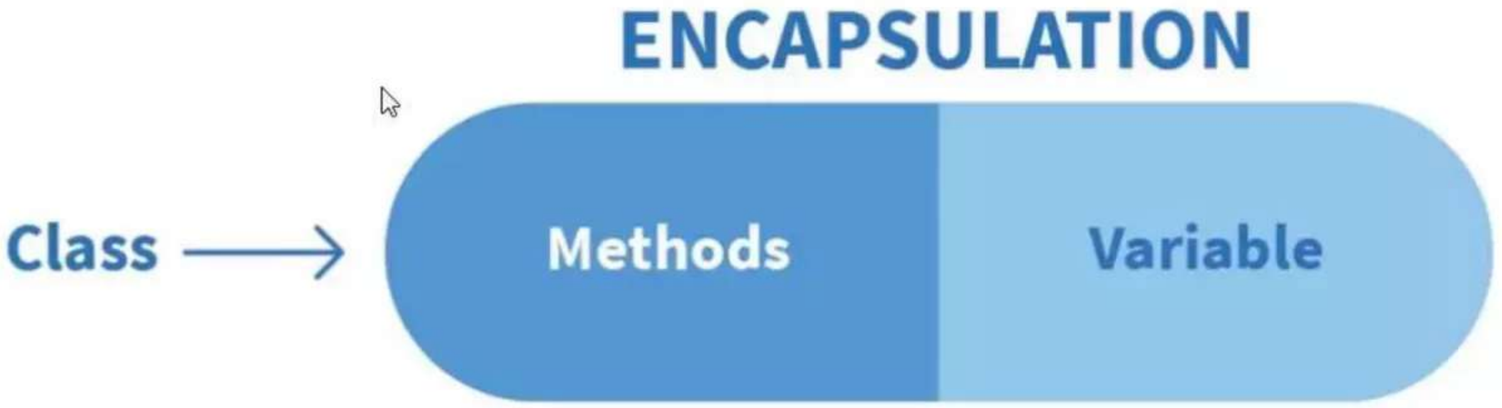
Python-ENCAPSULATION

Basics about Encapsulation?

Encapsulation is one of the critical features of object-oriented programming, which involves the bundling of data members and functions inside a single class. Bundling similar data members and functions inside a class also helps in data hiding. Encapsulation also ensures that objects are self-sufficient functioning pieces and can work independently.

What is Encapsulation?

Encapsulation is one of the cornerstone concepts of OOP. The basic idea of Encapsulation is to wrap up both data and methods into one single unit. The way that data and methods are organized does not matter to the end-user. The user is only concerned about the right way to provide input and expects a correct output on the basis of the inputs provided.



Why do we need encapsulation in python?

1. Encapsulation provides well-defined, readable code

2. Prevents Accidental Modification or Deletion

3. Encapsulation provides security

Access Modifier in Python Encapsulation

Sometimes there might be a need to restrict or limit access to certain variables or functions while programming. There we need an Access Modifier.

1. Public Member
2. Private Member
3. Protected Member

Class member access specifier	Access from own class	Accessible from derived class	Accessible from object
Private member	Yes	No	No
Protected member	Yes	Yes	No
Public member	Yes	Yes	Yes

Encapsulation in Python using Public Members

the public modifier allows variables and functions to be accessible from anywhere within the class and from any part of the program. All member variables have the access modifier as public by default.

```
# illustrating public members & public access modifier
class pub_mod:
    # constructor
    def __init__(self, name, age):
        self.name = name;
        self.age = age;

    def Age(self):
        # accessing public data member
        print("Age: ", self.age)
# creating object
obj = pub_mod("Jason", 35);
# accessing public data member
print("Name: ", obj.name)
# calling public member function of the class
obj.Age()

Output:
Name:  Jason
Age:  35
```

Encapsulation in Python using Private Members

The private access modifier allows member methods and variables to be accessed only within the class. To specify a private access modifier for a member, we make use of the double underscore __.

```
# illustrating private members & private access modifier
class Rectangle:
    __length = 0 #private variable
    __breadth = 0#private variable
    def __init__(self):
        #constructor
        self.__length = 5
        self.__breadth = 3
        #printing values of the private variable within the class
        print(self.__length)
        print(self.__breadth)

rect = Rectangle() #object created
#printing values of the private variable outside the class
print(rect.length)
print(rect.breadth)

Output:
5
3
Traceback (most recent call last) :
  File "main.py", line 14, in <module>
    print(rect.length)
AttributeError: 'Rectangle' object has no attribute 'length'
```

Encapsulation in Python Using Protected Members

What sets protected members apart from private members is that they allow the members to be accessed within the class and allow them to be accessed by the sub-classes involved. In Python, we demonstrate a protected member by prefixing with an underscore _ before its name.

```
# illustrating protected members & protected access modifier
class details:
    _name="Jason"
    _age=35
    _job="Developer"
class pro_mod(details):
    def __init__(self):
        print(self._name)
        print(self._age)
        print(self._job)

# creating object of the class
obj = pro_mod()
# direct access of protected member
print("Name:",obj.name)
print("Age:",obj.age)

Output:
Jason
35
Developer
Traceback (most recent call last):
  File "main.py", line 15, in <module>
    print("Name:",obj.name)
AttributeError: 'pro_mod' object has no attribute 'name'
```