Python-JSON

JSON significa notação de objeto JavaScript. É um formato leve de intercâmbio de dados. É semelhante ao picles. No entanto, a serialização pickle é específica do Python, enquanto o formato JSON é implementado por muitas linguagens. O módulo json na biblioteca padrão do Python implementa funcionalidade de serialização de objetos que é semelhante aos módulos pickle e marshal.

Assim como no módulo pickle, o módulo json também fornece funções dumps() e load() para serialização de objetos Python em string codificada JSON, e as funções dump() e load() escrevem e leem objetos Python serializados de/para o arquivo.

- **dumps()** - Esta função converte o objeto para o formato JSON.
- **load()** - Esta função converte uma string JSON de volta em um objeto Python.

O exemplo a seguir demonstra o uso básico dessas funções -

Exemplo 1

```
import json

data=['Rakesh',{'marks':(50,60,70)}]
s=json.dumps(data)
print (s, type(s))

data = json.loads(s)
print (data, type(data))
```

Ele produzirá a seguinte **saída** -

```
["Rakesh", {"marks": [50, 60, 70]}] <class 'str'>
['Rakesh', {'marks': [50, 60, 70]}] <class 'list'>
```

A função dumps() pode receber o argumento sort_keys opcional. Por padrão é falso. Se definido como True, as chaves do dicionário aparecerão em ordem de classificação na string JSON.

```
data=['Rakesh',{'marks':(50,60,70)}]
s=json.dumps(data, sort_keys=True)
```

Exemplo 2

A função dumps() possui outro parâmetro opcional chamado **indent** que leva um número como valor. Ele decide o comprimento de cada segmento da representação formatada da string json, semelhante à saída pprint.

```
import json
data=['Rakesh',{'marks':(50,60,70)}]
s=json.dumps(data, indent = 2)
print (s)
```

Ele produzirá a seguinte saída -

```
[
  "Rakesh",
  {
    "marks": [
      50,
      60,
      70
    ]
  }
]
```

O módulo json também possui API orientada a objetos correspondente às funções acima. Existem duas classes definidas no módulo - JSONEncoder e JSONDecoder.

Classe JSONEncoder

O objeto desta classe é o codificador para estruturas de dados Python. Cada tipo de dados Python é convertido no tipo JSON correspondente, conforme mostrado na tabela a seguir -

Pitão	JSON
Ditado	objeto
lista, tupla	variedade
Str.	corda
Enums derivados de int, float, int e float	número
Verdadeiro	verdadeiro
Falso	falso
Nenhum	nulo

A classe JSONEncoder é instanciada pelo construtor JSONEncoder(). Os seguintes métodos importantes são definidos na classe do codificador -

- **encode()** - serializa o objeto Python no formato JSON.
- **iterencode()** - Codifica o objeto e retorna um iterador produzindo a forma codificada de cada item no objeto.
- **indent** - Determina o nível de recuo da string codificada.
- **sort_keys** - é verdadeiro ou falso para fazer as chaves aparecerem em ordem de classificação ou não.
- **check_circular** - se True, verifica a referência circular no objeto do tipo contêiner.

O exemplo a seguir codifica o objeto de lista Python.

Exemplo

```
import json

data=['Rakesh',{'marks':(50,60,70)}]
e=json.JSONEncoder()
```

Usando o método iterencode(), cada parte da string codificada é exibida conforme abaixo -

```
import json
data=['Rakesh',{'marks':(50,60,70)}]
e=json.JSONEncoder()
for obj in e.iterencode(data):
   print (obj)
```

Ele produzirá a seguinte saída -

```
["Rakesh"
,
{
"marks"
:
[50
, 60
, 70
]
}
]
```

Classe JSONDEcoder

O objeto desta classe ajuda na decodificação em string json de volta para a estrutura de dados Python. O método principal nesta classe é decode(). O código de exemplo a seguir recupera o objeto de lista Python da string codificada na etapa anterior.

Exemplo

```
import json
data=['Rakesh',{'marks':(50,60,70)}]
e=json.JSONEncoder()
s = e.encode(data)
d=json.JSONDecoder()
obj = d.decode(s)
print (obj, type(obj))
```

Ele produzirá a seguinte **saída** -

```
['Rakesh', {'marks': [50, 60, 70]}] <class 'list'>
```

JSON com arquivos/streams

O módulo json define as funções load() e dump() para gravar dados JSON em um arquivo como um objeto - que pode ser um arquivo de disco ou um fluxo de bytes e ler os dados deles.

Função dump()

Esta função codifica dados de objetos Python no formato JSON e os grava em um arquivo. O arquivo deve ter permissão de gravação.

Exemplo

```
import json
data=['Rakesh', {'marks': (50, 60, 70)}]
fp=open('json.txt','w')
json.dump(data,fp)
fp.close()
```

Este código criará 'json.txt' no diretório atual. Ele mostra o conteúdo da seguinte forma -

```
["Rakesh", {"marks": [50, 60, 70]}]
```

Função carregar()

Esta função carrega dados JSON do arquivo e constrói o objeto Python a partir dele. O arquivo deve ser aberto com permissão de leitura.

Exemplo

```
import json
fp=open('json.txt','r')
ret=json.load(fp)
print (ret)
```