Python - Construtores

Na programação orientada a objetos, um objeto de uma classe é caracterizado por uma ou mais variáveis ou atributos de instância, cujos valores são únicos para cada objeto. Por exemplo, se a classe Employee tiver um atributo de instância como nome. Cada um de seus objetos e1 e e2 pode ter valores diferentes para a variável nome.

Um construtor é um método de instância em uma classe, que é chamado automaticamente sempre que um novo objeto da classe é declarado. A função do construtor é atribuir valor às variáveis de instância assim que o objeto for declarado.

Python usa um método especial chamado __init__() para inicializar as variáveis de instância do objeto, assim que ele é declarado.

O método __init__() atua como um construtor. Precisa de um argumento obrigatório self, que é a referência ao objeto.

```
def __init__(self):
#initialize instance variables
```

O método __init__(), bem como qualquer método de instância em uma classe, possui um parâmetro obrigatório, **self** . No entanto, você pode dar qualquer nome ao primeiro parâmetro, não necessariamente self.

Vamos definir o construtor na classe Employee para inicializar nome e idade como variáveis de instância. Podemos então acessar esses atributos de seu objeto.

Exemplo

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

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

Ele produzirá a seguinte **saída** -

```
Name: Bhavana
```

```
age: 24
```

Construtor parametrizado

Para a classe Employee acima, cada objeto que declaramos terá o mesmo valor para suas variáveis de instância nome e idade. Para declarar objetos com atributos variados em vez do padrão, defina argumentos para o método __init__(). (Um método nada mais é do que uma função definida dentro de uma classe.)

Exemplo

Neste exemplo, o construtor __init__() possui dois argumentos formais. Declaramos objetos Employee com valores diferentes -

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

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

Você pode atribuir padrões aos argumentos formais no construtor para que o objeto possa ser instanciado com ou sem passagem de parâmetros.

```
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 - Métodos de Instância

Além do construtor __init__(), pode haver um ou mais métodos de instância definidos em uma classe. Um método com self como um dos argumentos formais é chamado de método de instância, pois é chamado por um objeto específico.

Exemplo

No exemplo a seguir, um método displayEmployee() foi definido. Ele retorna os atributos name e age do objeto Employee que chama o método.

```
class Employee:
   def __init__(self, name="Bhavana", age=24):
      self.name = name
      self.age = age
   def displayEmployee(self):
      print ("Name : ", self.name, ", age: ", self.age)

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

e1.displayEmployee()
e2.displayEmployee()
```

Ele produzirá a seguinte **saída** -

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

Você pode adicionar, remover ou modificar atributos de classes e objetos a qualquer momento -

```
emp1.salary = 7000 # Add a 'salary' attribute.
emp1.name = 'xyz' # Modify 'name' attribute.
del emp1.salary # Delete 'salary' attribute.
```

Em vez de usar instruções normais para acessar atributos, você pode usar as seguintes funções -

- O **getattr(obj, name[, default])** - para acessar o atributo do objeto.
- O **hasattr(obj,name)** - para verificar se um atributo existe ou não.
- O **setattr(obj,name,value)** - para definir um atributo. Se o atributo não existir, ele será criado.
- O **delattr(obj, name)** - para excluir um atributo.

```
print (hasattr(e1, 'salary')) # Returns true if 'salary' attribute exists
print (getattr(e1, 'name')) # Returns value of 'name' attribute
setattr(e1, 'salary', 7000) # Set attribute 'salary' at 8
delattr(e1, 'age') # Delete attribute 'age'
```

Ele produzirá a seguinte **saída** -

```
False
Bhavana
```