Python OOP

18-dars

Abstraction (Abstraksiya)

Abstraction (abstraksiya) obyektga tegishli murakkab ma'lumotlar va funksionallikni yashirib, faqat zaruriy qismlarini taqdim etish usulidir. Bu OOP (obyektga yo‘naltirilgan dasturlash) ning muhim tamoyillaridan biri bo‘lib, abstraksiya yordamida foydalanuvchi faqat kerakli funksiyalar bilan ishlaydi va obyektning murakkab ichki tuzilishidan xabardor bo‘lishi shart emas.

Abstraksiya bilan biz obyektning eng muhim tomonlariga e'tibor qaratamiz, murakkab tafsilotlar esa yashiriladi. Masalan, bir avtomobilni boshqarishda biz faqat rullarni burish, tormoz va gaz pedalidan foydalanamiz; lekin uning ichki mexanizmlarini, dvigatelning qanday ishlashini tushunishimiz shart emas.

Abstractionning Asosiy Afzalliklari

1. **Murakkablikni yashirish**: Obyektning murakkab ichki mexanizmlari foydalanuvchidan yashiriladi, bu esa interfeysni soddalashtiradi.
2. **Ishlash qulayligi**: Foydalanuvchilar faqat kerakli funksiyalarni bilishadi va foydalanadilar, shuning uchun ular obyektni qanday ishlashiga emas, balki qanday foydalanishga e’tibor qaratadilar.
3. **Modullilik va moslashuvchanlik**: Dasturiy komponentlar orasidagi bog‘liqlikni kamaytiradi, bu esa ularni alohida rivojlantirish va o‘zgartirishni osonlashtiradi.

Abstract Class (Abstrakt class)

Abstract class bu obyektlarning umumiy funksiyalarini belgilab, ularning qanday amalga oshirilishini farzand classlarga qoldiradigan classdir. Abstract classning o‘zi to‘g‘ridan-to‘g‘ri obyekt yaratishga yaroqsiz bo‘lib, u faqat boshqa classlarga umumiy interfeys sifatida xizmat qiladi.

Python'da abstract class yaratish uchun `abc` (abstract base classes) moduli va `ABC` klassidan foydalaniladi. Abstract classdagi abstract metodlar `@abstractmethod` dekoratori yordamida belgilanadi.

Bu misolda `Vehicle` abstract class bo‘lib, u faqat umumiy metodlar (`start` va `stop`) ni belgilaydi, lekin ularni amalga oshirmaydi. `Car` va `Bike` esa `Vehicle` classdan meros oladi va bu metodlarni o‘ziga xos tarzda amalga oshiradi.

```python
from abc import ABC, abstractmethod
class Vehicle(ABC):  # Abstract class
    @abstractmethod
    def start(self):
        pass
    @abstractmethod
    def stop(self):
        pass
class Car(Vehicle):  # Car class abstract classni meros oladi
    def start(self):
        print("Car is starting...")
    def stop(self):
        print("Car is stopping...")
class Bike(Vehicle):  # Bike class abstract classni meros oladi
    def start(self):
        print("Bike is starting...")
    def stop(self):
        print("Bike is stopping...")
# Abstract classdan to'g'ridan-to'g'ri obyekt yaratib bo'lmaydi
# vehicle = Vehicle()  # Bu xatolik beradi
# Abstract classdan olingan classlar orqali obyekt yaratish mumkin
car = Car()
car.start()  # Output: Car is starting...
car.stop()   # Output: Car is stopping...
bike = Bike()
bike.start()  # Output: Bike is starting...
bike.stop()   # Output: Bike is stopping...
```

Magic Methods (Dunder Methods)

Magic Methods yoki **Dunder Methods** (double underscore methods) — bu Python'da oldindan belgilangan maxsus metodlar bo‘lib, ular obyektlarning xatti-harakatlarini boshqaradi. Ushbu metodlarning nomi ikki pastki chiziq (`__`) bilan boshlanadi va tugaydi, shuning uchun ular "dunder" (double underscore) metodlari deb ham ataladi.

Magic metodlar ko‘pincha **obyektning ko‘rinishi**, **arifmetik operatsiyalar** yoki **obyektning o‘zini tutsishi** kabi vazifalarni avtomatik ravishda amalga oshirish uchun ishlatiladi. Siz bu metodlarni to‘g‘ridan-to‘g‘ri chaqirmaysiz, balki Python avtomatik ravishda ularni tegishli operatsiyalar paytida ishga tushiradi.

Magic Methodsning Afzalliklari

1. **Obyektni qulay boshqarish**: Magic metodlar yordamida obyektning ichki xatti-harakatlarini boshqarish mumkin.
2. **Murakkab operatsiyalarni sodda qilish**: Ular obyektlarga o‘ziga xos operatsiyalarni qo‘llash imkonini beradi, masalan, arifmetik operatorlar yoki solishtirish operatorlari.
3. **Pythonning ichki imkoniyatlarini kengaytirish**: Python’ning turli operatorlari va funksiyalarini o‘zingizning obyektlaringizda ishlatiladigan qilib sozlash mumkin.

Eng Ko‘p Foydalaniladigan Magic Methods

`__init__` (Initsializatsiya Metodi):

Bu metod obyekt yaratilganda avtomatik ravishda chaqiriladi. Odatda obyektning atributlarini boshlang'ich qiymatga o'rnatish uchun ishlatiladi.

```
class Car:
    def __init__(self, model, year):
        self.model = model
        self.year = year

car = Car("Tesla", 2023)
print(car.model)   # Output: Tesla
```

`__str__` va `__repr__` (Obyektni Ko‘rinishi)

`__str__` metodi obyektni foydalanuvchi uchun tushunarli tarzda chop etish uchun ishlatiladi.

`__repr__` metodi obyektni rasmiy tasvirini (developerlar uchun tushunarli ko'rinishini) yaratadi.

```
class Car:
    def __init__(self, model, year):
        self.model = model
        self.year = year

    def __str__(self):
        return f"Car: {self.model}, {self.year}"

    def __repr__(self):
        return f"Car('{self.model}', {self.year})"

car = Car("Tesla", 2023)
print(car)   # Output: Car: Tesla, 2023
print(repr(car))   # Output: Car('Tesla', 2023)
```

`__add__`, `__sub__`, `__mul__`, va boshqa arifmetik metodlar

Bu metodlar arifmetik operatorlarni (`+`, `-`, `*`, `/`) obyektlar orasida ishlatilishini belgilaydi.

Bu yerda `__add__` metodi yordamida `+` operatorini obyektlar orasida ishlatish mumkin.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f"Vector({self.x}, {self.y})"
v1 = Vector(2, 3)
v2 = Vector(1, 5)
v3 = v1 + v2
print(v3)  # Output: Vector(3, 8)
```

`__len__` (Obyektning uzunligini qaytarish)

Bu metod `len()` funksiyasini obyektlarga ishlatish imkonini beradi.

Bu misolda `__len__` metodi `len()` funksiyasini `MyList` obyektida ishlatish imkonini beradi.

```
class MyList:
    def __init__(self, data):
        self.data = data
    def __len__(self):
        return len(self.data)
my_list = MyList([1, 2, 3, 4])
print(len(my_list))   # Output: 4
```

`__getitem__` va `__setitem__` (Elementlarga kirish)

Bu metodlar obyektning elementlarini indeks orqali o‘qish yoki o‘zgartirish uchun ishlatiladi.

Bu yerda `__getitem__` va `__setitem__` metodlari obyektga indeks orqali kirishni boshqaradi.

```
class MyList:
    def __init__(self, data):
        self.data = data

    def __getitem__(self, index):
        return self.data[index]

    def __setitem__(self, index, value):
        self.data[index] = value

my_list = MyList([10, 20, 30])
print(my_list[1])  # Output: 20
my_list[1] = 99
print(my_list[1])  # Output: 99
```

`__call__` (Obyektni chaqirilishi mumkin qilish)

`__call__` metodi obyektni funksiya kabi chaqirish imkonini beradi.

Bu misolda `greet` obyekti funksiya kabi chaqirilgan va `__call__` metodi ishga tushgan.

```
class Greeting:
    def __init__(self, name):
        self.name = name

    def __call__(self):
        return f"Hello, {self.name}!"

greet = Greeting("John")
print(greet())  # Output: Hello, John!
```

`__eq__`, `__lt__`, va boshqa solishtirish metodlari

Bu metodlar obyektlar orasidagi solishtirish operatorlarini (`==`, `<`, `>`, va boshqalar) boshqaradi.

Bu yerda `__eq__` va `__lt__` metodlari obyektlarni solishtirishni boshqaradi.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __eq__(self, other):
        return self.age == other.age

    def __lt__(self, other):
        return self.age < other.age

p1 = Person("Alice", 25)
p2 = Person("Bob", 30)
print(p1 == p2)  # Output: False
print(p1 < p2)   # Output: True
```