# INFORMATIKA

# Vybrané matematické úlohy MO řešitelné pomocí žákovského programování (1. část)

*LADISLAV PERK*

Přírodovědecká fakulta UJEP, Ústí nad Labem

## Úvod

Tento článek je volným pokračováním článku, který seznamoval čtenáře s využitím programování při řešení vybraných matematických úloh MO v jazyce Python [1]. Na celkem sedmi úlohách byly ukázány jednodušší algoritmické konstrukce vybraných úloh MO, se kterými se čtenáři a zájemci při řešení těchto úloh mohou setkat. Jistou slabinou prvního článku bylo, že prezentoval relativně malý počet řešení a nevyužíval pokročilejší algoritmické konstrukce jazyka Python: cílem článku bylo seznámit čtenáře s danou problematikou a nezatěžovat jej pokročilejšími možnostmi jazyka Python.

Cílem tohoto článku je tento nedostatek vykompenzovat. Z tohoto důvodu je prezentován relativně větší počet úloh MO, jsou uplatněny další algoritmické konstrukce, které mohou čtenáři při řešení analogicky podobných úloh MO využít a také jsou použity pokročilé algoritmické konstrukce jazyka Python. S ohledem na rozsah prezentovaných řešení je proto článek rozdělen na dvě části. Tento článek je první částí dvoudílného článku.

V následující úloze 1 se pracuje se šesticifernými přirozenými čísly, přičemž se hledají jejich tři cifry tak, aby šesticiferné číslo bylo dělitelné 7, 8 a 9. V úloze se využívá dekadického zápisu víceciferných čísel a formulace podmínek dělitelnosti.

**Úloha 1** ([2])

Nahraďte hvězdičky v čísle 683∗∗∗ vhodnými číslicemi tak, aby výsledné šestimístné číslo bylo dělitelné 7, 8 a 9.

*Autorské řešení*: V úloze se hledají tři jednociferná kladná celá čísla, která po dosazení za hvězdičky vytvoří šesticiferné číslo, které má být dělitelné čísly 7, 8 a 9. Hvězdičky nahradíme třemi vhodně pojmenovanými proměnnými, např. $a$, $b$ a $c$. Číslo $\overline{683abc}$ má být dělitelné čísly 7, 8 a 9.

Pro účely vyčíslení proměnných $a$, $b$ a $c$ zavedeme tři vzájemně vnořené cykly s pevným počtem opakování s vhodně pojmenovanými proměnnými, např. `a`, `b` a `c`, s vyčíslením od 0 do 9 (ř. 1–3). Pro účely zvýšení přehlednosti zdrojového kódu je vhodné pro hledané šesticiferné číslo zavést pomocnou proměnnou, např. `cislo` s příslušným dekadickým vyjádřením (ř. 4). Nyní lze provést posouzení dělitelnosti beze zbytku hledaného šesticiferného čísla a čísel 7, 8 a 9. To lze zrealizovat pomocí podmíněného příkazu `if` s posouzením jednotlivých dělitelností v konjunktivním tvaru pomocí operátoru `and` a operátoru modulo `%` (ř. 6). V případě kladného vyhodnocení této podmínky lze přistoupit k výpisu číselné hodnoty hledaného šesticiferného čísla (ř. 7).

```
1 for a in range(0, 10):
2  for b in range(0, 10):
3    for c in range(0, 10):
4      cislo = 683 * 1000 + a * 100 + b * 10 + c
5
6      if cislo % 7 == 0 and cislo % 8 == 0 and cislo % 9 == 0:
7        print(cislo)
```

Program vypíše jako řešení dvě šesticiferná čísla, a to 683 424 a 683 928. Jednoduchou početní kontrolou lze ověřit správnost těchto dvou vypsaných výsledků. Zároveň je možné se přesvědčit o správnosti těchto výsledků nahlédnutím do řešení MO.

Proměnné `a`, `b` a `c` je možné také vyčíslovat s využitím pokročilých syntaktických konstrukcí. Pro účely vyčíslení proměnných `a`, `b` a `c` lze místo tří vzájemně vnořených *for* cyklů (ř. 1–3) uplatnit metodu `product` (ř. 3) z knihovny `itertools` (ř. 1), která realizuje vyčíslování proměnných `a`, `b`, `c` ekvivalentně prostřednictvím proměnné `x` (ř. 4–6).

```
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     cislo = 683 * 1000 + a * 100 + b * 10 + c
8
```

```
9       if cislo % 7 == 0 and cislo % 8 == 0 and cislo % 9 == 0:
10          print(cislo)
```

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 hledá deset po sobě jdoucích přirozených čísel, u kterých každé z těchto čísel musí být dělitelné jedním jednociferným číslem. Úloha je ukázkou použití cyklu *while*. Hledání po sobě jdoucích právě deseti přirozených čísel se ukončí v situaci, kdy všech těchto deset přirozených čísel, je dělitelno každým z definovaných čísel.

**Úloha 2** ([3])

Existuje deset po sobě jdoucích přirozených čísel, která jsou po řadě dělitelná čísly 9, 7, 5, 3, 1, 1, 3, 5, 7, 9?

*Autorské řešení*: V úloze se hledá deset po sobě jdoucích přirozených čísel, které musí být postupně dělitelné čísly 9, 7, 5, 3, 1, 1, 3, 5, 7 a 9.

V rámci uvažování nad strategií řešení úlohy je vhodné si uvědomit, že stačí nalézt první z deseti přirozených čísel, zbylých devět po sobě jdoucích přirozených čísel je nalezeno postupným přičítáním $+1, +2, \ldots, +9$ od prvního přirozeného čísla. Pokud tedy číselnou hodnotu prvního nalezeného přirozeného čísla označíme $A$, pak druhé číslo lze vyjádřit jako $A+1$, třetí $A+2$. Poslední desáté číslo lze vyjádřit jako $A+9$. Ze zadání týkající se dělitelnosti pak plyne, že $9|A$, $7|(A+1)$, $5|(A+2)$, …, $9|(A+9)$.

Při hledání prvního z deseti po sobě jdoucích přirozených čísel je pak vhodné uplatnit cyklus s podmínkou na začátku – *while* cyklus [1)] s řídící proměnnou např. `splneno` (ř. 4), kterou je vhodné před prohledáváním stavového prostoru nastavit na hodnotu `False` (ř. 2). Zároveň je nutné zavést řídící proměnnou, např. `i`, která se s každým dalším pořadím prohledávání inkrementuje (ř. 5). Tuto proměnnou je nutné před prohledáváním stavového prostoru vynulovat (ř. 1).

Stavový prostor, který není shora nijak ohraničený, se bude systematicky prohledávat do situace, kdy se nalezne první číselná hodnota `i` taková, která odpovídá podmínkám zadání úlohy. Touto podmínkou je vyjádření bezezbytkové dělitelnosti `i` pomocí operátoru modulo % s porovnáním

[1)] Bylo by také možné pro účely vyčíslení řídící proměnné uplatnit cyklus s pevným počtem opakování – *for* cyklus, avšak s přihlédnutím k faktu, že není znám ani ciferný řád hledaného prvního přirozeného čísla, je tento typ cyklu – vzhledem k nutnosti stanovení horní číselné hodnoty prohledávání – nevhodný.

s hodnotou 0 (zapsáno jako `== 0`) v podmíněném příkazu `if` v konjunktivním tvaru pomocí operátorů `and` (ř. 7). V případě splnění takto formulované podmínky se vypíše číselná hodnota nalezeného `i` (ř. 8) a proměnná `splneno` se nastaví na hodnotu `True` (ř. 7), čímž se ukončí činnost *while* cyklu (ř. 7).

```
1 i = 0
2 while splneno == False:
3     i = i + 1
4
5     if i % 9 == 0 and (i + 1) % 7 == 0 and (i + 2) % 5 == 0
          and (i + 3) % 3 == 0 and (i + 4) % 1 == 0 and (i + 5)
          % 1 == 0 and (i + 6) % 3 == 0 and (i + 7) % 5 == 0
          and (i + 8) % 7 == 0 and (i + 9) % 9 == 0:
6         print(i)
7         splneno = True
```

Program vypíše jako řešení číslo 153. 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 vybírá jedno přirozené číslo z celkem osmi čísel, které není dělitelem určitého přirozeného čísla, resp. všech kladných celých čísel menších než 1 000.

**Úloha 3** ([4])

Pro kolik kladných čísel menších než 1000 platí, že mezi čísly 2, 3, 4, 5, 6, 7, 8 a 9 je právě jedno, které není jeho dělitelem?

*Autorské řešení*: V úloze se hledá počet všech kladných celých čísel menších než 1 000, kdy právě jedno z čísel od 2 do 9 není jeho dělitelem. Hledaná čísla mají být menší než 1 000 a zároveň dělitelná právě sedmi přirozenými čísly (byť některé z nich soudělné), proto budou posuzována všechna přirozená čísla od 10 do 999.

Je vhodné si uvědomit, že pro účely nalezení všech hledaných čísel není zapotřebí znát konkrétní číslo, které není dělitelem hledaných čísel. Celkový počet dělitelů 2 až 9 je 8 a pokud právě jedno z těchto čísel nemá být dělitelem hledaných čísel, pak takový počet dělitelů je 7. Tedy pokud počet dělitelů od 2 do 9 hledaného čísla je právě 7, pak hledané číslo splňuje podmínky zadání úlohy.

Pro účely hledání daného čísla uplatníme jeden cyklus s pevným počtem opakování s vhodným pojmenováním řídící proměnné, např. `cislo`, který nabývá číselných hodnot od 10 do 999 (ř. 3).

Pro určení právě sedmi dělitelů z možných právě osmi dělitelů 2 až 9 zavedeme pomocnou proměnnou reprezentující počet dělitelů čísla v pro-

měnné `cislo`, označme ji `pocet`, a kterou je nutné vynulovat (ř. 4). Následně lze přistoupit k postupnému posuzování dělitelnosti čísla v `cislo` s každým přirozeným číslem od 2 do 9: pomocí operátoru modulo % se posoudí dělitelnost beze zbytku čísla v proměnné `cislo` s každým přirozeným číslem od 2 do 9, v případě kladného vyhodnocení se inkrementuje proměnná `pocet` v rámci každého posuzovaného z čísel od 2 do 9 (ř. 6 až 28). V případě, že hodnota proměnné `pocet` bude rovna 7 (ř. 30), pak lze vypsat číselnou hodnotu proměnné `cislo` (ř. 31). Tato hodnota odpovídá situaci, kdy mezi osmi přirozenými čísly od 2 do 9 se nachází právě jedno, které není dělitelem posuzovaného čísla.

```
1  print("Nalezena cisla: ")
2
3  for cislo in range(10,1000):
4      pocet=0
5
6      if cislo%2 == 0:
7          pocet=pocet+1
8
9      if cislo%3 == 0:
10         pocet=pocet+1
11
12     if cislo%4 == 0:
13         pocet=pocet+1
14
15     if cislo%5 == 0:
16         pocet=pocet+1
17
18     if cislo%6 == 0:
19         pocet=pocet+1
20
21     if cislo%7 == 0:
22         pocet=pocet+1
23
24     if cislo%8 == 0:
25         pocet=pocet+1
26
27     if cislo%9 == 0:
28         pocet=pocet+1
29
30     if pocet == 7:
31         print(cislo)
```

Program vypíše jako řešení čtyři trojciferná čísla, a to 360, 504, 720 a 840. Jednoduchou početní kontrolou lze ověřit správnost těchto vypsaných výsledků. Zároveň je možné se přesvědčit o správnosti těchto výsledků nahlédnutím do řešení MO.

Protože posuzovaní dělitelé z předchozího řešení jsou všechna přirozená čísla od 2 do 9, lze je vyjádřit pomocí jednoho cyklu s pevným počtem opakování. Pokud použijeme označení řídící proměnné například `i`, pak

takový zápis nabude do podoby `for i in range(2, 10):` (ř. 6). Nyní mírně upravíme předchozí zdrojový kód a vytvoříme tak nový. Posouzení dělitelnosti čísla v proměnné `cislo` s čísly od 2 do 9, nyní v proměnné `i`, lze vyjádřit jako `if cislo % i == 0:` (ř. 7). V případě kladného posouzení se inkrementuje proměnná `pocet` (ř. 8). Číselnou hodnotu proměnné `cislo` (ř. 11) je možné vypsat v případě, že číselná hodnota v proměnné `pocet` je rovna číslu 7 (ř. 10), stejně jako v předchozím zdrojovém kódu. Oba zdrojové kódy pracují analogicky, proto jsou i jejich výstupy shodné. Proto lze předchozí zdrojový kód přepsat do podoby:

```
1  print("Nalezena cisla:")
2
3  for cislo in range(10, 1000):
4      pocet = 0
5
6      for i in range(2, 10):
7          if cislo % i == 0:
8              pocet = pocet + 1
9
10     if pocet == 7:
11         print(cislo)
```

Program také vypíše jako řešení čtyři trojciferná čísla, a to 360, 504, 720 a 840. Jedná se o shodné vypisované výsledky jako v případě výpisu předchozího programu.

V následující úloze 4 se pracuje s dělitelností přirozených čísel, resp. posuzováním sudosti a lichosti přirozených čísel.

**Úloha 4** ([5])

Trojmístné přirozené číslo budeme nazývat sudomilé, jestliže obsahuje dvě sudé číslice a číslici 1. Trojmístné přirozené číslo budeme nazývat lichomilé, jestliže obsahuje dvě liché číslice a číslici 2. Zjistěte, kolik je všech sudomilých a kolik lichomilých čísel.

*Autorské řešení*: V úloze se hledají dva typy trojciferných čísel: prvním typem je trojciferné číslo obsahující číslici 1 a právě dvě sudé číslice, druhým typem je trojciferné číslo obsahující číslici 2 a právě dvě liché číslice.

Zároveň je úloha specifická tím, že není zapotřebí formulovat hledaná trojciferná čísla; v úloze se pracuje pouze s dílčími ciframi těchto trojciferných čísel. Trojciferná čísla označíme $\overline{abc}$. Sudomilá čísla lze tedy označit jako $\overline{1bc}$, $\overline{a1c}$ a $\overline{ab1}$ s tím, že cifry $a$, $b$ a $c$ jsou sudé, lichomilá čísla pak $\overline{2bc}$, $\overline{a2c}$ a $\overline{ab2}$ s tím, že cifry $a$, $b$ a $c$ jsou liché.

Podmínku pro identifikaci sudomilých čísel lze vyjádřit jako

$$\underbrace{(a = 1 \wedge 2|b \wedge 2|c)}_{\overline{1bc}} \vee \underbrace{(2|a \wedge b = 1 \wedge 2|c)}_{\overline{a1c}} \vee \underbrace{(2|a \wedge 2|b \wedge c = 1)}_{\overline{ab1}}.$$

V programu bude tato podmínka vyjádřena pomocí operátoru modulo `%` s posouzením bezezbytkové dělitelnosti pomocí `== 0` (tj. sudosti cifer) v podmíněném příkazu `if` v disjunktním tvaru pomocí operátoru `or` (ř. 7). V případě kladného vyhodnocení této podmínky se bude inkrementovat proměnnou registrující počet sudomilých čísel (ř. 8), kterou je však nutné před prohledáváním stavového prostoru vynulovat (ř. 1).

Analogicky podobně lze přistoupit k určování počtu lichomilých čísel. Podmínku pro identifikaci lichomilých čísel lze vyjádřit jako

$$\underbrace{(a = 2 \wedge 2 \nmid b \wedge 2 \nmid c)}_{\overline{2bc}} \vee \underbrace{(2 \nmid a \wedge b = 2 \wedge 2 \nmid c)}_{\overline{a2c}} \vee \underbrace{(2 \nmid a \wedge 2 \nmid b \wedge c = 2)}_{\overline{ab2}}.$$

V programu bude tato podmínka vyjádřena pomocí operátoru modulo `%` s posouzením bezezbytkové nedělitelnosti pomocí `== 1` (tj. lichosti cifer) v podmíněném příkazu `if` v disjunktním tvaru pomocí operátoru `or` (ř. 10). V případě kladného vyhodnocení této podmínky je možné inkrementovat proměnnou registrující počet lichomilých čísel (ř. 11), kterou je však nutné před prohledáváním stavového prostoru vynulovat (ř. 2).

```
1  sudomile = 0
2  lichomile = 0
3
4  for a in range(1, 10):
5      for b in range(0, 10):
6          for c in range(0, 10):
7              if (a == 1 and b % 2 == 0 and c % 2 == 0) or (a %
                   2 == 0 and b == 1 and c % 2 == 0) or (a % 2
                    == 0 and b % 2 == 0 and c == 1):
8                  sudomile = sudomile + 1
9
10             if (a == 2 and b % 2 == 1 and c % 2 == 1) or (a %
                   2 == 1 and b == 2 and c % 2 == 1) or (a % 2
                    == 1 and b % 2 == 1 and c == 2):
11                 lichomile = lichomile + 1
12
13 print("Pocet sudomilych: ", sudomile)
14 print("Pocet lichomilych: ", lichomile)
```

Program vypíše jako řešení počet sudomilých čísel 65 a počet lichomilých čísel 75. Nahlédnutím do řešení MO je možné se přesvědčit o správnosti těchto nalezených výsledků.

Úlohu je také možné řešit s využitím pokročilých syntaktických konstrukcí. Místo uplatnění tří vzájemně vnořených *for* cyklů lze uplatnit metodu `product` se třemi proměnnými (ř. 6–9).

```
1  from itertools import product
2
3  sudomile = 0
4  lichomile = 0
5
6  for x in product(range(1, 10), repeat = 3):
7      a = x[0]
8      b = x[1]
9      c = x[2]
10
11     if a != 0:
12         ...
```

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

V následující úloze, úloze 5, se pracuje s racionálním lomeným výrazem, kdy se po dosazení kladného celého čísla do čitatele i jmenovatele má rozhodnout, zda hodnota výrazu je či není celým číslem. Úloha je ukázkou posuzování bezezbytkové dělitelnosti jak pomocí operátoru modulo, tak také pomocí pythonovské metody `.is_integer()`.

**Úloha 5** ([6])

Pro která celá čísla $x$ je podíl $\frac{x+11}{x+7}$ celým číslem? Najděte všechna řešení.

*Autorské řešení*: V úloze se hledají všechna celá čísla $x$, pro které je číslo $\frac{x+11}{x+7}$ celým číslem. Pokud číslo $\frac{x+11}{x+7}$ je celým číslem, pak číslo $x+7$ beze zbytku dělí číslo $x+11$, resp. číslo $x+11$ je celočíselným násobkem čísla $x+7$. Je zapotřebí si také uvědomit, že číslo $x+7$ nesmí být nulové, tzn. $x \neq -7$.

Jedním z možných řešení je definování podílu $\frac{x+11}{x+7}$ jako výrazu, kdy pro výraz v čitateli podílu $x+11$ zavedeme pomocnou proměnnou, např. `citatel` (ř. 1), stejně tak pro výraz ve jmenovateli podílu $x+7$ zavedeme pomocnou proměnnou, např. `jmenovatel` (ř. 3). Podíl výrazů je možné zformulovat jako podíl se zavedením pomocné proměnné, např. `zlomek` (ř. 6). Protože ale číselná hodnota jmenovatele nesmí být nulová, proto je nutné před zformulováním podílu posoudit nenulovost jmenovatele pomocí podmíněného příkazu `if` (ř. 5).

Pro účely posouzení celočíselnosti číselné hodnoty v proměnné `zlomek` je vhodné uplatnit metodu `.is_integer()`, která vrací hodnotu `True`

v případě, že v proměnné `zlomek` je celé číslo (tj. podíl je celočíselný), v opačném případě hodnotu `False`. Tuto metodu lze zavolat v podmíněném příkazu `if` jako `zlomek.is_integer()` (ř. 8). V případu kladného vyhodnocení této podmínky lze vypsat číselnou hodnotu proměnné `x` jako řešení úlohy (ř. 9).

```
1 for x in range(-1000, 1001):
2     citatel = x + 11
3     jmenovatel = x + 7
4
5     if jmenovatel != 0:
6         zlomek = citatel / jmenovatel
7
8         if zlomek.is_integer():
9             print(x)
```

Program vypíše jako řešení 6 záporných čísel: $-11, -9, -8, -6, -5, -3$. Postupnou kontrolou šesti vypsaných výsledků lze ověřit správnost těchto vypsaných výsledků. Zároveň lze správnost těchto výsledků ověřit nahlédnutím do výsledků MO [6].

```
1 import math
2
3 for x in range(-1000, 10001):
4     citatel = x + 11
5     jmenovatel = x + 7
6
7     if jmenovatel != 0:
8         if citatel % jmenovatel == 0:
9             print(x)
```

Program také vypíše jako řešení 6 záporných čísel: $-11, -9, -8, -6, -5, -3$. Jednoduchou početní kontrolou lze ověřit správnost těchto vypsaných výsledků. Zároveň je možné se přesvědčit o správnosti těchto výsledků nahlédnutím do řešení MO.

V následující úloze, úloze 6, se doplňují do čtyř, resp. šesti čtyřciferných přirozených čísel právě dvě cifry tak, aby výsledek dvou příkladů odpovídal určité podmínce. Úloha je tak ukázkou řešení takových matematických úloh, ve kterých se doplňují cifry a mají vzniklá čísla splňovat dané podmínky plynoucí z pravidel součtů a rozdílů čtyřciferných čísel.

**Úloha 6** ([7])
Doplňte místo hvězdiček číslice tak, aby součet výsledků následujících dvou příkladů byl 5 842:

$$\begin{array}{r} \star\ 2\ \star\ 7 \\ 3\ \star\ 4\ \star \\ \hline 4\ \star\ 0\ 0 \end{array} \qquad \begin{array}{r} 2\ \star\ 9\ \star \\ -\ \star\ 2\ \star\ 4 \\ \hline \star\ 5\ 4\ \star \end{array}$$

*Autorské řešení*: V úloze se hledá několik jednociferných číslic, které se doplňují za hvězdičky jako chybějící cifry do čtyř čtyřciferných čísel tak, aby oba příklady byly vyřešeny správně. Je vhodné si uvědomit, že hvězdičky ve výsledcích obou příkladů není nutné vyjadřovat a vyčíslovat pomocí proměnných, jejich hodnoty vzejdou z příslušného součtu a rozdílu čtyřciferných čísel v obou příkladech.

Hvězdičky – pouze u čísel nad podtržítkem – nahradíme proměnnými $a$ až $h$ následovně:

$$\begin{array}{r} a\ 2\ b\ 7 \\ 3\ c\ 4\ d \\ \hline 4\ \star\ 0\ 0 \end{array} \qquad \begin{array}{r} 2\ e\ 9\ f \\ -\ g\ 2\ h\ 4 \\ \hline \star\ 5\ 4\ \star \end{array}$$

Na cifry nejsou ze zadání explicitně kladena žádná číselná omezení. Proto cifry $a$ až $h$ mohou nabývat celočíselných hodnot od 0 do 9. Čísla v prvním příkladu proto budou $\overline{a2b7}$ a $\overline{3c4d}$, čísla ve druhém příkladu pak budou $\overline{2e9f}$ a $\overline{g1h4}$. Výsledky obou příkladů mohou být zapsány ve tvaru $\overline{4 * 00}$ a $\overline{*54*}$.

Pro účely vyčíslení proměnných $a$ až $h$ uplatníme osm vzájemně vnořených cyklů s pevným počtem opakování s vyčíslováním od 0 do 9 s vhodným pojmenováním řídících proměnných, např. `a` až `g` (ř. 1–8).

Pro účely zvýšení přehlednosti zdrojového kódu je vhodné dekadicky vyjádřit a pojmenovat pomocí proměnných obě čísla nad podtržítkem prvního příkladu (ř. 9 a 10), stejně tak druhého příkladu (ř. 22 a 23). Vyčíslení výsledků příkladu je vhodné vyjádřit jako součet obou čísel nad podtržítkem (ř. 11), resp. rozdíl prvního a druhého čísla čísel nad podtržítkem (ř. 24). Tím jsme dostali oba výsledky příkladů jako čtyřciferná čísla.

Protože však potřebujeme zjistit, zda ve výsledku prvního příkladu je na místě tisíců cifra 4, desítek 0 a jednotek 0, stejně tak ve výsledku druhého příkladu je na místě stovek cifra 5 a na místě desítek cifra 4, pak je nutné provést číselný rozklad obou výsledků příkladů.

Uvedený postup vysvětlíme na následujícím modelovém příkladu: předpokládejme, že máme čtyřciferné přirozené číslo ve tvaru $\overline{tsdj}$:[2)] trojciferné

[2)] Označení proměnných znamenají: t – tisíce, s – stovky, d – desítky, j – jednotky.

číslo $\overline{sdj}$ získáme jako $\overline{tsdj}$ mod 1 000, dvojciferné číslo $\overline{dj}$ získáme jako $\overline{sdj}$ mod 100 a jednotky $j$ získáme jako $\overline{dj}$ mod 10. V rámci tohoto rozkladu je třeba si postupně ukládat jednotlivé cifry do příslušných proměnných $t$, $s$, $d$ a $j$ pomocí celočíselného dělení division.

Pro realizaci číselného rozkladu pro výsledek prvního příkladu je vhodné zavést pomocnou proměnnou (ř. 13), např. `soucet`. Tuto pomocnou proměnnou budeme číselně rozkládat pomocí operace modulo `%` (ř. 15, 17, 19), mezitím však pomocí operace celočíselného dělení `//` postupně zaznamenávat jednotlivé nejvyšší cifry daných čísel, které se zaznamenávají do proměnných `t`, `s`, `d`, `j` (ř. 14, 16, 18, 20). Analogicky stejně bude proveden číselný rozklad výsledku druhého příkladu (ř. 26–33). Tím jsou v proměnných `t1`, `s1`, `d1`, `j1` uvedeny všechny cifry výsledku prvního příkladu, v proměnných `t2`, `s2`, `d2`, `j2` pak všechny cifry výsledku druhého příkladu.

V této situaci je možné posoudit, zda příslušné proměnné obsahují příslušné cifry ze zadání úlohy:

$$t_1 = 4 \wedge t_1 = 4 \wedge d_1 = 0 \wedge j_1 = 0 \wedge s_2 = 5 \wedge d_2 = 4.$$

To lze zrealizovat pomocí podmíněného příkazu `if` v konjunktivním tvaru pomocí operátoru `and` (ř. 35). V případě kladného vyhodnocení této podmínky lze vypsat číselné hodnoty všech šesti čísel v podobě zadaných dvou příkladů (ř. 36–46).

```
1  for a in range(0, 10):
2   for b in range(0, 10):
3     for c in range(0, 10):
4        for d in range(0, 10):
5          for e in range(0, 10):
6             for f in range(0, 10):
7                for g in range(0, 10):
8                   for h in range(0, 10):
9                      cislo1 = a * 1000 + 2 * 100 + b * 10 + 7
10                     cislo2 = 3 * 1000 + c * 100 + 4 * 10 + d
11                     vysledek1 = cislo1 + cislo2
12
13                     soucet = vysledek1
14                     t1 = soucet // 1000
15                     soucet = soucet%1000
16                     s1 = soucet // 100
17                     soucet = soucet % 100
18                     d1 = soucet // 10
19                     soucet = soucet % 10
20                     j1 = soucet
21
22                     cislo3 = 2 * 1000 + e * 100 + 9 * 1 + f
23                     cislo4 = g * 1000 + 2 * 100 + h * 10 + 4
24                     vysledek2 = cislo3 - cislo4
25
```

```
                    rozdil = vysledek2
                    t2 = rozdil // 1000
                    rozdil = rozdil % 1000
                    s2 = rozdil // 100
                    rozdil = rozdil % 100
                    d2 = rozdil // 10
                    rozdil = rozdil % 10
                    j2 = rozdil

                    if t1 == 4 and d1 == 0 and j1 == 0 and s2 == 5
                        and vysledek1 + vysledek2 == 5842:
                        print(cislo1)
                        print(cislo2)
                        print("------")
                        print(vysledek1)
                        print()

                        print(cislo3)
                        print(-cislo4)
                        print("------")
                        print(vysledek2)
                        print()
```

Program vypíše jedno řešení, které lze zapsat ve tvaru:

$$\begin{array}{r} 1\,2\,5\,7 \\ 3\,0\,4\,3 \\ \hline 4\,3\,0\,0 \end{array} \qquad \begin{array}{r} 2\,7\,9\,6 \\ -\,1\,2\,5\,4 \\ \hline 1\,5\,4\,2 \end{array}$$

Jednoduchou početní kontrolou lze ověřit správnost tohoto vypsaného výsledku. Zároveň je možné se přesvědčit o správnosti tohoto výsledku nahlédnutím do řešení MO.

Úlohu se jaké možné také řešit s využitím pokročilých syntaktických konstrukcí. Místo uplatnění osmi vzájemně vnořených *for* cyklů lze uplatnit metodu `product` s osmi proměnnými (ř. 3–11).

```
1  from itertools import product
2
3  for x in product(range(1, 10), repeat = 8):
4      a = x[0]
5      b = x[1]
6      c = x[2]
7      d = x[3]
8      e = x[4]
9      f = x[5]
10     g = x[6]
11     h = x[7]
12
13     cislo1 = a * 1000 + 2 * 100 + b * 10 + 7
14     ...
```

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

*Poznámka.* Úlohu je možné také řešit tak, že všechny hvězdičky v obou příkladech budou nahrazeny řídícími proměnnými $a$ až $k$ se systematickým vyčíslením každé z nich. Pak oba příklady lze přepsat do tvaru:

$$\begin{array}{cccc} a & 2 & b & 7 \\ 3 & c & 4 & d \\ \hline 4 & e & 0 & 0 \end{array} \qquad\qquad \begin{array}{cccc} & 2 & f & 9 & g \\ - & h & 2 & i & 4 \\ \hline & j & 5 & 4 & k \end{array}$$

Celkem tedy bylo použito 11 proměnných. Je třeba zdůraznit, že celková doba hledání řešení na počítači je již neúměrně vysoká, proto tento způsob řešení není vhodný.

**Shrnutí**

V předloženém příspěvku jsme prezentovali celkem 6 vybraných matematických úloh MO, které jsme řešili pomocí programování v jazyce Python. U některých úloh – tam kde to bylo vhodné –, bylo představeno také řešení s využitím pokročilých syntaktických konstrukcí.

## Literatura


[1] *Perk, L.*: K možnosti využití programování při žákovském řešení vybraných matematických úloh MO. Matematika–Fyzika–Informatika, roč. 34 (2025), č. 2, s. 121–133.

[2] MO ČR. 52. ročník, domácí kolo, kategorie Z9, úloha 1.

[3] MO ČR. 73. ročník, domácí kolo, kategorie C, úloha 1.

[4] MO ČR. 68. ročník, II. kolo, kategorie Z8, úloha 2.

[5] MO ČR. 61. ročník, II. kolo, kategorie Z7, úloha 3.

[6] MO ČR. 69. ročník, I. kolo, kategorie Z9, úloha 3.

[7] MO ČR. 59. ročník, I. kolo, kategorie Z6, úloha 6.