Python OOP

# Osztályok

Az osztályokat arra használjuk, hogy felhasználó által definiált adatszerkezeteket hozzunk létre. Az osztályok meghatározzák a függvényeket (metódusok), amelyek azokat a viselkedéseket és műveleteket határozzák meg, amelyeket az osztályból létrehozott objektum elvégezhet az adataival.

Létre fogunk hozni egy Kutya osztályt, amely tárol néhány információt a jellemzőkről és viselkedésekről, amiket egy adott kutya felvehet.

Az osztály (class) egy tervrajz arról, hogy hogyan kell valamit meghatározni. Valójában nem tartalmaz adatokat. A Kutya osztály meghatározza, hogy egy kutya meghatározásához szükséges egy név és egy kor, de nem tartalmazza egy adott kutya nevét vagy korát.

Míg az osztály a tervrajz, egy példány egy olyan objektum (object, instance), amelyet az osztályból építünk és valós adatokat tartalmaz. Egy Kutya osztály példánya már nem egy tervrajz. Ez egy valódi kutya egy névvel, mint például Miles, aki négy éves.

Más szóval, az osztály olyan, mint egy űrlap vagy kérdőív. Egy példány pedig olyan, mint egy kitöltött űrlap információkkal. Ahogyan sok ember kitöltheti ugyanazt az űrlapot saját egyedi információikkal, ugyanúgy sok példány hozható létre egyetlen osztályból.

Minden osztálydefiníció a **class** kulcsszóval kezdődik, amelyet az osztály neve és egy kettőspont követ. Az osztálydefiníció alatt behúzott kód része az osztály testének számít.

Itt egy példa a Kutya osztályra:

class Dog:

pass

A Kutya osztály teste egyetlen utasításból áll: a **pass** kulcsszóból. A pass gyakran használatos placeholderként, amely jelzi, hogy kód később kerül beillesztésre. Lehetővé teszi a kód futtatását anélkül, hogy a Python hibát dobna.

Megjegyzés: A Python osztályneveket hagyomány szerint CapitalizedWords jelöléssel írják. Például, egy Jack Russell Terrier nevű specifikus kutyafajta osztályát JackRussellTerrier-ként írnánk.

A Kutya osztály jelenleg nem túl érdekes, ezért feldobjuk egy kicsit azzal, hogy definiálunk néhány tulajdonságot, amelyekkel minden Kutya objektum rendelkezhet. Több tulajdonság közül választhatunk, beleértve a nevet, kort, bundaszínt és fajtát. A dolgok egyszerűsítése érdekében csak a nevet és kort fogjuk használni.

Az összes Kutya objektum által rendelkezett tulajdonságokat egy **.** **\_\_init \_\_()** nevű metódusban kell definiálni. Minden alkalommal, amikor létrehozunk egy új Kutya objektumot, a . \_\_init \_\_() beállítja az objektum kezdeti állapotát az objektum tulajdonságainak értékeivel. Tehát, . \_\_init \_\_() inicializálja az osztály új példányait.

. \_\_init \_\_() tetszőleges számú paramétert kaphat, de az első paraméter mindig egy **self** nevű változó lesz. Amikor egy új osztály példányt hozunk létre, az példány automatikusan átadódik a self paraméternek a . \_\_init \_\_()-ben, hogy az új **attribútumok** definiálhatók legyenek az objektumon.

Frissítsük a Kutya osztályt egy . \_\_init \_\_() metódussal, amely létrehozza a .name és .age attribútumokat:

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Fontos megjegyezni, hogy a . \_\_init \_\_() metódus fejléce négy szóközzel behúzott. A metódus testét nyolc szóközzel behúzzuk. Ez a behúzás lényeges. A Pythonnak ezzel jelzed, hogy a . \_\_init \_\_() metódus a Kutya osztályhoz tartozik.

A . \_\_init \_\_() metódus testében két utasítás található a self változó mellett:

* A self.name = name egy name nevű attribútumot hoz létre és hozzárendeli neki a name paraméter értékét.
* A self.age = age egy age nevű attribútumot hoz létre és hozzárendeli neki az age paraméter értékét.

A . \_\_init \_\_()-ben létrehozott attribútumokat **példányattribútum**oknak nevezzük. Egy példányattribútum értéke specifikus a osztály egy adott példányára. Minden Kutya objektumnak van neve és kora, de a nevük és koruk attribútumainak értéke más az egyes Kutya példányokban.

Másrészt, **osztályattribútum**ok olyan attribútumok, amelyeknek az értéke az összes osztálypéldányra azonos. Osztályattribútumot hozhatsz létre, ha értéket rendelsz egy változónak a . \_\_init \_\_()-en kívül.

Például az alábbi Kutya osztálynak van egy species nevű osztályattribútuma, aminek az értéke "Canis familiaris":

class Dog:

# Class attribute

species = "Canis familiaris"

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Osztályattribútumokat a osztálynév első sorának alatt definiálunk és négy szóközzel behúzzuk. Mindig **inicializált** értéket kell hozzárendelnünk. Amikor létrehozunk egy osztály példányt, az osztályattribútumok automatikusan létrejönnek és az inicializált értékeket kapják.

Osztályattribútumokat használjunk olyan tulajdonságok definiálására, amelyeknek az értéke az összes osztálypéldányra azonos. Példányattribútumokat használjunk olyan tulajdonságokra, amelyek különböznek az egyes példányok között.

Most, hogy van egy Kutya osztályunk, hozzunk létre néhány kutyát!

# Példányosítás

Írjuk be a következő kódot:

class Dog:

pass

Ez létrehoz egy új, attribútumokkal vagy metódusokkal nem rendelkező Kutya osztályt.

Egy új objektum létrehozását egy osztályból objektum példányosításnak nevezzük. Egy új Kutya objektumot példányosíthatsz a osztály nevének beírásával, amelyet zárójelek követnek:

(az >>> jelölés azt jelenti, hogy az adott parancsot terminalban adtuk ki)

>>> Dog()

<\_\_main\_\_.Dog object at 0x106702d30>

Ezzel most egy új Kutya objektumot hoztál létre, amelynek a memóriacíme 0x106702d30. Ez a furcsán kinéző betűk és számok sorozata egy memóriacím, amely jelzi, hogy a Kutya objektum hol található a számítógéped memóriájában. Fontos megjegyezni, hogy a képernyődön látható cím más lesz, mint ami itt van.

Most hozz létre egy második Kutya objektumot:

>>> Dog()

<\_\_main\_\_.Dog object at 0x0004ccc90>

Az új Kutya példány egy másik memóriacímen található. Ez azért van így, mert ez teljesen új példány, és más, mint az első példány, amit létrehoztál.

Hogy másképp is lássuk ezt, írd be a következőt:

>>> a = Dog()

>>> b = Dog()

>>> a == b

False

Ebben a kódban két új Kutya objektumot hozol létre és hozzárendeled őket az a és b változókhoz. Amikor az a és b változókat az == operátorral hasonlítod össze, az eredmény False lesz. Habár az a és b mindkettő a Kutya osztály példányai, két különböző objektumot képviselnek a memóriában.

Hozz létre most egy új Kutya osztályt, amelynek van egy osztályattribútuma, a .species, valamint két példányattribútuma, a .name és az .age:

class Dog:

species = "Canis familiaris"

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Az osztály példányainak létrehozásához meg kell adnod értékeket a name és age paramétereknek. Ha nem teszed meg, akkor a Python TypeError kivételt dob:

>>> Dog()

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

Dog()

TypeError: \_\_init\_\_() missing 2 required positional arguments: 'name' and 'age'

Az értékek átadásához a name és age paraméterekhez, helyezz értékeket a zárójelekbe az osztály neve után:

buddy = Dog("Buddy", 9)

miles = Dog("Miles", 4)

Ezzel két új Kutya példányt hozol létre – egyet Buddy nevű, kilenc éves kutya számára, és egyet Miles nevű, négy éves kutya számára.

A Dog osztály .init() metódusának három paramétere van, tehát miért csak két argumentumot adunk át neki a példában?

Amikor egy Kutya objektumot példányosítasz, a Python létrehoz egy új példányt és átadja azt a .init() metódus első paraméterének. Ez gyakorlatilag kiszedi a self paramétert, így csak a name és age paraméterekkel kell foglalkoznod.

Miután létrehoztad a Kutya példányokat, elérheted a példányattribútumaikat pont jelöléssel (**dot notation**):

>>> buddy.name

'Buddy'

>>> buddy.age

9

>>> miles.name

'Miles'

>>> miles.age

4

Az osztályattribútumokhoz ugyanúgy férhetsz hozzá:

>>> buddy.species

'Canis familiaris'

Az osztályok használatának egyik legnagyobb előnye, hogy a példányoknak garantáltan megvannak az elvárt attribútumai. Minden Kutya példánynak van .species, .name és .age attribútuma, így ezeket az attribútumokat magabiztosan használhatod, tudva, hogy mindig fognak értéket visszaadni.

Bár az attribútumok létezése garantált, az értékeik dinamikusan megváltoztathatók:

>>> buddy.age = 10

>>> buddy.age

10

>>> miles.species = "Felis silvestris"

>>> miles.species

'Felis silvestris'

Ebben a példában megváltoztatod a buddy objektum .age attribútumát 10-re. Ezután megváltoztatod a miles objektum .species attribútumát "Felis silvestris"-re, ami egy macskafajta. Tehát Miles egy elég furcsa kutya lesz, de ez érvényes Python kód!

A kulcs megállapítás itt az, hogy a saját osztályok alapértelmezetten módosíthatók. Egy objektum módosítható (**mutable**), ha dinamikusan megváltoztatható. Például listák és szótárak módosíthatóak, míg stringek és tuple-ök nem módosíthatóak.

# Példány metódusok

Az **instance metódusok** olyan függvények, amelyeket az osztályon belül definiálnak, és csak az adott osztály példányából hívhatóak meg. Pontosan úgy, mint a .init(), az instance metódus első paramétere mindig a self.

Nyiss egy új szerkesztőablakot, és írd be a következő Kutya osztályt:

class Dog:

species = "Canis familiaris"

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# Instance method

def description(self):

return f"{self.name} is {self.age} years old"

# Another instance method

def speak(self, sound):

return f"{self.name} says {sound}"

Ez a Kutya osztály két instance metódust tartalmaz:

* .description() egy olyan karakterláncot ad vissza, amely tartalmazza a kutya nevét és életkorát.
* .speak() egy sound nevű paraméterrel rendelkezik, és egy olyan karakterláncot ad vissza, amely a kutya nevét és a hangot tartalmazza, amit a kutya kiad.

Nézzük, mit csinál a kód:

miles = Dog("Miles", 4)

miles.description()

miles.speak("Woof Woof")

miles.speak("Bow Wow")

A fenti Kutya osztályban a .description() egy karakterláncot ad vissza, amely információkat tartalmaz a miles Kutya példányról. Amikor a saját osztályokat írod, hasznos lehet olyan metódust létrehozni, amely visszaad egy olyan karakterláncot, amely hasznos információkat tartalmaz az osztály egy példányáról. Azonban a .description() nem a legpythonikusabb módja ennek.

Amikor egy lista objektumot hozol létre, használhatod a print()-et egy olyan karakterlánc megjelenítésére, amely úgy néz ki, mint a lista. Nézzük ezt a kódot:

names = ["Fletcher", "David", "Dan"]

print(names)

Nézzük meg, mi történik, amikor a miles objektumot kiíratod a print()-tel:

print(miles)

Amikor a print(miles)-t írod, egy rejtélyes üzenetet kapsz, amely azt mondja neked, hogy a miles egy Kutya objektum a 0x00aeff70 memóriacímen. Ez az üzenet nem túl hasznos. Megváltoztathatod, hogy mi jelenjen meg a print()-tel, ha definiálsz egy különleges instance metódust, amely .str() néven fut.

class Dog:

# Leave other parts of Dog class as-is

# Replace .description() with \_\_str\_\_()

def \_\_str\_\_(self):

return f"{self.name} is {self.age} years old"

A szerkesztőablakban változtasd meg a Kutya osztály .description() metódusának nevét .str() –ra és nézzük mit ír ki ez:

miles = Dog("Miles", 4)

print(miles)

Az olyan metódusokat, mint a .init() és .str(), **dunder metódus**oknak nevezik, mert kétszeres aláhúzásjellel kezdődnek és végződnek. Sok dunder metódust használhatsz az osztályok testreszabásához Pythonban. A dunder metódusok megértése fontos része a Pythonban az objektumorientált programozás elsajátításának.

# Feladat

Hozz létre egy Car osztályt két példány attribútummal:

* .color, amely egy stringként tárolja az autó színét
* .mileage, amely egy egész számként tárolja az autó megtett mérföldjeinek számát

Ezután hozz létre két Car objektumot - egy kék autót 20 000 mérfölddel és egy piros autót 30 000 mérfölddel - és írd ki a színeiket és a megtett mérföldjeiket. Az eredmény a következőképpen nézzen ki:

The blue car has 20,000 miles.

The red car has 30,000 miles.

# Öröklés

Az **öröklődés** azt a folyamatot jelenti, amikor egy osztály átveszi egy másik osztály attribútumait és metódusait. Az így létrehozott új osztályokat **gyermek osztály**oknak nevezzük, míg azokat az osztályokat, amelyekből a gyermek osztályok származnak, **szülő osztály**oknak nevezzük.

A gyermek osztályok felülírhatják vagy kibővíthetik a szülő osztályok attribútumait és metódusait. Más szóval, a gyermek osztályok öröklik a szülő összes attribútumát és metódusát, de saját, egyedülálló attribútumokat és metódusokat is megadhatnak.

Bár az analógia nem tökéletes, az objektum öröklődést hasonlóan lehet elképzelni, mint a genetikai öröklődést.

Lehet, hogy hajszínedet anyukádtól örökölted. Ez egy olyan attribútum, amellyel megszülettél. Tegyük fel, hogy úgy döntesz, lila hajat festesz. Feltéve, hogy anyukádnak nincs lila haja, felülírtad (**override**) az anyukádtól örökölt hajszín attribútumot.

A nyelvet is örökölted szülőidtől. Ha szüleid angolul beszélnek, akkor te is angolul fogsz beszélni. Most képzeld el, hogy úgy döntesz, megtanulsz egy második nyelvet, például németül. Ebben az esetben bővítetted (**extend**) az attribútumokat, mert hozzáadtál egy olyan attribútumot, amely szüleidnek nincs.

Példa kutyaparkkal

Tegyük fel, hogy egy kutyaparkban vagy. Sokféle kutyafajta van a parkban, amelyek különböző kutyaviselkedéseket mutatnak.

Most tegyük fel, hogy Python osztályokkal szeretnéd modellezni a kutyaparkot. A korábban írt Dog osztály különbséget tud tenni a kutyák között név és életkor alapján, de nem fajta szerint.

A Dog osztályt módosíthatod úgy, hogy hozzáadod a .breed attribútumot:

class Dog:

species = "Canis familiaris"

def \_\_init\_\_(self, name, age, breed):

self.name = name

self.age = age

self.breed = breed

A korábban definiált példánymetódusokat kihagyam mert nem fontosak ebben a részben.

Most modellezheted a kutyaparkot a különböző kutyák példányosításával:

miles = Dog("Miles", 4, "Jack Russell Terrier")

buddy = Dog("Buddy", 9, "Dachshund")

jack = Dog("Jack", 3, "Bulldog")

jim = Dog("Jim", 5, "Bulldog")

Minden kutyafajtának kissé más viselkedése van. Például, a bulldogok alacsony hangon ugatnak, ami olyan, mint a vuh, de a tacskók magasabb hangon ugatnak, ami inkább hasonlít a jipp hangra.

Csak, hogy használjuk a Dog osztályt, minden alkalommal meg kell adnod egy karakterláncot a .speak() metódus sound argumentumaként:

buddy.speak("Yap")

jim.speak("Woof")

jack.speak("Woof")

Minden híváshoz karakterlánc átadása ismétlődő és kényelmetlen. Ráadásul a hangot reprezentáló karakterláncot, amelyet minden Dog példány kibocsát, a .breed attribútuma határozza meg, de itt manuálisan meg kell adnod a megfelelő karakterláncot minden .speak() metódus hívásakor.

Egyszerűsítheted a Dog osztály használatát azzal, hogy létrehozol egy gyermek osztályt minden kutyafajtához. Ez lehetővé teszi, hogy kiterjeszd a gyermek osztályok által örökölt funkcionalitást, ideértve a .speak() metódusnak egy alapértelmezett argumentumának meghatározását is.

## Gyermek és szülő osztályok

Hozzunk létre egy-egy gyermekosztályt a fent említett három fajtának: Jack Russell Terrier, Tacskó és Bulldog.

Emlékeztetőként itt van a Dog osztály teljes definíciója:

class Dog:

species = "Canis familiaris"

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return f"{self.name} is {self.age} years old"

def speak(self, sound):

return f"{self.name} says {sound}"

Ne felejtsd el, hogy egy gyermek osztály létrehozásához új osztályt hozol létre saját névvel, majd a szülő osztály nevét helyezed zárójelekbe. Add hozzá a következő kódot a dog.py fájlhoz, hogy három új gyermek osztályt hozz létre a Dog osztályból:

class JackRussellTerrier(Dog):

pass

class Dachshund(Dog):

pass

class Bulldog(Dog):

pass

A gyermek osztályok definiálása után most példányosíthatsz néhány specifikus fajtájú kutyát:

miles = JackRussellTerrier("Miles", 4)

buddy = Dachshund("Buddy", 9)

jack = Bulldog("Jack", 3)

jim = Bulldog("Jim", 5)

A gyermek osztályok példányai öröklik a szülő osztály összes attribútumát és metódusát. Nézzük meg, mit ad vissza ez:

miles.species

buddy.name

print(jack)

jim.speak("Woof")

Ahhoz, hogy meghatározzuk, hogy egy adott objektum melyik osztályhoz tartozik, használhatjuk a beépített type() függvényt:

>>> type(miles)

<class '\_\_main\_\_.JackRussellTerrier'>

Mi van akkor, ha szeretnénk megállapítani, hogy a miles példány is a Dog osztály egy példánya-e? Ezt megteheted a beépített isinstance() függvénnyel:

>>> isinstance(miles, Dog)

True

Figyeld meg, hogy az isinstance() függvény két argumentumot vesz át: egy objektumot és egy osztályt. A fenti példában az isinstance() megvizsgálja, hogy a miles példány a Dog osztály egy példánya-e, és True értéket ad vissza.

A miles, buddy, jack és jim objektumok mind Dog példányok, de a miles nem Bulldog példány, és a jack nem Tacskó példány:

>>> isinstance(miles, Bulldog)

False

>>> isinstance(jack, Dachshund)

False

Általánosságban elmondható, hogy minden gyermek osztályból létrehozott objektum az ősosztály példánya, bár más gyermek osztályok példányai nem lehetnek.

Most, hogy létrehoztál gyermek osztályokat különböző kutyafajtákhoz, adjunk meg minden fajtának saját hangot.

Mivel a különböző kutyafajták enyhén eltérő ugatással rendelkeznek, szeretnél egy alapértelmezett értéket megadni a hang argumentumra az .speak() metódusukban. Ehhez felül kell írnod az .speak() metódust minden fajta osztályának a definíciójában.

Az ősosztályon definiált metódus felülírásához az azonos nevű metódust kell definiálnod a gyermek osztályon. Íme, hogy néz ki ez a JackRussellTerrier osztályra:

class JackRussellTerrier(Dog):

def speak(self, sound="Arf"):

return f"{self.name} says {sound}"

Most az .speak() metódus van meghatározva a JackRussellTerrier osztályban, ahol az alapértelmezett argumentum hangja "Arf".

Frissítsd a dog.py fájlt az új JackRussellTerrier osztállyal, majd nyomj F5-öt a fájl mentéséhez és futtatásához. Most már hívhatod az .speak() metódust egy JackRussellTerrier példányon anélkül, hogy átadnád a hang argumentumot:

miles = JackRussellTerrier("Miles", 4)

miles.speak()

Néha a kutyák másféle ugatásokat is hallatnak, tehát ha Miles mérges lesz és morogni kezd, akkor is hívhatod az .speak() metódust másféle hanggal:

>>> miles.speak("Grrr")

'Miles says Grrr

Egy dolog, amit érdemes megjegyezni az osztály öröklődésével kapcsolatban, hogy a változások az ősosztályra automatikusan átterjednek a gyermek osztályokra. Ez akkor történik, ha az adott attribútumot vagy metódust nem írják felül a gyermek osztályban.

Például az editor ablakban változtasd meg a Dog osztályban visszaadott stringet az .speak() metódusban:

class Dog:

# Leave other attributes and methods as they are

# Change the string returned by .speak()

def speak(self, sound):

return f"{self.name} barks: {sound}"

Mentsd el a fájlt és nyomj F5-öt. Most, amikor létrehozol egy új Bulldog példányt, jim néven, jim.speak() az új stringet adja vissza:

>>> jim = Bulldog("Jim", 5)

>>> jim.speak("Woof")

'Jim barks: Woof'

Azonban az .speak() hívása egy JackRussellTerrier példányon nem fogja mutatni az új kimeneti stílust:

>>> miles = JackRussellTerrier("Miles", 4)

>>> miles.speak()

'Miles says Arf'

Néha ésszerű teljesen felülírni egy metódust az ősosztályból. De ebben az esetben nem akarod, hogy a JackRussellTerrier osztály elveszítse azokat a változásokat, amelyeket a Dog.speak() kimeneti stringjének formázására végeztek.

Ehhez még mindig szükséged van egy .speak() metódus meghatározására a JackRussellTerrier gyermekosztályban. De ahelyett, hogy kifejezetten definiálnád a kimeneti stringet, a gyermekosztály .speak() metódusán belül meg kell hívnod a Dog osztály .speak() metódusát, ugyanazokkal az argumentumokkal, amelyeket átadtál a JackRussellTerrier.speak() metódusnak.

Az ősosztályhoz való hozzáférést egy gyermekosztály metódusából a super() függvénnyel teheted meg:

class JackRussellTerrier(Dog):

def speak(self, sound="Arf"):

return super().speak(sound)

Amikor a JackRussellTerrier osztályon belül meghívod a super().speak(sound) kódot, a Python az ősosztályban, a Dog osztályban keresi a .speak() metódust, majd meghívja azt a sound változóval.

Frissítsd a dog.py fájlt az új JackRussellTerrier osztállyal. Nézzük, így hogy működik:

miles = JackRussellTerrier("Miles", 4)

miles.speak()

Most, amikor meghívod a miles.speak() metódust, látni fogod a kimenetet, amely tükrözi a Dog osztályban történt új formázást.

Megjegyzés: A fenti példákban az osztályhierarchia egyértelmű. A JackRussellTerrier osztálynak egyetlen szülőosztálya van, a Dog. A valóságban az osztályhierarchia nagyon bonyolulttá válhat.

A super() sokkal többet tesz, mint csak keres egy módszert vagy attribútumot a szülőosztályban. Az egész osztályhierarchiát átfutja egyező módszerek vagy attribútumok keresése érdekében. Ha nem vagy óvatos, a super() meglepő eredményeket produkálhat.