EBook Gratis

APRENDIZAJE python-requests

Free unaffiliated eBook created from
Stack Overflow contributors.

#python-requests

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: python-requests

It is an unofficial and free python-requests ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official python-requests.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Comenzando con las solicitudes de python

Observaciones

HTTP para humanos

Requests es la única biblioteca HTTP sin OMG para Python, segura para el consumo humano.

Las solicitudes le permiten enviar solicitudes `HTTP/1.1` orgánicas y alimentadas con pasto, sin la necesidad de trabajo manual. No hay necesidad de agregar manualmente cadenas de consulta a sus URL, o de codificar de forma sus datos POST. Keep-alive y la agrupación de conexiones HTTP son 100% automáticas, impulsadas por `urllib3` , que está incrustada dentro de las solicitudes.

El poder de las solicitudes:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type":"User"...'
>>> r.json()
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

Examples

Instalación o configuración

`python-requests` está disponible en PyPI, el Índice de Paquetes de Python, lo que significa que se puede instalar a través de pip:

```
pip install requests
```

Puede encontrar el código fuente actualizado en el repositorio de solicitudes de GitHub

Si desea instalarlo desde la fuente, puede hacerlo clonando el repositorio de GitHub:

```
git clone git://github.com/kennethreitz/requests.git
```

O obteniendo el archivo comprimido (`-O` escribe la salida en el archivo; `-L` sigue las

redirecciones):

```
curl -OL https://github.com/kennethreitz/requests/tarball/master
```

Luego puedes instalarlo ejecutando `setup.py`

```
python setup.py install
```

Sin embargo, lo instaló, puede comenzar a usarlo importando de la manera habitual.

```
>>> import requests
>>> requests.get('http://stackoverflow.com')
```

Peticiones GET

`requests.get()` crea una solicitud GET:

```
response = requests.get('https://example.com/')
```

Pase los parámetros de consulta como un diccionario al argumento `params` :

```
response = requests.get('https://example.com/', params={"a": 1, "b": 2})
```

Para solicitudes GET que puedan requerir autenticación básica, puede incluir el `auth` autenticación de la siguiente manera:

```
response = requests.get('https://api.github.com/user', auth=('user', 'pass'))
```

Solicitudes POST

Las solicitudes POST se realizan con el método `request.post()` .

Si necesita enviar una solicitud de formulario web como un cuerpo POST, pase un diccionario con pares clave-valor como argumento de `data` ; `requests` se codificarán en un cuerpo de tipo de `application/x-www-form-urlencoded` :

```
r = requests.post('https://github.com/', data={"a": 1, "b": 2})
```

Si necesita POSTAR una carga json, puede usar `json=` . Esto establecerá automáticamente el encabezado Content-Type en `application/json`

```
r = requests.post('https://github.com/', data={"a": 1, "b": 2})
```

Otros métodos de solicitud

El módulo de `requests` tiene funciones de nivel superior para la mayoría de los métodos HTTP:

```
r = requests.put('https://example.com/', data=put_body)
r = requests.delete('https://example.com/')
r = requests.head('https://example.com/')
r = requests.options('https://example.com/')
r = requests.patch('https://example.com/', data=patch_update)
```

Leyendo la respuesta

```
response = requests.get("https://api.github.com/events")
text_resp = response.text
```

Respuesta JSON : para respuestas con formato json, el paquete proporciona un decodificador incorporado

```
response = requests.get('https://api.github.com/events')
json_resp = response.json()
```

Este método generará un `ValueError` en caso de respuesta vacía o contenido no analizable.

Leer códigos de estado

El atributo `status_code` contiene el código de estado de la respuesta.

```
good_req = requests.get('https://api.github.com/events')
code_200 = good_req.status_code

notfound_req = requests.get('https://api.github.com/not_found')
code_404 = notfound_req.status_code
```

`requests.codes.__dict__` proporcionará una lista de códigos de estado http disponibles.

Es posible que el usuario `raise_for_status` compruebe si el código de estado era 4xx o 5xx y genera la excepción correspondiente en ese caso.

```
good_req = requests.get('https://api.github.com/events')
good_req.raise_for_status()
# is a 200 status code so nothing happens

notfound_req = requests.get('https://api.github.com/not_found')
notfound_req.raise_for_status()
# raises requests.exceptions.HTTPError: 404 Client Error
```

Lea Comenzando con las solicitudes de python en línea: https://riptutorial.com/es/python-requests/topic/1115/comenzando-con-las-solicitudes-de-python

Capítulo 2: Archivos

Parámetros

Parámetros	Función
expediente	JSON Lista de rutas a los archivos.
tipo de contenido	Tipos mime
encabezados	Encabezados HTTP

Observaciones

La variable `r` en los ejemplos contiene los datos binarios completos de cualquier archivo que esté enviando.

Examples

Carga simple de archivos

```
url = 'http://your_url'
files = {'file': open('myfile.test', 'rb')}
r = requests.post(url, files=files)
```

Carga de archivos con parámetros manuales

```
url = 'http://httpbin.org/post'
files = {'file': ('report.xls', open('report.xls', 'rb'), 'application/vnd.ms-excel',
{'Expires': '0'})}
r = requests.post(url, files=files)
```

Enviando cadenas como archivos

```
url = 'http://httpbin.org/post'
files = {'file': ('report.csv', 'some,data,to,send\nanother,row,to,send\n')}
r = requests.post(url, files=files)
r.text
```

Lea Archivos en línea: https://riptutorial.com/es/python-requests/topic/5929/archivos

Capítulo 3: Django Framework

Examples

Instalación y configuración

Django es un framework de pila completa para desarrollo web. Potencia algunos de los sitios web más populares en Internet.

Para instalar el marco; usar la herramienta `pip` :

```
pip install django
```

Si está instalando esto en OSX o Linux, el comando anterior puede dar como resultado un error de permiso; para evitar este error, instale el paquete para su cuenta de usuario o use un entorno virtual:

```
pip install --user django
```

Una vez que esté instalado, tendrá acceso a la herramienta de arranque `django-admin` , que creará un directorio con algunos valores predeterminados para iniciar el desarrollo.

```
django-admin startproject myproject
```

Esto creará un *`myproject`* directorio con el diseño de proyecto predeterminado.

Conceptos básicos de Django

Django es un framework de desarrollo web rico en características y de pila completa. Combina una gran cantidad de funcionalidades para proporcionar una experiencia común, rápida y productiva para los desarrolladores web.

Los proyectos Django consisten en configuraciones comunes y una o más *aplicaciones* . Cada aplicación es un conjunto de funcionalidades junto con dependencias (como plantillas y modelos) que se agrupan como módulos de Python.

La secuencia de comandos de arranque de django crea automáticamente un archivo de configuración para su proyecto, con las funciones más comunes habilitadas.

Este concepto de aplicaciones permite una fácil funcionalidad de plug-and-play, y hay una gran biblioteca de aplicaciones disponibles para manejar las tareas más comunes. Este concepto de aplicaciones es fundamental para django; gran parte de la funcionalidad incorporada (como la autenticación de usuario y el sitio de administración) son simplemente aplicaciones django.

Para crear su primera aplicación, desde dentro del directorio del proyecto:

```
python manage.py startapp yourapp
```

`yourapp` es el nombre de su aplicación personalizada.

Cada aplicación te permite desarrollar:

1. Una serie de *vistas* : son piezas de código que se ejecutan en respuesta a una solicitud.
2. Uno o más *modelos* ; Que son una abstracción a las bases de datos. Estos le permiten definir sus objetos como objetos de Python, y el ORM incorporado proporciona una API amigable para almacenar, recuperar y filtrar objetos de bases de datos.
3. Están estrechamente relacionadas con los modelos las *migraciones* que son scripts que se generan para proporcionar un método consistente y confiable para aplicar cambios en sus modelos a la base de datos.
4. Un conjunto de *urls a las* que la aplicación responderá.
5. Una o más clases de administración; para personalizar cómo se comporta la aplicación en la aplicación de administración de django incorporada.
6. Cualquier prueba que puedas escribir.

Conceptos básicos - Vistas

Una `view` es cualquier fragmento de código que responde a una solicitud y devuelve una respuesta. Las vistas normalmente devuelven plantillas junto con un diccionario (denominado *contexto*) que generalmente contiene datos para los marcadores de posición en la plantilla. En los proyectos de django, las vistas se encuentran en el módulo `views.py` de aplicaciones.

La vista más simple, devuelve una respuesta directa:

```
from django.http import HttpResponse

def simple_view(request):
   return HttpResponse('<strong>Hello World</strong>')
```

Sin embargo, la mayoría de las vistas utilizan una plantilla:

```
from django.shortcuts import render

def simple_template_view(request):
    return render(request, 'some_template.html')
```

Una plantilla es simplemente cualquier archivo, y opcionalmente puede contener un marcado especial para la funcionalidad agregada; lo que esto significa es que las vistas de django pueden devolver cualquier tipo de respuesta, no solo HTML.

Conceptos básicos - Plantillas

En django, una plantilla es simplemente un archivo que contiene etiquetas especiales que pueden ser reemplazadas por datos de la vista.

El ejemplo de la plantilla canónica sería:

```
<strong>Hello {{ name }}, I am a template!</strong>
```

Aquí, la cadena `{{ name }}` identifica un marcador de posición que puede ser reemplazado por un contexto.

Para representar esta plantilla desde una vista, podemos pasar el valor como un diccionario:

```
from django.shortcuts import render

def simple_view(request):
    return render(request, 'template.html', {'name': 'Jim'})
```

Una vez que se muestre esta vista, el HTML resultante será **Hello Jim, ¡soy una plantilla!** .

Conceptos básicos - URLs

En django, hay un mapeador de URL que asigna direcciones URL a funciones específicas (vistas) que devuelven respuestas. Esta separación estricta entre el diseño del sistema de archivos y el diseño de la URL permite una gran flexibilidad al escribir aplicaciones.

Todos los patrones de url se almacenan en uno o más archivos `urls.py` , y hay un archivo maestro `urls.py` que primero lee django.

Django analiza los patrones en el orden en que se escriben y se detiene cuando encuentra una coincidencia con la URL solicitada por el usuario. Si no se encuentran coincidencias, se genera un error.

En el modo de depuración (activado al establecer `DEBUG = True` en `settings.py`), django imprimirá un mensaje de error detallado cuando una URL solicitada no coincida con ningún patrón. En producción, sin embargo, django mostrará un mensaje 404 normal.

Un patrón de url consiste en una expresión regular de Python, seguida de un *llamable* (un método o función) que se llamará cuando ese patrón coincida. Esta función debe devolver una respuesta HTTP:

```
url(r'/hello$', simple_view)
```

Lea Django Framework en línea: https://riptutorial.com/es/python-requests/topic/6579/django-framework

Capítulo 4: Enviando y recibiendo JSON

Examples

PUBLICACIÓN DE JSON

Para POSTAR un cuerpo JSON, pase una estructura de datos de Python al argumento `json` ; Aquí se publica un diccionario, pero cualquier cosa que se pueda codificar en JSON hará:

```
import requests

# Create a dictionary to be sent.
json_data = {'foo': ['bar', 'baz'], 'spam': True, 'eggs': 5.5}

# Send the data.
response = requests.post(url='http://example.com/api/foobar', json=json_data)
print("Server responded with %s" % response.status_code)
```

`requests` se encargan de codificar a JSON por usted y establecen el tipo de `Content-Type` en `application/json` .

Recibiendo a JSON en una respuesta

Cuando una respuesta contiene JSON válido, solo use el método `.json()` en el objeto `Response` para obtener el resultado descodificado:

```
response = requests.get('http://example.com/')
decoded_result = response.json()
```

Sin embargo, esto no falla con gracia; generará un `JSONDecodeError` si el objeto de respuesta no es analizable por JSON.

Es posible que desee comprobar primero el tipo de contenido MIME, para un manejo de errores más elegante:

```
if 'application/json' in response.headers['Content-Type']:
    decoded_result = response.json()
else:
    non_json_result = response.data
```

ETL de web API's con módulos json y peticiones.

En primer lugar, importar módulos y establecer cadenas de conexión. Si necesita parámetros, puede ponerlos directamente en la cadena de URL (una API en este caso) o compilarlos como dict y pasarlos al argumento params.

```
import requests
import json
```

```
params = {'id': 'blahblah', 'output': 'json'} # You could use
https://www.somesite.com/api/query?id=blahblah&output=json directly.
API = 'https://www.somesite.com/api/query'
APIcred = 'username','password'
```

Las solicitudes manejan HTTPBasicAuth y HTTPDigestAuth automáticamente. Esta API de ejemplo devolverá una cadena JSON. Realiza la solicitud GET y captura la salida. Levante un error para el estado HTTP malo.

```
r = requests.get(API, params = params, auth = APIcred)
r.raise_for_status()
#print(r.status) # Optionally print HTTP status code
```

Convierte la cadena de JSON al objeto de Python con el que puedes trabajar. JSON se ve visualmente similar a como un dict de Python, pero hay diferencias significativas en nulos, verdadero / falso, etc.

```
r_dict = json.loads(r.text)
print(r_dict)
```

Imagine que la salida que acaba de imprimir proviene de una base de datos de varias líneas y varias columnas y es difícil de leer:

> {'row': [{'Country': 'United States', 'pid': 'cc12608f-4591-46d7-b8fe-6222e4cde074', 'Status': '', 'FormerLastName': '', 'Degree': 'Administración de empresas'}, {'País': 'Gran Bretaña', 'pid': 'c9f2c6f7-f736-49d3-8adf-fd8d533bbd58', 'Estado': '', 'FormerLastName': '', 'Degree': ' Administración General'}]}

Puedes imprimir () una versión más legible por humanos con json.dumps (). La siguiente línea codifica el objeto python en una cadena de caracteres JSON con pestañas y lo imprime.

```
print(json.dumps(r_dict, indent = 4))
```

Salida:

```
{
    "row": [
        {
            "Country": "United States",
            "pid": "cc12608f-4591-46d7-b8fe-6222e4cde074",
            "Status": "",
            "FormerLastName": "",
            "Degree": "Business Administration"
        },
        {
            "Country": "Britain",
            "pid": "c9f2c6f7-f736-49d3-8adf-fd8d533bbd58",
            "Status": "",
            "FormerLastName": "",
            "Degree": "General Management"
        }
```

```
    ]
}
```

Puedes acceder a elementos anidados en un dict como este:

```
print(some_dict['BuildingA']['Room12'])
```

Pero nuestra muestra tiene un número arbitrario de objetos en una matriz, ¡que a su vez está anidado como el valor de una clave! Se puede acceder a estos con un número de fila, comenzando con 0.

Cambiemos uno de nuestros valores de 'País' de 'Gran Bretaña' a 'Albania':

```
r_dict['row'][1]['Country'] = 'Albania'
```

Ahora enviemos estos datos a otra API. Las solicitudes pueden aceptar un dict directamente al argumento json, en lugar de codificar una cadena con json.dumps ().

```
r = requests.post('https://www.somesite.com/" + 'api/carrots', json = r_dict, auth = APIcred)
r.raise_for_status()
```

Lea Enviando y recibiendo JSON en línea: https://riptutorial.com/es/python-requests/topic/3099/enviando-y-recibiendo-json

Capítulo 5: Inicio de sesión automatizado utilizando solicitudes sobre inicio de sesión único

Examples

Ejemplo de acceso a páginas autenticadas mediante peticiones.

A veces tenemos el requisito de analizar las páginas, pero hacerlo requiere que usted sea un usuario autorizado. Aquí hay un ejemplo que muestra cómo hacerlo en el inicio de sesión de Oracle.

```
import sys
import requests
import json
from bs4 import BeautifulSoup


def mprint(x):
    sys.stdout.write(x)
    print
    return


headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101
Firefox/7.0.1'}

mprint('[-] Initialization...')
s = requests.session()
s.headers.update(headers)
print 'done'


mprint('[-] Gathering JSESSIONID..')

# This  should redirect us to the login page
# On looking at the page source we can find that
# in the submit form 6 values are submitted (at least at the time of this script)
# try to take those values out using beautiful soup
# and then do a post request. On doing post https://login.oracle.com/mysso/signon.jsp
# we will be given message we have the data which is more than necessary
# then it will take us to the form where we have to submit data here
# https://login.oracle.com/oam/server/sso/auth_cred_submit
# once done we are signed in and doing and requests.get(url) will get you the page you want.

r = s.get("company's local url- a link which requires authentication")
if r.status_code != requests.codes.ok:
  print 'error'
  exit(1)
print 'done'

c = r.content
```

```
soup = BeautifulSoup(c,'lxml')
svars = {}

for var in soup.findAll('input',type="hidden"):
    svars[var['name']] = var['value']


s = requests.session()
r = s.post('https://login.oracle.com/mysso/signon.jsp', data=svars)

mprint('[-] Trying to submit credentials...')
inputRaw = open('credentials.json','r')
login = json.load(inputRaw)


data =  {
        'v': svars['v'],
        'OAM_REQ': svars['OAM_REQ'],
        'site2pstoretoken': svars['site2pstoretoken'],
        'locale': svars['locale'],
        'ssousername': login['ssousername'],
        'password': login['password'],
}

r = s.post('https://login.oracle.com/oam/server/sso/auth_cred_submit', data=data)

r = s.get("company's local url- a link which requires authentication")
# dumping the html page to html file
with open('test.html','w') as f:
    f.write(r.content)
```

credentials.json como se menciona en el código es el siguiente:

```
{
  "ssousername":"example@oracle.com",
  "password":"put your password here"
}
```

Enlace a github - gist

Lea Inicio de sesión automatizado utilizando solicitudes sobre inicio de sesión único en línea: https://riptutorial.com/es/python-requests/topic/6240/inicio-de-sesion-automatizado-utilizando-solicitudes-sobre-inicio-de-sesion-unico

Capítulo 6: Usando solicitudes detrás de un proxy

Examples

Configuración de proxy en código Python

Si su código se está ejecutando detrás de un proxy y conoce el punto final, puede establecer esta información en su código.

`requests` acepta un parámetro `proxies` . Este debe ser un diccionario que asigna el protocolo a la URL del proxy.

```
proxies = {
  'http': 'http://proxy.example.com:8080',
  'https': 'http://secureproxy.example.com:8090',
}
```

Observe que en el diccionario hemos definido la URL del proxy para dos protocolos separados: HTTP y HTTPS. Cada uno se asigna a una URL individual y un puerto. Sin embargo, esto no significa que los dos no puedan ser iguales. Esto también es aceptable:

```
proxies = {
  'http': 'http://secureproxy.example.com:8090',
  'https': 'http://secureproxy.example.com:8090',
}
```

Una vez que se define su diccionario, lo pasa como un parámetro.

```
requests.get('http://example.org', proxies=proxies)
```

Usando variables de entorno proxy

`requests` utiliza variables de entorno específicas automáticamente para la detección de proxy.

- `HTTP_PROXY` definirá la URL del proxy a usar para las conexiones HTTP
- `HTTPS_PROXY` definirá la URL del proxy para usar en las conexiones HTTPS

Una vez que se configuran estas variables de entorno, el código Python no necesita pasar nada al parámetro `proxies` .

```
requests.get('http://example.com')
```

Lea Usando solicitudes detrás de un proxy en línea: https://riptutorial.com/es/python-requests/topic/5933/usando-solicitudes-detras-de-un-proxy

Creditos

S. No	Capítulos	Contributors
1	Comenzando con las solicitudes de python	Andy, Batsu, Burhan Khalid, Community, dansyuqri, icesin, k-nut, Martijn Pieters, Meysam, mickeyandkaka, mphuie, OJFord, supersam654
2	Archivos	ReverseCold
3	Django Framework	Burhan Khalid
4	Enviando y recibiendo JSON	bendodge, Martijn Pieters, OJFord, Olli, Stan
5	Inicio de sesión automatizado utilizando solicitudes sobre inicio de sesión único	lordzuko
6	Usando solicitudes detrás de un proxy	Andy