Řízení běhu programu

Podmínky

Příkaz if

Podmínky patří mezi základní struktury programovacího jazyka. Obecně rozlišujeme:

- jednoduchou podmínku *if … then,*
- alternativní podmínku *if … then … else,*
- mnohonásobnou – výběr *case či switch*.

V jazyce Python je jediná varianta **`if,`** která však v sobě zahrnuje všechny tři druhy podmínek.

Příkaz **`if`** začíná podmínkou (*„jestliže*“), případně následuje jedna nebo více částí **`elif`** (*„jestliže ne, potom zkus, jestli“*) a na závěr můžeme zařadit část **`else`** *(„jestliže ne, tak“).*

Když je `if` spuštěn, vyhodnotí se nejprve první (`if`) podmínka. Je-li splněna, provede se patřičný vnořený blok a `if` se ukončí. Jinak se postupně prochází jednotlivé `elif` podmínky, a není-li splněná žádná z nich, provede se (vnořený blok kódu části) `elif`. Obecně vypadá `if` takto:

```
if <podmínka1>:                 # if podmínka
   <příkazy1>
elif <podmínka2>                # jedna nebo více nepovinných částí elif
   <příkazy2>
else:                           # nepovinná část else
   <příkazy3>
```

Tab. 1 – Schéma podmínky if

Příklad podmínky:

```
>>> D=2
>>> if D>0:
    print "rovnice ma dve reseni"
elif D==0:
   print "rovnice ma jedno reseni"
else:
   print "rovnice nema zadne reseni v oboru realnych cisel"

rovnice ma dve reseni
```

Možnosti realizace vícenásobné podmínky

Speciální konstrukce pro vícenásobnou podmínku (typu *case* či *switch*) v Pythonu není. Avšak samotná stavba `if` je vlastně realizací mnohonásobné podmínky. Výběr lze však výhodně realizovat i jinak. S výhodou lze užít seznamů či slovníků. Nejprve bude výběr realizován příkazem `if:`

```
>>> vyber = "sunka"
>>> if vyber == "dzem":
   print 6.00
```

```
elif vyber == "sunka":
   print 19.00
elif vyber == "vejce":
   print 15.00
elif vyber == "slanina":
   print 21.00
else:
   print "spatny vyber"
```

```
19.00
```

Nyní následuje totéž pomocí slovníku:

```
>>> vyber = "sunka"
>>> print {"dzem":       6.00,
           "sunka":      19.00,
           "vejce":      15.00,
           "slanina":    21.00} [vyber]
19.00
```

Cykly

Pojem cyklu

Cyklus (*loop*) je opakování části programu. Jednotlivé průchody cyklem se nazývají **iterace.** Počet průchodů cyklem (iterací) je řízen podmínkou nebo je před vstupem do cyklu předem dán. Obvykle programovací jazyky obsahují:

- **Cyklus řízený podmínkou,** v tomto případě zpravidla existují cykly:
  - **s podmínkou na začátku – *while***. Podmínka je na začátku za slovem while, pak následuje jádro, do kterého řízení programu vstoupí, když je podmínka splněna, po provedení jádra se znovu vyhodnocuje podmínka atd. Jádro cyklu nemusí být provedeno ani jednou (je-li podmínka nesplněna hned na začátku), může být provedeno jednou či n-krát. Pokud je podmínka cyklu chybná, může dojít k zacyklení, tj. nekonečnému provádění jádra.
  - **s podmínkou na konci – *until*.** Rozdíl je ten, že jádro se provede vždy alespoň jednou.
- **Cyklus s předepsaným počtem opakování – *for*.** Před vstupem do jádra je předem znám počet iterací. V některých jazycích existují ještě trochu odlišné varianty cyklu for.

Python obsahuje cyklus s podmínkou na začátku **`while`** a cyklus s předepsaným počtem opakování **`for`.**

Cyklus while

Python zná jediný cyklus s podmínkou a to cyklus `while`. Je obecný tvar je:

```
while <podmínka>                    # uvozující řádek s podmínkou
      <příkazy>                     # jádro cyklu
else:                               # nepovinná část else
      <příkazy>
```

Tab. 2 – Cyklus s podmínkou na začátku while

Do jádra cyklu vstoupí řízení, je-li podmínka splněna, není-li, provedou se příkazy za (nepovinnou) částí `else`.

V cyklu **while** v Pythonu:

- **break** umožní ukončit provádění cyklu. Program pokračuje za složeným příkazem tohoto cyklu
- **continue** předá řízení na začátek (na podmínku) nejbližšího vnějšího cyklu
- **pass** nedělá vůbec nic pouze zabírá místo (viz dále)
- část **else** se provede pouze v případě, že cyklus nebyl ukončen příkazem `break`.

Break a **continue** se vztahují pouze na nejvnitřnější cyklus, nelze rovnou vyskočit ze všech uzavřených cyklů, protože Python nemá žádný příkaz *goto*. Python patří důsledně k tzv. „*goto-less programing*“ (tedy k programovacím jazykům bez goto). Ukončit provádění všech cyklů lze provést výjimkou.

Rozšířený obecný popis cyklu **while**:

```
while <podmínka1>                    # break a continue kdekoli
      if <podmínka2>: break          # ve vnitřním bloku; break – konec
       if <podmínka3>: continue      # continue – na uvozující řádek
            <příkazy1>               # jádro cyklu
else:                                          # else – pokud bez break
            <příkazy2>
```

Tab. 3 – Rozšířený popis příkazu cyklu while

Příklad – postupné odřezávání prvního znaku řetězce:

```
>>> x = "potraviny"
>>> while x:
      print x,
      x = x[1:]

potraviny otraviny traviny raviny aviny viny iny ny y
```

Příklad – použití nekonečného prázdného cyklu.

```
>>> while 1: pass   # stiskem Ctrl-C ukončíme interpret
KeyboardInterrupt
```

Příklad – výpis sudých čísel (pozn. ne právě elegantní řešení – zde na ukázku možnosti continue)

```
>>> x = 20
>>> while x:
    x = x-1
    if x % 2 !=0: continue    #liche cislo nevypisovat
    print x,

18 16 14 12 10 8 6 4 2 0
```

Příklad – test, je-li číslo prvočíslem

```
>>> y = 121
>>> x = y / 2
>>> while x > 1:
    if y % x == 0:
        print y, "je delitelne", x
        break                          # preskocime else
    x = x - 1
else:                                  # "normalni" konec
    print y, "je prvocislo"

121 je delitelne 11
```

Cyklus for

Cyklus **for** se užívá pro cykly s předepsaným počtem opakování. Rozšířený obecný popis cyklu for:

```
for <cíl> in <procházený objekt>
     if <podmínka2>: break        # ve vnitřním bloku; break – konec
      if <podmínka3>: continue          # continue – na uvozující řádek
                 <příkazy1>       # jádro cyklu
else:                                              # else – pokud bez break
            <příkazy2>
```

Tab. 4 – Cyklus s předepsaným počtem opakování for

Příklad – sčítáme prvky seznamu:

```
>>> sum = 0
>>> for x in [2, 5, -3, 0]:
    sum += x

>>> sum
4
```

Složitější příklad – hledání v seznamu. V seznamu hledáme prvky (hledane).

```
>>> data = ["aaa", 111, (4, 5), 2.01]
>>> hledane - [(4, 5), 3.14]
>>> for hledany in hledane:
        for aktualni in data:
            if aktualni == hledany:
                print hledany, "byl nalezen"
                break
        else:
            print hledany, "nebyl nalezen"


(4, 5) byl nalezen
3.14 nebyl nalezen
```

Pozn.: zde je nutné správné začlenění else (ne pod první for ani ne pod if)!

range a for

Standardní funkce **range** umožní používat cyklus for podobně jako v jiných jazycích pro přesně celočíselně daný počet opakování. Range vrací seznam naplněný čísly. Jeden argument vrací čísla od nuly do argument - 1, dva argumenty vrací čísla od argument1 do argument2 - 1, je-li třetí argument, je to hodnota kroku:

```
>>>range(9)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> range(3,7)
[3, 4, 5, 6]
>>> range(20, 0, -4)
[20, 16, 12, 8, 4]
```

Příklad – malá násobilka pěti:

```
>>> for i in range(1,11):
    print "%d x 5 = %d" % (i, i*5)

1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50
```

Literatura:

[1] Rubeš, J.: Nebojte se programovat, ComputerMedia, Bedihošť 2001

[2] Lutz, M., Ascher, D.: Naučte se Python, Grada, Praha 2003

[3] Beazley, D. M.: Python, Neocortex, Praha 2002

[4] Python Reference Manual

[5] Švec, J.: Létající cirkus, Python tutoriál, 2003

[6] Harms, D., McDonald K.: Začínáme programovat v jazyce Python, Computer Press, Brno 2003