PROGRAMOVÁNÍ V JAZYCE PYTHON

INGRID NAGYOVÁ

ČÍSLO OPERAČNÍHO PROGRAMU: CZ.1.07
NÁZEV OPERAČNÍHO PROGRAMU:
VZDĚLÁVÁNÍ PRO KONKURENCESCHOPNOST
ČÍSLO PRIORITNÍ OSY: 7.1
ČÍSLO OBLASTI PODPORY: 7.1.3

CHYTŘÍ POMOCNÍCI VE VÝUCE ANEB VYUŽÍVÁME ICT JEDNODUŠE A KREATIVNĚ

REGISTRAČNÍ ČÍSLO PROJEKTU: CZ.1.07/1.3.00/51.0009

OSTRAVA 2014

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky

Recenzent: ing. Aleš Oujezdský, Ph.D.

Název: Grafické a animační techniky
Autor: Ingrid Nagyová
Vydání: první, 2014
Počet stran: 56

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

POUŽITÉ GRAFICKÉ SYMBOLY

Průvodce studiem

Cíl kapitoly

Klíčová slova

Čas na prostudování kapitoly

Kontrolní otázky

Shrnutí

Korespondenční úkol

Doporučená literatura

Řešený příklad

Otázky k zamyšlení

Úlohy k textu

Obsah

Slovo úvodem

Programovací jazyk Python získává ve světě stále větší oblibu. Je to jistě dáno jeho vlastnostmi, a to zejména nezávislosti na platformě, která vyplývá z toho, že se jedná o interpretovaný jazyk. Velkou předností Pythonu pro výuku je struktura jeho zápisu vyžadující striktně odsazení jistých částí programu. Velkou výhodou Pythonu je také využití nejmodernějších prvků programování – jedná se o objektově orientovaný jazyk, všechny proměnné zde jsou referenční, tj. není nutné udávat jejich typ.

Studijní text, který začínáte číst, je určen především pro učitele, kteří chtějí svoji výuku obohatit a doplnit o prvky programování v moderním prostředí. Studijní text předpokládá jistou znalost základů algoritmizace – znalost principů opakování a větvení, hlubší znalosti a zkušenosti nejsou potřebné. Text vede studujícího ke zvládnutí základů jazyka Python od úplného začátku. Orientuje se zejména na grafické aplikace a práci s grafickými objekty. Konstruovaný obrázek poskytuje totiž rychlou vizuální zpětnou vazbu a může být dobrou motivací ke konstrukci dalších programů, zaměřených více abstraktně. Tato forma výuky je vhodná také pro žáky na základních a středních školách.

Programování není jednoduchou oblastí a jeho zvládnutí vyžaduje zejména čas zkoušet, hledat řešení apod. Ne jinak to jistě bude i v Pythonu. Přejeme Vám, aby byl tento čas pro Vás přínosem, aby Vám poskytl zábavu a poučení. Věříme, že právě tak se mohou budovat důležité dovednosti a schopnosti využitelné následně nejenom v programování.

Hodně úspěchů přeje

Autorka

1 Seznámení se s jazykem Python

Cíl kapitoly

Po nastudování této kapitoly byste měli být schopni:

- nainstalovat prostředí jazyka Python a zvládnout základní orientaci v tomto prostředí;
- používat dva režimy práce – interaktivní režim a programovací režim;
- vyjmenovat a použít základní datové typy – int, float a str;
- popsat způsob reprezentace proměnných v paměti počítače.

Klíčová slova

Python, instalace, interaktivní režim, programovací režim, datový typ, proměnná, reprezentace proměnné v paměti, přetypování.

Čas na prostudování

2 hodiny

Průvodce studiem

Programování není dnes již nezbytností při práci s počítačem, přesto je hodně oblíbené a pro žáky zahalené rouškou tajemnosti a nedosažitelnosti. Programování není jednoduchá činnost, jeho dokonalé zvládnutí není jenom otázkou znalostí, ale zejména praktických dovedností a zkušeností budovaných v průběhu delší doby. Přesto nebo právě proto lze výuku programování doporučit zejména pro zájmové kroužky již na základní škole.

V této kapitole se zaměříme na úvodní seznámení se s jazykem Python, na jeho instalaci a základní principy práce. Naučíme se používat základní datové typy a vysvětlíme si, co to je proměnná a jak je reprezentována v paměti počítače.

1.1 Charakteristika jazyka Python

Python je moderní programovací jazyk, jehož popularita a využitelnost neustále roste. Jeho autorem je Guido van Rossum, který ho vymyslel v roce 1989 a od té doby se na jeho rozšiřování a využití spolupodílí. Python je využíván ve velkých mezinárodních společnostech jako Google, YouTube, Mozillla apod.

Pyhton se za dobu své existence stal oblíbeným programovacím jazykem také pro výuku. Na řadě špičkových univerzit je využíván jako úvodní jazyk pro výuku, vzpomeňme například Massachusettský technologický institut, univerzitu v Berkeley, ale také například na Univerzitě Komenského v Bratislavě. Komunita lidí věnujících se programovacímu jazyku Pyhton existuje také v České republice, i když zde je jeho využití ve výuce zatím velice omezené. Informace v češtině lze najít na portálu http://python.cz/.

Python je objektově orientovaný programovací jazyk. Přesto lze při tvorbě programů využít i procedurální paradigma. Python má dobré vyjadřovací schopnosti, kód programu je krátký, přehledný a dobře čitelný. Je velice blízký přirozenému jazyku.

Na rozdíl od klasických programovacích jazyků využívaných pro výuku (Pascal, C/C++) je Python interpretovaný jazyk, tj. nevytváří se spustitelný kód ve tvaru spustitelného EXE souboru (v operačním systému Windows). Zapsaný kód programu je přímo interpretován interpretrem, který je nutné mít pro spuštění programu nainstalovaný v počítači. To umožňuje snadnou, rychlou a interaktivní úpravu programů při jejich ladění.

1.2 Instalace jazyka Python

Programovací jazyk Python je vyvíjen jako open source projekt, tj. je dostupný zdarma pro většinu platforem (Windows, Unix, Mac OS). Jednotlivé verze programu lze zdarma stáhnout na webových stránkách na adrese https://www.python.org/. Momentálně je k dispozici verze 3.4.2. Pro operační systém Windows stáhneme a spustíme instalační program ve formátu MSI.

Po úspěšné instalaci Pythonu spustíme IDLE (Python GUI), tzv. vývojové prostředí Pythonu. Objeví se okno s výpisem základních informací o nainstalované verzi a řádek začínající >>>, který slouží k zadávání příkazů.

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

1.3 Režimy práce

Vývojové prostředí Pythonu pracuje v několika různých režimech. Mezi základní režimy patří interaktivní a programovací režim.

1.3.1 Interaktivní režim

Základní režim, tzv. interaktivní režim, je automaticky přístupný po spuštění vývojového prostředí a umožňuje přímé zadávání příkazů do příkazového řádku označeného >>>. V interaktivním režimu se vše provádí v okně, které je označeno nápisem *Shell*.

Zadat můžeme například různé matematické výrazy s čísly.

```
>>> 1234
1234
>>> 1+2
3
>>> 8*9
72
>>> 100/7
14.285714285714286
>>> 0.1+0.2+0.3+0.4
1.0
>>> 9999999999*99999999999.
9.9999999989e+20
```

Ve výrazech můžeme využít základní matematické operace +, -, * (násobení) a /. V desetinných číslech se využívá desetinná tečka nebo se čísla zobrazují s exponentem (například 1e+20).

Při pokusech o zadávání znaků v interaktivním režimu vypíše Python chybu.

```
>>> python
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    python
NameError: name 'python' is not defined
```

Python zadané slovo nezná. Pro zadání znakových řetězců je potřeba tyto uzavřít do uvozovek nebo apostrofů.

```
>>> 'pyhton'
'pyhton'
>>> "dobrý den"
'dobrý den'
```

Řetězec musí končit stejným znakem (uvozovkou, apostrofem), jakým začal. Délka řetězce nesmí překročit jeden řádek. V řetězcích lze využít české znaky s diakritikou.

1.3.2 Programovací režim

V IDLE vytvoříme nový soubor výběrem volby *File – New File*. V otevřeném prázdném okně můžeme zadávat programové příkazy. Příkaz `print()` vypíše hodnoty v závorkách.

```
print(2+3)
print('Dobrý den!!!')
print('Výpočet je', 1+2+3+4+5)
```

Vytvořený program spustíme volbou *Run – Run Mudule* nebo klávesou *F5*. Před spuštěním programu budeme vyzváni k uložení souboru s programem. Soubor se ukládá ve formátu PY a lze ho spustit i mimo vývojové prostředí IDLE.

Při spuštění programu ve vývojovém prostředí IDLE je vše v okně *Shell* zapomenuto a restartováno (viz nápis RESTART). Poté jsou vypsány výsledky běhu programu, v našem případě jednotlivé výpisy dané příkazem `print()`.

```
>>> ========================== RESTART ==========================
>>>
5
Dobrý den!!!
Výpočet je 15
>>>
```

Spuštění programu mimo vývojové prostředí IDLE vyvolá otevření okna, v němž se zobrazí výsledky. Okno se po skončení běhu programu uzavře, a proto výpis nelze zahlédnout. Program proto doporučujeme ukončit příkazem `input()`, který vypíše řetězec v závorkách a následně čeká na vložení nového řetězce ukončeného klávesou *ENTER*. Vložený řetězec je výstupem (výsledkem) funkce `input()`. Původní program proto upravíme.

```
print(2+3)
print('Dobrý den!!!')
print('Výpočet je ', 1+2+3+4+5)
# čekání na zmáčknutí klávesy ENTER
input('Stlač ENTER')
```

Znak # uvozuje poznámku v programu. Řádek označený tímto znakem představuje poznámku programátora a při vykonávání programu je ignorován.

Funkci `input()` lze využít také k načítání vstupní hodnoty do programu. Hodnota se zadává v okně *Shell*.

```
>>> n=input("Zadej číslo ")
Zadej číslo 3
>>> print(n)
3
```

1.4 Základní datové typy a operace s nimi

Prozatím jsme se zabývali třemi typy proměnných – celými a desetinnými čísly a znaky. Tyto typy jsou po řadě označovány jako `int`, `float` a `str`. Typ dané hodnoty nebo proměnné zjistíme pomocí funkce `type()`.

```
>>> type(10)
<class 'int'>
>>> type(0.5)
<class 'float'>
>>> type('Dobrý den')
<class 'str'>
```

Jednotlivé datové typy umožňují různé operace s nimi. Číselné typy umožňují základní aritmetické operace `+`, `-`, `*` a `/`. V rámci celých čísel lze provádět pouze celočíselné dělení označované znakem `//`. Pro operaci určení zbytku po dělení používáme znak `%` a pro umocňování znak `**`.

```
>>> 13//2
6
>>> 13%2
1
>>> 5.5//2
2.0
>>> 5.5%2
1.5
>>> 2**10
1024
>>> 5.5**7
152243.5234375
```

Znakové řetězce lze spojovat pomocí operace `+`. Python umožňuje i vícenásobné zřetězení – zadává se znakem hvězdička *.

```
>>> ‚Janka‘+‘Danka‘
'JankaDanka'
>>> 10*'*'
'**********'
>>> '='*5
'====='
```

Úkol k textu

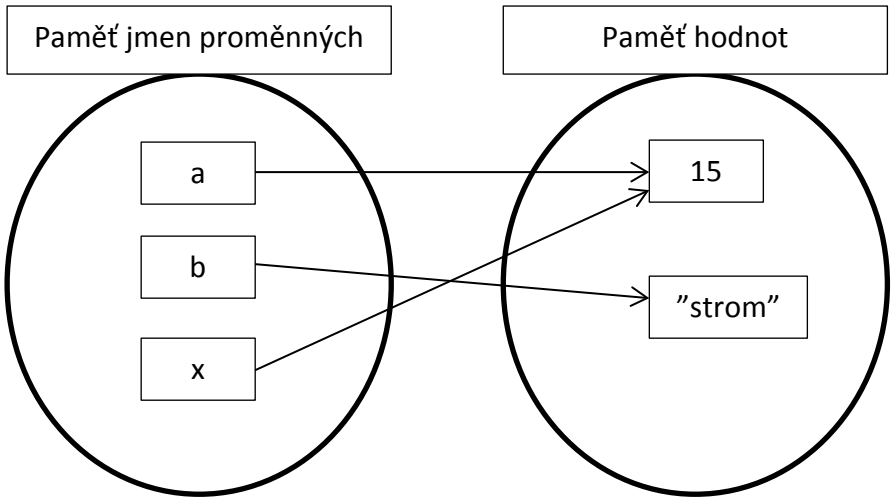
Otestujte interaktivní a programovací režim práce v prostředí jazyka Python. Vymyslete jednoduché příklady pro výpočty v interaktivním režimu. Stejné příklady zkuste otestovat také v programovacím režimu a výsledky vypište pomocí příkazu `print()`.

1.5 Reprezentace proměnných v paměti počítače

Proměnná je označena jménem, které se skládá z písmen, číslic a znaku podtržítko. Ve jménech proměnných závisí na velkých a malých písmenech. Pro přehlednost programu doporučujeme využívat v názvech proměnných pouze malá písmena.

Proměnná vzniká přiřazovacím příkazem, kdy se do proměnné přiřazuje její hodnota. V jazyce Python se do proměnné nepřiřadí přímo hodnota, ale pouze odkaz, reference na tuto hodnotu v paměti počítače. Pokud se v jisté části paměti nachází jména proměnných a v jiné části hodnoty, pak reference odkazují z části paměti pro jména do části hodnot. Každá proměnná může odkazovat pouze na jednu hodnotu, na danou hodnotu (v paměti hodnot) může odkazovat jedna nebo více proměnných.

```
>>> a=15
>>> b="strom"
>>> x=a
```



Při přiřazování hodnot proměnným se obecně postupuje následovně:

- vypočte se hodnota na pravé straně výrazu a zapíše se do paměti hodnot;
- upraví se reference z paměti jmen proměnných do paměti hodnot.

Příklad: Přiřazení hodnoty proměnné

```
>>> nedele=13
>>> nedele=nedele+7
```

Proměnná `nedele` ukazuje na hodnotu 13. Poté se vypočte nová hodnota 20 (druhý výraz na pravé straně) a vytvoří se v paměti hodnot. Nakonec se upraví odkaz z proměnné `nedele` na tuto novou hodnotu.

Při volbě jmen proměnných dodržujeme jistá pravidla:

- volíme jména srozumitelná, odpovídající významu použití proměnné v programu;
- vhodná jsou krátká srozumitelná pojmenování;
- předcházíme možným záměnám v pojmenování proměnných.

1.5.1 Přetypování

Proměnné a hodnoty tří základních typů `int`, `float` a `str` lze rozeznat pomocí funkce `type()`, která vrátí název daného typu. Kromě toho funguje mezi těmito typy možnost přetypování, tj. změna datového typu proměnné.

Funkce `int()` vrátí pro zadané reálné číslo jeho dolní celou část a zadaný řetězec, pokud obsahuje celé číslo, převede na toto číslo. Pro řetězec, který celé číslo neobsahuje, funkce `int()` vrátí chybu.

```
>>> int(12/7)
1
>>> int('3')
3
>>> int('12/7')
...
ValueError: invalid literal for int() with base 10: '12/7'
>>> int('ahoj')
…
ValueError: invalid literal for int() with base 10: 'ahoj'
```

Podobně funkce `float()` převede zadané celé číslo na číslo reálné (přidá nulovou desetinnou část) nebo převede řetězec, v němž je uloženo reálné číslo, na reálné číslo.

Funkce `str()`převede zadané celé nebo reálné číslo na příslušný řetězec.

Možnost přetypování proměnných nejčastěji využijeme v případě, kdy potřebujeme vypsat číslo do řetězce nebo naopak.

```
>>> cislo = 20
>>> retezec = 'číslo '+str(20)+' je hezké číslo'
>>> retezec
'číslo 20 je hezké číslo'
```

Shrnutí kapitoly

- Python je objektově orientovaný programovací jazyk, který si postupně razí cestu mezi nejpopulárnější programovací jazyky. Je to profesionální nástroj, který lze vhodně využít i při výuce základů programování.
- Prostředí jazyka Pyhton umožňuje práci ve dvou základních režimech – v interaktivním režimu a v programovacím režimu.
- Mezi základní datové typy využívané v jazyce Pyhton patří typ celá čísla (`int`), desetinná (reálná) čísla (`float`) a řetězcové proměnné (`str`).
- Proměnná je určena svým jménem. Proměnná je pouze referencí (odkazem) do paměti hodnot na konkrétní hodnotu. Proměnná vzniká přiřazením hodnoty do proměnné.
- Mezi základními datovými typy `int`, `float` a `str` existuje možnost přetypování.

Kontrolní otázky a úkoly:

1 Popište postup při instalaci a spuštění vývojového prostředí IDLE jazyka Python.

2 Charakterizujte interaktivní režim práce v Pythonu – jaké akce zde lze provádět a jakým způsobem.

3 Charakterizujte programovací režim práce v prostředí Python.

4 Vyjmenujte a charakterizujte základní datové typy jazyka Python a popište operace, které lze s těmito typy provádět.

5 Definujte pojem proměnné a popište principy reprezentace proměnných v paměti počítače.

6 Specifikujte pravidla pro přetypování základních datových typů proměnných.

Korespondenční úkol

Společně jsme vytvořili první program v jazyce Python. Dolaďte program v programovacím módu tak, aby byl funkční a nebyly v něm chyby. Smyslem úkolu je seznámit se se soubory, které je nutné odevzdat.

Vytvořený program uložte ve formátu PY a odevzdejte ke kontrole tutorovi.

2 Základní programové struktury

Cíl kapitoly

Po nastudování této kapitoly byste měli být schopni:

- kreslit obrázky pomocí objektu želva, tj. pomocí řízení pohybu kreslícího objektu;
- strukturovat program do podprogramů;
- vytvořit jednoduchý cyklus s pevným počtem opakování;
- použít opakování při kreslení symetrických obrazců;
- vytvořit podmíněný příkaz pro větvení programu podle dané podmínky;
- vytvořit jednoduché pole, i pole kreslících objektů želva a využít ho.

Klíčová slova

Objekt želva, grafická plocha, podprogram, funkce, cyklus s pevným počtem opakování, podmíněný cyklus, podmíněný příkaz, strukturovaný typ pole.

Čas na prostudování kapitoly

10 hodin

Průvodce studiem

Tvorba programů je náročný proces, kterého zvládnutí není jednoduché. Jedním z důvodů je vysoká abstrakce potřebného myšlení. V tomto procesu se ukazuje velice důležitým prostředkem názornost a vizualizace celého procesu. I když klasicky se často vyučuje programování na matematických příkladech, žáky může více zaujmout tvorba a kreslení obrázků.

V této kapitole se seznámíme se základními programovými strukturami. Naučíme se pracovat s objektem želva, který je vhodný pro kreslení obrázků a využijeme ho k procvičování nezbytných teoretických znalostí. Věříme, že vás a vaše žáky práce s objektem želva zaujme. Přejeme hodně úspěchů.

2.1 Objekt želva

V prostředí jazyka Python můžeme pro kreslení využít objekt želva (anglicky turtle). Želvička se prochází po grafické ploše, která se otvírá v samostatném okně. Želva je daná následujícími parametry:

- **směr (`heading()`)** – želva má tvar malé šipky, která ukazuje aktuální směr želvy;
- **pozice (`pos()`)** – x-ová a y-ová souřadnice želvy na grafické ploše. Střed grafické plochy se nachází uprostřed, přesně podle zvyklostí kartézského souřadného systému.

Modul pro práci s objektem želva je potřeba nejprve importovat příkazem `import turtle`. Následně musíme objekt želva vytvořit – `t=turtle.Turtle()`. Tím se otevře nové okno `Graphics`, želva se nachází uprostřed grafické plochy a je natočena vpravo (viz šipka, velikost úhlu 0).

```
>>> import turtle
>>> t=turtle.Turtle()
>>> print(t.pos())
(0.00,0.00)
>>> print(t.heading())
0.0
```

Směr natočení želvy se udává ve stupních měřených proti směru hodinových ručiček:

- východ (vpravo) – 0°;
- sever (nahoru) – 90°;
- západ (vlevo) – 180°;
- jih (dolů) – 270°;

Pro pohyb želvy využíváme následující příkazy:

Příkaz pro pohyb	Zkratka příkazu	Význam, funkce příkazu
`t.forward(d)`	`t.fd(d)`	Posun želvy dopředu o d bodů (pixelů)
`t.back(d)`	`t.bk(d)`	Posun želvy dozadu (couvání) o d pixelů
`t.right(uhel)`	`t.rt(uhel)`	Otočí želvu o uhel stupňů vpravo, po směru hodinových ručiček
`t.left(uhel)`	`t.lt(uhel)`	Otočí želvu o uhel stupňů vlevo, proti směru hodinových ručiček
`t.clear()`		Smaže grafickou plochu a želvu nechá na místě, kde se nacházela
`t.reset()`		Smaže grafickou plochu a želvu umístí na počátek – do středu souřadného

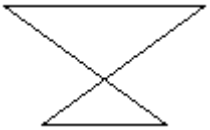
		systému natočenou na východ
`t.penup()`	`t.pu()`	Zvedne kreslící pero, želva se dále pohybuje bez kreslení
`t.pendown()`	`t.pd()`	Položí kreslící pero, želva se dále pohybuje s kreslením

V programech lze využívat pro zkrácení zápisu pouze zkratky jednotlivých příkazů.

Příklady: Kreslení pomocí želvy

Pro kreslení použijeme programovací režim. Příklady lze zkoušet také v interaktivním režimu, v němž je ovšem při chybách nutné znovu přepisovat celý program.

```
import turtle
t=turtle.Turtle()
t.lt(36)
t.fd(100)
t.rt(36)
t.bk(100)
t.rt(36)
t.fd(100)
t.rt(144)
t.fd(63)
```



Želva při svém pohybu vykreslí „amforu“ – viz obrázek vpravo.

Pokud část programu pro pohyb želvy uzavřeme mezi příkazy `begin_fill()` a `end_fill()`, amfora se vybarví aktuálně nastavenou barvou výplně, tj. černou barvou.

```
t.begin_fill()
t.lt(36)
t.fd(100)
t.rt(36)
t.bk(100)
t.rt(36)
t.fd(100)
t.rt(144)
t.fd(63)
t.end_fill()
```

2.2 Podprogramy

Vybrané části programu lze pojmenovat a využít pak v programu opakovaně. Pojmenované části programu nazýváme podprogramy, někdy také procedury nebo funkce. Pojmenování části programu provádíme pomocí klíčového slova `def`, za kterým následuje jméno podprogramu, jeho případné parametry a dvojtečka. Tělo podprogramu je odsazené pomocí tabulátoru.

```
def vypis():
    print('*********')
    print('*********')
vypis()
```

Definovaný podprogram se nevykoná, dokud není v programu zavolán, tj. dokud se v programu neobjeví volání podprogramu (jeho jméno se závorkami). Podprogram může být v programu volán i opakovaně.

Podprogram může mít také parametr, tj. vstupní proměnnou, prostřednictvím které můžeme do podprogramu vložit nějakou hodnotu. Parametr podprogramu se zadává v hlavičce podprogramu a při volání podprogramu musí být definována hodnota, která má být parametru podprogramu přiřazena.

```
def vypis_hvezdy(pocet):
    print('*'*pocet)
vypis_hvezdy(5)
vypis_hvezdy(10)
vypis_hvezdy(15)
```

Program postupně vypíše 5, 10 a 15 hvězdiček. Počet hvězdiček je zadán parametrem `pocet` podprogramu `vypis_hvezdy`.

Příklad: Podprogram s objektem želva

Definujeme podprogram `amfora()`, který vykreslí amforu podobně jako v předešlém příkladu. Podprogram má jeden parametr – délka strany amfory. Tři strany jsou stejně dlouhé, čtvrtá kratší (pouze 63% původní délky).

Pokud budeme vykreslovat více amfor do řádku, je potřeba želvu pro kreslení přesunout na místo pro kreslení nové amfory. K tomu slouží podprogram `posun()`. Želva zvedne kreslící pero, posune se, natočí se do počátečního směru (vpravo). Podprogram má opět parametr – délku posunu vlevo.

Program vykreslí tři amfory – dvě menší a střední větší vyplněnou.

```
import turtle
def amfora(delka):
    t.lt(36)
    t.fd(delka)
    t.rt(36)
    t.bk(delka)
    t.rt(36)
    t.fd(delka)
```

```
    t.rt(144)
    t.fd(0.63*delka)
def posun(delka):
    t.pu()
    t.fd(delka)
    t.lt(180)
    t.pd()

t=turtle.Turtle()
amfora(100)
posun(125)
t.begin_fill()
amfora(150)
t.end_fill()
posun(100)
amfora(100)
posun(50)
```

Upozornění: Objekt želva vytváříme před voláním jednotlivých podprogramů tak, aby tyto využívaly pro kreslení tuto želvu – viz program.

Úkol k textu

Navrhněte vlastní útvar (domeček, hvězdu apod.), který vykreslíte pomocí objektu želva. Definujte podprogram pro vykreslení daného útvaru. Vytvořte program, který vykreslí útvar třikrát vedle sebe.

2.3 Cyklus s pevným počtem opakování

Často se stává, že některé činnosti potřebujeme víckrát opakovat. Například pokud potřebujeme víckrát (pětkrát) vypsat stejný text, můžeme sestavit program následovně:

```
print("Pracujeme v Pythonu")
print("Pracujeme v Pythonu")
print("Pracujeme v Pythonu")
print("Pracujeme v Pythonu")
print("Pracujeme v Pythonu")
```

Pro opakované činnosti můžeme použít také cyklus a předchozí program zapsat pomocí cyklu `for`.

```
for i in 1,2,3,4,5:
    print("Pracujeme v Pythonu")
```

Jak to funguje? Do proměnné `i` se postupně přiřazují hodnoty ze seznamu hodnot uvedených za slovem `in`. Začneme první hodnotou v seznamu, tj. hodnotou 1. Pro tuto

hodnotu proměnné `i` se vykoná tělo cyklu – příkazy, které následují na dalších řádcích a jsou mírně odsazené. Text se tedy vypíše poprvé. Následně se pokračuje další hodnotu proměnné `i`, hodnotou 2. Výpočet skončí, když se tělo cyklu vykonalo pro všechny hodnoty proměnné `i`.

Upozornění: Příkaz `for` musí být ukončen dvojtečkou – viz poslední znak hlavičky cyklu.

Stejný výsledek dostáváme i pro cyklus `for i in 0,0,0,0,0:` nebo pro `for i in 1,10,100,1000,10000:` - text mezi apostrofy se vypíše pětkrát.

Do programu můžeme přidat další řádek.

```
for i in 1,2,3,4,5:
    print("Pracujeme v Pythonu")
print("==================")
```

Výsledek pak vidíme v okně *Shell*.

```
Pracujeme v Pythonu
Pracujeme v Pythonu
Pracujeme v Pythonu
Pracujeme v Pythonu
Pracujeme v Pythonu
==================
```

Tělo cyklu `for` obsahovalo pouze jediný řádek, proto se nově přidaný řádek vykonal až po skončení cyklu `for`. Pokud přidaný řádek také odsadíme pomocí tabulátoru, dostáváme jiný výsledek – oba příkazy `print` se stávají součásti těla cyklu.

```
for i in 1,2,3,4,5:
    print("Pracujeme v Pythonu")
    print("==================")
```

```
Pracujeme v Pythonu
==================
Pracujeme v Pythonu
==================
Pracujeme v Pythonu
==================
Pracujeme v Pythonu
==================
Pracujeme v Pythonu
==================
```

Proměnnou cyklu `for` lze využít pro výpočty v těle cyklu.

```
for i in 1,2,3,4,5:
    print(str(i)+" na druhou je "
    +str(i*i))
```

```
for i in 10,12,15,18,25:
    print(str(i)+" na druhou je "
    +str(i*i))
```

```
1 na druhou je 1
2 na druhou je 4
3 na druhou je 9
4 na druhou je 16
5 na druhou je 25
```

```
10 na druhou je 100
12 na druhou je 144
15 na druhou je 225
18 na druhou je 324
25 na druhou je 625
```

Cyklus `for` lze využít i v interaktivním režimu (pro vykonání příkazu je potřeba dvakrát zmáčknout klávesu *Enter*). Také lze vytvářet různé seznamy pro hodnoty proměnné cyklu, lze dokonce měnit jejich datové typy – v seznamu mohou být celá čísla, reálná čísla i řetězce.

```
>>> for a in 3.14,'pes ',2+3,'{}':
      print(a*10)

31.400000000000002
pes pes pes pes pes pes pes pes pes pes
50
{}{}{}{}{}{}{}{}{}{}
>>> a=1
>>> for i in 'Python':
      print(i*a)
      a=a+1

P
yy
ttt
hhhh
ooooo
nnnnnn
```

Pokud je seznamem hodnot proměnné cyklu řetězec, tento se rozloží na jednotlivé znaky a příkazy v těle cyklu se vykonají pro jednotlivé znaky.

Pro generování seznamu hodnot proměnné cyklu `for` lze využít funkci `range()` v jedné ze tří verzí:

- `range(stop);`
- `range(start, stop);`
- `range(start, stop, krok).`

Parametr `start` určuje první prvek generované posloupnosti (pokud chybí, je první prvek 0). Parametr `stop` určuje poslední prvek, který se už v posloupnosti nebude vyskytovat. Parametr `krok` udává hodnotu, o kterou další prvek vyšší/nižší (pokud chybí, je krok 1).

Příklad: Součet čísel

Následující program sečte čísla od jedné do zadaného čísla.

```
n=int(input("Zadej číslo "))
soucet=0
for i in range(n):
    soucet=soucet+(n+1)
print('Součet čísel od 1 do', n, 'je', soucet)
```

```
Zadej číslo 25
Součet čísel od 1 do 25 je 650
```

Příklad: Pyramida

Následující program vykreslí pyramidu zadané výšky.

```
n=int(input("Zadej výšku "))
for i in range(n):
    print(' '*(n-i), '*'*(2*i-1))
```

```
Zadej číslo 7

       *
      ***
     *****
    *******
   *********
  ***********
```

2.3.1 Objekt želva v příkladech s využitím cyklu

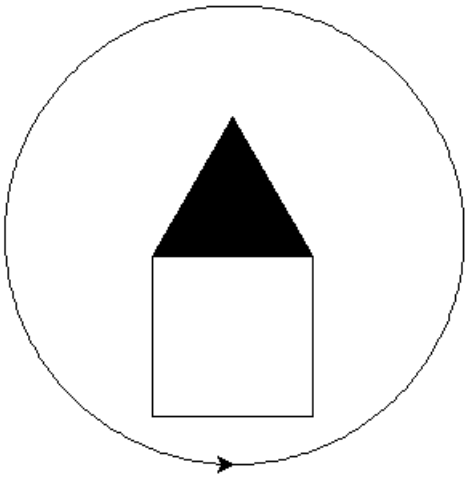
Pomocí objektu želva lze kreslit pravidelné obrazce. V tom případě obvykle používáme v programech cyklus s pevným počtem opakování.

Příklad: Čtverec, trojúhelník, kružnice

Definujeme podprogramy pro vykreslení čtverce, trojúhelníku a kružnice (kružnice je pravidelný 360tiúhelník s krátkou stranou) a využijeme je při vykreslení výsledného obrázku domečku v kruhu.

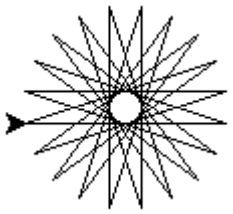
```
import turtle
def ctverec(delka):
    for i in range(4):
        t.fd(delka)
        t.lt(90)
def trojuhel(delka):
```

```
    for i in range(3):
        t.fd(delka)
        t.lt(120)
def kruh(delka):
    for i in range(360):
        t.fd(delka)
        t.lt(1)

t=turtle.Turtle()
ctverec(100)
t.lt(90)
t.fd(100)
t.rt(90)
t.begin_fill()
trojuhel(100)
t.end_fill()
t.rt(90)
t.fd(100)
t.lt(90)
t.fd(50)
t.pu()
t.rt(90)
t.fd(30)
t.lt(90)
t.pd()
kruh(2.5)
```

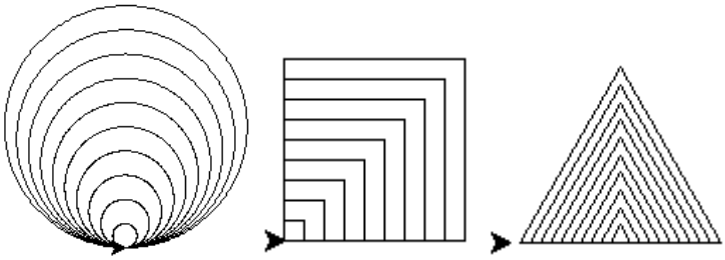


Úkoly k textu

- Vytvořte podprogram pro vykreslení libovolného n-úhelníku – parametrem podprogramu bude *n*, tj. počet vrcholů n-úhelníku.
- Vytvořte program pro vykreslení pěticípé hvězdy – úhel otočení želvy je 144°.



- Vytvořte podprogram pro vykreslení libovolné hvězdy. Podprogram bude mít tyto parametry: délku strany hvězdy, počet vrcholů a úhel otočení.
- Upravte předchozí program. Definujte podprogram pro vykreslení domečku a vytvořte program, který vykreslí do řady několik domků vedle sebe – jednoduchou ulici.
- Vytvořte program, který vykreslí 10 postupně se zvětšujících kružnic. Podobný princip můžete aplikovat na postupně se zvětšující čtverce nebo trojúhelníky.

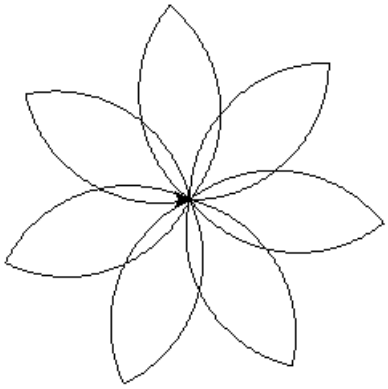


Jinou skupinu příkladů využívajících cyklus jsou příklady na kreslení útvarů pravidelně rozmístěných po kruhu. Počet opakování těla cyklu je dán počtem útvarů vykreslených po kruhu.

Příklad: Kytka

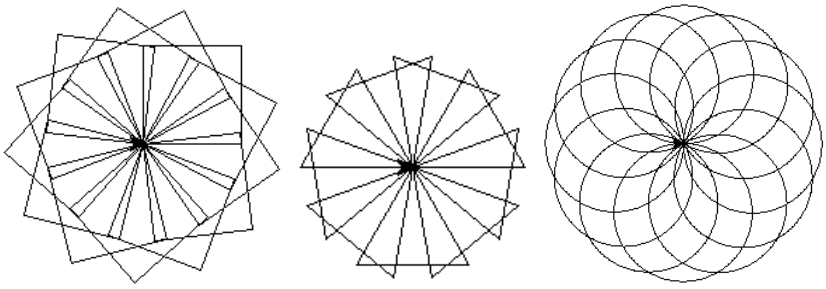
Definujeme podprogram pro vykreslení čtvrtkruhu. Ze dvou čtvrtkruhů vznikne jeden lupen květu. Lupeny vykreslíme postupně rovnoměrně po kruhu.

```
import turtle
def ctvrt(delka):
    for i in range(90):
        t.fd(delka)
        t.lt(1)
def lupen(delka):
    ctvrt(delka)
    t.lt(90)
    ctvrt(delka)
    t.lt(90)
def kvet(n, delka):
    for i in range(n):
        lupen(delka)
        t.lt(360/n)

t=turtle.Turtle()
kvet(7, 1.5)
```



Úkoly k textu

- Vytvořte program pro vykreslení čtverců, trojúhelníků a kružnic po kruhu.



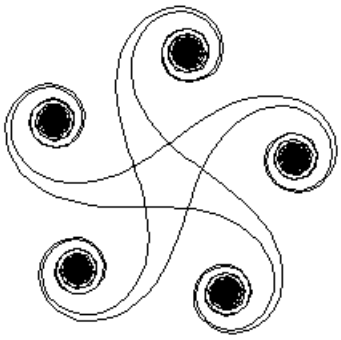
- Vykreslete paprsky slunce – čáry vycházející se středu kruhu stejnoměrně.
- Upravte předchozí program. Definujte podprogram, který kromě květu přikreslí také stonek a listy na stonku. Program vylaďte tak, aby velikost stonku a lístků odpovídala velikosti květu.
- Vykreslete postupně se zvětšující čtverce (trojúhelníky, kružnice) do kruhu. Jaký obrazec vznikne? Co vám připomíná.

Pomocí cyklů lze vykreslovat také spirály. Spirála se na rozdíl od kružnice liší tím, že se buď krok posunu vpřed, nebo úhel otočení želvy mění s rostoucí/klesající tendencí.

Příklad: Spirála

Definujeme podprogram pro vykreslení spirály, v které se bude měnit úhel otočení želvy s mírně rostoucí tendencí.

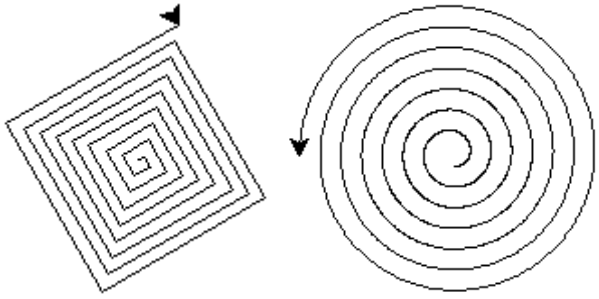
```
import turtle

t = turtle.Turtle()
t.speed(0)
for uhel in range(1, 2000):
    t.fd(8)
    t.rt(uhel+0.1)
```



Příkaz `t.speed(0)` slouží k nastavení rychlosti pohybu želvy. Parametr 0 odpovídá nejvyšší rychlosti, nejnižší rychlost je 10. Standardně je nastavena rychlost 3. Nastavení rychlosti využijeme zejména při vykreslování kružnic a oblouků.

Úkoly k textu

- Vytvořte program pro vykreslení čtverce „do spirály“ – strana čtverce se postupně zvětšuje. Podobně lze vykreslit do spirály trojúhelníky nebo jiné hvězdicovité útvary.



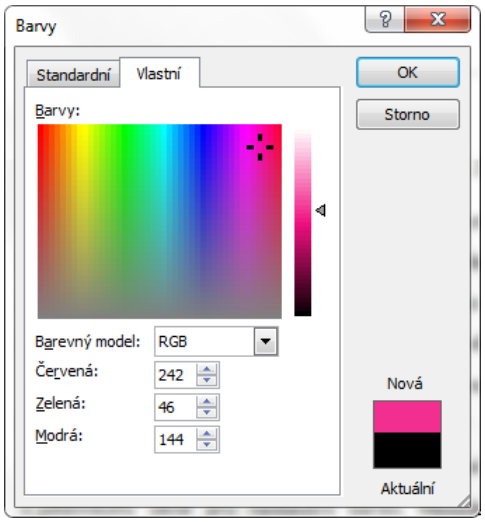
- Vytvořte spirálu s rostoucích čtvrtkruhů – každý další čtvrtkruh je o málo větší než předešlý, čtvrtkruhů vykreslete alespoň 30.

2.3.2 Barevné kreslení

Prozatím jsme využívali pouze černobílé kreslení. Želva ale může kreslit i barevně.

Problém použití barev v jazyce Python sestává z jejich kódování ve formě hexadecimálních čísel. Používá se standardní model RGB (red – červená, green – zelená a blue – modrá). Hodnota se ale zadává jako hexadecimální číslo v apostrofech. Například `'#FF0000'` odpovídá červené barvě, `'#00FF00'` odpovídá zelené a `'#0000FF'`odpovídá modré barvě. Další barevné kombinace dostáváme kombinací těchto základních čísel, například `'#FFFF00'` odpovídá žluté barvě, `'#00FFFF'` odpovídá tyrkysové a `'#FF00FF'`odpovídá růžové barvě. Další hodnoty barev lze najít v jakémkoliv okně pro nastavení barev. Hexadecimální kódy využívají však pouze grafické editory.

Jednodušší možnost představuje zadávat barvu pomocí tří čísel v rozmezí 0 až 255 – každé číslo reprezentuje poměr tří základních barev (červená, zelená, modrá) v dané barvě. Tyto hodnoty pro jednotlivé barvy najdeme například v textovém editoru MS Word (viz obrázek). Pro převod použijeme funkci, kterou si musíme vytvořit. Funkce `rgb` vrátí pro dané hodnoty tři složek barev hodnotu barvy v hexadecimálním zápisu.




```
def rgb(r, g, b):
    return '#{:02x}{:02x}{:02x}'.format(r, g, b)
```

Některé z barev mají své pojmenování a v programu v jazyce Python lze přímo využít tyto názvy:

- `'white'` –bílá;
- `'black'` – černá;
- `'red'` – červená;
- `'blue'` – modrá;
- `'yellow'` – žlutá;

- `'green'` – zelená;
- `'orange'` – oranžová;
- `'violet'` – fialová.

Objekt želva umožňuje nastavit barvy v těchto případech:

- barva kreslící čáry – `t.pencolor(rgb(242, 46, 144));`
- barva výplně – `t.fillcolor('red')`, barva výpně musí být nastavena před začátkem kreslení vyplňované oblasti, tj. před `begin_fill()`.

Při barevném kreslení využijeme také možnost změny tloušťky čáry. To provedeme pomocí příkazu `t.pensize(5)` – tloušťka čáry se změní na pět bodů.

Příklad: Barevná amfora

Využijeme podprogram `amfora()`, který jsme již vytvořili. Barvu čáry nastavíme na oranžovou a amforu vyplníme žlutou barvou.

```
import turtle
def amfora(delka):
    t.lt(36)
    t.fd(delka)
    t.rt(36)
    t.bk(delka)
    t.rt(36)
    t.fd(delka)
    t.rt(144)
    t.fd(0.63*delka)

t=turtle.Turtle()
t.pensize(3)
t.pencolor('orange')
t.fillcolor('yellow')
t.begin_fill()
amfora(100)
t.end_fill()
t.hideturtle()
```

Pokud šipka znázorňující želvu ruší kreslení, můžeme ji učinit neviditelnou příkazem `t.hideturtle()`. Kurzor želvy opět zviditelníme příkazem `t.showturtle()`.

Úkol k textu

Vyzkoušejte další barevné kombinace amfory. Zkuste zadat barvu také pomocí funkce RGB, tj. pomocí jejich tří složek.

Příklad: Barevná spirála

Vykreslíme barevnou spirálu ze čtvrtkružnic. Každá čtvrtkružnice bude vykreslena jinou, náhodně vygenerovanou barvou.

```
import turtle
import random
def rgb(r, g, b):
    return '#{:02x}{:02x}{:02x}'.format(r, g, b)
def ctvrt(delka):
    for i in range(90):
        t.fd(delka)
        t.lt(1)

t=turtle.Turtle()
t.pensize(5)
for i in range(20):
    t.pencolor(rgb(random.randint(0, 255),
                   random.randint(0, 255),
                   random.randint(0, 255)))
    ctvrt((i+2)*0.1)
```

Pro generování náhodné barvy využijeme modul `random`. Tem musíme do prostředí našeho programu importovat – viz `import random`. Funkce `randint(a, b)` generuje náhodná čísla od a do b (včetně a a b). Protože jednotlivé složky barvy jsou čísla v rozmezí 0 až 255, používáme `random.randint(0, 255)`.

Úkol k textu

Obarvěte všechny doposud nakreslené obrazce. Pokud lze obrazce vybarvit (vyplnit barvou, pokuste se také o jejich vybarveni.

2.4 Podmíněný cyklus a podmíněný příkaz

Kromě přeného vymezení parametrů a počtů opakování cyklů lze v programech využít omezující nebo rozhodovací podmínky. Ty využijeme v podmíněných příkazech (cyklech a příkazech pro větvení programu).

Dále uvidíme, že podmínky jsou logického typu `bool`. Proměnné tohoto typu nabývají jedné z dvou hodnot `False` nebo `True` (pozor na velká písmena v označeních hodnot).

2.4.1 Podmíněný cyklus

Někdy není předem jasné kolik průchod cyklem je potřeba provést. Počet opakování dokážeme omezit pouze podmínkou – jdi dopředu, dokud nenarazíš na zeď, čekej, dokud nepřijde zima apod. V takovém případě uplatníme cyklus `while`, tzv. podmíněný cyklus.

```
import turtle
t=turtle.Turtle()
while t.xcor()<250:
    t.fd(2)
```

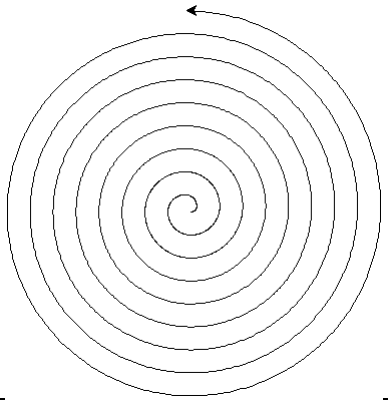
Funkce `xcor()` a `ycor()` určují x-ovou, resp. y-vou souřadnici aktuální polohy želvy. Podle programu jede želva dopředu vždy o 2 body, dokud její x-ová souřadnice je menší než 250. Pak se pohyb želvy zastaví.

Cyklus `while` obsahuje podmínku, tělo cyklu se vykonává tak dlouho, dokud je podmínka splněná.

Příklad: Spirála

Vykreslíme spirálu ze čtvrtkružnic. Vykreslování ukončíme, pokud se želva dostane od středu souřadného systému dále než 200 bodů.

```
import turtle

def ctvrt(delka):
    for i in range(90):
        t.fd(delka)
        t.lt(1)

t=turtle.Turtle()
t.speed(0)
i=1
while t.xcor()<200 and t.ycor()<200:
    ctvrt(i*0.1)
    i = i+1
```



Podmínka ukončení je testována po každé čtvrtkružnici, tj. želva se může zastavit v libovolném bodě na osách *x* nebo *y*, a to tam, kde poprvé její x-ová nebo y-ová souřadnice dosáhne nebo překročí hodnotu 200.

Z příkladu je patrné, že podmínky lze spojovat pomocí logických spojek – `and` (obě podmínky musí platit současně) a `or` (musí platit alespoň jedna z podmínek). Negaci podmínky vyznačíme slovem `not`.

Při porovnání v podmínkách využíváme znaky menší (<) a větší (>) a znaky menší nebo rovný (<=) a větší nebo rovný (>=). Rovnost označujeme dvojitým „rovná se“ (==), nerovnost vykřičníkem a rovná se (!=).

```
>>> a<3
False
>>> a>3
True
>>> b!=7
False
>>> a==5 or b<=5
True
```

Podmínka může být splněna (označuje se `True`) nebo nemusí být splněna (označeno `False`). Podmínka je typu `bool` – logický typ nabývající hodnot `True` nebo `False`. Proměnná logického typu vznikne přiřazením hodnoty `True` nebo `False` nebo přiřazením podmínky do proměnné.

```
>>> c=True
>>> c
True
>>> c = not c
>>> c
False
```

Úkol k textu

Zamyslete se nad tím, jak byste vykreslili čtverec pomocí cyklu `while` (bez použití cyklu `for`). Rozmyslete, zda lze každý `for` cyklus přepsat pomocí cyklu `while`. Jakým způsobem je nutné zabezpečit proměnnou cyklu? Lze také každý cyklus `while` nahradit cyklem `for`?

2.4.2 Podmíněný příkaz

Kromě opakování jistých činností je často v programech důležité rozhodnout se, jak bude dále výpočet pokračovat. V daném bodě mohou existovat dvě nebo více možností, podle kterých může výpočet dále pokračovat. Závidí obvykle na splnění nebo nesplnění nějaké podmínky.

V těchto situacích se ocitáme i v běžném životě. Na křižovatkách se rozhodujeme, kterou cestou se vydáme, hezky je to vidět v pohádce o Honzovi, který neví kudy-kam, aby došel v princezně. Podle aktuálního ročního období a aktuálního počasí se rozhodujeme, co si oblečeme na cestu do školy nebo zda si vezmeme deštník. Podle aktuální finanční situace se rozhodujeme o tom, zda si pořídíme nový mobil nebo obyčejnou či luxusnější bundu na zimu.

Podobně je tomu v programech. K větvení výpočtu slouží podmíněný příkaz `if` nebo jeho varianta `if-else`. Protože zadávání podmínek jsme již zvládli, je práce s podmíněným příkazem již jednoduchá.

```
>>> teplota=25
>>> if teplota >20:
      print('Je horko!')

Je horko!
>>> teplota =15
>>> if teplota >20:
      print('Je horko!')

>>>
```

Podle aktuální teploty vypíšeme, zda je horko. Program můžeme vylepšit – pokud není horko, vypíšeme jinou informaci.

```
>>> teplota=25
>>> if teplota>20:
      print('Je horko')
else:
      print('Je teplo')

Je horko
```

Příkazy následující bezprostředně za podmínkou se vykonají, pokud je podmínka splněná. Pokud podmínka splněná není, vykonají se příkazy za konstrukcí `else`.

Příklad: Počasí

Program vypíše podle aktuální teploty, jak je venku. Použijeme k tomu několik podmíněných příkazů vložených do sebe.

```
teplota=int(input('Zadej aktuální teplotu'))
if teplota>25:
    print('Je horko')
else:
```

```
    if teplota>15:
        print('Je teplo')
    else:
        if teplota>10:
            print('Je akorát')
        else:
            print('Je chladno')
```

Příklad: Kolečko nebo čtverec?

```
import tkinter, random

g = tkinter.Canvas(width=300, height=300)
g.pack()

co=random.randint(0, 1)
x=random.randint(0, 300)
y=random.randint(0, 300)
if co == 0:
    g.create_oval(x-5, y-5, x+5, y+5, fill='red', outline='red')
else:
    g.create_rectangle(x-5, y-5, x+5, y+5, fill='blue',
                       outline='blue')
```

Program vykreslí červené kolečko nebo modrý čtvereček podle náhodně vygenerovaného čísla 0 nebo 1.

Úkoly k textu

- Vytvořte jednoduchý program, který podle toho, zda prší, vypíše, zda si máme nebo nemáme vzít deštník.
- Vytvořte program, který o daném čísle vypíše, zda je prvočíslo či nikoliv – zjistí, že dané číslo není dělitelné ničím jiným než jedničkou a sebou samým.
- Vytvořte program, který vypíše všechny dělitele daného čísla.

2.4.3 Náhodné procházky

Náhodná procházka želvy je pohyb, kdy želva se rozhodne pro náhodný směr a provede tímto směrem krok. Tento proces se dokola opakuje. Pomocí podmíněných příkazů lze vymezit prostor pro pohyb želvy.

Základem náhodné procházky želvy je následující princip:

```
import turtle, random

turtle.delay(0)
t=turtle.Turtle()
```

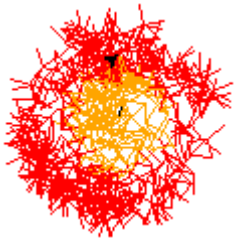
```
for i in range(1000):
    t.rt(random.randint(0, 359))
    t.fd(10)
```

Želva z grafické plochy takto brzy uteče. Jednoduše vymezíme její prostor do kruhu.

```
for i in range(2000):
    t.rt(random.randint(0, 359))
    t.fd(10)
    if t.distance(0, 0)>50:
        t.back(10)
```

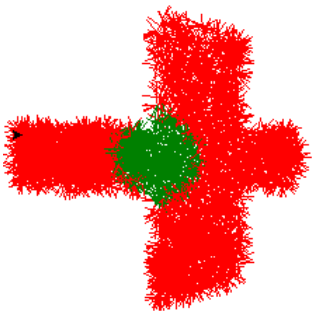
Příkaz `distance(a, b)` vrací vzdálenost želvy od bodu [*a*, *b*]. Pokud je tato vzdálenost větší než 50 bodů, želva se vrátí zpět. Želva tak náhodnou procházkou vybarví plochu kruhu. Plochu můžeme barevně zvýraznit.

```
for i in range(1000):
    t.rt(random.randint(0, 359))
    t.fd(10)
    if t.distance(0, 0)>50:
        t.back(10)
    else:
        if t.distance(0, 0)>25:
            t.pencolor('red')
        else:
            t.pencolor('orange')
```



Pomocí x-ové a y-ové souřadnice želvy můžeme vymezit prostory čtverců nebo obdélníků. Často použijeme funkci `abs()` – absolutní hodnota. Následující program vybarví plochu kříže v kruhu.

```
for i in range(10000):
    t.rt(random.randint(0, 359))
    t.fd(10)
    if t.distance(0,0) < 30:
        t.pencolor('green')
    else:
        t.pencolor('red')
    if t.distance(0, 0)<100:
        if abs(t.xcor()-30)>30 and
           abs(t.ycor())>20:
            t.back(10)
    else:
        t.back(10)
```



Úkoly k textu

- Vytvořte program, který vybarví náhodnou procházkou želvy plochu obdélníku.
- Vyberte si vlajku nějakého státu a vytvořte program, který vybarví náhodnou procházkou želvy plochu této vlajky. Vhodné jsou vlajky severských zemí (Norka, Finska), vlajky Německa nebo Maďarska. Můžete se pokusit také o vlajku Česka.

- Vytvořte program, který náhodnou procházkou želvy vybarví plochu kytky podle obrázku.

- Navrhněte vlastní obrázek z kružnic a obdélníků a vytvořte tento obrázek pomocí náhodné procházky želvy. Můžete například vykreslit malou želvičku.

2.5 Strukturovaný typ pole

Pole je strukturovaný typ – je to posloupnost hodnot různých typů oddělených čárkami, která je uzavřená v hranatých závorkách. Výhodou pole je, že ho můžeme procházet pomocí cyklu `for`.

Pole můžeme vytvořit pomocí funkce `append()`.

```
pole=[]
for i in range(10):
    pole.append(0)
print(pole)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Můžeme vytvořit pole prvků s náhodnými hodnotami.

```
import random
pole=[]
for i in range(10):
    pole.append(random.randrange(300))
print(pole)
```

```
[138, 183, 269, 194, 268, 248, 123, 37, 199, 213]
```

Na jednotlivé prvky pole se můžeme odkazovat pomocí indexů. Indexy jsou číslovány od nuly. Vytvořené náhodné pole můžeme pak uspořádat pomocí bublinkového tříděni.

```
import random
pole=[]
for i in range(10):
    pole.append(random.randrange(300))
print(pole)
for i in range(10):
```

```
    for j in range(9):
        if pole[j]>pole[j+1]:
            pom=pole[j]
            pole[j]=pole[j+1]
            pole[j+1]=pom
print(pole)
```

```
[246, 99, 194, 285, 71, 118, 150, 57, 87, 126]
[57, 71, 87, 99, 118, 126, 150, 194, 246, 285]
```

Libovolný prvek pole lze odstranit pomocí funkce `pop(a)`, kde a je intex daného prvku. Funkce `len()` vrací aktuální počet prvků pole.

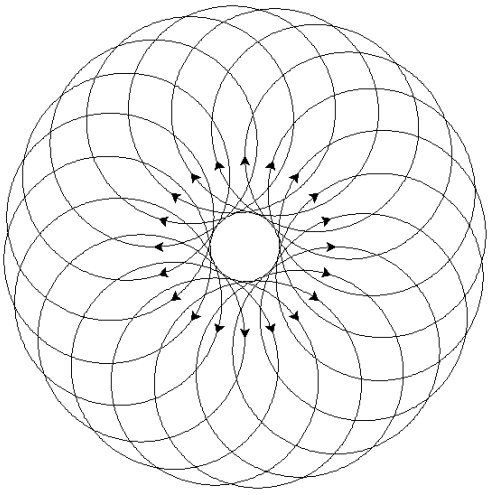
Úkol k textu

Vymyslete příklad, kde využijete možnost přidávání a ubírání prvků pole během výpočtu programu. Může se jednat například o algoritmus pro generování pole všech prvočísel v rozmezí 0 až 1000.

2.5.1 Pole objektů typu želva

Prvky pole mohou být libovolného typu. Můžeme například vytvořit pole želv. Želvy rozmístíme stejnoměrně po kruhu. Poté všechny želvy vykreslí najednou kružnice.

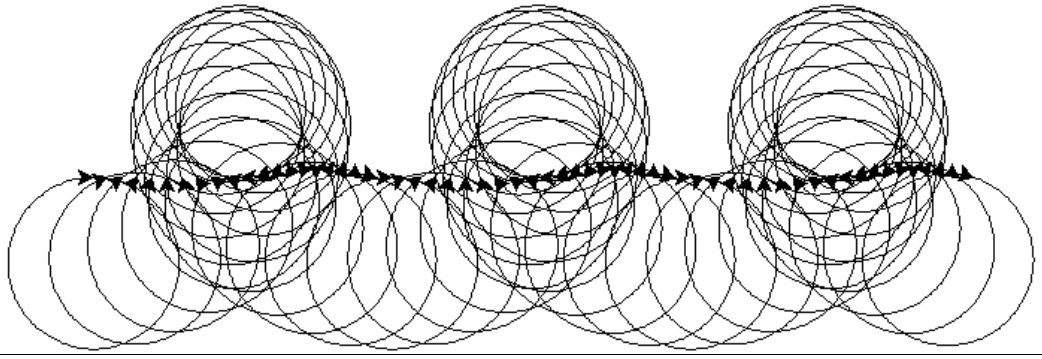
```
import turtle

turtle.delay(0)
pole = []
for i in range(60):
    t = turtle.Turtle()
    t.seth(i*18)
    t.pu()
    t.fd(100)
    t.pd()
    pole.append(t)

for j in range(360):
    for i in range(len(pole)):
        pole[i].fd(2)
        pole[i].rt(1)
```



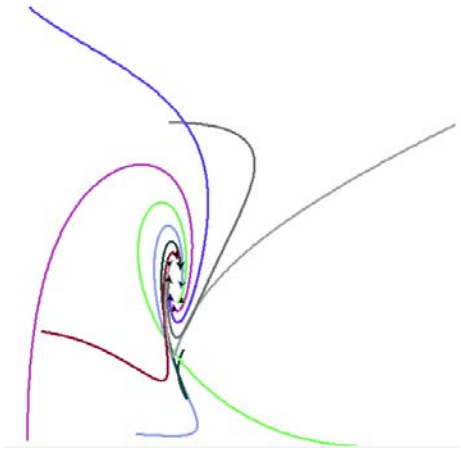
Želvy můžeme rozmístit také na přímku a použít stejný princip kreslení.

```
import turtle

turtle.delay(0)
pole = []
for i in range(60):
    t = turtle.Turtle()
```

```
    t.seth(i*18)
    t.pu()
    t.setpos(-300+i*10, 0)
    t.pd()
    pole.append(t)

for j in range(180):
    for i in range(len(pole)):
        pole[i].fd(2)
        pole[i].rt(2)
```



Další možností je, že se želvy v poli mohou honit. Vygenerujeme želvy náhodně rozmístěné po grafické ploše. Nastavíme náhodnou barvu kreslícího pera a tloušťku pera 3 pixely. Želvy umístíme do pole.



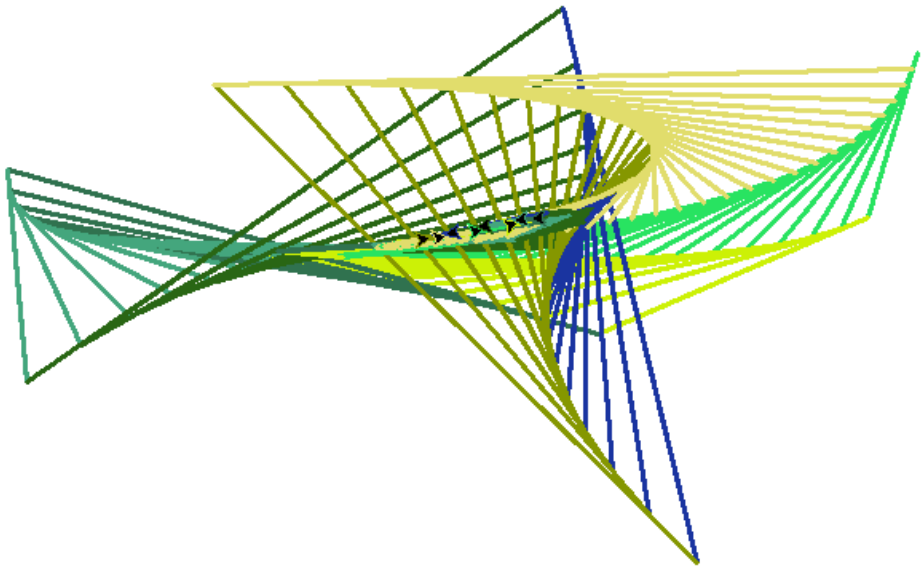
```
import turtle, random

pole = []
turtle.delay(0)
for i in range(8):
    t = turtle.Turtle()
    t.pu()
    t.setpos(random.randint(-300, 300),
             random.randint(-300, 300))
    t.pencolor('#{:06x}'.format(
             random.randrange(256**3)))
    t.pensize(3)
    t.pd()
    pole.append(t)

while True:
    for i in range(len(pole)):
        j = (i+1) % 8
        pole[i].seth(pole[i].towards(pole[j]))
        pole[i].fd(pole[i].distance(pole[j])/100)
```

Vlastní proces honění probíhá tak, že *i*-ta želva se natočí směrem k *i*+1-ní želvě a posune se k „sousedce“ o setinu vzdálenosti. Funkce `towards(a, b)` natočí aktuální želvu směrem k bodu [*a*, *b*].

Zajímavé je také použití podmíněného cyklu `while`. Pokud místo podmínky zadáme pouze logickou hodnotu `True`, cyklus se stane nekonečným, tj. program sám nikdy neskončí. Program musí být násilně ukončen například zavřením grafického okna. Tento příklad uvádíme pouze pro ilustraci, v praxi takové „nekonečné“ programy nevytváříme, jejich uživatelé by je neuměli obsluhovat.

Obrázek se zajímavě promění, v každém kroku želva nejprve projde cestu k následující želvě a pak se vrátí o 90% vzdálenosti zpět (příprava pole želv je stejná, jako v předešlém případě).

```
for k in range(100):
    for i in range(len(pole)):
        j = (i+1) % 8
        pole[i].seth(pole[i].towards(pole[j]))
        vzdal = pole[i].distance(pole[j])
        pole[i].fd(vzdal)
        pole[i].fd(-0.9*vzdal)
```



Shrnutí kapitoly

- Objekt želva je určen pro kreslení obrázků. Základním principem je želva pohybující se po grafické ploše a kreslící při svém pohybu čáru. Želva se může pohybovat vpřed nebo vzad a otáčet vlevo nebo vpravo.
- Složitější programy je vhodné kvůli přehlednosti strukturovat do podprogramů. V jazyce Python jsou všechny podprogramy funkce. Podprogram je určen svým jménem, kterým následně můžeme podprogram zavolat pro jeho vykonání. Vstupními hodnotami do podprogramů jsou jeho parametry.

- Cyklus s pevným počtem opakování `for` slouží k opakování části výpočtu. Počet opakování je vymezen rozmezím zadaným v hlavičce cyklu.
- Podmíněný cyklus `while` slouží k opakování části výpočtu. Počet opakování těla cyklu není předem určen, je vymezen podmínkou – tělo cyklu se opakuje do té doby, pokud je podmínka splněna.
- Podmíněný příkaz `if-else` slouží k větvení výpočtu podle zadané podmínky.
- Pole je strukturovaný typ prvků různých typů. Prvky lze v průběhu výpočtu do pole přidávat nebo je z pole odebírat. Lze definovat pole želv a využít pro kreslení více želv. Výhodou pole je možnost procházet jeho prvky pomocí cyklu s pevným počtem opakování.

Kontrolní otázky a úkoly:

1 Vysvětlete pojem podprogram. Proč je vhodné členit program do podprogramů?

2 Popište cyklus `for` s pevným počtem opakování. Popište průběh výpočtu při vykonávání cyklu.

3 Jakým způsobem lze definovat hodnoty pro proměnnou cyklu `for`?

4 Popište podmíněný cyklus `while`. Popište průběh výpočtu při vykonávání cyklu.

5 Popište možnosti definování podmínek. Jaké hodnoty může podmínka nabývat?

6 Popište podmíněný příkaz `if-else`. Popište průběh výpočtu při vykonávání příkazu – při větvení programu.

7 Definujte pojem pole. Jakým způsobem definujeme prázdné pole? Jak lze do pole přidávat prvky? Jak lze prvky z pole odebírat?

8 Popište objekt želva. Popište principy kreslení obrázků pomocí želvy a formulujte základy želví grafiky.

Otázky k zamyšlení:

- Vyhledejte název programovacího jazyka, z kterého je převzata myšlenka objektu želva. Kdy byl tento jazyk vytvořen a kým? Využívá se tento programovací jazyk i v současnosti?
- Znáte jiný programovací jazyk než jazyk Python? Jaká je struktura tohoto jazyka, jaká je struktura cyklů a podmíněných příkazů. Vyhledejte společné znaky i odlišnosti s jazykem Python.
- Zamyslete se nad tím, jakým způsobem byste připravili výuku žáků pro jejich seznámení s cyklem `for`. Vytvořte sadu příkladů, které by žáci postupně řešili a prakticky je vyzkoušejte (vytvořte dané programy).

- Navrhněte strukturu výuky základů programování, jak jsme se s nimi seznámili v této kapitole. Svůj návrh rozepište do plánu pro jednotlivé vyučovací hodiny.

Korespondenční úkol

Pomocí objektu želva nakreslete nějaký obrázek. Při kreslení využijte jednotlivé příkazy pro pohyb želvy, ale také příkaz cyklu a podmíněné příkazy. Dbejte na přehlednost vytvořeného programu, program strukturujte do podprogramů. Obrázek průběžně vybarvujte.

Ověřte funkčnost programu. Hotový program uložte ve formátu PY a odevzdejte ke kontrole tutorovi.

3 Tvorba grafických her

Cíl kapitoly

Po nastudování této kapitoly byste měli být schopni:

- kreslit jednoduché obrázky pomocí objektu tkinter;
- pracovat s událostmi myši a klávesnice a obsluhovat je;
- definovat jednoduchou třídu;
- vytvořit jednoduchou grafickou hru.

Klíčová slova

Grafika, tkinter, událost myši, událost klávesnice, čas, událost času, grafická hra.

Čas na prostudování kapitoly

5 hodiny

Průvodce studiem

Poslední částí naší práce bude práce s grafikou a obsluha událostí. To vše je důležité proto, abychom mohli společně vytvořit jednoduchou hru, která bude ovládána myší nebo klávesnicí. Nebude to jednoduché, ale věříme, že právě na tuto část výuky se nejvíc těšíte. Proto přejeme hodně úspěchů a trpělivosti.

3.1 Standardní grafický režim

Pro práci s grafikou lze využít objekt `tkinter`. Příkazem `tkinter.Canvas()` vytvoříme odkaz na grafickou plochu. Vytvořenou plochu zobrazíme příkazem `pack`.

```
import tkinter

g = tkinter.Canvas()
g.pack()
```

Grafická plocha je určena souřadnicemi:

- x-ová prochází zleva (hodnota 0) doprava;
- y-ová prochází zhora (hodnota 0) nadol.

Počátek souřadné soustavy – bod [0, 0] se nachází v levém horním rohu grafické plochy. Možná jsou i záporné souřadnice, ty se nachází mimo grafickou plochu.

Počáteční velikost grafické plochy je 378x284 bodů, je možné ji změnit. Pro grafickou plochu obecně můžeme nastavit tyto parametry:

- `bg` – barva pozadí;
- `width` – šířka grafické plochy;
- `height` – výška grafické plochy.

Parametry nastavujeme při vytváření grafické plochy.

```
g = tkinter.Canvas(bg='white',width=500,height=300)
```

3.1.1 Kreslení

Úsečku kreslíme pomocí příkazu `create_line(a, b, x, y)`, který vykreslí úsečku z bodu [*a*, *b*] do bodu [*x*, *y*]. Pro úsečku lze nastavit tloušťku čáry – parametr `width`, barvu čáry – parametr `fill`.

```
g.create_line(50, 120, 150, 70)
g.create_line(10, 160, 110, 200, width=7, fill='blue')
```

Obdélník kreslíme pomocí příkazu `create_rectangle(a, b, x, y)`, který vykreslí obdélník, jehož levý horní vrchol je v bodě [*a*, *b*] a pravý dolní v bodě [*x*, *y*]. Strany obdélníku jsou rovnoběžné se souřadnými osami. Opět lze nastavit tloušťku čáry kreslení – parametr `width`, barvu čáry – parametr `outline` a barvu výplně - parametr `fill`.

```
g.create_rectangle(10, 100, 110, 160, outline='blue', fill='magenta')
```

Elipsu vykreslujeme pomocí příkazu `create_oval(a, b, x, y)`. Princip je podobný jako u kreslení obdélníku, do něhož je elipsa vepsaná.

```
x=150
y=120
for r in range(100,0,-10):
    g.create_oval(x-r,y-r,x+r,y+r,fill='red')
```

Text vypíšeme pomocí příkazu `create_text(a, b, x)`, který vypíše text *x* se středem v bodě [*a*, *b*]. Lze nastavit barvu textu – parametr `fill` a font - parametr `font`.

```
g.create_text(150, 50, text='Python', fill='green',
              font='arial 30 bold')
```

Obrázek z grafického souboru ve formátu PNG nebo GIF vykreslíme pomocí příkazu `create_image(a, b, obr)`, který vykreslí obrázek *obr* na souřadnicích [*a*, *b*] (souřadnice středu obrázku). Obrázek *obr* musíme předem vytvořit pomocí příkazu `tkinter.PhotoImage()`.

```
obr = tkinter.PhotoImage(file='Kostka.png')
g.create_image(50,100,image=obr)
```

3.1.2 Změny v nakreslených grafických útvarech

Jednotlivé vykreslované grafické objekty lze pojmenovat, například `kolo=g.create_oval(50, 100, 80, 130)`.

Pojmenované objekty lze z grafické plochy odstranit pomocí konstrukce `delete()` – `g.delete(kolo)`.

Pojmenovaný objekt lze po grafické ploše posouvat pomocí konstrukce `move()` – například `g.move(kolo, 2, 3)` odpovídá posunu objektu kolo o 2 body vpravo ve směru osy x a o dva body dolů ve směru osy y.

Příklad: Animace kolečka

```
import tkinter, time, random

g = tkinter.Canvas(width=300, height=300)
g.pack()

x=random.randint(0, 300)
y=random.randint(0, 300)
kolo=g.create_oval(x-5, y-5, x+5, y+5, fill='red', outline='red')
dx=random.randint(1, 5)
```

```
dy=random.randint(1, 5)
for i in range(10000):
    g.move(kolo, dx, dy)
    g.update()
    x = x+dx
    y = y+dy
    if x<0 or x>300:
        dx = -dx
    if y<0 or y>300:
        dy = -dy
    time.sleep(0.1)
```

Červené kolečko se pohybuje po grafické ploše jistým směrem (podle souřadnic změny `dx` a `dy`), a pokud narazí na hranice grafické plochy, odráží se do nového směru. Kolečko je pouze jedno a směna polohy je prováděna pomocí metody `move()`.

Další změny lze podobným způsobem provádět pomocí následujících metod:

- `g.coords()` – změna souřadnic daného objektu;
- `g.itemconfig()` – změna parametrů daného objektu, například barvy textu nebo textového řetězce u textových objektů.

```
import tkinter

g=tkinter.Canvas()
g.pack()
pom = g.create_text(150, 50, text='Python', fill='green', font='arial
30 bold')
g.coords(pom, 200, 100)
g.itemconfig(pom, text='Python jinak!')
```

3.1.3 Události myší

Pokud potřebujeme, aby grafická plocha reagovala na kliknutí myši, musíme ji provázat s příslušnými událostmi. K tomu slouží metoda grafické plochy `bind()`, která má dva parametry: znakový řetězec s popisem události a název funkce (podprogramu), který se při dané události vykoná. Definice události je pak jednoduchá – po kliknutí do grafické plochy (řetězec `'<Button-1>'`) se na místě kliknutí (bod o souřadnicích `[event.x, event.y]`) vykreslí červené kolečko.

```
import tkinter

g = tkinter.Canvas(bg='white', width=400, height=400)
g.pack()
```

```
def klik(event):
    g.create_oval(event.x-5, event.y-5, event.x+5, event.y+5,
                  fill='red')

g.bind('<Button-1>', klik)
```

Pro naši práci budou důležité následující události myši:

- **kliknutí** (stisknutí tlačítka myši) – řetězec `'<Button-1>'`: 1 označuje levé tlačítko myši, 2 středové tlačítko a 3 pravé tlačítko myši;
- **tahání** (posun myši se zmáčknutým tlačítkem) – řetězec `'<B1-Motion>'`;
- **puštění** tlačítka myši – řetězec `'<ButtonRelease-1>'`.

Událost myši zrušíme zrušením odkazu na jméno příslušného volaného podprogramu nebo pomocí metody `unbind()`.

```
g.unbind('< Button-1>')
```

Úkoly k textu

- Vytvořte program, který po kliknutí do grafické plochy vykreslí křížek (jako v piškvorkách) se středem v bodě kliknutí.
- Vytvořte program, který bude kreslit úsečky. Při prvním kliknutí do grafické plochy si pouze zapamatuje souřadnice počátečního bodu úsečky. Při táhnutí myši vykresluje alternativu aktuální úsečky a při puštění tlačítka myší vykreslí celou úsečku.

Události kliknutí lze propojit také na objekty želva. Každému vytvořenému objektu želva lze definovat událost při kliknutí pomocí metody objektu želva `onclick()`, který má jediný parametr – název podprogramu pro obsluhu dané události. Události kliknutí reagují při kliknutí na konkrétní objekt želva na grafické ploše (na její šipku).

```
import turtle

def klik(x, y):
    wn.title("Kliknuto na bod {0}, {1}".format(x, y))
    t.left(42)
    t.forward(30)

t = turtle.Turtle()
t.color("purple")
t.onclick(klik)
```

Po kliknutí na objekt želva se želva otočí vlevo o 42° a posune se dopředu.

3.1.4 Události klávesnice

Události od klávesnice je opět nutné provázat s příslušným objektem, v našem případě s grafickou plochou. K tomu slouží metoda `bind_all()`, která má opět dva parametry: znakový řetězec se znakem (názvem) klávesy a název funkce, která se má při zachycení dané události vykonat. Zmáčknutí libovolné klávesy je detekováno řetězcem `'<Key>'`. Další klávesy lze snadno odvodit od jejich názvu.

Standardním využitím události klávesnice je pohyb objektů po grafické ploše pomocí kurzorových šipek. Pro tento pohyb je nutné znát aktuální pozici objektu nebo lépe jeho změnu. Tyto údaje je nutné definovat globálně pro celý program. Výhodné je proto použít třídu.

Ve třídě jsou globální proměnné označené „předponou“ `self` – jsou vlastněny třídou. Parametr `self` je nutným parametrem všech metod třídy.

Každá třída obsahuje metodu pro inicializaci svých globálních proměnných (atributů třídy), tzv. konstruktor. V našem příkladu jsou atributy třídy grafická plocha `g`, pohybující se objekt `id` a souřadnice změny jeho polohy `dx` a `dy`. Třída Program obsahuje následující metody:

- `__init__(self)` – konstruktor třídy, inicializuje hodnoty atributů, nastavuje grafickou plochu a inicializuje zachytávání událostí z klávesnice pro jednotlivé kurzorové šipky;
- `kresli(self)` – metoda posune objekt na nové místo;
- `udalost_vlevo(self, e)`, `udalost_vpravo(self, e)`, `udalost_nahoru(self, e)`, `udalost_dolu(self, e)` – metody pro obsluhu jednotlivých událostí z klávesnice, parametr `e` odpovídá zachycené události.

Pro spuštění třídy je potřeba vytvořit objekt třídy `p` voláním konstruktoru. Ten zavoláme jménem třídy – konstrukce `p=Program()`.

```
import tkinter

class Program:

    def __init__(self):
        self.g = tkinter.Canvas(bg='white',width=400,height=400)
        self.dx = 0
        self.dy = 0
        self.g.pack()
        self.g.bind_all('<Left>', self.udalost_vlevo)
```

```
        self.g.bind_all('<Right>', self.udalost_vpravo)
        self.g.bind_all('<Up>', self.udalost_nahoru)
        self.g.bind_all('<Down>', self.udalost_dolu)
        self.id = self.g.create_oval(200,200,210,210, fill='red',
                                     outline='red')

    def kresli(self):
        self.g.move(self.id, self.dx, self.dy)
        self.dx = 0
        self.dy = 0

    def udalost_vlevo(self,e):
        self.dx = -10
        self.kresli()

    def udalost_vpravo(self,e):
        self.dx = 10
        self.kresli()

    def udalost_nahoru(self,e):
        self.dy = -10
        self.kresli()

    def udalost_dolu(self,e):
        self.dy = 10
        self.kresli()

p = Program()
```

Úkoly k textu

- Upravte předchozí program tak, aby červená kulička nemohla „utéct“ z vymezené grafické plochy, tj. kontrolujte její aktuální souřadnice.
- Doplňte program o další drobné grafické objekty (modré kuličky), které zaznamenáte v poli a vykreslíte. Červená kulička bude při svém pohybu sbírat body za každou modrou kuličku, do které narazí.

Události z klávesnice lze využít také pro objekt želva. Události zachytáváme pomocí metody `onkey()`, která je propojena na grafickou plochu želvy (`screen`). Pro události z klávesnice je nutné zavolat také „naslouchače“ pomocí metody `listen()`.

Následující program umožňuje ovládat želvu pomocí kurzorových kláves dopředu, vlevo a vpravo (otočí želvu vždy o 45° - vlevo nebo vpravo). Program skončí po zmáčknutí klávesy q.

```
import turtle

def h1():
```

```
   t.forward(30)

def h2():
   t.left(45)

def h3():
   t.right(45)

def h4():
    wn.bye()

t = turtle.Turtle()
t.screen.onkey(h1, "Up")
t.screen.onkey(h2, "Left")
t.screen.onkey(h3, "Right")
t.screen.onkey(h4, "q")
t.screen.listen()
```

3.1.5 Události času

Práci s časem jsme si ukázali na procesu animace kuličky. Vykonávání procesu lze pozastavit na určitou dobu – metoda `time.sleep()`.

Podobnou možnost nabízí i objekt želva. Zde máme možnost pozastavit výpočet pomocí metody `ontimer()`, parametrem metody je počet milisekund pro pozastavení výpočtu. Metoda opět náleží grafické ploše želvy. Pokud chceme výpočet pozastavovat opakovaně, musíme to provést v cyklu nebo pomocí rekurze – viz příklad, kde želva `t2` se prochází náhodně po grafické ploše a želva `t1` se vždy na jistou dobu pozastaví, pak se natočí směrem k `t2` a udělá krok. Obě želvy se chovají, jakoby měly „motorek“, který je v pravidelných intervalech pohání.

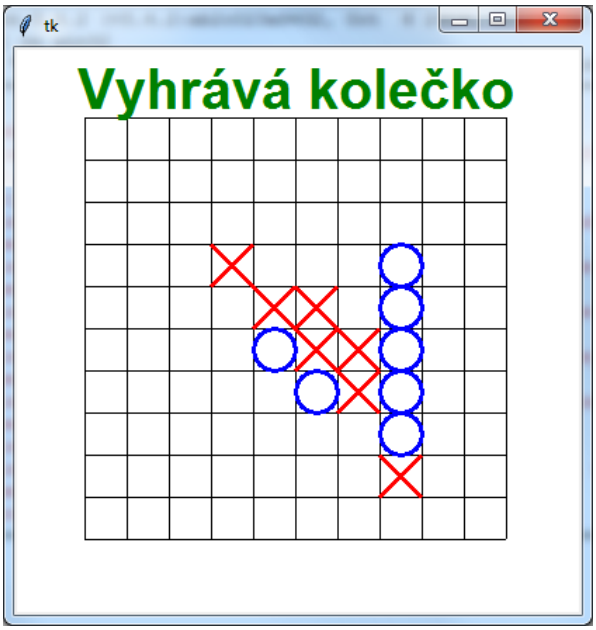
```
import turtle, random
def h1():
    t1.seth(t1.towards(t2))
    t1.forward(20)
    t1.screen.ontimer(h1, 2000)
def h2():
    t2.right(random.randint(0, 360))
    t2.forward(10)
    t2.screen.ontimer(h2, 200)

t1 = turtle.Turtle()
t1.color("purple")
t1.pensize(3)
t2 = turtle.Turtle()
t2.color("red")
t2.pensize(3)
h1()
h2()
```

3.2 Grafické hry

3.2.1 Piškvorky

Pro hru piškvorky definujeme novou třídu. Jejími atributy budou grafická plocha `g`, dvourozměrné pole `pole` celých čísel odpovídající polohám křížků a koleček na grafické ploše (0 – není ani křížek ani kolečko, 1 – křížek, 2 – kolečko) a logická proměnná `kdo` (`True` – na tahu je křížek, `False` – na tahu je kolečko). Všechny atributy inicializujeme v konstruktoru `__init__()`.

Hrací plán vykreslíme pomocí sítě – metoda `sit()` – 11 svislých a 11 vodorovných čar vymezí hrací plochu 10 x 10 polí.



Dále definujeme metody pro kreslení kolečka a křížku. Ty jsou vyvolávány střídavě metodou `kresli()` podle aktuální hodnoty atributu `kdo`. Metoda `kresli()` kontroluje vymezený prostor pro kliknutí a křížek nebo kolečko umístí na místo kliknutí pouze v případě, že bylo kliknuto do sítě na místo, kde žádný objekt prozatím není. Metoda `kresli()` je volána automaticky při kliknutí na grafickou plochu metodou `kliknuti()`, která nejprve upraví souřadnice kliknutí tak, aby křížek i kolečko byly zarovnány do sítě.

Metoda `kresli()` se stará také o kontrolu výherce. Zde je nutné zkontrolovat čtyři různé směry – vodorovně, svisle, šikmo zleva doprava a šikmo zprava doleva. To kontroluje metoda `kontrola()`. Pokud křížek nebo kolečko vyhrály, je program ukončen – vypíše se zpráva o výherci a zruší se zpracování události při kliknutí, viz metoda `konec()`.

Úkol k textu

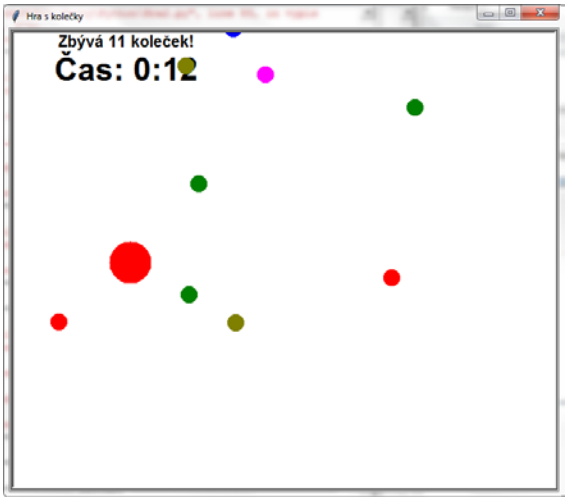
Vytvořte program pro hru piškvorky podle návodu a postupu zde uvedeného. Pokud si nebudete vědět rady a nebudete si jisti v kódu, využijte přiložený program. Doporučujeme ale vše si vyzkoušet osobně, pouze tak se lze v oblasti programování doopravdy vzdělávat a zdokonalovat.

```
import tkinter

class Piskvorky:
    kdo = True
    def __init__(self):
        self.g = tkinter.Canvas(bg='white', width=400, height=400)
                                self.g.pack()
        self.g.bind('<Button-1>', self.kliknuti
        self.pole = []
        for i in range(10):
            self.pole.append([0]*10)

    def __repr__(self):
        vysl = ''
        for i in range(10):
            for j in range(10):
                vysl = vysl+' '+str(self.pole[j][i])
            vysl += '\n'
        return vysl

    def sit(self):
        for i in range(11):
            self.g.create_line(50, 30*i+50, 350, 30*i+50
        for i in range(11):
            self.g.create_line(30*i+50, 50, 30*i+50, 350)

    def krizek(self, x, y):
        self.g.create_line(x-15, y-15, x+15, y+15, width=3,
                           fill='red')
        self.g.create_line(x-15, y+15, x+15, y-15, width=3,
                           fill='red')

    def kolo(self, x, y):
        self.g.create_oval(x-15, y-15, x+15, y+15, width=3, fill='',
                           outline='blue')

    def konec(self, cis):
        if cis==1:
            vypis='křížek'
        else:
            vypis='kolečko'
        self.g.create_text(200, 30, text='Vyhrává '+vypis,
                           fill='green', font='arial 30 bold')
        self.g.bind('<Button-1>', '')

    def pomKontrola(self, j, i, jj, ii, prvek):
```

```
        vrat = True
        for k in 1, 2, 3, 4:
            vrat = vrat and self.pole[j+k*jj][i+k*ii]==prvek
        if vrat:
            self.konec(prvek)

    def kontrola(self):
        for i in range(10):
            for j in range(10):
                prvek=self.pole[j][i]
                if prvek > 0:
                    if j+4<10:
                        self.pomKontrola(j, i, 1, 0, prvek)
                        if i+4<10:
                            self.pomKontrola(j, i, 1, 1, prvek)
                    if i+4<10:
                        self.pomKontrola(j, i, 0, 1, prvek)
                        if j-4>=0:
                            self.pomKontrola(j, i, -1, 1, prvek)

    def kresli(self, x, y):
        xx = int((x-50)/30)
        yy = int((y-50)/30)
        if x>50 and x<350 and y>50 and y<350 and self.pole[xx][yy]==0:
            if self.kdo:
                self.krizek(x, y)
                self.pole[xx][yy]=1
            else:
                self.kolo(x, y)
                self.pole[xx][yy]=2
            self.kdo = not self.kdo
            self.kontrola()
        # print(self)

    def kliknuti(self, event):
        xx=int((event.x-50)/30)*30+50+15
        yy=int((event.y-50)/30)*30+50+15
        self.kresli(xx, yy)

p = Piskvorky()
p.sit()
```

3.2.2 Hra kolečka

Hra využívá události klávesnice. Velké kolečko se pohybuje pomocí kurzorových šipek nahoru, dolů, vlevo nebo vpravo. Kromě něho jsou na grafické ploše další barevná kolečka. Úkolem je sesbírat všechna malá kolečka pomocí velkého kolečka. Problém je komplikován tím, že velké kolečko „sežere“ malé kolečko pouze v případě, že mají stejnou barvu. V opačném případě kolečko na grafické ploše přibyde. Barva velkého



kolečka se generuje náhodně a mění se každých 5 sekund. Úkolem je sesbírat kolečka v nejkratším možném čase.

Úkol budeme řešit pomocí objektů želva. Malá kolečka budou objekty nově vytvořené třídy `MojeZelva`. Ta je potomkem třídy `turtle.Turtle` a dědí všechny její vlastnosti. Nově mají kolečka (želvy) nastavený tvar kolečka, nastavenou barvu a automatický pohyb po grafické ploše (metoda `pohyb()`).

Vlastní hra – třída `Hra` – má následující důležité atributy (inicializované v konstruktoru):

- `cas` pro měření času, na začátku se zde zaznamená doba, kdy hra začala;
- pole `t` – pole malých koleček;
- `cela` – je velké kolečko ovládané pomocí události z klávesnice;
- `konec` – označuje ukončení programu, sesbírání všech koleček.

Metoda `pridejZelva()` vytvoří a přidá na plochu nové malé kolečko.

Velké kolečko – objekt `cela` – využívá metodu `zmenaBarvy()` pro změnu barvy v pravidelných intervalech 5 sekund. Velké kolečko je ovládáno pomocí kurzorových šipek a jsou k němu proto připojeny příslušné metody pro obsluhu zachycených událostí (metody označené slovem `udalost_`). Tyto metody nastavují pouze změnu v souřadnicích velkého kolečka, přesné nastavení souřadnic a omezení na pohyb pouze v rámci grafické plochy je záležitostí metody `zmenaPozice()`.

Metoda `vypis()` vypisuje informace o aktuálním stavu hry na grafickou plochu. Je volána v pravidelném intervalu 100 milisekund, tj. cca 10krát za vteřinu. Pro aktualizaci výpisů používáme metody `coords()` a `itemconfig()`, nelze dokola vytvářet podobné objekty. Kromě aktualizace výpisů se zabývá také kontrolou srážek velkého kolečka s malými kolečky (metoda `kontroluj()`) a změnou polohy velkého kolečka (metoda `zmenaPozice()`).

Metoda `kontrola()`kontroluje vzdálenost středu velkého kolečka s malými kolečky. Pokud je vzdálenost malá (došlo ke srážce) malé kolečko je odstraněno ze hry, naopak při nesprávném střetu (podle barvy koleček) se malé kolečko přidává do hry. Tato metoda je vyvolaná pokaždé při změnách v pohybu velkého kolečka.

Všechny procesy jsou zastaveny v okamžiku, kdy zmizí všechna malá kolečka, tj. v okamžiku, kdy proměnná `konec` má hodnotu `False`.

Úkol k textu

Vytvořte program pro hru s kolečky. Pokud si nebudete vědět rady a nebudete si jisti v kódu, využijte přiložený program. Doporučujeme ale vše si vyzkoušet osobně, pouze tak se lze v oblasti programování doopravdy vzdělávat a zdokonalovat. Vhodné jsou i pouze drobné úpravy, které v kódu dokážete provést.

```
import turtle, random, time

class MojeZelva(turtle.Turtle):
    def __init__(self):
        super().__init__()
        self.dx = random.randint(0, 10)-5
        self.dy = random.randint(0, 10)-5
        self.pu()
        self.shape('circle')
        self.shapesize(1, 1, 1)
        pom = random.choice(('blue', 'red', 'yellow', 'green',
                             'orange', 'magenta', 'olive'))
        self.color(pom, pom)

    def pohyb(self):
        self.setpos(self.xcor()+self.dx, self.ycor()+self.dy)
        if self.xcor()<-300 or self.xcor()>300:
            self.dx = -self.dx
        if self.ycor()<-300 or self.ycor()>300:
            self.dy = -self.dy
        if self.isvisible():
            self.screen.ontimer(self.pohyb, 100)

class Hra:
    barva = 'black'
    zmena = False
    cas = time.time()
    konec = False

    def __init__(self):
        self.t=[]
        self.cela=turtle.Turtle()
        turtle.delay(0)
```

```
        for i in range(10):
            self.pridejZelva()
        self.dx = 0
        self.dy = 0
        self.cela.pu()
        self.cela.setpos(0, 0)
        self.cela.shape('circle')
        self.cela.shapesize(2.5, 2.5, 2)
        self.cela.screen.onkey(self.udalost_vlevo, 'Left')
        self.cela.screen.onkey(self.udalost_vpravo, 'Right')
        self.cela.screen.onkey(self.udalost_nahoru, 'Up')
        self.cela.screen.onkey(self.udalost_dolu, 'Down')
        self.cela.screen.title('Hra s kolečky')
        self.cela.screen.listen()
        okno = self.cela.screen.getcanvas()
        self.textCas = okno.create_text(-200, -240, text='Čas: 0:00',
                                        fill='black', font='arial 30 bold')
        self.textKol =  okno.create_text(-200, -275,
         text='Zbývá 10 koleček!', fill='black', font='arial 15 bold')
        self.zmenaBarvy()
        self.vypis()

    def pridejZelva(self):
        nova=MojeZelva()
        nova.setpos(random.randint(-200,200),random.randint(-200,200))
        nova.pohyb()
        self.t.append(nova)

    def zmenaBarvy(self):
        self.barva = random.choice(('blue', 'red', 'yellow', 'green',
                                    'orange', 'magenta', 'olive'))
        self.zmena = False
        self.cela.color(self.barva, self.barva)
        if not self.konec:
            self.cela.screen.ontimer(self.zmenaBarvy, 5000)

    def kontrola(self):
        for i in range(len(self.t)):
            if self.t[i].distance(self.cela)<30:
                if self.t[i].pencolor()==self.cela.pencolor():
                    self.t[i].hideturtle()
                    del self.t[i]
                    return
                else:
                    if not self.zmena:
                        self.pridejZelva()
                        self.zmena = True

    def vypis(self):
        self.zmenaPozice()
        self.kontrola()
        pom=time.time()-self.cas
        s=str(int(pom/60))+":"+'{:02d}'.format(int(pom)%60)
        okno = self.cela.screen.getcanvas()
        okno.itemconfig(self.textCas, text='Čas: '+s)
        if len(self.t)==0:
            okno.coords(self.textKol, 0, 0)
            okno.itemconfig(self.textKol, text='Vyhrál si!!!',
                                      fill='red', font='arial 50 bold')
```

```
            self.konec = True
        else:
            okno.itemconfig(self.textKol,
                            text='Zbývá '+str(len(self.t))+' koleček!')
        if not self.konec:
            self.cela.screen.ontimer(self.vypis, 100)

    def zmenaPozice(self):
        self.cela.setpos(self.cela.xcor()+self.dx,
                         self.cela.ycor()+self.dy)
        if self.cela.xcor()<-300:
            self.cela.setpos(-300, self.cela.ycor())
        if self.cela.xcor()>300:
            self.cela.setpos(300, self.cela.ycor())
        if self.cela.ycor()<-300:
            self.cela.setpos(self.cela.xcor(), -300)
        if self.cela.ycor()>300:
            self.cela.setpos(self.cela.xcor(), 300)

    def udalost_vlevo(self):
        self.dx = -5
        self.zmenaPozice()

    def udalost_vpravo(self):
        self.dx = 5
        self.zmenaPozice()

    def udalost_nahoru(self):
        self.dy = 5
        self.zmenaPozice()

    def udalost_dolu(self):
        self.dy = -5
        self.zmenaPozice()

Hra()
```

Shrnutí kapitoly

- Kromě objektu želva lze pro práci s grafikou využít přímo grafickou plochu a její metody. Vše je obsaženo v modulu `tkinter`. Grafickou plochu pro kreslení musíme inicializovat (vytvořit). Na ní pak můžeme kreslit – přidávat další grafické objekty a definovat jejich vlastnosti.
- Pro zachytávání událostí myši nebo klávesnice je potřeba připojit k příslušným objektům tzv. naslouchače těchto událostí. Tyto naslouchače při zachycení příslušných událostí vyvolají definované metody.
- Události myši a klávesnice lze zpracovávat přímo na grafické ploše nebo také přímo v objektech želva.
- Složitější programy je vhodné strukturovat do tříd. Ty mají své atributy (globální proměnné označené `self`) a metody. Speciální metodou je konstruktor (`__init__()`) pro inicializaci hodnot stavových proměnných.

- V závěru kapitoly jsme definovali dvě hry – piškvorky a hru kolečka.

Korespondenční úkol

Navrhněte jednoduchou hru. Inspirovat se můžete známými hrami nebo se můžete pokusit pouze o tvorbu hry podle příkladů v textu. Připravte si vhodné obrázky objektů i pozadí a hru se pokuste vytvořit. Hra by měla být ovládána pomocí myši nebo klávesnice.

Ověřte funkčnost programu a hotový program ve formátu PY odevzdejte ke kontrole tutorovi.

Citovaná a doporučená literatura

BLAHO, A. *Programovanie v Pythone.* Bratislava: MFF UK, 2014. Dostupné na www: <http://python.input.sk/>.

Programovací jazyk Python. Oficiální stránky, 2014. Dostupné na www: <https://www.python.org/>

WENTWORTH, P. & kol. *Learning with Python 3.* 2012. Dostupné na www: <http://openbookproject.net/thinkcs/python/english3e/>