**Encapsulation** means hiding internal details of a class and only exposing what’s necessary. It helps to protect important data from being changed directly and keeps the code secure and organized.

This example shows encapsulation by keeping \_\_salary variable private inside Employee class. It cannot be accessed directly from outside the class.

While Python does not enforce strict access modifiers like public, private or protected as in some other languages, encapsulation is achieved through.

Access Specifiers:

* Access specifiers define how class members (variable and methods) can be accessed from outside the class. They help in implementing encapsulation by controlling the visibility of data. There are three types of access specifiers.
* Public
* Protected
* Private

**1.Public Members**

* Public members are variables or methods that can be accessed from anywhere inside the class, outside the class or from other modules. By default, all members in Python are public

.

* They are defined without any underscore prefix (e.g., self.name).

Eg:

class Employee:

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

self.name = name

def display\_name(self):

print(self.name)

emp = Employee("John")

emp.display\_name()

print(emp.name)

output:

John

John

### 2. Protected members

* Protected members are variables or methods that are intended to be accessed only within the class and its subclasses. They are not strictly private but should be treated as internal.
* In Python, protected members are defined with a single underscore prefix (e.g., self.\_name).

Eg:

class Employee:

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

self.name = name # public

self.\_age = age # protected

class SubEmployee(Employee):

def show\_age(self):

print("Age:", self.\_age) # Accessible in subclass

emp = SubEmployee("Ross", 30)

print(emp.name) # Public accessible

emp.show\_age()

### 3. Private members

* Private members are variables or methods that cannot be accessed directly from outside the class. They are used to restrict access and protect internal data.
* In Python, private members are defined with a double underscore prefix (e.g., self.\_\_salary). Python applies [name mangling](https://www.geeksforgeeks.org/python/name-mangling-in-python/) by internally renaming them (e.g., \_\_salary becomes \_ClassName\_\_salary) to prevent direct access.

Eg:

class Employee:

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

self.name = name # public

self.\_\_salary = salary # private

def show\_salary(self):

print("Salary:", self.\_\_salary)

emp = Employee("Robert", 60000)

print(emp.name) # Public accessible

emp.show\_salary() # Accessing private correctly

# print(emp.\_\_salary) # Error: Not accessible directly