What is Encapsulation?

1. Encapsulation in python describes the concept of bindling data and methods witihin a single unit. For eg: When you create a class it means that you are implementing Encapsulation.
2. A class is an example of Encapsulation that binds all the data members and methods into a single unit.
3. We can hide object's internal representation from outside the function, this is called information hiding. Also, Encapsulation allows us to restrict accessing variable and methods directly and prevent accidental data modification by creating private data members and methods within a class.

Access Modifiers

1. Encapsulation can be achieved by declaring the data members and methods of a class either private or protected.
2. But in python we dont have directly access modifiers like public, private and protected.
3. We can achieve this by using a single underscore(' _ ') or double underscore(' __ ').
4. Accessing modifier limit the accessto the variables and method of the class:
   a)Public Members: Accessable anywhere from within a class or outside the class
   b)Private Members: Accessable within the class
   c)Protected Members: Accessable within the class and sub class

Public Member

In [1]:

```
class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.salary = salary

emp = Employee('Jessica', 10000)

print('Salary: ', emp.salary)
```

```
Salary:  10000
```

Private Member

1. We can protect variable in the class by making them private to define the private vehicle add two underscore before variable name.
2. Private members are accessable only within the class, and we can't access them directly from class objects.

In [2]:

```
class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.__salary = salary

emp = Employee('Jessica', 10000)

print('Salary: ', emp.__salary)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9748/1394218115.py in <module>
      6 emp = Employee('Jessica', 10000)
      7
----> 8 print('Salary: ', emp.__salary)

AttributeError: 'Employee' object has no attribute '__salary'
```

We can access private members from outside of the class using two approaches

1. Create a public method to craete a access private member

In [3]:

```
class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.__salary = salary

    def show(self):
        print('Name:', self.name, ' Salary:', self.__salary)

emp = Employee('Jessica', 10000)
emp.show()
```

```
Name: Jessica  Salary: 10000
```

2. Use name mangling

The name mangling is created on an identifier by adding one leading under score and two trailing underscore.

In [5]:

```
class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.__salary = salary

emp = Employee('Jessa',10000)

print("Name:",emp.name)
print("Salary:",emp._Employee__salary)
```

```
Name: Jessa
Salary: 10000
```

Protected members

1. Protected members are accessable within the class and also availabe to its sub class (child class).
2. To define a protected member we can use single underscore before variable name.
3. Protected data members are use when you implementing inheritance and wanted to allow data members access to only child class.

In [10]:

```
class Company:
    def __init__(self):
        self._project = "NLP"

class Employee(Company):
    def __init__(self,name):
        self.name = name
        Company.__init__(self)

    def show(self):
        print("Employee name :",self.name)
        print("Working on project :", self._project)

c = Employee("Jessa")
c.show()
print("Project",c._project)
```

```
Employee name : Jessa
Working on project : NLP
Project NLP
```

Getters and Setters (HW)

Advantages of Encapsulation

Security

1. Encapsulation provides an object from unauthorized access.
2. It allows private and protected access levels to prevent accidental data modification.

Data Hiding

1. The user dosen't know behind the seeen of the class.
2. They woukld only be knowing that to modify a data member, call the setter methods.
3. To read a data member, call the getter method. These getters and setters method are used in data hiding.

Simplicity

1. It simplifies the maintainance of the application by keeping classes seperated and preventing them from tightly coupling with each other.

Aesthetics

1. Bundling the data and methods within a class makes code more readable and maintainable.

In [ ]:

```
1
```

In [ ]:

```
1
```