Python - Reflexão

Na programação orientada a objetos, reflexão refere-se à capacidade de extrair informações sobre qualquer objeto em uso. Você pode conhecer o tipo do objeto, se é uma subclasse de alguma outra classe, quais são seus atributos e muito mais. A biblioteca padrão do Python possui diversas funções que refletem nas diferentes propriedades de um objeto. A reflexão às vezes também é chamada de introspecção.

Vamos fazer uma revisão das funções de reflexão.

A função type()

Já usamos essa função muitas vezes. Ele informa a qual classe um objeto pertence.

Exemplo

As instruções a seguir imprimem a respectiva classe de diferentes objetos de tipo de dados integrados

```
print (type(10))
print (type(2.56))
print (type(2+3j))
print (type("Hello World"))
print (type([1,2,3]))
print (type({1:'one', 2:'two'}))
```

Aqui, você obterá a seguinte **saída** -

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'list'>
<class 'dict'>
```

Vamos verificar o tipo de um objeto de uma classe definida pelo usuário -

```
class test:
   pass
```

```
obj = test()
print (type(obj))
```

Ele produzirá a seguinte **saída** -

```
<class '__main__.test'>
```

A função isinstance()

Esta é outra função integrada em Python que verifica se um objeto é uma instância de uma determinada classe

Sintaxe

```
isinstance(obj, class)
```

Esta função sempre retorna um valor booleano, verdadeiro se o objeto realmente pertence à classe fornecida e falso se não.

Exemplo

As instruções a seguir retornam True -

```
print (isinstance(10, int))
print (isinstance(2.56, float))
print (isinstance(2+3j, complex))
print (isinstance("Hello World", str))
```

Em contraste, essas declarações imprimem False.

```
print (isinstance([1,2,3], tuple))
print (isinstance({1:'one', 2:'two'}, set))
```

Ele produzirá a seguinte **saída** -

```
True
True
True
True
False
False
```

Você também pode realizar a verificação com uma classe definida pelo usuário

```
class test:
   pass

obj = test()
print (isinstance(obj, test))
```

Ele produzirá a seguinte **saída** -

```
True
```

Em Python, até as classes são objetos. Todas as classes são objetos da classe de objeto. Isso pode ser verificado seguindo o código -

```
class test:
   pass

print (isinstance(int, object))
print (isinstance(str, object))
print (isinstance(test, object))
```

Todas as instruções de impressão acima imprimem True.

A função issubclass()

Esta função verifica se uma classe é uma subclasse de outra classe. Refere-se às classes, não às suas instâncias.

Conforme mencionado anteriormente, todas as classes Python são subclasses da classe de objeto. Conseqüentemente, a saída das seguintes instruções de impressão é verdadeira para todos.

```
class test:
   pass

print (issubclass(int, object))
print (issubclass(str, object))
print (issubclass(test, object))
```

Ele produzirá a seguinte **saída** -

```
True
True
True
```

A função callable()

Um objeto pode ser chamado se invocar um determinado processo. Uma função Python, que executa um determinado processo, é um objeto que pode ser chamado. Portanto, callable(function) retorna True. Qualquer função, interna, definida pelo usuário ou método pode ser chamada. Objetos de tipos de dados integrados, como int, str, etc., não podem ser chamados.

Exemplo

```
def test():
   pass

print (callable("Hello"))
print (callable(abs))
print (callable(list.clear([1,2])))
print (callable(test))
```

Um objeto string não pode ser chamado. Mas abs é uma função que pode ser chamada. O método pop de list pode ser chamado, mas clear() é na verdade uma chamada para a função e não um objeto de função, portanto, não pode ser chamado

Ele produzirá a seguinte **saída** -

```
False
True
True
False
True
```

Uma instância de classe pode ser chamada se tiver um método __call__(). No exemplo abaixo, a classe de teste inclui o método __call__(). Portanto, seu objeto pode ser usado como se estivéssemos chamando uma função. Portanto, o objeto de uma classe com a função __call__() pode ser chamado.

```
class test:
   def __init__(self):
      pass
   def __call__(self):
      print ("Hello")

obj = test()
```

```
obj()
print ("obj is callable?", callable(obj))
```

Ele produzirá a seguinte **saída** -

```
Hello
obj is callable? True
```

A função getattr()

A função integrada getattr() recupera o valor do atributo nomeado do objeto.

Exemplo

```
class test:
   def __init__(self):
      self.name = "Manav"

obj = test()
print (getattr(obj, "name"))
```

Ele produzirá a seguinte **saída** -

```
Manav
```

A função setattr()

A função integrada setattr() adiciona um novo atributo ao objeto e atribui um valor a ele. Também pode alterar o valor de um atributo existente.

No exemplo abaixo, o objeto da classe de teste possui um único atributo - nome. Usamos setattr para adicionar o atributo idade e modificar o valor do atributo name.

```
class test:
   def __init__(self):
      self.name = "Manav"

obj = test()
setattr(obj, "age", 20)
setattr(obj, "name", "Madhav")
print (obj.name, obj.age)
```

Ele produzirá a seguinte **saída** -

```
Madhav 20
```

A função hasattr()

Esta função integrada retorna True se o atributo fornecido estiver disponível para o argumento do objeto e false se não estiver. Usamos a mesma classe de teste e verificamos se ela possui um determinado atributo ou não.

```
class test:
   def __init__(self):
      self.name = "Manav"

obj = test()
print (hasattr(obj, "age"))
print (hasattr(obj, "name"))
```

Ele produzirá a seguinte **saída** -

```
False
True
```

A função dir()

Se sua função interna for chamada sem argumento, retorne os nomes no escopo atual. Para qualquer objeto como argumento, ele retorna uma lista dos atributos do objeto fornecido e dos atributos acessíveis a partir dele.

- **Para um objeto de módulo** - a função retorna os atributos do módulo.
- **Para um objeto de classe** - a função retorna seus atributos e recursivamente os atributos de suas bases.
- **Para qualquer outro objeto** - seus atributos, os atributos de sua classe e recursivamente os atributos das classes base de sua classe.

Exemplo

```
print ("dir(int):", dir(int))
```

Ele produzirá a seguinte **saída** -

```
dir(int): ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__
```

Exemplo

```
print ("dir(dict):", dir(dict))
```

Ele produzirá a seguinte **saída** -

```
dir(dict): ['__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
```

Exemplo

```
class test:
   def __init__(self):
      self.name = "Manav"

obj = test()
print ("dir(obj):", dir(obj))
```

Ele produzirá a seguinte **saída** -

```
dir(obj): ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge_
```