IB113 Úvod do programování a algoritmizace

Cvičení 2

Jaromír Plhák

Želvy - opakování

```
from turtle import Turtle, done
barbara = Turtle()
barbara.reset()
…
barbara.penup()
barbara.pendown()
…
barbara.forward(100)
barbara.backward(30)
barbara.right(120)
…
done()
```

```
def centipede(n):
    pass

centipede(7)
```

Proměnné

- Místo v paměti, kde je uložená hodnota pod nějakým jménem
  - Smysluplné a výstižné pojmenování[1], ne my_super_variable
  - Obvykle malé písmena, čísla a podtržítka mezi slovy
  - Ne rezervovaná slová
  - Používejte angličtinu pro pojmenovávání proměnných
- Přiřazovací příkaz: proměnná = hodnota
- Porovnání: proměnná == hodnota

- [1] http://www.hovnokod.cz/450

Datový typ

- Množina hodnot spolu s operacemi na těchto hodnotách
- Celé číslo (int): 42, 0, -14
- Desetinné číslo (float): 42.0, 0.21, -7.59876, 3e8
- Logická hodnota (bool): True, False
- Řetězec (string): "The answer is 42!", '0.21'
- Seznam (list): [1, 4, 9, 16, 25]
- …
- Python: implicitní typovaní (typ se k proměnné nepíše)

Datový typ

- Zjištění typu: funkce type()
- **>>> type (1)**
- **<class 'int '>**
- **>>> type (" Hello ")**
- **<class 'str'>**
- Přetypování: funkce int(), float(), str()
- **>>> str (1)**
- **'1'**

Operátory

- Jazyková konstrukce podobná funkci, která umožňuje operace na hodnotách
- Jeho argument: operand
- Operátory přiřazení: =, +=, -=, *=, /=, %=, **=
- Aritmetické operátory: +, -, *, /, //, %, **
- Relační operátory: ==, !=, <, >, <=, >=
- Logické operátory: and, or, not
- Bitové operátory: <<, >>, &, |, ^, ~
- Jiné: [], ., is, is not, in, not in

Relační operátory

- Úloha: jaké hodnoty budou v proměnných a až f?

```
>>> a = 4 == 4
>>> b = 8 != 5
>>> c = 9 < 4
>>> d = 9 <= 4
>>> e = 5 > 3
>>> f = 5 >= 5
```

```
>>> a = True
>>> b = True
>>> c = False
>>> d = False
>>> e = True
>>> f = True
```

- Oddělujte operátory mezerou na obou stranách

Logické operátory

- Úloha: jaké hodnoty budou v proměnných a až i?

```
>>> a = True and True
>>> b = True and False
>>> c = False and False
>>> d = True or True
>>> e = True or False
>>> f = False or False
>>> g = not True
>>> h = not False
>>> i = not not True
```

```
>>> a = True
>>> b = False
>>> c = False
>>> d = True
>>> e = True
>>> f = False
>>> g = False
>>> h = True
>>> i = True
```

Výraz

- Kombinace konstant, proměnných, operátorů a funkcí
- Nejprve aritmetické, potom relační, nakonec logické

>>> 5 + 16 >= 5 + 4*2

>>> 5 + 16 >= 5 + 8

>>> 21 >= 5 + 8

>>> 21 >= 13

>>> True

- Při pochybnostech použijte závorky

Příkaz print

- Co se vypíše?

```
>>> a = 1
>>> b = 2
>>> print (a + b)
>>> print (a, b)

>>> a = "1"
>>> b = "2"
>>> print (a + b)
>>> print (a, b)
```

Příkaz print

- Pozor na operace s nekompatibilnými typy:

```
>>> a = 1
>>> b = "2"
>>> print (a + b)
TypeError : unsupported operand type (s) for +: 'int' and 'str'
```

- Přetypování:

```
>>> print (a + int (b))
3
```

Základní konstrukce – podmíněný příkaz

if <podmínka>:

    příkaz1

else:

    příkaz2

- Podle toho, zda platí podmínka, se provede jedna z větví
- Podmínka – typicky výraz nad proměnnými
- else větev nepovinná
- Vícenásobné větvení: if - elif - ... - else

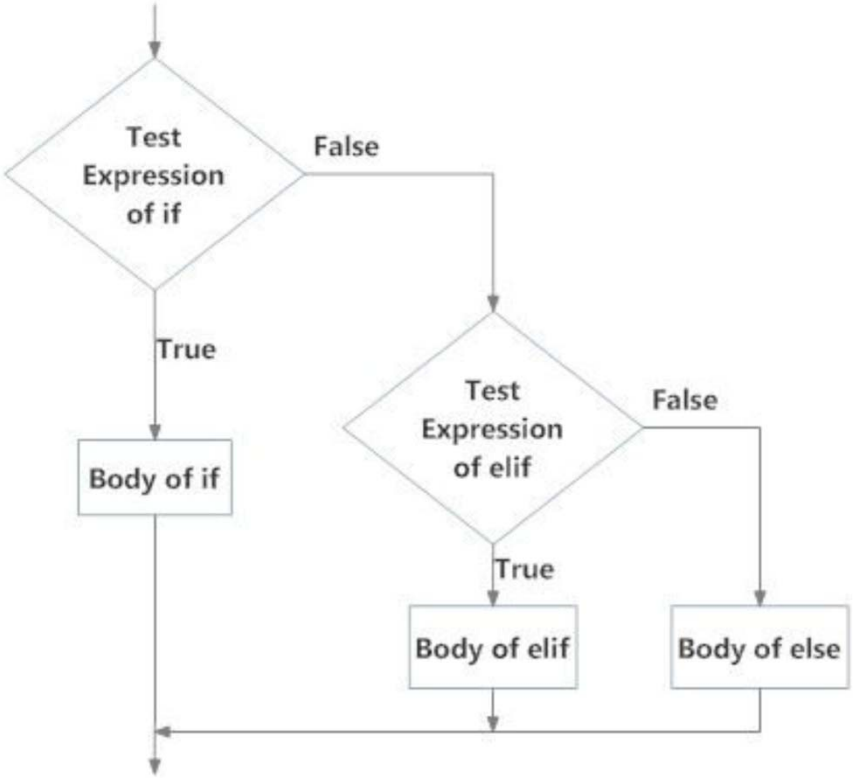

Fig: Operation of if...elif...else statement

Zdroj: http://www.programiz.com/python-programming/if-elif-else

Základní konstrukce – Ternární podmínka

- Mějme kód:

```
max = 0
if a > b:
        max = a
else :
        max = b
```

- Ekvivalentní zápis:

```
max = a if (a > b) else b
```

- Čteme: „Nechť max je a pokud a je větší než b, jinak nechť max je b.“

Základní konstrukce – cyklus for

- Opakované provádění sekvence příkazů
- Známý počet opakování cyklu
- Příkaz for

```
for i in range(3):
    print(i)
```

- Proměnná *i* postupně nabývá celočíselné hodnoty z rozsahu [0, 3)
- range(n) – interval od 0 do n - 1 (tj. n opakování)

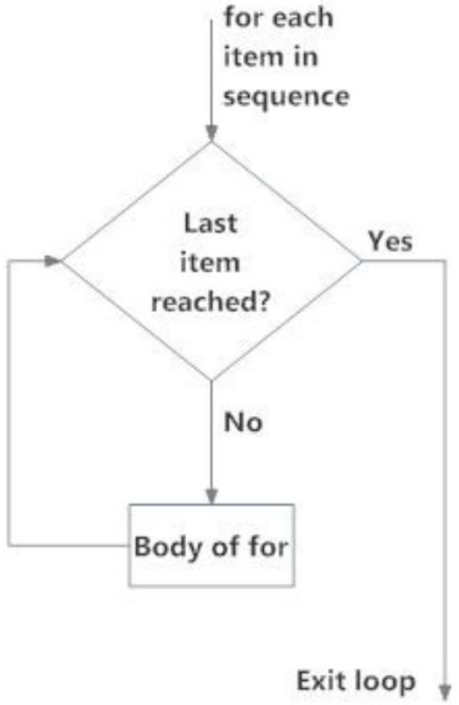

Fig: operation of for loop

Zdroj: http://www.programiz.com/python-programming/for-loop

Základní konstrukce – cyklus while

- Neznámý počet opakování cyklu
- Příkaz while
- Opakuj dokud není splněna podmínka

```
while <podmínka>:
        příkazy
```

```
choice = input ("Like the course ? (y/n)")

while choice != "y" and choice != "n":
        choice = input ("Sorry , I didn 't catch that . Enter again : ")
```

Procedury a funkce - motivace

- Znovupoužití kódu (s různými hodnotami):

```
def triangle (length):
    for i in range (3) :
        forward (length)
        right (120)
triangle (100)
```

- Jako refrén v písničce (s malými obměnami)

Procedury a funkce - definice

- Klíčové slovo def
- Mezera
- Jméno funkce
- Seznam parametrů v závorkách
  - Žádné parametry - prázdné závorky
- Dvojtečka
- Nový řádek
- Odsazení příkazu funkce

```
def my_funct (par1 , par2 , ...) : # header
        statement_1 # body
        ...
        statement_n
```

Procedury vs. funkce

- Funkce: vrátí hodnotu, může být použita ve výrazu
  - Poslední příkaz musí být return hodnota
- Procedura: jen sekvence příkazů, nevrací hodnotu
  - Python: procedura je speciální případ funkce, která obsahuje implicitní příkaz return None
  - Funkce, která „vrátí nic“, prázdný objekt
- Dále budeme označovat jako **funkce**

Příklad funkce

- Funkce by měla mít smysluplné a výstižné jméno
  - Ne my_super_function()
- Je dobrým zvykem komentovat chovaní funkce
  - Tzv. *docstring*

```
def hello ():
        """ Prints 'Hello !' to console . """
        print (" Hello !")
```

- Funkční volání: použití jména funkce a zadaní všech potřebných parametrů
  - Způsobí vykonaní kódu ve funkci

```
>>> hello () # No parameters
Hello !
```

Parametry funkcí

- Formální parametry: vstupy funkce v její definici

```
def power (base , exponent ):
        result = base ** exponent
        print (base, "to the power of", exponent, "is", result)
```

- Skutečné parametry (argumenty): hodnoty, které volající příkaz posílá funkci
- Formální parametry jsou při volání nahrazeny skutečnými

```
>>> power (3, 4)
# base initializes to 3, exponent to 4
3 to the power of 4 is 81.
```

Předávání skutečných parametrů

- Pokud je argument funkce neměnitelného typu (int, float, bool, str, tuple), jeho hodnota se zkopíruje do funkce
  - Předávání parametrů hodnotou (*pass by value*)
- Pokud je argument funkce měnitelného typu (list, dict, ...), sdílí se s funkcí a může v ní být změněný (tzv. vedlejší efekt)
  - Předávaní parametrů odkazem (*pass by reference*)
- Python: všechny proměnné jsou objekty, na které odevzdáváme odkaz hodnotou
  - Předávání parametrů přiřazením (*pass by assignment*)

Formátování kódu

- Smysluplná funkce[2]
- Vhodné jméno funkce[3]
- Vhodné komentáře (pokud jsou potřebné)[4]
- Krátké funkce (má jeden účel)[5]

[2]http://www.hovnokod.cz/367
[3]http://www.hovnokod.cz/392
[4]http://www.hovnokod.cz/401
[5]http://www.hovnokod.cz/353

Formátování kódu

- Dlouhou funkci rozdelte na menší[6]
- Nezanořujeme se příliš[7]
- Maximálně 3 úrovně (podmínek a cyklů)
- Pokud chcete kopírovat kód, vytvořte pro něj funkci[8]
- Použijte vlastní odhad: pokud si při čtení kódu chcete vypichnout oči[9] a skočit z okna[10], je pravděpodobně něco v nepořádku

[6]http://www.hovnokod.cz/441
[7]http://www.hovnokod.cz/1429
[8]http://www.hovnokod.cz/461
[9]http://www.hovnokod.cz/501
[10]http://www.hovnokod.cz/470

Pep 8

- Style Guide for Python Code
  - Oficiální doporučení
  - https://www.python.org/dev/peps/pep-0008/
- Nástroje: pep8, autopep8, flake8 (příkazový řádek)
- Důležité body:
  - Zarovnávání kódu; max. délka řádku 79 znaků; dva prázdné řádky mezi funkcemi
  - Mezery kolem operátorů; žádné mezery za otevírací (před uzavírací) závorkou
    - Výjimka: žádné mezery kolem = u parametrů funkcí
  - Pojmenování věcí proměnné, funkce: lower_case_with_underscores
  - Konstanty: UPPER_CASE_WITH_UNDERSCORES
  - Třídy (později): CamelCase

Úkoly

- http://www.fi.muni.cz/IB111/sbirka/02-zakladni_struktury.html
- 2.1.1. Sudá čísla
- 2.1.2. Mocniny
- 2.1.3. Dělitelé
- 2.3.1. Vyplněný čtverec
- 2.3.2. Prázdný čtverec