

# Cykly.

Cyklus while. Cyklus for. Range. Break. Continue.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

# Obsah přednášky

- 1 Cykly
- 2 Cyklus while
- 3 Cyklus for
- 4 Příkaz range()
- 5 Příkaz break()
- 6 Příkaz continue()
- 7 Závěr

# 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 ridici 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 cyklu prováděno, dokud je podmínka pravdivá.

Pokud není pravdivá, 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 přímo nad prvky datových struktur [], {}, () či s využitím jejich indexu (příkaz range()).

### Iterovatelný objekt:

Objekt (dynamická datová struktura), můžeme přistupovat k jeho prvkům.

Sekvenční procházení jeho prvků, přímý přístup.

Alternativně lze použít index prvku (většinou levostranný).

### 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 funkcionality.

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

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

Inicializace odhadů  $\underline{x} = x_1$ ,  $\bar{x} = x_1$  prvním prvkem, alternativně  $\underline{x} = \infty$ ,  $\bar{x} = -\infty$ .

Postupné porovnání  $\underline{x}$ ,  $\bar{x}$  s  $x_i$

$$\underline{x} = \min(\underline{x}, x_i), \quad \bar{x} = \max(\bar{x}, x_i).$$

Pro průchodu celou posloupností

$$\underline{x} = \underline{x}, \quad \bar{x} = \bar{x}.$$

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:
    if x < xmin:
        xmin = x
    if x > xmax:
        xmax = x
print(xmax)                         #Print results
print(xmin)
>>> 95,
>>> -27
```

#Repeat for  $\forall x \in X$   
#Compare current x to xmin  
#Actualize xmin  
#Compare current x to xmax  
#Actualize xmax

## 7. Ukázka cyklu for: bankomat

Vstup: částka  $c$ ,  $c \in \mathbb{N}$ .

Výstup: vyplacení co nejmenším počtem bankovek.

$$B = \{100, 200, 500, 1000, 2000, 5000\}.$$

Použití celočíselného dělení // a operace  $(\text{mod})$ .

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]
c = int(input("Zadej částku: "))
for b in B:
    n = c // b
    c = c%b
    if c > 0:
        print(str(n) + " x " + str(b)) #Vytiskni počet bankovek
    .
```

#Opakuj pro každou bankovku/minci

#Urči počet bankovek

#Zbytkova částka:  $c = c - n * b$

#Pokud ještě něco zbyva

```
>>> Zadej částku: 2300
>>> 1 x 2000
>>> 1 x 200
>>> 1 x 100
```

## 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 sestu z existujícího.

```
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.

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 range()

Vytváří celočíselnou aritmetickou posloupnost  $\{a_k\}$

$$a_k = a_1 + (k - 1)d, \quad d \in \mathbb{N}, k \in \mathbb{N},$$

$d$  je diference posloupnosti.

$$\text{range}(a_1, a_n, d); \quad a_k < a_n < a_{k+1}.$$

Parametry  $a_1$  a  $d$  nepovinné, implicitní hodnoty  $a_1 = 0$ ,  $d = 1$ .

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(i1, in, 1);
```

Generuje aritmetickou posloupnost indexů

$$\{i_1, i_1 + 1, \dots, i_n - 1\}.$$

Příklady použití:

```
for i in range(max):      #Prvky 0, 1, 2, ..., max-1
    pass;
for i in range(min, max): #Prvky min, min+1, min+1, ..., max-1
    pass;
```

# 10. Použití range a for

Kombinace for a range: přístup k prvkům s využitím indexu.

Vygenerujeme aritmetickou posloupnost

$$i \in <0, n - 1>, \quad d = 1.$$

Její prvky používány jako index.

Zpravidla indexujeme ve směru zleva doprava.

```
for i in range(len(X)):  
    X[i] #Přístup indexem
```

Ve většině programovacích jazyků běžnější.

Z hlediska rychlosti přístupu stejné.

Hledání maxima/minima, varianta s indexem.

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 i in range(len(X)):  
    if X[i] < xmin:  
        xmin = X[i] #Repeat for  $\forall x_i \in X$   
    if X[i] > xmax:  
        xmax = X[i] #Compare X[i] to xmin  
    print(xmax) #Actualize xmin  
    print(xmin) #Compare X[i] to xmax  
    #Actualize xmax  
    #Print results
```

# 11. Příkaz break()

Okamžité (předčasné) ukončení provádění těla cyklu.

Provedeno, pokud podmínka vyhodnocena jako pravdivá.

```
if výraz:  
    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 podmínu 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.

## 12. Příkaz continue()

Přeskovení 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!

## 13. Ukázka použití příkazu continue

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

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

Ukázka řešení v jazyce Python:

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

## 14. 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).

