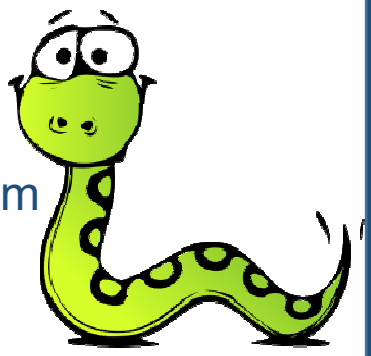
Programování v jazyce Python pro střední školy

Metodický list pro učitele

Lekce 19 – Podprogram s parametrem



Andrej Blaho

Ľubomír Salanci

Václav Šimandl

Cíle lekce

- Seznámit se s parametrem podprogramu jako označením neznámé hodnoty
- Naučit se zobecnit zápis programů pomocí podprogramů s parametrem
- Naučit se volat podprogram s parametrem

Osvojená syntaktická pravidla

- Zápis definice podprogramu s parametrem
- Zápis volání podprogramu s parametrem

Průběh výuky

1. Vytvoř nový program `cifry_cisla.py`, ve kterém přiřadíš do proměnné `cislo` číslo od 0 do 999. Použij větvení na to, aby program rozhodl a správně vypsal hlášení o tom, zda je číslo jedno-, dvou- nebo trojciferné. Například pro `cislo = 128` program vypíše:

```
Číslo 128 je trojciferné.
```

Řešení:

```
cislo = 128
if cislo < 10:
    print('Číslo', cislo, 'je jednociferné')
else:
    if cislo < 100:
        print('Číslo', cislo, 'je dvouciferné')
    else:
        print('Číslo', cislo, 'je trojciferné')
```

2. Vytvoř nový program `muj_vek.py`. Přepiš do něj následující kód a dokonči jednotlivé podprogramy, aby vypisovaly správný věk:

```
def jemi10():
    vek = 10
    print('Je mi', vek, 'let')

def jemi20():
    vek = .....
    print('Je mi', vek, 'let')

def jemi30():
    vek = 30
    print(.....)

jemi10()
jemi20()
jemi30()
```

Udělej to tak, aby se všechny tři podprogramy navzájem co nejvíc podobaly.

Řešení:

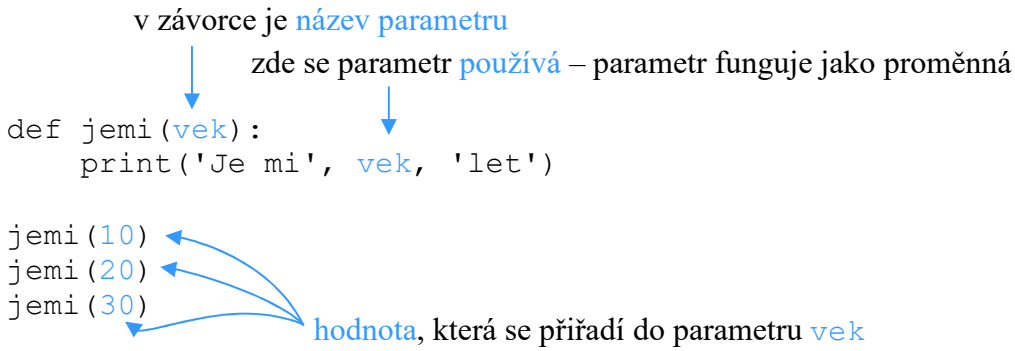
```
def jemi10():
    vek = 10
    print('Je mi', vek, 'let')

def jemi20():
    vek = 20
    print('Je mi', vek, 'let')

def jemi30():
    vek = 30
    print('Je mi', vek, 'let')
```

3. Předchozí řešení se dá zapsat pomocí jediného podprogramu:

```
def jemi(vek):
    print('Je mi', vek, 'let')

jemi(10)
jemi(20)
jemi(30)
```

Vyzkoušej jej.

Jak program funguje?



Cílem 2. a 3. úlohy je objevit koncept **parametru**: místo přiřazení nějaké hodnoty do proměnné `vek` uvnitř podprogramu vytvoříme podprogram, který má v hlavičce za názvem podprogramu v kulatých závorkách uvedený název proměnné (v našem případě `vek`). Tímto zápisem jsme Pythonu oznámili, že při volání tohoto podprogramu bude ještě potřeba uvést i **hodnotu parametru.** Pokud tuto hodnotu při volání podprogramu neuvedeme, program nebude fungovat (skončí chybou). Hodnotu parametru určujeme opět v kulatých závorkách.

Z terminologického hlediska:

```
def jemi(vek):                  # definování podprogramu
                                   i s názvem parametru
    print('Je mi', vek, 'let')   # používání parametru

jemi(10)                        # volání podprogramu s určením
                                   hodnoty parametru
```

Uvnitř podprogramu je parametr **obyčejnou proměnnou**, která má už na začátku podprogramu určenu svou počáteční hodnotu. Zvídavější žáci mohou vyzkoušet, že parametr se uvnitř podprogramu opravdu chová jako obyčejná proměnná:

```
def jemi(vek):
    print('Je mi', vek, 'let')
    vek = vek + 5
    print('Za pět let mi bude ', vek, 'let')

jemi(10)
```

4. Vytvoř nový program `druha_mocnina_parametr.py`. Přepiš do něj následující kód a dokonči podprogram `vypis`, který používá parametr `x` na to, aby vypsal hodnotu parametru `x` a jeho druhou mocninu:

```
def vypis(x):
    print('Číslo', …)
    print('Umocněné na druhou se rovná', ……)

vypis(1)
vypis(2)
vypis(3)
```

Program by měl po spuštění vypsat:

```
Číslo 1
Umocněné na druhou se rovná 1
Číslo 2
Umocněné na druhou se rovná 4
Číslo 3
Umocněné na druhou se rovná 9
```

Řešení:

```
def vypis(x):
    print('Číslo', x)
    print('Umocněné na druhou se rovná', x * x)

vypis(1)
vypis(2)
vypis(3)
```

5. Doplň do předchozího podprogramu příkaz, kterým se vypíše i převrácená hodnota `x`. Připomeňme, že převrácená hodnota čísla *x* je rovna $\frac{1}{x}$. Program by měl po spuštění vypsat:

```
Číslo 1
Umocněné na druhou se rovná 1
Převrácená hodnota se rovná 1.0
Číslo 2
Umocněné na druhou se rovná 4
Převrácená hodnota se rovná 0.5
Číslo 3
Umocněné na druhou se rovná 9
Převrácená hodnota se rovná 0.3333333333333333
```

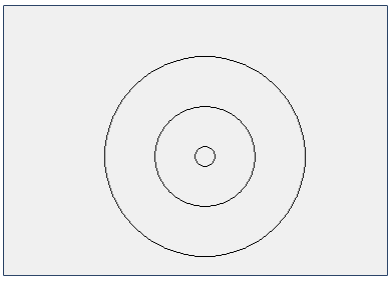
Řešení:

```
def vypis(x):
    print('Číslo', x)
    print('Umocněné na druhou se rovná', x * x)
    print('Převrácená hodnota se rovná', 1 / x)
```

6. Vytvoř nový program `kruh_parametr.py`. Přepiš do něj následující kód a dokonči podprogram `kruh` tak, aby kreslil kruhy se středem `200, 150` a poloměrem `r`, který bude parametrem podprogramu:

```
import tkinter

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

def kruh(r):
    canvas.create_oval(....., ....., ....., .....)

kruh(10)
kruh(100)
kruh(50)
```

Jestli jsi postupoval správně, program by měl nakreslit takovýto obrázek:



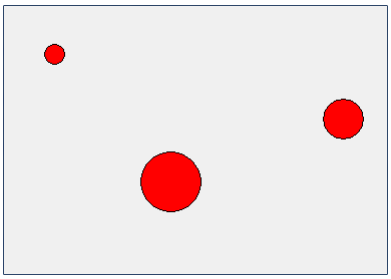
Řešení:

```
def kruh(r):
    canvas.create_oval(200 - r, 150 - r, 200 + r, 150 + r)
```

V předchozích lekcích jsme v podobných případech obvykle souřadnice středu přiřadili do proměnných `x` a `y`. Kdybychom to učinili i zde, získáme následující zápis podprogramu:

```
def kruh(r):
    x = 200
    y = 150
    canvas.create_oval(x - r, y - r, x + r, y + r)
```

7. Vytvoř nový program `nahodny_kruh_parametr.py` a v něm vytvoř podprogram `nahodny_kruh` s parametrem `r`, který nakreslí na náhodných souřadnicích červený kruh o poloměru `r`. Zavolej tento podprogram pro různé hodnoty parametru. Výsledek může vypadat například jako na následujícím obrázku:



Řešení:

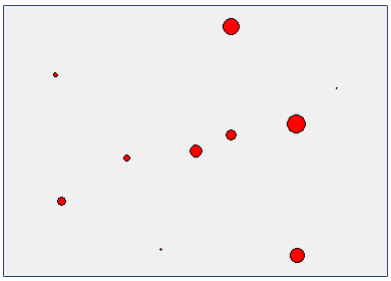
```
import tkinter
import random

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

def nahodny_kruh(r):
    x = random.randint(10, 350)
    y = random.randint(10, 250)
    canvas.create_oval(x - r, y - r, x + r, y + r,
        fill='red')

nahodny_kruh(10)
nahodny_kruh(20)
nahodny_kruh(30)
```

8. Vyzkoušej, co předchozí program nakreslí, když zavoláš podprogram `nahodny_kruh` následujícím způsobem:

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

Diskutuj se svým spolužákem, jak program funguje.

Program nakreslí:



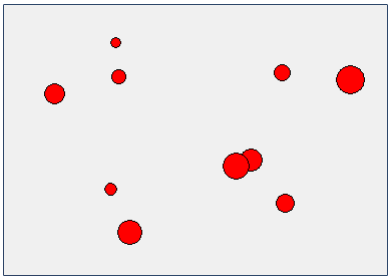
V tomto případě je hodnotou parametru `r` hodnota proměnné for cyklu. Jestliže tento for cyklus nabývá postupně hodnot `0`, `1`, … `9`, na náhodné pozice se nakreslí 10 červených kruhů. První z nich má poloměr `0`, a proto se z něj nakreslí jen jedna malá tečka.

9. Poloměr kruhu můžeme určit i takto:

```
for i in range(10):
    nahodny_kruh(i + 5)
```

Spusť program, abys viděl, co udělá, a vyplň následující tabulku:

Program nakreslí:



hodnota v proměnné `i`	hodnota parametru `r`
0	5
1	6
2	7
3	8
4	9
5	10
6	11
7	12
8	13
9	14

Program opět nakreslí 10 červených kruhů, avšak jejich poloměry nyní budou `5`, `6`, …, `14`.

Cílem další úlohy je vytvořit podprogram, který ve svém těle obsahuje příkaz větvení s rozhodovací podmínkou založenou na hodnotě parametru podprogramu.

10. Vytvoř nový program `oblibene_cislo.py` a v něm definuj podprogram `obliba` s parametrem `cislo`. Podprogram podle následujících pravidel vypíše, zda má číslo v oblibě:

    - když je číslo menší než 7, vypíše `Mám rád číslo ...`
    - jinak vypíše `Číslo ... se mi nelíbí`

    Podprogram zavolej z příkazového řádku a ověř, že vypíše:

```
>>> obliba(1)
Mám rád číslo 1
>>> obliba(5)
Mám rád číslo 5
>>> obliba(10)
Číslo 10 se mi nelíbí
```

Řešení:

```
def obliba(cislo):
    if cislo < 7:
        print('Mám rád číslo', cislo)
    else:
        print('Číslo', cislo, 'se mi nelíbí')
```

11. Uprav předchozí program tak, abys pomocí cyklu zavolal podprogram `obliba` pro čísla od 0 do 10. Výsledek by měl vypadat následovně:

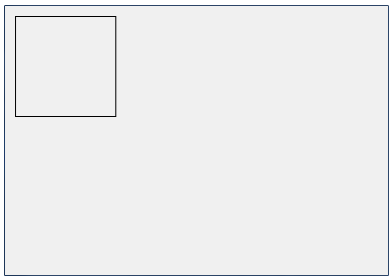
```
Mám rád číslo 0
Mám rád číslo 1
...
Mám rád číslo 6
Číslo 7 se mi nelíbí
...
Číslo 10 se mi nelíbí
```

Řešení – doplníme následující kód:

```
for i in range(11):
    obliba(i)
```

12* Vytvoř nový program `ctverec_parametr.py` a v něm definuj podprogram `ctverec` s parametrem `a`, který udává délku strany čtverce. Podprogram by měl fungovat tak, že čtverec kreslí jen pro kladné hodnoty parametru `a`, ale pro záporné hodnoty vypíše zprávu „`Nedá se`". Levý horní roh kresleného čtverce bude mít souřadnice `[10, 10]`. Zprávu vypiš přibližně do středu grafické plochy. Otestuj, že podprogram pracuje správně.

`ctverec(-50)`



`ctverec(100)`

Řešení:

```
import tkinter

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

def ctverec(a):
    if a > 0:
        canvas.create_rectangle(10, 10, 10 + a, 10 + a)
    else:
        canvas.create_text(200, 150, text='Nedá se')

ctverec(-50)
ctverec(100)
```

13* Uprav podprogram `ctverec` z předchozí úlohy tak, aby se pro záporné hodnoty parametru `a` nikde nic nevypsalo. Stačí, když smažeš větev `else:` i s příkazem pro výpis. Takovýto příkaz se nazývá `if` bez větve `else`:

```
if podmínka:
    příkaz
    příkaz
    ...
```

} větev if

Řešení:

```
def ctverec(a):
    if a > 0:
        canvas.create_rectangle(10, 10, 10 + a, 10 + a)
```

Pokud bychom nechtěli použít variantu příkazu větvení bez větve `else`, mohli bychom příkaz větvení zapsat například takto:

```
def ctverec(a):
    if a > 0:
        canvas.create_rectangle(10, 10, 10 + a, 10 + a)
    else:
        pass
```

Protože žádná z větví příkazu větvení nesmí být prázdná (Python by v tom případě hlásil chybu), použili jsme zde speciální příkaz `pass`, který se používá právě v takových případech. Tento příkaz opravdu nedělá nic. Ačkoliv se takový zápis považuje za méně čitelný než příkaz větvení bez větve `else`, můžeme jej využít při vysvětlování, jak příkaz větvení bez větve `else` funguje.

Zvídavějším žákům můžeme dát za úkol prozkoumat následující zápis:

```
def ctverec(a):
    if a <= 0:
        pass
    else:
        canvas.create_rectangle(10, 10, 10 + a, 10 + a)
```

14. Znáš hru *Myslím si číslo*, ve které je potřeba uhádnout neznámé číslo? Vytvoř takovou hru na počítači – počítač si vymyslí číslo od 1 do 5 a my ho musíme uhádnout. Vytvoř nový program `uhadni_cislo.py`, který bude fungovat následujícím způsobem:
    - po spuštění programu počítač přiřadí do proměnné `cislo` náhodně vygenerované číslo,
    - potom vypíše zprávu „`Myslím si číslo od 1 do 5. Zkus ho uhádnout...`“
    - ve svém programu budeš mít definovaný podprogram `zkus` s parametrem `n`, který porovná `cislo` s hodnotou `n` a vypíše: buď „`Hurá, uhádl jsi!`“, nebo „`Ne, moje číslo je jiné...`“.

    Hra může probíhat následovně – spustíme program a v příkazovém řádku odpovídáme tím, že voláme podprogram `zkus`:

    ```
    ========== RESTART ==========
    Myslím si číslo od 1 do 5. Zkus ho uhádnout...
    >>> zkus(3)
    Ne, moje číslo je jiné...
    >>> zkus(2)
    Ne, moje číslo je jiné...
    >>> zkus(5)
    Hurá, uhádl jsi!
    >>>
    ```

Řešení:

```
import random

cislo = random.randint(1, 5)

def zkus(n):
    if n == cislo:
        print('Hurá, uhádl jsi!')
    else:
        print('Ne, moje číslo je jiné...')

print('Myslím si číslo od 1 do 5. Zkus ho uhádnout...')
```

Příkaz přiřazení `cislo = random.randint(1, 5)` jsme zapsali ještě před definici podprogramu `zkus`, čím jsme chtěli vyjádřit, že proměnná `cislo` je globální a bude se používat v podprogramu `zkus`. Program jsme mohli zapsat i následujícím způsobem a fungoval by stejně:

```
import random

def zkus(n):
    if n == cislo:
        print('Hurá, uhádl jsi!')
    else:
        print('Ne, moje číslo je jiné...')

cislo = random.randint(1, 5)
print('Myslím si číslo od 1 do 5. Zkus ho uhádnout...')
```

15* Vylepši předchozí program tak, aby nám podprogram `zkus` poradil, zda je hádané číslo větší nebo menší, než jsme tipnuli:

```
========== RESTART ==========
Myslím si číslo od 1 do 5. Zkus ho uhádnout...
>>> zkus(3)
Ne, moje číslo je menší...
>>> zkus(1)
Ne, moje číslo je větší...
>>> zkus(2)
Hurá, uhádl jsi!
>>>
```

Řešení:

```
import random

cislo = random.randint(1, 5)

def zkus(n):
    if n == cislo:
        print('Hurá, uhádl jsi!')
    else:
        if n < cislo:
            print('Ne, moje číslo je větší...')
        else:
            print('Ne, moje číslo je menší...')

print('Myslím si číslo od 1 do 5. Zkus ho uhádnout...')
```

16. Vytvoř nový program `kviz.py`, který bude fungovat jako jednoduchý kvíz na sčítání čísel. Počítač na začátku vygeneruje dvě náhodná čísla z rozsahu od `1` do `10`, vypíše je a my musíme odpovědět tím, že z příkazového řádku zavoláme podprogram `over`. Počítač poté zkontroluje, zda byla naše odpověď správná, nebo ne:

```
========== RESTART ==========
Kolik je 10 + 7 ?
>>> over(5)
Nesprávně, mělo to být 17
>>>
========== RESTART ==========
Kolik je 4 + 9 ?
>>> over(13)
Správně
>>>
```

Řešení:

```
import random

a = random.randint(1, 10)
b = random.randint(1, 10)

def over(soucet):
    if soucet == a + b:
        print('Správně')
    else:
        print('Nesprávně, mělo to být', a + b)
print('Kolik je', a, '+', b, '?')
```

Z příkazového řádku je nutné volat podprogram `over` s parametrem odpovídajícím vypočítanému výsledku. Pokud by některý žák do příkazového řádku zapsal jen `over` bez kulatých závorek a parametru, Python odpoví nepříliš srozumitelnou informací `<function over at 0x0362C660>`.

Někteří žáci mohou mít naopak tendenci do příkazového řádku psát pouze výsledek výpočtu (např. číslo `17`) místo toho, aby volali metodu `over` s parametrem odpovídajícím danému výsledku. Python v takovém případě nevypíše informaci o správnosti výsledku, ale zobrazí pouze žákem zapsané číslo.

V případě, že jsme žáky již dříve seznámili s konstrukcí `input`, můžeme řešení této úlohy zapsat i lépe, a to dokonce bez použití parametru:

```
import random

def over():
    a = random.randint(1, 10)
    b = random.randint(1, 10)
    print('Kolik je', a, '+', b, '?')
    soucet = int(input('? '))
    if soucet == a + b:
        print('Správně')
    else:
        print('Nesprávně, mělo to být', a + b)

over()
```

17* Je potřeba prozkoumat, jak často padne 6, když mnohokrát házíme hrací kostkou. Vytvoř nový program `stesti.py` a v něm podprogram `stesti` s parametrem `n`, který nasimuluje `n` hodů běžnou hrací kostkou. Podprogram `n`-krát vygeneruje náhodné číslo od 1 do 6, a když padne šestka, zvýší počítadlo o `1`. Podprogram na závěr vypíše zprávu ve tvaru:

```
Pravděpodobnost výhry při 10 hodech: 0.2
```

Nech podprogram vypsat, jaké budou pravděpodobnosti výhry při `10`, `100`, `1000`, `10000`, `100000` hodech.

Řešení:

```
import random

def stesti(n):
    pocet = 0
    for i in range(n):
        hod = random.randint(1, 6)
        if hod == 6:
            pocet = pocet + 1
    print('Pravděpodobnost výhry při', n, 'hodech:',
        pocet / n)

stesti(10)
stesti(100)
stesti(1000)
stesti(10000)
stesti(100000)
```

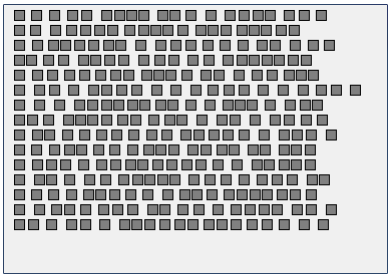
Pokud žáci spustí svůj program opakovaně, může je překvapit, že se pravděpodobnost výhry při určitém počtu hodů mezi jednotlivými pokusy liší. V našem případě vyšla pravděpodobnost při 100 000 hodech poprvé `0.16819`, podruhé `0.16511` a potřetí `0.16691`. Je to dáno tím, že je výpočet založen na četnostech náhodně generovaných čísel. Se zvyšujícím se počtem hodů by se však vypočítaná pravděpodobnost výhry měla blížit hodnotě teoretické pravděpodobnosti výhry, která je jedna šestina, tedy cca 0.166667.

18* Dlaždič měl rovnoměrně poskládat dlažební kostky do jedné řady. Měl však dobrou náladu a mezi kostkami nechával náhodné mezery. Vytvoř nový program `dlazebni_kostky.py` a v něm podprogram `rada` s parametrem `y`, který nakreslí do grafické plochy vedle sebe 20 kostek. Kostky kresli jako šedé čtverečky velikosti `10` x `10`. První kostka má x-ovou souřadnici levého horního rohu 10 a každá další ji má větší o náhodné číslo z rozsahu od 12 do 20. Y-ovou souřadnici levého horního rohu mají všechny kostky stejnou; tato souřadnice je dána parametrem `y`.

Zavolej podprogram `rada` následujícím způsobem:

```
for i in range(15):
    rada(5 + i * 15)
```

Jestli jsi postupoval správně, výsledek by měl vypadat podobně jako na následujícím obrázku:



Řešení:

```
import tkinter
import random

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

def rada(y):
    x = 10
    for i in range(20):
        canvas.create_rectangle(x, y, x + 10, y + 10,
            fill='gray')
        x = x + random.randint(12, 20)

for i in range(15):
    rada(5 + i * 15)
```