Cykly.

Cyklus for. Cyklus while. Break. Continue.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

Obsah přednášky

1. Příkazy pro opakování

Příkazy pro opakování nazývány iteračními příkazy.
Umožňují provádět příkaz/blok za splnění podmínky.

V praxi nejčastěji realizovány prostřednictvím cyklů.
Vykytují se ve většině programovacích jazycích.

Alternativou k cyklům rekurze.

Řídící podmínka cyklu:
Představována booleovským výrazem.
Vyhodnocení: hodnota `true` nebo `false`.

Opakování prováděno v těle cyklu (blok, příkaz).

Dle vstupní podmínky cykly děleny:

- s předem neznámým počtem opakování: `while`,
- s předem známým počtem opakování: `for`, `for-each`,
- s neznámým počtem opakování, alespoň 1x proběhne: `do while`.

2. Cyklus while

Používán, pokud počet opakování není předem znám.
Vyhodnocení řídící podmínky *před* průchodem cyklu.

Pokud nesplněna, tělo cyklu neproběhne.

Dokud splněna, opakovaně prováděno tělo cyklu.

```
while vyraz:          # Ridici podminka cyklu
   telo cyklu         # Pokud rici podminka true
else:
   telo podminky      # Ridici podminka false
```

V Pythonu specifická syntaxe, blok `else`: pokud řídící podmínka `false`.
Ve většině jazyků podobná konstrukce chybí.

Vnořený cyklus:
V těle cyklu další cyklus

```
while vyraz:          # Ridici podminka cyklu
   while vyraz:
      telo cyklu
```

Nekonečný cyklus:
V těle cyklu nutno modifikovat řídící proměnnou, jinak nekonečný cyklus.
Pozor na tautologie a kontradikce!

3. Ukázka cyklu while: faktoriál

Definice faktoriálu

$$n! = \begin{cases} n(n-1)!, & \text{pro } n > 1, \\ 1, & \text{pro } n = 1. \end{cases}$$

Ukázka řešení v jazyce Python:

```
n = 10              #Initialize n
f = 1               #Initialize factorial
while n > 1:        #Repeat, while n > 1
      f *= n        #Actualize factorial
      n -= 1        #Decrement n
print(f);           #Print result, factorial.

>>> 3628800
```

4. Cyklus for

Používán, pokud je znám počet opakování.

Vyhodnocení řídící podmínky před každým provedením těla cyklu.
Tělo se prováděno dokud je pravdivá.
Pokud není pravdivý, skok do `else` bloku.

Syntaxe v Pythonu odlišná od jiných programovacích jazyků (`foreach` cyklus)

```
for prvek in iterovatelny objekt:
      telo_cyklu
else:
   telo podminky;
```

Iterace nad prvky datových struktury: [], {}, () či intervalem.

Iterovatelný objekt:
Objekt (dynamická datová struktura), můžeme přistupovat k jeho prvkům.
Sekvenční procházení jeho prvků, přímý přístup.

Alternativně procházení prvků v nějakém intervalu, příkaz `range()`.

Vnořený cyklus:
V těle cyklu další cyklus

```
for prvek in in iterovatelny objekt:          # Ridici podminka cyklu
   for prvek in in iterovatelny objekt:
      telo cyklu
```

5. For v jiných programovacích jazycích

V jazycích C/C++/Java/C# trochu jiná funkcionalita.
Chybí `else` blok.

Obecná syntaxe:

```
for (inicializacni_vyraz; testovaci_vyraz; zmenovy_vyraz)
{
      telo_cyklu
}
```

Inicializační výraz:
Je vykonán 1x před vyhodnocením testovacího výrazu

Testovací výraz:
Určuje, zda se má provést tělo cyklu.
Prováděno, dokud hodnota testovacího výrazu `true`.

Změnový výraz:
Vyhodnocen na konci cyklu po vykonání těla cyklu.
Použit pro změnu hodnoty testovacího výrazu.

Ukázka:

```
for (int i = 0; i < n; i++)
```

6. Ukázka cyklu for: hledání maxima, minima

Vstup $X = \{x_i\}_{i=1}^{n}$.
Hledáme

$$\underline{x} = \min_{x_i \in X}(x_i), \qquad \overline{x} = \max_{x_i \in X}(x_i).$$

Inicializace odhadů $\underset{\sim}{x} = x_1$, $\widetilde{x} = x_1$ prvním prvkem, alternativně $\underset{\sim}{x} = \infty$, $\widetilde{x} = -\infty$.
Postupné porovnání $\underset{\sim}{x}$, $\widetilde{x}$ s x_i

$$\underset{\sim}{x} = \min(\underset{\sim}{x}, x_i), \qquad \widetilde{x} = \max(\widetilde{x}, x_i).$$

Pro průchodu celou posloupností

$$\underline{x} = \underset{\sim}{x}, \qquad \overline{x} = \widetilde{x}.$$

Ukázka řešení v jazyce Python:

```
X = [10, -27, 3, 48, 0, 95]                  #Create list
xmin = X[0]                                  #Initialize minimum
xmax = X[0]                                  #Initialize maximum
for x in X:                                  #Repeat for ∀x ∈ X
    if x < xmin:                             #Compare x[i] to xmin
        xmin = x                             #Actualize xmin
    if x > xmax:                             #Compare x[i] to xmax
        xmax = x                             #Actualize xmax
print(xmax)                                  #Print results
print(xmim)
>>> 95,
>>> -27
```

7. Ukázka cyklu for: bankomat

Vstup: částka c, $c \in \mathbb{N}$.
Výstup: vyplacení co nejmenším počtem bankovek/mincí

$$B = \{1, 2, 5, \ldots, 2000, 5000\}.$$

Použití celočíselného dělení $//$ a operace $(\mathrm{mod}\ )$.
Počet bankovek

$$n = c \,//\, b.$$

Zbytek částky k vyplacení

$$c \equiv c \pmod{b} = c - nb.$$

Postupně opakujeme poslední dva kroky.

Ukázka řešení v jazyce Python:

```
B=[5000,2000,1000,500,200,100,50,20,10,5,2,1]
c = int(input("Zadej castku: "))
for b in B:                              #Opakuj pro kazdou bankovku/minci
    n = c // b                           #Urci pocet bankovek
    c = c%b                              #Zbytkova castka: c = c - n * b
    if c > 0:                            #Pokud jeste neco zbyva
        print(str(n) + " x " + str(b)) #Vytiskni pocet bankovek
.
>>> Zadej castku: 2345
>>> 1 x 2000
>>> 1 x 200
>>> 1 x 100
>>> 2 x 20
>>> 1 x 5
```

8. List/Set/Dir comprehension

Využití `for` při konstrukci nových dynamických struktur z existujících.
Speciální (úspornější) notace, umožňuje efektivní konstrukci.
Změnový výraz představuje libovolný Pythonovský výraz.

List comprehension:
Vytvoření nového seznamu z existujícího (nemění se).

```
L =[1, 2, 3, 4, 5]
L2 = [10 * l for l in L]
>>> [10, 20, 30, 40, 50]
```

Set comprehension:
Vytvoření nového setu z existujícího (nemění se).

```
S ={1, 2, 3, 4, 5}
S2 = {10 * s for s in S}
>>> {10, 20, 30, 40, 50}
```

Directory comprehension:
Vytvoření nového seznamu z původního (nemění se).
Lze modifikovat key i value.

```
D ={'a':1, 'b':2, 'c':3, 'd':4, 'e':5}
D2 = {key:10 * value for key, value in D.items()}
>>> {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
```

9. Příkaz break()

Okamžité (předčasné) ukončení provádění těla cyklu.
Provedeno, pokud podmínka vyhodnocena jako pravdivá.

```
if vyraz:
   break;
```

Pokračování prvním příkazem následujícím za cyklem.
U vnořeného cyklu skok o 1 úroveň výše.

Používán, pokud byla splněna nějaká, a není třeba pokračovat.
Tuto podmínku nelze otestovat před průchodem těla cyklu.

Příklad: nalezení prvku `val` v množině `X`:

```
found = False          #Test, whether x in X
for x in X:            #Test x one by one
   if (x == val)       #Element val found in X
       found = true
       break
```

Výhoda použití:
Zjednodušení konstrukce řídící podmínky cyklu.
Část podmínky nemusí být v řídícím výrazu cyklu.

10. Příkaz continue()

Přeskočení zbývající části příkazů v těle cyklu.
Provede přechod na další iteraci, pokud podmínka pravdivá.

```
if vyraz:
   continue;
```

Příklad, třídící algoritmus Merge Sort:

```
while i < n + m):
      if i == n :       #Seznam a uz nema dalsi prvky
         c[k] = b[j]
         j = j+1
         continue       #Skok na dalsi iteraci

      if (j == m):      #Seznam b uz nema dalsi prvky
         c[k] = a[i];
         i = i + 1
         continue       #Skok na dalsi iteraci
```

V praxi se příliš často nepoužívá, znepřehledňuje tělo cyklu.

Nekombinovat více příkazů `continue` v jednom těle (max 2), nepřehledné.

Skok v algoritmu, chyba v návrhu!

11. Příkaz range()

Vytváří celočíselnou aritmetickou posloupnost (a_n), když $\exists d \in \mathbb{N}, \forall d \in \mathbb{N}$ a platí

$$a_n = a_1 + (n-1)d,$$

d je diference posloupnosti.

`range(`a_1`,` a_n`,` d`);` $\qquad a_1 + kd < a_n, \qquad d \in \mathbb{N}.$

Ukázka:

```
X=range(1, 20, 2)
for x in X:
    print(x)
>>> 1 3 5 7 9 11 13 15 17 19
```

Často kombinován s cyklem `for`, tvorba celočíselného indexu

`range(`i_1`,` i_n`,` 1`);`

Generuje posloupnost indexů $[i]$

$$(i_1, i_1 + 1, ..., i_1 + n - 1).$$

Přes a_i lze iterovat, 1. a 3. parametr nepovinný.

Příklady použití:

```
for i in range(max):       #Interval 0, max-1
   pass;
for i in range(min, max): #Interval min, max-1
   pass;
```

12. Ukázka použití příkazu continue

Generování 10 náhodných čísel $x \in \langle x_{min}, x_{max} \rangle$ v $\mathbb{R}$, $x_{min} = -x_{max}$.
Výpočet odmocniny

$$y = \sqrt{x}, \qquad x \geq 0.$$

Ukázka řešení v jazyce Python:

```
import random                                #Import libraries
import math
max = 1000                                   #Maximum/minimum value
for i in range(10):                          #i=0,1,..., 9
    x = random.randint(-max, max)            #Create random number
    if x < 0:                                #x is negative
        continue                             #Skip next steps
    y = math.sqrt(x)                         #Compute y=sqrt(x)
    print(y)                                 #Print value y
```

13. Zásady pro práci s cykly

1) Cyklus by měl mít pouze jednu řídící proměnnou.

2) Hodnota řídící proměnné u for by neměla být ovlivňována v těle cyklu.

3) Řídící podmínku cyklu zapisovat v kladném tvaru.

4) Řídící podmínku zjednodušovat.

5) Vyvarovat se vzniku nekonečných cyklů.

6) Preference cyklů `while`, `for` před `do while` (přehlednost).

7) Příkaz `break` v těle cyklu používat pouze na jednom místě.

8) ... Nebo lépe se mu zcela vyhnout :-).

9) V těle cyklu nepoužívat skoky nebo jen výjimečně (`continue`).