# ZAL – 3. cvičení

2016

# Funkce - opakování

- Funkce může přijímat parametry na vstupu a může vracet parametry na výstupu.
- Délka funkce by měla být „rozumná“. Tipněte si co je rozumná délka funkce.
- V Pythonu se proměnné do funkce přenášejí odkazem (referencí) nikoliv hodnotou. Víte jaký je mezi tím rozdíl?
- V Pythonu je možné, že funkce vrací více jak jednu proměnnou – nezvykejte si na to ostatní jazyky jsou mnohem striktnější

# Návrh dobré funkce

- Pojďme si nyní navrhnout funkci, která vyřeší lineární rovnici o jedné neznáme.

- Napište funkci, která vyřeší jednoduchou lineární rovnici typu: ax + b = c, předpokládejte, že proměnné a, b, c budou zadány. Funkce tedy vypočte hodnotu jedné neznáme x.

- Tip: Zamyslete se jak se výpočet provádí a jaké vlastnosti by měla funkce mít.

# Lineární rovnice

- Jak může probíhat obecný výpočet?

- 12x + 10 = 23
- 12x - 13 = 0
- 12x = 13
- x= 1,08….

- ```
  def linearVeryNaiveEquationSolver(a,b,c):
      b = b-c
      a = a-a-a
      x = b/a
      return x
  ```
  ```

# Lineární rovnice

- **def** linearVeryNaiveEquationSolver(a,b,c):
  b = b-c
  a = a-a-a #Ehm  - nebylo by lepší řešení?
  x = b/a
  **return** x

Bylo:

- 12x + 10 = 23
- 12x = 13
- x= 1,08….

# Lineární rovnice

Bylo:

- 12x + 10 = 23
- 12x = 13
- x= 1,08….

```
def linearNaiveEquationSolver(a,b,c):
    rightSide = (c-b)
    x = rightSide /a
    return x
```

Opravdu to funguje? Vidíte nějaké možné problémy?

# Lineární rovnice

- Neměly bychom něco kontrolovat?
- Mějme rovnici: 0x -10 = 10
  - Jaký je výsledek?
  - Co se stane s funkcí, kterou jsme napsali?
- Výpočet spadne na dělení nulou.
- Toto zjištění implikuje potřebu řídit program nebo algoritmus, ale jak?

# Řízení programu

- Řízení programu nám nabízí možnosti jakými určovat větve algoritmu či programu, které budou zpracovány. Program můžeme řídit na základě jeho aktuálního stavu. Program tedy přehází mezi stavy.

- K tomuto účelu existuje mnoho mechanismů. Nejzákladnějším a nejběžnějším jsou podmínky.

- Podmínka je logická funkce. Pokud je podmínka splněná vykoná se program v jejím těle.

# Podmínky

- Základní podmínka je if (v překladu když)

- Podmínka if má obecně následující syntax:

```
if<výraz>:
        blok kódu, který se splní je-li výraz
        vyhodnocen jako pravdivý
pokračování programu
```

ifexample.py

# Podmínky - pokračování

- Podmínka může mít několik dalších vlastností. Výraz může být složenina z dvou výrazů:

```
if number >2 and number<4:
```

- Podmínka může mít další vyhodnocovací podmínky nebo-li else větve. Tyto else větve se vykonávají pokud není splněna první podmínka. Viz příklad:

ifelseexample.py

# Lineární rovnice

- Zpátky k naší rovnici. Měli jsme problém s dělením nulou? Co s tím? Ošetříme vstupy?

- **def** linearNaiveEquationSolver(a,b,c):
  rightSide = (c-b)
  x = rightSide /a
  **return** x

- Doplňte kontrolu tak abychom již neměli chybu, kterou způsobuje dělení 0.

# Lineární rovnice

- ```
  def linearSmarterEquationSolver(a,b,c):
      rightSide = (c-b)
      if(a == 0):
          return 0
      x = rightSide /a
      return x
  ```

- Chyba na dělení nulou již není, ale je výsledek těchto rovnic opravdu 0?
- 1) 0x + 10 = 10
- 2) 0x – 10 = 10 – tato instance nemá řešení co s tím?
  ```

# Řízení běhu programu - výjimky

- S výjimkou (exception) jste již mohli setkat – například pokud dělíte nulou. Co to výjimka je?
- Výjimka je mechanismus s jakým se vyšší programovací jazyky vypořádají s chybou, která by mohla vést k pádu systému.
- Výjimka je obvykle vyhozena při operaci a pokud je tato výjimka vyhozena tak se blok v kterém byla vyhozena již dále neprovádí.

# Výjímka (Exception)

- Naštěstí pro nás, lze výjimku očekávat ☺ a zpracovat ji a tím zabránit úplnému pádu programu, který by mohl způsobit, že by zdroje s kterými pracujeme zůstali v nekonzistentním stavu.

- Obvykle se používá klíčové slovo try

  try: #urcuje blok v kterem ocekavame funkci
  except EXCEPTIONTYPE as e: # zde odchytime
  finally: #operace se provedou vzdy

ifelseexceptionexample.py

# Lineární rovnice

- ```
  def linearSmarterEquationSolver(a,b,c):
      rightSide = (c-b)
      if(a == 0):
          return 0
      x = rightSide /a
      return x
  ```

- Doplňte tuto funkci tak, aby v případě, že nemá instance řešení vyhodila VallueError
  ```

# Lineární rovnice

- ```
  def linearMuchMoreSmarterEquationSolver(a,b,c):
      rightSide = (c-b)
      if(a == 0):
          raise ValueError('The instance has no solution in real numbers!')
      x = rightSide /a
      return x
  ```
- Funkce již produkuje výjimky v případě, že instance nemá řešení. Co ale následující příklady?
- 0x - 10 = 23
- 0x -10 = 10 – výsledkem by neměla být exception
  ```

# Lineární rovnice

- ```
  def linearEquationSoler(a,b,c):
      rightSide = (c-b)
      if(a == 0 and rightSide != 0):
          raise ValueError('The instance has no solution in real numbers!')
      else:
          return 0
      x = rightSide /a
      return x
  ```

- Je to již OK?
  ```

# Návrh dobré funkce

- Název linearSolver, linearFunction linearEquationSolver nebo jiný?
- Vstupní parametry – očekáváme tři
- Výstup z funkce – číslo (nalezené řešení rovnice) – nebo výjimka.
- Ošetření vstupních parametrů.
- Řízení na základě mezi výpočtů
- Přeci jen jsme na něco zapomněli – komentáře. Komentujte svůj kód prosím!

# Lineární rovnice

- Vyzkoušejte si: Chceme, aby řešení bylo pouze celočíselné.
- Jak to změní algoritmus?
- Lze to?

# Zadání třetího domácího úkolu

- Celé detailní zadání je zde: https://cw.fel.cvut.cz/wiki/courses/b6b36zal/zadani/3_calculator
- Body: 2
- Termín do dalšího cvičení – po termínu -2b