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

10. Desetinná čísla

Kromě celých čísel máme i desetinná čísla, které zadáváme (podle anglické konvence) pomocí desetinné tečky. Například:

```
>>> 0.1
0.1
```

Aritmetické operace můžeme provádět i s desetinnými čísly. Například:

```
>>> 0.1 + 0.1
0.2
```

Ve výpočtech můžeme kombinovat desetinné a celé číslo. Výsledek je pak vždy desetinné číslo. Například:

```
>>> 0.2 * 2
0.4
>>> 0.2 * 5
1.0
```

Hodnotu `1.0` považujeme za desetinné číslo.

Operátor `/` složí k výpočtu podílu. Výsledkem je vždy desetinné číslo.

```
>>> 1 / 2
0.5
>>> 4 / 2
2.0
```

Priorita a asociativita operátoru `/` je stejná jako u operátoru `//`.

Desetinné číslo dostaneme i v případě mocniny se záporným mocnitelem.

```
>>> 2 ** -3
0.125
```

Výpočty s desetinnými čísly jsou pouze přibližné. Například:

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Proto:

```
 >>> 0.1 + 0.1 + 0.1 == 0.3
False
```

Úkol 1 *Napište funkci* `my_abs`*, která vrátí absolutní hodnotu čísla. Například:*

```
>>>  my_abs(-0.2)
0.2
>>>  my_abs(0.2)
0.2
```

Úkol 2 *Napište funkci* `get_distanct`*, která spočítá vzdálenost dvou čísel. Například:*

```
>>> get_distanct(1, 0.9)
0.1
```

Při zadávání desetinných čísel lze použít i vědeckou notaci. Číslo ve tvaru $mantissa \times 10^{exponent}$ zapíšeme tak, že mezi mantisu a exponent vložíme písmeno `e`:

mantissa `e` *exponent*

Exponent vždy zadáváme i se znaménkem.

Použitím vědecké notace můžeme v kilogramech pohodlně vyjádřit jak nejmenší hmotnost atomu $1,67 \times 10^{-27}$ výrazem `1.67e-27` tak hmotnost sluce $1,9891 \times 10^{30}$ výrazem `1.9891e+30`.

Při tisku interpret používá vědeckou notaci pouze pro čísla s větší absolutní hodnotou exponentu.

```
>>> 1e-2
0.01
>>> 1.23e+2
123.0
>>> 10000000000000000.0
1e+16
>>> 0.00001
1e-05
```

Úkol 3 *Napište funkci* `are_similar`*, která pro čísla* `číslo1`*,* `číslo2` *a* `přesnost` *rozhodne, zda vzdálenost čísel* `číslo1` *a* `číslo2` *je menší než* `přesnost`*. Například:*

```
>>> are_similar(0.3, 0.1 + 0.1 + 0.1, 1e-4)
True
```

Úkol 4 *Zlatý řez je přibližně* 1.618033988749 . . . *Zlatý řez můžeme přibližně určit jako podíl dvou po sobě jdoucích Fibonacciho čísel. Přesněji pokud* `number1` *a* `number2` *jsou po sobě jdoucí Fibonacciho čísla, pak podíl* `number2 / number1` *je přibližně zlatý řez. Například:*

```
>>> 21 / 13
1.6153846153846154
```

Přitom čím pozdější Fibonacciho čísla vezmeme, tím přesnější odhad dostaneme. Máme tedy posloupnost odhadů zlatého řezu, která se stále zpřesňuje. Napište funkci `get_golden_ratio`*, která očekává číslo* `precision`*. Funkce vrátí takový odhad zlatého řezu, který je přibližně stejný jako předchozí odhad (s použitím přesnosti* `precision`*). Například:*

```
>>>  get_golden_ratio(1e-5)
1.618032786885246
```