Olomoucký informatický korespondenční seminář – Olinx

Programování s čísly v Pythonu, 1. díl

Jan Laštovička

olinx.inf.upol.cz

Tento seminář zkoumá čísla pomocí programovacího jazyka Python. Soustředíme se pouze na celá nezáporná čísla, tedy na celá čísla, která jsou větší nebo rovno než nula. Za nulou následuje číslo jedna a za ním číslo dva a tak dále. Úsporněji můžeme napsat, že se jedná o čísla $0, 1, 2, 3, \ldots$ Protože jiná čísla zde neuvidíme, budeme jim prostě říkat *čísla*.

Jazyk Python úkony s čísly (sčítání, odčítání, násobení, ...) rovnou podporuje. My si ale postupně představíme, jak by úkony s čísly mohly být definovány.

1 První kroky

Budeme potřebovat interpret jazyka Python. To je program, který umožňuje spouštět programy napsané v Pythonu. Můžete si jej stáhnout ze stránek `https://www.python.org`.

Spolu s interpretem se vám do počítače nainstaluje i jednoduché vývojové prostředí `IDLE`. Jméno `IDLE` je zkratka za Integrated Development and Learning Environment (jednotné vývojové a výukové prostředí). Vývojové prostředí je program, který nám pomáhá psát programy. Spusťte vývojové prostředí `IDLE`. Po spuštění se otevře okno nazvané `Shell` obsahující interpret jazyka Python.

Interpret za znaky `>>>` očekává váš vstup. Napište zde `0` a stiskněte klávesu `Return` (nebo `Enter`). Interpret odpověděl tak, že nulu zopakoval a na dalším řádku opět očekává vstup:

```
>>> 0
0
>>>
```

Nula patří mezi *hodnoty*. Zadání hodnoty na vstup interpretu vede pouze k jejímu vytištění.

Pro popis hodnoty výpočtem se používají *výrazy*. Každá hodnota je velmi jednoduchým výrazem, jehož hodnota je opět ona sama. Tedy `0` je i výraz, jehož hodnota je zase nula. Přestože všechna čísla jsou hodnoty, jediné číslo, které nyní dokážeme získat, je nula.

Každé číslo má svého *následovníka*. To je číslo, které obdržíme přičtením jedničky. Následovníkem nuly je jednička, následovníkem jedničky dvojka a tak dále. Zavedeme si výraz, jehož hodnota je následovník určitého čísla. Pokud v je výraz, pak

`(`v `+ 1)`

je *výrazem následovníka*. Předchozí větu je třeba chápat jako recept na vytváření nových výrazů. Například již víme, že `0` je výraz. Použijeme `0` na místo v a obdržíme, že i `(0 + 1)` je výraz. Zkusme jej zadat interpretu jako vstup:

```
>>> (0 + 1)
1
```

Vidíme, že interpret jako odpověď vytiskl `1`. Hodnota výrazu `(`v `+ 1)` je následovník hodnoty výrazu v. Tedy hodnota `(0 + 1)` je následovník čísla nula neboli číslo jedna.

Protože `(0 + 1)` je opět výraz, můžeme opět použít výše uvedené pravidlo a získat výraz `((0 + 1) + 1)`. Jeho hodnota je předvídatelně `2`.

Výrazy, které sestavujeme z jiných výrazů, se nazývají *složené*. Výraz následovníka je složeným výrazem, ale číslo nula složeným výrazem není.

Vnější závorky kolem výrazů budeme vynechávat. Proto výraz `((0 + 1) + 1)` můžeme zkráceně zapsat jako `(0 + 1) + 1`. Ostatní závorky, přestože se vám zdají zbytečné, ponecháme. Tedy `0 + 1 + 1` není výraz.

Pomocí následovníka se můžeme dostat k libovolně velikému číslu. Opravdu, i číslo 299 792 458 vyjadřující rychlost světla ve vakuu v metrech za sekundu lze získat jako následovníka čísla 299 792 457, které opět je následovníkem jistého čísla a tak dále až se po úmorně dlouhé době dostaneme k nule. Teoreticky lze tedy napsat výraz, jehož hodnota by byla světelná rychlost. Ve skutečnosti byste měli už dost práce s napsáním výrazu, jehož hodnota bude rovna vašemu věku.

Pro jednoduchost si umožníme zadat libovolné číslo tak, že napíšeme za sebe jeho číslice. Připomeňme, že číslice je znak, kterým zapisujeme čísla. V naší desítkové soustavě používáme deset číslic: 0, 1, 2, 3, 4, 5, 6, 7, 8 a 9. Tedy přesněji zápis číslic libovolného čísla a je výraz s hodnotou a. Díky tomu výrazy `((0 + 1) + 1) + 1` a `3` mají oba hodnotu číslo tři.

Hodnota výrazu je zatím vždy jedno pevné číslo. Ať výraz `(5 + 1) + 1` zadáme interpretu kdykoliv, vždy bude jeho hodnota sedm. Aby se hodnoty výrazů mohly s časem měnit, zavedeme si *proměnné*. Pokud p je proměnná a v výraz, pak

p `=` v

je *příkaz nastavení hodnoty proměnné*. Jako proměnné budeme zatím používat pouze malá písmena anglické abecedy (`a`, `b`, `c`, ...). Pokud p položíme rovno `a` a v rovno `0`, obdržíme příklad příkazu nastavení hodnoty proměnné:

```
a = 0
```

Příkaz můžeme předat interpretu na vstup:

```
>>> a = 0
>>>
```

Všimněte si, že na rozdíl od výrazu interpret nevytiskl žádný výsledek. Vykonání příkazu se provede tak, že se nastaví hodnota proměnné p na hodnotu výrazu v. Příklad tedy nastavil hodnotu proměnné `a` na číslo `0`.

Každá proměnná je výrazem, kde hodnotou výrazu je hodnota proměnné. Výraz `a` bude nyní mít hodnotu `0`:

```
>>> a
0
```

Protože proměnná je výrazem, můžeme ji použít při vytváření složených výrazů. Například:

```
>>> (a + 1) + 1
2
```

Hodnota výrazu `(a + 1) + 1` je závislá na hodnotě proměnné `a`. Příkazem nastavení hodnoty proměnné můžeme hodnotu proměnné změnit. Nastavme hodnotu proměnné `a` na `1`:

```
>>> a = 1
```

Změnila se i hodnota výrazu:

```
>>> (a + 1) + 1
3
```

Na místě v v příkazu nastavení hodnoty proměnné může být libovolný výraz. Můžeme tedy nastavit hodnotu proměnné `b` na následovníka hodnoty proměnné `a`:

```
>>> b = a + 1
>>> b
2
```

Můžeme také zvýšit hodnotu proměnné `a` o jedna:

```
>>> a = a + 1
>>> a
2
```

Detailněji se stalo to, že se nejprve získala hodnota výrazu `a + 1`, která je číslo `2`, a poté se změnila hodnota proměnné `a` na číslo `2`.

Každé nenulové číslo má svého předchůdce. *Předchůdce* nenulového čísla a je číslo b, pro které platí, že následovník b je číslo a. Například předchůdce čísla jedna je nula, protože následovník nuly je jednička. Podobně předchůdce čísla dva je číslo jedna a tak dále. Nula nemá žádného předchůdce.

Zavedeme si výraz, jehož hodnota bude předchůdce nenulového čísla. Je-li v výraz s nenulovou hodnotou, pak

`(`v `- 1)`

je *výraz předchůdce*. Hodnotou výrazu je předchůdce hodnoty výrazu v. Ukážeme si příklady použití:

```
>>> (0 + 1) - 1
0
>>> a = 2
>>> a - 1
1
>>> a = a - 1
>>> a
1
```

2 Program

Interpret můžeme spustit tak, aby vykonal všechny příkazy zapsané v souboru. Takový soubor příkazů nazýváme *programem*. Nejprve vytvoříme jednoduchý program. Volbou `File / New File` hlavní nabídky otevřete nové okno. Napište do něj následující obsah:

```
a = 5
vystup = (a + 1) + 1
```

Nyní volbou `File / Save` uložte obsah do souboru na disk pod názvem `p1.py`. Přípona souboru `py` prozrazuje operačnímu systému, že se jedná o program napsaný v jazyce Python. Právě jste vytvořili svůj první program. Můžete jej spustit volbou `Run / Run Module` z hlavní nabídky nebo stiskem klávesy `F5`. Vstupem `vystup` zadaným do okna `Shell` zobrazíte výstup programu:

```
>>> vystup
7
```

Náš první program se skládá ze dvou po sobě následujících příkazů. Obecně platí, že každý program je tvořen pod sebou napsanými příkazy. Vykonání programu probíhá tak, že se postupně vykonávají jeho příkazy. V našem případě se nejprve vykoná příkaz

```
a = 5
```

a tím proměnná `a` získá hodnotu `5`. Dále se vykoná příkaz

```
vystup = (a + 1) + 1
```

který nastaví hodnotu proměnné `vystup` na `7` ($5+1+1=7$). Proměnná `vystup` má pro nás ten význam, že její hodnotu po skončení vykonávání programu, budeme považovat za výstup programu. Program začíná nastavením hodnot *vstupním proměnným*. Počet vstupních proměnných a jejich možné hodnoty určuje zadání programu. Zadání našeho programu by mohlo znít:

> Na vstup je dáno libovolné číslo a. Napište program, jehož výstup bude číslo $a + 2$.

Zkráceně můžeme také psát:

> Vraťte číslo o dva větší než zadané číslo.

Zajímavé je, že v této zkrácené verzi nevystupuje jméno zadaného čísla. Víme jen, že máme napsat program, který má jednu vstupní proměnnou. Ještě stručnější zápis téhož vypadá takto:

> Zvětšete zadané číslo o dva.

Hodnotu vstupní proměnné můžeme v rámci dohodnutých možností změnit:

```
a = 7
vystup = (a + 1) + 1
```

a obdržíme jiný výstup:

```
>>> vystup
9
```

Do programu můžeme pro zvýšení čitelnosti vkládat komentáře. Vše, co se nalézá za znakem `#` (křížek) do konce řádky, je komentář. Program budeme začínat komentářem, který popisuje jeho vstupní proměnné a výstup. Ukázkový program by tedy vypadal takto:

```
# Výstupem programu bude číslo
# o dva větší než zadané číslo.
a = 5
vystup = (a + 1) + 1
```

Popis programů v semináři najdete v textu nad programem. Proto si dovolíme komentáře vypouštět. Každý váš program by však vždy měl začínat komentářem popisujícím vstupní proměnné a výstup programu.

3 Aritmetika

V této části si zavedeme aritmetiku s čísly. Budeme vycházet z možnosti získat předchůdce a následovníka čísla představené v předchozí části. Postupně si ukážeme, jak s jejich pomocí zavést další úkony s čísly. Začneme sčítáním, pomocí kterého zavedeme násobení. Násobení čísel nás dovede k mocnině. Ani další početní úkony (rozdíl, dělení, ...) neuniknou našemu pohledu.

3.1 Součet a rozdíl

Stanovme si za cíl napsat program, který pro dvě zadaná čísla a a b na vstupu vrátí na výstup jejich součet $a + b$.

Předpokládejme nejdříve, že číslo a je rovno nule. Víme, že pro libovolné číslo b platí

$$0 + b = b.$$

Součet je tedy přímo číslo b. Vyjádřeno programem dostáváme:

```
a = 0
b = 5
vystup = b
```

Co když by a bylo rovno jedné? Obecně platí, že součet nenulového a a libovolného b bude stejný jako součet předchůdce a a následovníka b. Vyjádřeno symbolicky dostáváme, že

$$a + b = (a - 1) + (b + 1).$$

Pokud tedy a je rovno jedna, pak předchůdce a je nula a můžeme použít předchozí program:

```
a = 1
b = 5
a = a - 1
b = b + 1
vystup = b
```

Pro libovolné `a` bychom chtěli opakovat třetí a čtvrtý řádek programu, dokud je `a` nenulové.

Za tímto účelem si zavedeme nový příkaz. Pokud v je výraz a $p_1, \ldots, p_n$ je aspoň jeden příkaz, pak

```
while v:
    p_1
    ⋮
    p_n
```

je *příkaz podmíněného opakování*. Příkazy $p_1, \ldots, p_n$ jsou odsazeny o čtyři mezery. V editoru vložíte čtyři mezery jedním stiskem klávesy tabulátor. Příkaz se vykoná tak, že se opakuje následující. Pokud je hodnota výrazu v nenulová, vykonají se postupně příkazy $p_1 \ldots, p_n$ a pokračuje opakování. Je-li ale hodnota výrazu v nula, opakování končí a tím skončí i vykonávání příkazu.

Nyní již dokážeme napsat program, jehož výstupem je součet dvou zadaných čísel.

```
a = 6
b = 5
while a:
    a = a - 1
    b = b + 1
vystup = b
```

Po spuštění programu se můžeme přesvědčit, že $6 + 5 = 11$:

```
>>> vystup
11
```

Pro získání představy o tom, jak se program vykonával, je užitečné tisknout měnící se hodnoty proměnných do okna `Shell`. K tomu slouží následující příkaz. Jsou-li $v_1 \ldots, v_n$ výrazy, pak

```
print(v_1, ..., v_n)
```

je *příkaz tisku*. Příkaz tisku se vykoná tak, že se postupně vytisknou hodnoty výrazů $v_1 \ldots, v_n$ oddělené mezerami a odřádkuje se. Například:

```
>>> print(1, 2 + 1)
1 3
```

Použitím příkazu tisku se můžeme podívat, jak se měnily hodnoty proměnných `a` a `b`:

```
a = 6
b = 5
while a:
    a = a - 1
    b = b + 1
    print(a, b)
vystup = b
```

Program vytiskne:

```
5 6
4 7
3 8
2 9
1 10
0 11
```

Vidíme, jak hodnota proměnné `a` postupně klesala k nule.

Výraz následovníka si můžeme představit jako součet daného čísla a čísla jedna. Představený program sčítající libovolná čísla nás ospravedlňuje rozšířit působnost výrazu na součet libovolných čísel. Přesněji jsou-li v_1 a v_2 výrazy, pak

```
(v_1 + v_2)
```

je *výraz součtu*. Hodnota výrazu, jak nejspíše očekáváte, je součet hodnot výrazů v_1 a v_2. Proto například máme:

```
>>> 6 + 5
11
```

Úkol 1 *5 bodů*

Napište program, jehož výstup bude rozdíl dvou zadaných čísel a a b, kde b je menší nebo rovno než a.

Seminář obsahuje bodované úkoly, ve kterých budete vytvářet programy. V programech můžete použít pouze prostředky představené před zadáním úkolu. Řešení odevzdávejte jako jeden archiv formátu ZIP přes web semináře https://olinx.inf.upol.cz. Termín odevzdání najdete na stejném webu.

Předpokládejme splnění prvního úkolu. To nás vede k zavedení následujícího výrazu. Pokud první úkol bude nad vaše síly, stále můžete tento výraz používat. To platí i pro další úkoly. Jsou-li v_1 a v_2 výrazy, kde hodnota výrazu v_1 je větší nebo rovna než hodnota výrazu v_2, pak

```
(v_1 - v_2)
```

je *výraz rozdílu*. Hodnota výrazu je rozdíl hodnot výrazů v_1 a v_2. Vstup

```
4 - 5
```

pro nás nemá smysl. Hodnotou by bylo záporné číslo, které nepřipouštíme.

Následují ukázky použití výrazu rozdílu.

```
>>> 5 - 2
3
>>> (5 - 2) - 1
2
```

3.2 Násobení a mocniny

Součin dvou čísel zavedeme podobně jako jsme zavedli sčítání. Chceme vynásobit dvě čísla a a b. Pokud je a nulové, pak

$$0 \cdot b = 0.$$

Tedy:

```
a = 0
b = 2
vystup = 0
```

Uvědomme si, že násobení je jen opakované sčítání. Proto pro nenulové číslo a a libovolné číslo b platí:

$$a \cdot b = (a - 1) \cdot b + b,$$

Tedy když by a bylo rovno jedné můžeme situaci převést na předchozí případ:

```
a = 1
b = 3
vystup = 0
vystup = vystup + b
a = a - 1
```

a dostáváme:

```
>>> a
0
>>> vystup
3
```

K dosažení řešení již stačí čtvrtý a pátý řádek opakovat, dokud je hodnota proměnné `a` nenulová:

```
a = 2
b = 3
vystup = 0
while a:
    vystup = vystup + b
    a = a - 1
```

Program spočítal, že dva krát tři je:

```
>>> vystup
6
```

Když víme, jak zavést součin, můžeme jednodušeji použít k násobení čísel následující výraz. Pro výrazy v_1 a v_2 je

`(`v_1 `*` v_2`)`

výraz součinu. Hodnotou výrazu je součin hodnot výrazů v_1 a v_2. Můžeme tedy počítat:

```
>>> 5 * 5
25
>>> 0 * 5
0
>>> 5 * 1
5
```

Protože mocnění je opakované násobení, hlásí se o slovo další úkol.

Úkol 2 *6 bodů*

Jsou dána čísla a a b. Vraťte b-tou mocninu čísla a. Pro a a b rovno nule vraťte jedničku: $0^0 = 1$.

Při programování mocniny lze využít toho, že pro libovolné číslo a a nenulové číslo b platí

$$a^b = a^{b-1} \cdot a.$$

Dále pro libovolné číslo a máme

$$a^0 = 1.$$

Hodnota nulté mocniny čísla nula je obecně neurčitá. My si situaci zjednodušujeme a pokládáme $0^0 = 1$.

Pro výrazy v_1 a v_2 je

`(`v_1 `**` v_2`)`

výraz umocňování. Hodnota výrazu v_1 určuje mocněnce a hodnota výrazu v_2 mocnitele. Následují ukázky výpočtů mocnin.

```
>>> 2 ** 3
8
>>> 5 ** 1
5
>>> 0 ** 3
0
>>> 3 ** 0
1
>>> 0 ** 0
1
```

Pomocí mocnin můžeme dosáhnout obřích čísel. Například hodnota výrazu `10 ** (10 ** 3)` má 1 001 číslic. Pokud by nějaký výpočet trval příliš dlouho a vy byste bezradně sledovali, jak vám dochází místo operační paměti počítače, můžete využít volby `Shell / Reset Shell` z hlavní nabídky pro restart interpretu.

3.3 Rovnost a nerovnost

Jak programem rozhodneme, zda se dvě čísla rovnají? Vyjdeme z toho, že nula se rovná pouze nule. Dále pokud jsou obě čísla nenulová, porovnáme jejich předchůdce. To si můžeme dovolit, protože pro nenulová čísla a a b je

$a = b$ právě tehdy, když $a - 1 = b - 1$.

Opakováním tohoto pravidla se nakonec dostaneme do situace, kde aspoň jedno číslo je nulové. Nyní stačí odpovědět ano v případě, že jsou obě čísla nulová, jinak je odpověď ne.

K napsání programu potřebujeme zodpovědět tři otázky. Za prvé co znamená ano a co ne? Za druhé jak vyjádřit, že musí platit dvě podmínky současně? Za třetí jak zjistíme, že je číslo nulové?

Odpověď na první otázku je, že číslo nula znamená ne (budeme také říkat *nepravdu*) a ostatní čísla znamenají ano (*pravdu*). Program tedy musí vrátit nulu, pokud se čísla nerovnají a libovolné jiné číslo v případě, že se rovnají. Pokud nás na hodnotě zajímá pouze to, je-li rovna nule, říkáme jí *pravdivostní hodnota*. Pokud nás na výrazu zajímá pouze nulovost jeho hodnoty, říkáme mu *podmínka*. Pokud má podmínka nenulovou hodnotu říkáme, že je *splněná*. Připomeňme si příkaz podmíněného opakování:

```
while v:
    p_1
    ⋮
    p_n
```

Výraz v tedy můžeme nazvat podmínkou a říci, že příkaz vykonává příkazy $p_1, \ldots, p_n$ dokud je podmínka v splněná.

Zamysleme se nad druhou otázkou. Máme dvě podmínky v_1 a v_2. Chceme vytvořit podmínku, která je splněná právě tehdy, když jsou obě podmínky v_1 a v_2 splněné. Připomeňme, že podmínka je splněná, když je její hodnota nenulovým číslem. Tedy chceme za použití výrazů v_1 a v_2 vytvořit výraz, jehož hodnota bude nenulové číslo právě tehdy, když hodnoty obou výrazů v_1 a v_2 jsou nenulová čísla. Když se chvíli zamyslíme, přijdeme na to, že stačí použít výraz součinu `(`v_1 `*` v_2`)`.

Součin čísel je nenulový právě tehdy, když oba činitelé jsou nenuloví:

```
>>> 1 * 2
2
>>> 0 * 2
0
>>> 0 * 0
0
```

Podobně součet pravdivostních hodnot rozhoduje, zda je aspoň jedna z nich pravdivá:

```
>>> 1 + 0
1
>>> 0 + 0
0
>>> 1 + 1
2
```

Dostáváme se ke třetí otázce o zjištění nulovosti čísla. Pokud v je výraz, pak

`(`v `== 0)`

je *podmínka rovnosti čísla s nulou*. Podmínka je splněná v případě, že hodnota výrazu v je nula a nesplněná v případě, že hodnota v je nenulové číslo. Když si výraz vyzkoušíme, zjistíme, že hodnotou výrazu jsou pro nás zatím neznámé hodnoty `True` a `False`:

```
>>> 0 == 0
True
>>> 1 == 0
False
```

Hodnota `True` je ale jen jiný název pro jedničku a hodnota `False` pro nulu. Můžeme například spočítat:

```
>>> True + 4
5
>>> False + 2
2
```

Otázky jsou zodpovězeny a můžeme se pustit do tvorby programu. Máme na vstupu dvě čísla:

```
a = 10
b = 10
```

Chceme provádět níže uvedené příkazy, dokud jsou čísla `a`, `b` nenulová.

```
a = a - 1
b = b - 1
```

Toho dosáhneme příkazem podmíněného opakování:

```
while a * b:
    a = a - 1
    b = b - 1
```

Opakujeme dokud je `a` nenulové a současně `b` je nenulové. Výstup nyní bude pravdivý právě tehdy, když `a` bude nulové a současně `b` bude nulové:

```
vystup = (a == 0) * (b == 0)
```

Celkový program vypadá následovně:

```
a = 10
b = 10
while a * b:
    a = a - 1
    b = b - 1
vystup = (a == 0) * (b == 0)
```

Po spuštění programu ověříme jeho výstup.

```
>>> vystup
1
```

Číslo jedna je nenulové a znamená tedy pravdu. Dostáváme, že čísla se rovnají. Změníme vstup `a` na číslo `11` a spustíme. Výstup bude `0` tedy nepravda.

Říkáme, že program *rozhoduje* o tom, zda se dvě čísla rovnají. To znamená, že pokud je odpověď ano (čísla se rovnají), pak musí být výstupem programu pravda (nenulové číslo). V opačném případě musí být výstupem nepravda (číslo nula).

Nyní si můžeme dovolit porovnávat dvě čísla na rovnost. Pokud v_1 a v_2 jsou výrazy, pak

`(`v_1 `==` v_2`)`

je *podmínka rovnosti*, jejíž hodnota je pravda v případě, že hodnoty v_1 a v_2 se rovnají, jinak je hodnota výrazu nepravda. Můžeme také stručně říci, že výraz rozhoduje, zda se hodnoty výrazů v_1 a v_2 rovnají. Například:

```
>>> (1 + 1) == 2
True
```

Úkol 3 *5 bodů*

Jsou dána dvě čísla a a b. Rozhodněte, zda je a menší nebo rovno než b.

Výraz, který rozhoduje, zda je jedno číslo menší nebo rovno než druhé následuje. Pokud v_1 a v_2 jsou výrazy, pak

`(`v_1 `<=` v_2`)`

je podmínka, která je splněná právě tehdy, když hodnota výrazu v_1 je menší nebo rovna než hodnota výrazu v_2. Proto:

```
>>> 3 <= (2 * 3)
True
```

Jak z podmínky v vytvořit podmínku v', která je splněná právě tehdy, když v splněná není? Chceme vytvořit výraz v', který bude nulový právě tehdy, když bude výraz v nenulový. To je ale přesně porovnání v s nulou. Proto stačí položit v' rovno `(`v `== 0)`. Skutečně:

```
>>> 0 == 0
True
>>> 1 == 0
False
```

Připomeňme, že `True` je jednička a `False` nula.

Úkol 4 *4 body*

Jsou dána dvě čísla a a b. Rozhodněte, zda je číslo a menší než číslo b.

Pro výrazy v_1 a v_2 je

$(v_1$ `<` $v_2)$

podmínka, která je splněna právě tehdy, když hodnota v_1 je menší než hodnota v_2. Nerovnosti můžeme otočit. Proto bez dalšího vysvětlování zaveďme pro výrazy v_1 a v_2 podmínky:

$(v_1$ `>` $v_2)$
$(v_1$ `=>` $v_2)$

3.4 Dělení

V poslední části se zaměříme na celočíselné dělení nazývané zde pouze dělení. Pro libovolné číslo a a nenulové číslo b chceme zjistit výsledek dělení čísla a číslem b. Přesněji hledáme největší číslo c takové, že

$$b \cdot c \leq a.$$

Například pro a rovno 7 a b rovno 2 dostáváme, že c je 3, protože $2 \cdot 3 = 6 \leq 7$, ale $2 \cdot 4 = 8 \not\leq 7$. Trojka je tedy největší číslo, které splňuje předepsanou nerovnost, a proto sedm děleno dvěma je tři.

Pokud by a bylo menší než b, pak výsledkem dělení je nula. V této situaci jedině pro c rovno nule platí $b \cdot c \leq a$. Například výsledek dělení čísla tři číslem čtyři je nula. Tuto skutečnost zohledníme zárodkem programu počítajícího celočíselné dělení:

```
a = 3
b = 4
vystup = 0
```

Pokud je b menší nebo rovno než a, tak platí, že výsledek dělení čísla a číslem b je roven následovníku výsledku dělení čísla $a - b$ číslem b. Vyjádřeno programem dostáváme:

```
vystup = vystup + 1
a = a - b
```

Tyto příkazy opakujeme dokud b je menší nebo rovno než a:

```
while b <= a:
    vystup = vystup + 1
    a = a - b
```

Dostáváme výsledný program:

```
a = 14
b = 3
vystup = 0
while b <= a:
    vystup = vystup + 1
    a = a - b
```

Co by se stalo, když by `b` bylo rovno nule? Vidíme, že program po spuštění nechce skončit. Podmínka podmíněného opakování `b <= a` bude navždy splněna, protože číslo nula je menší nebo rovno než jakékoliv číslo. Musíme ještě zkontrolovat, že se s hodnotou proměnné `a` nedostaneme do zakázané oblasti záporných čísel. Hodnotu proměnné `a` měníme pouze příkazem `a = a - b`, ale hodnota proměnné `b` je nula, takže tento příkaz ve skutečnosti hodnotu proměnné `a` nezmění. Nekonečné opakování ukončíme stiskem `Ctrl + C` nebo volbou `Shell / Interrupt Execution` z hlavní nabídky.

Úkol 5 *5 bodů*

Pro zadané číslo a a nenulové číslo b zjistěte zbytek po dělení čísla a číslem b.

Pokud a děleno b je c, pak zbytek po dělení čísla a číslem b je číslo $a-(b \cdot c)$. Například sedm děleno třemi je dva, a proto zbytek po dělení sedmičky trojkou je jedna $(7 - (3 \cdot 2))$.

Pro výrazy v_1 a v_2 je

$(v_1$ `//` $v_2)$

výrazem dělení a

$(v_1$ `%` $v_2)$

výrazem zbytku po dělení. Několik výpočtů následuje.

```
>>> 4 // 2
2
>>> 4 // 3
1
>>> 4 // 5
0
>>> 8 % 3
2
>>> 9 % 3
0
>>> 0 % 3
0
```