DOI:10.30546/gdu.2023.24-31

UOT:004.3

OBYEKTYÖNÜMLÜ PROQRAMLAŞDIRMANIN ƏSASLARI. PYTHONDA SİNİFLƏR, OBYEKTLƏR, ONLARIN XASSƏLƏRİ VƏ METODLARI

Babayeva Şəkər Maarif qızı, Əsgərova Tamilla Əhməd qızı
Gəncə Dövlət Universiteti
shakarbabayeva@mail.ru,
asgerovatamilla998@mail.ru

Xülasə: *Obyekt yönümlü proqramlaşdırma (OOP) sinif və obyekt anlayışına əsaslanan proqram təminatının hazırlanması bir metodologiyadır, proqramın özü isə bir-biri ilə və xarici aləmlə qarşılıqlı əlaqədə olan obyektlər çoxluğu şəklində yaradılır. Hər obyekt hansısa sinifin nümayəndəsidir. Siniflər ierarxiya təşkil edir. Python proqramlaşdırma dili obyekt yönümlü proqramlaşdırmanın prisiplərinə uyğun yaradılıb. Pythonda hər nə var obyektlərdir: ədədlər, sətirlər, siyahılar, lüğətlər və s. Proqramçı özünün də verilənlər tipini (sinif) yarada bilər və bunda öz metodlarını təyin edə bilər. OOP proqram kodunun yazılması vaxtı onun təkrar istifadə edilməsini nəzərə alır. Cari məqalədə obyektyönümlü proqramlaşdırmanın prinsipləri (abstraksiya, inkapsulyasiya, varislik və polimorfizm) qısa izah edilmişdir. Məqalə Pythonda siniflərin və obyektlərin yaradılması, siniflərin atributları və metodları, obyektlərin atributları və metodları, metodların növlərinə və atributlara giriş rejimlərinə həsr edilmişdir və məqalədə bunlara aid misal nümunələri göstərilmişdir.*

Açar sözlər: OOP, varislik, inkapsulyasiya, polimorfizm, abstraksiya, sinif, obyekt, xassələr, metodlar.

Ключевые слова: ООП, наследование, инкапсуляция, полиморфизм, класс, объект, свойства, методы.

Key words: OOP, inheritance, encapsulation, polymorphism, abstraction, class, object, properties, methods

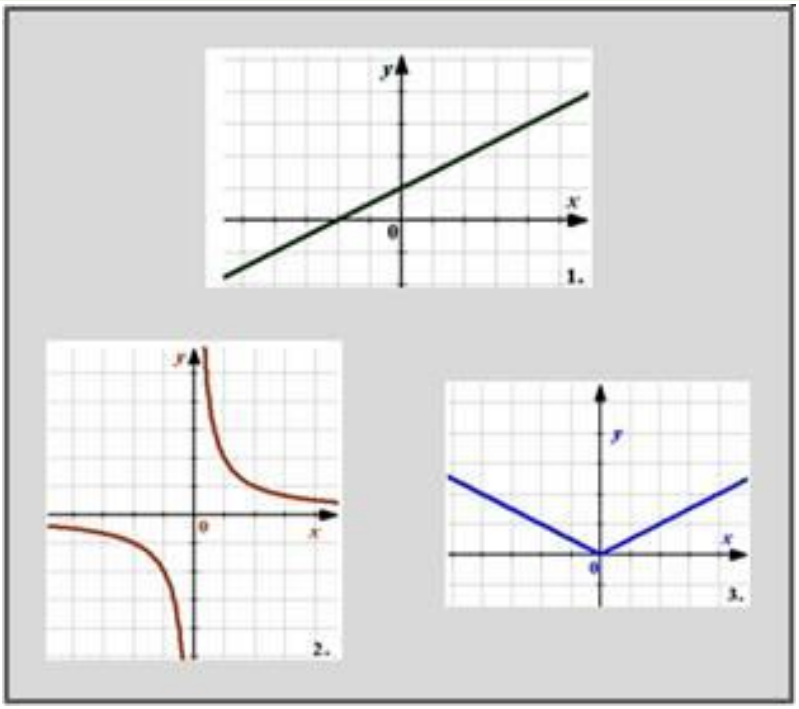
Hal-hazırda populyar OOP dilləri: C++, C#, Java, JavaScript, Ruby, PHP, Pythondur. 1990-cı illərin əvvəllərinə qədər, OOP dominant tendensiyaya çevrilənə və ən populyar (o dövrdə) C++ proqramlaşdırma dilinə daxil olana qədər proqramçılar OOP-dan istifadə etmədən işləyirdilər. Bəs OOP nədir və nəyə görə özünə hörmət edən hər bir təcrübəli proqramçı indi OOP bilməlidir? Demək olar ki, həmişə proqramlarda informasiya (verilənlər) obyektləri ilə işləyirik. Məsələn, tutaq ki, bizə bir pəncərədə bir neçə müstəqil funksiya qrafiklərini göstərmək lazımdır. Qrafikləri göstərmək və onlarla manipulyasiya etmək (köçürülmə, miqyasın dəyişilməsi və s.) üçün Graphs sinfini təyin etsək, onda hər bir xüsusi qrafik yalnız bu sinifin obyektinə çevriləcək:

DOI:10.30546/gdu.2023.24-31

Graphs

verilənlər

def draw()



Bunun sayəsində proqramımızda bir kod bloku proqram pəncərəsində eyni anda bir neçə qrafikin müstəqil nümayişinə cavabdeh olur.

Obyekt yönümlü proqramlaşdırma (OOP) sinif və obyekt anlayışına əsaslanan proqram təminatının hazırlanması bir metodologiyadır, proqramın özü isə bir-biri ilə və xarici aləmlə qarşılıqlı əlaqədə olan obyektlər çoxluğu şəklində yaradılır. Hər obyekt hansısa sinifin nümayəndəsidir. Siniflər ierarxiya təşkil edir.

Obyekt yönümlü proqramlaşdırmanın dörd əsas ideyası bunlardır:

- Abstraksiya (ing. abstraction)
- Miras (varislik, irsi mənimsəmə – ing. inheritance)
- Kapsülləşdirmə (inkapsulyasiya – ing. encapsulation)
- Polimorfizm (ing. polymorphism)

İnkapsulyasiya metodun həyata keçirilməsini, verilənləri və s. kənardan gizlətmək deməkdir. Məsələn, verilənləri (istehsalçı, həcm, saxlama kameraların sayı, enerji istehlakı və s.) və metodları (soyuducunu açın / bağlayın, yandırın / söndürün) ehtiva edən bir "soyuducu" sinfi təyin etmək olar. Amma eyni zamanda yandırma və söndürmənin necə baş verdiyinin həyata keçirilməsi sinfin istifadəçisinə məlum deyil. Bu da onun "soyuducu" sinfindən istifadə edən proqrama təsir edə biləcəyindən qorxmadan dəyişdirilməsinə imkan verir. Bununla sinif hazırlanmış proqram çərçivəsində verilənlərin yeni növü olur.

Varislik varis (uşaq - child) sinfi ilə əcdad (valideyn - parent) sinfi arasında "**is** (olur)" münasibətini nəzərdə tutur. Bununla varis sinfi əcdad sinfi ilə eyni atribut və metodları ehtiva edəcək, lakin yeni metodlar və atributlar əlavə etməklə genişləndirilə bilər (və genişləndirilməlidir).

Varisliyi nümayiş etdirən əsas sinif nümunəsi kimi atributlara (çəki, mühərrik gücü, yanacaq çəninin həcmi) və metodlara (işə salmaq və dayandırmaq) malik "avtomobil" sinfini təyin etmək olar. Belə sinifin varisi "yük maşını" ola bilər. O, "avtomobil" sinfi kimi eyni atributları və metodları və əlavə xüsusiyyətləri (oxların sayı, kompressor gücü və s.) ehtiva edəcək.

DOI:10.30546/gdu.2023.24-31

Polimorfizm, obyektin həyata keçirilməsindən (icrasından) asılı olmayaraq, eyni tipli interfeysə malik olan obyektləri eyni şəkildə işləməyə imkan verir. Məsələn, "yük maşını" sinfinin obyekti ilə siz "avtomobil" sinfinin obyekti ilə eyni əməliyyatları yerinə yetirə bilərsiniz, çünki birinci ikincinin varisidir, əksinə isə doğru deyil. Başqa sözlə, polimorfizm eyni adlı metodların müxtəlif reallaşdırılmasını əhatə edir (güman edir). Bu, varis sinfində ana sinifin metodlarını yeninən təyin edə bildiyiniz zaman mirasda çox faydalıdır.

Poly çoxsaylı, morf isə formanı bildirir. Buna görə də, bütövlükdə polimorfizm müxtəlif formaları olan nəyəsə aiddir. OOP-da polimorfizm eyni adlı, lakin fərqli davranışları (icrası) olan funksiyaları təsvir edir. Polimorfizm, mövcud olan obyektlərə eyni yanaşmağa imkan verir.

Abstraksiya real obyektlərin strukturunu yaratmaq üçün istifadə olunan obyekt yönümlü proqramlaşdırma anlayışıdır. O, yalnız ən vacib keyfiyyətləri "göstərir" və xarici dünyadan kənar məlumatları "gizlədir". Abstraksiyanın əsas məqsədi insanları lazımsız məlumatlardan qorumaqdır.

Obyekt yönümlü proqramlaşdırma (ing. OOP - Object Oriented Programing) əsas anlayışları obyektlər və siniflər olan proqramlaşdırma paradiqmasıdır. Sinif obyektlərin quruluşunu göstərən (şərh edən) tipdir. Sinif - əsasında yaradılan obyekt şablonudur. Hər bir obyekt müəyyən sinfin ekzemplyarıdır (nümayəndəsidir). Elan edilmiş sinif – obyektin yalnız şərhidir: onun üçün yaddaşda yer ayrılmır. Siniflər ierarxiya təşkil edir. Proqramın özü bir birilə və xarıcı aləmlə qarşılıqlı əlaqədə olan obyektlərin müəyyən çoxluği şəklində tərtib edilir. Python proqramlaşdırma dili obyekt yönümlü proqramlaşdırmanın prisiplərinə uyğun yaradılıb. Pythonda hər nə var obyektlərdir: ədədlər, sətirlər, siyahılar, lüğətlər və s. Ancaq Pythonun imkanları bunlarla məhdudlaşmır. Proqramçı ozünün də verilənlər tipini (sinif) yarada bilər və bunda öz metodlarını təyin edə bilər. Bu məcburi deyil, daxili (quraşdırılmış) siniflərdən və onların obyektlərindən istifadə edərək kifayətlənmək olar. Ancaq OOP-nın istifadəsi proqramın bir neçə proqramçı tərəfindən birlikdə uzun müddətli işlənib hazırlanması zamanı əlverişlidir, çünki proqramın başa düşülməsini sadələşdirir (birgə iş vaxtı proqramın oxunaqlı olması çox vacibdir). OOP proqram kodunun yazılması vaxtı onun təkrar istifadə edilməsini nəzərə alır. Belə yanaşma DRY (don't repeat yourself - "özünüzü təkrar eməyin") adlanır (o mənada ki, yazdığınız kodu təkrar istifadə edəcəksiniz, yenidən (təkrar) onu yazmağa vaxt itirməyin).

OOP-nı kompüter oyununun nümunəsindən istifadə edərək başa düşmək olar. Deyək ki, oyunda xassələrə (silah, sağlamlıq səviyyəsi, güc) malik olan oyun qəhrəmanı (iştirakçısı) - obyekti yaradılıb. Obyektin metodları var: gəhrəman gəzir, döyüşür, əşyalar yığır.

Pythonda sinifin yaradılması ***class*** instruksiyasından başlayır, sonra sinifə ad verilir (sinfin adı boyük hərflə başlamalıdır və sinfin mahiyyətini əks etdirməlidir.) və sinfin gövdəsində sinfin xassələri (atributları) və metodları göstərilir. Məsələn, minimal sinif (atributsuz və metodsuz) belə olacaq:

```
class Person:
    pass
```

Obyektin iki növ atributları var: xassələr (dəyişənlər) və davranış (metodlar).

Sinfin xassələri dəyişənlərlə təsvir edilir, onların qiymətləri mənisətmə əmri ilə təyin olunur. Metodlar – sinfin daxilində elan olunmuş fuksiyalardır. Onlar obyektin davranışını (hərəkətlərini) təyin edir. Metodlar adi funkiyalar kimi (***def*** konstruksiyası vasitəsilə) təsvir olunur. Məsələn:

```
class Person:
name='Sevda'
age=18
```

DOI:10.30546/gdu.2023.24-31

```
def greeting():
 print('Hello')
```

Sinfin bütün (istifadəçi) atributlarını ekranda görmək üçün xüsusi __dict__ metodundan istifadə edilir:

```
Person.__dict__
mappingproxy({'__module__': '__main__', 'age': 18, 'name': 'Sevda', 'greeting': <function Person.greeting at 0x00000227EDA883A0>, '__dict__': <attribute '__dict__' of 'Person' objects>, '__weakref__': <attribute '__weakref__' of 'Person' objects>, '__doc__': None})
```

Nəticədə sinfin atributları lüğət şəklində (yuxarıda gördüyünüz kimi) əks olunur.

Sinif obyektinin yaradılmasının sintaksisi belədir:

obyektin adı = sinfin adı()

Məsələn:

```
a=Person()
```

Analoji olaraq Person adlı sinfin ixtiyarı sayda obyektlərini (nümayəndələrini) yaratmaq olar:

```
b=Person()
Yoxlamaq olar ki,
type(a)
<class '__main__'.Person>
isinstance(a, Person) # isinstance () f-sı a obyetinin Person sinfinə daxil olmasını #yoxlayır
True
Type(a) == Person
True
```

Sinfin atributları onun bütün obyektləri üçün ümumi olur.

```
a.age
18
b.age
18
a.__dict__
{}
b.__dict__
{}
```

Sinfin atributlarını ekrana çıxarmaq və dəyişmək olar:

```
Person.name
'Sevda'
Person.age=20
Person.age
20
```

Sinfin atributunu ***getattr()*** daxili funksiyası ilə də ekrana çıxarmaq olar:

```
getattr(Person, 'age')
20
```

Sinifə yeni atribut əlavə etmək olar:

```
Person.gender='female'
Person.gender
'female'
```

DOI:10.30546/gdu.2023.24-31

Bu işi ***setattr()*** daxili funksiyası ilə də aparmaq olar:

```
setattr(Person, ‘gender’, ‘female’)
Person.gender
‘female’
```

Əgər *gender* adlı atribut sinifdə olsaydı, onda ancaq onun qiyməti dəyişiləcəkdi.

Sinfin atributunu del əmri ilə və ya ***delattr()*** daxili funksiyası ilə silmək olar:

```
del Person.gender  # Person sinfinin gender atributunu silir
delattr(Person, ’gender’)  # Person sinfinin gender atributunu silir
```

Əgər sinifdə müəyyən atribut, məsələn, x atributu yoxdur, onda onun silinməsi səhvə gətirəcək. Ona görə sinfin atributunu silməkdən əvvəl onun sinifdə olmasını ***hasattr()*** funksiyası ilə yoxlamaq lazımdir:

```
hasattr(Person, ‘gender’)
False
```

Obyetlərin atributları.

Sinfin atributları onun bütün obyektləri üçün ümumi olduğuna görə, əgər sinifə yeni atribut əlavə edilərsə, sinfin atributunun qiyməti dəyişilərsə və ya sinfin atributu silinərsə, onda bütün bu dəyişiklər sinifin bütün obyektlərində də baş verəcək. Ancaq əgər obyektin atributunun qiyməti dəyişilərsə,onda bu sinifin və onun digər obyektinin atributuna təsir etməyəcək, məsələn:

```
a.age=20
a.age
20
b.age
18
Person.age
18
a.__dict__
{'age': 20}
b.__dict__
{}
Person.__dict__
mappingproxy({'__module__': '__main__', 'age': 18, 'name': 'Sevda', 'greeting': <function Person.greeting at 0x00000227EDA883A0>, '__dict__': <attribute '__dict__' of 'Person' objects>, '__weakref__': <attribute '__weakref__' of 'Person' objects>, '__doc__': None})
```

Sinifin obyekti yaradılanda sinifin atributlarına istinad yaradılır və obyektin özünün adlar sahəsi boş olur. Əgər obyektin öz atributu yaradılır, onda bu obyektin özünün adlar sahəsi yaradılır və bu obyektin yeni atributuna müraciət olunursa, o, əvvəl obyektin adlar sahəsində axtarılır, orda olmasa, sonra sinifin adlar sahəsində axtarılır. Bu alt proqramların lokal və qlobal dəyişənləri ilə işə bənzəyir.

__doc__ dəyişənində sinifin təsvirinin daxilindəki birinci sətri (dırnaq arasında yazılmış sinifin qısa təsviri) ekranda göstərilir.

```
Person.__doc__
```

Sinifin hər bir yeni ekzemplyarı (nümayəndəsi) __init__() istifadə edərək işə salınır. __init__() istənilən sayda parametr qəbul edə bilsə də, self həmişə birinci parametrdir. Sinifin təsvirində birinci arqument ***self*** mütləqdir, çağırılan metodun aid oluğu obyektə olan ümumi qəbul edilmiş istinaddır. Bu parametr sinifin metodunu adi funksiyadan fərqləndirir.

DOI:10.30546/gdu.2023.24-31

self – cari obyektə (sinifin ekzemplyarına) istinaddır. C++, Java kimi proqramlaşdırma dillərində bunun analoqu ***this*** açar sözüdür. ***self*** parametri vasitəsilə sinifin daxilindəki atributlara (dəyişənlərə və metodlara) müraciət etmək olur. Aşağıdakı kodda area metodu düzbucaqlının sahəsini hesablamaq üçün width və height atributlarını ***self*** vasitəsilə istifadə edə bilir:

```
class Rectangle:
    def __init__(self, width , height):
        self.width = width
        self.height = height
        def area(self):
        return self.width*self. height
```

Metodlar 3 növ olur: statik, dinamik və sinif səviyəsində (eləcə, metod deyilir). Statik metodu ***@staticmethod*** dekoratoru ilə, sinif metodu ***@classmethod*** dekoratoru ilə yaradılır. Adi metod dekoratorsuz yaradılır, onun birinci arqumenti, bildiyimiz kimi, ***self*** olur. Birinci arqument kimi sinif metoduna ***cls*** yazılır. Sinifin metodu sinifin obyektinə yox sinifə bağlı olur və statik metod kimi sinifin ekzemplyarının yaradılmasını tələb etmir. Statik metodun sinif metodundan fərqi ondadır ki, statik metod sinif haqqında məlumatı olmur və ancaq parametrlə iş aparır. Sinif metodu isə siniflə işləyir, çünkü onun parametri sinfin özü olur. Sinfin metodu sinfin özü ilə də və sinfin obyekti ilə də çağırila bilər.

```
Class MyClass:
@staticmethod
def ex_static_method():
 print('static_method')

@classmethod
def ex_class_method(cls):
    print('class_method')

def ex_method(self):
    print('method')
```

Static və sinif metodlarına sinifin obtektini yaratmamaqla da müraciət etmək olar, adi metodu cağırmaq üçün obyektin yaradılması mütləqdir.

```
MyClass. ex_static_method()
static method
MyClass. ex_class_method()
class method
MyClass. ex_method()
        Traceback (most recent call last):
File "C:/Users/Shakar/AppData/Local/Programs/Python/Python310/Новая папка/class2.py", line 16, in <module>
MyClass.ex_method()
        TypeError: MyClass.ex_method() missing 1 required positional argument: 'self'
m=MyClass()
m. ex_method()
method
```

Atributlara giriş rejimləri.

_atribut (bir alt xətt ilə) – qorunan giriş rejimi (sinif daxilində və onun bütün varis siniflərində istifadə etmək üçün təyin olunur).

DOI:10.30546/gdu.2023.24-31

__atribut (iki alt xətt ilə) – şəxsi (özəl) giriş rejimi (yalnız sinif daxilində istifadə etmək üçün təyin olunur).

```
class A:
    def _private(self):
        print("Bu özəl metoddur!")
>>> a = A()
>>> a._private()
Bu özəl metoddur!
```

Atribut adının əvvəlindəki qoşa alt xətt daha çox qorunma təmin edir: atribut bu adla əlçatmaz olur.

```
>>> class B:
...     def __private(self):
...         print("Bu özəl metoddur!")
...
>>> b = B()
>>> b.__private()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'B' object has no attribute '__private'
```

Ədəbiyyat

1.K.Tahiroğlu “Python ilə proqramlaşdırma” Bakı-2016.
2.Дэвид Бизли “Python. Подробный справочник”.4-ое издание, СПб Символ-Плюс, 2017.
3.Д. Ю. Федоров “Основы программирования на примере языка PYTHON”. .Учебное пособие, Санкт-Петербург, 2019.
4.A Byte of Python (Russian) Версия 2.02 Swaroop C.H. (Перевод: Владимир Смоляр), 2020.
5.Эрик Мэтиз «Изучаем Python», «Питер», Санкт-Петербург-2017.
6.Задорожный С.С., Фадеев Е. П. “Объектно-ориентированное программирование на языке Python” (учебно-методическое пособие), Москва, МГУ, 2022.
7.https://ru.wikipedia.org/wiki/Python
8.https://pythonru.com/primery/primery-raboty-s-klassami-v-python
9.https://docs-python.ru/tutorial/klassy-jazyke-python/
10.https://hashdork.com/az/object-oriented-programming-in-python-for-beginners/
11.https://proproprogs.ru/python_oop/koncepciya-oop-prostymi-slovami
12.https://smartiqa.ru/courses/python/lesson-6
13.https://younglinux.info/oopython/oop

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ. КЛАССЫ, ОБЪЕКТЫ И ИХ СВОЙСТВА И МЕТОДЫ В PYTHON

Бабаева Шакяр Маарифовна, Аскерова Тамилла Ахмедовна

Резюме

Объектно-ориентированное программирование (ООП) является методологией разработки программного обеспечения, в основе которой лежит понятие класса и объекта, при этом сама программа создается как некоторая совокупность объектов, которые взаимодействую друг с другом и с внешним миром. Каждый объект является экземпляром некоторого класса. Классы образуют иерархии. Язык программирования

DOI:10.30546/gdu.2023.24-31

Python создан в соответсвии с принципами объектно-ориентированного программирования. Все в Python является объектом: числа, строки, списки, словари и т. д. Программист может создать свой тип данных (класс) и определить в нем свои методы. ООП учитывает повторное использование кода при его написании. В настоящей статье кратко изложены основные принципы ООП (абстракция, инкапсуляция, наследование и полиморфизм). Статья посвящена созданию в Python классов, объектов и методов классов, атрибутов и методов объектов, видам методов и режимам доступа к атрибутам. В статье приведены образцы примеров к ним.

FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING. CLASSES, OBJECTS AND THEIR PROPERTIES AND METHODS IN PYTHON.

Babayeva Shakar Maarif, Asgerova Tamilla Ahmed

Summary

Object-oriented programming (OOP) is a software development methodology based on the concept of a class and an object, while the program itself is created as a set of objects that interact with each other and with the outside world. Every object is an instance of some class. Classes form hierarchies. The Python programming language was created in accordance with the principles of object-oriented programming. Everything in Python is an object: numbers, strings, lists, dictionaries, etc. A programmer can create his own data type (class) and define his methods in it. OOP takes code reuse into account when writing it. This article summarizes the basic principles of OOP (abstraction, encapsulation, inheritance and polymorphism). The article is devoted to the creation in Python of classes, objects and methods of classes, attributes and methods of objects, types of methods and access modes to attributes. The article provides sample examples for them.

Rəyçi: Aslanov Ə.Ə.
İnformatika və cəbr kafedrasının 19 May 2023-cü il tarixli iclasın 09 saylı protokol
Daxil olma tarixi: 26 may 2023-cü il