УДК 004.43

PYTHON ТИЛИНДЕГИ ОБЬЕКТКЕ-БАГЫТТАЛГАН ПРОГРАММАЛОО

Пирматов Абдыманап Зияйдинович,
ф.-м.и.к., доцент, pirmatov@mail.ru,
Камалов Султанбек Садырбекович,
окутуучу, sultan.kamalov.96@mail.ru,
Абдукадыр кызы Айнагул - улук окутуучу,
abdukadyrkyzy2014@mail.ru
Б.Осмонов атындагы ЖАМУ,
Сүйөркул кызы Нурайым,
магистрант, suiorkulova2201@gmail.ru
ОшМУ, Кыргыз Республикасы,

Аннотация: *Бул макалада азыркы учурда эң актуалдуу болуп жаткан Python программалоо тилинин объектке багытталган программалоо стили каралып, ал жөнүндө толук маалыматтар берилди. Объектке багытталган программалоонун негизин түзгөн-* ***класстартарга*** *жана андагы атрибуттар менен методдорго кеңири токтолуп, атрибуттар жана методдор менен иштөөчү функциялар каралып, мисалдар менен коштолду. Класс атрибуту жана методдору менен класс нускасынын атрибуттары жана методдорунун айырмачылыктары каралды, мисалдар келтирилди. Объектке багытталган программалоону адистештирилген тилдердин өнүгүшү үчүн колдонууга болгондуктан, алар, адатта, үч негизги өзгөчөлүктөр: мурастоо, полиморфизм жана инкапсуляция менен мүнөздөлөт. Бул ар биринеөзгөчөлүктөргө токтолуп, мисалдар келтирилип, программалык коддо көрсөтүлдү.*

Ачкыч сөздөр: *объектке багытталган программалоо, класстар, атритуттар, функциялар, класстар обьектиси, класстар нускасы.*

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON

Пирматов Абдыманап Зияйдинович,
к.ф.-м.н., доцент, pirmatov@mail.ru
Камалов Султанбек Садырбекович,
преподаватель, sultan.kamalov.96@mail.ru,
Абдукадыр кызы Айнагул – ст. преподаватель,
abdukadyrkyzy2014@mail.ru
ЖАГУ им.Б.Осмонова,
Сүйөркул кызы Нурайым, магистрант,
suiorkulova2201@gmail.ru
ОшГУ, Кыргызская Республика,

Аннотация: *В этой работе изучены наиболее актуальный в настоящее время - объектно-ориентированная программирования на языке Python и дается подробная информация о нем. Основой объектно - ориентированного программирования является* ***классы,*** *включающий набор переменных и функций для управления значениями, хранящимися в этих переменных. Переменные называют атрибутами, а функции — методами. Класс является фабрикой объектов, т. е. позволяет создать неограниченное количество экземпляров, основанных на этом классе. В нем подробно рассматриваются функции, которые работают с атрибутами и методами, а также рассматриваются и приводятся примеры, которые составляют основу объектно - ориентированного программирования-классов и содержащихся в них атрибутов и методов. Были рассмотрены различия между атрибутами и методами класса объекта с атрибутами и методами экземпляра класса, приведены примеры. Поскольку объектно-ориентированное*

программирование используется для развития специализированных языков, они обычно характеризуются тремя основными характеристиками: наследование, полиморфизм и инкапсуляция. Для каждого приводятся примеры и показаны в программном коде.

Ключевые слова: *объектно-ориентированное программирование, классы, атрибуты, функции, объекты, экземпляры классов.*

OBJECT-ORIENTED PROGRAMMING IN PYTHON

Pirmatov Abdymanap Ziyaydinovich,
associate professor, pirmatov@mail.ru
Kamalov Sultanbek Sadyrbekovich,
teacher, sultan.kamalov.96@mail.ru,
Abdikadyr kyzy Ainagul, senior lecturer,
abdukadyrkyzy2014@mail.ru
JASU named after B.Osmonov
Suiorkul kyzy Nuraiym, master's degree student,
suiorkulova2201@gmail.ru, OshSU, Kyrgyzstan,

Annotation: *In this work, the most relevant at the present time - object-oriented programming in Python and provides detailed information about it. The basis of object-oriented programming is classes, which include a set of variables and functions for managing the values stored in these variables. Variables are called attributes, and functions are called methods. The class is an object factory, i.e. it allows you to create an unlimited number of instances based on this class. It details the functions that work with attributes and methods, as well as discusses and provides examples that form the basis of object-oriented programming classes and the attributes and methods they contain. Differences between attributes and methods of an object class with attributes and methods of a class instance were considered, examples were given. Since object-oriented programming is used to develop specialized languages, they are usually characterized by three main characteristics: inheritance, polymorphism, and encapsulation. For each, examples are given and shown in the program code.*

Keywords: *object-oriented programming, classes, attributes, functions, objects, class instances.*

Объектке багытталган программалоо техникасы же методологиясы -1960-жылдары иштелип чыгып - бул татаал программаларды өзгөртүүгө арналып түзүлгөн.

Объектке багытталган программалоо– бул бир эле программалык кодду бир жана бир нече жолу кайра-кайра колдонула тургандай кылып программаны уюштуруунун жолу. Функциялар жана модулдардан айырмаланып, объектке багытталган программалоо - программаны фрагменттерге бөлүүгө гана эмес, ошондой эле реалдуу дүйнөнүн буюмдарын - объекттер түрүндө сүрөттөөгө, ошондой эле бул объектилердин ортосунда байланыштарды уюштурууга мүмкүндүк берет.

Ал аралык программа аркылуу, башкача айтканда, трансляция аркылуу аткарылат. Бул жарым чечмеленген тил болгондуктан, ийкемдүү жана портативдүү. Ал динамикалык түрдө терилет, башкача айтканда, маалыматтын түрү - иштөө убагында аныкталат.

Өзгөрмөлөргө, функцияларга түздөн-түз жана башка функциялар аркылуу кайрылуу методдору колдонулду.

Объектке багытталган программалоонун негизги бөлүгүн *класстар* түзөт. Класс – бул маанилерди башкаруудагы өзгөргөмөлөрдүн жана функциялардын топтомун камтыган, татаал берилгендердин тиби болуп саналат. Өзгөргөмөлөр - *атрибуттар*, ал эми функциялар *методдор* деп аталат. Класс – ушул класска таандык болгон чексиз сандагы объекттердин нускасын түзө алат [1].

Класс. Объектке багытталган программалоодо, класс - объект үчүн план катары иштейт. Классты үйдүн картасы катары кароого болот. Биз анын картасын карап, үйдү кандай курууга боло тургандыгын гана биле алабыз.

Класс менен обьектин ортосундагы катышты, Ауди жана машина деген мисалда карайлы. Мында, Ауди – бул машина. Ал эми машина - бул жалпы абстрактуу түшүнүк, ага Мерседес, Тойота, Хонда ж.б. да кирет.

Класс - *class* өзөктүү сөзү менен төмөндөгүдөй түрдө берилет:

class <Класстын аты> [(<Класс1>[, ..., <КлассN>])]:

[" " " Документтештирүү жолчосу " " "]

< Атрибуттарды жана методдорду аныктоо>

Бул инструкция боюнча жаңы объект түзүлөт жана *class* өзөктүү сөзүнөн кийинки идентификаторго шилтемени ыйгарат, ошондуктан класстын аты өзгөрмөлөрдү атоо эрежелерине толук жооп берүүсү зарыл. Эгерде базалык класстар берилсе, аларды класстын атынан кийин каашалардын ичинде үтүр менен, бир же бир нече базалык класстар түрүндө көрсөтүүгө болот.

Мисал катары, жөн гана кабар чыгарган, класстын мисалын түзөлү:

```
# coding: utf-8
class MyClass:
""" Бул документ үчүн жолчо """
print("Инструкция аткарылат")
input()
```

Бул мисал - *MyClass* классын гана аныктайт, бирок класстын нускасын түзбөйт.

Класстын *атрибутун* түзүү кадимки өзгөрмөнү түзүүгө окшош. Класстын ичиндеги *метод*, кадимки функцияны түзүүдөй эле, *def* инструкциясы аркылуу түзүлөт.

Класстын методундагы биринчи параметр анык берилет, ошондой эле автоматтык түрдө шилтемеси класстын нускасына берилет жана ал параметрди *self* деген ат менен аталат. Класстар методдорундагы *x* атрибутуна *self* өзгөрмөсү аркылуу төмөндөгүдөй аныктайбыз:

s*elf.x*

Класстын методдорун жана атрибуттарын колдонуу үчүн, төмөндөгүдөй синтаксис менен класстын нускасын түзүү зарыл:

< Класстын нускасы > = <Класстын аты> ([<Параметрлер>])

Класстар методуна кайрылуу үчүн төмөндөгүдөй формат колдонулат:

< Класстын нускасы > .<Методдун аты> ([<Параметрлер>])

Класстын атрибуттарына кайрылуу төмөндөгүдөй аткарылат:

< Класстын нускасы >.<Атрибуттун аты>

Myclass классын - *x* атрибуту жана ушул атрибуттун маанисин чыгаруучу *print_x()* методу менен аныктайлы, андан соң класс нускасын түзөбүз жана методду чакырабыз:

```
class MyClass:
def __init__(self) :     # Конструктор
self.x = 10                  # Класс нускасынын атрибуту
def print_x(self):           # self — Класс нускасынына шилтеме
print(self.x)                # Атрибуттун маанисин чыгарабыз
c = MyClass()            # Класс нускасын түзүү
                             # print_x() методун чакырабыз
c.print_x()                  # Методду чакырууда self көрсөтүлбөйт

print(c.x)                   # Атрибутка кайрылууга болот
```

Атрибуттар жана методдор менен иштөө үчүн төмөндөгүдөй функциялар колдонулат:

♦ *getatt()* — атрибуттун жолчолук типтеги маанисин, анын аты боюнча кайтарат.
Бул функциянын жардамында, программа аткарылып жатканда атрибуттун аты динамикалык түрдө аныкталат. Функциянын форматы:

- *getattr(<Объект>, <Атрибут>[, <Значение по умолчанию>])*

Эгерде берилген атрибут табылбаса, *AttributeError* каталыгы чыгат. Бул ката чыкпоо үчүн, үчүнчү параметрде маанини көрсөтүп коюу керек;

- *setattr()* — атрибуттун маанисин берет.Атрибуттун аты жолчолук типте көрсөтүлөт. Функциянын форматы:
  setattr(<Объект>, <Атрибут>, <Мааниси>)

setattr() методунун экинчи параметри менен башка атрибуттун атын берүүгө болот – бул учурда атрибут берилген ат менен түзулөт;

- *delattr (<Объект>, <Атрибут>)* — көрсөтүлгөн атрибутту өчүрот. Атрибуттун аты жолчолук типте көрсөтүлөт;
- *hasattr (<Объект>, <Атрибут>)* — Берилген атрибут бар экендигин тектерет, эгер атрибут бар болсо, функция *True* маанисин кайтарат.

```
class MyClass:
def __init__(self) :
self.x = 10
def get_x(self):
return self.x
c = MyClass{)                # Класстын нускасы
print(getattr(c, "x")        # 10 чыгат
print(getattr(c, "get_x")())  # 10 чыгат
print(getattr(c, "y", 0))    # 0 чыгат, себеби атрибут табылбады
setattr(c, "y",20)           # y атрибутун түзөбүз
print(getattr(c, "y", 0))    # 20 чыгат
delattr(c, "y")              # y атрибутун өчүзөбүз
print(getattr(c, "y", 0))    # 0 чыгат, себеби атрибут табылбады
print(hasattr(c, "x") )      # True
print(hasattr(c, "y"))       # False
```

Python тилинде бардык класстын атрибуттары ачык (public) болуп саналат, б.а. атрибуттарды класстын өзүндө, ошондой эле башка класстарда жана негизги программанын кодунда түздөн түз өзгөртүүгө мүмкүн. Класстар түзүлгөндөн кийин атрибуттар динамикалык түрдө түзүлөт, алар класстын обьекттинин атрибуту жана класстын нускасынын атрибуту катары түзүлөт.

Мисалда карайлы:

```
class MyClass:                    #Бош класс аныктайбыз
pass
MyClass.x = 50                    #Класс обьекттин атрибутун аныктайбыз
c1, c2 = MyClass(), MyClass()     # Эки класс нускасын түзөбүз
c1.y = 1                          #Класс нускасынын атрибутун түзөбүз
c2.y = 20                         # Класс нускасынын атрибутун түзөбүз
print(c1.x, c1.y)                 # 50,10 чыгат
print(c2.x, c2.y)                 # 50,20 чыгат
```

Бул мисалда, *pass* операторун жайгаштырып, бош класс аныктадык. Андан кийин бардык класс нускаларга иштөөгө мүмкүн болгон, класс обьектинин атрибутун *(x)* түздүк. Мындан соң эки класс нускасын түзүп жана аларга бир түрдүү *(y)* атрибуттарын коштук. Алардын маанилери ар бир класс нускасы үчүн ар түрдүү бердик.

Эң негизгиси класс обьектисинин атрибуту менен класс нускасынын атрибуттарынын айырмасын түшүнүү зарыл.

Ал үчүн класс обьектинин *(x)* атрибуту менен жана класс нускасынын *(y)* атрибуту менен класс түзүүнү, төмөнкү мисалда карайлы:

```
class MyClass:
        x = 10                          # Класса объектинин атрибуту
        def _init__(self):
                self.y = 20             # Класс нускасынын атрибуту
```

Эми бул класс үчүн эки нуска түзөбүз:

```
c1 = MyClass()                          # Класс нускасын түзөбүз
c2 = MyClass()                          # Класс нускасын түзөбүз
```

x атрибутунун маанисин чыгарып, андан соң өзгөртүп дагы чыгарабыз:

```
print(c1.x, c2.x)                       # 10 10

MyClass.x = 55                          # Класс  объектинин атрибутун өзгөртөбүз
print(c1.x, c2.x)                       # 55 55
```

Эми *y* атрибутунун маанисин чыгарып, андан соң өзгөртүп дагы чыгарабыз:

```
print(c1.y, c2.y)                       # 20 20
c1.y = 55                               # Класс  нускасынын атрибутун өзгөртөбүз
print(c1.y, c2.y)                       # 55 20
```

Объектке багытталган программалоону адистештирилген тилдердин өнүгүшү үчүн да колдонууга болот. Алар, адатта, үч негизги өзгөчөлүктөр менен мүнөздөлөт [2]:

1. Мурастоо - иерархия боюнча жаңы объект түзүү, ошондой эле мүмкүндүк алуу кодун берүү жана бардык объекттер жөнүндө маалыматтарды түзүү үчүн объектти аныктоо жөндөмү менен жаратылган объектилердин иерархияны куруу үчүн, анын андан аркы пайдалануюга берүү. Мурастоо объектке багытталган программалоонун негизги түшүнүктөрүнөн болуп саналат. Ал класс түзүлгөн жаңы класска бардык касиеттерин мурастап берет. Мисал катары, төмөнкү мисалды карайлы:

```
class Class1:                           # Базалык класс
        def func1 (self):
                print ("Класс Class1дин  fund1 () методу ")
        def func2(self):
                print("Класс Class1дин  fund2 () методу "!
class Class2(Class1):                   # Класс Class2 класс Class1 ди мурастайт
        def func3(self):
                print("Класс Class2дин  fund3 () методу ")
c = Class2()                            # Класс Class2нин нускасын түзөбүз
c.fund1()                               # Класс Classl  ден funcl()методун чыгарат
c.func2()                               # Класс Classl  ден func2()методун чыгарат
c.func3()                               # Класс Class2  ден func3()методун чыгарат
```

2. Полиморфизм – бул обьектин программада өзүн – өзү ар түрдүү формада алып жүрүүсүн тушундүрөт, б.а. программалоодо полиморфизм – кайра жүктөө же кайра аныктоо методдору менен ишке ашат.

Кайра жүктөө методу параметрлердин санына же тибине карата ар түрдүү аткарылуучу метод болуп саналат. Мисалда карап көрөлү:

```
class Class:
  def  start(self, a, b=None):
```

```
    if b is not None:
        print (a + b)
    else:
        print (a)
cl_a=Class()                # Класс Classынын нускасын түзөбүз
cl_a.start(10)              # start () методун бир параметр менен чыгарат
cl_a.start(10,20)                   # start () методун эки параметр менен чыгарат
```

3. Инкапсуляция – бул берилгендерди көзөмөлдөп колдонуу. Объектке багытталган программалоодо - бир класс башка класстын берилгендерин түздөн-түз колдонууга болбойт, ошондуктан класстар методу аркылуу козөмөлдөнөт.

Python 3 тилинде берилгендерди 3 деңгээлде колдонууга уруксат берилет [3]:

- жалпы (public);
- корголгон (protected, аты бир төмөнкү сызык менен башталат, _protectedName);
- жекече (private, аты эки төмөнкү сызык менен башталат, __protectedName).

Бул макалада Python тилиндеги обьектке багытталган программалоо стилинин аныктамасын, берилүү жолдорун, алар менен иштөөчү функцияларды жана негизги өзгөчөлүктөр карап чыгылды жана мисалдарда көрсөттүлдү. Макала кафедранын семинарында талкууланган.

Корутунду. Бул макалада каралган материалдарды студенттерге жана окуучуларга Python 3 программалоо тилинде сабак өтүүдө бышыктоо үчүн колдонууга болот.

Программалык жабдылышы: программалар коддор Python 3 платформасында компьютерде жана уюлдук телефондо [4,5] текшерилди.

Адабияттар:

1. Лутц М. Изучаем Python, 5-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2019. – 1280 с
2. Марк Саммерфилд. Программирование на Python 3. Подробное руководство. – Пер. с англ. – СПб.: Символ Плюс, 2009 – 608 с., ил.
3. Dusty Phillips. Python 3 Object-oriented Programming. Second Edition. Packt Publishing, Birmingham – Mumbai, 2015, 460 p.
4. Пирматов А. З., Идрисова Г.И. Работа с языком программирования Python в операционной системе Андроид, /Вестник ЖАГУ. №1 Жалал-Абад, 2021. –С.69-74
5. Интернет ресурс: https://www.python.org/ (акыркы кайрылган күнү: 15.10.2022)