KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
jan.lastovicka@upol.cz
www.inf.upol.cz/lide/jan-lastovicka
17. listopadu 12, 771 46 Olomouc

Základy programování pro IT 1

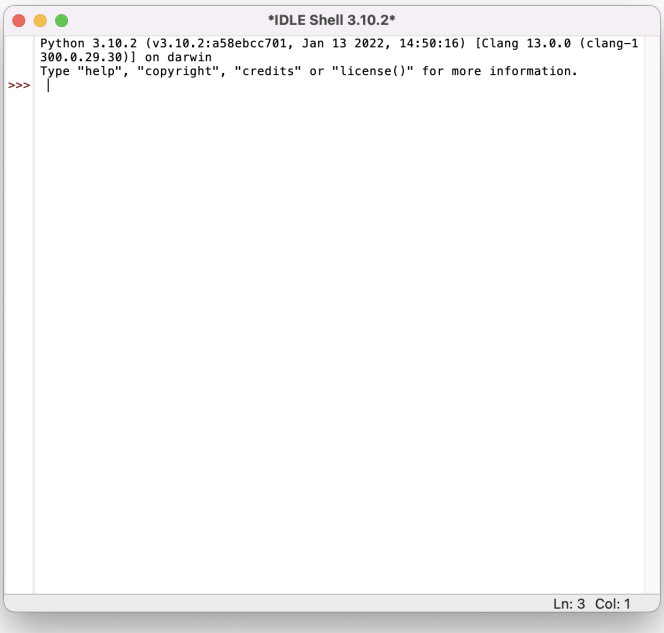
1. Výrazy a příkazy

verze z 23. září 2024

V tomto předmětu se budeme zabývat základy imperativního programování, které budeme demonstrovat v jazyce Python. Principy však budou použitelné i na další jazyky, které imperativní programování podporují.

Budeme potřebovat interpret jazyka Python verze 3. Interpret je program, který umožňuje vykonávat 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 nám pomáhá vytvářet programy. Spusťte vývojové prostředí `IDLE`. Po spuštění se otevře okno nazvané `Shell` obsahující interpret jazyka Python. Okno `Shell` je zachyceno na Obrázku 1.



Obrázek 1: Okno `Shell` vývojového prostředí `IDLE`.

Interpret za znaky `>>>` očekává vstup.

1 Hodnoty

Program má data, se kterými pracuje, k dispozici ve formě **hodnot**. Podle druhu dat dělíme hodnoty do různých **typů**. Zatím si představíme jediný typ hodnot a to **celá čísla**. Nezáporné číslo můžeme zadat tak, že za sebe napíšeme jeho číslice. Například číslo dvanáct zadáme jako `12`. Jak zadat záporná čísla se dozvíme později. Pokud interpretu zadáme na vstup hodnotu a volbu potvrdíme stiskem klávesy `Return` (nebo `Enter`), interpret hodnotu vytiskne a vyžádá si další vstup:

```
>>> 12
12
>>>
```

2 Výrazy

K zadání hodnoty výpočtem používáme **výrazy**. **Vyhodnocením výrazu** získáme jeho hodnotu. Každá hodnota je výrazem a vyhodnocuje se na sebe samu. Například číslo `12` je výraz, který se vyhodnotí na číslo `12`. Zadáme-li interpretu výraz, vyhodnotí jej a vytiskne jeho hodnotu.

Výrazy můžeme skládat. Vezmeme-li dva výrazy označené jako *`expr1`* a *`expr2`* můžeme vytvořit **výraz součtu**:

```
(expr1 + expr2)
```

Například již víme, že čísla `1` a `2` jsou výrazy. Můžeme je tedy použít místo *`expr1`* a *`expr2`* a vytvořit výraz `(1 + 2)`.

Výraz se nazývá **složený**, pokud vznikl z jiných výrazů. Například `(1 + 2)` je složený výraz, ale výrazy `1` a `2` složené nejsou.

Vyhodnocení výrazu součtu `(`*`expr1`*` + `*`expr2`*`)` se provede tak, že se nejprve získá hodnota *`number1`* výrazu *`expr1`*, poté se získá hodnota *`number2`* výrazu *`expr2`* a nakonec se provede součet čísel *`number1`* a *`number2`*. Součet čísel bude hodnotou výrazu součtu.

Například vyhodnocení výrazu `(1 + 2)` se provede tak, že se nejprve vyhodnotí výraz `1`, tím se získá číslo jedna, poté se vyhodnotí výraz `2`, který má jako hodnotu číslo dva, nakonec se sečte jedna a dva. Tím se získá číslo tři. Hodnota výrazu `(1 + 2)` je tedy číslo tři. Proto:

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

Protože `(1 + 2)` je opět výraz, můžeme jej použít na místě *`expr1`* ve výrazu součtu a při dosazení výrazu `3` za *`expr2`* dostat, že i `((1 + 2) + 3)` je výraz. Vyhodnocení výrazu `((1 + 2) + 3)` proběhne tak, že se nejprve sečte jedna a dva, tím se získá číslo tři, a poté se sečte tři a tři a obdrží hodnota šest. Výsledek můžeme porovnat s hodnotou, kterou spočítá interpret:

```
>>> ((1 + 2) + 3)
6
```

Výraz `(1 + (2 + 3))` se také vyhodnotí na číslo šest, ale průběh vyhodnocení se liší. Zde se nejprve sečte dva a tři. Získáme číslo pět a poté součtem jedničky a pětky obdržíme číslo šest. Ověříme:

```
>>> (1 + (2 + 3))
6
```

Nejvíce vnější závorky kolem výrazu budeme vynechávat. Tedy místo `(1 + (2 + 3))` můžeme psát `1 + (2 + 3)`.

Pro jednodušší vyjadřování nebudeme často rozlišovat mezi výrazem a jeho hodnotou. Můžeme například říci, že pokud máme čísla *`number1`* a *`number2`*, pak

```
(number1 + number2)
```

je výraz součtu. Myslíme tím, že místo čísel *`number1`* a *`number2`* můžeme uvést i výrazy, které mají jako hodnotu číslo.

3 Operátory

Výrazy můžeme vytvářet pomocí **operátorů**. Každý operátor má vymezeno, kolik výrazů potřebuje, aby se s jeho pomocí dal vytvořit výraz nový. Například operátor `+` vyžaduje dva výrazy. Operátory, které vyžadují jeden výraz, se nazývají **unární** a dva **binární**. Počet výrazů, které operátor vyžaduje, se nazývá jeho **arita**. Operátor `+` je binární a má tedy aritu dva.

Pokud je *`operator`* unární operátor a *`operand`* výraz, pak

```
(operator operand)
```

je výraz. Dále pokud je *operator* binární operátor a *operand1* a *operand2* jsou výrazy, pak

```
(operand1 operator operand2)
```

je výraz. Výrazy *operand*, *operand1* a *operan2* se nazývají **operandy**. Výraz vytvořený operátorem se vyhodnotí tak, že se nejprve vyhodnotí jeho operandy (u binárního operátoru nejprve první a poté druhý operand). Tím se získají hodnoty nazývané **argumenty**. Operátor pak podle typů argumentů vybere **operaci**, kterou na ně aplikuje. Výsledek aplikace operace je hodnotou výrazu.

Následuje přehled operátorů a příslušných operací, které operátory zvolí pro celá čísla.

operátor	arita	operace
`-`	1	opačné číslo
`-`	2	rozdíl čísel
`+`	2	součet čísel
`*`	2	součin čísel
`//`	2	celočíselný podíl
`%`	2	zbytek po celočíselném podílu
`**`	2	mocnění čísel

Operátor minus (`-`) lze požít jako unární i jako binární operátor. U unárního operátoru minus vynecháváme mezeru mezi operátorem a jeho operandem: `(-`*`number`*`)`.

Záporné číslo můžeme získat jako opačné číslo ke kladnému číslu. Například:

```
>>> -5
-5
```

Výrazy můžeme kombinovat. Například soušet dvou záporných čísel:

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

nebo opačné číslo k součtu:

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

Rozdíl čísel:

```
>>> 5 - 2
3
```

Součin čísel:

```
>>> 2 * 3
6
```

Podíl:

```
>>> 8 // 2
4
```

Jedná se o celočíselný podíl, proto:

```
>>> 7 // 2
3
```

Zbytek po celočíselném podílu:

```
>>> 8 % 2
0
>>> 7 % 2
1
```

Příklad umocňování:

```
>>> 2 ** 3
8
```

Zvláštnost:

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

Zatím budeme používat pouze nezáporné mocnitele.

Při vyhodnocování výrazu se může vyvolat **výjimka**, která zastaví činnost interpretu. Interpret pak červeným textem informuje o typu výjimky. Například pokus o dělení nulou vyvolá výjimku. Vyzkoušíme:

```
>>> 5 // 0
ZeroDivisionError: integer division or modulo by zero
>>> 5 % 0
ZeroDivisionError: integer modulo by zero
```

4 Proměnné

K pojmenovávání hodnot slouží proměnné. Přesněji je **proměnná** pojmenovanou přihrádkou uchovávající hodnotu. Proměnné většinou pojmenováváme anglickými podstatnými jmény vystihující jejich význam. Například: `number`, `size` nebo `area`.

Poslední druh vstupu, který interpret může obdržet je **příkaz**. Interpret obdržený příkaz **vykoná**. Každý výraz je příkazem, který se vykoná tak, že se výraz vyhodnotí.

Pokud *`variable`* je jméno proměnné, *`value`* libovolná hodnota, pak

```
variable = value
```

je **příkaz nastavení hodnoty proměnné**. Příkaz interpret vykoná tak, že nastaví hodnotu proměnné *`variable`* na hodnotu *`value`*. Pokud by zadaná proměnná neexistovala, interpret nejprve vytvoří novou. Například:

```
>>> number = 1
>>>
```

vytvoří proměnnou `number` s hodnotou `1`.

Interpret si udržuje hodnoty proměnných ve formě **prostředí**. Prostředí znázorníme tabulkou se dvěma sloupci, kde v prvním sloupci budou jména všech proměnných a druhý sloupec udává jejich hodnoty. Například prostředí, kde proměnná `number` má hodnotu jedna, znázorníme tabulkou:

`number`	`1`

Vyhodnocování výrazů a vykonávání příkazů probíhá vždy vzhledem k aktuálnímu prostředí interpretu. Každá proměnná je výraz, který se vyhodnotí tak, že se získá hodnota proměnné z aktuálního prostředí. Například výraz `number` se ve výše uvedeném prostředí vyhodnotí na `1`. Tedy:

```
>>> number
1
```

Jelikož je proměnná výrazem, můžeme ji použít ve složeném výrazu:

```
>>> number + 1
2
```

Hodnotu proměnné můžeme změnit:

```
>>> number = 2
>>> number
2
```

Takto nyní vypadá aktuální prostředí:

`number`	2

Výraz `number + 1` má nyní jinou hodnotu:

```
>>> number + 1
3
```

Můžeme nastavit hodnotu proměnné na hodnotu složeného výrazu:

```
>>> number = 2 + 2
>>> number
4
```

Dokonce můžeme nastavit hodnotu proměnné na hodnotu výrazu, který obsahuje měněnou proměnnou:

```
>>> number = number + 1
>>> number
5
```

Zde se nejprve vyhodnotil výraz `number + 1`. Proměnná `number` měla hodnotu čtyři. Proto je hodnota výrazu pět. Poté se nastavila hodnota proměnné `number` na pětku.

Jména proměnných můžou obsahovat číslice, ale nesmí číslicí začínat. Například: `number1`, `area2`. Jméno proměnné se může skládat z více slov oddělených podtržítkem. Například `triangle_area`.

5 Podmínky

Zavedeme si typ **logických hodnot**, který obsahuje pouze dvě hodnoty: `True` a `False`. Jak již víme, jedná se o hodnoty a proto se vyhodnocují sami na sebe:

```
>>> True
True
>>> False
False
```

Hodnota `True` reprezentuje **pravdu** a hodnota `False` **nepravdu**. Pokud je hodnotou výrazu logická hodnota, nazýváme výraz **podmínkou**. Tedy `True` a `False` jsou podmínky. Přitom pokud je hodnota podmínky `True`, říkáme, že je podmínka **splněna**. Pokud je hodnota podmínky `False`, pak podmínka splněna není. Tedy `True` je vždy splněná podmínka a `False` nikdy nesplněná podmínka.

Představíme si binární operátory nazývané **komparátory** sloužící k porovnávání hodnot: `==` (rovnost), `!=` (nerovnost), `>` (větší než), `<` (menší než), `>=` (větší nebo rovno) a `<=` (menší nebo rovno). Pokud jsou argumenty komparátorů čísla, použije se na ně příslušná operace porovnávající čísla. Výsledkem operace bude pravda (hodnota `True`), pokud jsou čísla v zadaném vztahu, jinak nepravda (hodnota `False`). Například v prostředí:

`x`	`1`
`y`	`2`

je hodnota výrazu `(x + 1) == 2` rovna `True`. Tedy v tomto prostředí je podmínka `(x + 1) == 2` splněna.

6 Složené podmínky

Za použití unárního operátoru `not` a dvou binárních operátorů `and` a `or` můžeme podmínky skládat. Operandy operátorů musí být podmínky. Operace přiřazená operátoru `not` prohazuje logické hodnoty:

```
>>> not False
True
>>> not True
False
```

Vyhodnocování podmínek s operátorem `and` nebo `or` se řídí speciálními pravidly. Přesněji podmínka

```
(condition1 and condition2)
```

se vyhodnotí tak, že se nejprve vyhodnotí podmínka *condition1*. Pokud je podmínka pravdivá, pak se pokračuje ve vyhodnocování podmínky *condition2*, jejíž hodnota je výsledkem, jinak je výsledkem vyhodnocování `False`. Například hodnota podmínky

```
(0 == 1) and ((1 // 0) == 0)
```

je `False`, přestože vyhodnocení podmínky `((1 // 0) == 0)` by vedlo k vyvolání výjimky. Podobně podmínka

```
(condition1 or condition2)
```

se vyhodnotí tak, že se nejprve vyhodnotí podmínka *condition1*. Pokud je podmínka nepravdivá, pak se pokračuje ve vyhodnocování podmínky *condition2*, jejíž hodnota je výsledkem, jinak je výsledkem vyhodnocování `True`.

7 Priorita a asociativita operátorů

Pro možnost vynechávání závorek ve výrazech zavedeme prioritu a asociativitu skupin operátorů. **Priorita** udává přednost operátorů před jinými. Například operátor `*` má větší prioritu než operátor `+` a proto ve výrazu:

```
1 + (2 * 3)
```

můžeme závorky vynechat:

```
1 + 2 * 3
```

Následuje souhrn skupin operátorů seřazených podle priority od největší:

`**`
`-`*`expr`*
`*`, `//`, `%`
`+`, `-`
`<`, `<=`, `>`, `>=`, `!=`, `==`
`not`
`and`
`or`

Řádek `-`*`expr`* značí unární minus. Každý řádek určuje jednu skupinu. Operátory v každé skupině mají stejnou prioritu. Například ve výrazu:

```
((1 + 2) == 3) and (1 <= 2)
```

Můžeme všechny závorky vynechat:

```
1 + 2 == 3 and 1 <= 2
```

Pozor výraz:

```
-1 ** 2
```

znamená:

```
-(1 ** 2)
```

Asociativita skupiny operátorů určuje pořadí vyhodnocování operátorů ze skupiny. Asociativita je buď **zleva**, nebo **zprava**. Asociativita zleva znamená, že se nejprve vyhodnotí operátory napsané vlevo. Asociativita zprava naopak znamená, že se nejprve vyhodnotí operátory napsané vpravo. Obecně jsou operátory ve skupině asociativní zleva. Proto například:

```
1 + 2 - 3 - 2
```

je:

```
((1 + 2) - 3) - 2
```

Výjimku tvoří operátor umocňování `**`, který je asociativní zprava. Tedy:

```
2 ** 2 ** 3
```

je:

```
2 ** (2 ** 3)
```

Otázky a úkoly k semináři

1. Napište výraz, jehož hodnota je poslední cifra kladného čísla `x`. Například pro `x` rovno `123` je hodnota výrazu `3`.

2. Napište výraz, jehož hodnota je rovna kladnému číslu `x` bez poslední cifry. Například pro `x` rovno `123` je hodnota výrazu `12`.

3. Napište podmínku, která je splněna, právě když je číslo `x` sudé.

4. Napište podmínku, která je splněna, právě když číslo `x` dělí číslo `y`. Přitom `x` dělí `y`, jestliže existuje číslo `z` takové, že je podmínka `x * z == y` splněná. Vyhodnocování podmínky nesmí vyvolat výjimku v případě, že `x` je nula.

5. Uvažujme aktuální prostředí interpretu:

`x`	`1`
`y`	`2`

   Jak bude vypadat prostředí po vykonání následujících příkazů?

```
>>> x = y + 2
>>> z = x
>>> y = x - 2
```

6. V aktuálním prostředí interpretu je splněna podmínka `x == x0 and y == y0`. Jaké příkazy musíme vykonat, aby byla splněna podmínka `y == x0 and x == y0`? Proměnné `x0` a `y0` nesmíte v příkazech používat.

7. V aktuálním prostředí interpretu má proměnná `x` hodnotu tříciferné číslo. Vykonejte příkazy tak, aby byla splněna podmínka `x == c3 * 100 + c2 * 10 + c1`.