Opjektum Orientált Programozás

OSZTÁLYOK SPECIÁLIS FÜGGVÉNYEI

Készítette: Vastag Atila

2020

A speciális függvényeket az osztály belsejében kell definiálnunk.

Az összes speciális függvény az __*init*__-hez hasonlóan működik. Az osztály törzsében kell definiálnunk őket. Innen tudja a Python, hogy milyen típusú objektumokon használható, mert amúgy a nevéből nem derül ki. Ezen felül, mindegyiknek az első paramétere az az objektum (*self*), amelyre meghívták. Ezt az összes függvénynél self-nek hívjuk.

Pythonban azokat a metódusokat, amik __-el kezdődnek- és végződnek, **magic method**-nak hívják, és rengeteg létezik belőlük (link).

A magic method-nak szokás szerint az első paraméterét mindig a *self*-nek hívják, de igazából akárhogy el lehet nevezni, viszont nagyon hülyén fognak rátok nézni ha nem így hívjátok.

Binary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__truediv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
|	object.__or__(self, other)

Extended Assignments

+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
|=	object.__ior__(self, other)
+=	object.__iadd__(self, other)

Unary Operators

-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self

Comparison Operators

<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

def __init__(self):

Az *__init__()* név egy különleges metódust takar az osztályon belül: ez az úgynevezett konstruktor (constructor). Ez azt jelenti, hogy a metódus automatikusan fut akkor, amikor példányosítjuk az osztályt, azaz amikor egy új objektumot készítünk az osztály alapján. Ebben a metódusban lehet megadni, hogy a létrejövő objektum milyen kezdeti értékekkel rendelkezzen.

A konstruktorból nem lehet semmilyen értékkel se visszatérni.

def __str__(self):

```
class Car:
    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage

    def __str__(self):
        return (f"A {self.color} car.")

#main.py
car: Car = Car("red", 37281)
print(car)

#output
A red car.
```

Objektum átalakítása szövegre (string).

def __len__(self)

```
class Car:
    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage

    def __str__(self):
        return (f"A {self.color} car.")
```

```
import typing

class CarCollection:
    def __init__(self):
        self.collection: List[Car] = []

    def __len__(self) -> int:
        return len(self.collection)

    def addToCollection(self, car: Car) -> None:
        self.collection.append(car)
```

def __len__(self)

```
#main
ferrari: Car = Car("red", 0)
lamborghini: Car = Car("black", 0)

cars: CarCollection = CarCollection()
cars.addToCollection(ferrari)
cars.addToCollection(lamborghini)

print(f"There are {len(cars)} cars in the collection")

for car in cars.collection:
    print(car)

#output

There are 2 cars in the collection
A red car.
A black car.
```

A **len()** függvény megpróbálja meghívni az osztály __len()__() függvényét.

def __del__(self)

```
def __del__(self):
  # body of destructor
```

A destruktort akkor hívjuk meg, amikor egy objektumot el akarunk pusztítani, azaz ki akarjuk törölni a memóriából. A Pythonban a destruktátorokra nincs annyira szükség, mint a C vagy a C++-ban, mert a Python rendelkezik egy szemétszedővel *(garbage collector),* amely automatikusan kezeli a memóriát.

Az __del __ () metódus destruktor módszerként ismert Pythonban. Ezt akkor hívják, ha az objektumra vonatkozó összes hivatkozás törlődik, azaz már egyetlen mutató sem mutat (hivatkozik) az objektumra.

def __add__(self, *parameter*)

```
import typing

class Day(object):
    def __init__(self, visits, contacts):
        self.visits = visits
        self.contacts = contacts

    def __add__(self, other: Day):
            total_visits = self.visits + other.visits
            total_contacts = self.contacts + other.contacts
            return Day(total_visits, total_contacts)

    def __str__(self):
        return f"Visits: {self.visits}, Contacts: {self.contacts}"

#main
day1 = Day(10, 1)
day2 = Day(20, 2)
day3 = day1 + day2
print(day3)
#output
Visits: 30, Contacts: 3
```