Web Scrapping

BY… SAKEEB SHEIKH

CONTENT

1. WORKING WITH **URLLIB** PACKAGE
2. FILTERING DATA USING REGULAR EXPRESSIONS (**RE**)
3. WEB SCRAPPING USING **BEAUTIFULSOUP**

Introduction

- **urllib.request** is a Python module for fetching URLs
- It offers a very simple interface, in the form of the ***urlopen*** function.
- Capable of fetching URLs using a variety of different protocols
- Offers more complex interface for handling common situations - like basic authentication, cookies, proxies and so on
- Installing urllib → ***pip install urllib***

Fetching URLs

```
import urllib.request
with urllib.request.urlopen(' https://www.w3schools.com/') as response:
        html = response.read()
```

Note : Instead of an ‘http:’ URL we could have used a URL starting with ‘ftp:’, ‘file:’, etc.

Steps

- Creating url request using **urllib.request.Request()**
- Calling **urllib.request.urlopen()** with this Request object returns a response object for the URL requested.
- This response is a file-like object, which means you can for example call **.read()** on the response

Sending Data along with URL

- Methods to send data
  - POST
  - GET
- URL Encoding
  - urllib.parse
  - Used to encode the data before sending

Sending Data along with URL

- Using POST

```
import urllib.parse
import urllib.request
url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }
data = urllib.parse.urlencode(values)
data = data.encode('ascii')  #’utf-8’
# data should be bytes
req = urllib.request.Request(url, data)
with urllib.request.urlopen(req) as response:
          the_page = response.read()
```

Sending Data along with URL

- Using GET

```
import urllib.request
import urllib.parse

data = {}
data['name'] = 'Somebody Here'
data['location'] = 'Northampton'
data['language'] = 'Python‘

url_values = urllib.parse.urlencode(data)
print(url_values) # The order may differ from below.

url = 'http://www.example.com/example.cgi'
full_url = url + '?' + url_values

data = urllib.request.urlopen(full_url)
```

Handling Exceptions

- *urlopen* raises URLError when it cannot handle a response

- **URLError**

```
req = urllib.request.Request('http://www.pretend_server.org')
try: urllib.request.urlopen(req)
    except urllib.error.URLError as e:
        print(e.reason)
```

- **HTTPError**

```
from urllib.request import Request, urlopen
from urllib.error import URLError
req = Request(someurl)
try:
        response = urlopen(req)
except URLError as e:
        if hasattr(e, 'reason'):
                print('We failed to reach a server.')
                print('Reason: ', e.reason)
        elif hasattr(e, 'code'):
                print('The server couldn\'t fulfill the request.')
                print('Error code: ', e.code)
        else:
                # everything is fine
```

info and geturl

- The response returned by urlopen as two useful methods info() and geturl()
- **geturl** - this returns the real URL of the page fetched.
- **info** - this returns a dictionary-like object that describes the page fetched. (header information)

Using Regular Expression Module

- **Identifiers**
  - **\d** – Any number
  - **\D** – Anything but a number
  - **\s** – Space
  - **\S** – Anything but a space
  - **\w** – Any Character
  - **\W** – Anything but a Character
  - **.** – Any Character except for a new line
  - **\b** – The whitespace arround data(word)
  - **\.** – A Period

- **Modifiers**
  - {1,3} – were expecting number sized 1-3
  - + – Match one or more
  - * – Match 0 or more
  - ? – 0 or 1
  - $ – Match end of a string
  - ^ – Match beginning of a string
  - | – either or
  - [ ] – range or variance
  - {x} – expecting x amount

- White Space Characters
  - \n - New Line
  - \s - space
  - \t - tab
  - \e - escape
  - \f - form feed
  - \r - return

Don’t Forget :

. + * ? [ ] $ ^ ( )
**{ } **

**To use above symbols in RE we have to escape them by **

Syntax to Search a pattern within a String

- re.findall(r'<requered-pattern>', inputstring)

Example -1

```
import re
exampstr = 'John is 15 years old, and Deniel is 27 years old'
ages = re.findall(r'\d{1,3}',exampstr)
names = re.findall(r'[A-Z][a-z]*', exampstr)

print(ages)
print(names)

ageDict={}
i=0
for eachname in names:
    ageDict[eachname] = ages[i]
    i=i+1
print(ageDict)
```

Example-2 (re with urllib)

```
import urllib.request
import urllib.parse
import re

weburl = 'https://www.tutorialspoint.com/html/index.htm'
resp = urllib.request.urlopen(weburl)
respData = resp.read()

#Now Filtering Using Regular Expression
paragraph = re.findall(r'<li>(.*?)</li>',str(respData))
Print(paragraph)
```

Web Scraping with Beautiful Soup

- It is Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

Important Functions from Beautiful Soup

- soup.title *# <title>The Dormouse's story</title>*

- soup.title.name *# u'title'*
- soup.title.string *# u'The Dormouse's story'*
- soup.title.parent.name *# u'head'*
- soup.p *# The Dormouse's story *
- soup.p['class'] *# u'title'*
- soup.a
  - *# Elsie*
- soup.find_all('a')

  - *# [Elsie,*
  - *# Lacie,*
  - *# Tillie*

Example

```
from bs4 import BeautifulSoup
import urllib
import re
r = urllib.request.urlopen('https://analytics.usa.gov').read()
soup = BeautifulSoup(r, "lxml")
type(soup)
```

Continue…

#Scraping a webpage and saving your result

```
print(soup.prettify()[:100])

for link in soup.find_all('a'): print(link.get('href'))

for link in soup.findAll('a', attrs={'href': re.compile("^http")}): print(link)
```

Continue…

```
file = open('parsed_data.txt', 'w')
for link in soup.findAll('a', attrs={'href': re.compile("^http")}):
    soup_link = str(link)
    print(soup_link)
    file.write(soup_link)
file.flush()
file.close()
```