Encapsulamento

segunda-feira, 10 de outubro de 2022 16:31

Serve para proteger uma classe ou algum objeto que queremos proteger

metodos e atributos publicos são os que são acessados dentro e fora da classe,protected são os que só podem ser acessados dentro da classe ou das filhas daquela classe, private só está disponivel dentro da classe.

Quando estamos falando de atributos publicos eles podem ser alterados a qualquer momento dentro do código quando chamado na instância.

Exemplo:

```
class BancoDeDados:
        def __init__(self):
            self.dados = {}

        def inserir_dados(self,id,nome):
            if 'Clientes' not in self.dados:
                self.dados['Clientes'] = ({id: nome})
            else:
                self.dados['Clientes'].update({id: nome})

        def listar_clientes(self):
            for id,nome in self.dados.items():
                print(id,nome)

        def apaga_clientes(self,id):
            del self.dados['Clientes'][id]


BD = BancoDeDados()
BD.inserir_dados(1,'Geronimo')
BD.inserir_dados(2,'Ana')
```

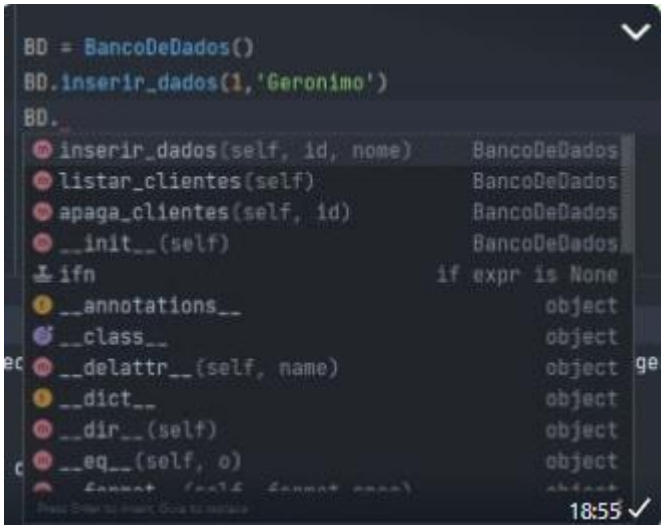
Fazendo alteração no atributo publico :

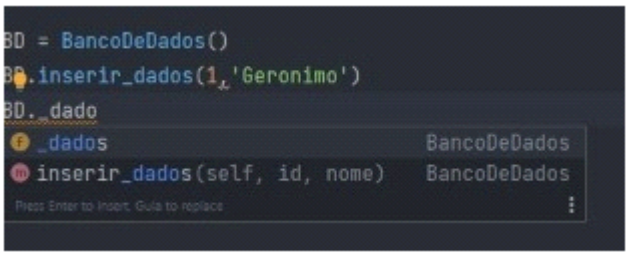
```
BD = BancoDeDados()
BD.inserir_dados(1,'Geronimo')
BD.dados = 'Outro Valor'
print(BD.dados)
```

Logo o valor de dados que é essencial publico vira 'outro valor' desordenando tudo.

Por convenção temos como colocar o _ antes do nome do atributo para que ele não seja alterado, porém ele ainda pode ser alterado se alguem procurar por ele, mas inicialmente o python n oferece essa mudança.

Exemplo:





é como se ele funcionasse como um atributo protected.

logo quando queremos deixar um atributo privado sem que seja alterado de forma alguma ou quase alguma devemos colocar __ antes do nome do atributo.

Exemplo:

```python
class BancoDeDados:
        def __init__(self):
            self.__dados = {}

        def inserir_dados(self,id,nome):
            if 'Clientes' not in self.__dados:
                self.__dados['Clientes'] = ({id: nome})
            else:
                self.__dados['Clientes'].update({id: nome})

        def listar_clientes(self):
            for id,nome in self.__dados.items():
                print(id,nome)

        def apaga_clientes(self,id):
            del self.__dados['Clientes'][id]


BD = BancoDeDados()
BD.inserir_dados(1,'Geronimo')
print(BD._BancoDeDados__dados)
```

Dessa forma fazemos com que o atributo fique privado.

único meio de alterar ou acessar é sendo chamado como foi no print acima.

se quiser modificar o nome da variavel podemos criar um property da seguinte forma:

```
        @property
        def dados(self):
            return self.__dados

        def listar_clientes(self):
            for id,nome in self.__dados.items():
                print(id,nome)

        def apaga_clientes(self,id):
            del self.__dados['Clientes'][id]


BD = BancoDeDados()
BD.inserir_dados(1,'Geronimo')
print(BD.dados)
```