PYTHON DASTURLASH TILIDA INKAPSULYATSIYA, ATRIBUTLAR, XUSUSIYATLAR, VA VORISLIK

Tojimamatov Israil Nurmamatovich
Farg'ona davlat universiteti amaliy matematika va informatika kafedrasi katta o'qituvchisi, israiltojimamatov@gmail.com
Ne'matjonova Navro'za Qahramonjon qizi
Farg'ona davlat unversiteti talabasi, navrozajorayeva03@icloud.com

Annotatsiya: *Python dasturlash tilida* ***atribut*** *— bu obyektga bog'langan ma'lumot yoki funksiyadir. Atributlar ikki turga bo'linadi:* ***klass atributlari*** *va* ***instansiya atributlari. Klass atributi*** *— bu klass darajasida aniqlanadi va barcha obyektlar uchun umumiy bo'ladi. U klass ichida to'g'ridan-to'g'ri aniqlanadi va barcha obyektlar bu atributni baham ko'radi.* ***Instansiya atributi*** *esa _init_ metodida self orqali aniqlanadi va faqat obyektning o'ziga tegishli bo'ladi. Har bir obyekt o'z atributlariga ega bo'lib, ular bir-biriga ta'sir qilmaydi. Shuningdek,* ***inkapsulyatsiya*** *atributlarga (yoki metodlarga) to'g'ridan-to'g'ri kirishni cheklash imkonini beradi. Bu atributlar nomi oldidan _ or _ belgilari bilan belgilanadi.* ***Xususiyatlar*** *esa bir xil metod nomi bilan turli klasslarda turlicha ishlov berishga imkon yaratadi — bu kodni soddalashtiradi va qayta ishlatishni osonlashtiradi.*

Kalit so'zlar:*Atribut, metod, obyektga yo'naltirilgan dasturlash, obyekt, akselerometr, sensor.*

Annotation: *In the Python programming language, an* ***attribute*** *is data or a function that is associated with an object. Attributes are divided into two types:* ***class attributes*** *and* ***instance attributes****. A* ***class attribute*** *is defined at the class level and is shared among all objects. It is defined directly within the class, and all instances of the class share this attribute. An* ***instance attribute****, on the other hand, is defined in the __init__ method using self, and it belongs only to the specific object. Each object has its own set of attributes, and they do not affect one another. Additionally,*

encapsulation *allows you to restrict direct access to attributes (or methods). This is done by prefixing the attribute name with a single underscore _ or_ double underscores.* ***Polymorphism*** *allows the use of the same method name in different classes with different implementations — this simplifies code and makes it easier to reuse.*

Keywords*: Attribute, method, object-oriented programming, object, accelerometer, sensor."*

Аннотация: *В языке программирования Python* ***атрибут*** *— это данные или функция, связанные с объектом. Атрибуты делятся на два типа:* ***классовые атрибуты*** *и* ***атрибуты экземпляра****.* ***Классовый атрибут*** *определяется на уровне класса и является общим для всех объектов. Он определяется непосредственно внутри класса, и все экземпляры класса разделяют этот атрибут.* ***Атрибут экземпляра****, с другой стороны, определяется в методе __init__ с использованием self и принадлежит только конкретному объекту. Каждый объект имеет свои собственные атрибуты, которые не влияют друг на друга. Кроме того,* ***инкапсуляция*** *позволяет ограничить прямой доступ к атрибутам (или методам). Это достигается с помощью префиксов в виде одного подчеркивания _ или_ двойного подчеркивания.* ***Полиморфизм*** *позволяет использовать одинаковые имена методов в разных классах с различными реализациями — это упрощает код и облегчает его повторное использование.*

Ключевые слова: *Атрибут, метод, объектно-ориентированное программирование, объект, акселерометр, сенсор.*

Biz ilgari OYD (obyektga yo‘naltirilgan dasturlash) kuchini klass va obyekt funksiyalari orqali, ya’ni ma’lumot va metodlarni birlashtirish orqali ko‘rgan edik. Bundan tashqari, OYDda yana uchta muhim tushuncha mavjud:

1. Vorislik – bu OYD kodini yanada modullashtiradi, qayta ishlatishni osonlashtiradi va klasslar o‘rtasida aloqa (bog‘liqlik) yaratadi.
2. Inkapsulyatsiya– bu klassning ayrim xususiy (shaxsiy) tafsilotlarini boshqa obyektlardan yashirish imkonini beradi.

3. Xususiyatlar – bu bir xil amalni turli xil usullarda bajarish imkonini beradi.

Quyidagi bo‘limda biz ushbu tushunchalarni qisqacha muhokama qilamiz.

Vorislik bizga boshqa klassdan barcha metodlar va atributlarni meros qilib oladigan klassni aniqlash imkonini beradi. Odatda, yangi klass — farzand klass (child class) deb, undan meros olinayotgan klass esa ota klass (parent class) yoki superklass deb ataladi.

Agar siz klass tuzilmasi haqidagi oldingi ta’rifga nazar tashlasangiz, quyidagi sintaksisni ko‘rasiz:

class ClassName(superclass)

Bu degani — yangi klass superklassdagi barcha atributlar va metodlarga ega bo‘ladi. Vorislik orqali farzand klass va ota klass o‘rtasida aloqa o‘rnatiladi. Odatda, ota klass umumiy turdagi, farzand klass esa maxsus turdagi bo‘ladi.

Quyidagi misolni ko‘rib chiqamiz:

```
class Sensor():
    def __init__(self, name, location, record_date):
        self.name = name
        self.location = location
        self.record_date = record_date
        self.data = {}

    def add_data(self, t, data):
        self.data['time'] = t
        self.data['data'] = data
        print(f'We have {len(data)} points saved')

    def clear_data(self):
        self.data = {}
        print('Data cleared!')
```

Endi bizda umumiy sensor ma’lumotlarini saqlash uchun klass mavjud, shu orqali ma’lumotlarni saqlash uchun sensor obyektini yaratishimiz mumkin.

Misol:Sensor obyektini yarating.

```
import numpy as np

sensor1 = Sensor('sensor1', 'Berkeley', '2019-01-01')
data = np.random.randint(-10, 10, 10)
sensor1.add_data(np.arange(10), data)
sensor1.data
```

```
We have 10 points saved
```

```
{'time': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'data': array([-4, -7,  2, -3, -8,  6,  4,  3,  5, -9])}
```

Faraz qilaylik, bizda boshqa turdagi sensor mavjud — akselerometr (tezlanishni o‘lchovchi sensor). U Sensor klassidagi atribut va metodlarga ega, lekin unga xos bo‘lgan qo‘shimcha atributlar yoki metodlar ham mavjud bo‘lishi mumkin. Bu yerda vorislik bizga yordam beradi va ishimizni osonlashtiradi. Yangi klass — Accelerometer — Sensor klassidan meros oladi, ya’ni u Sensor klassidagi barcha atribut va metodlarga ega bo‘ladi.

Keyin biz xohlasak:mavjud atribut va metodlarni kengaytirishimiz (extend) mumkin, yoki ularga yangi funksiyalar qo‘shishimiz (override yoki append**)** mumkin.

Endi keling, ushbu yangi klass — Accelerometer — ni yaratamiz va unga show_type nomli yangi metod qo‘shamiz, bu metod sensorning qanday turga mansubligini bildiradi.

```
class Accelerometer(Sensor):

    def show_type(self):
        print('I am an accelerometer!')

acc = Accelerometer('acc1', 'Oakland', '2019-02-01')
acc.show_type()
data = np.random.randint(-10, 10, 10)
acc.add_data(np.arange(10), data)
acc.data
```

```
I am an accelerometer!
We have 10 points saved
```

```
{'time': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'data': array([ -2,   2, -10,   6,   2,  -8,   2,   3,   7,  -6])}
```

Yangi Accelerometer klassini yaratish juda oddiy. Biz Sensor klassidan meros olamiz (u superklass deb yuritiladi), va yangi klass avtomatik tarzda superklassdagi barcha atributlar va metodlarga ega bo‘ladi.So‘ngra biz yangi metod — show_type ni qo‘shamiz. Bu metod Sensor klassida mavjud emas, ammo biz uni farzand klassga qo‘shib, funksiyani kengaytira olamiz. Bu — meros olishning kuchli tomonlarini ko‘rsatadi. Biz Sensor klassining katta qismini qayta ishlatdik, va unga yangi funksiyalar qo‘shdik. Bundan tashqari, meros olish real hayotdagi obyektlar modellashtirilishida mantiqiy bog‘lanishni o‘rnatadi. Sensor klass — bu umumiy (generik) klass, Accelerometer esa — undan xoslashtirilgan (aniqlashtirilgan) farzand klass bo‘lib, ota klassning barcha xususiyatlarini meros qilib oladi.

Agar biz ota klassdan (parent class) meros olgan bo‘lsak, u holda ota klassdagi mavjud metodni qayta yozishimiz (ya'ni, o‘zimizning versiyamizni berishimiz) mumkin.
Bu hodisa metodni qayta aniqlash yoki overriding deb ataladi.

Quyidagi misolni ko‘raylik:

Misol:UCBAcc nomli klass yarating — bu UC Berkeley tomonidan yaratilgan aniq turdagi akselerometr. Bu klass Accelerometer klassidan meros oladi, ammo show_type metodini yangilab (qayta yozib), u sensor nomini chop etadigan qilib o‘zgartiradi.

```
class UCBAcc(Accelerometer):

    def show_type(self):
        print(f'I am {self.name}, created at UC Berkeley!')

acc_ucb = UCBAcc('UCBAcc', 'Berkeley', '2019-03-01')
acc_ucb.show_type()
```

```
I am UCBAcc, created at UC Berkeley!
```

Biz ko‘rib turibmizki, yangi UCBAcc klassimiz aslida show_type metodini

yangi funksiyalar bilan yangilab (qayta yozib) ishlatmoqda. Ushbu misolda biz nafaqat ota klassimizdan xususiyatlarni meros olyapmiz, balki ba'zi metodlarni o‘zgartiryapmiz yoki takomillashtiryapmiz.

Endi esa Sensor klassidan meros oluvchi, lekin yangi atribut qo‘shish orqali atributlarni yangilaydigan NewSensor nomli klass yaratamiz. Bu holatda biz yangi atribut — brand (brend) ni qo‘shmoqchimiz. Albatta, bunday o‘zgartirish uchun biz __init__ metodini to‘liq qayta aniqlashimiz mumkin, bu orqali ota klassdagi funksiyani qayta yozib, yangilab chiqamiz (quyidagi misolda ko‘rsatilganidek).

```
class NewSensor(Sensor):
    def __init__(self, name, location, record_date, brand):
        self.name = name
        self.location = location
        self.record_date = record_date
        self.brand = brand
        self.data = {}

new_sensor = NewSensor('OK', 'SF', '2019-03-01', 'XYZ')
new_sensor.brand
```

```
'XYZ'
```

Biroq, bunga erishishning yanada qulayroq yo‘li mavjud**.** Biz bu yerda ota klassga to‘g‘ridan-to‘g‘ri murojaat qilmasdan, ya’ni nomini aniq yozmasdan ishlash uchun super() metodidan foydalanishimiz mumkin. Bu usul orqali kodimiz soddaroq, moslashuvchanroq va foydalanuvga qulayroq bo‘ladi. Keling, quyidagi misolda meros olingan klassda atributlarni qanday qayta aniqlash mumkinligini ko‘rib chiqamiz:

Misol:Meros olishda atributlarni qayta aniqlash.

```
class NewSensor(Sensor):
    def __init__(self, name, location, record_date, brand):
        super().__init__(name, location, record_date)
        self.brand = brand

new_sensor = NewSensor('OK', 'SF', '2019-03-01', 'XYZ')
new_sensor.brand
```

```
'XYZ'
```

Endi biz ko‘rib turibmizki, super() metodi yordamida barcha atributlarni qayta sanab o‘tishga hojat qolmaydi.

Bu yondashuv kodni kelajakda saqlash va boshqarishni osonlashtiradi, ya’ni

kengaytiriladigan va texnik xizmat ko‘rsatish uchun qulay bo‘ladi.

Ayniqsa bu usul ko‘p martalik meros olish (multiple inheritance) holatlarida juda foydali bo‘ladi, garchi bu kitob doirasida bu mavzu batafsil muhokama qilinmasa-da.

Inkapsulyatsiya — bu obyektga yo‘naltirilgan dasturlash (OYD) ning asosiy tushunchalaridan biridir.

U klass ichidagi metod va atributlarga to‘g‘ridan-to‘g‘ri kirishni cheklash g‘oyasini ifodalaydi. Bu orqali murakkab tafsilotlar foydalanuvchilardan yashiriladi, ma’lumotlarning tasodifan o‘zgartirib yuborilishining oldi olinadi.

Python’da xususiy (private) metod va atributlarni yaratish uchun oldiga pastki chiziq (_) yoki ikki pastki chiziq (__) qo‘yiladi:

_atribut — himoyalangan (protected)

__atribut — xususiy (private)

Misol:

```
class Sensor():
    def __init__(self, name, location):
        self.name = name
        self._location = location
        self.__version = '1.0'

    # a getter function
    def get_version(self):
        print(f'The sensor version is {self.__version}')

    # a setter function
    def set_version(self, version):
        self.__version = version
```

```
sensor1 = Sensor('Acc', 'Berkeley')
print(sensor1.name)
print(sensor1._location)
print(sensor1.__version)
```

```
Acc
Berkeley
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-ca9b481690ba> in <module>
      2 print(sensor1.name)
      3 print(sensor1._location)
----> 4 print(sensor1.__version)

AttributeError: 'Sensor' object has no attribute '__version'
```

Yuqoridagi misol inkapsulyatsiya qanday ishlashini ko‘rsatadi.

Bitta pastki chiziq (_) yordamida biz shaxsiy (private) o‘zgaruvchini belgiladik.

Bunday o‘zgaruvchilar to‘g‘ridan-to‘g‘ri murojaat qilinmasligi kerak. Biroq bu faqat odat (konvensiya) hisoblanadi — sizga bu o‘zgaruvchiga to‘g‘ridan-to‘g‘ri kirishni hech narsa taqiqlamaydi, ya’ni xohlasangiz baribir foydalanishingiz mumkin. Ikki pastki chiziq (__) bilan esa, misolda ko‘rganimizdek, __version atributiga to‘g‘ridan-to‘g‘ri kirish yoki uni o‘zgartirish mumkin emas. Bu holda haqiqiy yashirish (data hiding) sodir bo‘ladi. Shu sababli, ikki pastki chiziq bilan aniqlangan atributlarga kirish yoki ularni o‘zgartirish uchun biz maxsus funksiyalar — getter (oluvchi) va setter (o‘rnatuvchi) metodlardan foydalanishimiz kerak.

Quyidagi misolda bu qanday amalga oshirilishi ko‘rsatiladi.

```
sensor1.get_version()
```

```
The sensor version is 1.0
```

```
sensor1.set_version('2.0')
sensor1.get_version()
```

```
The sensor version is 2.0
```

Bitta va ikki pastki chiziq shuningdek xususiy metodlarga ham tatbiq etiladi. Ular xususiy atributlar bilan bir xil tarzda ishlaydi, shuning uchun biz bu haqda to‘xtalmaymiz.

Python’da **atributlar—** bu **obyektlarga bog‘langan xossalar** hisoblanadi. Ular **klass yoki klassning obyekti (instansiyasi)** ichida aniqlangan **o‘zgaruvchilar yoki metodlar** bo‘lishi mumkin. Obyektga yo‘naltirilgan dasturlash (OYD)da, **klass atributlari** va **instansiya atributlari** o‘rtasidagi farqni tushunish juda muhim.

Obyektga yo‘naltirilgan dasturlash (OYD)da **klass** — bu obyektlar yaratish uchun ishlatiladigan **shablon (qolip)** hisoblanadi. **Klass atributlari** esa — bu **klassning o‘ziga tegishli o‘zgaruvchilar** bo‘lib, **klassning barcha obyektlari (instansiyalari) o‘rtasida umumiy bo‘ladi**.

Instansiya atributi — bu **obyektga (ya'ni klassdan yaratilgan instansiyaga) tegishli o‘zgaruvchi**. Ular **klassdagi __**init**__ metodida** self **orqali aniqlanadi** va **faqat shu obyekt (instansiya) uchun mavjud** bo‘ladi.

Xususiyatlar — bu obyektga yo‘naltirilgan dasturlash (OYD) ning yana bir asosiy tushunchasidir va bu bir nechta shakl degan ma’noni anglatadi. Xususiyatlar bizga bir xil interfeysni turli asosiy shakllarda (masalan, ma’lumot turlari yoki klasslar) ishlatish imkonini beradi. Masalan, biz klasslar yoki farzand klasslar o‘rtasida bir xil nomdagi metodlarni ishlatishimiz mumkin. Avvalgi misolda show_type metodini UCBAcc klassida qayta yozganimiz (override)ni ko‘rdik. Ota klass (parent class) Accelerometer va farzand klass (child class) UCBAcc ikkala klassda ham show_type nomli metod mavjud, lekin ularning amalga oshirishlari (implementation) turlicha. Bu bitta nomdan turli xil shakllarda foydalanish va ularning turli holatlarda har xil ishlashiga imkon yaratish bizning kodni juda soddalashtiradi va murakkablikni kamaytiradi.

Biz xususiyatlarni yanada batafsilroq muhokama qilmaymiz, ammo agar qiziqsangiz, onlayn manbalarda yanada chuqurroq o‘rganib chiqishingizni tavsiya qilamiz.

Xulosa

Python’da obyektga yo‘naltirilgan dasturlash (OOP) asosiy tushunchalari — **klass va instansiya atributlari, inkapsulyatsiya** va **polimorfizm** orqali kodni modullashtirish, qayta ishlatish va himoyalash mumkin bo‘ladi. **Klass atributlari** barcha obyektlarga umumiy bo‘lsa, **instansiya atributlari** har bir obyektga alohida tegishli. **Inkapsulyatsiya** yordamida atribut va metodlarga bevosita kirishni cheklash orqali ma’lumotlar himoyalanadi. **Polimorfizm** esa bir xil metod nomidan turli klasslarda turlicha foydalanish imkonini beradi. Bu tushunchalar OYD asoslarini tashkil qilib, kodni soddalashtirish va unumdorlikni oshirishda muhim rol o‘ynaydi.

FOYDALANILGAN ADABIYOTLAR:

1. **Lutz, Mark.** *Learning Python* (5th Edition). O'Reilly Media, 2013.

2. Downey, Allen B. *Think Python: How to Think Like a Computer Scientist* (2nd Edition). O'Reilly Media, 2015.

3. Python Software Fo Tojimamatov, I., & Mirsiddiqova, M. (2025). BERILGANLAR BAZASIDA HAYOTIY SIKL. Модели и методы в современной науке, 4(6), 66-70.

4. undation. The Python Tutorial — *Official Python Documentation*

5. Sharma, Yashavant Kanetkar. *Let Us Python*. BPB Publications, 2019.

6. Tojimamatov, I., & Siddiqova, G. (2025). TRANZAKSIYALARNI TAQSIMLANGAN TARZDA QAYTA ISHLASH MODELLARI. Современные подходы и новые исследования в современной науке, 4(6), 30-35.

7. Нурмаматович, Т. И., & Рахила, А. (2025). НА ОСНОВЕ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ПОВЫШЕНИЕ УСТОЙЧИВОСТИ К ПОЛОМКАМ И АВАРИЯМ. YANGI O ‘ZBEKISTON, YANGI TADQIQOTLAR JURNALI, 2(8), 197-204.

8. Tojimamatov, I., & Ahmataliyeva, S. (2025). BERILGANLARNI MARKAZLASHGAN TARZDA BOSHQARISH TAMOYILLARI. Академические исследования в современной науке, 4(21), 59-64.

9. Tojimamatov, I., & Adxamova, C. (2025). AMALIY TIZIMLARDA BERILGANLAR BAZASINI BOSHQARISH TIZIMLARI O ‘RNI. Академические исследования в современной науке, 4(21), 77-82.

10. Tojimamatov, I., & Fazliddinov, X. (2025). BERILGANLAR BAZASI ADMINISTRATORI VA UNING XUSUSYATLAR. Академические исследования в современной науке, 4(21), 90-95.

11. Karimberdiyevich, O. M., Nurmamatovich, T. I., & Abdulaziz o‘g‘li, Y. M. (2024). BIG DATA SOHASIDAGI XALQARO LOYIHALAR. IZLANUVCHI, 1(1), 39-45.

Foydalanilgan saytlar:

1. https://www.bing.com/ck/a?!&&p=bc29a7b1887c0ee6c9f8e97c1faf5145da85297f8a094dd199f63a41531e4fddJmltdHM9MTc0Nzc4NTYwMA&ptn=3&ver=2&hsh=4&fclid=03393c23-bb05-6dd5-008b-2fe5ba036ce6&psq=ENCAPSULATION%2c+ATTRIBUTES%2c+PROPERTIES%2c+AND+INHERITANCE+IN+THE+PYTHON+PROGRAMMING+LANGUAGE&u=a1aHR0cHM6Ly9weXRob25udW1lcmljYWxtZXRob2RzLmJlcmtlbGV5LmVkdS9ub3RlYm9va3MvY2hhcHRlcjA3LjAzLUluaGVyaXRhbmNlLUVuY2Fwc3VsYXRpb24tYW5kLVBvbHltb3JwaGlzbS5odG1s&ntb=1

2. https://www.bing.com/ck/a?!&&p=c665753af5b0abd61a61e8a2d4349f557fb31031339a03e1e96ab30baba1e2a2JmltdHM9MTc0Nzc4NTYwMA&ptn=3&ver=2&hsh=4&fclid=03393c23-bb05-6dd5-008b-2fe5ba036ce6&psq=ENCAPSULATION%2c+ATTRIBUTES%2c+PROPERTIES%2c+AND+INHERITANCE+IN+THE+PYTHON+PROGRAMMING+LANGUAGE&u=a1aHR0cHM6Ly93d3cuZ2Vla3Nmb3JnZWVrcy5vcmcvZW5jYXBzdWxhdGlvbi1pbi1weXRob24v&ntb=1