Access Specifiers/Modifiers in python

Access specifiers or access modifiers in python programming are used to limit the access of class variables and class methods outside of class while implementing the concepts of inheritance.

Let us see the each one of access specifiers in details:

Types of Specifiers

1. Public Access Modifiers

2. Private Access Modifiers

3. Protected Access Modifiers

Private Access Modifier

By definition, Private members of a class variables or methods are those members which are only accessible inside the class. We cannot use private members outside of class.

In python, there is no strict of concept of “private” access modifiers like some other programming language. However a convention has been established to indicate that a variable or method should be considered private by prefixing its name with a double under sore (__). This is known as a “week internal use indicator” and it is a convention only, not a strict rule. Code outside the class can still access these “private” variables and methods, but it is generally understood that they should not be accessed or modified.

Example:

```
class Student:
    def __init__(self):
        self.name="Munawar"
        self.__PrivateName="Johar" #this is a private variable

s=Student()
```

```python
print(s.name)
# print(s.__PrivateName) #it is cannot be accessiable because it is a private
vaiable
```

Name Mangling

Name mangling in python is a technique used to protect class private and superclass private attributes from being accidentally overwritten by subclasses. Names of class private and superclass private attributes are transformed by the addition of a single leading under score and a double leading underscore respectively.

Example:

```python
class Student:
    def __init__(self):
        self.name="Munawar"
        self.__PrivateName="Johar" #this is a private variable

s=Student()
print(s.name)
# print(s.__PrivateName) #it is cannot be accessiable because it is a private
vaiable

#Can be access indirectly
# print(s.__Student__PrivateName)


# print(s.__dir__)
```

Protected Access Modifier

In object oriented programming (OOP), the term “protected” is used to describe be a member (i. e a method or attribute) of a class that is intended to be accessed only by the class itself and its subclass. In python, the convention for indicating that a member is protected us to prefix its name with a single under score (_). For example, if a class has a method _my method, it is indicating that the method should only be accessed by the class itself and its subclasses.

It’s important to note that the single under score is just a naming conversion, and does not actually provide any protection or restrict access to the member. The syntax we follow to make any variable protected is to write variable name followed by a single under score (_) is _variable name.

Example:

```
class Programmer:
    def __init__(self):
        self.Pname="Munawar"
    def _Method(): # protected
        return "MunawarJohar"


class Developer(Programmer):#inherited
    pass
```

```
obj=Programmer();
obj2=Developer()
print(obj.Pname)
print(obj._Method())
# print(obj2._Method())
```

Source Code

```
# class Student:
#     def __init__(self):
#         self.name="Munawar"
#         self.__PrivateName="Johar" #this is a private variable



class Programmer:
    def __init__(self):
```

```python
        self.Pname="Munawar"
    def _Method(): # protected
        return "MunawarJohar"


class Developer(Programmer):#inherited
    pass

# s=Student()
# print(s.name)
# # print(s.__PrivateName) #it is cannot be accessiable because it is a private vaiable

# #Can be access indirectly
# # print(s.__Student__PrivateName)


# # print(s.__dir__)


obj=Programmer();
obj2=Developer()
print(obj.Pname)
print(obj._Method())
# print(obj2._Method())
```

Thank You