Python - Módulos

Uma função é um bloco de código organizado e reutilizável usado para executar uma única ação relacionada. As funções fornecem melhor modularidade para seu aplicativo e um alto grau de reutilização de código.

Módulos Python

O conceito de **módulo** em Python aumenta ainda mais a modularidade. Você pode definir mais de uma função relacionada juntas e carregar as funções necessárias. Um módulo é um arquivo que contém definição de funções, classes , variáveis , constantes ou qualquer outro objeto Python. O conteúdo deste arquivo pode ser disponibilizado para qualquer outro programa. Python possui a palavra-chave **import** para essa finalidade.

Exemplo de módulo Python

```
import math
print ("Square root of 100:", math.sqrt(100))
```

Ele produzirá a seguinte **saída** -

```
Square root of 100: 10.0
```

Módulos integrados do Python

A biblioteca padrão do Python vem com um grande número de módulos. Eles são chamados de módulos integrados. A maioria desses módulos integrados são escritos em C (já que a implementação de referência do Python está em C) e pré-compilados na biblioteca. Esses módulos incluem funcionalidades úteis, como gerenciamento de sistema operacional específico do sistema, E/S de disco, rede, etc.

Aqui está uma lista selecionada de módulos integrados -

Sr. Não.	Nome e breve descrição
1	**sistema operacional** Este módulo fornece uma interface unificada para diversas funções do sistema operacional.
2	**corda**

	Este módulo contém uma série de funções para processamento de strings
3	**ré** Este módulo fornece um conjunto de recursos poderosos de expressões regulares. Expressão regular (RegEx), permite pesquisa poderosa de string e correspondência para um padrão em uma string
4	**matemática** Este módulo implementa uma série de operações matemáticas para números de ponto flutuante. Essas funções geralmente são wrappers finos em torno das funções da biblioteca da plataforma C.
5	**cmath** Este módulo contém uma série de operações matemáticas para números complexos.
6	**data hora** Este módulo fornece funções para lidar com datas e horas dentro de um dia. Ele envolve a biblioteca de tempo de execução C.
7	**gc** Este módulo fornece uma interface para o coletor de lixo integrado.
8	**assíncio** Este módulo define a funcionalidade necessária para processamento assíncrono
9	**Coleções** Este módulo fornece tipos de dados avançados de contêiner.
10	**Ferramentas funcionais** Este módulo possui funções e operações de ordem superior em objetos que podem ser chamados. Útil em programação funcional
11	**operador** Funções correspondentes aos operadores padrão.
12	**salmoura** Converta objetos Python em fluxos de bytes e vice-versa.
13	**tomada** Interface de rede de baixo nível.
14	**sqlite3** Uma implementação DB-API 2.0 usando SQLite 3.x.
15	**Estatisticas** Funções de estatística matemática

16	**digitando** Suporte para dicas de tipo
17	**venv** Criação de ambientes virtuais.
18	**JSON** Codifique e decodifique o formato JSON.
19	**wsgiref** Utilitários WSGI e implementação de referência.
20	**teste de unidade** Estrutura de testes unitários para Python.
21	**aleatório** Gerar números pseudo-aleatórios

Módulos definidos pelo usuário Pythin

Qualquer arquivo de texto com extensão .py e contendo código Python é basicamente um módulo. Pode conter definições de uma ou mais funções, variáveis, constantes e também classes. Qualquer objeto Python de um módulo pode ser disponibilizado para a sessão do interpretador ou outro script Python por meio da instrução import. Um módulo também pode incluir código executável.

Criando um Módulo Python

Criar um módulo nada mais é do que salvar um código Python com a ajuda de qualquer editor. Vamos salvar o seguinte código como **mymodule.py**

```
def SayHello(name):
   print ("Hi {}! How are you?".format(name))
   return
```

Agora você pode importar mymodule no terminal Python atual.

```
>>> import mymodule
>>> mymodule.SayHello("Harish")
Hi Harish! How are you?
```

Você também pode importar um módulo em outro script Python. Salve o código a seguir como exemplo.py

```
import mymodule
mymodule.SayHello("Harish")
```

Execute este script no terminal de comando

```
C:\Users\user\examples> python example.py
Hi Harish! How are you?
```


A declaração de importação

Em Python, a palavra-chave **import** foi fornecida para carregar um objeto Python de um módulo. O objeto pode ser uma função, classe, variável etc. Se um módulo contiver múltiplas definições, todas elas serão carregadas no namespace.

Vamos salvar o código a seguir com três funções como **mymodule.py.**

```
def sum(x,y):
   return x+y

def average(x,y):
   return (x+y)/2

def power(x,y):
   return x**y
```

A instrução **import mymodule** carrega todas as funções deste módulo no namespace atual. Cada função no módulo importado é um atributo deste objeto de módulo.

```
>>> dir(mymodule)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__pa
```

Para chamar qualquer função, use a referência do objeto módulo. Por exemplo, meumódulo.sum().

```
import mymodule
print ("sum:",mymodule.sum(10,20))
print ("average:",mymodule.average(10,20))
print ("power:",mymodule.power(10, 2))
```

Ele produzirá a seguinte **saída** -

```
sum:30
average:15.0
power:100
```


A instrução from ... import

A instrução import carregará todos os recursos do módulo no namespace atual. É possível importar objetos específicos de um módulo usando esta sintaxe. Por exemplo -

Das três funções em **mymodule** , apenas duas são importadas no seguinte script executável **example.py**

```
from mymodule import sum, average
print ("sum:",sum(10,20))
print ("average:",average(10,20))
```

Ele produzirá a seguinte saída -

```
sum: 30
average: 15.0
```

Observe que a função não precisa ser chamada prefixando o nome de seu módulo.

A declaração from...import *

Também é possível importar todos os nomes de um módulo para o namespace atual usando a seguinte instrução de importação -

```
from modname import *
```

Isto fornece uma maneira fácil de importar todos os itens de um módulo para o namespace atual; no entanto, esta afirmação deve ser usada com moderação.

A importação ... como declaração

Você pode atribuir um nome alternativo ao módulo importado.

```
from modulename as alias
```

O **alias** deve ser prefixado à função durante a chamada.

Dê uma olhada no **exemplo** a seguir -

```
import mymodule as x
print ("sum:",x.sum(10,20))
print ("average:", x.average(10,20))
print ("power:", x.power(10, 2))
```

Atributos do módulo

Em Python, um módulo é um objeto da classe de módulo e, portanto, é caracterizado por atributos.

A seguir estão os atributos do módulo -

- __file__ retorna o nome físico do módulo.
- __package__ retorna o pacote ao qual o módulo pertence.
- __doc__ retorna a docstring no topo do módulo, se houver
- __dict__ retorna todo o escopo do módulo
- __name__ retorna o nome do módulo

Exemplo

Supondo que o código a seguir seja salvo como **mymodule.py**

```
"The docstring of mymodule"
def sum(x,y):
   return x+y

def average(x,y):
   return (x+y)/2

def power(x,y):
   return x**y
```

Vamos verificar os atributos de mymodule importando-o no seguinte script -

```
import mymodule

print ("__file__ attribute:", mymodule.__file__)
print ("__doc__ attribute:", mymodule.__doc__)
print ("__name__ attribute:", mymodule.__name__)
```

Ele produzirá a seguinte **saída** -

```
__file__ attribute: C:\Users\mlath\examples\mymodule.py
__doc__ attribute: The docstring of mymodule
__name__ attribute: mymodule
```

O __name__Attribute

O atributo __name__ de um módulo Python tem grande significado. Vamos explorá-lo com mais detalhes.

Em um shell interativo, o atributo __name__ retorna '__main__'

```
>>> __name__
'__main__'
```

Se você importar qualquer módulo na sessão do interpretador, ele retornará o nome do módulo como o atributo __name__ desse módulo.

```
>>> import math
>>> math.__name__
'math'
```

De dentro de um script Python, o atributo __name__ retorna '__main__'

```
#example.py
print ("__name__ attribute within a script:", __name__)
```

Execute isto no terminal de comando -

```
__name__ attribute within a script: __main__
```

Este atributo permite que um script Python seja usado como executável ou como módulo. Ao contrário de C++, Java, C# etc., em Python, não existe o conceito da função main(). O script do programa Python com extensão .py pode conter definições de funções, bem como instruções executáveis.

Salve **mymodule.py** e com o seguinte código -

```
"The docstring of mymodule"
def sum(x,y):
   return x+y

print ("sum:",sum(10,20))
```

Você pode ver que a função sum() é chamada no mesmo script em que está definida.

```
C:\Users\user\examples> python mymodule.py
sum: 30
```

Agora vamos importar esta função em outro script **example.py** .

```
import mymodule
print ("sum:",mymodule.sum(10,20))
```

Ele produzirá a seguinte **saída** -

```
C:\Users\user\examples> python example.py
sum: 30
sum: 30
```

A saída "soma:30" aparece duas vezes. Uma vez quando o módulo mymodule é importado. As instruções executáveis no módulo importado também são executadas. A segunda saída vem do script de chamada, ou seja, programa **example.py .**

O que queremos que aconteça é que quando um módulo for importado, apenas a função seja importada, suas instruções executáveis não sejam executadas. Isso pode ser feito verificando o valor de __name__. Se for __main__, significa que está sendo executado e não importado. Inclua condicionalmente as instruções executáveis, como chamadas de função.

Adicione a instrução **if** em **mymodule.py** conforme mostrado -

```
"The docstring of mymodule"
def sum(x,y):
   return x+y

if __name__ == "__main__":
   print ("sum:",sum(10,20))
```

Agora, se você executar o programa **example.py** , descobrirá que a saída sum:30 aparece apenas uma vez.

```
C:\Users\user\examples> python example.py
sum: 30
```

A função recarregar()

Às vezes você pode precisar recarregar um módulo, especialmente ao trabalhar com a sessão do interpretador interativo do Python.

Suponha que temos um módulo de teste (test.py) com a seguinte função -

```
def SayHello(name):
   print ("Hi {}! How are you?".format(name))
   return
```

Podemos importar o módulo e chamar sua função no prompt do Python como -

```
>>> import test
>>> test.SayHello("Deepak")
Hi Deepak! How are you?
```

No entanto, suponha que você precise modificar a função SayHello(), como -

```
def SayHello(name, course):
   print ("Hi {}! How are you?".format(name))
   print ("Welcome to {} Tutorial by TutorialsPoint".format(course))
   return
```

Mesmo se você editar o arquivo test.py e salvá-lo, a função carregada na memória não será atualizada. Você precisa recarregá-lo, usando a função reload() no módulo imp.

```
>>> import imp
>>> imp.reload(test)
>>> test.SayHello("Deepak", "Python")
Hi Deepak! How are you?
Welcome to Python Tutorial by TutorialsPoint
```