Introduction to Web Science

Assignment 4

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: November 23, 2016, 10:00 a.m.
Tutorial on: November 25, 2016, 12:00 p.m.

In this assignment we cover two topics: 1) **HTTP** & 2) **Web Content**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: India
Group Member : Jalpa Patel, Amani Gaddamedi, Jasvinder Kaur

1 Implementing a simplified HTTP GET Request (15 Points)

The goal of this exercise is to review the hyptertext transfer protocol and gain a better understanding of how it works.

Your task is to use the python programming language to create an HTTP client (http-client.py) that takes a URL as a command line argument and is able to download an arbitrary file from the World Wide Web and store it on your hard drive (in the same directory as your python code is running). The program should also print out the complete HTTP header of the response and store the header in a seperated file.

Your programm should only use the socket library so that you can open a TCP socket and and sys library to do command line parsing. You can either use urlparse lib or your code from assignment 3 in order to process the url which should be retrieved.

Your programm should be able to sucessfully download at least the following files:

1. http://west.uni-koblenz.de/en/studying/courses/ws1617/introduction-to-web-science
2. http://west.uni-koblenz.de/sites/default/files/styles/personen_bild/public/_IMG0076-Bearbeitet_03.jpg

Use of libraries like `httplib`, `urllib`, etc are not allowed in this task.

1.1 Hints:

There will be quite some challenges in order to finnish the task

- Your program only has to be able to process HTTP-responses with status 200 OK.
- Make sure you receive the full response from your TCP socket. (create a function handling this task)
- Sperated the HTTP header from the body (again create a function to do this)
- If a binary file is requested make sure it is not stored in a corrupted way

1.2 Example

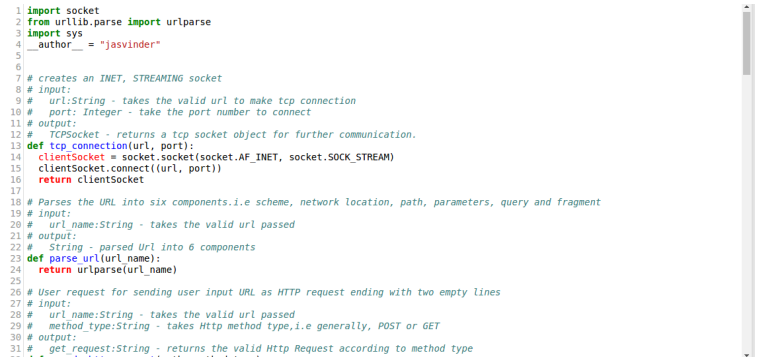
```
1: python httpclient.py http://west.uni-koblenz.de/index.php
2:
3: HTTP/1.1 200 OK
4: Date: Wed, 16 Nov 2016 13:19:19 GMT
5: Server: Apache/2.4.7 (Ubuntu)
6: X-Powered-By: PHP/5.5.9-1ubuntu4.20
7: X-Drupal-Cache: HIT
8: Etag: "1479302344-0"
9: Content-Language: de
```

```
10: X-Frame-Options: SAMEORIGIN
11: X-UA-Compatible: IE=edge,chrome=1
12: X-Generator: Drupal 7 (http://drupal.org)
13: Link: <http://west.uni-koblenz.de/de>; rel="canonical",<http://west.uni-koblenz.de
14: Cache-Control: public, max-age=0
15: Last-Modified: Wed, 16 Nov 2016 13:19:04 GMT
16: Expires: Sun, 19 Nov 1978 05:00:00 GMT
17: Vary: Cookie,Accept-Encoding
18: Connection: close
19: Content-Type: text/html; charset=utf-8
```

The header will be printed and stored in index.php.header. The retrieved html document will be stored in index.php

Answer:

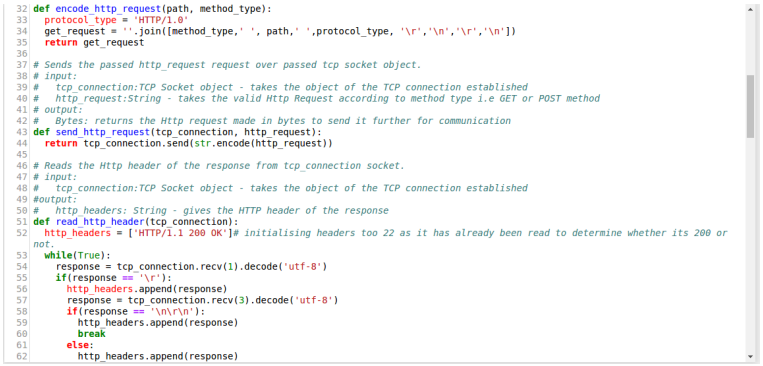
1.



```python
import socket
from urllib.parse import urlparse
import sys
__author__ = "jasvinder"


# creates an INET, STREAMING socket
# input:
#   url:String - takes the valid url to make tcp connection
#   port: Integer - take the port number to connect
# output:
#   TCPSocket - returns a tcp socket object for further communication.
def tcp_connection(url, port):
  clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  clientSocket.connect((url, port))
  return clientSocket

# Parses the URL into six components.i.e scheme, network location, path, parameters, query and fragment
# input:
#   url_name:String - takes the valid url passed
# output:
#   String - parsed Url into 6 components
def parse_url(url_name):
  return urlparse(url_name)

# User request for sending user input URL as HTTP request ending with two empty lines
# input:
#   url_name:String - takes the valid url passed
#   method_type:String - takes Http method type,i.e generally, POST or GET
# output:
#   get_request:String - returns the valid Http Request according to method type
```

Figure 1: Code

2.



```python
def encode_http_request(path, method_type):
  protocol_type = 'HTTP/1.0'
  get_request = ''.join([method_type,' ', path,' ',protocol_type, '\r','\n','\r','\n'])
  return get_request

# Sends the passed http_request request over passed tcp socket object.
# input:
#   tcp_connection:TCP Socket object - takes the object of the TCP connection established
#   http_request:String - takes the valid Http Request according to method type i.e GET or POST method
# output:
#   Bytes: returns the Http request made in bytes to send it further for communication
def send_http_request(tcp_connection, http_request):
  return tcp_connection.send(str.encode(http_request))

# Reads the Http header of the response from tcp_connection socket.
# input:
#   tcp_connection:TCP Socket object - takes the object of the TCP connection established
#output:
#   http_headers: String - gives the HTTP header of the response
def read_http_header(tcp_connection):
  http_headers = ['HTTP/1.1 200 OK']# initialising headers too 22 as it has already been read to determine whether its 200 or
not.
  while(True):
    response = tcp_connection.recv(1).decode('utf-8')
    if(response == '\r'):
      http_headers.append(response)
      response = tcp_connection.recv(3).decode('utf-8')
      if(response == '\n\r\n'):
        http_headers.append(response)
        break
      else:
        http_headers.append(response)
```

Figure 2: Code

3.

```
    else:
      http_headers.append(response)
  return ''.join(http_headers)

# Checks whether the response is 200 or not from the tcp socket.
# input:
#   tcp_connection:TCP Socket object - takes the object of the TCP connection established
#output:
#    Boolean : True for 200 False for all other http status code.
def http_status_200(tcp_connection):
  http_status = tcp_connection.recv(15).decode('utf-8')
  print(http_status)
  if(http_status.startswith('HTTP/1.1 200 OK')):
    return True
  return False

# Reads the body of the Http response as binary
# input:
#   tcp_connection:TCP Socket object - takes the object of the TCP connection established
#output:
#   list - gives the body of the Http response, as binary data
def read_http_body_as_binary(tcp_connection):
  http_body = []
  response = tcp_connection.recv(1024)
  while(response):
    http_body.append(response)
    response = tcp_connection.recv(1024)
  return http_body

# Decodes http header
# input:
#   http header: Http header string.
```

Figure 3: Code

4.

```
#   http_header: Http header string.
#output:
#    list - list of http header strings.
def decode_http_header(http_header):
  return http_header.strip().split('\r\n')

# Reads the decoded http header
# input:
#   decoded_http_header: list of decoded http header
#output:
#    'String: tells whether type is text in binary
def find_content_type(decoded_http_header):
  #print(decoded_http_header)
  for value in decoded_http_header:
    if(value.startswith('Content-Type:')):
      if(value.startswith('Content-Type: text/html')):
        return 'text'
      else:
        return 'binary'

# Reads the string and the file name and saves the string on the file.
# input:
#   string_data_to_be_written:String - String of text to be written on the file
#   file_name: String- file name.
#output:
#    None
def save_text_data(string_data_to_be_written, file_name):
  with open(file_name, 'w+') as file:
    file.write(string_data_to_be_written)

# Reads the string and the file name and saves the string on the file.
# input:
```

Figure 4: Code

5.

```
# Reads the string and the file name and saves the string on the file.
# input:
#   url_name:String - Url of the resource to be downloaded.
#   saveheader: Boolean- default is True. This is a flag for stating whether the header or not.
#output:
#    None
def call_http_server(url_name, saveheader = True):
  parsed_url = parse_url(url_name)
  tcp_socket = tcp_connection(parsed_url.netloc, 80)
  http_request = encode_http_request(parsed_url.path, 'GET')
  send_http_request(tcp_socket, http_request)
  print(url_name)
  if(http_status_200(tcp_socket)):
    http_header = read_http_header(tcp_socket)
    http_data = read_http_body_as_binary(tcp_socket)
    decoded_message = decode_http_header(http_header)
    file_name = parsed_url.path.split('/')[-1]
    if saveheader:
      header_file_name = file_name +  '.header'
      save_text_data(http_header, header_file_name)
    if(find_content_type(decoded_message) == 'text'):
      text_file_name = file_name + '.html'
      save_binary_data(http_data, text_file_name)
    elif(find_content_type(decoded_message) == 'binary'):
      save_binary_data(http_data, file_name)
  else:
      print("http status not 200")
  tcp_socket.close()

if __name__ == "__main__":
  call_http_server(sys.argv[1])
```

Figure 5: Code

6.

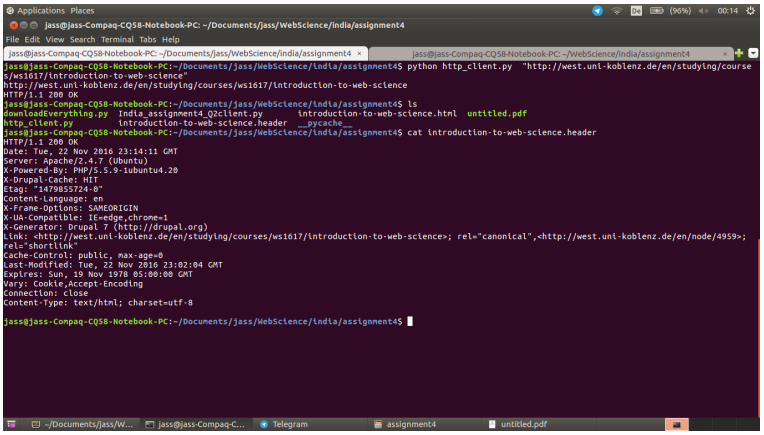


Figure 6: Output

7.

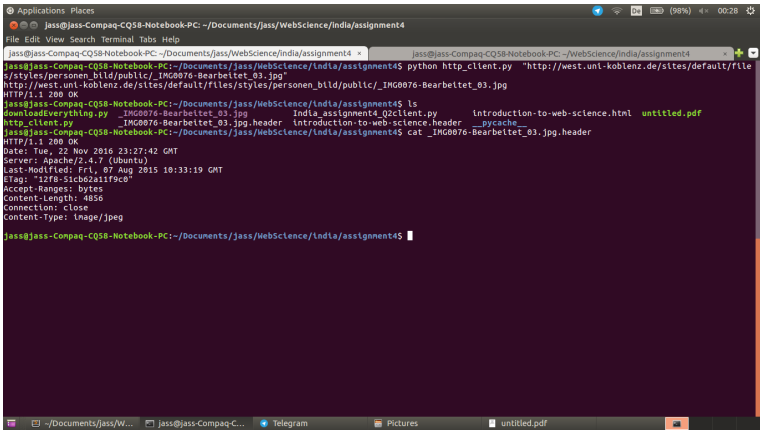


Figure 7: Output

8.

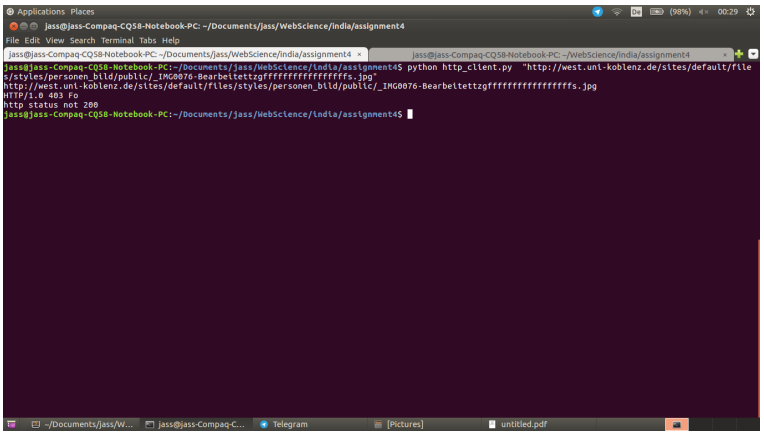


Figure 8: Output

2 Download Everything (15 Points)

If you have successfully managed to solve the previous exercise you are able to download a web page from any url. Unfortionately in order to successfully render that very webpage the browser might need to download all the included images

In this exercise you should create a python file (downloadEverything.py) which takes two arguments. The first argument should be a name of a locally stored html file. The second argument is the url from which this file was downloaded.

Your program should

1. be able to find a list of urls the images that need to be downloaded for successful rendering the html file.
2. print the list of URLs to the console.
3. call the program from task 1 (or if you couldn't complete task 1 you can call wget or use any python lib to fulfill the http request) to download all the necessary images and store them on your hard drive.

To finnish the task you are allowed to use the 're' library for regualar expressions and everything that you have been allowed to use in task 1.

2.1 Hints

1. If you couldn't finnish the last task you can simulate the relevant behavior by using the program wget which is available in almost any UNIX shell.
2. Some files mentioned in the html file might use relative or absolut paths and not fully qualified urls. Those should be fixed to the correct full urls.
3. In case you run problems with constructing urls from relative or absult file paths you can always check with your web browser how the url is dereferenced.

Answer:

Code:

1.

```python
import re
import sys
import http_client

# Reads the file saved at the file name and returns the data.
# input:
#   file_name:String - name of the fiel to be read.
#output:
#    String: Data from the file.
def read_data(file_name):
  with open(file_name, 'r') as f:
      data = f.read()
  return data


# finds the src from the img tag in full html string..
# input:
#   html:String - HTML data as strings
#output:
#    List: list of urls from image tags src.
def match_img_src(html):
  img_pattern = '<img.*?src="([^"]*)"'
  return re.findall(img_pattern, html)


# Converts the relative urls from url list to complete http urls.
# input:
#   url_list:List - URL list as list of string.
#   url: String- Url of main page from where html was downloaded.
#output:
#   List - List of complete http url.
```

Figure 9: Code

2.

```python
def make_absolute_url(url_list, url):
  downloaded_domain = http_client.parse_url(url).netloc
  complete_url = []
  for url in url_list:
    parsed_url = http_client.parse_url(url)
    if(parsed_url.netloc):
      complete_url.append(url)
    else:
      complete_url.append('http://' + downloaded_domain + url )
  return complete_url

# Reads the html files extracts all html links to image and download it and saves it to the disc and prints all the URLs at the
  console.
# input:
#   file_name - Name of the file in which html page is saved-
#   url:String - Url of the resource from where main file was  downloaded.
#output:
#    None
def download_urls(file_name, url):
  html_data = read_data(file_name)
  url_list = match_img_src(html_data)
  full_url_list = make_absolute_url(url_list, url)
  print('List of URLs extracted from downloaded html')
  print(full_url_list)
  for url in full_url_list:
    print('\ndownloading url')
    http_client.call_http_server(url, False)

if __name__ == "__main__":
  download_urls(sys.argv[1], sys.argv[2])
```

Figure 10: Code

Output:

1.

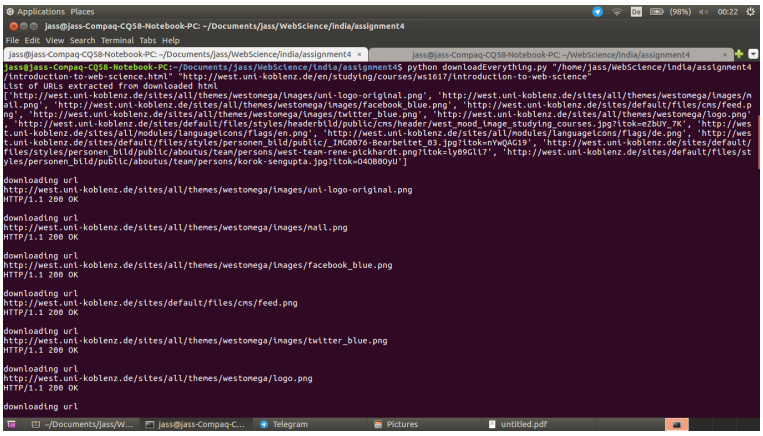


Figure 11: Output

2.

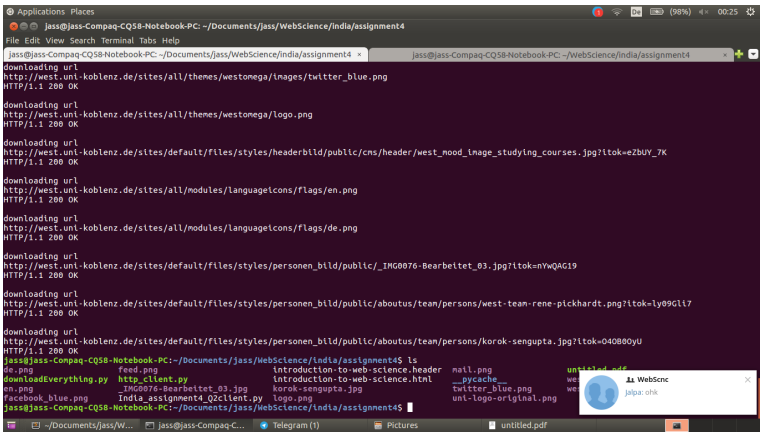


Figure 12: Output

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment4/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.