Objektno programiranje

Objektno programiranje je način kako mi združimo medseboj povezane podatke in funkcije, ki delujejo na te podatke.

Primer:

Želimo napisat program "pes".

- Za psa imamo nekja podatkov. Imamo njegovo ime in starost.
- Za psa imamo tudi funkcijo, ki nam ga opiše: f"{ime} je star {starost}"
- Vse to bi radi združili v en objekt

CLASS

Z zadevami, ki so nam znane do sedaj, bi to lahko poizkusili zapakirati v dictionary.

In [1]:
```
pes = {
    "ime": "Fido",
    "starost": 9,
    "opis": "" # radi bi, da nam kle pove opis psa
}
```

Vse to lahko združimo s pomočjo Class.

In [2]:
```
# Primer: Kako bi izgledal tak class.
# Kaj kera vrstica pomen si bomo pogledal v nadaljevanju
class Pes: #defines the class
    def __init__(self, ime, starost): #special function, dunder functions
        self.ime = ime #creating new variable called name inside our blank object
        self.starost = starost

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

Defining a Class

Začnemo z keyword `class` in nato ime razreda (uporablja se CamelCase način poimenovanja).

```
class ImeRazreda:
    pass
```

Ta razred sedaj predstavlja objekt "Pes". S pomočjo tega razreda lahko sedaj ustvarjamo specifične pse (kjer ima vsak svoje specifično ime, starost, itd.)

Da sedaj ustvarimo našega psa, lahko pokličemo razred in ga shranimo v spremenljivko.

```
x = ImeRazreda()
```

x je sedaj **instanca razreda**.

In [6]:
```
class Pes:
    pass
```

In [7]:
```
a = Pes()
print(a)
print(type(a))
```

```
<__main__.Pes object at 0x00000214DBF60730>
<class '__main__.Pes'>
```

Znotraj našega Class-a definiramo metodo `__init__()` .

Ko ustvarimo novo instanco našega Class-a (novega specifičnega psa), se pokliče ta metoda.

> `__init__()` se uporablja podobno kot "konstruktor" (iz drugih jezikov), čeprov ni točno konstruktor.

In [11]:
```
class Pes:
    def __init__(self):
        print("Ustvarili smo novega psa")

fido = Pes()
print(fido)
print(type(fido))

print()

rex = Pes()
print(rex)
print(type(rex))
```

```
Ustvarili smo novega psa
<__main__.Pes object at 0x00000214DBF99460>
<class '__main__.Pes'>

Ustvarili smo novega psa
<__main__.Pes object at 0x00000214DBF99730>
<class '__main__.Pes'>
```

Znotraj našega razreda lahko ustvarimo spremenljivke, specifične posamezni instanci razreda.

Da ustvarimo spremenljivko specifično instanci razreda se uporabi sledeča sintaksa:

```
self.ime = "Fido"
```

Do spremenljivke dostopamo na sledeč način:

```
print(moj_pes.ime)
Output: Fido
```

V našem primeru bomo vsakemu specifičnemu psu pripisali njegovo ime. Psu bomo ime določili takoj, ko ga ustvarimo. Zato bomo njegovo ime posredovali __init__ metodi.

In [13]:
```
class Pes: #defines the class
    def __init__(self, ime, starost): #ime, starost so argumenti, k jih posreduje
        self.ime = ime #creating new variable called name inside our blank object
        self.starost = starost
```

In [14]:
```
fido = Pes("Fido", 9)
print(fido.ime)
print(fido.starost)
print()

rex = Pes("Rex", 12)
print(rex.ime)
print(rex.starost)
```

```
Fido
9

Rex
12
```

`self` parameter je instanca razreda in je avtomatično posredovana kot prvi parameter vsaki metodi našega Class-a.

`self.ime = ime` v naši kodi tako pomeni, da naši instanci pripišemo to spremenljivko. Če bi za razliko napisali `Pes.ime = ime` pa bi spremenljivko name spremenili za celoten Class (ker pa vsi psi nimajo enakega imena ostanemo pri `self.ime = ime`).

> method definitions have self as the first parameter, and we use this variable inside the method bodies – but we don't appear to pass this parameter in. This is because whenever we call a method on an object, the object itself is automatically passed in as the first parameter. This gives us a way to access the object's properties from inside the object's methods.

> In some languages this parameter is implicit – that is, it is not visible in the function signature – and we access it with a special keyword. In Python it is explicitly exposed. It doesn't have to be called self, but this is a very strongly followed convention.

In [15]:
```
class Pes: #defines the class
    def __init__(self, ime, starost): #ime, starost so argumenti, k jih posreduje
        print(self) # dobiš isto, k če daš print(fido)
        print(type(self))
        print(id(self))
        self.ime = ime #creating new variable called name inside our blank object
        self.starost = starost

fido = Pes("Fido", 9)
print(fido.ime)
print(fido.starost)
print()

print(fido)
print(type(fido))
print(id(fido))
```

```
<__main__.Pes object at 0x00000214DBF991F0>
<class '__main__.Pes'>
2288613167600
Fido
9

<__main__.Pes object at 0x00000214DBF991F0>
<class '__main__.Pes'>
2288613167600
```

Naloga:

Ustvarite razred Vozilo. Vsaka instanca naj ima svojo specifično hitrost in kilometrino.

Izpišite njegove lastnosti na sledeč način:

```
"Max hitrost vozila: -hitrost-. Prevozenih je -kilometrina- km."
```

Primeri:

```
Input:
avto = Vozilo(300, 80)

Output:
Max hitrost vozila: 300. Prevozenih je 80 km.

Input:
motor = Vozilo(180, 33)

Output:
Max hitrost vozila: 180. Prevozenih je 33 km.
```

In [10]:
```
class Vozilo:
    def __init__(self, hitrost, kilometrina):
        self.hitrost = hitrost
        self.kilometrina = kilometrina

avto = Vozilo(300, 80)
motor = Vozilo(180, 33)

print(f"Max hitrost vozila: {avto.hitrost}. Prevozenih je {avto.kilometrina} km.'
print(f"Max hitrost vozila: {motor.hitrost}. Prevozenih je {motor.kilometrina} km
```

```
Max hitrost vozila: 300. Prevozenih je 80 km.
Max hitrost vozila: 180. Prevozenih je 33 km.
```

In [ ]:
```

```

Znotraj razreda lahko definiramo tudi naše metode s katerimi lahko dostopamo in obdelujemo podatke naših instanc.

Vsaka funkcija/metoda ima najmanj 1 parameter in to je `self` , ki predstavlja instanco razreda in je avtomatično posredovana kot prvi parameter vsaki metodi našega Class-a.

In [16]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        print("Metoda razreda Pes")
```

In [18]:
```
fido = Pes("Fido", 9)
rex = Pes("Rex", 12)

fido.opis()
rex.opis()
```

```
Metoda razreda Pes
Metoda razreda Pes
```

In [22]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

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

fido = Pes("Fido", 9)
rex = Pes("Rex", 12)

fido.opis()
rex.opis()
```

```
Fido je star 9
Rex je star 12
```

In [23]:
```
# Se prav mi lahko uporabmo našo instanco objekta in kličemo njeno metodo na slec
fido.opis()

print()

# Oziroma, lahko kličemo direktno Class metodo opis() in sami posredujemo "self"
Pes.opis(fido)
```

```
Fido je star 9

Fido je star 9
```

In [ ]:
```

```