Python OOP

17-dars

Encapsulation (Kapsulalash)

Encapsulation (kapsulalash) — bu obyektlar bilan ishlashda ma'lumotlarni yashirish va ularni tashqi ta'sirlardan himoya qilish prinsipidir. Bu tushuncha obyekt ichidagi ma'lumot va metodlarni bir kapsula ichida saqlashni anglatadi va faqat kerakli metodlar orqali ularga kirish imkonini beradi.

Encapsulation yordamida obyekt ichidagi ma'lumotlar to‘g‘ridan-to‘g‘ri o'zgartirilmasligi yoki ko'rilmasligi uchun himoya qilinadi. Bu esa ma'lumotlar yaxlitligini saqlaydi va obyektlarning qanday ishlashini aniqroq boshqarishga imkon beradi.

Encapsulationning Asosiy Afzalliklari

- **Ma'lumotlarni himoya qilish**: Obyektning atributlari bevosita tashqi dunyo tomonidan o'zgartirilmasligi uchun himoyalanadi.
- **Ma'lumotlarga nazoratlangan kirish**: Atributlar faqat metodlar orqali ko'riladi yoki o'zgartiriladi, bu ma'lumotlarga kirishni boshqarishga yordam beradi.
- **Kodni soddalashtirish va tashqi ta'sirlardan himoyalash**: Kodni kapsulalash orqali har xil obyektlar o'z ichki ishlarini yashirishlari va faqat kerakli funksionallikni taqdim qilishlari mumkin.

Python'da Encapsulation

Python'da encapsulation yordamida atributlar va metodlar **public** (ochiq), **protected** (himoyalangan), yoki **private** (yopiq) bo‘lishi mumkin:

- **Public**: Ma'lumotlarga to‘g‘ridan-to‘g‘ri kirish mumkin.
- **Protected**: Ma'lumotlarga faqat class va uning vorislari (subclass) kirishi mumkin.
- **Private**: Ma'lumotlarga faqat classning o‘zida kirish mumkin va tashqi dunyodan to‘g‘ridan-to‘g‘ri kirishni taqiqlaydi.

Public Atributlar

Odatda, Python'da oddiy nom berilgan atributlar public hisoblanadi. Bunga istalgan joydan kirish va o‘zgartirish mumkin.

Bu yerda `model` va `year` public atributlardir va ularga to‘g‘ridan-to‘g‘ri kirish va o‘zgartirish mumkin.

```python
class Car:
    def __init__(self, model, year):
        self.model = model  # Public atribut
        self.year = year    # Public atribut

car = Car("Tesla", 2023)
print(car.model)  # Output: Tesla
car.model = "BMW"  # Atributni o'zgartirish mumkin
print(car.model)  # Output: BMW
```

Protected Atributlar

Python'da protected atributlar nomining boshiga bitta pastki chiziq (_) qo‘shiladi. Bu konventsiya orqali ushbu atributlar faqat class va uning vorislari tomonidan ishlatilishi kerakligini bildiradi, lekin bu himoyani majburan qo‘llanilmaydi.

Protected atributlar faqat class va uning farzand classlari tomonidan foydalanilishi kerak, lekin Python’da ularni tashqaridan ham o‘zgartirish mumkin.

```python
class Car:
    def __init__(self, model, year):
        self._model = model   # Protected atribut
        self._year = year     # Protected atribut

car = Car("Tesla", 2023)
print(car._model)  # Kirish mumkin, lekin bu tavsiya qilinmaydi
```

Private Atributlar

Private atributlar nomining boshiga ikki pastki chiziq (`__`) qo‘yiladi. Bu atributlar faqat o‘z classi ichida foydalanish uchun mo‘ljallangan va tashqi dunyodan to‘g‘ridan-to‘g‘ri kirish taqiqlanadi.

Bu yerda `__model` va `__year` private atributlar bo‘lib, ularga to‘g‘ridan-to‘g‘ri kirishning imkoni yo‘q. Kirish va o‘zgartirish uchun getter va setter metodlar orqali foydalanish kerak.

```python
class Car:
    def __init__(self, model, year):
        self.__model = model   # Private atribut
        self.__year = year     # Private atribut

    def get_model(self):
        return self.__model

    def set_model(self, model):
        self.__model = model

car = Car("Tesla", 2023)
# print(car.__model)  # Xato: AttributeError, modelga to'g'ridan-to'g'ri kirish mumkin emas
print(car.get_model())  # Output: Tesla
car.set_model("BMW")
print(car.get_model())  # Output: BMW
```

Getter va Setter metodlari

Getter va setter metodlari orqali private atributlarga nazoratlangan kirishni ta'minlash mumkin. Getter atributning qiymatini qaytaradi, setter esa atributning qiymatini o'zgartiradi.

Getter va setter metodlari yordamida ma'lumotlarni qanday o'qish va o‘zgartirishni aniq nazorat qilish mumkin.

```python
class Car:
    def __init__(self, model, year):
        self.__model = model   # Private atribut
        self.__year = year

    # Getter metodi
    def get_model(self):
        return self.__model

    # Setter metodi
    def set_model(self, model):
        self.__model = model

car = Car("Tesla", 2023)
print(car.get_model())   # Output: Tesla
car.set_model("BMW")
print(car.get_model())   # Output: BMW
```

Vazifa

1. `Person` classida ism va yosh atributlarini o‘rnating. Yoshingizni faqat getter va setter metodlari orqali boshqariladigan qilib yarating.
2. Bank hisob raqami uchun `BankAccount` classini yarating, unda balans private bo‘lsin va faqat omonat kiritish va pul olish metodlari orqali boshqarilsin.
3. Classlarni import qilib ishlatinglar

Class va Obyektning Atributlar

Class atributi — bu classning barcha obyektlari uchun umumiy bo'lgan o'zgaruvchilar.

Obyekt atributi — har bir obyektga xos bo'lgan o'zgaruvchilar.

```
class Car:
    wheels = 4  # Bu class atributi bo'lib, barcha obyektlar uchun bir xil bo'ladi

    def __init__(self, model, color):
        self.model = model  # Obyekt atributi
        self.color = color  # Obyekt atributi

# Obyekt yaratish
car1 = Car("Toyota", "Red")
car2 = Car("BMW", "Blue")

print(car1.model)  # Output: Toyota
print(car2.color)  # Output: Blue
print(car1.wheels)  # Output: 4 (class atribut)
```

CLASSGA OID METODLAR

Klasslarning o'ziga xos metodlari ham bo'lishi mumkin. Misol uchun yuqoridagi `wheels` xususiyatini ko'rish uchun alohida metod yozishimiz mumkin. Klassga oid metodlar @classmethod dekoratori bilan boshlanadi va obyektga oid metodlardan farqli ravishda *`self`* emas *`cls`* (class) argumentini qabul qiladi.

CLASSLARNI MODULGA AJRATISH

Vaqt o'tishi bilan dasturimizda klasslar ko'payib borishi tabiiy. Bizning asosiy dasturimiz uzun va chigal bo'lmasligi uchun klasslarni ham huddi funksiyalar kabi alohida modullarga ajratish maqsadga muvofiq bo'ladi. Dastur davomida kerak bo'ladigan klasslarga esa modulni chaqirish (import) orqali murojat qilishimiz mumkin. Bunda, bir-biriga bog'liq klasslarni bitta faylga joylashimiz mumkin.

```python
from car import Car

car1 = Car("Toyota", "Red")
car2 = Car("BMW", "Blue")

print(car1.model)  # Output: Toyota
print(car2.color)  # Output: Blue
print(car1.get_wheels())  # Output: 4 (class atribut)
```