Лабораторно упражнение №5

Възможности на Python за разработване на мрежови приложения. Обработване на данни в HTTP протокол, обработване на уеб страници и файлове с urllib, обработване на HTML документи

Основни понятия:

- *BeautifulSoup* – това е библиотека на Python за анализ и извличане на данни от HTML документи. BeautifulSoup код може да се изтегли от www.crummy.com.
- *port* - число, което обикновено показва с коя услуга (service) се свързвате, когато правите сокет връзка със сървър. Например, услугата уеб сървър обикновено използва порт 80, а услугата електронна поща използва порт 25.
- *scrape* - когато програма се преструва на уеб браузър и извлича уеб страница, след което разглежда съдържанието на уеб страницата.
- *socket* - мрежова връзка между две приложения, при която приложенията могат да изпращат и получават данни във всяка посока.
- spider – действа като уеб търсачка за извличане на страници.

При разработване на мрежови приложения е необходимо да се добави библиотеката socket

Import socket

1. HyperText Transfer Protocol – HTTP

Мрежовият протокол, който захранва мрежата, всъщност е доста прост и има вградена поддръжка в Python. Сокетите спомагат за лесно създаване на мрежови връзки и извличане на данни.

Socket е файл, който осигурява двупосочна връзка между две програми. В един и същ сокет могат да се записват и прочитат данни.

Ако се запишат данни в сокет, те се изпращат до приложението в другия край на сокета. Ако се чете от сокета, се получават данните, които приложението в другият край на сокета е изпратило.

Протоколът е набор от точни правила, които определят кой трябва да се изпрати/получи пръв.

2. Най-простият уеб браузър

Може би най-лесният начин да се покаже как работи HTTP протоколът е да се напише много проста програма на Python, която осъществява връзка с уеб сървър и следва правилата на HTTP протокола, за да поиска документ от сървъра и да го върне обратно.

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt
HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)
while True:
    data = mysock.recv(512)
    print(data.decode(),end='')
    if len(data) < 1:
        break
mysock.close()
```

Първо програмата прави връзка с порт 80 на сървъра www.py4e.com. Тъй като нашата програма играе ролята на „уеб браузър", HTTP протоколът казва, че трябва да изпратим командата GET, последвана от празен ред. \ r \ n означава EOL (края на реда), така че \ r \ n \ r \ n не означава нищо между две EOL последователности.

Това е еквивалент на празен ред.

След като се изпрати празен ред, следва цикъл, който приема данни от 512 символа от сокета и ги отпечатва, докато няма повече данни за четене (т.е. recv () връща празен низ).

Примерен изход

```
The program produces the following output:
HTTP/1.1 200 OK
Date: Wed, 11 Apr 2018 18:52:55 GMT
Server: Apache/2.4.7 (Ubuntu)
```

Изходът започва със заглавни редове, в които уеб сървърът изпраща описание на документа.

Например, Content-Type показва, че документът е обикновен текстов документ (text/plain).

След като сървърът ни изпрати описанията, той добавя празен ред, за да посочи края на описанията и след това изпраща действителните данни на файла romeo.txt.

Този пример показва как се осъществява мрежова връзка със сокети на ниско ниво.

Сокетите могат да се използват за комуникация с уеб сървър, с пощенски сървър или много други видове сървъри. Всичко, което е необходимо, е да се намери документ, който описва протокола, и да се напише кода, за да се изпратят и получат данните в съответствие с протокола.

Тъй като протоколът, който се използва най-често е уеб протокол HTTP, Python има специална библиотека, която поддържа HTTP протокола за извличане на документи и данни през мрежата.

Едно от изискванията за използване на HTTP протокола е необходимостта да се изпращат и получават данни като байтови обекти вместо низове. В предходния пример методите encode() и decode() преобразуват низове в байтови обекти и обратно.

Следващият пример използва b '' нотация, за да посочи, че променлива трябва да се съхранява като байтов обект. Действието на функцията encode() е еквивалентно на нотацията b ''.

```
>>> b'Hello world'
b'Hello world'
>>> 'Hello world'.encode()
     b'Hello world'
```

3 Извличане на изображение през HTTP

В горния пример се извлече обикновен текстов файл, който имаше нови редове и данните се изведоха на екрана. Може да се използва подобна програма за извличане на изображение в целия HTTP. Вместо да се извеждат данните на екрана, данните може да се натрупват в низ и след това да се запазват във файл, както следва:

```
import socket
import time
HOST = 'data.pr4e.org'
PORT = 80
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))
mysock.sendall(b'GET http://data.pr4e.org/cover3.jpg
HTTP/1.0\r\n\r\n')
count = 0
picture = b""
while True:
    data = mysock.recv(5120)
    if len(data) < 1: break
    #time.sleep(0.25)
    count = count + len(data)
    print(len(data), count)
    picture = picture + data
mysock.close()
# Look for the end of the header (2 CRLF)
pos = picture.find(b"\r\n\r\n")
print('Header length', pos)
print(picture[:pos].decode())
# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg", "wb")
```

```
fhand.write(picture)
fhand.close()
```

Резултат: *?*

Може да се види, че за този URL адрес Content-Type показва, че тялото на документа е изображение (image / jpeg).

Докато се изпълнява програмата, може да се види, че не получаваме 5120 знака всеки път, когато се обърнем към метода recv(). Получаваме толкова символи, колкото са ни прехвърлени през мрежата от уеб сървъра в момента, в който извикаме recv().

Може да се забави последователността чрез използване на time.sleep(). Със закъснението програмата се изпълнява по следния начин:

Резултат: *?*

По този начин чрез recv(), се получават 5120 знака всеки път, когато се поискат нови данни.

4. Извличане на уеб страници с urllib

Въпреки, че може ръчно да се изпращат и получават данни през HTTP чрез библиотеката на сокетите, има много по-опростен начин да се изпълни тази обща задача в Python, използвайки библиотеката urllib.

Използвайки urllib, може уеб страница да се третира подобно на файл. Чрез посочване на уеб страница за изтегляне urllib обработва всички HTTP протоколи и подробности за описанието.

Еквивалентният код за четене на romeo.txt файла от интернет с помощта на urllib е:

```
import urllib.request
fhand =
urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
     print(line.decode().strip())
```

След като уеб страницата е отворена с urllib.urlopen, може да се третира като файл и да се чете, използвайки for loop.

Когато програмата работи, се вижда само изхода на съдържанието на файла.

Самостоятелна задача 1:

Да се напише програма за извличане на данните от файл и да се изчисли честотата на всяка дума във файла.

5. Четене на двоични файлове с помощта на urllib.

Данните в бинарните файлове по принцип не са подходящи за разпечатване, но лесно може да се направи копие на URL в локален файл на твърд диск с помощта на urllib.

Необходимо е да се отвори URL адреса и да се прочете, за да се изтегли цялото съдържание на документа в низ (img), след което да се запише тази информация в локален файл.

Примерна реализация

```
import urllib.request, urllib.parse, urllib.error
img =
urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').
read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()
```

Тази програма чете всички данни наведнъж в мрежата и ги съхранява в променливата img в оперативната памет, след това отваря файла cover.jpg и записва данните на диска. Аргументът wb на функция open() отваря двоичен файл само за писане. Тази програма ще работи, ако размерът на файла е по-малък от размера на паметта на компютъра.

Ако обаче това е голям аудио или видео файл, тази програма може да се срине или поне да работи изключително бавно, когато на компютъра липсва памет. За да се избегне изчерпване на паметта, се извличат данните в блокове (или буфери) и след това се записват на диска, преди да се извлече следващия блок. По този начин програмата може да чете файл с всякакъв размер, без да използва цялата памет, която има компютърът.

```
import urllib.request, urllib.parse, urllib.error
img =
urllib.request.urlopen('http://data.pr4e.org/cover3.jpg')
fhand = open('cover3.jpg', 'wb')
size = 0
while True:
     info = img.read(100000)
    if len(info) < 1: break
    size = size + len(info)
    fhand.write(info)
print(size, 'characters copied.')
fhand.close()
```

В този пример се четат само 100 000 знака наведнъж и след това се записват във файла cover.jpg, преди да се извлекат следващите 100 000 знака от мрежата.

Резултат: *?*

6. Разбор на HTML с помощта на регулярни изрази

Един прост начин за разбор на HTML е използването на регулярни изрази за многократно търсене и извличане на подстраници, които съответстват на определен модел. Ето една проста уеб страница:

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
```

Следният регулярен израз търси съвпадения и извлича стойностите на връзката от горния текст:

```
href="http[s]?://.+?"
```

Самостоятелна задача 2:

Напишете програма, която извлича адреси при съвпадение с въведен регулярен израз.

7. Разбор на HTML с помощта на BeautifulSoup

Има редица библиотеки на Python, които могат да помогнат при анализиране и извличане на данни от HTML страниците. Всяка от библиотеките има своите силни и слаби страни. Библиотеката BeautifulSoup позволява лесно да се извличат необходимите данни. Можете да изтеглите и инсталирате BeautifulSoup от:
https://pypi.python.org/pypi/beautifulsoup4

Информацията за инсталиране на BeautifulSoup с инструмента Python Package Index е достъпна на:

https://packaging.python.org/tutorials/installing-packages/

Пример:

Използване на urllib за четене на страницата и след това чрез BeautifulSoup се извличат атрибутите на href от маркерите anchor (a).

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl
```

```
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
url = input('Enter - ')
html = urllib.request.urlopen(url,
context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
     print(tag.get('href', None))
```

Програмата подканва за уеб адрес, след това отваря уеб страницата, чете данните и ги предава на парсера BeautifulSoup и след това извлича всички анкерни маркери и отпечатва атрибута href за всеки маркер.

Резултат: *?*

Самостоятелна задача 3:

Напишете програма за изтегляне на различни части от маркер.

Самостоятелна задача 4:

Напишете програма за осъществяване на връзка със сървър, така че да може да се зарежда всяка уеб страница. Можете да използвате split ('/') за разбиване на URL адреса на съставните му части, за да можете да извлечете името на хоста за повикване на сокет. Добавете проверка за грешка, при която потребителят въвежда неправилно форматиран или несъществуващ URL адрес.

Самостоятелна задача 5:

Променете вашата програма от самостоятелна задача 4, така че да се преброи броя на получените символи и да спре да се показва текст, след като са показани 300 символа. Програмата трябва да извлече целия документ и да преброи общия брой символи и да покаже броя на символи в края на документа. Тествайте например с url 'http://data.pr4e.org/romeo.txt'.

Самостоятелна задача 6:

Използвайте urllib, за да реализирате задача 4 и задача 5.

Самостоятелна задача 7:

Напишете програма за извличане, преброяване и извеждане на параграфите в HTML документ.

Самостоятелна задача 8:

Променете сокетната програма от самостоятелна задача 5, така че тя да показва данни само след като са получени хедърите и празният ред. Не забравяйте, че recv() получава символи (включително и нови редове).

Полезни връзки.

1.1. https://docs.python.org/3/
1.2. https://www.programiz.com/python-programming
1.3. https://www.tutorialspoint.com/python/index.htm
1.4. https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ch1.html

Допълнителни задачи:

1. Напишете програма на Python, която проверява дали дадена web страница е налична на сървъра.

2. Напишете програма на Python, която извлича съдържанието на тага title от даден html документ.

3. Напишете програма на Python, която извлича съдържанието на тага h1 от example.com.

4. Напишете програма на Python, която извежда адреса на атрибута **href** на първия таг **a** в даден HTML документ.

5. Напишете програма на Python, която извлича и извежда връзките на всички изображения от en.wikipedia.org/wiki/Peter_Jeffrey_(RAAF_officer).

6. Напишете програма на Python, която проверява дали дадена web страница съдържа заглавие (т.е. title).

7. Напишете програма на Python, която извлича целия текст на дадена web страница.

8. Напишете програма на Python, която извежда всички тагове, които са вложени в тага body на даден html документ (т.е. всички descendants на body).

9. Напишете програма на Python, която извежда всички редове от страницата www.python.org, които съдържат низа “Python”.

10. Напишете програма на Python, която извежда данни за 250 най-гледани филми от IMDB (име на филм, дата на премиерата, имена на режисьор и най-известните актьори).