FERGANA BRANCH OF THE TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES NAMED AFTER MUHAMMAD AL-KHORAZMI
ISSN 1102-3321 «AL-FARG'ONIY AVLODLARI» ELECTRONIC SCIENTIFIC JOURNAL № 1 (1) | 2023

PUBLIC, PROTECTED, PRIVATE MEMBERS IN PYTHON

https://doi.org/10.5281/zenodo.7716867

Musayev Khurshid,
assistant of Fergana branch of Tashkent University of Information Technologies named after Muhammad al-Khorazmi

Soliev Bakhromjon,
assistant of Fergana branch of Tashkent University of Information Technologies named after Muhammad al-Khorazmi

Abstract: This article focuses on the public, protected, and private view members in the Python programming language and provides information and examples of how to manage view rights in the Python programming language without using these keywords.

Keywords: Python, programming, encapsulation, polymorphism, inheritance, public, private, protected, class, self, property.

Introduction. In Python, class members can be classified into three categories: public, protected, and private. These access modifiers determine the visibility and accessibility of the class members to other parts of the program.

Public members are accessible from anywhere in the program, both inside and outside the class. They can be accessed using the dot operator after creating an instance of the class.

Protected members are not accessible from outside the class, but they can be accessed from within the class and its subclasses. In Python, this is achieved by adding a single underscore before the name of the member.

Private members are not accessible from outside the class or from its subclasses. In Python, this is achieved by adding two underscores before the name of the member.

Understanding the difference between public, protected, and private members is important in object-oriented programming, as it allows for better control over the visibility and accessibility of class members. This can help in improving the overall design and security of a program.

Literature review and methodology. Python is a dynamically typed programming language [1, 2]. Python is an object-oriented programming language. But unlike many object-oriented languages, in Python, the scope of an object's use is determined by its current set of methods and fields, as opposed to inheriting from a specific class. The approach used in Python is called "duck typing". The name of the term comes from the English "duck test" ("duck test"), which in the original sounds like "If it looks like a duck, swims like a duck and quacks like a duck, then it probably is a duck" ("If it looks like a duck, swims like a duck, and quacks like a duck, then it's probably a duck") [3]. In Listing 1, the parse_node() function takes several parameters; Python does not specify the types of the parameters. Referring to parameter methods in the function body will be correct if the objects passed as parameters have these methods. This means that if the visitor object class has the prepare() and visit() methods defined (it "behaves like a duck"), then accessing them will succeed (it is considered that "it is a duck"). If, for example, the visit() method is not defined for the object class, then the interpreter will generate an exception AttributeError: A instance has no attribute 'visit' if visitor is an object of class A.[4]

Many practical problems of our time in economics, architecture, electronics and other fields of science at a certain stage of development require the solution of complex problems consisting of many interrelated parts. To solve them, a systematic approach is used, which allows you to get a single answer that takes into account all subtasks. But when the question concerns a formalized mathematical version of the problem of optimizing a solution for many factors and criteria at the same time, then the number of relatively general methods that give a single answer is very small. By constructing hypersurfaces corresponding to the output data of the experiment, their sections by level hyperplanes and their intersections, the optimization area is reduced to the minimum possible dimension for subsequent selection among equivalent points by the user.

To implement the algorithm, the Python programming language was chosen, since it has wide capabilities for working with data arrays, is easy to learn and portable to any devices and systems. The software library includes two working classes: the main class is used for the optimization operation and working with database tables; auxiliary contains three basic groups of methods for processing sets of points and curves on a plane. The optimization methods of the main class of the library allow two main operations to be carried out: a section by a level plane and a search for the intersection of two hypersurfaces in spaces with the same dimensions. In the first case, one database table and a given height value for the level plane are used. On the basis of a complete enumeration of the values of the table, the contour curves of the hypersurface are constructed in all two-dimensional planes, except for those parallel to the section, which are subspaces of the workspace and fill the regular grid of table values. For each curve, the points of intersection with the projection of the section plane are found and stored in a new table representing the result of the section in the new dimension space one less than the original one. The intersection of two surfaces, unlike the first method, requires working with two source tables in the same dimensions. The resulting table is filled with the values of the intersection points of the contour curves of two hypersurfaces and has the dimension equal to the original one.[5]

Encapsulation is one of the fundamental principles of object-oriented programming (OOP). It refers to the practice of hiding the implementation details of an object from the outside world, and instead providing a public interface through which other code can interact with the object.

In Python, encapsulation can be achieved through the use of access modifiers, which control the visibility of class attributes and methods. There are two main types of access modifiers in Python:

Public: attributes and methods that are accessible from outside the class. These are not prefixed with any special characters, and can be accessed using the dot notation.

Private: attributes and methods that are only accessible from within the class itself. These are prefixed with a double underscore (e.g. "__my_private_attribute"), and can only be accessed using special name mangling syntax (e.g. "_ClassName__my_private_attribute").

Here's an example of how encapsulation can be implemented in Python:

Listing 1. An exampleEncapsulation in Python
```
class Car:
    def __init__(self, make, model):
        self.__make = make
        self.__model = model

    def get_make(self):
        return self.__make

    def get_model(self):
        return self.__model

    def set_make(self, make):
        self.__make = make

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

In this example, the attributes "make" and "model" are declared as private using the double underscore prefix. The class provides public getter and setter methods for these attributes, which allow other code to access and modify them indirectly. By doing this, the implementation details of the Car class are hidden from the outside world, and the code is better organized and easier to maintain.

Results. Public Members. Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method. This arrangement of private instance variables and public methods ensures the principle of data encapsulation.

All members in a Python class are public by default. Any member can be accessed from outside the class environment.

Listing 1: Public Attributes
```
class Student:
     schoolName = 'XYZ School' # class
attribute

    def __init__(self, name, age):
        self.name=name # instance attribute
        self.age=age # instance attribute
```

You can access the Student class's attributes and also modify their values, as shown below.

Example: Access Public Members

```
>>> std = Student("Steve", 25)
>>> std.schoolName
'XYZ School'
>>> std.name
'Steve'
>>> std.age = 20
>>> std.age
20
```

Protected Members. Protected members of a class are accessible from within the class and are also available to its sub-classes. No other environment is permitted access to it. This enables specific resources of the parent class to be inherited by the child class.

Python's convention to make an instance variable protected is to add a prefix _ (single underscore) to it. This effectively prevents it from being accessed unless it is from within a sub-class.

Listing 2: Protected Attributes

```
class Student:
    _schoolName = 'XYZ School'
# protected class attribute

    def __init__(self, name, age):
        self._name=name  #
protected instance attribute
        self._age=age #
protected instance attribute
```

In fact, this doesn't prevent instance variables from accessing or modifying the instance. You can still perform the following operations:

Example: Access Protected Members

```
>>> std = Student("Swati", 25)
>>> std._name
'Swati'
>>> std._name = 'Dipa'
>>> std._name
'Dipa'
```

However, you can define a property using property decorator and make it protected, as shown below.

Listing 3: Protected Attributes

```
class Student:
    def __init__(self,name):
        self._name = name
    @property
    def name(self):
        return self._name
    @name.setter
    def name(self,newname):
        self._name = newname
```

Above, @property decorator is used to make the name() method as property and @name.setter decorator to another overloads of the name() method as property setter method. Now, _name is protected.

Example: Access Protected Members

```
>>> std = Student("Swati")
>>> std.name
'Swati'
>>> std.name = 'Dipa'
>>> std.name
'Dipa'
>>> std._name # still accessible
```

Above, we used std.name property to modify _name attribute. However, it is still accessible in Python. Hence, the responsible programmer would refrain from accessing and modifying instance variables prefixed with _ from outside its class.

Private Members. Python doesn't have any mechanism that effectively restricts access to any instance variable or method. Python prescribes a convention of prefixing the name of the variable/ method with a single or double underscore to emulate the behavior of protected and private access specifiers.

The double underscore __ prefixed to a variable makes it private. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError:

Listing 4: Private Attributes

```
class Student:
    __schoolName = 'XYZ School'
# private class attribute

    def __init__(self, name, age):
        self.__name=name  #
```

```
private instance attribute
         self.__salary=age #
private instance attribute
     def __display(self):  #
private method
         print('This is
private method.')
```

Example: Access Private Members and Errors

```
 >>> std = Student("Bill", 25)
 >>> std.__schoolName
 AttributeError: 'Student' object
has no attribute '__schoolName'
 >>> std.__name
 AttributeError: 'Student' object
has no attribute '__name'
 >>> std.__display()
 AttributeError: 'Student' object
has no attribute '__display'
```

Python performs name mangling of private variables. Every member with a double underscore will be changed to _object._class__variable. So, it can still be accessed from outside the class, but the practice should be refrained.[6]

Example: Access Private Members in Python

```
>>> std = Student("Bill", 25)
>>> std._Student__name
'Bill'
>>> std._Student__name = 'Steve'
>>> std._Student__name
'Steve'
>>> std._Student__display()
'This is private method.'
```

Conclusion. Python is a modern fast-growing programming language that continues to grow in technology market share. So, according to the monthly ranking of the popularity of programming languages PyPL, created on the largest web service of projects Github, based on search queries on the Internet, Python takes the first place at the time of June 2022 with 27.61% of the total number of requests related to programming languages. One of the reasons for the popularity of the Python programming language is that this programming language has a large number of additional specialized software libraries designed to process and analyze large amounts of data. [7]

Thus, in the Python programming language, members with public, protected, and private views can be used. But the use of these members is quite different from how it is in programming languages like C#, Java, C++. This aspect is also another priority of the Python programming language.

References

1. Полсон Л. Разработчики переходят на динамические языки / Л. Полсон: http://www.osp.ru/os/2007 /02/4108153.

2. Лутц М. Изучаем Python / М. Лутц. – М.: Символ-Плюс, 2010. – 1280 с.

3. Duck Typing in Python: http://www.voidspace.org.uk/python/articles/duck_typing.shtml.

4. Зубов, М. В. Получение типов данных в языках с динамической типизацией для статического анализа исходного кода с помощью универсального классового представления / М. В. Зубов, А. Н. Пустыгин, Е. В. Старцев // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2013. – № 2. – С. 66-74. – EDN QJBSCD.

5. Московцев, М. Н. Программная реализация геометрического алгоритма многокритериальной оптимизации / М. Н. Московцев // Омский научный вестник. – 2014. – № 3(133). – С. 15-17. – EDN TKDJPD.

6. Muminjonovich K. A. Sun'yiy inellektni rivojlantirishda dasturlash tillarining ro 'li //Journal of new century innovations. – 2023. – Т. 12. – №. 4. – С. 159-161.

7. Musayev X.SH., Ermatova Z.Q., Kotlin dasturlash tilida korutinlar bilan ishlashni talabalarga o 'rgatish //Journal of Integrated Education and Research. – 2022. – Т. 1. – №. 6. – С. 119-125.