Python - Encapsulamento

O princípio do Encapsulamento é um dos principais pilares em que se baseia o paradigma da programação orientada a objetos. Python adota uma abordagem diferente para a implementação do encapsulamento.

Sabemos que uma classe é um protótipo definido pelo usuário para um objeto. Ele define um conjunto de membros e métodos de dados, capazes de processar os dados. De acordo com o princípio do encapsulamento de dados, os membros dos dados que descrevem um objeto ficam ocultos do ambiente externo à classe. Eles estão disponíveis para processamento apenas para métodos definidos na classe. Os próprios métodos, por outro lado, são acessíveis fora do contexto da classe. Conseqüentemente, diz-se que os dados do objeto são encapsulados pelos métodos. O resultado desse encapsulamento é que qualquer acesso injustificado aos dados do objeto é evitado.

Linguagens como C++ e Java usam modificadores de acesso para restringir o acesso aos membros da classe (isto é, variáveis e métodos). Essas linguagens possuem palavras-chave public, protected e private para especificar o tipo de acesso.

Um membro da classe é considerado público se puder ser acessado de qualquer lugar do programa. Membros privados só podem ser acessados dentro da classe. Normalmente, os métodos são definidos como públicos e as variáveis de instância são privadas. Este arranjo de variáveis de instância privadas e métodos públicos garante a implementação do encapsulamento.

Ao contrário dessas linguagens, Python não tem nenhuma provisão para especificar o tipo de acesso que um membro da classe pode ter. Por padrão, todas as variáveis e métodos em uma classe Python são públicos, conforme demonstrado no exemplo a seguir.

Exemplo 1

Aqui, temos uma classe Employee com variáveis de instância **name** e **age** . Um objeto desta classe possui esses dois atributos. Eles podem ser acessados diretamente de fora da classe, pois são públicos.

```
class Student:
   def __init__(self, name="Rajaram", marks=50):
      self.name = name
      self.marks = marks

s1 = Student()
s2 = Student("Bharat", 25)
```

```
print ("Name: {} marks: {}".format(s1.name, s2.marks))
print ("Name: {} marks: {}".format(s2.name, s2.marks))
```

Ele produzirá a seguinte **saída** -

```
Name: Rajaram marks: 50
Name: Bharat marks: 25
```

No exemplo acima, as variáveis de instância são inicializadas dentro da classe. Porém, não há restrição ao acesso ao valor da variável de instância de fora da classe, o que vai contra o princípio do encapsulamento.

Embora não existam palavras-chave para reforçar a visibilidade, o Python tem uma convenção de nomear as variáveis de instância de uma maneira peculiar. Em Python, prefixar o nome da variável/método com sublinhado simples ou duplo para emular o comportamento de modificadores de acesso protegidos e privados.

Se uma variável for prefixada por um único sublinhado duplo (como " **__age** "), a variável de instância será privada, da mesma forma, se o nome de uma variável for prefixado com um único sublinhado (como " **_salary** ")

Exemplo 2

Vamos modificar a classe Student. Adicione outro salário variável de instância. Torne o nome privado e **marque** -o como privado prefixando-os com sublinhados duplos.

```
class Student:

   def __init__(self, name="Rajaram", marks=50):
      self.__name = name
      self.__marks = marks
   def studentdata(self):
      print ("Name: {} marks: {}".format(self.__name, self.__marks))

s1 = Student()
s2 = Student("Bharat", 25)

s1.studentdata()
s2.studentdata()
print ("Name: {} marks: {}".format(s1.__name, s2.__marks))
print ("Name: {} marks: {}".format(s2.__name, __s2.marks))
```

Quando você executa este código, ele produzirá a seguinte **saída** -

```
Name: Rajaram marks: 50
Name: Bharat marks: 25
Traceback (most recent call last):
 File "C:\Python311\hello.py", line 14, in <module>
  print ("Name: {} marks: {}".format(s1.__name, s2.__marks))
AttributeError: 'Student' object has no attribute '__name'
```

A saída acima deixa claro que as variáveis de instância nome e idade, embora possam ser acessadas por um método declarado dentro da classe (o método studentdata()), mas como o prefixo de sublinhado duplo torna as variáveis privadas e, portanto, acessa-as fora a classe não é permitida, gerando erro de atributo.

Python não bloqueia totalmente o acesso a dados privados. Apenas deixa para a sabedoria do programador não escrever nenhum código que o acesse de fora da classe. Você ainda pode acessar os membros privados pela técnica de manipulação de nomes do Python.

A manipulação de nomes é o processo de alteração do nome de um membro com sublinhado duplo para o formato **object._class__variable** . Se necessário, ainda poderá ser acessado de fora da sala de aula, mas a prática deverá ser evitada.

Em nosso exemplo, a variável de instância privada "__name" é alterada alterando-a para o formato

```
obj._class__privatevar
```

Portanto, para acessar o valor da variável de instância "__marks" do objeto "s1", altere-o para "s1._Student__marks".

Altere a instrução print() no programa acima para -

```
print (s1._Student__marks)
```

Agora imprime 50, as marcas de s1.

Portanto, podemos concluir que Python não implementa encapsulamento exatamente de acordo com a teoria da programação orientada a objetos. Ele adapta uma abordagem mais madura, prescrevendo uma convenção de nomes e permitindo que o programador use a manipulação de nomes se for realmente necessário ter acesso a dados privados no escopo público.