Lekce 2

Začínáme programovat II.

Logické operátory, podmínky, cykly. Kolekce - seznamy, N-tice a slovníky. Vstup a výstup.

C2184 Úvod do programování v Pythonu
podzim 2015

Mgr. Stanislav Geidl
Národní centrum pro výzkum biomolekul
Masarykova univerzita

Logické operátory

`and`	a
`or`	nebo
`not`	negace
`==`	je rovno
`!=`, `<>`	není rovno
`>`	je větší
`<`	je menší
`>=`	je větší rovno
`<=`	je menší rovno
`is`, `is not`	je, není (např. `1 is int()` nebo `a is not None`)
`in`, `not in`	je v, není v (např. `'pes' not in 'Harapes'` nebo `'a' in 'test'`)

- pořadí operací: matematické operace; < > <= >=; == !=; <>; is, is not; in, not in; not; and; or

Bloky

- **Bloky slouží k seskupení příkazů**, například uvnitř cyklů, funkcí, objektů, struktur atd.
- Například v Pascalu jsou bloky označeny slovy BEGIN a END a v jazyce C slouží pro vytváření bloků špičaté závorky . V Pythonu se pro vytváření bloků používá odsazování.
- Nový blok vytvoříte tak, že napíšete na začátku řádků před příkazy, které spolu tvoří blok, libovolný počet mezer nebo tabulátorů. Ale **před každým dalším příkazem v bloku musí být stejný počet mezer a tabulátorů**! Odsadíte-li o mezeru více, nebo pokud místo tabulátoru použijete mezery, začnete tím jiný blok. Pokud vytvoříte nový blok na místě, kde nemá být, skončí program s chybou.

```
prikaz1:
  blok1
  blok1
prikaz2:
  blok2
  blok2
  prikaz3:
    blok3
prikaz4:
  blok4
```


Podmínky

Na základě vyhodnocení logického výrazu se rozhodne, jestli se daný blok provede nebo nikoliv.

Podmínka if

```
if `podmínka`:
  print("Podmínka splněna.")
```

```
if 9 > 5:
  print("Devět je větší než pět.")
```

```
x = 5
if x >= 2 and x < 10:
  print("Proměnná náleží do intervalu <2;10)")
```


Podmínka if .. else

```
if `podmínka`:
  print("Podmínka splněna.")
else:
  print("Podmínka nesplněna.")
```

```
if 9 < 5:
  print("Devět je menší než pět.")
else:
  print("Devět není menší než pět.")
```

```
x = 13
if x >= 2 and x < 10:
  print("Proměnná náleží do intervalu <2;10)")
else:
  print("Proměnná nenáleží do intervalu <2;10)")
```


Podmínka if .. elif .. else

```
if `podmínka` and `podmínka2`:
  print("Podmínka splněna.")
elif `podmínka2`:
  print("Alespoň podmínka2 splněna.")
else:
  print("Podmínky nesplněny.")
```

```
x = 11
if x >= 2 and x < 10:
  print("Proměnná náleží do intervalu <2;10)")
elif x >= 5 and x < 15:
  print("Proměnná náleží do intervalu <5;15)")
else:
  print("Proměnná x mimo povolený rozsah!")
```

Kdyby *x* bylo rovno 8, který blok by se provedl?

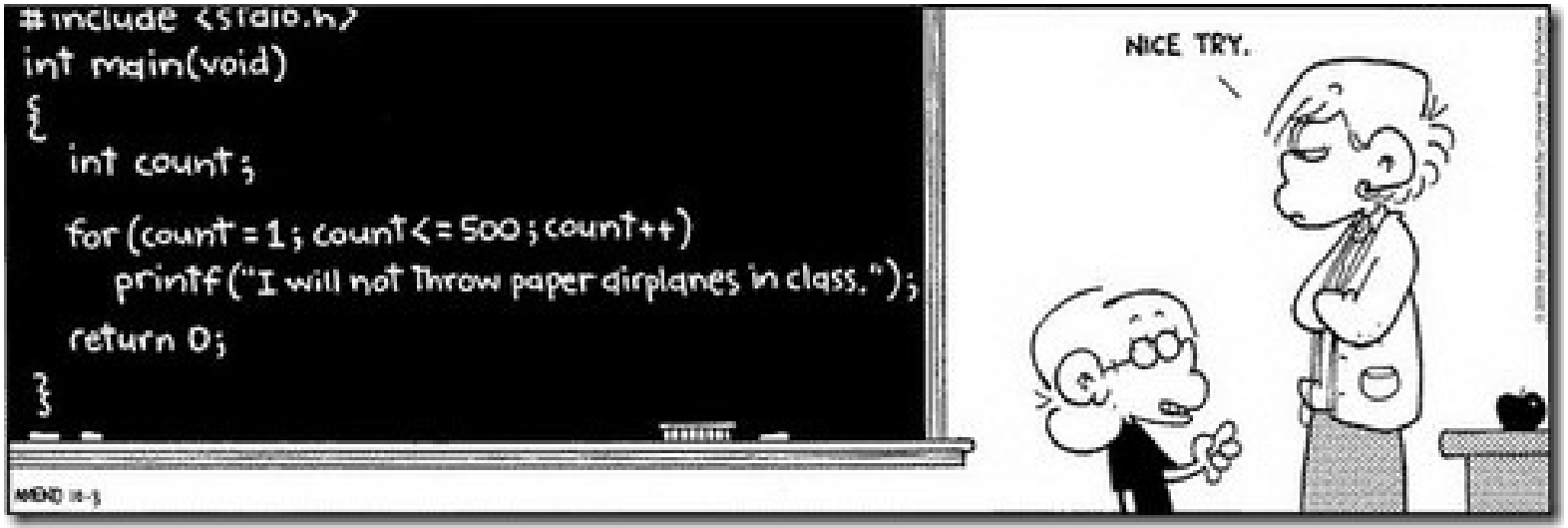
Příklad - je číslo dělitelné třemi nebo pěti?

Příklad - je číslo dělitelné třemi nebo pěti?

```
cislo = 14
if cislo%3 == 0 and cislo%5 == 0:
    print("Číslo je dělitelné třemi i pěti.")
elif cislo%3 == 0:
    print("Číslo je dělitelné třemi.")
elif cislo%5 == 0:
    print("Číslo je dělitelné pěti.")
else:
    print("Číslo není dělitelné třemi ani pěti.")
```


Cykly

Podobně jako podmínky se řídí logickým výrazem, který rozhoduje o spuštění příslušného bloku, tyto bloky mohou (většinou to i chceme :)) běžet i několikrát po sobě.



Šetří nám čas s psaním kódu a ruce od používání Ctrl+C/V.

Cyklus while

```
while `podminka`:
    blok
    blok

i = 0
while i < 10:
    print(i)
    i += 1
```


Cyklus for

```
for i in ...:
    blok
    blok

for i in range(10):
    print(i)
```


Řízení běhu cyklu pomocí break a continue

- `break` vynutí ukončení cyklu
- `continue` vynutí ukončení vykonávání bloku a spustí další smyčku

```
for i in range(100):
    if i%3 == 0:
        print("<3")
        continue
    if i >= 10:
        break
    print(i)
```


Řízení běhu cyklu pomocí break a else

- `else` pokud cyklus nebyl ukončen pomocí `break` spustí svůj blok

```
for i in range(1,10):
    print(i)
    if i % 11 == 0:
        break
else:
    print("Žádné číslo dělitelné 11 nenalezeno.")
```

Příklad - výpočet odmocniny pomocí Newtonovy metody

Tato metoda hledá řešení rovnice $f(x) = 0$ a odhaduje (zlepšuje odhad) na základě tohoto výpočtu:

$$x_{betterguess} = x_{guess} - \frac{f(x_{guess})}{f'(x_{guess})} \tag{1}$$

tento výpočet je neustále iterován, dokud nejsme s výsledkem spokojeni.

Příklad - výpočet odmocniny pomocí Newtonovy metody

$$x = \sqrt{a} \tag{2}$$

$$f(x) = x^2 - a = 0 \tag{3}$$

$$f'(x) = 2x \tag{4}$$

$$x_{betterguess} = x - \frac{x^2 - a}{2x} = \frac{x + \frac{a}{x}}{2} \tag{5}$$

```
a = 9
x = 1.0 # initial guess
while abs(x**2-a) > 0.000000001:
    x = (x+a/x) / 2
    print(x)
```


List (seznam)

- patří do kolecí, podobně jako N-tice a slovník
- vytváříme pomocí hranatých závorek `[]`
  `["a", "b", "c", "d"]`
- každý prvek má svůj automatický index, který odpovídá pořadí

Práce se seznamy I.

- vytvoření

```
seznam1 = [1, 1, 2, 3, 5, 8, 13]
seznam2 = list(seznam1)
seznam3 = seznam1[:]
seznam4 = seznam1 # nejedna se o novy list,
pouze odkaz na stary!!!
seznam5 = range(2,20,2)
# [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- přidáváme prvky

```
seznam1.append(21)
# [1, 1, 2, 3, 5, 8, 13, 21]
seznam2.insert(2, 90)
# [1, 1, 90, 2, 3, 5, 8, 13]
seznam3.extend([21, 34])
# [1, 1, 90, 2, 3, 5, 8, 13, 21, 34]
seznam3.append([21, 34])
# [1, 1, 2, 3, 5, 8, 13, [21, 34]]
```


Práce se seznamy II. - přístup k hodnotám

- můžeme přistupovat k jakémukoliv prvku pomocí jeho indexu
  `seznam[x]`, kde nula a kladné číslo n určuje index zleva a zaporné číslo určuje index zprava

```
[1, 2, 3, 4, 5][0] # 1
[1, 2, 3, 4, 5][1] # 2
[1, 2, 3, 4, 5][-1] # 5
```

- přes dvojtečku můžeme nadefinovat rozsah
  `seznam[x:y]`, kde tyto výrazy si odpovídají:

```
[1, 2, 3, 4, 5][:] # [1, 2, 3, 4, 5]
[1, 2, 3, 4, 5][2:] # [3, 4, 5]
[1, 2, 3, 4, 5][:2] # [1, 2]
```

- pozor na číslovaní! v Pythonu začínáme od nuly!

```
[1, 2, 3, 4, 5]
 0. 1. 2. 3. 4.
```

- co bude výsledkem?

```
seznam = [1, 2, 4, 5, 6]
```

```
x[1:4]          x[2:]          x[:2]
x[2:2]          x[-2:]         x[:-2]
```


Práce se seznamy III.

```
seznam1 = [1, 1, 2, 3, 5, 8, 13]
```

- mazání

```
seznam1.remove(1)
# [1, 2, 3, 5, 8, 13, 21]
last = seznam1.pop()[1]
# last = 21; [1, 1, 2, 3, 5, 8, 13]
```

- přehození směru

```
seznam1.reverse()
# [13, 8, 5, 3, 2, 1, 1]
```

- vyhledávání

```
seznam1.index(5) # 4
seznam1.count(1) # 2
```

- seřazení

```
seznam = [1, 4, 3, 6, 2, 5]
seznam.sort() # [1, 2, 3, 4, 5, 6]
seznam.sort(reverse=True) # [6, 5, ...
```

[1] list funguje jako zásobník, FIFO = first in, first out

Práce se seznamy IV.

- počítání

```
seznam1 = [1, 1, 2, 3, 5, 8, 13]
len(seznam1) # 7
sum(seznam1) # 33
min(seznam1) # 1
max(seznam1) # 13
```

- procházení

```
for item in [1, 2, 3]:
  print(item)
for item in range(1,4):
  print(item)
```


Tuple (N-tice)

- vytváříme pomocí jednoduchých závorek `()`
- můžeme s nimi pracovat podobně jako se seznamy, jenom je nemužeme měnit, tzn. že funkce `append` a další nejsou dostupné
- můžeme jednoduše převádět na list pomocí `list((1,2))` a podobně zpět `tuple([1,2])`

Dictionary (slovník)

- vytváříme pomocí složených závorek `{}`
  `{1: 3, 2: 4}`
- prvek ve slovníku se skládá z klíče a jeho hodnoty, 1 a 2 jsou klíče, jejich hodnoty jsou 3, resp. 4
- nefungují zde indexy, na hodnoty se dotazujeme pomocí klíče
- každý klíč je unikatní, žádný slovník nemůže obsahovat dva stejné klíče

Práce se slovníky

- vytvoření
```
dict = {'Name': 'Zara', 'Age': 7, 'Class':
'First'}
```
- čtení/získání
```
print(dict['Name']) # Zara
```
- přidání nebo úprava hodnot
```
dict['Age'] = 8 # úprava stávající hodnoty
dict['School'] = "DPS School"# přidání nové
```
- smazání hodnot
```
del dict['Name']
dict.clear() # smaže všechny položky
del dict # smaže celý slovník
```
- procházení hodnot
```
for key in dict:
  print(key)
  print(dict[key])
```

Vícerozměrné kolekce

- kolekce můžeme kombinovat a vytvářet list listů, ...
  `[[1, 2], [2, 3], [4, 5]]`
- můžeme kombinovat i navzájem a vytvářet list N-tic, ...
  `[(1, 2), (2, 3), (4, 5)]`

Vstupy a výstupy

Vstup:

- `raw_input()` v Pythonu 2.x nebo `input` v Pythonu 3
- argumenty programu (přes modul `sys`):

```
import sys
if len(sys.args) > 1:
    print sys.argv[1]
```

- ze souboru

Výstup:

- print()
- `sys.stdout` a `sys.stderr`
- do souboru

Code style - podmínky

Správně:

```
if not users:
    print 'no users'

if foo == 0:
    self.handle_zero()

if i % 10 == 0:
    self.handle_multiple_of_ten()
```

Špatně:

```
if len(users) == 0:
    print 'no users'

if foo is not None and not foo:
    self.handle_zero()

 if not i % 10:
    self.handle_multiple_of_ten()
```