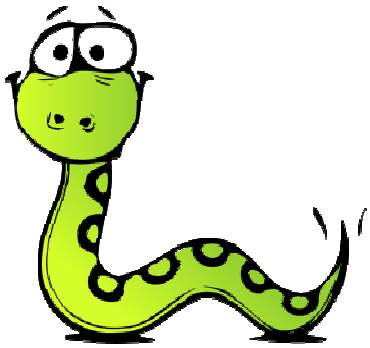
Programování v jazyce Python pro střední školy

Metodický list pro učitele

Lekce 18 – Vnořené větvení



Andrej Blaho

Ľubomír Salanci

Václav Šimandl

Cíle lekce

- Naučit se řešit úlohy, které vyžadují vícenásobné nebo vnořené větvení

Dovednosti

- Správný zápis podmínek s různými typy porovnání

Osvojená syntaktická pravidla

- Zápis operací rovnost == a nerovnost `!=`

Průběh výuky

Začínáme úlohou na opakování:

1. Vytvoř program `absolutni_hodnota.py`, který zobrazí absolutní hodnotu čísla. Do proměnné `a` přiřaď číslo. Použij příkaz větvení, abys vypsal absolutní hodnotu tohoto čísla. Například:
   - když bude `a = -7`, program vypíše `Absolutní hodnota -7 je 7`,
   - když bude `a = 13`, program vypíše `Absolutní hodnota 13 je 13`.

Řešení:

```
a = -7
if a < 0:
    print('Absolutní hodnota', a, 'je', -a)
else:
    print('Absolutní hodnota', a, 'je', a)
```

Řešení s použitím konstrukce `input` a reorganizací výpisu výsledku:

```
a = int(input('Zadej číslo: '))
if a < 0:
    vysledek = -a
else:
    vysledek = a
print('Absolutní hodnota', a, 'je', vysledek)
```

V Pythonu existuje standardní funkce `abs`, pomocí níž by bylo možno řešení zapsat následujícím způsobem:

```
a = -7
print('Absolutní hodnota', a, 'je', abs(a))
```

Předpokládáme, že o této funkci žáci zatím nevědí a úlohu budou řešit pomocí příkazu větvení. Když to uznáme za vhodné, můžeme žáky o funkci `abs` informovat.

V následující úloze se žáci seznámí s operátory pro test na rovnost a nerovnost. Je potřeba žáky nechat, aby v interaktivním režimu vyzkoušeli zápis operátorů a vyhodnocení výrazů.

2. Zatím umíš porovnávat dvě čísla pomocí <, >. Vyzkoušej, jak fungují testy rovnosti == a nerovnosti != . Napiš do příkazového řádku následující výrazy a zjisti, co Python vypíše:

```
a) >>> 1 == 1
b) >>> 1 == 2
c) >>> 0 != 2
d) >>> 0 != 0
e) >>> x = 100
   >>> x == 10 * 10
f) >>> x != 10 * 10
g) >>> 11 * 11 - 21 == x
h) >>> 1000 / 10 - 1 != x
```

Python v jednotlivých případech zobrazí:

```
a) >>> 1 == 1
   True
b) >>> 1 == 2
   False
c) >>> 0 != 2
   True
d) >>> 0 != 0
   False
e) >>> x = 100
   >>> x == 10 * 10
   True
f) >>> x != 10 * 10
   False
g) >>> 11 * 11 - 21 == x
   True
h) >>> 1000 / 10 - 1 != x
   True
```

V následující tabulce shrnujeme relační operátory, které jsou v Pythonu určené k porovnávání číselných hodnot:

Operátor	Význam	Příklad
<	Menší než	`9 * 11 < 100`
>	Větší než	`22 / 7 > 3.14`
<=	Menší nebo rovno	`2 * 2 <= 4`
>=	Větší nebo rovno	`5 * 5 >= 4 * 6`
==	Rovno	`1 + 2 == 3`
!=	Nerovnající se, různý	`1 + 2 != 3.14`

Je potřeba si však uvědomit, že aritmetika desetinných čísel není zcela přesná (bližší vysvětlení viz 4. lekce). Proto můžeme v některých případech dostávat překvapivé výsledky, jako jsou například tyto:

```
>>> 0.1 + 0.2 == 0.3
False
>>> 3 * 0.1 > 0.3
True
```

Problematiku přesnosti aritmetiky desetinných čísel nemusíme žákům vysvětlovat. Žáci se však při řešení úloh mohou setkat s problémy, které jsou spojené s chybovostí této aritmetiky. Na tyto situace bychom měli být připraveni a měli bychom být schopni jim danou situaci vysvětlit.

3. Víš, co se stane, když má počítač dělit nulou? Vyzkoušej to v příkazovém řádku. Potom vytvoř program `prevracena_hodnota.py`, který spočítá převrácenou hodnotu čísla (převrácená hodnota čísla *x* je rovna $\frac{1}{x}$). Do proměnné `n` přiřaď číslo. Použij příkaz větvení a test rovnosti, aby:
   - v případě, že `n = 0`, se zobrazila zpráva `Nulou dělit neumím`,
   - jinak se zobrazil výsledek, například pro `n = 10` se zobrazilo `1 / 10 = 0.1`.

Když v interaktivním okně vyzkoušíme dělení nulou:

```
>>> 1 / 0
```

Python vypíše chybové hlášení:

```
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    1 / 0
ZeroDivisionError: division by zero
```

V tomto chybovém hlášení je obsažen důvod chyby – `ZeroDivisionError: division by zero`. Program, ve kterém by se dělilo nulou, se nedokončí, ale skončí chybou („spadne“) a zobrazí podobnou chybovou zprávu. Proto je dobrým programátorským zvykem situace, ve kterých je riziko, že by se dělilo nulou, ochránit příkazem větvení.

Možné řešení může být například takovéto:

```
n = 10
if n == 0:
    print('Nulou dělit neumím.')
else:
    print('1 /', n, '=', 1 / n)
```

Poslední příkaz `print` lze zapsat i tímto způsobem: `print(f'1 / {n} = {1 / n}')`. Takový zápis však od žáků neočekáváme – žáky jsme tento zápis neučili a ani nepožadujeme, aby jej znali.

4. Vytvoř nový program `stejna_cisla.py`. Do proměnných `x, y` přiřaď dvě čísla. Napiš kód, který určí a vypíše, zda jsou tato čísla stejná nebo různá. Například:

`Čísla jsou různá.`	`Čísla jsou stejná.`
pro `x = 1, y = 0`	pro `x = 2, y = 2`

Řešení:

```
x = 1
y = 1
if x != y:
    print('Čísla jsou různá.')
else:
    print('Čísla jsou stejná.')
```

Alternativní řešení s použitím konstrukce `input`:

```
x = int(input('Zadej první číslo: '))
y = int(input('Zadej druhé číslo: '))
if x != y:
    print('Čísla jsou různá.')
else:
    print('Čísla jsou stejná.')
```

Místo operátoru `!=` lze pochopitelně použít operátor rovná se `==` a adekvátně tomu upravit zbylé části příkazu větvení.

5. Vytvoř nový program `obdelnik_nebo_ctverec.py`. Do proměnných `a, b` přiřaď délky stran útvaru, u nějž nevíme, zda je to obdélník nebo čtverec. Napiš kód, který určí a vypíše, zda je daný útvar obdélníkem nebo čtvercem. Například:

`Je to čtverec.`	`Je to obdélník.`
pro `a = 10, b = 10`	pro `a = 10, b = 20`

Řešení:

```
a = 10
b = 10
if a == b:
    print('Je to čtverec.')
else:
    print('Je to obdélník.')
```

6. Házíme desetkrát hrací kostkou a chceme vědět, kolikrát padla šestka a kolikrát jiné číslo. Vytvoř program `pocet_sestek.py`, který pomocí cyklu, generování náhodných čísel a vnořeného příkazu větvení simuluje deset hodů kostkou, hozená čísla vypíše a spočítá, kolikrát padla šestka a kolikrát jiné číslo. Tyto počty poté vypíše. Jestliže například budou hozena následující čísla:

```
5
6
5
2
2
3
1
4
5
3
```

program vypíše:

```
Padlo 1 šestek a 9 jiných čísel
```

Uprav program tak, aby bylo kostkou hozeno 6000krát. Kolikrát padla šestka v tomto případě?

Řešení:

```
import random

pocet_6 = 0
pocet_jine = 0
for i in range(10):
    n = random.randint(1, 6)
    print(n)
    if n == 6:
        pocet_6 = pocet_6 + 1
    else:
        pocet_jine = pocet_jine + 1
print('Padlo', pocet_6, 'šestek a', pocet_jine,
    'jiných čísel')
```

V případě 6000 hodů kostkou by měla šestka padnout přibližně 1000krát.

V této úloze používáme proměnné `pocet_6` a `pocet_jine` jako dvě počítadla. Do příslušné proměnné **připočítáme 1** vždy, když nastane určitá očekávaná situace, v našem případě na hrací kostce padlo číslo 6, resp. na hrací kostce padlo číslo jiné než 6. Zápis:

```
proměnná = proměnná + 1
```

je typický právě pro počítadla a znamená: **„zvyš hodnotu proměnné o 1“**. Tento princip však bude fungovat pouze tehdy, jestliže tato `proměnná` bude ještě před prvním zvýšením své hodnoty vynulovaná. To jsme u obou proměnných udělali ještě před začátkem for cyklu.

Při simulování 6000 hodů kostkou trvá vypsání hozených čísel delší dobu. Někteří žáci se mohou zajímat o to, zda by bylo možné příkaz `print(n)` uvnitř for cyklu dočasně skrýt, aby se neprováděl a přitom aby nebylo nutné jej zcela smazat. Můžeme jim prozradit, že je možné řádek s tímto příkazem tzv. zakomentovat. Toho lze dosáhnout zapsáním znaku # na začátek daného řádku. Takto označený řádek se stane pro Python neviditelným a příkazy v něm uvedené bude ignorovat. Kód programu by po této úpravě mohl vypadat například následovně:

```
import random

pocet_6 = 0
pocet_jine = 0
for i in range(6000):
    n = random.randint(1, 6)
#    print(n)
    if n == 6:
        pocet_6 = pocet_6 + 1
    else:
        pocet_jine = pocet_jine + 1
print('Padlo', pocet_6, 'šestek a', pocet_jine,
    'jiných čísel')
```

Programátoři znak # nepoužívají jen pro označení příkazů, které se nemají provádět, ale také pro zapisování komentářů, v nichž sami sobě připomínají či jiným programátorům vysvětlují význam jednotlivých částí kódu. Pokud má být takový komentář na více řádcích, je potřeba na začátek každého z nich zapsat znak # .

7* Hrajeme hru, ve které házíme desetkrát kostkou a získáváme prémii vždy, když za sebou padnou dvě stejná čísla. Vytvoř program `stejna_za_sebou.py`, který simuluje deset hodů kostkou, hozená čísla vypíše a spočítá počet prémií. Tento počet prémií potom vypíše. Jestliže například budou hozena následující čísla:

```
4
4
6
5
3
3
3
1
5
5
```

program vypíše:

```
Počet prémií: 4
```

Řešení:

```
import random

pocet_premii = 0
predtim = 0
for i in range(10):
    n = random.randint(1, 6)
    print(n)
    if n == predtim:
        pocet_premii = pocet_premii + 1
    else:
        predtim = n
print('Počet premií:', pocet_premii)
```

Při řešení této úlohy si žáci musí uvědomit, že kromě počítadla prémií (proměnná `pocet_premii`) si program musí zapamatovat, jaká hodnota padla na kostce v předchozím hodu. Toho lze dosáhnout použitím vhodné proměnné. My jsme ji v našem řešení nazvali `predtim` a před začátkem for cyklu jsme jí nastavili hodnotu `0` (což je hodnota, která není na hrací kostce). V případě, že na kostce padla jiná hodnota, než je hodnota proměnné `predtim`, zapamatujeme si v této proměnné momentální hod z proměnné `n`.

8. Fotbaloví rozhodčí stanovili, jak budou hráče hodnotit za přestupky proti pravidlům:

- když se hráč dopustil 0 přestupků, hraje férově,
- když se dopustil 1 nebo 2 přestupků, dostane žlutou kartu,
- jinak dostane červenou kartu a je vyloučen ze hry.

Vidíš, že v této úloze je více podmínek. Prohlédni si následující řešení, vytvoř nový program `prestupky.py` a vyzkoušej jeho funkčnost:

```
pocet = 0
if pocet == 0:
    print('Hraješ férově')
else:
    if pocet < 3:
        print('Máš žlutou kartu')
    else:
        print('Máš červenou kartu')
```

vnořený `if else` – je potřeba jej odsadit od kraje

tyto příkazy je potřeba odsadit od kraje ještě více

Doplň do tabulky, co program vypíše, když do proměnné `pocet` přiřadíme hodnotu:

Řešení:

	program vypíše:
`pocet = 0`	`Hraješ férově`
`pocet = 1`	`Máš žlutou kartu`
`pocet = 2`	`Máš žlutou kartu`
`pocet = 3`	`Máš červenou kartu`
`pocet = 4`	`Máš červenou kartu`
`pocet = -1`	`Máš žlutou kartu`

Někteří žáci se mohou dožadovat efektivnějšího způsobu vyzkoušení úlohy. Jedním z nich je použití konstrukce `input`, s jejímž využitím by program mohl vypadat například takto:

```
pocet = int(input('Zadej počet přestupků: '))
if pocet == 0:
    print('Hraješ férově')
else:
    if pocet < 3:
        print('Máš žlutou kartu')
    else:
        print('Máš červenou kartu')
```

Jiným přístupem k efektivnímu vyzkoušení úlohy může být vložení celého **vnořeného větvení** s výpisy do vhodného for cyklu, například takto:

```
for pocet in range(5):
    print('pocet =', pocet)
    if pocet == 0:
        print('Hraješ férově')
    else:
        if pocet < 3:
            print('Máš žlutou kartu')
        else:
            print('Máš červenou kartu´')
```

9. V televizní soutěži o pečení *Můj děda peče líp než tvůj* jsou následující pravidla:

- když soutěžící stihne upéct méně než 10 koláčků, je hodnocen jako začátečník,
- když stihne upéct aspoň 10, ale méně než 20 koláčků, je hodnocen jako pokročilý,
- když stihne upéct aspoň 20 koláčků, je hodnocen jako expert.

Vytvoř nový program `kolacky.py`, ve kterém přiřadíš do proměnné `kolace` počet upečených koláčků. Program poté rozhodne, zda je daný soutěžící začátečník, pokročilý nebo expert, a vypíše vhodnou hlášku.

Otestuj, zda program zobrazí správnou hlášku pro následující počty upečených koláčků: `1, 9, 10, 19, 20, 100`.

Řešení:

```
kolace = 100
if kolace < 10:
    print('Jsi začátečník, trénuj')
else:
    if kolace < 20:
        print('Hmmm, jsi pokročilý')
    else:
        print('Jsi expert, dostáváš Michelinskou hvězdu')
```

Při sestavování složitějších konstrukcí s více podmínkami mohou žáci vytvořit nesprávně fungující kód. Níže uvádíme jedno z chybných řešení, jehož vykonávání sice neskončí syntaktickou chybou, ale které přesto nefunguje správně. Oproti správnému řešení zde není vnořený příkaz větvení odsazený od kraje a je vynechán řádek `else:`

```
kolace = 9
if kolace < 10:
    print('Jsi začátečník, trénuj')
if kolace < 20:
    print('Hmmm, jsi pokročilý')
else:
    print('Jsi expert, dostáváš Michelinskou hvězdu')
```

Tento kód Python chápe jako dva na sobě nezávislé příkazy větvení, a tak by se pro 9 upečených koláčků za sebou vypsaly dvě zprávy: `Jsi začátečník, trénuj` a `Hmmm, jsi pokročilý`. Tuto chybu lze poměrně snadno odhalit na základě vyzkoušení programu s různými hodnotami proměnné `kolace`. K tomuto testování je potřeba žáky vést, byť jim může připadat jako nedůležité a časově náročné.

Pokud bychom chtěli otestovat všech šest různých počtů koláčů uvedených v zadání, tj. `1, 9, 10, 19, 20, 100`, nemůžeme k tomuto účelu použít doposud používaný for cyklus:

```
for kolace in range(6)
```

neboť bychom takto dostali jen počty `0, 1, 2, 3, 4, 5`.

Jestliže máme pokročilejší skupinu žáků, kteří přivítají různá programátorská vylepšení, můžeme zde použít jiný zápis for cyklu. V tomto zápisu místo `range(6)` vyjmenujeme hodnoty, jichž má proměnná cyklu postupně nabývat. Po této úpravě by náš program mohl vypadat například takto:

```
for kolace in 1, 9, 10, 19, 20, 100:
    print('Počet koláčů =', kolace)
    if kolace < 10:
        print('Jsi začátečník, trénuj')
    else:
        if kolace < 20:
            print('Hmmm, jsi pokročilý')
        else:
            print('Jsi expert, dostáváš Mišelinskou hvězdu')
```

10. V úloze 6 vidíš, že počítač neumí správně skloňovat slova. Měl by vypsat:

- `Padla 1 šestka...`
- `Padly 2 šestky...`
- `Padlo 5 šestek...`

Vytvoř nový program `sklonovani.py`, ve kterém do proměnné `n` přiřadíš počet hozených šestek a počítač zobrazí gramaticky správnou větu podobně, jako je uvedeno výše. Otestuj, zda program funguje správně pro různé hodnoty proměnné `n`.

Řešení:

```
n = 5
if n == 1:
    print('Padla 1 šestka')
else:
    if n < 5:
        print('Padly', n, 'šestky')
    else:
        print('Padlo', n, 'šestek')
```

Zkušenější programátoři v Pythonu vědí, že takovéto vnořené příkazy větvení `if … else` lze zapisovat úsporněji a pro pokročilejší programátory i čitelněji pomocí konstrukce `if … elif … else`. Naše řešení by s použitím této konstrukce vypadalo následovně:

```
n = 5       # nebo n = int(input('Zadej počet šestek: '))
if n == 1:
    print('Padla 1 šestka')
elif n < 5:
    print('Padly', n, 'šestky')
else:
    print('Padlo', n, 'šestek')
```

Tento zápis raději žáky učit nebudeme, neboť to jsou další syntaktická pravidla, která mnohým z nich mohou výrazně komplikovat chápání zápisů programů v Pythonu. Pokud však některý žák takový zápis objeví a správně použije, nebudeme mu to zakazovat.

11. Chceš porovnat svůj věk s věkem kamarádky. Vytvoř program `porovnani_veku.py`, ve kterém do proměnných `ja` a `ona` přiřadíš svůj věk a věk tvé kamarádky. Program tyto údaje porovná a podle toho vypíše: `Jsme stejně staří`, `Jsem mladší` nebo `Ona je mladší`.

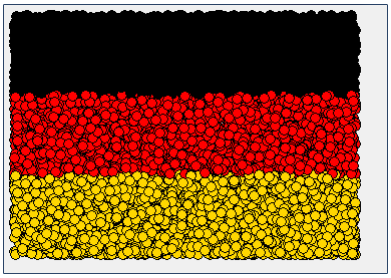
Řešení:

```
ja = 10
ona = 15
if ja == ona:
    print('Jsme stejně staří.')
else:
    if ja < ona:
        print('Jsem mladší.')
    else:
        print('Ona je mladší.')
```

12. Vytvoř nový program `nemecka_vlajka.py`, ve kterém budeš kreslit na plátno německou vlajku. Tu budeš vytvářet tak, že pomocí cyklu vygeneruješ 10 000krát náhodné souřadnice `x, y`. Souřadnice `x` bude z intervalu od `10` do `350` a souřadnice `y` bude z intervalu `od` 10 do `250`. Na tyto souřadnice `[x, y]` nakreslíš barevný kroužek s poloměrem 5. Barvu kroužku zvolíš podle y-ové souřadnice následovně:

    - když je `y < 90`, nakreslíš černý kroužek,
    - jinak, když je `y < 170`, nakreslíš červený kroužek,
    - jinak nakreslíš žlutý kroužek.

    Výsledek by měl vypadat podobně jako na obrázku níže:

    Vyzkoušej, jak bude výsledek vypadat, jestliže se bude generovat menší počet barevných kroužků (například 100 nebo 1000).

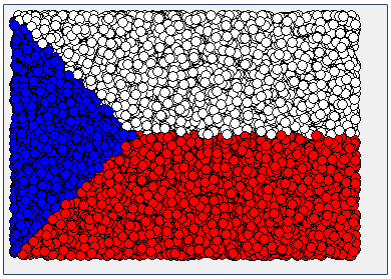
Řešení:

```
import tkinter
import random

canvas = tkinter.Canvas()
canvas.pack()

for i in range(10000):
    x = random.randint(10, 350)
    y = random.randint(10, 250)
    if y < 90:
        canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
            fill='black')
    else:
        if y < 170:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='red')
        else:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='gold')
```

Protože se ve všech třech větvích příkazu větvení provádí stejný příkaz `create_oval`, který se liší jen v parametru `fill=`, můžeme toto větvení reorganizovat podobně, jako jsme to učinili v předchozích lekcích (níže uvádíme pouze zápis for cyklu):

```
for i in range(10000):
    x = random.randint(10, 350)
    y = random.randint(10, 250)
    if y < 90:
        barva = 'black'
    else:
        if y < 170:
            barva = 'red'
        else:
            barva = 'gold'
    canvas.create_oval(x - 5, y - 5, x + 5, y + 5,fill=barva)
```

13* Vytvoř program `ceska_vlajka.py`, který podobnou technikou, jako byla použita v předchozí úloze, nakreslí obrázek podobný české vlajce. Výsledek může vypadat například takto:



Řešení:

```
import tkinter
import random

canvas = tkinter.Canvas()
canvas.pack()

for i in range(10000):
    x = random.randint(10, 350)
    y = random.randint(10, 250)
    if y < 130:
        if x < y:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='blue')
        else:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='white')
    else:
        if 130 - x > y - 130:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='blue')
        else:
            canvas.create_oval(x - 5, y - 5, x + 5, y + 5,
                fill='red')
```

Alternativně by bylo možné kód programu reorganizovat podobně jako v předchozí úloze (níže uvádíme pouze zápis for cyklu):

```
for i in range(10000):
    x = random.randint(10, 350)
    y = random.randint(10, 250)
    if y < 130:
        if x < y:
            barva = 'blue'
        else:
            barva = 'white'
    else:
        if 130 - x > y - 130:
            barva = 'blue'
        else:
            barva = 'red'
    canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=barva)
```

Všimněme si, že modrý klín se kreslí na dvou místech:

- Pro část v horní polovině vlajky (pro níž platí `y < 130`) musí platit `x < y`, tedy modré kroužky jsou pod úhlopříčkou.
- Pro část v dolní polovině vlajky (pro níž neplatí `y < 130`, neboli platí `y >= 130`) musí být modré kroužky nad otočenou uhlopříčkou `130 - x > y - 130`.

Vzorec pro kreslení modrého klínu v dolní polovině vlajky je už náročnější a mají jej patrně šanci zvládnout jen žáci, kteří jsou zdatnější v matematice.

Nakreslený obrázek se od české vlajky proporčně liší. Zatímco na české vlajce modrý klín zasahuje do poloviny délky vlajky, v našem případě je pouze do třetiny délky obrazce. Vytvoření proporčně odpovídajícího obrázku by však bylo pro žáky velice náročné, a tak jej nevyžadujeme (a ve vzorovém řešení ani neukazujeme).