Access Modifiers in Python

In most of the object-oriented languages access modifiers are used to limit the access to the variables and functions of a class. Most of the languages use three types of access modifiers, they are - **private**, **public** and **protected**.

Just like any other object oriented programming language, access to variables or functions can also be limited in python using the access modifiers. Python makes the use of **underscores** to specify the access modifier for a specific data member and member function in a class.

Access modifiers play an important role to protect the data from unauthorized access as well as protecting it from getting manipulated. When inheritance is implemented there is a huge risk for the data to get destroyed(manipulated) due to transfer of unwanted data from the parent class to the child class. Therefore, it is very important to provide the right access modifiers for different data members and member functions depending upon the requirements.

Python: Types of Access Modifiers

There are 3 types of access modifiers for a class in Python. These access modifiers define how the members of the class can be accessed. Of course, any member of a class is accessible inside any member function of that same class. Moving ahead to the type of access modifiers, they are:

Access Modifier: Public

The members declared as Public are accessible **from outside the Class** through an object of the class.

Access Modifier: Protected

The members declared as Protected are accessible **from outside the class but only in a class derived from it that is in the child or subclass.**

Access Modifier: Private

These members are only accessible from **within the class**. No outside Access is allowed.

Time for some Examples

In this section we will provide some basic code examples for each type of access modifier.

public Access Modifier

By default, all the variables and member functions of a class are public in a python program.

# defining a class Employee

class Employee:

# constructor

def \_\_init\_\_(self, name, sal):

self.name = name

self.sal = sal

All the member variables of the class in the above code will be by default public, hence we can access them as follows:

>>> emp = Employee("Ironman", 999000)

>>> emp.sal

999000

protected Access Modifier

According to Python convention adding a prefix \_(single underscore) to a variable name makes it protected. Yes, no additional keyword required.

# defining a class Employee

class Employee:

# constructor

def \_\_init\_\_(self, name, sal):

self.\_name = name # protected attribute

self.\_sal = sal # protected attribute

In the code above we have made the class variables **name** and **sal** protected by adding an \_(underscore) as a prefix, so now we can access them as follows:

>>> emp = Employee("Captain", 10000)

>>> emp.\_sal

10000

Similarly if there is a child class extending the class Employee then it can also access the protected member variables of the class Employee. Let's have an example:

# defining a child class

class HR(Employee):

# member function task

def task(self):

print "We manage Employees"

Now let's try to access protected member variable of class Employee from the class HR:

>>> hrEmp = HR("Captain", 10000)

>>> hrEmp.\_sal

10000

>>> hrEmp.task()

We manage Employees

private Access Modifier

While the addition of prefix \_\_(double underscore) results in a member variable or function becoming private.

# defining class Employee

class Employee:

def \_\_init\_\_(self, name, sal):

self.\_\_name = name # private attribute

self.\_\_sal = sal # private attribute

If we want to access the **private** member variable, we will get an error.

>>> emp = Employee("Bill", 10000)

>>> emp.\_\_sal

AttributeError: 'employee' object has no attribute '\_\_sal'

All in one Example

Now that we have seen each access modifier in separate examples, now let's combine all that we have learned till now in one example:

# define parent class Company

class Company:

# constructor

def \_\_init\_\_(self, name, proj):

self.name = name # name(name of company) is public

self.\_proj = proj # proj(current project) is protected

# public function to show the details

def show(self):

print("The code of the company is = ",self.ccode)

# define child class Emp

class Emp(Company):

# constructor

def \_\_init\_\_(self, eName, sal, cName, proj):

# calling parent class constructor

Company.\_\_init\_\_(self, cName, proj)

self.name = eName # public member variable

self.\_\_sal = sal # private member variable

# public function to show salary details

def show\_sal(self):

print("The salary of ",self.name," is ",self.\_\_sal,)

# creating instance of Company class

c = Company("Stark Industries", "Mark 4")

# creating instance of Employee class

e = Emp("Steve", 9999999, c.name, c.\_proj)

print("Welcome to ", c.name)

print("Here ", e.name," is working on ",e.\_proj)

# only the instance itself can change the \_\_sal variable

# and to show the value we have created a public function show\_sal()

e.show\_sal()

Now the code above show the correct usage of public, private and protected member variables and methods. You can try and change a few things a run the program to see what error those changes result into.

Without Underscore – public

Single underscore – protected

Double underscore - Private