Python Object Inheritance

S pomočjo dedovanja (inheritance) lahko iz že obstoječih razredov ustvarimo nove, bolj specifične razrede.

Tako novo ustvarjeni razredi so imenovani "child classes" in so izpeljani iz "parent classes".

Child-classes podedujejo vse attribute in metode parent-class-a, katere lahko tudi prepišemo (override) ali pa dodamo nove, bolj specifične attribute in metode.

In [30]:

```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        return (f'{self.ime} je star {self.starost}')

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append(hrana)


fido = Pes("Fido", 9)
print(fido.opis())
```

```
Fido je 9 let star in je pes. Najraje je ['svinjina'].
```

In [31]:

```
# Sedaj ustvarimo child class, ki bo dedoval iz class Pes

class Bulldog(Pes):
    pass

spencer = Bulldog("Spencer", 15) # ustvarimo novo instanco class Bulldog, ki deduje
print(type(spencer)) # vidimo, da je instanca class Bulldog
print(spencer)
print(spencer.opis()) # vidimo, da smo dedovali metodo opis() iz class Pes
# če deluje metoda opis pol mamo tud .ime in .starost spremenljivko
```

```
<class '__main__.Bulldog'>
<__main__.Bulldog object at 0x00000123A232DC18>
Spencer je star 15
```

Extending child class

Child class lahko tudi naprej razvijemo z novimi metodami.

In [32]:

```
class Bulldog(Pes):
    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes cl
        return(f'Woof, woof.')
```

In [34]:

```
spencer = Bulldog("Spencer", 15)
print(spencer.bark())

fido = Pes("Fido", 9)
print(fido.bark())
```

```
Woof, woof.

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call
 last)
<ipython-input-34-27c093936b16> in <module>
      3
      4 fido = Pes("Fido", 9)
----> 5 print(fido.bark())

AttributeError: 'Pes' object has no attribute 'bark'
```

Overriding methods and attributes

Metode in attribute parentclass-a lahko tudi prepišemo.

In [35]:

```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        return (f'{self.ime} je star {self.starost}')

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append("teletina")


fido = Pes("Fido", 9)
print(fido.opis())
```

```
Bulldog
Woof, woof.
Spencer je star 15 in je Bulldog.

pes
Fido je star 9
```

In [ ]:

```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def opis(self):
        return f"{self.ime} je star {self.starost} in je {self.vrsta}"

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes cl
        return(f'Woof, woof.')

spencer = Bulldog("Spencer", 15)
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bulldog
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog class
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class Bu

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

In [ ]:

```

```

Uporaba metod parent class-a

Sedaj želimo dodati najljubši hrano vsakega Bulldoga.

In [60]:

```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        return (f'{self.ime} je star {self.starost}')

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append("teletina")


fido = Pes("Fido", 9)
print(fido.opis())
```

```
Fido je star 9
```

TO lahko dosežemo tako, da prepišemo __init__ metodo Bulldog class-a:

In [61]:

```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def __init__(self, ime, starost, najljubsa_hrana):
        self.ime = ime
        self.starost = starost
        self.najljubsa_hrana = najljubsa_hrana

    def opis(self):
        return (f'{self.ime} je star {self.starost} in je {self.vrsta}. Najraje je

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes cl
        return(f'Woof, woof.')

spencer = Bulldog("Spencer", 15, "čevapi")
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bulldog
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog class
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class Bu

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

```
Bulldog
Woof, woof.
Spencer je star 15 in je Bulldog. Najraje je čevapi

pes
Fido je star 9
```

Vendar tako ponavljamo določeno kodo:

```
    self.ime = ime
    self.starost = starost
```

Namesto tega lahko uporabimo *super()* funkcijo s katero dostopamo do metod razreda iz katerega smo dedovali.

In [73]:

```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def __init__(self, ime, starost, najljubsa_hrana):
        super().__init__(ime, starost)
        self.najljubsa_hrana = najljubsa_hrana

    def opis(self):
        return (f'{self.ime} je star {self.starost} in je {self.vrsta}. Najraje je

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes cl
        return(f'Woof, woof.')

spencer = Bulldog("Spencer", 15, "čevapi")
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bulldog
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog class
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class Bu

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

```
Bulldog
Woof, woof.
Spencer je star 15 in je Bulldog. Najraje je čevapi

pes
Fido je star 9
```

In [ ]:

```
```

Naloga:

Ustvarite razred Vozilo. Vsaka instanca naj ima svojo specifično hitrost in kilometrino in koliko goriva je bilo porabljenega do sedaj.

Razred Vozilo naj ima funkcija **poraba()**, ki vrne koliko je povprečna poraba tega vozila.

Dodajte **class variable** razredu Vozilo. Spremenljivki naj bo ime **st_gum** in njena vrednost naj bo **4**. Dodajte metodo **opis()**, ki naj izpiše opis vozila.

*Ustvarite podrazreda **Avto** in **Motor**. Razreda naj dedujete od razreda Vozila. Motor razred naj prepiše spremenljivko **st_gum** v **2**. Vsak razred naj pravilno shrani ime vozila, ko ustvarimo novo instanco.*

Primeri:

Input:

```
avto = Avto(300, 80, 500)
avto.opis()
```

Output:

```
Max hitrost avto: 300. Prevozenih je 80 km. Poraba vozila je 6.25 l/km. Voz
ilo ima 4 gum.
```

Input:

```
motor = Motor(90, 220, 520)
motor.opis()
```

Output:

```
Max hitrost motor: 90. Prevozenih je 220 km. Poraba vozila je 2.36 l/km. Vo
zilo ima 2 gum.
```

In [85]:

```
class Vozilo:
    st_gum = 4

    def __init__(self, vozilo, hitrost, kilometrina, gorivo):
        self.vozilo = vozilo
        self.hitrost = hitrost
        self.kilometrina = kilometrina
        self.gorivo = gorivo

    def poraba(self):
        return self.gorivo / self.kilometrina

    def opis(self):
        print(f"Max hitrost {self.vozilo}: {self.hitrost}. Prevozenih je {self.kilo


class Avto(Vozilo):
    def __init__(self, hitrost, kilometrina, gorivo):
        super().__init__("avto", hitrost, kilometrina, gorivo)


class Motor(Vozilo):
    st_gum = 2

    def __init__(self, hitrost, kilometrina, gorivo):
        super().__init__("motor", hitrost, kilometrina, gorivo)


avto = Avto(300, 80, 500)
avto.opis()


motor = Motor(90, 220, 520)
motor.opis()
```

```
Max hitrost avto: 300. Prevozenih je 80 km. Poraba vozila je 6.25 l/k
m. Vozilo ima 4 gum.
Max hitrost motor: 90. Prevozenih je 220 km. Poraba vozila je 2.36 l/k
m. Vozilo ima 2 gum.
```

In [ ]:

```

```

Multiple inheritance

In [37]:

```
# Multiple inheritance
class SuperA:
    VarA = 10
    def funa(self):
        return 11

class SuperB:
    VarB = 20
    def funb(self):
        return 21

class Sub(SuperA,SuperB):
    pass

object_ = Sub() # podeduje metode in atribute razreda A in razreda B

print(object_.VarA, object_.funa())
print(object_.VarB, object_.funb())
# kle ni problem, ker se nobena stvar ne prekriva (ne instance, ne metode)
```

```
10 11
20 21
```

In [40]:

```
# Left to right
class A:
    def fun(self):
        print('a')

class B:
    def fun(self):
        print('b')


class C(B,A):
    pass
object_ = C()
object_.fun() # prvo dedujemo iz najbl desnega, pol proti levi in prepisujemo stvar
```

```
b
```

In [41]:

```
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
    def fun(self):
        return 101

class Level2(Level1):
    pass

object_ = Level2() # razred Level0 je parent. Level1 deduje iz Level0 in "overrida"
print(object_.Var, object_.fun())
```

```
100 101
```

isinstance() function

s pomočjo funkcijo `python isinstance()` lahko preverimo, če je naša instanca res instanca določenega razreda oziroma razreda, ki od njega deduje.

In [1]:

```
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
    def fun(self):
        return 101

class Level2(Level1):
    pass

l0 = Level0()
l1 = Level1()
l2 = Level2()

print(isinstance(l2, Level2)) #ali je instanca level2 del razreda Level2
print(isinstance(l2, Level1))
print(isinstance(l2, Level0))
print()
```

```
True
True
True
```

In [ ]:

```
```

inspect.getmro(class_name)

S pomočjo te funkcije lahko izpiše strukturo dedovanja.

In [79]:

```
import inspect
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
    def fun(self):
        return 101

class Level2(Level1):
    pass

inspect.getmro(Level2)
```

Out[79]:

```
(__main__.Level2, __main__.Level1, __main__.Level0, object)
```

In [ ]:

```
```

Banka

Napišite program:

```
class Oseba():
    def __init__(self, ime, priimek):
        # za to instanco naj ustvari spremenljivki ime in priimek

    def __str__(self):
        # vrne naj string, znotraj katerega imamo ime in priimek


class Stranka(): # class naj deduje od razreda Oseba()
    def nastavi_stanje(self, stanje):
        # metoda naj ustvari spremenljivko samo za to instaco razreda. Vred
nost naj bo "stanje" oziroma default vrednost naj bo 0. Metoda naj nato vrn
e vrednost spremenljivke stanje

    def dvig(self, znesek):
        # Od stanja naj se odšteje znesek.
        # V kolikor ni dovolj denarja na računu naj se dvigne z banke celot
no stanje
        # Na koncu naj metoda vrne dvignjen znesek

    def polog(self, znesek):
        # metoda naj doda velikost zneska stanju
        # nato naj metoda vrne novo stanje


INPUT:
objekt = Stranka("Gregor", "Balkovec")
print(objekt)
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))

OUTPUT:
Gregor Balkovec
0.0
5000.0
2000
Dal ti bom samo 3000.0
3000.0
```

In [2]:

```
class Oseba():
    def __init__(self, ime, priimek):
        self.ime = ime
        self.priimek = priimek

    def __str__(self):
        return self.ime + " " + self.priimek


class Stranka(Oseba):

    def nastavi_stanje(self, stanje=0.0):
        self.stanje = stanje
        return self.stanje

    def dvig(self, znesek):
        dvig = 0
        if znesek > self.stanje:
            print("Dal ti bom samo", self.stanje)
            dvig = self.stanje
            self.stanje = 0
        else:
            self.stanje -= znesek
            dvig = znesek
        return dvig

    def polog(self, znesek):
        self.stanje += znesek
        return self.stanje


objekt = Stranka("Gregor", "Balkovec")
print(objekt)
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))
```

```
Gregor Balkovec
0.0
5000.0
2000
Dal ti bom samo 3000.0
3000.0
```

In [ ]:

```

```

In [ ]:

```

```

Ustvarite razred `Polica` .

- Vsaka instanca razreda naj ima:

- **knjige** -> list naslovov knjig, ki se nahajajo na polici
- **max_knjig** -> integer vrednost, ki pove koliko knjig, gre maximalno na polico
- Ko ustvarimo instanco razreda vanj posredujemo številko maximalnih knjig na polici.
- Ko ustvarimo instanco razreda vanj posredujemo lahko tudi list naslovov knjig, ki se že nahajajo na polici. Če takega seznama ne posredujemo naj ima polica prazen seznam.
- Razred naj ima metodo `kaj_je_na_polici` , ki naj vrne list naslovov knjig
- Razred naj ima metodo `dodaj_knjigo` , ki kot argument prejme string naslova knjige. To knjigo naj doda v list naslovov knjig, če s tem ne presežemo maximalno število knjig. Če bi presegli to število knjige ne dodamo.
- Razred naj ima metodo `uredi_knjige` , ki kot argument **ascending** prejme boolean vrednost, ki nam pove ali naj bodo knjige urejene (glede na prvo črko) v A->Z (vrednost True) oziroma Z->A (vrednost False). Če ta vrednost ni bila posredovana naj bo default vrstni red A->Z. Metoda naj uredi list naslovov knjig in tega nato vrne

```
polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet", "Kraut
ov Strojniški Priročnik", "SSKJ"])
print(polica.kaj_je_na_polici())
==> ['The Witcher', 'Dune', 'Harry Potter', 'Hamlet', 'Krautov Strojniški P
riročnik', 'SSKJ']

print(polica.uredi_knjige())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik', 'SSK
J', 'The Witcher']

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik', 'SSK
J', 'The Witcher', 'Romeo in Julija']

polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik', 'SSK
J', 'The Witcher', 'Romeo in Julija']

print(polica.uredi_knjige(False))
==> ['The Witcher', 'SSKJ', 'Romeo in Julija', 'Krautov Strojniški Priročni
k', 'Harry Potter', 'Hamlet', 'Dune']

polica.shrani_knjige()
```

In [86]:

```
class Polica:
    def __init__(self, max_knjig, knjige=list()):
        self.max_knjig = max_knjig
        self.knjige = knjige

    def kaj_je_na_polici(self):
        return self.knjige

    def dodaj_knjigo(self, knjiga):
        if len(self.knjige) < self.max_knjig:
            self.knjige.append(knjiga)

    def uredi_knjige(self, ascending = True):
        if ascending:
            self.knjige.sort()
        else:
            self.knjige.sort(reverse=True)
        return self.kaj_je_na_polici()



polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet", "Tintin", "SSK
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige())

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())

polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige(False))
```

```
['The Witcher', 'Dune', 'Harry Potter', 'Hamlet', 'Tintin', 'SSKJ']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin', 'R
omeo in Julija']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin', 'R
omeo in Julija']
['Tintin', 'The Witcher', 'SSKJ', 'Romeo in Julija', 'Harry Potter',
'Hamlet', 'Dune']
```

In [ ]: