**Encapsulation in Python**

Bundling of attributes and methods that operate on the data into a single unit. Python achieves encapsulation through public, protected and private attributes.

class is an example of encapsulation

**Public Members**

Public members are accessible from anywhere, both inside and outside the class. These are the default members in Python.

**class** **Public**:

**def** \_\_init\_\_(self):

self.name="John" *# Public attribute*

**def** display\_name(self):

print(self.name) *# Public method*

obj = Public()

obj.display\_name() *# Accessible*

print(obj.name) *# Accessible*

**Protected members**

Protected members are identified with a single underscore **(\_)**. They are meant to be accessed only within the class or its subclasses.

**class** **Protected**:

**def** \_\_init\_\_(self):

self.\_age = 30 *# Protected attribute*

**class** **Subclass**(Protected):

**def** display\_age(self):

print(self.\_age) *# Accessible in subclass*

obj = Subclass()

obj.display\_age()

**Private members**

Private members are identified with a double underscore **(\_\_)** and cannot be accessed directly from outside the class

**class** **Private**:

**def** \_\_init\_\_(self):

self.\_\_salary = 50000 *# Private attribute*

**def** salary(self):

**return** self.\_\_salary *# Access through public method*

obj = Private()

print(obj.salary()) *# Works because you called method*

*#print(obj.\_\_salary) # Raises AttributeError*

**Access Modifiers in Python : Public, Private and Protected**

Python uses the '\_' symbol to determine the access control for a specific data member or a member function of a class.

**Public Access Modifier:**

The members of a class that are declared public are easily accessible from any part of the program.

**Protected Access Modifier:**

The members of a class that are declared protected are only accessible within the class where it is declared and its subclass.

**class** **Student**:

*# protected data members*

\_name = **None**

\_roll = **None**

\_branch = **None**

*# constructor*

**def** \_\_init\_\_(self, name, roll, branch):

self.\_name = name

self.\_roll = roll

self.\_branch = branch

*# protected member function*

**def** \_displayRollAndBranch(self):

*# accessing protected data members*

print("Roll:", self.\_roll)

print("Branch:", self.\_branch)

**Private Access Modifier:**

The members of a class that are declared private are accessible within the class only

**Private Variables in Python**

 Name prefixed with an underscore, For e.g. **\_geek** should be treated as a non-public part

**Mangling**

There is limited support for a valid use-case for class-private members basically to avoid name clashes of names with names defined by subclasses.

Any identifier of the form \_\_geek is replaced with \_classname\_\_geek

**Private Methods in Python**

Example shows that private methods of the class can neither be accessed outside the class nor by any base class.

**class** **Base**:

**def** fun(self):

print("Public method")

**def** \_\_fun(self):

print("Private method")

**class** **Derived**(Base):

**def** \_\_init\_\_(self):

Base.\_\_init\_\_(self)

**def** call\_public(self):

print("**\n**Inside derived class")

self.fun()

**def** call\_private(self):

self.\_\_fun()

obj1 = Base()

obj1.fun()

obj2 = Derived()

obj2.call\_public()

*# Uncommenting obj1.\_\_fun() will*

*# raise an AttributeError*

*# Uncommenting obj2.call\_private()*

*# will also raise an AttributeError*

private methods can be accessed by calling the private methods via public methods.

**class** **A**:

**def** fun(self):

print("Public method")

**def** \_\_fun(self):

print("Private method")

**def** Help(self):

self.fun()

self.\_\_fun()

obj = A()

obj.Help()

**Name mangling**

**class** **A**:

**def** fun(self):

print("Public method")

**def** \_\_fun(self):

print("Private method")

obj = A()

*# Calling the private member*

*# through name mangling*

obj.\_A\_\_fun()

**Protected variable in Python**

**Protected variables** are those data members of a class that can be accessed within the class and the classes derived from that class. In Python, there is no existence of “Public” instance variables. However, we use **underscore '\_'** symbol to determine the access control of a data member in a class.