Python - modificadores de acesso

As linguagens como C++ e Java usam modificadores de acesso para restringir o acesso aos membros da classe (ou seja, 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 princípio do encapsulamento.
- **Os membros protegidos** são acessíveis dentro da classe, bem como por classes derivadas dessa classe.

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 de uma classe são públicos.

Exemplo

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

```
class Employee:
   'Common base class for all employees'
   def __init__(self, name="Bhavana", age=24):
      self.name = name
      self.age = age

e1 = Employee()
e2 = Employee("Bharat", 25)

print ("Name: {}".format(e1.name))
print ("age: {}".format(e1.age))
print ("Name: {}".format(e2.name))
print ("age: {}".format(e2.age))
```

Ele produzirá a seguinte **saída** -

```
Name: Bhavana
age: 24
Name: Bharat
age: 25
```

Python não impõe restrições ao acesso a qualquer variável ou método de instância. No entanto, Python prescreve uma convenção de prefixar o nome da variável/método com sublinhado simples ou duplo para emular o comportamento de modificadores de acesso protegidos e privados.

Para indicar que uma variável de instância é privada, prefixe-a com sublinhado duplo (como " **__age** "). Para sugerir que uma determinada variável de instância está protegida, prefixe-a com um único sublinhado (como " **_salary** ")

Exemplo

Vamos modificar a classe Employee. Adicione outro salário variável de instância. Torne **a idade** privada e **o salário** protegidos prefixando sublinhados duplos e simples, respectivamente.

```
class Employee:
   def __init__(self, name, age, salary):
      self.name = name # public variable
      self.__age = age # private variable
      self._salary = salary # protected variable
   def displayEmployee(self):
      print ("Name : ", self.name, ", age: ", self.__age, ", salary: ", self._salary)

e1=Employee("Bhavana", 24, 10000)

print (e1.name)
print (e1._salary)
print (e1.__age)
```

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

```
Bhavana
10000
Traceback (most recent call last):
 File "C:\Users\user\example.py", line 14, in <module>
  print (e1.__age)
       ^^^^^^^^
AttributeError: 'Employee' object has no attribute '__age'
```

Python exibe AttributeError porque __age é privado e não está disponível para uso fora da classe.

Alteração de nome

Python não bloqueia o acesso a dados privados, apenas deixa para a sabedoria do programador não escrever nenhum código que os 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 "__age" do objeto "e1", altere-o para "e1._Employee__age".

Altere a instrução print() no programa acima para -

```
print (e1._Employee__age)
```

Agora imprime 24, a idade de **e1** .

Objeto de propriedade Python

A biblioteca padrão do Python possui uma função property() integrada. Ele retorna um objeto de propriedade. Ele atua como uma interface para as variáveis de instância de uma classe Python.

O princípio de encapsulamento da programação orientada a objetos exige que as variáveis de instância tenham acesso privado restrito. Python não possui mecanismo eficiente para esse propósito. A função property() fornece uma alternativa.

A função property() usa os métodos getter, setter e delete definidos em uma classe para definir um objeto de propriedade para a classe.

Sintaxe

```
property(fget=None, fset=None, fdel=None, doc=None)
```

Parâmetros

- **fget** - um método de instância que recupera o valor de uma variável de instância.
- **fset** - um método de instância que atribui valor a uma variável de instância.
- **fdel** - um método de instância que remove uma variável de instância
- **fdoc** - String de documentação da propriedade.

A função usa métodos getter e setter para retornar o objeto de propriedade.

Métodos getters e setter

Um método getter recupera o valor de uma variável de instância, geralmente denominada get_varname, enquanto o método setter atribui valor a uma variável de instância - denominada set_varname.

Vamos definir os métodos getter get_name() e get_age(), e os setters set_name() e set_age() na classe Employee.

Exemplo

```
class Employee:
   def __init__(self, name, age):
      self.__name = name
      self.__age = age

   def get_name(self):
      return self.__name
   def get_age(self):
      return self.__age
   def set_name(self, name):
      self.__name = name
      return
   def set_age(self, age):
      self.__age=age

e1=Employee("Bhavana", 24)
print ("Name:", e1.get_name(), "age:",

e1.get_age())
e1.set_name("Archana")
e1.set_age(21)
print ("Name:", e1.get_name(), "age:", e1.get_age())
```

Ele produzirá a seguinte **saída** -

```
Name: Bhavana age: 24
Name: Archana age: 21
```

Os métodos getter e setter podem recuperar ou atribuir valor a variáveis de instância. A função property() os utiliza para adicionar objetos de propriedade como atributos de classe.

A propriedade name é definida como -

```
name = property(get_name, set_name, "name")
```

Da mesma forma, você pode adicionar a propriedade **age** -

```
age = property(get_age, set_age, "age")
```

A vantagem do objeto de propriedade é que você pode usá-lo para recuperar o valor de sua variável de instância associada, bem como atribuir valor.

Por exemplo,

```
print (e1.name) displays value of e1.__name
e1.name = "Archana" assigns value to e1.__age
```

Exemplo

O programa completo com objetos de propriedade e seu uso é fornecido abaixo -

```
class Employee:
   def __init__(self, name, age):
      self.__name = name
      self.__age = age

   def get_name(self):
      return self.__name
   def get_age(self):
      return self.__age
   def set_name(self, name):
      self.__name = name
      return
   def set_age(self, age):
      self.__age=age
      return
   name = property(get_name, set_name, "name")
   age = property(get_age, set_age, "age")
```

```
e1=Employee("Bhavana", 24)
print ("Name:", e1.name, "age:", e1.age)

e1.name = "Archana"
e1.age = 23
print ("Name:", e1.name, "age:", e1.age)
```

Ele produzirá a seguinte **saída** -

```
Name: Bhavana age: 24
Name: Archana age: 23
```