INFORMATIKA

Vybrané matematické úlohy MO řešitelné pomocí žákovského programování (2. část)

LADISLAV PERK

Přírodovědecká fakulta UJEP, Ústí nad Labem

Úvod

Tento článek je přímým pokračováním článku, který byl publikován v předchozím čísle časopisu Matematika–Fyzika–Informatika [1]. V něm bylo představeno šest matematických úloh MO, ve kterých byla hledána řešení pomocí funkčních programů v jazyce Python. Cílem článku bylo popsat zdrojové kódy těchto programů. Zároveň byly představeny zdrojové kódy s využitím pokročilých syntaktických konstrukcí jazyka Python.

V podobném duchu je v tomto článku prezentováno dalších pět úloh. Stejně tak jako jsou u každé z těchto úloh také představeny zdrojové kódy s využitím pokročilých syntaktických konstrukcí jazyka Python.

V následující úloze 1 se pracuje s celočíselnými podíly, resp. podíly přirozených čísel, u kterých se posuzuje, zda je či není jejich podíl celočíselný.

Úloha 1 ([2])

Tři kamarádky veverky spolu vyrazily na sběr lískových oříšků. Zrzečka jich našla dvakrát víc než Pizizubka a Ouška dokonce třikrát víc než Pizizubka. Cestou domů si povídaly a přitom louskaly a jedly své oříšky. Pizizubka snědla polovinu všech oříšků, které nasbírala, Zrzečka třetinu všech svých oříšků a Ouška čtvrtinu těch svých. Doma veverky zjistily, že jim dohromady zbylo 196 oříšků. Kolik oříšků našla každá z veverek?

Autorské řešení: Označme Z jako počet oříšků Zrzečky, P jako počet oříšků Pizizubky a O jako počet oříšků Oušky. V úloze se hledají všechny trojice přirozených čísel (Z, P, O), které splňují všechny podmínky zadání úlohy.

V rámci rozboru zadání úlohy je nutné si uvědomit, že číselná hodnota Z závisí na číselné hodnotě P, stejně tak číselná hodnota O závisí na číselné hodnotě P; matematicky lze tuto situaci zapsat jako $Z = f(P)$ a $O = f(P)$. Proto není zapotřebí vyčíslovat Z a O, postačí vyčíslit P. Proto zavedeme jeden cyklus s pevným počtem opakování s vhodně pojmenovanou proměnnou.

Jak již bylo řečeno výše, proměnné Z a O číselně závisí na číselné hodnotě P, proto je třeba je číselně vyjádřit: (ř. 4 a 5). Po příchodu veverek domů budou pak počty oříšků následující:

$$P^* = \frac{P}{2}, \quad Z^* = \frac{2}{3}Z, \quad O^* = \frac{3}{4}O.$$

Protože pro vyčíslení počtů po příchodu veverek domů je uplatněno číselné dělení a zároveň víme, že výsledky těchto dělení musí být přirozená čísla (tj. celočíselné počty oříšků), pak je nutné při tvorbě zdrojového kódu uplatnit metodu, která posoudí celočíselnost těchto dělení. K tomu je vhodné uplatnit metodu `.is_integer()`, která nabývá hodnoty `True` v situaci, kdy číslo A (při volání metody `A.is_integer()`) je celým číslem; v opačném případě nabývá hodnoty `False`.

```
1  import math
2
3  for P in range(1, 1001):
4      Z= 2 * P
5      O= 3 * P
6
7      P2 = P / 2
8      Z3 = 2 * Z / 3
9      O4 = 3 * O / 4
10
11     if P2.is_integer() and Z3.is_integer() and O4.is_integer():
12         if P2 + Z3 + O4 ==  196:
13             print("P = ", P)
14             print("Z = ", Z)
15             print("O = ", O)
16             print()
```

Program vypíše jednu trojici přirozených čísel (P, Z, O): (48, 96, 144). Jednoduchou početní kontrolou lze ověřit správnost tohoto vypsaného výsledku. Zároveň je možné se přesvědčit o správnosti tohoto výsledku nahlédnutím do řešení MO. [2]

```
import math

P = 0
splneno = False

while splneno == False:
    P = P + 1
    Z = 2 * P
    O = 3 * P

    P2 = P / 2
    Z3 = 2 * Z / 3
    O4 = 3 * O / 4

    if P2.is_integer() and Z3.is_integer() and O4.is_integer():
        if P2 + Z3 + O4 == 196:
            print("P = ", P)
            print("Z = ", Z)
            print("O = ", O)
            splneno = True
```

Vypisované výsledky tohoto programu jsou shodné s vypisovanými výsledky předchozího programu. Oba programy jsou proto z hlediska vypisovaných výsledků identické.

V následující úloze 2 se pracuje s nejmenším společným násobkem a největším společným dělitelem dvou přirozených čísel. Úloha je náročnější jak z hlediska volby strategie řešení, tak také v náročnosti tvorby zdrojového kódu pro vyčíslení největšího společného dělitele, resp. nejmenšího společného násobku.

Úloha 2 ([3])

Určete všechny dvojice přirozených čísel a a b, pro něž platí

$$2 \cdot \mathrm{NSN}(a,b) + 3 \cdot \mathrm{NSD}(a,b) = ab, \qquad (*)$$

kde $\mathrm{NSN}(a,b)$ značí nejmenší společný násobek a $\mathrm{NSD}(a,b)$ největší společný dělitel přirozených čísel a a b.

Autorské řešení: V úloze se hledají všechny dvojice přirozených čísel (a,b), které splňují rovnici $(*)$, kde $\mathrm{NSN}(a,b)$, resp. $\mathrm{NSD}(a,b)$, je nejmenší společný násobek čísel a a b, resp. největší společný dělitel čísel a a b.

Pracuje se tedy s nejmenším společným násobkem $\mathrm{NSN}(a,b)$ a největším společným dělitelem $\mathrm{NSD}(a,b)$ čísel a a b, pro které platí vztah

$$a \cdot b = \mathrm{NSN}(a,b) \cdot \mathrm{NSD}(a,b). \qquad (1)$$

Vyčíslení $\mathrm{NSN}(a,b)$ a $\mathrm{NSD}(a,b)$ je možné pomocí jedné z uvedených možností:

(a) samostatné vyčíslení pomocí algoritmu $\mathrm{NSN}(a,b)$ a $\mathrm{NSD}(a,b)$;
(b) vyčíslení $\mathrm{NSN}(a,b)$ pomocí algoritmu a výpočet $\mathrm{NSD}(a,b)$ ze vzorce

$$\mathrm{NSD}(a,b) = \frac{a \cdot b}{\mathrm{NSN}(a,b)};$$

(c) vyčíslení $\mathrm{NSD}(a,b)$ pomocí algoritmu a výpočet $\mathrm{NSN}(a,b)$ ze vzorce

$$\mathrm{NSN}(a,b) = \frac{a \cdot b}{\mathrm{NSD}(a,b)}.$$

Možnost (a) je nevýhodná tím, že nevyužije vzorce (1). V prezentovaném řešení využijeme možnost (c). Budeme tedy vyčíslovat $\mathrm{NSD}(a,b)$ pomocí algoritmu a výpočet $\mathrm{NSN}(a,b)$ ze vzorce $\mathrm{NSN}(a,b) = \frac{a \cdot b}{\mathrm{NSD}(a,b)}$.

V rámci vyčíslování $\mathrm{NSD}(a,b)$ pomocí algoritmu je nutné si uvědomit, že největší společný dělitel čísel a a b je takové největší přirozené číslo, které beze zbytku dělí jak číslo a a zároveň číslo b. Zároveň platí, že největší společný dělitel čísel a a b nemůže být větší než menší z čísel a a b, tj. $\mathrm{NSD}(a,b) \leq \{a, b, \min(a,b)\}$. Z důvodu snížení výpočetní náročnosti bude hledán největší společný dělitel čísel a a b od 1 do $\min(a,b)$.

Pro účely vyčíslování největších společných dělitelů vytvoříme metodu `NSD(x,y)` s parametry `x` a `y` (ř. 8). Protože budeme hledat menší z čísel x a y, zavedeme proměnnou `mensi`, která nejprve bude nastavena na hodnotu v proměnné `x` (ř. 2) a v případě, že hodnota čísla v `y` je menší než hodnota `x` (ř. 3), pak bude přepsána hodnota v `mensi` na hodnotu `y` (ř. 4). Tím se zajistí, že v proměnné `mensi` bude uloženo z menších čísel `x` a `y`. Posléze v intervalu přirozených čísel od 1 do hodnoty v proměnné `mensi` pomocí cyklus s pevným počtem opakování s řídící proměnnou `i` (ř. 6) posuzujeme, zda přirozené číslo v `i` beze zbytku dělí číslo v `x` a zároveň `y` pomocí operátoru modulo `%` v konjunktivním tvaru pomocí spojky `and` v podmíněném příkazu `if` (ř. 7). V případě, že nastane tato situace, pak se uloží do pomocné proměnné `NSD` číselná hodnota `i` (ř. 8). Tím, že dochází k vzestupnému posuzování čísel v `i`, pak po ukončení cyklu s pevným počtem opakování se nachází hodnota největšího společného dělitele čísel v `x` a `y`. Tato číselná hodnota `NSD` bude návratová hodnota metody `NSD(x,y)` (ř. 9).

Pro účely vyčíslení nejmenších společných násobků vytvoříme metodu `NSN(x,y)` s parametry `x` a `y` (ř. 12). Využijeme vztahu $\mathrm{NSN}(x,y) = \frac{x \cdot y}{\mathrm{NSD}(x,y)}$ a tento vztah vyčíslíme s uložením do proměnné `NSN` (ř. 13). Tato číselná hodnota bude návratovou hodnotou metody `NSN(x,y)` (ř. 14).

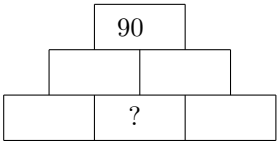
```
def NSD(x, y):
    mensi = a
    if b < mensi:
        mensi = b

    for i in range(1, mensi + 1):
        if x % i == 0 and y % i == 0:
            NSD = i
    return NSD


def NSN(x, y):
    NSN = x * y / NSD(x, y)
    return NSN


for a in range(1, 1001):
    for b in range(1, 1001):
        if 2 * NSN(a, b) + 3 * NSD(a, b) == a * b:
            print ('a = ', a)
            print ('b = ', b)
            print ()
```

Program vypíše (a, b): (3, 9), (5, 5) a (9, 3). Jednoduchou početní kontrolou lze ověřit správnost tohoto vypsaného výsledku. Zároveň je možné se přesvědčit o správnosti tohoto výsledku nahlédnutím do řešení MO.

V následující úloze 3 se do součinové pyramidy doplňují kladná celá čísla tak, aby číslo v nejvýše postaveném poli bylo 90. V rámci řešení hledáme všechny číselné hodnoty pouze pole nacházejícího se v nejnižší vrstvě uprostřed. Úloha je ukázkou práce s úlohami vyjadřujícími číselné vztahy pomocí grafického vyjádření.

Úloha 3 ([4])

V součinové pyramidě je v každém poli jedno kladné celé číslo, které je součinem čísel ze dvou sousedících polí z nižší vrstvy. Ve vrcholu trojvrstvé součinové pyramidy je číslo 90.

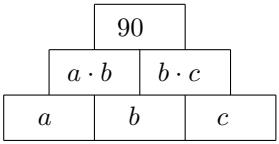


Jaké číslo může být ve vyznačeném poli? Určete všechny možnosti.

Autorské řešení: V úloze se hledají kladná celá čísla, která se mají doplnit do jednotlivých polí pyramidy s tím, že čísla ve vyšší vrstvě polí jsou součiny dotykových polí z nižší vrstvy.

Čísla v každém poli ve vyšší vrstvě jsou součiny čísel ze dvou sousedních polí nižší vrstvy, proto je nutné systematicky vyčíslovat pouze pole v nejnižší vrstvě; čísla v polích ve vyšší vrstvě pak vyjádřit jako příslušné součiny čísel polí z nižší vrstvy.

Pro účely vyjádření vztahů mezi jednotlivými poli pak zavedeme tři proměnné, např. a, b a c, pro tři pole v nejnižší vrstvě, zbylá pole pak budou vyčíslena dle následujícího obrázku:



Protože čísla v proměnných a, b a c mají být kladná celá čísla menší než 90, proto bez hlubšího matematického zamyšlení je možné každou z proměnných a, b a c vyčíslit od 1 do 89. Číslo 90 má vzniknout jako součin čísel $a \cdot b$ a $b \cdot c$, tedy $a \cdot b^2 \cdot c$. To znamená, že musí platit, že

$$a \cdot b^2 \cdot c = 90.$$

Pro účely zajištění vyčíslení proměnných a, b a c zavedeme tři vzájemně vnořené cykly s pevným počtem opakování se zavedením vhodně pojmenovaných proměnných `a`, `b` a `c` s vyčíslením od 1 do 88 (ř. 1–3). Podmínku pro posouzení rovnosti $a \cdot b^2 \cdot c = 90$ zrealizujeme pomocí podmíněného příkazu `if` (ř. 4). V případě kladného vyhodnocení této podmínky jsou vypsána čísla ze všech prázdných cihliček (ř. 5–12).

```
1  for a in range(1, 89):
2      for b in range(1, 89):
3          for c in range(1, 89):
4              if a * b * b * c == 90:
5                  print ('a = ', a)
6                  print ('b = ', b)
7                  print ('c = ', b)
8                  print ('a.b = ', a * b)
9                  print ('b.c = ', b * c)
10                 print ('a.b.b.c = ', a * b * b * c)
11                 print ('? = ', b)
12                 print ()
```

Program vypíše celkem 16 řešení s číselnými hodnota ve všech buňkách pyramidy. Ve výpisech se v poli s otazníkem vyskytuje pouze číslo 1 a 3. Nahlédnutím do řešení MO je možné se přesvědčit o správnosti těchto nalezených výsledků.

Úlohu se také možné také řešit s využitím pokročilých syntaktických konstrukcí. Místo uplatnění tří vzájemně vnořených *for* cyklů lze uplatnit metodu `product` se třemi proměnnými (ř. 3–6).

```
1 from itertools import product
2
3 for x in product(range(1, 10), repeat = 3):
4     a = x[0]
5     b = x[1]
6     c = x[2]
7     if a * b * b * c  ==  90:
8     ...
```

Vypisované výsledky tohoto programu jsou shodné s vypisovanými výsledky předchozího programu. Oba programy jsou proto z hlediska vypisovaných výsledků identické.

V následující úloze 4 pracujeme s kvadratickou rovnicí a jejími kořeny. Úloha je zajímavá tím, že v rámci tvorby zdrojového kódu musíme bezpodmínečně vyjádřit nejen kořeny kvadratické rovnice, ale také její diskriminant, podle kterého se rozhodne, zda úloha má ryze reálné kořeny.

Úloha 4 ([5])

Najděte všechny dvojice celých čísel a, b takových, že součet $a + b$ je kořenem rovnice

$$x^2 + ax + b = 0.$$

Autorské řešení: V úloze se hledají všechny takové dvojice celých čísel a a b, kdy alespoň jeden kořen kvadratické rovnice $x^2 + ax + b = 0$ je roven celému číslu $a + b$. V rámci řešení tedy hledáme vyčíslení kořenů rovnice $x^2 + ax + b = 0$ a porovnáváme alespoň jedno toto vyčíslení s hodnotou $a + b$.

Diskriminant rovnice $x^2 + ax + b = 0$ je $D = a^2 - 4 \cdot 1 \cdot b = a^2 - 4b$, druhá odmocnina diskriminantu pak $\sqrt{D} = \sqrt{a^2 - 4b}$. Kořeny kvadratické rovnice jsou

$$x_{1,2} = \frac{-a \pm \sqrt{D}}{2},$$

tzn. první kořen $x_1 = \frac{-a+\sqrt{D}}{2}$ a druhý kořen $x_2 = \frac{-a-\sqrt{D}}{2}$. Ze zadání plyne, že alespoň jeden kořen rovnice má být součtem $a + b$; protože ale rovnice může mít také dvojnásobný kořen (při $D = 0$), pak musí platit

$$(x_1 = a + b) \vee (x_2 = a + b) \vee (x_{1,2} = a + b).$$

Postačí ale pouze podmínka

$$(x_1 = a + b) \vee (x_2 = a + b).$$

```
import math

for a in range(-1000, 1001):
    for b in range(-1000, 1001):
        D = a * a - 4 * b

        if D >= 0:
            odmD = math.sqrt(D)

            x1 = (-a + odmD) / 2
            x2 = (-a - odmD) / 2
            koren = a +  b

            if x1 == koren or x2 == koren:
                print("a = ", a)
                print("b = ", b)
                print()
```

Program vypíše čtyři uspořádané dvojice (a, b): $(-6, 8)$, $(-6, 9)$, $(0, -1)$ a $(0, 0)$.

Úlohu se také možné také řešit s využitím pokročilých syntaktických konstrukcí. Místo uplatnění dvou vzájemně vnořených *for* cyklů lze uplatnit metodu `product` se dvěma proměnnými (ř. 4–6).

```
from itertools import product
import math

for x in product(range(1, 10), repeat = 2):
    a = x[0]
    b = x[1]
    D = a * a - 4 * b
    ...
```

Vypisované výsledky tohoto programu jsou shodné s vypisovanými výsledky předchozího programu. Oba programy jsou proto z hlediska vypisovaných výsledků identické.

V následující úloze 5 se pracuje s jedním algebrogramem s pěti písmeny.

Úloha 5 ([6])

Určete, kolik různých řešení má následující algebrogram. Každé písmeno odpovídá jedné číslici od 0 do 5, různá písmena odpovídají různým číslicím, stejná stejným.

K	O	S	A
S	A	K	O
B	A	B	A

Autorské řešení: V úloze se hledá počet všech pětic kladných celých čísel (A, B, K, O, S), které splňují rovnost

$$\overline{KOSA} + \overline{SAKO} = \overline{BABA}.$$

Proměnné (A, B, K, O, S) mohou nabývat celočíselných hodnot od 0 do 5.

Protože jsou číselné hodnoty proměnných A, B, K, O, S určeny jednoznačně, je vhodné pro účely jejich vyčíslení uplatnit pět vzájemně vnořených cyklů s pevným počtem opakování s vhodným pojmenováním řídících proměnných (ř. 1–5).

Protože ale musí být splněno, že číselné hodnoty v proměnných A, B, K, O, S musí být vzájemně různé, je nutné před další prací s těmito proměnnými posoudit vzájemnou číselnou různost. Protože obecně platí, že pokud platí nerovnost $X \neq Y$, pak není zapotřebí posuzovat nerovnost $Y \neq X$. Z toho plyne, že vzájemnou různost proměnných A, B, K, O, S lze vyjádřit jako

$$(A \neq B) \wedge (A \neq K) \wedge (A \neq O) \wedge (A \neq S) \wedge (B \neq K) \wedge \\ \wedge (B \neq O) \wedge (B \neq S) \wedge (K \neq O) \wedge (K \neq S) \wedge (O \neq S). \quad (2)$$

Je nutné provést dekadické vyjádření:

$\overline{KOSA}$ jako $1000 \cdot K + 100 \cdot O + 10 \cdot S + A$,

$\overline{SAKO}$ jako $1000 \cdot S + 100 \cdot A + 10 \cdot K + O$ a

$\overline{BABA}$ jako $1010 \cdot B + 101 \cdot A$ (ř. 11–13).

Pro účely vyhodnocení splnění rovnosti

$$\overline{KOSA} + \overline{SAKO} = \overline{BABA}$$

je nutné uplatnit podřízený příkaz (ř. 15). V případě kladného vyhodnocení této podmínky je možné přistoupit k inkrementaci proměnné registrující počet nalezených pětic (ř. 23), kterou je nutné před prohledáváním stavového prostoru vynulovat (ř. 1). Po ukončení prohledávání stavového prostoru je vhodné vypsat číselnou hodnotu proměnné registrující počet hledaných pětic (ř. 2). Tím došlo k vypsání hledaného počtu řešení.

```
1 pocet = 0
2
3 for A in range(0, 6):
4     for B in range(0, 6):
5         for K in range(0, 6):
6             for O in range(0, 6):
```

```
7                   for S in range(0, 6):
8
9                       if A!=B and A!=K and A!=O and A!=S and B!=K and
                            B!=O and B!=S and K!=O and K!=S and O!=S:
10
11                          KOSA = K * 1000 + O * 100 + S * 10 + A
12                          SAKO = S * 1000 + A * 100 + K * 10 + O
13                          BABA = B * 1000 + A * 100 + B * 10 + A
14
15                          if KOSA+SAKO == BABA:
16
17                              print("A = ", A)
18                              print("B = ", B)
19                              print("K = ", K)
20                              print("O = ", O)
21                              print("S = ", S)
22
23                              pocet=pocet+1
24                              print()
25
26  print("Pocet algebrogramu: ", pocet)
```

Program vypíše 16 algebrogramů ve tvaru (A, B, K, O, S):
$(1, 5, 2, 0, 3)$, $(1, 5, 3, 0, 2)$, $(2, 4, 1, 0, 3)$, $(2, 4, 3, 0, 1)$,
$(2, 5, 1, 0, 4)$, $(2, 5, 4, 0, 1)$, $(3, 5, 1, 0, 4)$, $(3, 5, 4, 0, 1)$,
$(4, 3, 1, 0, 2)$, $(4, 3, 2, 0, 1)$, $(4, 5, 2, 0, 3)$, $(4, 5, 3, 0, 2)$,
$(5, 3, 1, 0, 2)$, $(5, 3, 2, 0, 1)$, $(5, 4, 1, 0, 3)$, $(5, 4, 3, 0, 1)$.
Stejně tak program vypíše informaci o vypsaném počtu algebrogramů. Jednoduchou početní kontrolou lze ověřit správnost těchto vypsaných výsledků. Zároveň je možné se přesvědčit o správnosti těchto výsledků nahlédnutím do řešení MO.

Úlohu lze také řešit s využitím pokročilých syntaktických konstrukcí. Místo uplatnění pěti vzájemně vnořených *for* cyklů lze uplatnit metodu `product` s pěti proměnnými (ř. 6–11).

Posouzení vzájemné číselné různosti proměnných `A`, `B`, `K`, `O` a `S` je možné také realizovat s využitím pokročilých syntaktických konstrukcí. Pro účely posouzení vzájemné číselné různosti proměnných `A`, `B`, `K`, `O` a `S` lze místo pěti vzájemně vnořených *for* cyklů (ř. 1–3) uplatnit metodu `combinations` (ř. 3) z knihovny `itertools` (ř. 2). Na toto posuzování různosti lze totiž z hlediska kombinatoriky nahlížet jako na kombinace bez opakování. Z předchozího vyčíslení platí, že proměnná `A` je v proměnné `x[0]`, proměnná `B` je v proměnné `x[1]`, proměnná `K` je v proměnné `x[2]`, proměnná `O` je v proměnné `x[3]` a proměnná `S` je v proměnné `x[4]`. Indexy proměnné `x` nabývají celočíselných hodnot od 0 do 4. Vlastní posouzení vzájemné různosti číselných hodnot v proměnných `A`, `B`, `K`, `O` a `S` lze zrealizovat pomocí

metody `combinations` (číselný rozsah od 0 do 4 – indexace proměnných `A`, `B`, `K`, `O`, `S` a pro dvě proměnné `n[0]` a `n[1]`) v rámci *for* cyklu s řídící proměnnou `n` (ř. 14). V proměnných `n[0]` a `n[1]` jsou indexy 0 až 4 zapsané jako kombinace dvojic bez opakování; posouzení v rámci podmínky `x[n[0]]==x[n[1]]` je pak identické s posouzením podle vztahu (2).

Pro účely posuzování číselné různosti hodnot v proměnných `A`, `B`, `K`, `O` a `S` je algoritmicky nutné najít alespoň jednu číselnou dvojici, která je číselně shodná a tím zajistit, že není splněna podmínka vzájemné různosti číselných dvojic. Proto zavedeme pomocnou proměnnou, např. `ruznost`, kterou nastavíme na hodnotu `True` (ř. 13) a v případě nalezení alespoň jedné shodné číselné dvojice ji přepsat na hodnotu `False`(ř. 16). Program pak může dále pokračovat, pokud nedošlo k tomuto přepisu, tedy v proměnné `ruznost` je stále hodnota `True` (ř. 18).

```
1  from itertools import product
2  from itertools import combinations
3
4  pocet = 0
5
6  for x in product(range(0, 6), repeat = 5):
7      A = x[0]
8      B = x[1]
9      K = x[2]
10     O = x[3]
11     S = x[4]
12
13     ruznost=True
14     for n in combinations(range(0, 5), 2):
15         if x[n[0]] == x[n[1]]:
16             ruznost=False
17
18     if ruznost == True:
19         KOSA = K * 1000 + O * 100 + S * 10 + A
20  ...
```

Vypisované výsledky tohoto programu jsou shodné s vypisovanými výsledky předchozího programu. Oba programy jsou proto z hlediska vypisovaných výsledků identické.

Další možností je využití funkce v Pythonu `len()`, která vrací počet prvků v argumentu po odstranění duplicit. To znamená, že pokud uplatníme zápis `len({A, B, K, O, S})`, funkce `len()` vrátí počet proměnných obsahujících vzájemně různá čísla – tedy celá čísla od 1 do 5. Pokud mají být čísla v proměnných `A, B, K, O, S` vzájemně různá, pak počet `len({A, B, K, O, S})` bude roven číslu 5. Posouzení vzájemné číselné různosti proměnných `A`, `B`, `K`, `O`, `S` lze tedy zrealizovat pomocí podmínky `len({A, B, K, O, S}) == 5`.

Uvedený zápis demonstrujeme na následujícím zdrojovém kódu (ř. 12).

```
1  from itertools import product
2
3  pocet = 0
4
5  for x in product(range(0, 6), repeat = 5):
6      A = x[0]
7      B = x[1]
8      K = x[2]
9      O = x[3]
10     S = x[4]
11
12     if len({A, B, K, O, S}) == 5:
13         KOSA = K * 1000 + O * 100 + S * 10 + A
14 ...
```

Shrnutí

V předloženém příspěvku jsme prezentovali pět vybraných matematických úloh MO, které jsme řešili pomocí programování v jazyce Python. U všech úloh bylo představeno také řešení s využitím pokročilých syntaktických konstrukcí.

Literatura

[1] *Perk, L.*: Vybrané matematické úlohy MO řešitelné pomocí žákovského programování (1. část). Matematika–Fyzika–Informatika, roč. 34 (2025), č. 4, s. 299–311.

[2] MO ČR. 66. ročník, I. kolo, kategorie Z8, úloha 1.

[3] MO ČR. 69. ročník, domácí kolo, kategorie C, úloha 3.

[4] MO ČR. 70. ročník, II. kolo, kategorie Z6, úloha 2.

[5] MO ČR. 55. ročník, krajské kolo, kategorie A, úloha 1.

[6] MO ČR. 64. ročník, I. kolo, kategorie Z6, úloha 5.