Základné pojmy objektového programovania v Pythone

- premenné v Pythone sú vždy referencie na príslušné hodnoty
- pre rôzne typy máme v Pythone definované:
  – operácie: 7 * 8 + 9, 'a' * 8 + 'b', 7 * [8] + [9]
  – funkcie: len('abc'), sum(pole), min(ntica)
  – metódy:'117234'.split(),pole.append('novy'),g.create_line(1,2,3,4), t.fd(100)
- funkcia type(hodnota) vráti typ hodnoty

V Pythone sú všetky typy objektové, t.j. popisujú objekty, a takýmto typom hovoríme **trieda** (po anglicky **class**).
Všetky **hodnoty** (teda aj premenné) sú nejakého objektového typu, teda typu trieda, hovoríme, že sú **inštancie triedy**.

Zadefinujeme vlastnú triedu:

```
class Student:
        pass
```

- vytvorili sme prázdnu triedu, nový typ Student.
- keďže máme typ, môžeme vytvoriť premennú typu Student, na ktorú do premennej priradíme referenciu

```
fero=Student()
type(fero)
<class '__main__.Student'>
```

- inštanciu sme vytvorili volaním premenná=meno_typu()

Medzi pythonistami je ale dohoda, že nové typy, ktoré budeme v našich programoch definovať, budeme zapisovať s **prvým písmenom veľkým**. Preto sme zapísali napr. typ Student.

```
import turtle
t = turtle.Turtle()
```

Premenná t je referenciou na objekt triedy Turtle, ktorej definícia sa nachádza v module turtle (preto sme museli najprv urobiť import turtle, aby sme dostali prístup k obsahu tohto modulu). Už vieme, že t je inštanciou triedy Turtle.

Atribúty

O objektoch hovoríme, že sú to **kontajnery na dáta**. V našom prípade premenná fero je referenciou na prázdny kontajner. Pomocou priradenia môžeme objektu vytvárať nové súkromné premenné, tzv. atribúty. Takéto súkromné premenné nejakého objektu sa správajú presne rovnako ako bežné premenné, ktoré sme používali doteraz, len sa ***nenachádzajú v hlavnej pamäti*** (v globálnom mennom priestore) ale v ***pamäti objektu***.

Atribút vytvoríme tak, že za meno objektu fero zapíšeme meno tejto súkromnej premennej, pričom medzi nimi musíme zapísať bodku:

```
fero=Student()
type(fero)
<class '__main__.Student'>

fero.meno='Frantisek'
print(fero)
<__main__.Student object at 0x102715cc0>
print(fero.meno)
Frantisek

fero.priezvisko='Pocitacovy'
print(fero.meno,' ',fero.priezvisko)
Frantisek   Pocitacovy
```

Objekt fero teraz obsahuje dve ***súkromné premenné*** meno a priezvisko.

Aby sme ich vedeli slušne vypísať, môžeme vytvoriť pomocnú funkciu vypis:

```
def vypis(st):
    print('Volam sa ',st.meno,st.priezvisko)
```

Do tejto funkcie by sme mohli poslať ako parameter hodnotu ľubovoľného typu nielen Student: táto hodnota ale musí byť objektom s atribútmi meno a priezvisko, inak dostávame takúto chybu:

```
i=111
vypis(i)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    vypis(i)
  File "/Users/romankrakovsky/Python/trieda_student.py", line 5, in vypis
    print('Volam sa ',st.meno,st.priezvisko)
AttributeError: 'int' object has no attribute 'meno'
```

Vytvorme ďalšiu inštanciu triedy Student. Aj zuzka je objekt typu Student - je to zatiaľ prázdny kontajner atribútov. Ak zavoláme funkciu vypis nad objektom zuzka, ktorý nemá súkromné atribúty, dostaneme chybu. Až po pridaní súkromných atribútov meno a priezvisko môžeme použiť funkciu vypis().

```
zuzka=Student()
vypis(zuzka)
AttributeError: 'Student' object has no attribute 'meno'
zuzka.meno='Zuzana'
zuzka.priezvisko='Mala'
vypis(zuzka)
Volam sa  Zuzana Mala
```

Objekty sú ***meniteľné (mutable)***.
Atribúty objektu sú súkromné premenné, ktoré sa správajú presne rovnako ako "obyčajné" premenné. Premenným môžeme meniť obsah, napr.

```
fero.priezvisko='Pocitacovy'
vypis(fero)
Volam sa  Ferdinand Pocitacovy
```

Premenná fero stále obsahuje referenciu na rovnaký objekt (kontajner), len sa trochu zmenil jeden z atribútov. Takejto vlastnosti objektov sme doteraz hovorili meniteľné (mutable):
• napr. polia sú mutable, lebo niektoré operácie zmenia obsah poľa ale nie referenciu na objekt (pole.append('abc') pridá do poľa nový prvok)
• ak dve premenné referencujú ten istý objekt (napr. priradili sme pole2 = pole), tak takáto mutable zmena jedného z nich zmení obe premenné
• väčšina doterajších typov int, float, bool, str a tuple sú immutable teda nemenné, s nimi tento problém nenastáva nami definované nové typy (triedy) sú vo všeobecnosti mutable - ak by sme chceli vytvoriť novú immutable triedu, treba ju definovať veľmi špeciálnym spôsobom

```
mato=fero
vypis(mato)
Volam sa  Ferdinand Pocitacovy
mato.meno='Martin'
vypis(fero)
Volam sa  Martin Pocitacovy
```

Pozor: treba dávať naozaj veľký pozor na priradenie mutable objektov!

Funkcie

Už sme definovali funkciu vypis(), ktorá vypisovala dva konkrétne atribúty parametra (objektu). Táto funkcia nemodifikovala žiaden atribút, ani žiadnu doteraz existujúcu premennú. Zapíšme funkciu urob(), ktorá dostane dva
znakové reťazce a vytvorí z nich nový objekt typy Student, pričom tieto dva reťazce budú obsahom dvoch atribútov meno a priezvisko:

```
def urob(m,p):
    novy=Student()
    novy.meno=m
    novy.priezvisko=p
    return novy

fero=urob('Ferdinand','Velky')
zuzka=urob('Zuzka','Hraskovie')
matej=urob('Matej','Maly')
vypis(fero)
Volam sa  Ferdinand Velky
```

```
vypis(zuzka)
Volam sa  Zuzka Hraskovie
vypis(matej)
Volam sa  Matej Maly
```

Ani funkcia urob() nemodifikuje žiaden svoj parameter ani iné premenné, len vytvára novú inštanciu a tú vracia ako výsledok funkcie.
Funkcie, ktoré majú túto vlastnosť (nič nemodifikujú, len vytvárajú niečo nové) nazývame ***pravé funkcie*** (po anglicky ***pure function***). Pravou funkciou bude aj funkcia kopia , ktorá na základe jedného objektu vyrobí nový, ktorý je jeho kópiou. Predpokladáme, že robíme kópiu inštancie Student , ktorá má atribúty meno a priezvisko:

```
def kopia(iny):
    novy=Student()
    novy.meno=iny.meno
    novy.priezvisko=iny.priezvisko
    return novy

zuzka=urob('Zuzka','Hraskovie')
vypis(zuzka)
Volam sa  Zuzka Hraskovie
evka=kopia(zuzka)
evka.meno='Eva'
vypis(evka)
Volam sa  Eva Hraskovie
vypis(zuzka)
Volam sa  Zuzka Hraskovie
```

Obe inštancie sú teraz dva rôzne kontajnery, teda obe majú svoje vlastné súkromné premenné meno a priezvisko.
Okrem pravých funkcií existujú tzv. ***modifikátory*** (po anglicky ***modifier***). Je to funkcia, ktorá niečo zmení, najčastejšie atribút nejakého objektu. Funkcia nastav_hoby() nastaví danému objektu atribút hoby a vypíše o tom text:

```
def nastav_hobby(st,text):
    st.hoby=text
    print(st.meno,st.priezvisko,'ma  hoby',st.hoby)

nastav_hobby(fero,'gitara')
Ferdinand Pocitacovy ma  hoby gitara
```

Oba objekty fero aj evka majú teraz už 3 atribúty, pričom mato a zuzka majú len po dvoch.
Keďže vlastnosť funkcie ***modifikátor*** je pre všetky mutable objekty veľmi dôležitá, pri písaní nových funkcií si vždy musíme uvedomiť, či je to modifikátor alebo pravá funkcia a často túto informáciu zapisujeme aj do dokumentácie.

Metódy

Všetky doteraz vytvárané funkcie dostávali ako jeden z parametrov objekt typu Student alebo takýto objekt vracali ako výsledok funkcie. Lenže v objektovom programovaní platí:

• objekt je kontajner údajov, ktoré sú vlastne súkromnými premennými objektu (atribúty)

• trieda je kontajner funkcií, ktoré vedia pracovať s objektmi (aj týmto funkciám niekedy hovoríme atribúty)

Takže funkcie nemusíme vytvárať tak ako doteraz globálne v hlavnom mennom priestore (tzv. __main__), ale priamo ich môžeme definovať v triede. Pripomeňme si, ako vyzerá definícia triedy:

```
class Student:
        pass
```

Príkaz pass sme tu uviedli preto, lebo sme chceli vytvoriť prázdne telo triedy (podobne ako pre def ale aj for a if). Namiesto pass ale môžeme zadefinovať funkcie, ktoré sa stanú súkromné pre túto triedu. Takýmto funkciám hovoríme metóda. Platí tu ale jedno veľmi dôležité pravidlo: prvý parameter metódy musí byť premenná, v ktorej metóda dostane inštanciu tejto triedy a s ňou sa bude ďalej pracovať. Zapíšme funkcie vypis() a nastav_hoby() ako metódy:

Vykonané zmeny:

• obe funkcie sú vnorené do definície triedy a preto sú odsunuté vpravo

• obom funkciám sme zmenili prvý parameter st na self - toto sme robiť nemuseli, ale je to dohoda medzi pythonistami, že prvý parameter metódy sa bude vždy volať self bez ohľadu pre akú triedu túto metódu definujeme (obe funkcie by fungovali korektne aj bez premenovania tohto parametra)

Keďže vypis() už teraz nie je globálna funkcia ale metóda, nemôžeme ju volať tak ako doteraz vypis(fero).

a.) k menu uvedieme aj meno kontajnera (meno triedy), kde sa táto funkcia nachádza, teda Student.vypis(fero):

b.) použiť trochu pozmenený výpis pričom sa vynecháva meno triedy teda priamo len objekty bez mena kontajnera

```
fero=urob('Ferdinand','Pocitacovy')
zuzka=urob('Zuzka','Hraskovie')
Student.vypis(fero)
Volam sa  Ferdinand Pocitacovy
Student.vypis(zuzka)
Volam sa  Zuzka Hraskovie
fero.vypis()
Volam sa  Ferdinand Pocitacovy
zuzka.vypis()
Volam sa  Zuzka Hraskovie
```

Magické metódy

Okrem tohto štandardného mechanizmu volania metód, existuje ešte niekoľko špeciálnych metód, pre ktoré má Python aj iné využitie. Pre tieto špeciálne (tzv. magické) metódy má Python aj špeciálne pravidlá. My sa s niektorými z týchto magických metód budeme zoznamovať priebežne na rôznych prednáškach, podľa toho, ako ich budeme potrebovať. Magické metódy majú definíciu úplne rovnakú ako bežné metódy. Python ich rozpozná podľa ich mena:

ich meno začína aj končí dvojicou podčiarkovníkov. Pre Python je tento znak bežná súčasť identifikátorov, ale využíva ich aj na tento špeciálny účel. Ako prvé sa zoznámime s magickou metódou ***__repr__()*** a ***__init__()***.

Python má štandardnú funkciu ***repr()***, ktorá z ľubovoľnej hodnoty vyrobí reťazec. Túto funkciu zavolá Python napr. vždy vtedy, keď potrebuje v príkazovom režime vypísať nejakú hodnotu. Napr.

```
7 * 11 * 13
1001
>>> ['a'] * 3
['a', 'a', 'a']
```

Takže repr() z tých typov, ktoré Python už pozná, vyrobí reťazec, ktorý sa dá vypísať. Tie typy, ktoré Python nevie, ako ich treba vypísať, vypíše informáciu, čo je to za objekt a kde sa v pamäti nachádza (napr. aj pre inštancie triedy Student). Magická metóda __repr__() Pythonu vysvetlí, ako má vypísať samotnú inštanciu: keď zavoláme repr(fero) (alebo to zavolá Python v príkazovom režime), Python sa pozrie do príslušného typu, či sa tam nachádza metóda __repr__(). ak áno tak ju použije, inak použije náhradný reťazec:

```
repr(fero)
'<__main__.Student object at 0x102f15d68>'
```

Takže zadefinujme vlastnú metódu:

```
class Student:
    def vypis(self):
        print('Volam sa ',self.meno, self.priezvisko)

    def nastav_hoby(self, text):
        self.hoby=text
        print(self.meno,self.priezvisko, 'ma hoby',self.hoby)

    def __repr__(self):
        return 'Student: {} {}'.format(self.meno, self.priezvisko)

fero=urob('Ferdinand','Kratky')
 fero
Student: Ferdinand Kratky
print(fero)
Student: Ferdinand Fyzik
```

V oboch prípadoch sa zavolala naša magická metóda __repr__().

Metóda ***__init__()*** je magická metóda, ktorá slúži na inicializovanie atribútov daného objektu. Má tvar: def __init__(self, parametre):

Metóda môže mať (ale nemusí) ďalšie parametre za self. Metóda nič nevracia, ale najčastejšie obsahuje len niekoľko priradení.
Túto metódu (ak existuje) Python zavolá, v tom momente, keď sa vytvára nová inštancia.
Napr. keď zapíšeme instancia = trieda(parametre), tak sa postupne:

1. vytvorí sa nový objekt typu trieda - zatiaľ je to prázdny kontajner, teda vytvorí sa referencia objekt

2. ak existuje metóda __init__(), zavolá ju s príslušnými parametrami:
trieda.__init__(objekt,parametre)
3. do premennej instancia priradí práve vytvorený objekt
Hovoríme, že metóda __init__() ***inicializuje objekt*** (niekedy sa hovorí aj, že konštruuje, resp. že je to ***konštruktor***). Najčastejšie sa v tejto metóde priradzujú hodnoty do atribútov, napr.

```
class Student:
    def __init__(self,meno,priezvisko,hoby=''):
        self.meno=meno
        self.priezvisko=priezvisko
        self.hoby=hoby

    def vypis(self):
        print('Volam sa ',self.meno, self.priezvisko)

    def nastav_hoby(self, text):
        self.hoby=text
        print(self.meno,self.priezvisko, 'ma hoby',self.hoby)

    def __repr__(self):
        return 'Student: {}{}'.format(self.meno, self.priezvisko)

fero=Student('Ferdinand','Kratky')
fero.nastav_hoby('gitarista')
Ferdinand Kratky ma hoby gitarista
```

Zhrnutie triedy a inštancie:

a.) triedy sú kontajnery atribútov
b.) väčšinou sú to funkcie t.j. metódy
c.) niektoré metódy sú magické, a tie majú špeciálne využitie
d.) triedy sú vzory na vytváranie inštancií
e.) inštancie sú kontajnery atribútov
f.) väčšinou sú to súkromné premenné inštancií
g.) ak nejaký atribút nie je definovaný v inštancii, tak Python zabezpečí, že sa použije atribút z triedy (inštancia automaticky vidí triedne atribúty)

Základné vlastnosti OOP

je v programovacom jazyku charakterizované týmito tromi vlastnosťami:
• **zapuzdrenie (enkapsulácia, encapsulation)** označuje:
• v objekte sa nachádzajú premenné aj metódy, ktoré s týmito premennými pracujú (hovoríme, že údaje a funkcie sú zapuzdrené v jednom celku)
• vďaka metódam môžeme premenné v objekte ukryť, takže zvonku sa pracuje s údajmi len pomocou týchto metód
• pripomeňme si triedu Zlomok z cvičení: v atribútoch citatel a menovatel sa vždy nachádzajú tieto hodnoty v základnom tvare, pričom menovateľ by mal byť vždy kladné (nenulové) číslo, predpokladáme, že s týmito atribútmi nepracujeme priamo, ale len pomocou metód __init__(), sucet(), sucin(), ...
• z tohto dôvodu, by sme niekedy potrebovali dáta skryť a doplniť funkcie, tzv. getter a setter pre tie atribúty, ktoré chceme nejako ochrániť, neskôr uvidíme ďalší pojem, ktorý s týmto súvisí, tzv. vlastnosť (property)
• **dedičnosť (inheritance)**
• novú triedu nevytvárame z nuly, ale využijeme už existujúcu triedu
• **polymorfizmus -** viactvarosť v použití metód a funkcií