**Třídy a metody**

Vytvoření instance třídy, práce s atributy a volání metod:

**>>>** premenna = Meno\_triedy(...) *# vytvoření instance třídy*

**>>>** premenna.atribut = hodnota *# vytvoření /změna hodnoty atributu*

**>>>** premenna.metoda(parametre) *# volání metódy*

**Metody:**

Jsou soukromé funkce definované v třídě.

Metoda má při definování uvedený jako první parametr ‚self‘, který reprezentuje samotnou instanci třídy pro kterou je zavolaná.

V pythoně všechny funkce (tzn i metody) můžeme rozdělit do dvou kategorií:

1. Modifikátor – který mění nějakou hodnotu atributu (nebo atributů)
2. Pravá funkce – které nemění žádné atributy ani globální proměnné (nejčastěji vrací hodnotu)

Některé metody při zakládání třídy se přidají automaticky. Těmto metodám se říká magické a začínají a končí dvěma podčárkovníky. Python s nimi definuje speciální chování a slouží k lepší integraci.

Například metoda \_\_init\_\_() inicializuje atributy instance a volá se hned po vytvoření objektu.

**Magická metoda \_\_str\_\_():**

Slouží k tomu, aby python pochopil, že řetězcová reprezentace našeho objektu by mohla být výsledkem nějaké naší metody a tu by i automaticky použil například při volání funkce ‚print()‘ nebo ‚str()‘.

Když zadefinujem magickou metodu \_\_\_str\_\_(), pak ve chvíli, kdy Python bude potřebovat řetězcovou reprezentaci objektu, zavolá tuto metodu.

Výsledkem metody musí být řetězec.

**class** **Cas**:

**def** \_\_init\_\_(self, hodiny, minuty):

self.hodiny = hodiny

self.minuty = minuty

**def** \_\_str\_\_(self):

**return** f'*{*self.hodiny*}*:*{*self.minuty*:*02*}*'

**def** vypis(self):

print('cas je', self) *# Python tu za nas urobil self.\_\_str\_\_()*

Pokud máme definovanou metodu \_\_str\_\_(), tak při metodě ‚vypis()‘ stačí zadat příkaz ‚self‘, načež Python prohledá třídu, zda má definovanou metodu \_\_str\_\_() a pokud ji najde, vyvolá z ní výsledek.

V každé itegrované třídě Pythonu je zadefinovaná tato metoda a tak je možné u každého objektu vytisknout jeho řetězcový výstup.

Standartní funkce str() bychom si pak mohli představit nějak takto:

**def** str(objekt=''):

**return** objekt.\_\_str\_\_()

**Volání metody z jiné metody:**

V rámci třídy je možné v jedné metodě volat jinou metodu (patřící do stejné třídy).

Aby bylo možné v rámci třídy zavolat nějakou metodu, musí tato metoda začínat slovem ‚self‘ a za ním tečkou. Bez ‚self.‘ voláme ne metodu třídy, ale funkci v globálním nebo lokálním prostoru.

Metody tedy voláme ve tvaru: self.metoda()

**Příklad s časem:**

**class** **Cas**:

**def** \_\_init\_\_(self, hodiny=0, minuty=0, sekundy=0):

self.sek = abs(3600\*hodiny + 60\*minuty + sekundy)

**def** \_\_str\_\_(self):

**return** f'*{*self.sek // 3600*}*:*{*self.sek // 60 % 60*:*02*}*:*{*self.sek % 60*:*02*}*'

**def** sucet(self, iny):

**return** Cas(sekundy=self.sek + iny.sek)

**def** rozdiel(self, iny):

**return** Cas(sekundy=self.sek - iny.sek)

**def** vacsi(self, iny):

**return** self.sek > iny.sek

zoznam = [Cas(8, 10)]

**for** i **in** range(14):

zoznam.append(zoznam[-1].sucet(Cas(0, 50)))

**for** cas **in** zoznam:

print(cas, end=' ')

**Třídní a instanční atributy:**

Třídy jsou kontejnery na soukromé funkce (metody) a instance jsou kontejnery na soukromé proměnné (atributy).

Atributy můžeme definovat buď v třídě – v tu chvíli jsou to ‚třídní atributy‘,   
nebo na úrovni instance – v tu chvíli jsou to instanční atributy.

Třída nevidí a ani nevypisuje atributy instance.  
Instance vidí atributy definované na úrovni třídy, může s nimi pracovat a může si je přes proměnou přiřadit do svého kontejneru. Pak je možné hodnotu atributu měnit bez toho, aby se měnil třídní atribut.

Třídní atributy často definujeme už při definování třídy:

**class** **Test**:

z = 300

**def** \_\_init\_\_(self, x, y):

self.x = x

self.y = y

**def** \_\_str\_\_(self):

**return** f'test *{*self.x*}*,*{*self.y*}*,*{*self.z*}*'

Dobrou zásadou při definování třídy a metod je NEPOUŽÍVAT ŽÁDNÉ GLOBÁLNÍ PROMĚNNÉ.

Pokud tedy nechceme v metodách třídy pracovat s globální proměnou (např. ‚canvas‘), můžeme tuto globální proměnou přidat jako paramteri do inicializace \_\_init\_\_(). Tím se ‚canvas‘ zapamatuje jako atribut pro každou vytvořenou instanci.

**import** **tkinter**

**import** **random**

**class** **Bodka**:

canvas = **None**

pocet\_modrych = pocet\_cervenych = 0

**def** \_\_init\_\_(self, x, y):

self.id = self.canvas.create\_oval(x - 5, y - 5, x + 5, y + 5)

**def** prefarbi(self):

**if** random.randrange(2):

farba = 'red'

Bodka.pocet\_cervenych += 1

**else**:

farba = 'blue'

Bodka.pocet\_modrych += 1

self.canvas.itemconfig(self.id, fill=farba)

Bodka.canvas = tkinter.Canvas()

Bodka.canvas.pack()

bodky = []

**for** i **in** range(100):

bodky.append(Bodka(random.randint(10, 300), random.randint(10, 250)))

**for** b **in** bodky:

b.prefarbi()

print('pocet modrych =', Bodka.pocet\_modrych)

print('pocet cervenych =', Bodka.pocet\_cervenych)

**Příklad s grafickými objekty:**

**import** **tkinter**

**class** **Kruh**:

canvas = **None**

**def** \_\_init\_\_(self, x, y, r, farba='red'):

self.x = x

self.y = y

self.r = r

self.farba = farba

self.id = self.canvas.create\_oval(

self.x - self.r, self.y - self.r,

self.x + self.r, self.y + self.r,

fill=self.farba)

**def** posun(self, dx=0, dy=0):

self.x += dx

self.y += dy

self.canvas.move(self.id, dx, dy)

**def** zmen(self, r):

self.r = r

self.canvas.coords(self.id,

self.x - self.r, self.y - self.r,

self.x + self.r, self.y + self.r)

**def** prefarbi(self, farba):

self.farba = farba

self.canvas.itemconfig(self.id, fill=farba)

Kruh.canvas = tkinter.Canvas()

Kruh.canvas.pack()

k1 = Kruh(50, 50, 30, 'blue')

k2 = Kruh(150, 100, 80)

k1.posun(30,10)

k2.zmen(50)

k1.prefarbi('green')

Pokud bychom kromě třídy ‚Kruh‘ vytvořili stejně i třídu ‚Obdelník‘, pak můžeme zkusit vytvořit i třídu ‚Skupina‘ pomocí které můžeme přidávat různé útvary do jendé struktury:

**import** **tkinter**

**class** **Skupina**:

**def** \_\_init\_\_(self):

self.zoznam = []

**def** pridaj(self, utvar):

self.zoznam.append(utvar)

canvas = tkinter.Canvas()

canvas.pack()

Kruh.canvas = Obdlznik.canvas = canvas

sk = Skupina()

sk.pridaj(Kruh(50, 50, 30, 'blue'))

sk.pridaj(Obdlznik(100, 20, 100, 50))

sk.zoznam[0].prefarbi('green')

sk.zoznam[1].posun(50)

V případě, že budeme potřebovat měnit více útvarů, stačí použít ‚for‘ cyklus:

**for** utvar **in** sk.zoznam:

utvar.posun(dy=15)

Nebo do třídy Skupina můžeme doplnit metody, které pracují se všemi útvary ve skupině:

**class** **Skupina**:

...

**def** prefarbi(self, farba):

**for** utvar **in** self.zoznam:

utvar.prefarbi(farba)

**def** posun(self, dx=0, dy=0):

**for** utvar **in** self.zoznam:

Případně, pokud chceme ve třídě ‚Skupina‘ pracovat jen s útvarmi konkrétní třídy (Kruh, Obdelnik), pak do těchto tříd stačí doplnit další atribut (zde například ‚typ‘):

**class** **Kruh**:

canvas = **None**

typ = 'kruh'

**def** \_\_init\_\_(self, x, y, r, farba='red'):

...

**class** **Obdlznik**:

canvas = **None**

typ = 'obdlznik'

**def** \_\_init\_\_(self, x, y, sirka, vyska, farba='red'):

...

**class** **Skupina**:

...

**def** posun\_typ(self, typ, dx=0, dy=0):

**for** utvar **in** self.zoznam:

**if** utvar.typ == typ:

utvar.posun(dx, dy)

**def** prefarbi\_typ(self, typ, farba):

**for** utvar **in** self.zoznam:

**if** utvar.typ == typ:

utvar.prefarbi(farba)