Функција property()

Функција property() служи да врати атрибуте особина из датих гетера, сетера и дилетера.
Синтакса функције: property(fget = None, fset = None, fdel = None, doc = None)
Параметри функције су опциони:

- fget – функција за налажење вредности атрибута, по дифолту је None
- fset – функција за подешавање вредности атрибута, по дифолту је None
- fdel – функција за брисање вредности атрибута, по дифолту је None
- doc – стринг који садржи докстринг за атрибут, по дифолту је None

Методе за подешавање вредности

Неки ОО програмски језици подржавају приватне атрибуте објеката којима се не може приступити директно споља, па програмери пишу методе за добијање и постављање вредности како би се читале и писале вредности таквих атрибута.
У Пајтону су сви атрибути и методе јавни па за таквим методама нема потребе, али је донекле подржано и постојање таквих метода.
Нека је дефинисана класа Zivotinja и атрибут sakriveno_ime којима се не жели омогућити приступ директно.
Зато се дефинишу две методе: метода за налажење (get_ime()) и метода за подешавање (set_ime()).

```
class Zivotinja():
    def __init__(self, uneti_ime):
        self.sakriveno_ime = uneti_ime

    def get_ime(self):
        print('aktivan geter')
        return self.sakriveno_ime

    def set_ime(self, uneti_ime):
        print('aktivan seter')
        self.sakriveno_ime = uneti_ime

    ime = property(get_ime, set_ime)
```

Методе се понашају као нормални претраживачи (гетери) и подешавачи (сетери) све до последње линије кода.
Последња линија кода дефинише ове две методе као особине атрибута ime класе Zivotinja.
Овде се користи уграђена функција property() са два аргумента који су постављени као метода претраживач и метода подешавач.
Сада када се указује на ime било којег Zivotinja објекта, заправо се прво позива метода get_ime да би вратила то име.

```
pas = Zivotinja("Reks")
print(pas.ime)
```

Даје:
aktivan geter
Reks
И даље се може користити гетер метода директно:

```
pas.get_ime() # aktivan geter
```

Када се атрибуту ime додељује нека вредност, позива се метода сетера:

```
pas.ime = "Sava"
print(pas.ime)
```

Даје:
aktivan seter
aktivan geter
Sava
И даље се може користити сетер метода директно:

```
pas.set_ime("Miki") #aktivan seter
```

Додавање понашања

Један од циљева у ООП је раздвајање понашања од података, иако у Пајтону такве разлике имају тенденцију да нестану.

У већини ООП језика тражи се да се никада не приступа атрибутима директно (пошто су промењиве приватне) па се преко метода гетера и сетера приступа тим промењивима.

У Пајтону је другачије:

```
class Color:
    def __init__(self, rgb_value, name):
        self.rgb_value = rgb_value
        self.name = name

c = Color("#ff0000", "bright red")
print(c.name)
c.name = "red"
print(c.name)
```

Разлог због којег се инстистира на синтакси базираној на методама је то што се касније појављује потреба за додатком кода када се нека вредност поставља или враћа.

Пример: промена методе postavi_ime()

```
def postavi_ime(self, ime):
    if not ime:
        raise Exception("Ime nije dobro")
    self._ime = ime
```

Пајтон има службену реч property којом се чини да методе изгледају као атрибути.

Зато је могуће писати код којим се директно приступа члановима класе, па ако је потребно променити имплементацију, то може да се уради без мењања интерфејса.

```
class Boja:
    def __init__(self, rgb_vrednost, ime):
        self.rgb_vrednost = rgb_vrednost
        self._ime = ime

    def _set_ime(self, ime):
        if not ime:
            raise Exception("Ime nije dobro")
        self._ime = ime

    def _get_ime(self):
        return self._ime

    ime = property(_get_ime, _set_ime)
```

Прво се мења атрибут ime у полуприватни атрибут _ime, а затим се додају две полуприватне методе којима се узима и поставља та промењива.

На крају кода је декларација property, која креира нов атрибут на класи Boja, која мења претходни атрибут ime.

Овиме се поставља атрибут као property, којиме се позивају управо креиране две методе када год се особености приступи или промени.

Нова верзија класе Boja се може користити на исти начин као и раније осим што се сада извршава валидација при постављању атрибута ime.

```
c = Boja("#0000ff", "jarko crvena")
print(c.ime)
c.name = "crvena"
print(c.ime)
c.ime = ""
```

Даје:

jarko crvena

jarko crvena
Traceback (most recent call last):
  raise Exception("Ime nije dobro")
Exception: Ime nije dobro
Значи, прво је написан код за приступ атрибуту име, а затим је промењен да би се користио објекат property, претходни код и даље ради осим ако се пошаље празна вредност property, што је понашање које је и потребно.

Израда лабораторијских вежби:
Задатак 052: Креирати подкласу Пингвин надкласе Птица. Модификовати понашање класе Птица преправљањем методе инстанце ко_је_ово класе Пингвин. Користити функцију супер за позив иницијализатора надкласе Птице. Креирати нову методу инстанце класе Пингвин, трчање. Проверити наслеђивање у оваквој форми.

```
class Ptica:
    def __init__(self):
        print('Kiki je spreman.')

    def ko_je_ovo(self):
        print('Ovo je Kiki.')

    def plivanje(self):
        print('Kiki pliva brze.')

class Pingvin(Ptica):
    def __init__(self):
        super().__init__()
        print('Pingvin je spreman.')

    def ko_je_ovo(self):
        print('Ovo je pingvin.')

    def trcanje(self):
        print('Kiki trci brze.')

kiki = Pingvin()
kiki.ko_je_ovo()
kiki.plivanje()
kiki.trcanje()
```

Даје:
Kiki je spreman.
Pingvin je spreman.
Ovo je pingvin.
Kiki pliva brze.
Kiki trci brze.

Задатак 053: Креирати класу Запослени са промењивом класе повећање_плате, иницијализатором са атрибутима име, презиме и плата. Креирати подкласу Програмер са промењивом класе повећање_плате, иницијализатором који се ослања на атрибуте надкласе и са сопственим атрибутом програмски_језик.

```
class Zaposleni:
    povecanje_plate = 1.04

    def __init__(self, ime, prezime, plata):
        self.ime = ime
        self.prezime = prezime
        self.plata = plata
        self.emejl = ime + "." + prezime + "@mejl.com"

    def ime_i_prezime(self):
```

```
        return '{} {}'.format(self.ime, self.prezime)

    def primeni_povecanje(self):
        self.plata = int(self.plata * self.povecanje_plate)

class Programer(Zaposleni):
    povecanje_plate = 1.10

    def __init__(self, ime, prezime, plata, prog_jezik):
        super().__init__(ime, prezime, plata)
        self.prog_jezik = prog_jezik

def main():
    programer_1 = Programer("Ana", "Anica", 80000, "Pajton")
    programer_2 = Programer("Miki", "Milic", 75000, "Java")
    print(programer_1.emejl, programer_1.prog_jezik)
    print(programer_2.emejl, programer_2.prog_jezik)

main()
```

Задаци за самосталан рад:

48. У примеру задатка 053 креирати нову подкласу Менаџер класе Запослени. Менаџер треба да има листу запослених којима треба да управља. Зато постоји посебни атрибут радници чија је дифолт вредност None. Ако је то тачно, ствара се празна листа ако то није тачно, додељује јој се вредност радници. Омогућити да се може уносити у листу нови радник и избацити из листе неки радник. Креирати методу којим се исписује целокупна листа радника којима управља одређени менаџер.

49. Креирати надкласе ЧланТима и Радник и подкласу ВођаТима. Класа ЧланТимао има два атрибута име и ид. Класа БорбеноСредство има атрибуте плата и назив_посла. Проверити функционисање вишеструког наслеђивања у подкласи ако се у њој креира и промењива класе године_рада.