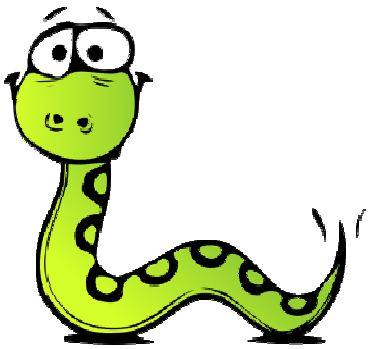
# Programování v jazyce Python pro střední školy

Metodický list pro učitele

Lekce 17 – Větvení a konstrukce



Andrej Blaho

Ľubomír Salanci

Václav Šimandl

## Cíle lekce

- Naučit se sestavit výpočet hodnoty proměnné pomocí příkazu větvení
- Seznámit se s kombinací cyklu a v něm vnořeného příkazu větvení
- Naučit se využívat proměnnou cyklu v podmínce a větvích příkazu větvení
- Naučit se vytvářet podprogramy, které obsahují příkaz větvení

## Osvojená syntaktická pravidla

- Vícenásobné odsazování vnořené konstrukce
- Dodržování známých syntaktických pravidel i při kombinování programových struktur

## Průběh výuky

Začínáme úlohou na opakování:

1. Kamarádku pozdravíš neformálně „`Ahoj`“, ale starší lidi pozdravíš formálněji, například „`Dobrý den`“. Napiš program `pozdravy_podle_veku.py`, ve kterém do proměnné `vek` přiřadíš věk člověka. Potom použij příkaz větvení na to, aby se program podle věku rozhodl, který z uvedených dvou pozdravů vypíše. Otestuj, jaké pozdravy se vypisují pro různé hodnoty proměnné `vek`.

Řešení:

```
vek = 10
if vek < 18:
    print('Ahoj')
else:
    print('Dobrý den')
```

2. Na brigádě ve stánku se zmrzlinou dostaneš mzdu podle následujícího pravidla:
   - když budeš pracovat méně než 10 hodin, vyděláš si 80 korun za hodinu,
   - jinak si vyděláš 100 korun za hodinu.

   Vytvoř nový program, ve kterém do proměnné `hodin` přiřaď počet hodin, které jsi odpracoval. Potom pomocí příkazu větvení vypiš, kolik si vyděláš. Program by měl vypsat:

| | |
|---|---|
| `Vyděláš si 560 korun.` | `Vyděláš si 1200 korun.` |
| pro `hodin = 7` | pro `hodin = 12` |

V zadání této úlohy, na rozdíl od většiny předchozích, není určeno, jak mají žáci svůj program pojmenovat. Žáci by tedy měli sami vymyslet smysluplný název programu. Pokud to uznáme za vhodné, můžeme s nimi o názvu diskutovat. Jako vhodné se jeví například názvy `brigada.py` nebo `vydelek.py`. Nevhodné jsou naopak obecné názvy nevystihující obsah úlohy, například `vetveni.py` nebo `program.py`.

Řešení:

```
hodin = 12
if hodin < 10:
    print('Vyděláš si', hodin * 80, 'korun.')
else:
    print('Vyděláš si', hodin * 100, 'korun.')
```

3. Předchozí úloha se dá řešit i takto:

```
hodin = 20
if ..........:
    mzda = ..........
else:
    mzda = ..........
print('Vyděláš si', mzda, 'korun.')
```

Doplň namísto vytečkovaných částí správné výrazy. Ověř, že program správně počítá mzdu pro různé hodnoty proměnné `hodin`.

Řešení:

```
hodin = 20
if hodin < 10:
    mzda = hodin * 80
else:
    mzda = hodin * 100
print('Vyděláš si', mzda, 'korun.')
```

Místo přiřazení `hodin = 20` můžeme opět využít příkaz `input`, například

```
hodin = int(input('Zadej počet odpracovaných hodin: '))
```

Z pohledu Pythonu je lhostejné, zda bude použito řešení ze 2. úlohy nebo z 3. úlohy. My preferujeme zápis uvedený jako řešení 3. úlohy, neboť jej považujeme za čitelnější.

Následují dvě velmi podobné úlohy trénující schopnost rozdělit řešení do dvou na sobě nezávislých větví a následně jej zapsat pomocí příkazu větvení.

4. Mobilní operátor *Vegafon* počítá platby za přenesená data podle následujících pravidel:

- když za den přeneseš méně než 10 megabajtů dat, zaplatíš za každý megabajt 2 koruny,
- jinak zaplatíš za celý den 20 korun.

Napiš program `mobilni_data.py`, ve kterém do proměnné `megabajty` přiřadíš počet přenesených megabajtů dat za jeden den. Použij příkaz větvení na to, abys do proměnné `cena` přiřadil vyúčtovanou cenu. Nakonec tuto cenu vypiš. Výpis může vypadat například takto:

| | |
|---|---|
| `Zaplatíš 12 korun.` | `Zaplatíš 20 korun.` |
| pro `megabajty = 6` | pro `megabajty = 20` |

Řešení:

```
megabajty = 6
if megabajty < 10:
    cena = megabajty * 2
else:
    cena = 20
print('Zaplatíš', cena, 'korun.')
```

5. Mobilní operátor *Zodrafon* počítá platby za přenesená data podle odlišných pravidel:

- když za den přeneseš méně než 10 megabajtů, zaplatíš za každý megabajt 1 korunu,
- jinak zaplatíš 10 korun a k tomu za každý megabajt nad limit 10 megabajtů 3 koruny.

Napiš program `mobilni_data2.py`, který počítá a vypisuje cenu podle těchto pravidel, a ověř, zda funguje správně. Program by měl například vypsat:

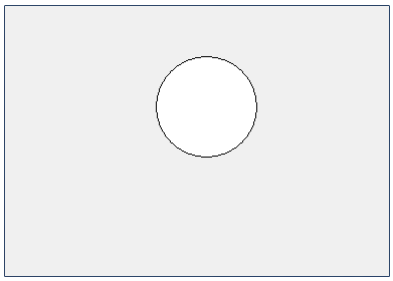
| | |
|---|---|
| `Zaplatíš 6 korun.` | `Zaplatíš 40 korun.` |
| pro `megabajty = 6` | pro `megabajty = 20` |

Cena pro více než 10 megabajtů má dvě složky – vždy zaplatíme 10 korun a megabajty nad 10 (jejich počet je `megabajty-10`) se počítají po 3 korunách, tedy za ně zaplatíme `(megabajty-10) * 3`.

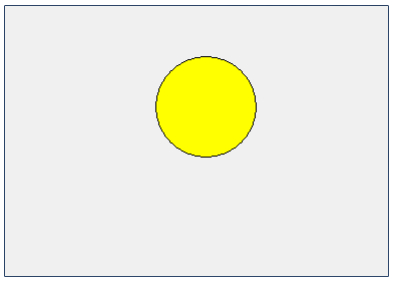
Celé řešení lze zapsat například takto:

```
megabajty = 20
if megabajty < 10:
    cena = megabajty * 1
else:
    cena = 10 + (megabajty - 10) * 3
print('Zaplatíš', cena, 'korun.')
```

6. Vytvoř program `den_noc.py`, který podle zadaného času nakreslí do grafické plochy slunce nebo měsíc. Do proměnné `cas` přiřaď počet hodin. Použij příkaz větvení na to, aby se pro `cas < 8` kreslil měsíc jako bílý kruh, jinak se kreslilo slunce jako žlutý kruh. Poloměr kruhu nechť je v obou případech `50` a střed kruhu má souřadnice [`200`, `100`]. Program by měl například nakreslit:



pro `cas = 4`



pro `cas = 14`

Řešení:

```
import tkinter

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

cas = 4
if cas < 8:
    canvas.create_oval(200 - 50, 100 - 50, 200 + 50,
        100 + 50, fill='white')
else:
    canvas.create_oval(200 - 50, 100 - 50, 200 + 50,
        100 + 50, fill='yellow')
```

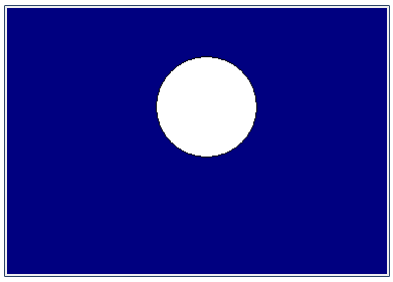
Alternativně můžeme řešení této úlohy zapsat podobným způsobem jako řešení předchozích úloh, což lze rozebrat i se žáky. V obou větvích příkazu větvení jsou totiž téměř stejné příkazy, které se liší jen v barvě výplně `fill=`. V takových případech můžeme zvolit následující použití větvení: ve větvích příkazu větvení se do proměnné `barva` přiřadí buď bílá barva (`'white'`), nebo žlutá barva (`'yellow'`) v závislosti na podmínce `cas < 8`. Až následně, po skončení příkazu větvení, se použije příkaz `create_oval`, pomocí něhož se nakreslí bílý, nebo žlutý kruh v závislosti na hodnotě proměnné `barva`.

Uvedené řešení lze zapsat takto:

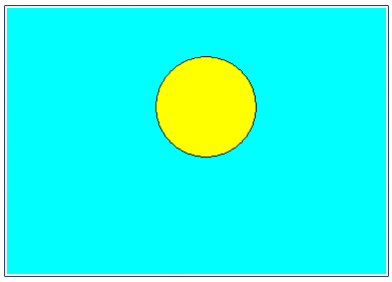
```
import tkinter

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

cas = 4
if cas < 8:
    barva = 'white'
else:
    barva = 'yellow'
canvas.create_oval(200 - 50, 100 - 50, 200 + 50, 100 + 50,
    fill=barva)
```

Cílem další úlohy je pochopit, že větev `if` i `else` může obsahovat více příkazů.

7. Do předchozího řešení doplň kreslení pozadí – měsíc se nakreslí na tmavomodré pozadí, slunce na světlemodré pozadí:



pro `cas = 4`



pro `cas = 14`

Stačí, když do každé větve přidáš příkaz na kreslení velkého obdélníku, který překryje celou grafickou plochu:

```
if ..............................:
    canvas.create_rectangle(..............................., fill='navy')
    canvas.create_oval(..............................., fill='white')
else:
    canvas.create_rectangle(..............................., fill='cyan')
    canvas.create_oval(..............................., fill='yellow')
```

Příkazy, které tvoří tělo větví `if` i `else`, nech odsazené od kraje (Python tam automaticky vložil 4 mezery)

Řešení:

```
import tkinter

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

cas = 4
if cas < 8:
    canvas.create_rectangle(0, 0, 380, 280, fill='navy')
    canvas.create_oval(200 - 50, 100 - 50, 200 + 50,
        100 + 50, fill='white')
else:
    canvas.create_rectangle(0, 0, 380, 280, fill='cyan')
    canvas.create_oval(200 - 50, 100 - 50, 200 + 50,
        100 + 50, fill='yellow')
```

Pokud jsme žákům v předchozích lekcích prozradili, že grafickou plochu je možné už při její tvorbě zabarvit libovolnou barvou pomocí parametru `bg=` , tedy například:

```
canvas = tkinter.Canvas(bg='white')
```

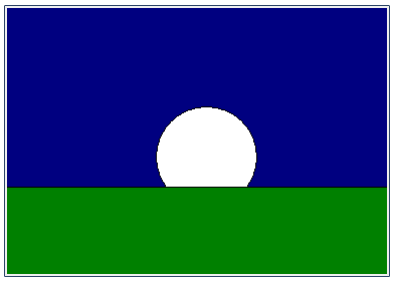
můžeme jim prozradit ještě jednu konstrukci, kterou lze změnit barvu pozadí grafické plochy nejen při její tvorbě, ale v libovolném okamžiku. Místo velkého obdélníku pro modré pozadí můžeme zapsat:

```
canvas['bg'] = 'navy'
```

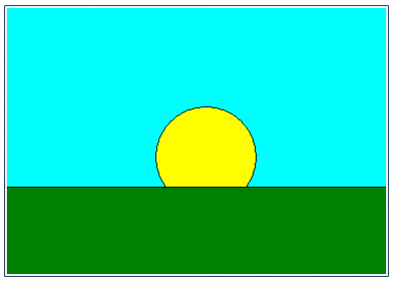
Celý program by nyní mohl vypadat následovně (všimněme si použití konstrukce `input` na začátku programu a také použití proměnné `barva`):

```
import tkinter

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

cas = int(input('Zadej aktuální čas v hodinách: '))
if cas < 8:
    canvas['bg'] = 'navy'
    barva = 'white'
else:
    canvas['bg'] = 'cyan'
    barva = 'yellow'
canvas.create_oval(200 - 50, 100 - 50, 200 + 50, 100 + 50,
    fill=barva)
```

8. Uprav předchozí program tak, aby se nejdříve do proměnných `x`, `y` přiřadily souřadnice středu kruhu a ty se potom použily v příkazech `create_oval`. Kromě toho přidej na úplný konec programu i kreslení zeleného obdélníku, který bude představovat krajinu. Potom program pro `x = 200` a `y = 150` bude kreslit scény jako na následujících obrázcích:



pro `cas = 4`



pro `cas = 14`

Horní okraj zeleného obdélníku umísti na y-ovou souřadnici 180.

Řešení:

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

cas = 14
x = 200
y = 150
if cas < 8:
    canvas.create_rectangle(0, 0, 380, 180, fill='navy')
    canvas.create_oval(x - 50, y - 50, x + 50, y + 50,
        fill='white')
else:
    canvas.create_rectangle(0, 0, 380, 180, fill='cyan')
    canvas.create_oval(x - 50, y - 50, x + 50, y + 50,
        fill='yellow')
canvas.create_rectangle(0, 180, 380, 280, fill='green')
```

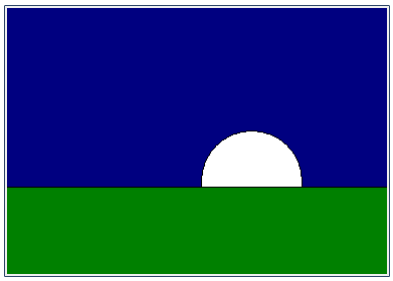
Alternativně by bylo možné řešení zapsat i následujícím způsobem:

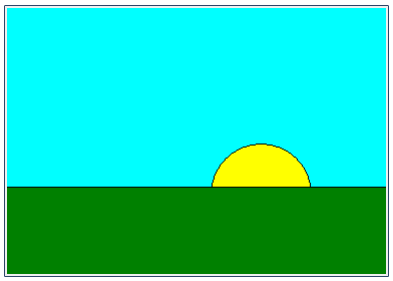
```
import tkinter
canvas = tkinter.Canvas()
canvas.pack()

cas = 14
x = 200
y = 150
if cas < 8:
    barva_obloha = 'navy'
    barva_kruh = 'white'
else:
    barva_obloha = 'cyan'
    barva_kruh = 'yellow'

canvas.create_rectangle(0, 0, 380, 180, fill=barva_obloha)
canvas.create_oval(x - 50, y - 50, x + 50, y + 50,
    fill=barva_kruh)
canvas.create_rectangle(0, 180, 380, 280, fill='green')
```

9. Vylepši předchozí program tak, aby fungoval jako náhodný generátor krajinek – přiřaď na začátku do proměnných `x`, `y`, `cas` náhodná čísla:

   - `x` z rozsahu `100, 300`
   - `y` z rozsahu `100, 200`
   - `cas` z rozsahu `0, 16`

   Program několikrát spusť a sleduj, zda se krajinky vytváří dle tvého očekávání:





Řešení:

```
import tkinter
import random

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

x = random.randint(100, 300)
y = random.randint(100, 200)
cas = random.randint(0, 16)
if cas < 8:
    canvas.create_rectangle(0, 0, 380, 180, fill='navy')
    canvas.create_oval(x - 50, y - 50, x + 50, y + 50,
        fill='white')
else:
    canvas.create_rectangle(0, 0, 380, 180, fill='cyan')
    canvas.create_oval(x - 50, y - 50, x + 50, y + 50,
        fill='yellow')
canvas.create_rectangle(0, 180, 380, 280, fill='green')
```

V této úloze se počítač rozhoduje, kterou větev příkazu `if ... else ...`, dále vykoná, podle náhodně vygenerované hodnoty proměnné `cas`. To může být pro některé žáky matoucí. Pokud to bude potřeba, je možné s nimi individuálně diskutovat, jakých hodnot může proměnná `cas` nabývat a jaká větev příkazu `if ... else ...`, se pro konkrétní hodnotu vykoná.

Alternativně by bylo možné řešení zapsat za použití proměnných `barva_obloha` a `barva_kruh` podobně jako v řešení 8. úlohy.

10. Následující program vypisuje denní harmonogram:

```
for i in range(8):
    print(i, 'ještě spím')
for i in range(6):
    print(8 + i, 'jsem ve škole')
```

Diskutuj se spolužákem, co konkrétně program vypíše. Poté program spusť v Pythonu a zkontroluj, zda byla Tvá domněnka správná.

Předchozí program se dá zapsat i takto, jen pomocí jediného cyklu:

```
for i in range(14):
    if i < 8:
        print(i, 'ještě spím')
    else:
        print(i, 'jsem ve škole')
```

tělo cyklu – je potřeba jej odsadit od kraje

tyto příkazy je potřeba odsadit od kraje ještě více

Tělo cyklu `for` obsahuje jeden vnořený příkaz větvení `if else`. Vyzkoušej, jak funguje tato verze.

Obě varianty programu vypíší:

```
0 ještě spím
1 ještě spím
2 ještě spím
3 ještě spím
4 ještě spím
5 ještě spím
6 ještě spím
7 ještě spím
8 jsem ve škole
9 jsem ve škole
10 jsem ve škole
11 jsem ve škole
12 jsem ve škole
13 jsem ve škole
```

Takovýto mechanismus **vnořování konstrukcí** funguje ve všech programovacích jazycích. V jazyce Python je nutné velmi důsledně dodržovat odsazení vnořených příkazů od kraje. Představme si, že chceme v cyklu odsadit vnořené větvení, které jsme zatím úmyslně zapsali na stejné úrovni jako příkaz for cyklu:

```
for i in range(14):

if i < 8:
    print(i, 'ještě spím')
else:
    print(i, 'jsem v škole')
```

Abychom všechny příkazy, které chceme vnořit do for cyklu, odsadili najednou (a správně), nejlépe to uděláme tak, že je nejprve všechny vybereme a potom jediným stiskem klávesy *Tab* odsadíme vpravo. Výsledek pak bude vypadat stejně jako v následující ukázce:

```
for i in range(14):
    if i < 8:
        print(i, 'ještě spím')
    else:
        print(i, 'jsem v škole')
```

Tento postup můžeme žákům prozradit hned, nebo je nejprve necháme se při plnění jednotlivých úloh „trápit“ a toto vylepšení jim ukážeme později.

11. Vytvoř nový program `chudy_bohaty.py`. V něm podobně jako v předchozí úloze použij cyklus s vnořeným větvením a vypiš:

```
Mám 0 korun, jsem chudý
Mám 10 korun, jsem chudý
Mám 20 korun, jsem chudý
Mám 30 korun, jsem chudý
Mám 40 korun, jsem chudý
Mám 50 korun, jsem bohatý
Mám 60 korun, jsem bohatý
Mám 70 korun, jsem bohatý
Mám 80 korun, jsem bohatý
Mám 90 korun, jsem bohatý
```

Proměnná cyklu se mění od `0` do `9` a vypisuje se desetinásobek hodnoty této proměnné.

Řešení:

```
for i in range(10):
    if i < 5:
        print('Mám', i * 10, 'korun, jsem chudý')
    else:
        print('Mám', i * 10, 'korun, jsem bohatý')
```

12. Víš, pro která čísla `n` platí, že $n^2$ je menší než `5 * n` ? Napiš program `nasobky_peti.py`, který pro všechna čísla od `0` do `10` otestuje tento vztah a vypíše o tom patřičnou informaci, například:

```
...
16 je menší než 20
25 je větší nebo rovno 25
36 je větší nebo rovno 30
...
```

Co tvůj program zjistil? Pro která `n` onen vztah platí?

Řešení:

```
for n in range(11):
    if n * n < n * 5:
        print(n * n, 'je menší než', n * 5)
    else:
        print(n * n, 'je větší nebo rovno', n * 5)
```

Očekáváme, že žáci program spustí a získají následující výsledky:

```
0 je větší než nebo rovno 0
1 je menší než 5
4 je menší než 10
9 je menší než 15
16 je menší než 20
25 je větší nebo rovno 25
36 je větší nebo rovno 30
49 je větší nebo rovno 35
64 je větší nebo rovno 40
81 je větší nebo rovno 45
100 je větší nebo rovno 50
```

Podle vypsaných hodnot by měli žáci odpovědět, že vztah platí pro

`n = 0, 5, 6, 7, 8, 9` a `10`.

Se žáky lze diskutovat, proč vztah platí i pro `0`.

Ve vzorovém řešení této úlohy jsme, na rozdíl od předchozích úloh, proměnnou cyklu označili jako `n`, nikoliv jako `i`. Jak je z ukázky zřejmé, v Pythonu není nutné proměnnou cyklu označovat vždy pouze jako `i`. Pokud se budeme řídit pravidly pro pojmenování „běžných“ proměnných (viz 2. lekce), její název může být prakticky libovolný. Žáci však tuto proměnnou mohou stejně jako v předchozích úlohách označit jednoduše jako `i`.

13. Vytvoř nový program `koralky_na_niti.py`, ve kterém pomocí jediného cyklu s vnořeným větvením nakresli 15 korálků jako na obrázku níže. Prvních 8 korálků bude červených a zbylých 7 modrých.

Řešení:

```
import tkinter

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

y = 100
for i in range(15):
    x = 50 + i * 20
    if i < 8:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill='red')
    else:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill='blue')
```

I v této úloze můžeme využít proměnnou `barva`, do které se v příkazu větvení přiřadí buď červená, nebo modrá barva. Až následně, po skončení tohoto příkazu větvení, se nakreslí pomocí příkazu `create_oval` barevný kruh v závislosti na proměnné `barva`.

Naznačené řešení lze zapsat například takto:

```
import tkinter

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

y = 100
for i in range(15):
    x = 50 + i * 20
    if i < 8:
        barva = 'red'
    else:
        barva = 'blue'
    canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
        fill=barva)
```

14* Uprav předchozí program tak, aby se prvních 8 korálků barvilo náhodně pomocí

```
random.choice(['red', 'yellow'])
```

a zbylých 7 pomocí

```
random.choice(['blue', 'green']).
```

Výsledek může vypadat například jako na následujícím obrázku:

Řešení:

```
import tkinter
import random

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

y = 100
for i in range(15):
    x = 50 + i * 20
    if i < 8:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill=random.choice(['red', 'yellow']))
    else:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill=random.choice(['blue', 'green']))
```

Výběr barvy lze zapsat i následujícím způsobem:

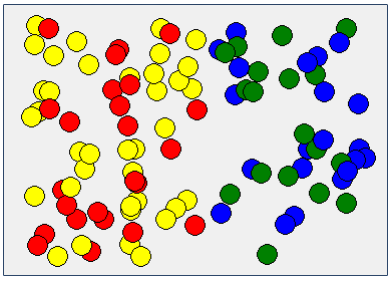
```
for i in range(15):
    x = 50 + i * 20
    if i < 8:
        barva = random.choice(['red', 'yellow'])
    else:
        barva = random.choice(['blue', 'green'])
    canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
        fill=barva)
```

15* Vytvoř nový program `rozsypane_koralky.py` a v něm definuj podprogram `koralek`. V tomto podprogramu vygeneruj náhodné pozice `x`, `y` pro střed korálku. Program potom podle x-ové souřadnice nakreslí:

- červený nebo žlutý korálek, pokud je `x < 200`;
- modrý nebo zelený korálek pro ostatní čísla.

Využij kreslení korálků z předchozí úlohy.

Když zavoláš podprogram `koralek` v cyklu stokrát, můžeš dostat například takovýto obrázek:



Řešení:

```
import tkinter
import random

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

def koralek():
    x = random.randint(20, 360)
    y = random.randint(20, 250)
    if x < 200:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill=random.choice(['red', 'yellow']))
    else:
        canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
            fill=random.choice(['blue', 'green']))

for i in range(100):
    koralek()
```

Podprogram `koralek` lze zapsat i následujícím způsobem:

```
def koralek():
    x = random.randint(20, 360)
    y = random.randint(20, 250)
    if x < 200:
        barva = random.choice(['red', 'yellow'])
    else:
        barva = random.choice(['blue', 'green'])
    canvas.create_oval(x - 10, y - 10, x + 10, y + 10,
        fill = barva)
```