Python - Processamento de URL

No mundo da Internet, diferentes recursos são identificados por URLs (Uniform Resource Locators). O pacote urllib que acompanha a biblioteca padrão do Python fornece vários utilitários para lidar com URLs. Possui os seguintes módulos -

- **O módulo urllib.parse** é usado para analisar um URL em suas partes.
- **O módulo urllib.request** contém funções para abrir e ler URLs
- **O módulo urllib.error** carrega definições das exceções levantadas por urllib.request
- **O módulo urllib.robotparser** analisa os arquivos robots.txt

O módulo urllib.parse

Este módulo serve como uma interface padrão para obter várias partes de uma string de URL. O módulo contém as seguintes funções -

urlparse(urlstring)

Analise uma URL em seis componentes, retornando uma tupla chamada de 6 itens. Cada item da tupla é uma string correspondente aos seguintes atributos -

Atributo	Índice	Valor
esquema	0	Especificador de esquema de URL
netloc	1	Parte de localização de rede
caminho	2	Caminho hierárquico
parâmetros	3	Parâmetros para o último elemento do caminho
consulta	4	Componente de consulta
fragmento	5	Identificador de fragmento
nome de usuário		Nome de usuário
senha		Senha
nome de anfitrião		Nome do host (minúsculas)

Porta		Número da porta como número inteiro, se presente

Exemplo

```
from urllib.parse import urlparse
url = "https://example.com/employees/name/?salary>=25000"
parsed_url = urlparse(url)
print (type(parsed_url))
print ("Scheme:",parsed_url.scheme)
print ("netloc:", parsed_url.netloc)
print ("path:", parsed_url.path)
print ("params:", parsed_url.params)
print ("Query string:", parsed_url.query)
print ("Frgment:", parsed_url.fragment)
```

Ele produzirá a seguinte **saída** -

```
<class 'urllib.parse.ParseResult'>
Scheme: https
netloc: example.com
path: /employees/name/
params:
Query string: salary>=25000
Frgment:
```

parse_qs(qs))

Esta função analisa uma string de consulta fornecida como um argumento de string. Os dados são retornados como um dicionário. As chaves do dicionário são os nomes exclusivos das variáveis de consulta e os valores são listas de valores para cada nome.

Para buscar ainda mais os parâmetros de consulta da string de consulta em um dicionário, use a função parse_qs() do atributo de consulta do objeto ParseResult da seguinte forma -

Exemplo

```
from urllib.parse import urlparse, parse_qs
url = "https://example.com/employees?name=Anand&salary=25000"
parsed_url = urlparse(url)
dct = parse_qs(parsed_url.query)
print ("Query parameters:", dct)
```

Ele produzirá a seguinte **saída** -

```
Query parameters: {'name': ['Anand'], 'salary': ['25000']}
```

urlsplit(urlstring)

Isso é semelhante a urlparse(), mas não separa os parâmetros da URL. Geralmente, isso deve ser usado em vez de urlparse() se for desejada a sintaxe de URL mais recente, permitindo que parâmetros sejam aplicados a cada segmento da parte do caminho da URL.

urlunparse(partes)

Esta função é o oposto da função urlparse(). Ele constrói uma URL a partir de uma tupla retornada por urlparse(). O argumento parts pode ser qualquer iterável de seis itens. Isso retorna um URL equivalente.

Exemplo

```
from urllib.parse import urlunparse

lst = ['https', 'example.com', '/employees/name/', '', 'salary>=25000', '']
new_url = urlunparse(lst)
print ("URL:", new_url)
```

Ele produzirá a seguinte **saída** -

```
URL: https://example.com/employees/name/?salary>=25000
```

urlunsplit(partes)

Combine os elementos de uma tupla retornados por urlsplit() em uma URL completa como uma string. O argumento parts pode ser qualquer iterável de cinco itens.

O módulo urllib.request

Este módulo define funções e classes que auxiliam na abertura de URLs

função urlopen()

Esta função abre o URL fornecido, que pode ser uma string ou um objeto Request. O parâmetro opcional timeout especifica um tempo limite em segundos para operações de bloqueio. Na verdade, isso só funciona para conexões HTTP, HTTPS e FTP.

Esta função sempre retorna um objeto que pode funcionar como gerenciador de contexto e possui as propriedades url, cabeçalhos e status.

Para URLs HTTP e HTTPS, esta função retorna um objeto http.client.HTTPResponse ligeiramente modificado.

Exemplo

O código a seguir usa a função urlopen() para ler os dados binários de um arquivo de imagem e gravá-los no arquivo local. Você pode abrir o arquivo de imagem em seu computador usando qualquer visualizador de imagens.

```
from urllib.request import urlopen
obj = urlopen("https://www.tutorialspoint.com/static/images/simply-easy-learning.jpg"
data = obj.read()
img = open("img.jpg", "wb")
img.write(data)
img.close()
```

Ele produzirá a seguinte **saída** -

O objeto de solicitação

O módulo urllib.request inclui a classe Request. Esta classe é uma abstração de uma solicitação de URL. O construtor requer um argumento de string obrigatório e um URL válido.

Sintaxe

```
urllib.request.Request(url, data, headers, origin_req_host, method=None)
```

Parâmetros

- **url** - Uma string que é um URL válido

- **data** - Um objeto que especifica dados adicionais para enviar ao servidor. Este parâmetro só pode ser usado com solicitações HTTP. Os dados podem ser bytes, objetos semelhantes a arquivos e iteráveis de objetos semelhantes a bytes.
- **headers** - Deve ser um dicionário de cabeçalhos e seus valores associados.
- **origin_req_host** - Deve ser o host de solicitação da transação de origem
- **method** - deve ser uma string que indica o método de solicitação HTTP. Um dos verbos GET, POST, PUT, DELETE e outros HTTP. O padrão é GET.

Exemplo

```
from urllib.request import Request
obj = Request("https://www.tutorialspoint.com/")
```

Este objeto Request agora pode ser usado como argumento para o método urlopen().

```
from urllib.request import Request, urlopen
obj = Request("https://www.tutorialspoint.com/")
resp = urlopen(obj)
```

A função urlopen() retorna um objeto HttpResponse. Chamar seu método read() busca o recurso na URL fornecida.

```
from urllib.request import Request, urlopen
obj = Request("https://www.tutorialspoint.com/")
resp = urlopen(obj)
data = resp.read()
print (data)
```

Envio de dados

Se você definir o argumento data para o construtor Request, uma solicitação POST será enviada ao servidor. Os dados devem ser qualquer objeto representado em bytes.

Exemplo

```
from urllib.request import Request, urlopen
from urllib.parse import urlencode

values = {'name': 'Madhu',
   'location': 'India',
   'language': 'Hindi' }
```

```
data = urlencode(values).encode('utf-8')
obj = Request("https://example.com", data)
```

Enviando cabeçalhos

O construtor Request também aceita argumento de cabeçalho para enviar informações de cabeçalho para a solicitação. Deve estar em um objeto de dicionário.

```
headers = {'User-Agent': user_agent}
obj = Request("https://example.com", data, headers)
```

O módulo urllib.error

As seguintes exceções são definidas no módulo urllib.error -

URLError

URLError é gerado porque não há conexão de rede (nenhuma rota para o servidor especificado) ou o servidor especificado não existe. Neste caso, a exceção levantada terá um atributo 'motivo'.

Exemplo

```
from urllib.request import Request, urlopen
import urllib.error as err

obj = Request("http://www.nosuchserver.com")
try:
   urlopen(obj)
except err.URLError as e:
   print(e)
```

Ele produzirá a seguinte **saída** -

```
HTTP Error 403: Forbidden
```

Erro HTTP

Cada vez que o servidor envia uma resposta HTTP ela é associada a um "código de status" numérico. Seu código indica por que o servidor não consegue atender à solicitação. Os manipuladores padrão tratarão algumas dessas respostas para você. Para aqueles que não conseguem controlar, a função urlopen() gera um HTTPError. Exemplos típicos de HTTPErrors

são '404' (página não encontrada), '403' (solicitação proibida) e '401' (autenticação necessária).

Exemplo

```
from urllib.request import Request, urlopen
import urllib.error as err

obj = Request("http://www.python.org/fish.html")
try:
   urlopen(obj)
except err.HTTPError as e:
   print(e.code)
```

Ele produzirá a seguinte **saída** -

```
404
```