# Understanding Encapsulation Using Examples

In this lesson, you will get a firmer understanding of encapsulation in Python with the help of examples.

WE'LL COVER THE FOLLOWING



As discussed earlier, encapsulation refers to the concept of binding **data and the methods operating on that data** in a single unit also called a class.

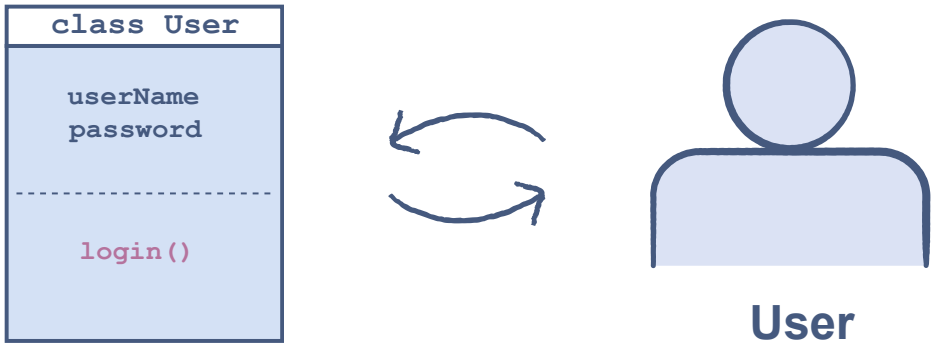
The goal is to prevent this bound data from any unwanted access by the code outside this class. Let's understand this using an example of a very basic `User` class.

Consider that we are up for designing an application and are working on modeling the **log in** part of that application. We know that a user needs a **username** and a **password** to log into the application.

An elementary `User` class will be modeled as:

- Having a property `userName`
- Having a property `password`
- A method named `login()` to grant access

Whenever a new user comes, a new object can be created by passing the `userName` and `password` to the constructor of this class.




## A Bad Example #

Now it is time to implement the above discussed `User` class.


The code for the above illustration is given below:

```
class User:
    def __init__(self, userName=None, password=None):
        self.userName = userName
        self.password = password
```

```
    def login(self, userName, password):
        if ((self.userName.lower() == userName.lower())
                and (self.password == password)):
            print("Access Granted!")
        else:
            print("Invalid Credentials!")


Steve = User("Steve", "12345")
Steve.login("steve", "12345")
Steve.login("steve", "6789")
Steve.password = "6789"
Steve.login("steve", "6789")
```

In the above coding example, we can observe that **anyone** can *access, change, or print* the `password` and `userName` fields directly from the main code. This is **dangerous** in the case of this `User` class because there is no encapsulation of the credentials of a user, which means anyone can access their account by manipulating the stored data. So, the above code did not follow good coding practices.

## A Good Example #

Let's move on to a better implementation of the `User` class!

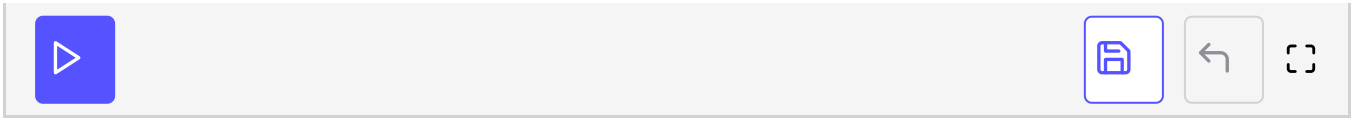

In the code below an `AttributeError` will be thrown because the code outside the `User` class tried to access a `private` property.

```
class User:
```

```
    def __init__(self, userName=None, password=None):
        self.__userName = userName
        self.__password = password


    def login(self, userName, password):
        if ((self.__userName.lower() == userName.lower())
                and (self.__password == password)):
            print(
                "Access Granted against username:",
                self.__userName.lower(),
                "and password:",
                self.__password)
        else:
            print("Invalid Credentials!")


# created a new User object and stored the password and username
Steve = User("Steve", "12345")
Steve.login("steve", "12345")  # Grants access because credentials are valid
# does not grant access since the credentails are invalid
Steve.login("steve", "6789")
Steve.__password  # compilation error will occur due to this line
```



> If you comment out **line 23**, the program will work.

## Explanation

- In the above example, the fields of `__userName` and `__password` are declared privately using the `__` prefix.
- We can observe that **no one** can *access, change, or print* the `__password` and `__userName` fields directly from the main code. This is a proper implementation of encapsulation.

> **Note:** For encapsulating a class, all the properties should be private and any access to the properties should be through methods such as *getters* and *setters*.

This is the concept of encapsulation. All the properties containing data are private, and the methods provide an interface to access those private properties.

Now let’s test your understanding of encapsulation with a quick quiz!