Python Requests and selenium

Christoffer Berglund

October 19, 2018

Overview

- Repetition
- Requests
- Beautiful soup
- Selenium

Repetition

Request types

get	requests data
post	send data to server
put	send data to server to create or update
head	same as get, but without body
delete	delete resource
...	...

HTTP status codes

200	ok
201	created
400	bad request
401	unauthorized
500	internal server error

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

What is flask?

- Microframework
- Simple

Simple example

```python
# file: simple.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

- To run it

```
export FLASK_ENV=development
export FLASK_APP=simple.py
flask run
```

Dynamic html, variable binding

- python file

```python
from flask import Flask, render_template

app = Flask(__name__)
@app.route('/index')
def index():
    user = {'name': 'Christoffer'}
    return render_template('index.html', title='MyTitle', user=name)
```

- html file

```html
<html>
  <head> <title>{{ title }}</title> </head>
  <body> <h1>Hello class, my name is {{ user.name }}!</h1> </body>
</html>
```

Control statements in html

```
<html>
  <head>
    {% if title %}
    <title> {{ title }} </title>
    {% else %}
    <title> Homepage </title>
    {% endif %}
  </head>
</html>
```

Get and Post data

```python
import time

from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def my_form():
    return render_template('my-form.html')

@app.route('/', methods=['POST'])
def my_form_post():
    user = request.form['username']
    password = request.form['password']
    print(password)
    time.sleep(1)
    print("Storing {} in hacker list".format(password))
    time.sleep(1)
    print("Selling password...:", password)
    return "login " + user + " we will keep your password safe"
```

Requests

Philosophy

- Beautiful is better than ugly

- Explicit is better then implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Installation

- Install with pip

```
pip install requests
```

Documentation and github

Website: http://docs.python-requests.org/en/master/ Github: https://github.com/requests/requests

Simple example

- Import requets
- Set up a get requets
- r is now a response object

```
import requests

r = requests.get('https://uia.no')
```

Request types

- Can use httpbin.org for testing
- Post data

```
import requests

r = requests.post('https://httpbin.org/post', data={'key': 'value'})
```

5min task

- Create a put request
- Create a delete request
- Create a head requests

5min task solution

```python
import requests

r = requests.put('https://httpbin.org/delete', data={'key': 'value'})
r = requests.delete('https://httpbin.org/delete')
r = requests.head('https://httpbin.org/delete')
```

Request with URL parameters

```python
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get('https://httpbin.org/get', params=payload)
```

- url will look like this:

```python
print(r.url)
# https://httpbin.org/get?key2=value2&key1=value1
```

JSON response content

- If the website support json response

```python
import requests

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

Headers

- To get response headers

```
import requests

r = requests.get('https://uia.no')
r.headers
```

- Send headers

```
import requests

headers = {'user-agent': 'my-app/0.0.1'}
r = requests.get('https://uia.no', headers=headers)
```

Fake headers

- As long as your options are correct it's allowed
- http://www.useragentstring.com/pages/useragentstring.php

```
  import requests

  url = 'https://uia.no'
  headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36'}

  response = requests.get(url, headers=headers)
  response.content
```

5min task

- Send a get request to uia.no using a fake header
- Hint: can be picked from this list:

http://www.useragentstring.com/pages/useragentstring.php

Response object

- Many properties which will tell you information

```
import requets

r = requests.get('https://uia.no')
```

```
r.status_code  # response status code
r.headers  # response headers
r.text  # response type string
r.content  # response type byte
r.cookies  # response cookies
r.url  # respnse url
```

Cookies

- Used by most webpages
- Can hold status information
- Can record user browsing activity
- Not always implemented correctly on server

Sessions

- Persistent cookies
- Will reuse the underlying TCP connection
- Session object has all the methods from regular requests

Session in use

```
import requests

s = requests.Session()

s.get('https://httpbin.org/cookies/set/sessioncookie/123456789')
r = s.get('https://httpbin.org/cookies')

print(r.text)
# '{"cookies": {"sessioncookie": "123456789"}}'
```

5min task

- Create a session and post something to this website:

https://httpbin.org/post

5min task solution

```
import requests

s = requests.Session()

s.post('https://httpbin.org/post', data={'key': 'value'})
```

Sessions with context manager

```
import requests

with requests.Session() as s:
    s.get('https://uia.no')
```

Beautiful soup

For parsing

- Since requests does not parse the data we can use beautiful soup
- Not a HTTP client, so we need to pass in the response from requests

or save the data to a file and parse that.

- Works on html tags

Installation

- Can be install system wide with apt-get

```
sudo apt-get install python3-bs4
```

- or with pip

```
pip install beautifulsoup4
```

Documentation

Website: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Simple example

- Most common to use with requests

```python
import requests
from bs4 import BeautifulSoup

r = requests.get('https://uia.no')
soup = BeautifulSoup(r.text)
print(soup.title)
```

5min task

- Find all links from uia.no
- Hint1: you can use `soup.findAll()` to find multiple

5min task solution

```python
import requests
from bs4 import BeautifulSoup

r = requests.get('https://uia.no')
soup = BeautifulSoup(r.text)
soup.findAll('a')
```

find()

- To find one
- Possible uses:
  - To find body tag or other tags that appears once
- Will match on first hit?
- Returns `None` if no match

```python
soup.find('body')
```

- This is the same

```python
soup.findAll('body', limit=1)
```

find$_{\text{all}}$()

- To find all
- Returns [], empty list if no match

```
soup.findAll('a')
```

find$_{\text{parents}}$()

- find and findAll are searching top down
- find$_{\text{parents}}$ are searching down and up

```
import requests
from bs4 import BeautifulSoup

r = requests.get('https://uia.no')
soup = BeautifulSoup(r.text)
tag = soup.find(string='For studenter')
tag.find_parent('a')
```

Other useful functions

- find$_{\text{siblings}}$()
- find$_{\text{children}}$()
- find$_{\text{next}}$()
- Read documentation for more info

Selenium

Installation

```
pip install selenium
```

- Download the drivers
- Extract and place them at: *usr/bin* or /usr/local/bin

Documentation and github

- Documentation:

https://selenium-python.readthedocs.io/

- Github:

https://github.com/SeleniumHQ/selenium

Simple example

- demo

Navigation

- demo

5min task

- Log into google or some other service with selenium
- If you don't want to use your login, just send in something random

Cookie

```
# Go to the correct domain
driver.get("http://www.example.com")

# Now set the cookie. This one's valid for the entire domain
cookie = {‘name’ : ‘foo’, ‘value’ : ‘bar’}
driver.add_cookie(cookie)

# And now output all the available cookies for the current URL
driver.get_cookies()
```

locating element

$\text{find}_{\text{elementbyid}}$	html id
$\text{find}_{\text{elementbyname}}$	html name tag
$\text{find}_{\text{elementbyxpath}}$	xml absolute or relative path
$\text{find}_{\text{elementbulinktext}}$	html link text
$\text{find}_{\text{elementbypartiallinktext}}$	html link text partial
$\text{find}_{\text{elementbytagname}}$	html tag name
$\text{find}_{\text{elementbyclassname}}$	html class name
$\text{find}_{\text{elementbycssselector}}$	p.content

$\text{css}_{\text{selector}}$ is just another syntax way of typing it

Locating elements

- Same as previous slide but typed:

$\text{find}_{\text{elementsbyid}} \cdots \cdots$

Waits

- Sometimes you need to wait for elements to load
- 2 main methods, implicit and explicit
- Can also use time.sleep() from os

Implicit

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicitly_wait(10)  # seconds
driver.get("http://somedomain/url_that_delays_loading")
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

Explicit

```
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)
element = wait.until(EC.element_to_be_clickable((By.ID, 'someid')))
```