1. Úvod

video prezentácie

1. úvod do predmetu
2. jazyk python
3. premenné a priradenie
4. program a vstup/výstup

Jazyk Python

Python je moderný programovací jazyk, ktorého popularita stále rastie.

- jeho autorom je Guido van Rossum (vymyslel ho v roku 1989)
- používajú ho napríklad v Google, YouTube, Dropbox, Mozilla, Quora, Facebook, Rasperry Pi, …
- na mnohých špičkových univerzitách sa učí ako úvodný jazyk, napríklad MIT, Carnegie Mellon, Berkeley, Cornell, Caltech, Illinois, …
- beží na rôznych platformách, napríklad Windows, Linux, Mac. Je to *freeware* a tiež *open source*.

Na rozdiel od mnohých iných jazykov, ktoré sú kompilačné (napríklad Pascal, C/C++, C#) je Python interpreter. To znamená, že

- interpreter nevytvára spustiteľný kód (napríklad .exe súbor vo Windows)
- na spustenie programu musí byť v počítači nainštalovaný Python
- interpreter umožňuje aj interaktívnu prácu s prostredím

Hlavné vlastnosti jazyka Python:

- veľmi jednoduchá a dobre čitateľná syntax a keďže Python je aj vysoko interaktívny, je veľmi vhodný aj pre vyučovanie programovania
- na rozdiel od staticky typovaných jazykov, pri ktorých je treba dopredu deklarovať typy všetkých dát, je Python dynamicky typovaný, čo znamená, že neexistujú žiadne deklarácie
- Python obsahuje pokročilé črty moderných programovacích jazykov, napríklad podpora práce s dátovými štruktúrami, objektovo-orientovaná tvorba softvéru, …
- je to univerzálny programovací jazyk, ktorý poskytuje prostriedky na tvorbu moderných aplikácií, takých ako analýza dát, spracovanie médií, sieťové aplikácie a pod.
- Python má obrovskú komunitu programátorov a expertov, ktorí sú ochotní svojimi radami pomôcť aj začiatočníkom

Naštartujeme Python

Ako ho získať

- zo stránky https://www.python.org/ stiahnete najnovšiu verziu Pythonu - momentálne je to verzia **3.10.0**
- spustite inštalačný program (podľa typu operačného systému napríklad `python-3.10.0-amd64.exe`)
- POZOR! Nesťahujte verziu začínajúcu 2 (napríklad 2.7.16) - tá nie je kompatibilná s verziou 3.x
- pri inštalácii odporúčame nastaviť:
  - for all users
  - add Python 3.10 to PATH
  - inštalovať pip, tkinter aj IDLE

Alternatívne vývojové prostredia

- Inštalácia najnovšej verzie Pythonu obsahuje aj vývojové prostredie **IDLE**. Toto prostredie je pre úplného začiatočníka ideálne a budeme ho používať aj v tomto kurze.
- Pre skúsenejších odporúčame vybrať si jedno z týchto prostredí (mali by ste už mať nainštalovaný Python)
  - PyCharm Edu - Easy and Professional Tool to Learn & Teach Programming with Python
  - Wing Personal - A free Python IDE for students and hobbyists

Spustíme IDLE

IDLE (Python GUI) je vývojové prostredie (Integrated Development Learning Environment), vidíme informáciu o verzii Pythonu a tiež na začiatku riadu tri znaky `>>>` (tzv. výzva, t.j. **prompt**). Za túto výzvu budeme písať príkazy pre Python.

```
   Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit
(AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Ako to funguje

- Python je interpreter a pracuje v niekoľkých možných režimoch
- teraz sme ho spustili v **interaktívnom režime** (niekedy hovoríme aj príkazový režim): očakáva zadávanie textových príkazov (do riadka za znaky `>>>`), každý zadaný príkaz vyhodnotí a vypíše prípadnú reakciu (alebo **chybovú správu**, ak sme zadali niečo nesprávne)
- po skončení vyhodnocovania riadka sa do ďalšieho riadka znovu vypíšu znaky `>>>` a očakáva sa opätovné zadávanie ďalšieho príkazu
- takémuto interaktívnemu oknu hovoríme **shell**
- niekedy sa môžete dočítať aj o tzv. REP cykle interpretera, znamená to **Read**, **Evaluate**, **Print**, teda prečítaj, potom tento zápis vyhodnoť a na koniec vypíš výsledok, toto celé stále opakuj

Môžeme teda zadávať, napríklad nejaké matematické výrazy:

```
>>> 12345
    12345
>>> 123 + 456
    579
>>> 1 * 2 * 3 * 4 * 5 * 6
    720
```

```
>>>
```

V tomto príklade sme pracovali s celými číslami a niektorými celočíselnými operáciami. Python poskytuje niekoľko rôznych **typov** údajov; na začiatok sa zoznámime s tromi základnými typmi: celými číslami, desatinnými číslami a znakovými reťazcami.

celé čísla

- majú rovnaký význam, ako ich poznáme z matematiky: zapisujú sa v desiatkovej sústave a môžu začínať znamienkom plus alebo mínus
- ich veľkosť (počet cifier) je obmedzená len kapacitou pracovnej pamäte Pythonu (hoci aj niekoľko miliónov cifier)

Pracovať môžeme aj s **desatinnými číslami** (tzv. **floating point**), napríklad:

```
>>> 22 / 7
    3.142857142857143
>>> .1 + .2 + .3 + .4
    1.0
>>> 9999999999 * 99999999999
    999999999890000000001
>>> 9999999999 * 99999999999.
    9.9999999989e+20
>>>
```

desatinné čísla

- obsahujú desatinnú bodku alebo exponenciálnu časť (napríklad `1e+15`)
- môžu vzniknúť aj ako výsledok niektorých operácií (napríklad delením dvoch celých čísel)
- majú obmedzenú presnosť (približne 16-17 platných cifier)

Všimnite si, že 3. výraz `9999999999*99999999999` násobí dve celé čísla a aj výsledkom je celé číslo. Hneď ďalší výraz `9999999999*99999999999.` obsahuje jedno desatinné číslo (číslo s bodkou) a teda aj výsledok je desatinné číslo.

V ďalšom príklade by sme chceli pracovať s nejakými textami. Ak ale zadáme:

```
>>> ahoj
    Traceback (most recent call last):
      File "<pyshell#10>", line 1, in <module>
        ahoj
    NameError: name 'ahoj' is not defined
>>>
```

dostaneme chybovú správu `NameError: name 'ahoj' is not defined`, t.j. Python nepozná takéto meno. Takýmto spôsobom sa texty ako postupnosti nejakých znakov nezadávajú: na to potrebujeme špeciálny zápis: texty zadávame uzavreté medzi apostrofy, resp. úvodzovky. Takýmto textovým zápisom hovoríme **znakové reťazce**. Keď ich zapíšeme do príkazového riadka, Python ich vyhodnotí (v tomto prípade s nimi neurobí nič) a vypíše ich hodnotu. Napríklad:

```
>>> 'ahoj'
    'ahoj'
>>> "hello folks"
```

```
    'hello folks'
>>> 'úvodzovky "v" reťazci'
    'úvodzovky "v" reťazci'
>>> "a tiež apostrofy 'v' reťazci"
    "a tiež apostrofy 'v' reťazci"
>>>
```

znakové reťazce

- ich dĺžka (počet znakov) je obmedzená len kapacitou pracovnej pamäte Pythonu
- uzatvárame medzi apostrofy `'text'` alebo úvodzovky `"text"`
    - oba zápisy sú rovnocenné - reťazec musí končiť tým istým znakom ako začal (apostrof alebo úvodzovka)
    - takto zadaný reťazec nesmie presiahnuť jeden riadok
- môže obsahovať aj písmená s diakritikou
- prázdny reťazec má dĺžku 0 a zapisujeme ho ako `''`

Zatiaľ sme nepísali príkazy, ale len zadávali výrazy (číselné a znakové) - ich hodnoty sa vypísali v ďalšom riadku. Toto funguje len v tomto príkazovom režime.

Základné typy údajov

Videli sme, že hodnoty (konštanty alebo výrazy) môžu byť rôznych typov. V Pythone má každý typ svoje meno:

- `int` pre **celé čísla**, napríklad `0`, `1`, `15`, `-123456789`, …
- `float` pre **desatinné čísla**, napríklad `0.0`, `3.14159`, `2.0000000001`, `33e50`, …
- `str` pre **znakové reťazce**, napríklad `'a'`, `"abc"`, `''`, `"I'm happy"`

Typ ľubovoľnej hodnoty vieme v Pythone zistiť pomocou štandardnej funkcie `type()`, napríklad:

```
>>> type(123)
    <class 'int'>
>>> type(22 / 7)
    <class 'float'>
>>> type(':-)')
    <class 'str'>
>>>
```

Rôzne typy hodnôt majú zadefinované rôzne operácie.

Celočíselné operácie

- oba operandy musia byť celočíselného typu

`+`	sčitovanie	`1 + 2` má hodnotu `3`
`-`	odčitovanie	`2 - 5` má hodnotu `-3`
`*`	násobenie	`3 * 37` má hodnotu `111`
`//`	celočíselné delenie,	`22 // 7` má hodnotu `3`

`%`	zvyšok po delení	`22 % 7` má hodnotu `1`
`**`	umocňovanie	`2 ** 8` má hodnotu `256`

- zrejme nemôžeme deliť 0

Operácie s desatinnými číslami

- aspoň jeden z operandov musí byť desatinného typu (okrem delenia `/`)

`+`	sčitovanie	`1 + 0.2` má hodnotu `1.2`
`-`	odčitovanie	`6 - 2.86` má hodnotu `3.14`
`*`	násobenie	`1.5 * 2.5` má hodnotu `3.75`
`/`	delenie	`23 / 3` má hodnotu `7.666666666666667`
`//`	delenie zaokrúhlené nadol	`23.0 // 3` má hodnotu `7.0`
`%`	zvyšok po delení	`23.0 % 3` má hodnotu `2.0`
`**`	umocňovanie	`3 ** 3.` má hodnotu `27.0`

- zrejme nemôžeme deliť 0

Operácie so znakovými reťazcami

`+`	zreťazenie (spojenie dvoch reťazcov)	`'a' + 'b'` má hodnotu `'ab'`
`*`	viacnásobné zreťazenie reťazca	`3 * 'x'` má hodnotu `'xxx'`

Zreťazenie dvoch reťazcov je bežné aj v iných programovacích jazykoch. Viacnásobné zreťazenie je dosť výnimočné. Na príklade vidíme, ako to funguje:

```
>>> 'ahoj' + 'Python'
    'ahojPython'
>>> 'ahoj' + ' ' + 'Python'
    'ahoj Python'
>>> '#' + '#' + '#' + '#'
    '####'
>>> 4 * '#'
    '####'
>>> '#' * 4
    '####'
>>> 10 * ' :-) '
    ' :-)  :-)  :-)  :-)  :-)  :-)  :-)  :-)  :-)  :-) '
>>>
```

Premenné a priradenie

Doteraz sme pracovali len s hodnotami, t.j. s číslami a znakovými reťazcami. Teraz ukážeme, ako si zapamätať nejakú hodnotu tak, aby sme ju mohli použiť aj neskôr. Na toto slúžia

tzv. **premenné**. Na rozdiel od premenných napríklad v Pascale alebo C, kde premenná je pomenovanie nejakého konkrétneho vyhradeného pamäťového miesta (nejakej veľkosti a typu), v Pythone hovoríme, že premenná je pomenovaná nejaká existujúca hodnota v pamäti.

Premenná **vzniká** nie zadeklarovaním a spustením programu (ako v Pascale a v C), ale vykonaním priraďovacieho príkazu (nejakej existujúcej hodnote sa priradí meno).

Meno premennej:

- môže obsahovať písmená, číslice a znak podčiarkovník
- pozor na to, že v Pythone sa rozlišujú malé a veľké písmená
- musí sa líšiť od pythonovských príkazov, tzv. **rezervovaných slov** (napríklad `for`, `if`, `def`, `return`, …)

Priraďovací príkaz

Zapisujeme:

```
premenna = hodnota
```

Tento zápis znamená, že do *premennej* (na ľavej strane príkazu pred znakom `=`) sa má priradiť zadaná *hodnota* (výraz na pravej strane za znakom `=`), t.j. zadaná hodnota dostáva meno a pomocou tohto mena s ňou budeme vedieť pracovať.

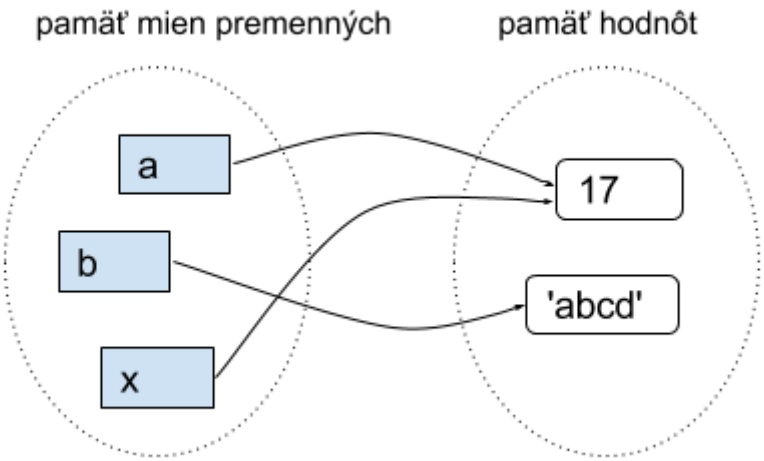
Premenná sa vytvorí priraďovacím príkazom (ak ešte doteraz neexistovala):

- v skutočnosti sa v Pythone do premennej priradí **referencia** (odkaz, adresa) na danú hodnotu (a nie samotná hodnota)
- ďalšie priradenie do tej istej premennej zmení túto referenciu
- na tú istú hodnotu sa môže odkazovať aj viac premenných
- meno môže referencovať (mať priradenú) maximálne jednu hodnotu (hoci samotná hodnota môže byť dosť komplexná)

Python si v svojej pamäti udržuje všetky premenné (v tzv. pamäti mien premenných) a všetky momentálne vytvorené hodnoty (v tzv. pamäti hodnôt). Po vykonaní týchto troch priraďovacích príkazov:

```
>>> a = 17
>>> b = 'abcd'
>>> x = a
```

To v pamäti Pythonu vyzerá nejako takto:



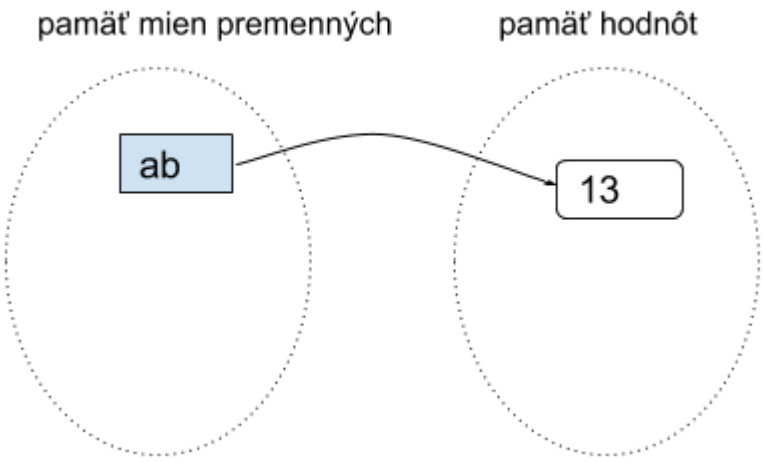
Vidíme, že

- ak priradíme do premennej nejakej hodnotu inej premennej (napríklad `x = a`) neznamená to referenciu na meno ale na jej hodnotu
- najprv sa zistí hodnota na pravej strane príkazu a až potom sa spraví referencovanie (priradenie) do premennej na ľavej strane

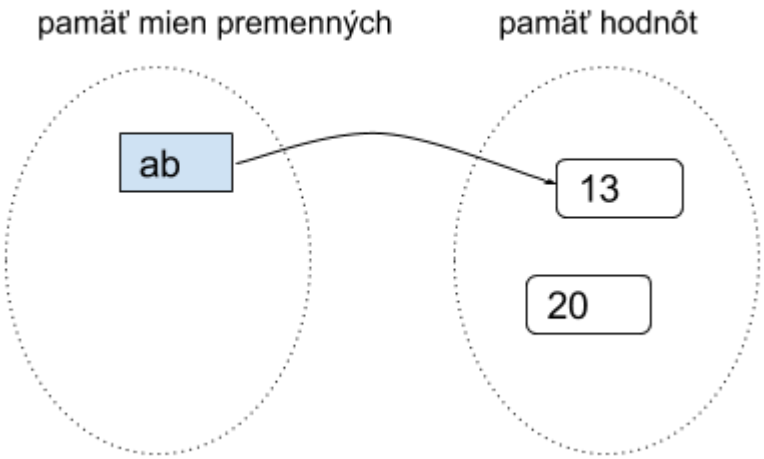
V ďalšom príklade vidíme, ako to funguje, keď vo výraze na pravej strane (kde je priraďovaná hodnota) sa nachádza tá istá premenná, ako na ľavej strane (teda kam priraďujeme):

```
>>> ab = 13
>>> ab = ab + 7
```

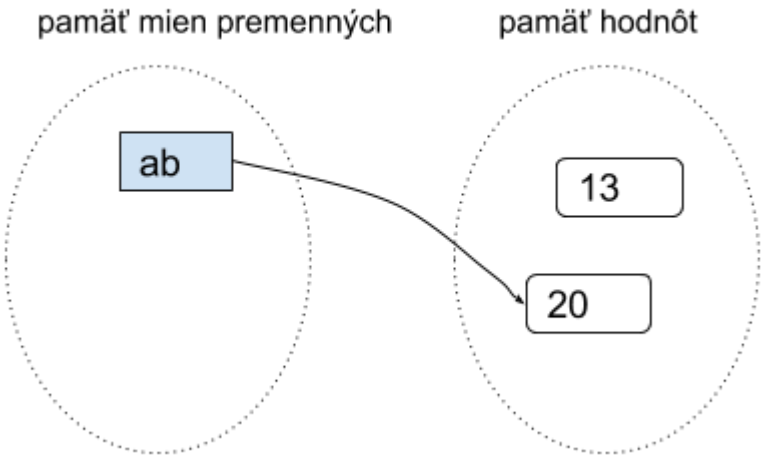
1. `ab` má najprv hodnotu `13`



2. potom sa vypočíta nová hodnota `20` (ako súčet `ab + 7`)



3. do premennej `ab` sa priradí nová hodnota



Desatinné čísla sa v počítači ukladajú s presnosťou len na istý počet desatinných miest a k tomu sa ešte pamätá aj tzv. exponenciálna časť, t.j. akou mocninou desiatky sa to celé vynásobí. Pozrite tento príklad:

```
>>> 10000.0
    10000.0
>>> 1000e0
    1000.0
>>> 1e4
    10000.0
>>> 0.0001e8
    10000.0
>>> 1000000e-3
    1000.0
```

Všetky zápisy reprezentujú tú istú hodnotu (desatinné číslo 1000), ale okrem prvého všetky obsahujú exponenciálnu časť, t.j. číslo uvedené za písmenom `e`. Ale táto exponenciálna časť má svoje limity a nemôže byť ľubovoľne veľká. Ilustruje to nasledovný príklad:

```
>>> y = 4.3 * 10 ** 100          # velmi velke desatinne cislo
>>> y
    4.3e+100
>>> y ** 2
    1.849e+201
>>> y ** 3
    7.9507e+301
>>> y ** 4
    Traceback (most recent call last):
      File "<pyshell#7>", line 1, in <module>
        y ** 4
    OverflowError: (34, 'Result too large')
```

Všimnite si, že exponenciálna časť môže byť väčšia ako 300 ale nesmie presiahnuť 400. `OverflowError: ..` označuje chybovú správu pretečenia aritmetiky s desatinnými číslami.

Programovací režim

V IDLE vytvoríme nové okno (menu **File** a položka **New Window**), resp. stlačíme <Ctrl-N>. Otvorí sa nové textové okno, ale už bez promptu `>>>`. Do tohto okna nebudeme zadávať výrazy, ale príkazy. Použime priraďovacie príkazy, s ktorými sme pracovali vyššie:

```
a = 17
b = 'abcd'
x = a
```

Takto vytvorený program (hovoríme mu aj **skript**) treba uložiť (najčastejšie s príponou **.py**) a spustiť (**Run**, alebo klávesom **<F5>**). V okne shell sa objaví:

```
========== RESTART ==========
>>>
```

Po spustení sa v pôvodnom okne **shell** najprv celý Python reštartuje (zabudne, všetko, čo sme ho doteraz naučili, t.j. vyčistí pamäť premenných) a takto vyčistený vykoná všetky príkazy programu. V našom prípade sa vykonali len tri priradenia. Môžeme otestovať:

```
>>> a
    17
>>> b
    'abcd'
```

Ak by sme chceli, aby priamo náš program vypísal tieto hodnoty, takýto zápis tomu nepomôže:

```
a = 17
```

```
b = 'abcd'
x = a

a
b
```

Po spustení tohto programu sa opäť nič nevypíše. Zapamätajte si:

- po zadaní výrazov v interaktívnom režime (za promptom `>>>`) sa tieto vyhodnotili a hneď aj vypísali
- po zadaní výrazov v programovom režime sa tieto tiež vyhodnotia, ich **hodnota ale sa nevypíše ale sa ignoruje**
- ak chceme, aby sa táto hodnota neignorovala, musíme ju spracovať napríklad priraďovacím príkazom alebo príkazom `print()` na výpis hodnôt (`print()` je v skutočnosti funkcia, uvidíme to neskôr)

Opravíme náš program:

```
a = 17
b = 'abcd'
x = a

print(a)
print(b)
print('výpočet je', 3 + x * 2)
```

Príkaz `print()` vypíše hodnoty uvedené v zátvorkách. Teda po spustení nášho programu dostávame:

```
========== RESTART ==========
17
abcd
výpočet je 37
>>>
```

Programy môžeme spúšťať nielen z vývojového prostredia IDLE, ale aj dvojkliknutím na súbor v operačnom systéme.

spustenie skriptu priamo zo systému

- ak je Python v operačnom systéme nainštalovaný korektne, dvojkliknutie na príponu súboru `.py` pre neho znamená spustenie programu:
  - otvorí sa nové konzolové okno, vykonajú sa príkazy a okno sa hneď aj zatvorí
- aby sa takéto okno nezatvorilo hneď, ale počkalo, kým nestlačíme napríklad kláves **ENTER**, pridáme do programu nový riadok s príkazom `input()`

Do nášho skriptu dopíšeme nový riadok:

```
a = 17
b = 'abcd'
x = a

print(a)
print(b)
print('výpočet je', 3 + x * 2)
```

```
input()
```

Po spustení takéhoto programu sa najprv vypíšu zadané texty a okno sa nezavrie, kým nestlačíme **ENTER**. Príkaz `input()` môže obsahovať aj nejaký text, ktorý sa potom vypíše pred čakaním na **ENTER**, napríklad takýto nový program:

```
# môj prvý program
print('programujem v Pythone')
print()

input('stlač ENTER')
```

Po spustení v operačnom systéme (nie v IDLE) sa v konzolovom okne objaví:

```
programujem v Pythone

stlač ENTER
```

Až po stlačení klávesu **ENTER** sa okno zatvorí.

Všimnite si, že do prvého riadka programu sme zapísali tzv. **komentár**, t.j. text za znakom `#`, ktorý sa Pythonom ignoruje.

Zhrňme oba tieto nové príkazy:

`print()`

- uvidíme neskôr, že je to volanie špeciálnej funkcie
- táto funkcia vypisuje hodnoty výrazov, ktoré sú uvedené medzi zátvorkami
- hodnoty sú pri výpise oddelené medzerami
- `print()` bez parametrov spôsobí len zariadkovanie výpisu, teda vloží na momentálne miesto prázdny riadok

`input()`

- je tiež funkcia, ktorá najprv vypíše zadaný znakový reťazec (ak je zadaný) a potom čaká na vstupný reťazec ukončený ENTER
- funkcia vráti tento nami zadaný znakový reťazec

Funkciu `input()` môžeme otestovať aj v priamom režime:

```
>>> input()
    pisem nejaky text
    'pisem nejaky text'
>>>
```

Príkaz `input()` tu čaká na stlačenie **ENTER**. Kým ho nestlačíme, ale píšeme nejaký text, tento sa postupne zapamätáva. Stlačenie **ENTER** (za napísaný text `pisem nejaky text`) spôsobí, že sa tento zapamätaný text vráti ako výsledok, teda v príkazovom režime sa vypísala hodnota zadaného znakového reťazca. Druhý príklad najprv vypíše zadaný reťazec `'? '` a očakáva písanie textu s ukončením pomocou klávesu ENTER:

```
>>> input('? ')
    ? matfyz
    'matfyz'
```

```
>>>
```

Môžeme to otestovať aj takýmto programom:

```
meno = input('ako sa volas? ')
print('ahoj', meno)
```

V tomto príklade sa využíva funkcia `input()`, ktorá najprv zastaví bežiaci výpočet, vypýta si od používateľa, aby zadal nejaký text a tento uloží do premennej `meno`. Na koniec toto zadané meno vypíše. Program spustíme klávesom **F5**:

```
ako sa volas? Jozef
ahoj Jozef
>>>
```

Týmto program skončil a môžeme pokračovať aj v skúmaní premenných, napríklad v programovom režime zistíme hodnotu premennej `meno`:

```
>>> meno
    'Jozef'
>>>
```

V našich budúcich programoch bude bežné, že na začiatku sa vypýtajú hodnoty nejakých premenných a ďalej program pracuje s nimi.

Ďalší program ukazuje, ako to vyzerá, keď chceme načítať nejaké číslo:

```
# prevod euro na ceske koruny

suma = input('zadaj eura: ')
koruny = suma * 26
print(suma, 'euro je', koruny, 'korun')
```

Po spustení programu dostávame:

```
=========== RESTART ===========
zadaj eura: 100
100 euro je 1001001001001001001001001001001001001001001001001001001001001001001001001001
00 korun
>>>
```

Takýto výsledok je zrejme nesprávny: očakávali sme 2600 korún. Problémom tu bolo to, že funkcia `input()` zadanú hodnotu vráti nie ako číslo, ale ako znakový reťazec, teda `'100'`. Na tomto mieste potrebujeme takýto reťazec **prekonvertovať** na desatinné číslo. Využijeme jednu z troch konvertovacích (pretypovacích) funkcií:

pretypovanie hodnôt

Mená typov `int`, `float` a `str` zároveň súžia ako mená pretypovacích funkcií, ktoré dokážu z jedného typu vyrobiť hodnotu iného typu:

- `int(hodnota)` z danej hodnoty vyrobí celé číslo, napríklad:
  - `int(3.14)` => `3`

    - `int('37')` => `37`
- `float(hodnota)` z danej hodnoty vyrobí desatinné číslo, napríklad:
    - `float(333)` => `333.0`
    - `float('3.14')` => `3.14`
- `str(hodnota)` z danej hodnoty vyrobí znakový reťazec, napríklad:
    - `str(356)` => `'356'`
    - `str(3.14)` => `'3.14'`

Zrejme pretypovanie reťazca na číslo bude fungovať len vtedy, keď je to správne zadaný reťazec, inak funkcia vyhlási chybu.

Program by mal vyzerať správne takto:

```
# prevod euro na ceske koruny

retazec = input('zadaj eura: ')
suma = float(retazec)               # pretypovanie
koruny = suma * 26
print(suma, 'euro je', koruny, 'korun')
```

Prečítaný reťazec sa najprv prekonvertuje na desatinné číslo a až potom sa s týmto číslom pracuje ako so zadanou cenou jedného výrobku v eurách. Po spustení dostávame:

```
zadaj eura: 100
100.0 euro je 2600.0 korun
```

Pre čitateľnosť programu sme tu použili tri rôzne premenné: `retazec`, `suma` a `koruny`. Neskôr budeme takto jednoduché programy zapisovať kompaktnejšie. Tu je príklad, ktorý by vám mal ukázať, že takto zatiaľ programy nezapisujte: sú len pre pokročilého čitateľa a hlavne sa v takomto zápise ťažšie hľadajú a opravujú chyby. Nasledovné programy robia skoro presne to isté ako náš predchádzajúci program:

```
suma = float(input('zadaj eura: '))
koruny = suma * 26
print(suma, 'euro je', koruny, 'korun')

suma = float(input('zadaj eura: '))
print(suma, 'euro je', suma * 26, 'korun')

print('zadana suma v euro je', float(input('zadaj eura: ')) * 26, 'korun')
```

Veľa našich programov bude začínať načítaním niekoľkých vstupných hodnôt. Podľa typu požadovanej hodnoty môžeme prečítaný reťazec hneď prekonvertovať na správny typ, napríklad takto:

```
cele = int(input('zadaj celé číslo: '))              # konverovanie na celé číslo
desatinne = float(input('zadaj desatinné číslo: '))  # konverovanie na desatinné
číslo
retazec = input('zadaj znakový reťazec: ')           # reťazec netreba konvertova
ť
```

Úprava pythonovského programu

Programátori majú medzi sebou dohodu, ako správne zapisovať pythonovský kód (oficiálny dokument je PEP 8). My zatiaľ píšeme len veľmi jednoduché zápisy, ale je dôležité si zvykať už od začiatku na správne zápisy. Takže niekoľko základných pravidiel:

- mená premenných obsahujú len malé písmená
- pre znak `=` v priraďovacom príkaze dávame medzeru pred aj za
- operácie v aritmetických výrazoch sú väčšinou tiež oddelené od operandov medzerami
- riadky programu by nemali byť dlhšie ako 79 znakov
- za čiarky, ktoré napríklad oddeľujú parametre v príkaze `print()`, dávame vždy medzeru

Postupne sa budeme zoznamovať aj s ďalšími takýmito odporúčaniami.

V Pythone je zadefinovaných niekoľko štandardných funkcií, ktoré pracujú s číslami. Ukážeme si dve z nich: výpočet absolútnej hodnoty a zaokrúhľovaciu funkciu:

abs() absolútna hodnota

`abs(`*`cislo`*`)`

Parametre

cislo – celé alebo desatinné číslo

Funkcia `abs(cislo)` vráti absolútnu hodnotu zadaného čísla, napríklad:

- `abs(13)` => `13`
- `abs(-3.14)` => `3.14`

Funkcia nemení typ parametra, s ktorým bola zavolaná, t.j. :

```
>>> type(abs(13))
    <class 'int'>
>>> type(abs(-3.14))
    <class 'float'>
```

Ak vyskúšame zistiť typ nie výsledku volania funkcie, ale samotnej funkcie, dostávame:

```
>>> type(abs)
    <class 'builtin_function_or_method'>
```

Totiž aj každá funkcia (teda aj `print` aj `input`) je hodnotou, s ktorou sa dá pracovať podobne ako s číslami a reťazcami, teda ju môžeme napríklad priradiť do premennej:

```
>>> a = abs
>>> print(a)
    <built-in function abs>
```

Zatiaľ je nám toto úplne na nič, ale je dobre o tom vedieť už teraz. Keď už budeme dobre rozumieť mechanizmu priraďovania mien premenných rôznymi hodnotami, bude nám jasné, prečo funguje:

```
>>> vypis = print
>>> vypis
    <built-in function print>
>>> vypis('ahoj', 3 * 4)
    ahoj 12
```

Ale môže sa nám „prihodiť“ aj takýto nešťastný preklep:

```
>>> print=('ahoj')
>>> print('ahoj')
    ...
    TypeError: 'str' object is not callable
```

Do premennej `print`, ktorá obsahovala referenciu na štandardnú funkciu, sme omylom priradili inú hodnotu (znakový reťazec `'ahoj'`) a tým sme znefunkčnili vypisovanie hodnôt pomocou pôvodného obsahu tejto premennej.

Ďalšia funkcia `help()` nám niekedy môže pomôcť v jednoduchej nápovedi k niektorým funkciám a tiež typom. Ako parameter pošleme buď meno funkcie, alebo hodnotu nejakého typu:

```
>>> help(abs)
    Help on built-in function abs in module builtins:

    abs(...)
        abs(number) -> number

        Return the absolute value of the argument.

>>> help(0)
    Help on int object:

    class int(object)
     |  int(x=0) -> integer
     |  int(x, base=10) -> integer
    ...
```

… ďalej pokračuje dlhý výpis informácií o celočíselnom type.

Druhou štandardnou číselnou funkciou je zaokrúhľovanie.

round() zaokrúhľovanie čísla

round(*cislo*)
round(*cislo, pocet*)

Parametre

- **cislo** – celé alebo desatinné číslo
- **pocet** – celé číslo, ktoré vyjadruje na koľko desatinných miest sa bude zaokrúhľovať; ak je to záporné číslo, zaokrúhľuje sa na počet mocnín desiatky

Funkcia `round(cislo)` vráti zaokrúhlenú hodnotu zadaného čísla na celé číslo.
Funkcia `round(cislo, pocet)` vráti zaokrúhlené číslo na príslušný počet desatinných miest, napríklad:

- `round(3.14)` => `3`
- `round(-0.74)` => `-1`
- `round(3.14, 1)` => `3.1`
- `round(2563, -2)` => `2600`

Tiež si o tom môžete prečítať pomocou:

```
>>> help(round)
```

```
    Help on built-in function round in module builtins:

    round(...)
        round(number[, ndigits]) -> number

        Round a number to a given precision in decimal digits (default 0 digits).
        This returns an int when called with one argument, otherwise the
        same type as the number. ndigits may be negative.
```

Ešte raz priraďovacie príkazy

Vráťme sa k priraďovaciemu príkazu:

```
meno_premennej = hodnota
```

- najprv sa zistí hodnota na pravej strane priraďovacieho príkazu => táto hodnota sa vloží do **pamäte hodnôt**
- ak sa toto `meno_premennej` ešte nenachádzalo v **pamäti mien premenných**, tak sa vytvorí toto nové meno
- `meno_premennej` dostane referenciu na novú vytvorenú hodnotu

Pozrime sa na takéto priradenie:

```
>>> ab = ab + 5
    ...
    NameError: name 'ab' is not defined
```

Ak premenná `ab` ešte neexistovala, Python nevie vypočítať hodnotu `ab + 5` a hlási chybovú správu. Skúsme najprv do `ab` niečo priradiť:

```
>>> ab = 13
>>> ab = ab + 5
>>> ab
    18
```

Tomuto hovoríme aktualizácia (update) premennej: hodnotu premennej `ab` sme zvýšili o 5 (najprv sa vypočítalo `ab` + 5, čo je 18) a potom sa toto priradilo opäť do premennej `ab`. Konkrétne takto sme zvýšili (inkrementovali) obsah premennej. Podobne by fungovali aj iné operácie, napríklad:

```
>>> ab = ab * 11
>>> ab
    198
>>> ab = ab // 10
>>> ab
    19
```

Python na tieto prípady aktualizácie nejakej premennej ponúka špeciálny zápis priraďovacieho príkazu:

```
meno_premennej += hodnota        # meno_premennej = meno_premennej + hodnota
```

```
meno_premennej -= hodnota        # meno_premennej = meno_premennej - hodnota
meno_premennej *= hodnota        # meno_premennej = meno_premennej * hodnota
meno_premennej /= hodnota        # meno_premennej = meno_premennej / hodnota
meno_premennej //= hodnota       # meno_premennej = meno_premennej // hodnota
meno_premennej %= hodnota        # meno_premennej = meno_premennej % hodnota
meno_premennej **= hodnota       # meno_premennej = meno_premennej ** hodnota
```

Každý z týchto zápisov je len skrátenou formou bežného priraďovacieho príkazu. Nemusíte ho používať, ale verím, že časom si naň zvyknete a bude pre vás veľmi prirodzený.

Všimnite si, že fungujú aj tieto zaujímavé prípady:

```
>>> x = 45
>>> x -= x          # to isté ako x = 0
>>> x += x          # to isté ako x *= 2
>>> z = 'abc'
>>> z += z
>>> z
    'abcabc'
```

Ďalším užitočným tvarom priraďovacieho príkazu je to, že môžeme priradiť tú istú hodnotu naraz do viacerých premenných. Napríklad:

```
x = 0
sucet = 0
pocet = 0
ab = 0
```

Môžeme to nahradiť jediným priradením:

```
x = sucet = pocet = ab = 0
```

V takomto hromadnom priradení dostávajú všetky premenné tú istú hodnotu, teda referencujú na tú istú hodnotu v pamäti hodnôt.

Posledným užitočným variantom priradenia je tzv. **paralelné priradenie**: naraz priraďujeme aj rôzne hodnoty do viacerých premenných. Napríklad:

```
x = 120
y = 255
meno = 'bod A'
pi = 3.14
```

Môžeme zapísať jedným paralelným priradením:

```
x, y, meno, pi = 120, 255, 'bod A', 3.14
```

Samozrejme, že na oboch stranách priraďovacieho príkazu musí byť **rovnaký počet mien premenných a počet hodnôt**. Veľmi užitočným využitím takéhoto paralelného priradenia je napríklad výmena obsahov dvoch premenných:

```
>>> a = 3.14
>>> b = 'hello'
>>> a, b = b, a          # paralelné priradenie
```

```
>>> a
    'hello'
>>> b
    3.14
```

Paralelné priradenie totiž funguje takto:

- najprv sa zistí postupnosť všetkých hodnôt na pravej strane priraďovacieho príkazu (bola to dvojica `b`, `a`, teda hodnoty `'ahoj'` a `3.14`)
- tieto dve zapamätané hodnoty sa naraz priradia do dvoch premenných `a` a `b`, teda sa vymenia ich obsahy

Zamyslite sa, čo sa dostane do premenných po týchto príkazoch:

```
>>> p1, p2, p3 = 11, 22, 33
>>> p1, p2, p3 = p2, p3, p1
```

alebo

```
>>> x, y = 8, 13
>>> x, y = y, x+y
```

Paralelné priradenie funguje aj v prípade, že na pravej strane príkazu je jediný znakový reťazec nejakej dĺžky a na pravej strane je presne toľko premenných, ako je počet znakov, napríklad:

```
>>> a, b, c, d, e, f = 'Python'
>>> print(a, b, c, d, e, f)
    P y t h o n
```

Keďže tento znakový reťazec je vlastne postupnosť 6 znakov, priradením sa táto postupnosť 6 znakov paralelne priradí do 6 premenných.

Znakové reťazce

Ak znakový reťazec obsahuje dvojicu znakov `'\n'`, tieto označujú, že pri výpise funkciou `print()` sa namiesto nich prejde na nový riadok. Napríklad:

```
>>> a = 'prvý riadok\nstredný\ntretí riadok'
>>> a
    'prvý riadok\nstredný\ntretí riadok'
>>> print(a)
    prvý riadok
    stredný
    tretí riadok
```

Takáto dvojica znakov `'\n'` zaberá v reťazci len jeden znak.

Python umožňuje pohodlnejšie vytvárania takýchto „viacriadkových“ reťazcov. Ak reťazec začína tromi apostrofmi `'''` (alebo úvodzovkami `"""`), môže prechádzať aj cez viac riadkov, ale opäť musí byť ukončený rovnakou trojicou, ako začal. Prechody na nový riadok v takomto reťazci sa nahradia špeciálnym znakom `'\n'`. Napríklad:

```
>>> ab = '''prvý riadok
    stredný
    tretí riadok'''
>>> ab
    'prvý riadok\nstredný\ntretí riadok'
>>> print(ab)
    prvý riadok
    stredný
    tretí riadok
```

Takýto reťazec môže obsahovať aj apostrofy a úvodzovky.

Niekedy potrebujeme vytvárať znakový reťazec pomocou komplikovanejšieho zápisu, v ktorom ho budeme skladať (zreťaziť) z viacerých hodnôt, napríklad:

```
>>> meno, x, y = 'A', 180, 225
>>> r = 'bod ' + meno + ' na súradniciach (' + str(x) + ',' + str(y) + ')'
>>> r
    'bod A na súradniciach (180,225)'
```

Python poskytuje špeciálny typ reťazca (tzv. formátovací znakový reťazec), pomocou ktorého môžeme vytvárať aj takto komplikované výrazy. Základom je formátovacia šablóna, do ktorej budeme vkladať ľubovoľné aj číselné hodnoty. V našom prípade bude šablónou reťazec `f'bod {meno} na súradniciach ({x},{y})'`. V tejto šablóne sa každá dvojica znakov `{...}` nahradí príslušnou hodnotou, v našom prípade týmito hodnotami sú postupne `meno`, `x`, `y`. Všimnite si, znak `f` pred začiatkom reťazca. Zápis takejto formátovacej metódy bude:

```
>>> meno, x, y = 'A', 180, 225
>>> r = f'bod {meno} na súradniciach ({x},{y})'
>>> r
    'bod A na súradniciach (180,225)'
```

V Pythone existuje aj špeciálny typ funkcie (tzv. metódu znakového reťazca), aj nej môžeme vytvárať takto komplikované výrazy. Základom je opäť formátovacia šablóna, do ktorej budeme vkladať ľubovoľné aj číselné hodnoty. V našom prípade bude šablónou reťazec `'bod {} na súradniciach ({},{})'`. V tejto šablóne sa každá dvojica znakov `{}` nahradí nejakou konkrétnou hodnotou, v našom prípade týmito hodnotami sú postupne `meno`, `x`, `y`. Zápis takejto formátovacej metódy bude:

```
>>> meno, x, y = 'A', 180, 225
>>> r = 'bod {} na súradniciach ({},{})'.format(meno, x, y)
>>> r
    'bod A na súradniciach (180,225)'
```

To znamená, že za reťazec šablóny píšeme znak bodka a hneď za tým volanie funkcie `format()` s hodnotami, ktoré sa do šablóny dosadia (zrejme ich musí byť rovnaký počet ako dvojíc `{}`). Neskôr sa zoznámime aj s ďalšími veľmi užitočnými špecialitami takéhoto formátovania.

Cvičenia

L.I.S.T.

- riešenia **aspoň 8 úloh** (ľubovoľných) odovzdaj na úlohový server https://list.fmph.uniba.sk/
  - všetky riešenia môžeš umiestniť do jedného súboru
- používaj len konštrukcie z prednášky (**nepoužívaj žiadne cykly ani iné konštrukcie**)
- pozri si Riešenie úloh 1. cvičenia

1. Pomocou operácie `**` vieme vypočítať mocniny čísel. Ak je exponentom, napríklad zlomok `1/2`, vypočítame tým druhú odmocninu čísla. Zapíš v Pythone:
   - do premennej `a1` priraď druhú odmocninu z `3`
   - do premennej `a2` priraď tretinu tretej odmocniny z `5`
   - do premennej `a3` priraď piatu mocninu piatej odmocniny z `1024`
   - do premennej `a4` priraď desiatu odmocninu z dvadsiatej mocniny `2`

   Hodnoty týchto štyroch premenných potom vypíš v tvare:

```
a1 = 1.7320508075688772
```

2. Ludolfovo číslo `pi` rôzni matematici v histórii počítali podľa zaujímavých magických vzorcov.
   - predpokladaj, že

```
pi = 3.141592653589793
```

   - zisti, ktorý so vzorcov sa k tomuto číslu `pi` priblížil najviac:
     - podiel `223` a `71`
     - súčet zlomkov `22/17`, `37/47` a `88/83`
     - druhá mocnina `99` lomeno súčin `2206` krát druhá odmocnina z `2`
     - druhá odmocnina z `5`, k tomu plus `6`, to celé druhá odmocnina, k tomu plus `7` a to celé opäť druhá odmocnina
     - `10` na `100` lomeno `11222.11122` a to celé `193` odmocnina

   Napríklad podiel `223` a `71` sa od `pi` líši o `0.0007475831672580924`.

3. Napíš program, ktorý pomocou príkazu `input` prečíta `meno` študenta a jeho `vek`. Potom to vypíše pomocou príkazu `print` a tiež vypíše informáciu jeho veku o rok a aj o 10 rokov. Po spustení programu môžeš dostať takýto výpis:

```
zadaj meno: Ema
zadaj vek: 17
Ema má 17 rokov
Ema bude mať o rok 18
Ema bude mať o 10 rokov 27
```

   alebo

```
zadaj meno: Boris
zadaj vek: 5
Boris má 5 rokov
Boris bude mať o rok 6
Boris bude mať o 10 rokov 15
```

4. Napíš program, ktorý prečíta polomer kruhu a vypíše obvod a obsah tohto kruhu. Môžeš predpokladať, že `pi = 3.14159`. Po spustení môžeš dostať:

```
zadaj polomer: 10
obvod je 62.8318
obsah je 314.159
```

5. Napíš program, ktorý prečíta veľkosť strany kocky a vypíše dĺžku stenovej a telesovej uhlopriečky - obe tieto dĺžky zaokrúhli na 2 desatinné miesta (využiješ funkciu `round`). Po spustení môžeš dostať:

```
zadaj veľkosť strany kocky: 18
stenová uhlopriečka je 25.46
telesová uhlopriečka je 31.18
```

6. Napíš program, ktorý prečíta nejaké (aspoň štvorciferné) celé číslo. Potom do štyroch riadkov postupne vypíše:
   - číslo celočíselne delené `10` a zvyšok po delení `10`
   - číslo celočíselne delené `100` a zvyšok po delení `100`
   - číslo celočíselne delené `1000` a zvyšok po delení `1000`
   - číslo celočíselne delené `10000` a zvyšok po delení `10000`

   Takto vieme rozložiť dané číslo na dve časti. Po spustení môžeš dostať:

```
zadaj číslo: 98765
9876 5
987 65
98 765
9 8765
```

   alebo

```
zadaj číslo: 2743
274 3
27 43
2 743
0 2743
```

7. Napíš program, ktorý prečíta tri slová a vypíše všetkých 6 rôznych permutácií. Po spustení môžeš dostať:

```
zadaj 1. slovo: biela
```

```
zadaj 2. slovo: modrá
zadaj 3. slovo: červená
biela modrá červená
biela červená modrá
modrá biela červená
modrá červená biela
červená biela modrá
červená modrá biela
```

alebo

```
zadaj 1. slovo: x
zadaj 2. slovo: ***
zadaj 3. slovo: :-)
x *** :-)
x :-) ***
*** x :-)
*** :-) x
:-) x ***
:-) *** x
```

8. Napíš program, ktorý najprv prečíta do premennej `txt` nejaký text a potom ho pomocou jediného `print` vypíše do 10 riadkov pod seba. Premennú `txt` v príkaze `print` použi len raz. Zrejme využiješ operáciu viacnásobného zreťazenia (`*`) a tiež zreťazenia so špeciálnym znakom `'\n'`. Po spustení môžeš dostať:

```
zadaj text: programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
programujem v Pythone
```

9. Euro je zavedené na Slovensku od 1. januára 2009. Zisti, koľko približne dní od vtedy uplynulo (do dnes uplynulo 12 rokov, 8 mesiacov a 21 dní). Predpokladaj, že každý rok má 365 dní a každý mesiac má 30 dní. Potom vypočítaj koľko je to hodín a aj sekúnd. Po spustení môžeš dostať:

```
počet dní je ????
počet hodín je ?????
počet sekúnd je ?????????
```

Takto by si mohol vypočítať približný počet dní, hodín a sekúnd aj pre svoj vek.

10. Do jedného bajtu (8 bitov) môžeme zapísať čísla od `0` do `255`. Keď máme dvojbajtovú pamäť, môžeme sem zapísať číslo od `0` až do `256*256-1`. Do trojbajtovej pamäte sa

zmestí číslo do `256*256*256-1`. Napíš program, ktorému zadáme počet bajtov a ten potom vypíše maximálne číslo, ktoré sa dá zapísať do takejto pamäte. Program vyskúšaj pre rôzne počty bajtov, napríklad:

```
zadaj počet bajtov: 4
maximálna hodnota je 4294967295
```

alebo

```
zadaj počet bajtov: 10
maximálna hodnota je 1208925819614629174706175
```

Aké veľké číslo sa zmestí do 100 bajtov alebo jedného kilobajtu (1000 bajtov)? Vedel/a by si zistiť, koľko cifier má takéto číslo?

11. V jednom starodávnom príbehu sa na políčka šachovnice kládli zrniečka ryže: na 1. políčko `1` zrnko ryže, na ďalšie `2`, na každom ďalšom je dvojnásobok predchádzajúceho. Napíš program, ktorý vypíše, koľko zrniek bude na `n`-tom políčku. Po spustení môžeš dostať:

```
zadaj n: 5
na 5. políčku bude 16 zrniek ryže
```

Všimni si, že za poradovým číslom vo výpise je bodka (`5.`) a je to zapísané bez medzery.

Zisti počet zrniek ryže na 64. políčku šachovnice. Odhadni koľko je to ton ryže, keď 50 zrniek váži asi 1 gram.

12. Napíš program, ktorý prečíta dve celé čísla (napríklad `27` a `342`) a vypíše ich v tvare takejto rovnosti: `27+342=369`, teda **bez medzier**. Použi na to formátovaciu šablónu `f'...{hodnota}...'`. Po spustení teda môžeš dostať:

```
zadaj 1. číslo: 27
zadaj 2. číslo: 342
27+342=369
```

alebo

```
zadaj 1. číslo: 8
zadaj 2. číslo: 999997
8+999997=1000005
```