What is Encapsulation in Python?

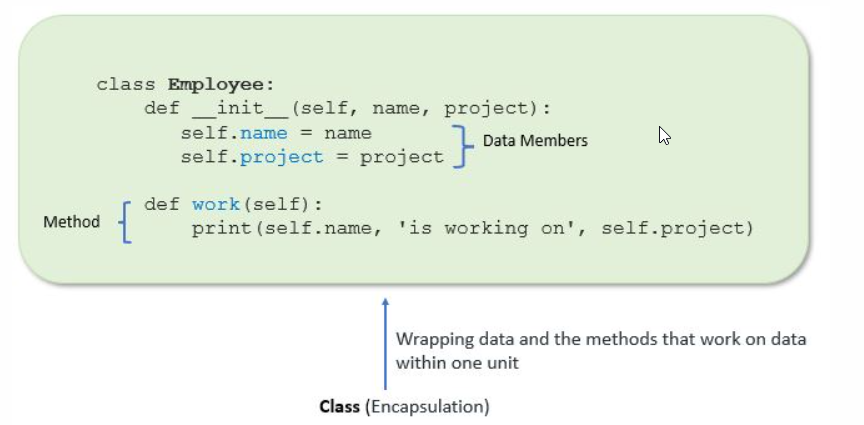
Encapsulation is one of the fundamental principles of object-oriented programming (OOP). It refers to the concept of **bundling data (attributes) and methods (functions) that operate on the data into a single unit** called a class. This combination of data and methods within a class forms the basis for creating objects in OOP languages.

The primary goal of encapsulation is to hide the internal details of an object's implementation from the outside world and provide controlled access to the object's properties and behaviors. This helps in achieving several important objectives:

1.**Data Protection**: Encapsulation allows you to restrict direct access to an object's data (attributes) from external code. Instead of manipulating the data directly, external code interacts with the object through well-defined methods. This safeguards the integrity and consistency of the object's state.

2. **Abstraction**: Encapsulation enables you to present a simplified, high-level interface to the users of the class. Users of the class don't need to know the internal implementation details; they only need to know how to use the provided methods.

3. **Flexibility and Maintainability**: By encapsulating the implementation details within the class, you can change the internal workings of the class without affecting the external code that relies on it. This allows for easier updates, maintenance, and improvements to the codebase.

4. **Code Organization**: Encapsulation promotes modular code design by grouping related data and methods together in a structured manner. This enhances the organization and readability the code. 

encapsulation allows us to restrict accessing variables and methods directly and prevent accidental data modification by creating private data members and methods within a class.

Encapsulation is a way to can restrict access to methods and variables from outside of class. Whenever we are working with the class and dealing with sensitive data, providing access to all variables used within the class is not a good choice.

For example, suppose you have an attribute that is not visible from the outside of an object and bundle it with methods that provide read or write access. In that case, you can hide specific information and control access to the object’s internal state. Encapsulation offers a way for us to access the required variable without providing the program full-fledged access to all variables of a class. This mechanism is used to protect the data of an object from other objects.

**Access Modifiers in Python:**

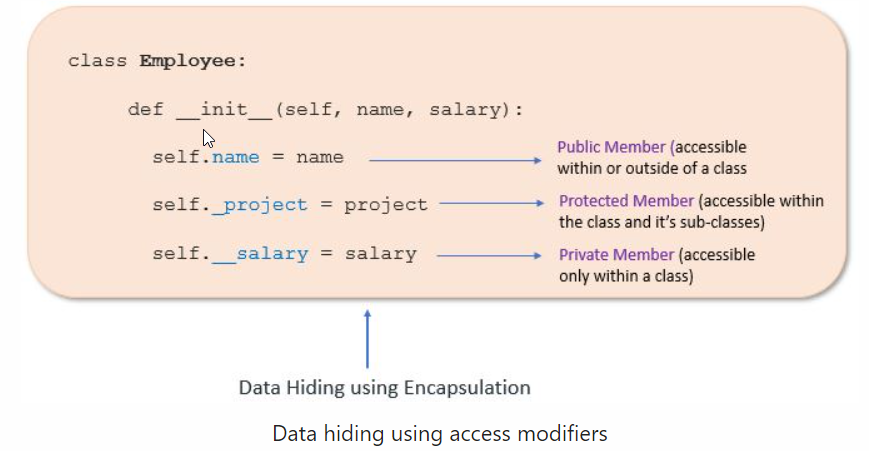
Encapsulation can be achieved by declaring the data members and methods of a class either as private or protected. But In Python, we don’t have direct access modifiers like public, private, and protected. We can achieve this by using single underscore and double underscores.

Access modifiers limit access to the variables and methods of a class. Python provides three types of access modifiers private, public, and protected.

**Public Member**: Accessible anywhere from outside class.

**Private Member**: Accessible within the class

**Protected Member**: Accessible within the class and its sub-classes



class Employee:

    def \_\_init\_\_(self,name,project,salary) -> None:

        self.name = name          # Public Specifier (Accessible within or outside of the class)

        self.\_project = project   # Protected Specifier (Accessible within class or subclass)

        self.\_\_salary = salary    # Private Specifier (Accessible within class )

    def detail(self):

        print(f"name is : {self.name}")

        print(f"name is : {self.\_project}")

        print(f"name is : {self.\_\_salary}")

Object\_rohit = Employee("ROhit","Python",100000)

Object\_rohit.detail()

print(Object\_rohit.name)   #output ROhit

print(Object\_rohit.\_project) #output Python

print(Object\_rohit.\_\_salary) #output AttributeError: 'Employee' object has no attribute '\_\_salary'

Object\_rohit.detail()

#OUTPUT

name is : ROhit

name is : Python

name is : 100000