívnejšie pomocou volania metódy `index()`:

```
def hladaj(zoznam, hodnota):
    for r, riadok in enumerate(zoznam):
        try:
            s = riadok.index(hodnota)
            return r, s
        except ValueError:
            pass
```

```
    raise ValueError(f'{hodnota!r} is not in list')
```

Ak si ale uvedomíme, že neúspešné hľadanie prvku v `r`-tom riadku zoznamu pomocou `index()` vyvolá presne tú istú chybu, ktorú sme zachytili a potom znovu vyvolali, môžeme to celé skrátiť takto:

```
def hladaj(zoznam, hodnota):
    for r, riadok in enumerate(zoznam):
        try:
            s = riadok.index(hodnota)
            return r, s
        except ValueError:
            if r == len(zoznam)-1:     # posledný prechod for-cyklom
                raise
```

Teda zachytená chyba `ValueError` v poslednom riadku dvojrozmerného zoznamu označuje, že sa hodnota nenachádza v žiadnom riadku zoznamu a teda sa opätovne vyvolá zachytená chyba. (Zamyslite sa, ako bude toto riešenie fungovať pre prázdny dvojrozmerný zoznam, teda zoznam, ktorý neobsahuje ani jeden riadok).

Vytváranie vlastných výnimiek

Keď chceme vytvoriť vlastný typ výnimky, musíme vytvoriť novú triedu odvodenú od základnej triedy `Exception`. Môže to vyzerať napríklad takto:

```
class MojaChyba(Exception): pass
```

Príkaz `pass` tu znamená, že nedefinujeme nič nové oproti základnej triede `Exception`. Použiť to môžeme napríklad takto:

```
def podiel(p1, p2):
    if not isinstance(p1, int):
        raise MojaChyba('prvy parameter nie je cele cislo')
    if not isinstance(p2, int):
        raise MojaChyba('druhy parameter nie je cele cislo')
    if p2 == 0:
        raise MojaChyba('neda sa delit nulou')
    return p1 // p2

>>> podiel(22, 3)
    7
>>> podiel(2.2, 3)
    ...
    MojaChyba: prvy parameter nie je cele cislo
>>> podiel(22, 3.3)
    ...
    MojaChyba: druhy parameter nie je cele cislo
>>> podiel(22, 0)
    ...
    MojaChyba: neda sa delit nulou
>>>
```

Kontrola pomocou assert

Ďalší nový príkaz `assert` slúži na kontrolu nejakej podmienky: ak táto podmienka nie je splnená, vyvolá sa výnimka `AssertionError` aj s uvedeným komentárom. Tvar tohto príkazu je:

```
assert podmienka, 'komentár'
```

Toto sa často používa pri ladení, keď potrebujeme mať istotu, že je splnená nejaká konkrétna podmienka (vlastnosť) a v prípade, že nie je, chceme radšej aby program spadol, ako pokračoval ďalej. Prepíšme funkciu `podiel()` tak, že namiesto `if` a `raise` zapíšeme volanie `assert`:

```
def podiel(p1, p2):
    assert isinstance(p1, int), 'prvy parameter nie je cele cislo'
    assert isinstance(p2, int), 'druhy parameter nie je cele cislo'
    assert p2 != 0, 'neda sa delit nulou'
    return p1 // p2
```

```
>>> podiel(2.2, 3)
    ...
    AssertionError: prvy parameter nie je cele cislo
>>> podiel(22, 3.3)
    ...
    AssertionError: druhy parameter nie je cele cislo
>>> podiel(22, 0)
    ...
    AssertionError: neda sa delit nulou
```

Príklad s triedou Ucet

Na minulých cvičeniach ste riešili príklad, v ktorom ste definovali triedu `Ucet`, resp. `UcetHeslo` aj jej metódy `vklad()` a `vyber()`. Ukážme riešenie trochu zjednodušeného zadania len s jednou triedou, zatiaľ bez ošetrovania výnimiek:

```
class UcetHeslo:
    def __init__(self, meno, heslo, suma=0):
        self.meno, self.heslo, self.suma = meno, heslo, suma

    def __str__(self):
        return f'ucet {self.meno} -> {self.suma} euro'

    def vklad(self, suma):
        if self.heslo == input(f'heslo pre {self.meno}? '):
            self.suma += suma
        else:
            print('chybne heslo')

    def vyber(self, suma):
        if self.heslo == input(f'heslo pre {self.meno}? '):
            if suma <= 0:
                return 0
            suma = min(self.suma, suma)
```

```
            self.suma -= suma
            return suma
        else:
            print('chybne heslo')

#---- test ----

mbank = UcetHeslo('mbank', 'heslo1', 100)
csob = UcetHeslo('csob', 'heslo2', 30)
print('*** stav uctov:', mbank, csob, sep='\n')
mbank.vklad(csob.vyber(55))
print('*** stav uctov:', mbank, csob, sep='\n')
```

Po spustení a zadaní správnych hesiel, dostávame takýto výpis:

```
*** stav uctov:
ucet mbank -> 100 euro
ucet csob -> 30 euro
heslo pre csob? heslo2
heslo pre mbank? heslo1
*** stav uctov:
ucet mbank -> 130 euro
ucet csob -> 0 euro
```

Zdá sa, že test prebehol v poriadku.

Prepíšme riešenie s využitím výnimiek. Zadefinujeme pritom vlastnú výnimku `ChybnaTransakcia` a budeme sa snažiť ošetriť všetky možné chybové situácie:

```
class ChybnaTransakcia(Exception): pass

class UcetHeslo:
    def __init__(self, meno, heslo, suma=0):
        self.meno, self.heslo, self.suma = meno, heslo, suma

    def __str__(self):
        return f'ucet {self.meno} -> {self.suma} euro'

    def kontrola(self):
        if self.heslo and self.heslo != input(f'heslo pre {self.meno}? '):
            raise ChybnaTransakcia('chybne heslo pre pristup k uctu ' + self.meno)

    def vklad(self, suma):
        self.kontrola()
        try:
            if suma <= 0:
                raise TypeError
            self.suma += suma
        except TypeError:
            raise ChybnaTransakcia('chybne zadana suma pre vklad na ucet ' + self.
meno)

    def vyber(self, suma):
        self.kontrola()
        try:
            if suma <= 0:
                raise TypeError
            suma = min(self.suma, suma)
```

```
            self.suma -= suma
            return suma
        except TypeError:
            raise ChybnaTransakcia('chybne zadana suma pre vyber z uctu ' + self.m
eno)

def prevod(ucet1, ucet2, suma):
    '''prevedie sumu z ucet1 na ucet2'''
    suma = ucet1.vyber(suma)
    try:
        ucet2.vklad(suma)
    except ChybnaTransakcia:
        ucet1.suma += suma   # vrati vybratu sumu na ucet1
        raise

#---- test ----

mbank = UcetHeslo('mbank', 'heslo1', 100)
csob = UcetHeslo('csob', 'heslo2', 30)
print('*** stav uctov:', mbank, csob, sep='\n')
prevod(csob, mbank, 55)
print('*** stav uctov:', mbank, csob, sep='\n')
```

Všimnite si, že pomocná funkcia `prevod()` sa snaží pri neúspešnej transakcii vrátiť vybratú sumu peňazí - hoci to asi nerobí úplne korektne …

Cvičenia

L.I.S.T.

- riešenia **aspoň 8 úloh** odovzdaj na úlohový server https://list.fmph.uniba.sk/
- pozri si Riešenie úloh 17. cvičenia

1. Do premennej `a` sme priradili reťazec `'7 3'`. Potom sme v príkazom riadku zapisovali pythonovské výrazy, ktoré obsahovali túto premennú `a`, okrem nej možno nejaké identifikátory, operátory, jednoznakové reťazce a z čísel len `0`. Zakaždým sme dostali nejakú chybovú správu. Pre každú túto chybovú správu nájdi pythonovský výraz, ktorý ju vyvolal. Nepoužívaj príkaz `raise`:

```
>>> a = '7 3'
>>> ...
    ValueError: ...
>>> ...
    TypeError: ...
>>> ...
    FileNotFoundError: ...
>>> ...
    ZeroDivisionError: ...
>>> ...
    AttributeError: ...
>>> ...
    NameError: ...
>>> ...
```

```
    AssertionError: ...
>>> ...
    OverflowError: ...
```

2. Napíš funkciu `cele(hodnota)`, ktorá pomocou `int` prevedie danú hodnotu na celé číslo. Ak sa to nedá, funkcia vráti `0`. Nepoužívaj `try-except` bez vymenovaných typov chýb. Malo by fungovať, napríklad:

```
>>> cele(12.3)
    12
>>> cele('13')
    13
>>> cele([])
    0
>>> cele('12.3')
    0
```

3. Napíš funkciu `desatinne(retazec)`, ktorá zistí, či je daný reťazec desatinným číslom. Funkcia vráti `True` alebo `False`. Napríklad:

```
>>> desatinne('123')
    True
>>> desatinne('  22.7 ')
    True
>>> desatinne('22/7')
    False
```

4. Napíš funkciu `zoznam(retazec)`, ktorá zo znakového reťazca `retazec` vyrobí zoznam čísel. Predpokladaj, že reťazec začína a končí znakmi `'['` a `']'` a prvky zoznamu sú oddelené znakmi `','`. Daj pozor, aby sa celé čísla v reťazci prerobili na celé, desatinné na desatinné a nečíselné prvky ignoruj. Napríklad:

```
>>> z = zoznam('[0, 1., 2, 3.14]')
>>> z
    [0, 1.0, 2, 3.14]
>>> zoznam('[0, -.1, None, +2, [7], a5, -3.14, "8"]')
    [0, -0.1, 2, -3.14]
```

5. Napíš funkciu `sucet(post)`, ktorá vypočíta „súčet“ prvkov postupnosti `post`. Predpokladaj, že všetky prvky sú rovnakého typu ako prvý prvok postupnosti. Ak je niektorý prvok iného typu ako prvý prvok, tak sa ho funkcia snaží najprv prekonvertovať na tento typ, ak sa ani to nedá, tak tento jeden prvok odignoruje. Napríklad:

```
>>> sucet([2, '3', 4.0, 'päť'])
    9
>>> sucet(['1', 2, 0.3, 'abc'])
    '120.3abc'
>>> sucet([[1,2], 3, '4x'])
```

```
    [1, 2, '4', 'x']
>>> sucet([(1, 2), (3, 4), [5]])
    (1, 2, 3, 4, 5)
>>> print(sucet([]))     # pre prázdnu postupnosť
    None
```

Pomôcka: ak potrebuješ pretypovať `b` na rovnaký typ ako je `a`, môžeš to urobiť takto: `type(a)(b)`.

6. V textovom súbore sa nachádza nejaký text, pričom slová sú oddelené medzerami. Tento súbor môže obsahovať aj čísla. Napíš funkciu `iba_cisla(meno_suboru)`, ktorá vráti zoznam celých čísel z tohto súboru. Ak sa súbor nedá otvoriť, funkcia vráti prádny zoznam. Napríklad, ak súbor obsahuje:

```
farmar ma 15 ovci a 127 sliepok
pricom 18. februara bolo od -15 do +2 stupnov
```

volanie funkcie vráti:

```
>>> print(iba_cisla('subor.txt'))
    [15, 127, -15, 2]
```

7. Napíš funkciu `daj_cislo(meno_suboru, index, vynimka=None)`, ktorá predpokladá, že daný súbor má v každom riadku po jednom celom čísle. Funkcia vráti číselnú hodnotu z tohto riadka a ak sa to nedá (napríklad súbor neexistuje, nemá dosť riadkov, nie je v tomto riadku iba jediná celočíselná hodnota), vyvolá príslušnú výnimku s upresňujúcim textom. Ak treti parameter chýba, v prípade chyby funkcia vráti `None`. Napríklad:

```
>>> daj_cislo('cvicenie.py', 17, IndexError)
    ...
    IndexError: chybne zadany index
>>> daj_cislo('cvicenie.txt', 17, IndexError)
    ...
    IndexError: neexistujuci subor
>>> daj_cislo('data.txt', 3, IndexError)
    ...
    IndexError: chybne cislo v zadanom riadku
>>> daj_cislo('data.txt', 1, IndexError)
    2020
>>> print(daj_cislo('data.txt', '1'))
    None
```

Nezabudni zatvoriť otvorený súbor.

8. Napíš funkciu `rgb(r, g, b)`, ktorá z troch celých čísel vyrobí znakový reťazec. ktorý reprezentuje príslušnú farbu vo formáte `'#rrggbb'`. Funkcia pomocou troch príkazov `assert` skontroluje, či sú všetky tri parametre v poriadku. Napríklad:

```
>>> rgb(100, 150, 20.0)
    ...
```

```
    AssertionError: chybny treti parameter b
>>> rgb(100, 350, 20.0)
    ...
    AssertionError: chybny druhy parameter g
>>> rgb('100', 350, 20.0)
    ...
    AssertionError: chybny prvy parameter r
>>> rgb(100, 150, 200)
    '#6496c8'
```

9. Napíš funkciu `sustavy(retazec)`, ktorá sa pokúsi daný reťazec previesť na číslo v rôznych číselných sústavách. Využi to, že štandardná funkcia `int()` môže byť zavolaná aj s druhým parametrom - číselnou sústavou, napríklad `int('ff', 16)` vráti 255, t.j. `'ff'` je v 16-ovej sústave číslo 255. Funkcia `sustavy()` vráti 17-prvkový zoznam, pričom i-ty prvok zoznamu obsahuje prevod daného reťazca na číslo v i-sústave. Ak sa to pre nejakú sústavu urobiť nedá, prvok zoznamu na danom mieste bude mať hodnotu `None`. Otestuj štandardnú funkciu `int`, napríklad:

```
>>> int('1101')
    1101
>>> int('1101', 2)      # dvojkova sustava
    13
>>> int('1101', 3)
    37
>>> int('1101', 16)
    4353
>>> int('3a')
    ...
    ValueError: invalid literal for int() with base 10: '3a'
>>> int('3a', 11)
    43
>>> int('3a', 16)
    58
```

Napríklad:

```
>>> sustavy('11')
    [11, None, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
>>> sustavy('1a1')
    [None, None, None, None, None, None, None, None, None, None, None, 232,
265, 300, 337, 376, 417]
>>> sustavy('FF')
    [None, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, 255]
>>> sustavy('x')
    [None, None, None, None, None, None, None, None, None, None, None, None
, None, None, None, None, None]
```

10. Zadefinuj metódy triedy `TelefonnyZoznam` podobnej z 15. cvičenia:

```
class TelefonnyZoznam:
    def __init__(self, meno_suboru=None):
        ...
```

```
14.
15.     def pridaj(self, meno, telefon):
16.         ...
17.
18.     def zisti(self, meno):
19.         ...
20.
21.     def citaj(self, meno_suboru):
22.         ...
23.
24.     def zapis(self, meno_suboru):
25.         ...
26.
27.     def vypis(self):
28.         ...
```

Metódy:

- `__init__(meno_suboru=None)` inicializuje `self.zoznam` a zavolá `self.citaj(meno_suboru)`, prípadnú výnimku bude teraz ignorovať
- `pridaj(meno, telefon)` pridá dvojicu (`tuple`) do zoznamu alebo nahradí, ak už také `meno` bolo v zozname
  - ak `meno` alebo `telefon` nie je reťazec, vyvolá výnimku `TypeError`
- `zisti(meno)` vráti príslušné telefónne číslo alebo vyvolá výnimku `KeyError`
- `citaj(meno_suboru)` prečíta obsah súboru (v každom riadku dva reťazce oddelené medzerou), pri chybe vyvolá `ValueError`
- `zapis(meno_suboru)` do každého riadka zapíše meno a telefón, oddelí medzerou
- `vypis()` vypíše do textovej plochy