Основе наслеђивања

Технички, свака креирана класа користи наслеђивање, пошто све класе у Пајтону су подкласе посебне класе која се назива објекат (object).

У смислу података и понашања (мада пружа понашања која су са двоструким ундерскор линијама што значи да су искључиво намењена интерној употреби) пружа ваома мало, али омогућава Пајтону да третира све објекте на исти начин.

Ако се експлицитно не наведе наслеђивање из друге класе, наше класе ће аутоматски наследити из класе објекат.

Ипак, може се навести да наша класа потиче из класе object коришћењем следеће синтаксе:

```
class MojaPodklasa(object):
    pass
```

У Пајтону3, све класе аутоматски наслеђују из класе object ако се не напише експлицитно друга надкласа.

Надкласа (superclass, или родитељска класа) је класа из које се наслеђује.

Подкласа (subclass) је класа која наслеђује из надкласе.

У датом примеру, надкласа је object, а MojaPodklasa је подкласа.

Каже се да је подкласа изведена (derived) из њене родитељске класе или да подкласа проширује (extends) родитељску класу.

У синтакси је довољно само навести име подкласе а у загради име надкласе и тако извести наслеђивање.

Давање нових функција постојећој класи

У пракси је примена наслеђивања најочигледнија приликом додавања нове функција постојећој класи.

Пример: менаџер контаката води рачуна о именима и е-мејл адресама неких особа; класа контакт служи за одржавање листе свих контаката у промењивој класе и за иницијализацију имена и адреса за појединачни контакт:

```
class Kontakti:
    svi_kontakti = []

    def __init__(self, ime, email):
        self.ime = ime
        self.email = email
        Kontakti.svi_kontakti.append(self)
```

Овде се први пут спомињу промењиве класе.

Листа svi_kontakti, пошто је део дефиниције класе, се дели са свим инстанцама ове класе.

То значи да постоји само једна `Kontakti.svi_kontakti` листа (**промењива класе**), којој се може прићи са `Kontakti.svi_kontakti`.

Такође, може јој се прићи као self.svi_kontakti на било којем објекту који је инстанциран из класе Kontakti.

Ово је једноставна класа која омогућава да пратимо податке о сваком од контаката.

Али, шта ако су неки од контаката такође и добављачи који су нам потребни за достављање поруџбине?

Може се додати један метод porudzbina у класу Kontakti, али би то омогућило људима да случајно наруче ствари од контаката који су муштерије или пријатељи породице.

Уместо тога, креираћемо нову класу Dobavljac која се понаша исто као и класа Kontakti, али са додатним методом porudzbina:

```
class Dobavljac(Kontakti):
    def porudzbina(self, porudzbina):
        print("Da je ovo realna situacija, poslali bi smo "
                "{} porudzbinu prema {}".format(porudzbina, self.ime))
```

Сада, ако се тестира ова класа, види се да сви контакти, укључујући и добављаче, прихватају име и емејл у њиховом __init__, али само добављачи имају функционалну методу porudzbina.

Сада, класа Dobavljac може урадити све што и класа Kontakti може (укључујући додавање самог себе на листу svi_kontakti) као и све остале посебне ствари које се раде као добављач.

```
c = Kontakti("Neko", "neko@primer.net")
s = Dobavljac("Dobavljac", "dobavljac@primer.net")
print(c.ime, c.email, s.ime, s.email)
c.svi_kontakti
s.porudzbina("Hocu kljesta")
```

Даје:
Neko neko@primer.net Dobavljac dobavljac@primer.net
Да је ово realna situacija, poslali bi smo ‘Hocu kljesta’ porudzbinu prema ‘Dobavljac’

Види се да с је инстанца класе Kontakti а s је инстанца класе Dobavljac.
Свака од ових инстанци има два засебна аргумента који одговарају параметрима ime, email.
Позивањем листе svi_kontakti, добија се листа свих елемената пошто су они приликом позивања функције додавани у листу преко методе append.
Када би се позвала листа преко инстанце s (s.svi_kontakti), добија се исти резултат као у примеру (c.svi_kontakti) пошто се позив у оба случаја односи на исту листу са истим елементима, а класе се међусобно наслеђују.
Позивањем методе porudzbina преко s инстанце, добија се тело методе porudzbina које исписује поруку на екрану са повезаним параметром s.ime.
Ако би написали следећу линију кода: `c.porudzbina("Hocu kljesta")`
Добија се: AttributeError: 'Kontakti' object has no attribute 'porudzbina'
Пошто је с инстанца класе Kontakti, која нема методу porudzbina, добија се AttributeError грешка.
То се дешава иако је porudzbina метода дефинисана у класи Dobavljac, која је наследила особености класе Kontakti.

Промена постојећих класа
Нека и даље постоји класа Kontakti са подацима о имену и емејлу особе; сада је циљ додати и телефон исте особе као податак.
Један начин је поставити атрибут telefon на контакт после његовог формирања.
Али ако је циљ да telefon постане трећа промењива при иницијализацији, мораће да се премости (override) __init__ метода.
Премошћавање значи мењање или замену методе надкласе са новом методом истог имена у подкласи.
За ово се не користи нека посебна синтакса већ ново креирани метод подкласе се аутоматски позива уместо метода надкласе.
Пример:

```
class Kontakti:
    svi_kontakti = []

    def __init__(self, ime, email):
        self.ime = ime
        self.email = email
        self.svi_kontakti.append(self)

class Prijatelj(Kontakti):
    def __init__(self, ime, email, telefon):
        self.ime = ime
        self.email = email
        self.telefon = telefon
```

Сваки метод се може премостити не само __init__.
Међутим, класе Kontakti и Prijatelj имају дуплирани код којим се постављају особености ime и email; ово може учинити одржавање кода тешким пошто се онда део кода везан за особености мора апдејтовати на више места.
Посебно, класа Prijatelj нема додавања самог себе на svi_kontakti листу која је креирана на класи Kontakti.
Оно што је неопходно јесте извршење оригиналне __init__ методе на класи Kontakti.
За ово се користи super функција; она враћа објекат као инстанцу родитељске класе, чиме нам омогућава позивање родитељског метода директно:

```
class Prijatelj(Kontakti):
    def __init__(self, ime, email, telefon):
        super().__init__(ime, email)
        self.telefon = telefon
```

У примеру се прво добија инстанца родитељског објекта коришћењем функције super() и позива метода __init__ на том објекту, чиме се придодају очекивани аргументи.
Затим се извршава сопствена иницијализација, тј постављање атрибута telefon.
Позив функције super() се може убацити унутар било које методе, не само __init__, а то значи да се све методе могу модификовати помоћу премошћавања и функције super().
Позив функције super() се може извршити било где у методи (не мора бити прва линија у методи), па тако се може манипулисати улазним параметрима пре њиховог прослеђивања надкласи.

Израда лабораторијских вежби:
Задатак 046: Класа Контакти има иницијализатор са атрибутима инстанце име и емејл. Подкласа Добављач нема своје методе и атрибуте. Доказати да је подкласа наследила особине надкласе.

```
class Kontakti:
    def __init__(self, ime, emejl):
        self.ime = ime
        self.emejl = emejl

class Dobavljac(Kontakti):
    pass

prvi = Kontakti("Neko", "neko@primer.net")
drugi = Dobavljac("Dobavljac", "dobavljac@primer.net")
print(prvi.ime, prvi.emejl, drugi.ime, drugi.emejl)
```

Даје:
Neko neko@primer.net Dobavljac dobavljac@primer.net

Задатак 047: Класа Контакти има иницијализатор са атрибутима инстанце име и емејл. Креирати подкласу Добављач која има методу инстанце поруџбина. Доказати да принцип наслеђивања не важи у обрнутом смеру, тј да надкласа не наслеђује специфичну методу подкласе.

```
class Kontakti:
    def __init__(self, ime, emejl):
        self.ime = ime
        self.emejl = emejl

class Dobavljac(Kontakti):
    def porudzbina(self, porudzbina):
        print("Da je ovo realna situacija, poslali bi smo "
              "{} porudzbinu prema {}".format(porudzbina, self.ime))

prvi = Kontakti("Neko", "neko@primer.net")
drugi = Dobavljac("Dobavljac", "dobavljac@primer.net")
print(prvi.ime, prvi.emejl, drugi.ime, drugi.emejl)
drugi.porudzbina("Hocu kljesta")
prvi.porudzbina("Hocu kljesta")
```

Даје:
Neko neko@primer.net Dobavljac dobavljac@primer.net
Da je ovo realna situacija, poslali bi smo Hocu kljesta porudzbinu prema Dobavljac
Traceback (most recent call last):
File "C:\Users\nera\source\repos\PythonApplication6\PythonApplication6.py", line 19, in <module>
c.porudzbina("Hocu kljesta")
AttributeError: 'Kontakti' object has no attribute 'porudzbina'

Задатак 048: Класа Контакти има атрибут класе листу сви_контакти и иницијализатор са атрибутима инстанце име и емејл. Иницијализатор придодаје свки ново унети пар име–емејл у листу сви_контакти. Написати програм са функцијама за унос имена и емејла и приказ садржаја листе сви_контакти.

```
class Kontakti:
```

```
    svi_kontakti = []

    def __init__(self, ime, emejl):
        self.ime = ime
        self.emejl = emejl
        self.svi_kontakti.append(self)

def main():
    unos_podataka()
    prikaz_podataka()
    unos_podataka()
    prikaz_podataka()

def unos_podataka():
    i = input("Uneti ime:")
    e = input("Uneti emejl:")
    Kontakti(i, e)

def prikaz_podataka():
    for a in Kontakti.svi_kontakti:
        print(a.ime, a.emejl)

main()
```

Daje:
Uneti ime:ja
Uneti emejl:ja@proba.net
ja ja@proba.net
Uneti ime:on
Uneti emejl:on@proba.net
ja ja@proba.net
on on@proba.net

Задаци за самосталан рад:

43. Креирати надкласу Пас са иницијализатором са атрибутима име и године. Креирати подкласу Булдог класе Пас. Инстанцирати објекат из подкласе и проверити принцип наслеђивања.

44. У надкласу Пас, из задатака 31, додати две методе инстанце: опис (користи име и године пса) и лавеж (користи атрибут звучи). Креирати подкласу Булдог са методом инстанце трчи и атрибутом брзина, која враћа опис трчања и име пса. Инстанцирати објекат из подкласе.

45. Написати код са надкласом Контакти из задатака 003 и подкласом Добављач из задатка 002. Користити објекат инстанциран класом Добављач за унос вредности атрибута име и емејл који ће се уносити у атрибут класе сви_контакти.