Metoda lahko tudi vrne vrednost.

In [21]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        print(f"{self.ime} je star {self.starost}")

    def vrni_starost(self):
        return self.starost

fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(fido.vrni_starost())
print(rex.vrni_starost())

if fido.vrni_starost() > rex.vrni_starost():
    print("Fido je starejši")
else:
    print("Rex je starejši")
```

```
9
10
Rex je starejši
```

## 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.

Primeri:

```
Input:
avto = Vozilo(300, 80, 100)
print(avto.poraba())

Output:
Vozilo porabi 1.25l/km

Input:
kamion = Vozilo(90, 5500, 125000)
print(f"Vozilo porabi {kamion.poraba()}l/km")

Output:
Vozilo porabi 22.73l/km
```

In [9]:
```
class Vozilo:
    def __init__(self, hitrost, kilometrina, gorivo):
        self.hitrost = hitrost
        self.kilometrina = kilometrina
        self.gorivo = gorivo

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


avto = Vozilo(300, 80, 100)
print(f"Vozilo porabi {avto.poraba():.2f}l/km")

kamion = Vozilo(90, 5500, 125000)
print(f"Vozilo porabi {kamion.poraba():.2f}l/km")
```

```
Vozilo porabi 1.25l/km
Vozilo porabi 22.73l/km
```

In [ ]:
```

```

Razredi imajo lahko tudi skupne spremenljivke - spremenljivke, ki so enake vsaki instanci.

> Vsak pes ima 4 noge. Vsak pes ima rad svinjino.

Če želimo, da je spremenljivka enotna celotnemu razredu:

In [29]:
```
class Pes:
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"

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

Do spremenljivke lahko sedaj dostopamo preko razreda samege:

In [31]:
```
print(f"Psi najraje jejo {Pes.hrana}")
```

```
Psi najraje jejo ['svinjina']
```

Oziroma, spremenljivka je dostopna preko vsake instance.

In [32]:
```
fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')
```

```
Fido najraje je ['svinjina'].
Rex najraje je ['svinjina'].
```

Spremenljivko lahko tudi spremenimo in jo tako spremenimo tudi za vse instance razreda.

In [35]:
```
class Pes:
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"

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

fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')

Pes.hrana = ["teletina"]

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')
```

```
Fido najraje je ['svinjina'].
Rex najraje je ['svinjina'].
Fido najraje je ['teletina'].
Rex najraje je ['teletina'].
```

In [8]:
```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"

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

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta # to nrdi instance variable, ki overwrida Class variab

    def dodaj_hrano(self, hrana):
        self.hrana.append(hrana) # to modify-a variable. In ker je list mutable t

    #def add_to_set(self, el):
        #self.set_ = el
        #self.set_.add(el)


fido = Pes("Fido", 9)
rex = Pes("Rex", 10)
ace = Pes("Ace", 3)

print(f'{fido.ime} je {fido.starost} let star in je {fido.vrsta}. Najraje je {fid
print(f'{rex.ime} je {rex.starost} let star in je {rex.vrsta}. Najraje je {rex.hr
print(f'{ace.ime} je {ace.starost} let star in je {ace.vrsta}. Najraje je {ace.hr

print(30*"*")
Pes.vrsta = "kuščar" # tuki spremenimo variable celotnemu razredu. Vsi, ki so ins
fido.spremeni_vrsto("opica") # to naredu self.vrsta = opica za fido instanco razr

rex.dodaj_hrano("teletina")



print(f'{fido.ime} je {fido.starost} let star in je {fido.vrsta}. Najraje je {fid
print(f'{rex.ime} je {rex.starost} let star in je {rex.vrsta}. Najraje je {rex.hr
print(f'{ace.ime} je {ace.starost} let star in je {ace.vrsta}. Najraje je {ace.hr

#print(30*"*")
#print(Pes.vrsta)
#print(Pes.hrana)
#ace.add_to_set(66)
#print(f'{fido.set_} \n{rex.set_} \n{ace.set_}')
```

```
Fido je 9 let star in je pes. Najraje je ['svinjina'].
Rex je 10 let star in je pes. Najraje je ['svinjina'].
Ace je 3 let star in je pes. Najraje je ['svinjina'].
******************************
Fido je 9 let star in je opica. Najraje je ['svinjina', 'teletina'].
```

```
Rex je 10 let star in je kuščar. Najraje je ['svinjina', 'teletina'].
Ace je 3 let star in je kuščar. Najraje je ['svinjina', 'teletina'].
```

Treba bit pozoren, če za spremenljivko instance uporabimo enako ime kot za spremenljivko razreda, potem bo spremenljivka instance override class spremenljivko.

Če je spremenljivka mutable (list, itd..) in jo **modify-amo** (dodajamo elemente, odvzemamo, itd..) potem jo spremenimo za celoten razred.

> When we set an attribute on an instance which has the same name as a class attribute, we are overriding the class attribute with an instance attribute, which will take precedence over it. We should, however, be careful when a class attribute is of a mutable type – because if we modify it in-place, we will affect all objects of that class at the same time. Remember that all instances share the same class attributes:

# 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]:
```python
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 dedu
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
        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
        return(f'Woof, woof.')

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

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 j

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

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

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 j

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

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

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.
Vozilo 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.
Vozilo ima 2 gum.
```

In [13]:
```
class Vozilo:
    st_gum = 4
    vozilo = "vozilo"

    def __init__(self, hitrost, kilometrina, gorivo):
        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.ki


class Avto(Vozilo):
    vozilo = "avto"

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


class Motor(Vozilo):
    st_gum = 2
    vozilo = "motor"

    def __init__(self, hitrost, kilometrina, gorivo):
        super().__init__(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/km. Vozilo
ima 4 gum.
Max hitrost motor: 90. Prevozenih je 220 km. Poraba vozila je 2.36 l/km. Vozilo
ima 2 gum.
```

## 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 stv
```

```
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 "overric
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
```

## 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)
```