Модификаторы доступа класса

Private

Модификатор private делает члены класса (переменные, методы и т.д.) доступными только внутри самого этого класса.

Protected

Члены класса, помеченные модификатором protected, будут доступны в пределах того же пакета и во всех классах-наследниках.

Public

Если член класса объявлен как public, он будет доступен в любом месте программы.

Пример:

Модификатор доступа Public

```
class Student:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def display(self):
    print("Name:", self.name)
    print("Age:", self.age)

s = Student("John", 20)
s.display()
```

Вывод:

```
Name: John
Age: 20
```

Модификатор доступа Private

```
class BankAccount:
  def __init__(self, account_number, balance):
    # Закрытые члены класса
    self.__account_number = account_number
    self.__balance = balance

  # Закрытый метод класса
  def __display_balance(self):
    print("Balance:", self.__balance)

b = BankAccount(1234567890, 5000)
b.__display_balance()
```

Вывод:

AttributeError: 'BankAccount' object has no attribute '__display_balance'

Ибо метод закрытый

Модификатор доступа Protected

```
# Родительский класс
class Person:
  def __init__(self, name, age):
    # Защищенные члены класса
    self._name = name
    self._age = age

  # Защищенный метод класса
  def _display(self):
    print("Name:", self._name)
    print("Age:", self._age)


# Класс производный от родительского
class Student(Person):
  def __init__(self, name, age, roll_number):
    super().__init__(name, age)
    # Инициализация защищенного члена класса
    self._roll_number = roll_number

  # Открытый  метод класса
  def display(self):
    # Вызов защищенного метода класса, унаследованного от родителя
    self._display()
```

```
        print("Roll Number:", self._roll_number)

    s = Student("John", 20, 123)
    s.display()
```

Вывод:

```
    Name: John
    Age: 20
    Roll Number: 123
```

Пример с GitHub:

```
    class Rating:
        def __init__(self, *marks) # marks (5,3,4) | (2) | ()
            # Закрытый член класса
            self.__mark = self.avarage_mark(marks)

        # Открытый метод класса
        def avarage_mark(self, marks):
            s = 0
            for mark in marks:
                s += mark
            return s/len(marks)

        # Обращение к закрытому члену класса через свойство
        @property
        def mark(self):
            return self.__mark

        # Определение оператора >
        def __gt__(self, other):
            return self.mark > other.mark # возвращается логическое значение

        # Определение оператора +
        def __add__(self,other):
            return self.mark + other.mark

        # Строковое представление класса
        def __str__(self):
            return "avarage mark: " + str(self.__avarage_mark )

    # rating = Rating(5,5,5,5)
    # print(rating > Rating(1,1,1) )
    # print((rating + Rating(1,1,1)+Rating(2,2,2))/3 )
```