**Třídy a dědičnost**

Co už víme o třídách:

třídy jsou kontejnery atributů:

• funkce – těm říkáme metody

• proměnné – těm říkáme třídní atributy

• magické – těm říkáme magické atributy

třídy jsou vzorami na vytváření instancí

i instance jsou kontejnery atributů:

• většinou jsou to soukromé proměnné instancí

když nějaký atribut není v instanci definovaný, Python automaticky použije atribut z třídy

**Objektové programování** je charakterizované následujícími vlastnostmi:

1. Zapouzdření – údaje a funkce jsou zapouzdřené v jednom celku
2. Dědičnost – novou třídu nevytváříme z nuly, ale použijeme existující třídu
3. Polymorfizmus –

**Zapouzdření**

Pokud chceme, aby jsme mohli měnit atributy i bez použití metod, je dobré nadefinovat sadu metod, které při změně hodnoty, udělají i další funkce (např. volají metody za nás).

Pokud u některých atributů nechceme, aby uživatel měnil jejich hodnoty, je možné, tyto atributy označit jako neveřejné (soukromé). Všeobecně se pro tento případ používá podtržítko umístěné před názvem atributu (\_xy). Pro práci s neveřejnými atributy je možné definovat speciální atribut **property**

**Property**

může obsahovat více parametrů, nejčastěji to bude **getter** a **setter:**

**getter** – je parametr který vrací hodnotu atributu

**setter** – je metoda umožňující změnu hodnoty atributu

Property, getter a setter mají dvě podoby zápisu:

1. standartní – nejpreve zadefinujeme metody pro získání a změnu hodnot a po té je zapíšeme jako parametry do přiřazovacího příkazu property()

getter:

**def** daj\_farbu(self):

**return** self.\_farba

setter:

**def** zmen\_farbu(self, farba):

self.\_farba = farba

self.canvas.itemconfig(self.\_id, fill=farba)

definice property:

farba = property(daj\_farbu, zmen\_farbu)

1. zkrácené – nad metodou pro získání hodnoty (getter) zapíšeme zavináč a property, tím vytvoříme jméno property a hodnotu getter. Při přidání setter pak je potřeba za zavináč uvést nejprve jméno property, tečku a za ní slovo setter

getter:

**@property**

**def** farba(self):

**return** self.\_farba

setter:

**@farba**.setter

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

self.\_farba = farba

self.canvas.itemconfig(self.\_id, fill=farba)

**Dědičnost**

v pythoně, vždy když vytváříme novou třídu, vytváříme ji na základě jiné třídy, z které si nová třída převezme její atributy, které je možné v nové třídě změnit.

Nejzákladnější definovaná třída pythonu, od které se ostatní odvíjejí, se jmenuje **object,** ta má již v sobě předdefinované následující magické funkce:

**>>>** dir(objekt)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_']

Každá námi vytvořená třída atomaticky tyto magické funkce přebírá a obsahuje taky.

Pokud chceme vytvořit třídu na základu jiné třídy, uvádí se tato třída do závorek:

**class** **NovaTrida**(PuvodniTrida):

Toto neplatí u vytváření nové třídy na základu třídy **objekct**, v takovémto případě není nutné tuto třídu uvádět do závorek, protože to Python dělá automaticky za nás.

* Třída z které vytváříme nějakou novou třídu se říká **základní třída,** nebo **bázová třída,** nebo **super třída** (base class, super class)
* Třída která vznikne děděním z jiné třídy se říká **odvozená třída**, nebo **pod třída** (derived class, subclass)
* Někdy se také popisují tyto dvě třídy jako **rodič** a **potomek**

**Odvozená třída**

Začneme definicí jednoduché základní třídy:

**class** **Bod**:

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

self.x, self.y = x, y

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

**return** f'Bod(*{*self.x*}*, *{*self.y*}*)'

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

self.x += dx

self.y += dy

Nyní vytvoříme novou třídu z této třídy:

**class** **FarebnyBod**(Bod):

**def** zmen\_farbu(self, farba):

self.farba = farba

Díky tomuto zápisu nová třída (potomek) získává atributy své rodičovské třídy, takže kromě nové metody pro změnu barvy, můžeme používat všechny funkce z té předešlé.

Zděděné metody je možné v nové třídě i předefinovat – například můžeme změnit inicializaci \_\_init\_\_:

**class** **FarebnyBod**(Bod):

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

self.x = x

self.y = y

self.farba = farba

**def** zmen\_farbu(self, farba):

self.farba = farba

A pokud jako v tomto případě používáme hodnoty atributů předešlé třídy, je lépe definovat tyto hodnoty z původní třídy, pomocí její instance a metody \_\_init\_\_:

**class** **FarebnyBod**(Bod):

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

Bod.\_\_init\_\_(self, x, y)

self.farba = farba

Co se dá zapsat univerzálněji pomocí standardní funkce super(), která říká, udělej na tomto místě přesně to, co by udělal můj rodič:

**class** **FarebnyBod**(Bod):

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

super().\_\_init\_\_(x, y)

self.farba = farba

**Grafické objekty**

pokud bychom chtěli využít dědičnost pro grafické objekty kreslené do grafické plochy Canvas, pak dobrým způsobem, je vytvořit nejprve třídu, která v sobě definuje všechno to, co mají další objekty společné:

**class** **Utvar**:

canvas = **None**

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

self.\_x, self.\_y, self.\_farba = x, y, farba

self.\_id = **None**

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

self.\_x += dx

self.\_y += dy

self.canvas.move(self.\_id, dx, dy)

**def** zmen\_farbu(self, farba):

self.\_farba = farba

self.canvas.itemconfig(self.\_id, fill=farba)

Utvar.canvas = tkinter.Canvas(width=400, height=400)

Utvar.canvas.pack()

A po té definovat třídy jednotlivých objektů již na základu této rodičovské třídy a v nových třídách už pak stačí jen dodefinovat to, čím se jednotlivé objekty liší:

**class** **Kruh**(Utvar):

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

super().\_\_init\_\_(x, y, farba)

self.\_r = r

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

self.\_x - self.\_r, self.\_y - self.\_r,

self.\_x + self.\_r, self.\_y + self.\_r,

fill=farba)

**def** zmen\_r(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)

**class** **Obdlznik**(Utvar):

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

super().\_\_init\_\_(x, y, farba)

self.\_sirka, self.\_vyska = sirka, vyska

self.\_id = self.canvas.create\_rectangle(

self.\_x, self.\_y,

self.\_x + self.\_sirka, self.\_y + self.\_vyska,

fill=farba)

**def** zmen\_velkost(self, sirka, vyska):

self.\_sirka, self.\_vyska = sirka, vyska

self.canvas.coords(self.\_id,

self.\_x, self.\_y,

self.\_x + self.\_sirka, self.\_y + self.\_vyska)

**Testování typu instance**

pomocí standardní funkce type() můžeme testovat, zda je instance určitého typu:

**>>>** t1 = Kruh(10, 20, 30)

**>>>** type(t1) == Kruh

True

**>>>** type(t1) == Utvar

False

takto ale můžeme testovat pouze přímé rodiče, pokud bychom chtěli testovat i vzdálenější předky, musíme použít standardní funkci **isinstance(i, t),** která zjistí, zda je instance(i) typu (t) a nebo je typem jeho předka:

**>>>** isinstance(t1, Kruh)

True

**>>>** isinstance(t1, Utvar)

True

Zažité je používat pro zjištění instance v obou případech používat druhou metodu.

Příklad použití funkce u změny barvy útvaru, kdy v první případě měníme všechny útvary seznamu:

**def** zmen\_farbu(self, farba):

**for** utvar **in** self.\_zoznam:

utvar.zmen\_farbu(farba)

A za pomocí funkce **isinstance,** můžeme měnit jen ty útvary, které odpovídají určitému typu:

**def** zmen\_farbu\_typ(self, typ, farba):

**for** utvar **in** self.\_zoznam:

**if** isinstance(utvar, typ):

utvar.zmen\_farbu(farba)

**Příklad odvození třídy od Turtle:**

**import** **turtle**

**class** **MojaTurtle**(turtle.Turtle):

**pass**

tímto jsme definovali novou třídu MojaTurtle, která přebrala všechny atributy z třídy turtle, takže v tuto chvíli jsou identické a vše co mi unožňuje třída trutle i umožňuje i třída MojaTurtle.

V této nové třídě, pak můžeme rozšířit metody předdefinováním určitých postupů, například pro nakreslení čtverce:

**def** stvorec(self, velkost):

**for** i **in** range(4):

self.fd(velkost)

self.rt(90)

Nebo binárního stromu:

**def** strom(self, n, d):

self.fd(d)

**if** n > 0:

self.lt(40)

self.strom(n-1, d\*0.6)

self.rt(90)

self.strom(n-1, d\*0.7)

self.lt(50)

self.bk(d)

Či, domečku:

**def** domcek(self, dlzka):

**for** uhol **in** 90, 90, 90, 30, 120, -60:

self.fd(dlzka)

self.rt(uhol)

V nové třídě, také můžeme předdefinovat jinou výchozí pozici pera:

**def** \_\_init\_\_(self, x=0, y=0):

super().\_\_init\_\_()

self.speed(0)

self.pu()

self.setpos(x, y)

self.pd()

Nebo pozměnit již existující funkce, například funkci pro posun do předu, v tomto případě bude namísto rovné čáry, kreslit klikatou:

**def** fd(self, dlzka):

**while** dlzka >= 5:

self.lt(60)

super().fd(5)

self.rt(120)

super().fd(5)

self.lt(60)

dlzka -= 5

super().fd(dlzka)

a v tomto případě, bude kreslit za pomoci modulu random namísto jedné čáry 3, lehce se překrývající se čáry:

**def** fd(self, dlzka):

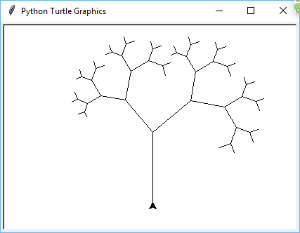
super().fd(dlzka)

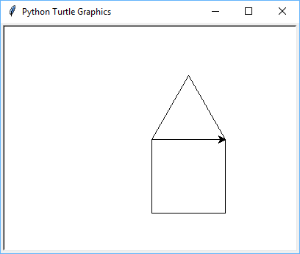
self.rt(180 - random.randint(-3, 3))

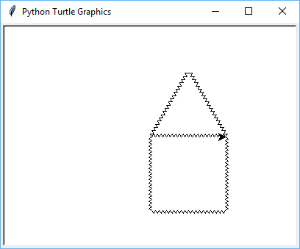
super().fd(dlzka)

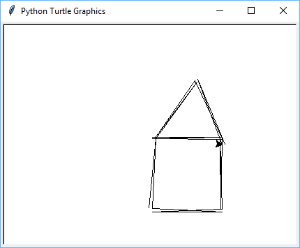
self.rt(180 - random.randint(-3, 3))

super().fd(dlzka)

[](http://python.input.sk/_images/16_02.png)

[](http://python.input.sk/_images/16_03.png)

[](http://python.input.sk/_images/16_04.png)

[](http://python.input.sk/_images/16_05.png)