Introduction to Web Science

Assignment 5

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 30, 2016, 10:00 a.m.
Tutorial on: December 2, 2016, 12:00 p.m.

Please look at the lessons 1) **Dynamic Web Content** & 2) **How big is the Web?**

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: Golf

Members: Atique Baig, Mtarji Adam, Deepak Garg

1 Creative use of the Hyptertext Transfer Protocol (10 Points)

HTTP is a request response protocol. In that spirit a client opens a TCP socket to the server, makes a request, and the server replies with a response. The server will just listen on its open socket but cannot initiate a conversation with the client on its own.

However you might have seen some interactive websites which notify you as soon as something happens on the server. An example would be Twitter. Without the need for you to refresh the page (and thus triggering a new HTTP request) they let you know that there are new tweets available for you. In this exercise we want you to make sense of that behaviour and try to reproduce it by creative use of the HTTP protocol.

Have a look at `server.py`[1] and `webclient.html` (which we provide). Extend both files in a way that after `webclient.html` is servered to the user the person controlling the server has the chance to make some input at its commandline. This input should then be send to the client and displayed automatically in the browser without requiring a reload. For that the user should not have to interact with the webpage any further.

1.1 webclient.html

```
1: <html>
2: <head>
3:         <title>Abusing the HTTP protocol - Example</title>
4: </head>
5: <body>
6:         <h1>Display data from the Server</h1>
7:         The following line changes on the servers command line
8:         input: <br>
9:         <span id="response" style="color:red">
10:                 This will be replaced by messages from the server
11:        </span>
12: </body>
13: </html>
```

1.2 Hints:

- This exercise is more like a riddle. Try to focuse on how TCP sockets and HTTP work and how you could make use of that to achieve the expected behaviour. Once you have an idea the programming should be straight-forward.
- The Javascript code that you need for this exercise was almost completely shown in one of the videos and is available on Wikiversity.

[1] you could store the code from http://blog.wachowicz.eu/?p=256 in a file called `server.py`

- In that sense we only ask for a ”proof of concept” nothing that would be stable out in the wilde.
  - In particular, don't worry about making the server uses multithreading. It is ok to be blocking for the sake of this exercise.
- Without use of any additional libraries or AJAX framework we have been able to solve this with 19 lines of Javascript and 11 lines of Python code (we provide this information just as a way for you to estimate the complexity of the problem, don't worry about how many lines your solution uses).

Solution:
We use long polling by continuously sending request to the server for new broadcast.
HTML :

```
1: <html >
2:  <head >
3:  <title >Abusing the HTTP protocol - Example </title >
4:  </head >
5:  <body >
6:  <h1 >Display data from the Server </h1 >
7:  The following line changes on the servers command line
8:  input: <br >
9:  <span id="response" style =" color:red">
10:  This will be replaced by messages from the server
11:  </span >
12:  <script>
13:
14: function polling(){
15:         console.log('Sending request');
16:         if (window.XMLHttpRequest) { // code for IE7+, Firefox, Chrome, Opera, Sa
17:                                 var xhr = new XMLHttpRequest();
18:                         }
19:                         else { // code for IE6, IE5
20:                                 var xhr = new ActiveXObject("Microsoft.XMLHTTP");
21:                         }
22:                         xhr.open('GET', 'last_broadcast', true);
23:                         xhr.timeout = 15000; // Give 10s timeout for the server
24:                         xhr.onreadystatechange = function(){
25:                                 if (xhr.readyState == 4 ){
26:                                         if(xhr.status == 200){
27:                                                         document.getElementById("
28:                                         }
29:                                 setTimeout(polling(), 15000);
30:                                 }
31:                         };
32:                         xhr.ontimeout = function () {
33:                           console.log('request timed out, calling poll again in 5s
34:                           setTimeout(polling(), 15000);
```

```
35:                             };
36:                             xhr.send();
37:                             console.log('Polling done, Waiting for next poll');
38:
39:                     }
40:             window.onload = polling()
41:
42: </script>
43:  </body >
44:  </html >
```

server.py:

```
# -*- coding: utf-8 -*-
"""
Introduction to Web Science
Assignment 5
Question 1
Team : golf

**Comments starting with 'A51' are the extend parts

Script description :
    we extend the giving server script to be able to handle a special root
    MO: we use long polling, and prompt the server for an input when the client
    request the root '/last_broadcast'


"""


import socket  # Networking support
import signal  # Signal support (server shutdown on signal receive)
import time    # Current time

class Server:
 """ Class describing a simple HTTP server objects."""

 def __init__(self, port = 80):
     """ Constructor """
     self.host = ''   # <-- works on all avaivable network interfaces
     self.port = port
     self.www_dir = 'www' # Directory where webpage files are stored
     # A51 : Used to store the last broadcast by the server admin
```

```
        self.last_broadcast=''

    def activate_server(self):
        """ Attempts to aquire the socket and launch the server """
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try: # user provided in the __init__() port may be unavaivable
            print("Launching HTTP server on ", self.host, ":",self.port)
            self.socket.bind((self.host, self.port))

        except Exception as e:
            print ("Warning: Could not aquire port:",self.port,"\n")
            print ("I will try a higher port")
            # store to user provideed port locally for later (in case 8080 fails)
            user_port = self.port
            self.port = 8080

            try:
                print("Launching HTTP server on ", self.host, ":",self.port)
                self.socket.bind((self.host, self.port))

            except Exception as e:
                print("ERROR: Failed to acquire sockets for ports ", \
                      user_port, " and 8080. ")
                print("Try running the Server in a privileged user mode.")
                self.shutdown()
                import sys
                sys.exit(1)

        print ("Server successfully acquired the socket with port:", self.port)
        print ("Press Ctrl+C to shut down the server and exit.")
        self._wait_for_connections()

    def shutdown(self):
        """ Shut down the server """
        try:
            print("Shutting down the server")
            s.socket.shutdown(socket.SHUT_RDWR)

        except Exception as e:
            print("Warning: could not shut down the socket."\
            " Maybe it was already closed?",e)

    def _gen_headers(self,  code):
        """ Generates HTTP response Headers. Ommits the first line! """
```

```
        # determine response code
        h = ''
        if (code == 200):
            h = 'HTTP/1.1 200 OK\n'
        elif(code == 404):
            h = 'HTTP/1.1 404 Not Found\n'

        # write further headers
        current_date = time.strftime("%a, %d %b %Y %H:%M:%S", time.localtime())
        h += 'Date: ' + current_date +'\n'
        h += 'Server: Simple-Python-HTTP-Server\n'
        h += 'Connection: close\n\n'

        return h

    def _wait_for_connections(self):
        """ Main loop awaiting connections """
        while True:
            print ("Awaiting New connection")
            self.socket.listen(3) # maximum number of queued connections

            conn, addr = self.socket.accept()
            # conn - socket to client
            # addr - clients address

            print("Got connection from:", addr)

            data = conn.recv(1024) #receive data from client
            string = bytes.decode(data) #decode it to string

            #determine request method  (HEAD and GET are supported)
            request_method = string.split(' ')[0]
            print ("Method: ", request_method)
            print ("Request body: ", string)

            #if string[0:3] == 'GET':
            if (request_method == 'GET') | (request_method == 'HEAD'):
                #file_requested = string[4:]

                # split on space "GET /file.html" -into-> ('GET','file.html',...)
                file_requested = string.split(' ')
                file_requested = file_requested[1] # get 2nd element
```

```
#Check for URL arguments. Disregard them
# disregard anything after '?'
file_requested = file_requested.split('?')[0]

# if we receive a '/last_broadcast' GET request
# we prompt the server for an input to broadcast to the client
if (file_requested == '/last_broadcast'):
     if (request_method == 'GET'):
         response_content = input('Client requesting'\
                                  'broadcast!\n').encode()
         response_headers = self._gen_headers( 200)
else:
    # in case no file is specified by the browser
    if (file_requested == '/'):
        # load index.html by default
        file_requested = '/webclient.html'

    file_requested = self.www_dir + file_requested
    print ("Serving web page [",file_requested,"]")

    ## Load file content
    try:
        file_handler = open(file_requested,'rb')
         #only read the file when GET
        if (request_method == 'GET'):
             # read file content
            response_content = file_handler.read()
        file_handler.close()

        response_headers = self._gen_headers( 200)
    #in case file was not found, generate 404 page
    except Exception as e:
        print ("Warning, file not found. Serving"\
               "response code 404\n", e)
        response_headers = self._gen_headers( 404)

        if (request_method == 'GET'):
           response_content = b"<html><body><p>Error 404:"\
           "File not found</p><p>Python HTTP "\
           "server</p></body></html>"
# return headers for GET and HEAD
server_response =  response_headers.encode()
if (request_method == 'GET'):
    # return additional conten for GET only
```

```
                server_response +=  response_content

            conn.send(server_response)
            print ("Closing connection with client")
            conn.close()

        else:
            print("Unknown HTTP request method:", request_method)

def graceful_shutdown(sig, dummy):
    """ This function shuts down the server. It's triggered
    by SIGINT signal """
    s.shutdown() #shut down the server
    import sys
    sys.exit(1)


###########################################################
# shut down on ctrl+c
signal.signal(signal.SIGINT, graceful_shutdown)
print ("Starting web server")
s = Server(80)  # construct server object
s.activate_server() # aquire the socket
```

Screenshots:

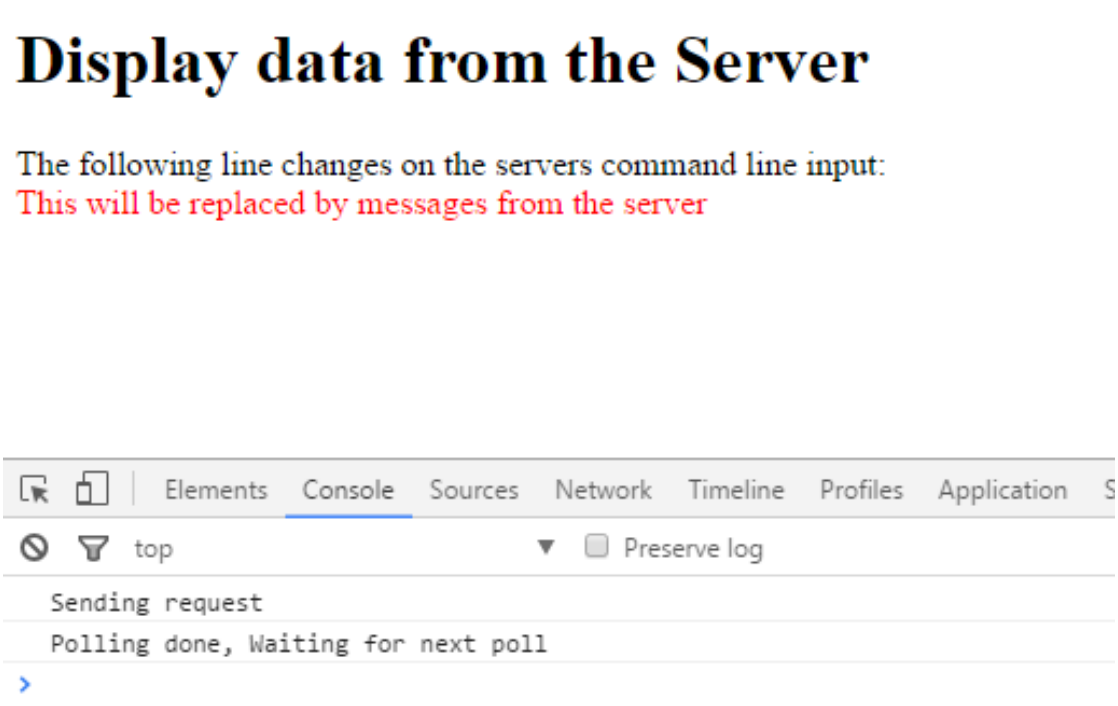

Figure 1: Serving html and calling the poll function

```
Serving web page [ www/webclient.html ]
Closing connection with client
Awaiting New connection
Got connection from: ('127.0.0.1', 53357)
Method:  GET
Request body:  GET /last_broadcast HTTP/1.1
Host: localhost
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) App
Chrome/54.0.2840.99 Safari/537.36
Accept: */*
Referer: http://localhost/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4



Client requestingbroadcast!
hi
```

Figure 2: Input broadcast from sever

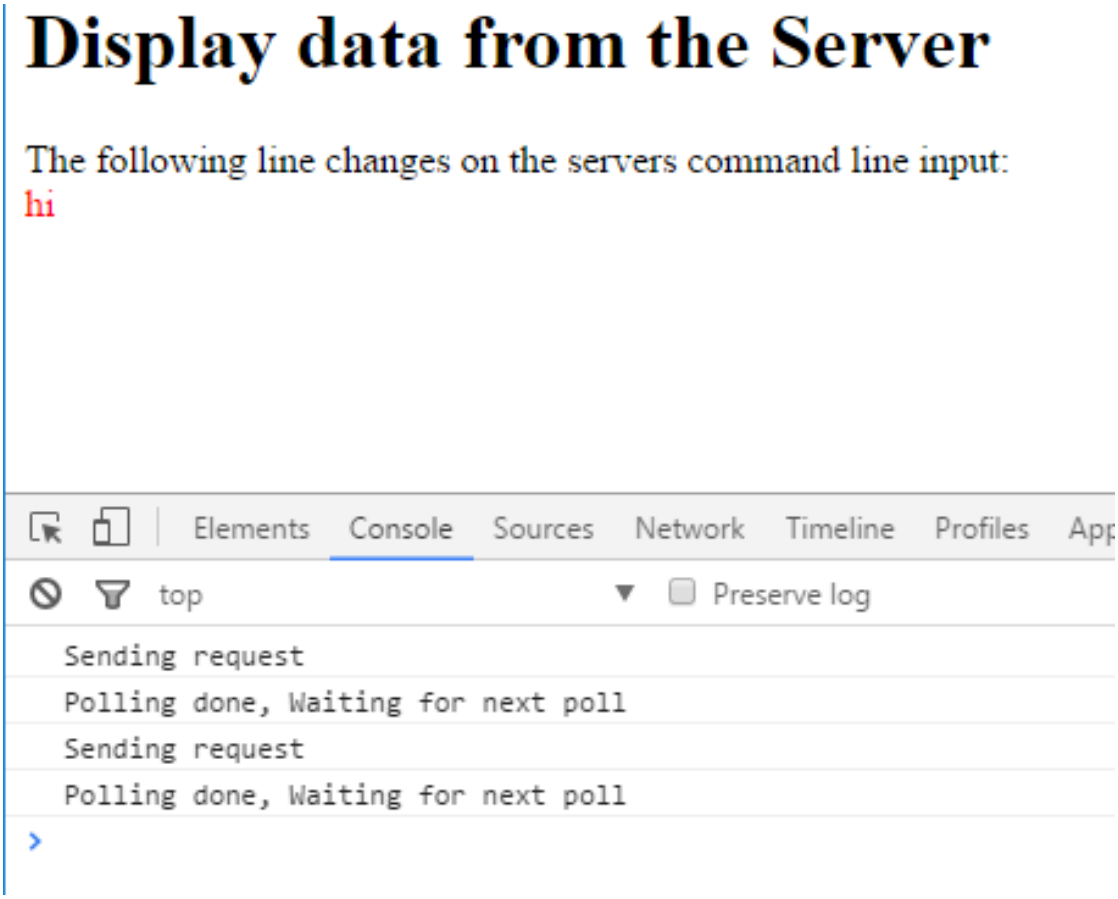

Figure 3: Update on clients

2 Web Crawler (10 Points)

Your task in this exercise is to "crawl" the `Simple English Wikipedia`. In order to execute this task, we provide you with a mirror of the Simple English Wikipedia at `141.26.208.82`.

You can start crawling from `http://141.26.208.82/articles/g/e/r/Germany.html` and you can use the `urllib` or `doGetRequest` function from the last week's assignment.

Given below is the strategy that you might adopt to complete this assignment:

1. Download `http://141.26.208.82/articles/g/e/r/Germany.html` and store the page on your file system.
2. Open the file in python and extract the local links. (Links within the same domain.)
3. Store the file to your file system.
4. Follow all the links and repeat steps 1 to 3.
5. Repeat step 4 until you have downloaded and saved all pages.

2.1 Hints:

- Before you start this exercise, please have a look at Exercise 3.
- Make really sure your crawler doesn't follow external urls to domains other than `http://141.26.208.82`. In that case you would start crawling the entire web
- Expect the crawler to run about 60 Minutes if you start it from the university network. From home your runtime will most certainly be even longer.
- It might be useful for you to have some output on the crawlers commandline depicting which URL is currently being fetched and how many URLs have been fetched so far and how many are currently on the queue.
- You can (but don't have to) make use of breadth-first search.
- It probably makes sense to take over the full paths from the pages of the Simple English Wikipedia and use the same folder structure when you save the html documents.
- You can (but you don't have to) speed up the crawler significantly if you use multithreading. However you should not use more than 10 threads in order for our mirror of Simple English Wikipedia to stay alive.

Screenshots:

```
http://141.26.208.82/articles/g/e/r/Germany.html;54;572
http://141.26.208.82/articles/f/l/a/Image%7EFlag_of_Lower_Saxony.svg_0a8f.html;32;27
http://141.26.208.82/articles/t/e/s/User%7ETest-tools_5bd3.html;9;14
http://141.26.208.82/articles/j/a/n/Jan_Mayen_c355.html;16;175
http://141.26.208.82/articles/r/u/s/Russia.html;17;418
http://141.26.208.82/articles/i/t/a/Italy.html;41;449
http://141.26.208.82/articles/g/u/a/User%7EGuanabot_b902.html;9;25
http://141.26.208.82/articles/t/u/v/User%7ETuvicBot_bacc.html;9;89
http://141.26.208.82/articles/j/o/h/Johann_Sebastian_Bach_d0b5.html;23;261
http://141.26.208.82/articles/r/e/p/Republic_of_Bulgaria_696a.html;0;1
http://141.26.208.82/articles/l/a/t/Latvia.html;18;324
http://141.26.208.82/articles/n/o/r/Norway.html;23;362
http://141.26.208.82/articles/c/o/u/Courts_of_Germany_6acc.html;21;52
http://141.26.208.82/articles/h/o/r/Horst_K%C3%B6hler_7544.html;12;136
http://141.26.208.82/articles/g/e/r/German_reunification.html;9;92
http://141.26.208.82/articles/m/a/n/User%7EManecke_76da.html;25;52
http://141.26.208.82/articles/r/i/c/User%7ERicky81682_f4ed.html;10;53
```

Figure 4: Result file for our crawler

webcrawler:

```python
# -*- coding: utf-8 -*-
"""
Introduction to Web Science
Assignment 5
Question 2
Team : golf

Script description :
    This script crawls the domain http://141.26.208.82 starting from the
    url : http://141.26.208.82/articles/g/e/r/Germany.html
    it counts the number of external and internel links on each web pages
    and writes the result into a txt file
"""

import os
from urllib import parse,request as req,error
import re
import logging

# we initialize the logging parameter
logging.basicConfig(filename='webCrawler.log',
                    format='%(asctime)s -  %(levelname)s - %(message)s',
                    level=logging.DEBUG)
# we clear the lof file and results file for a fresh start
with open('webCrawler.log', 'w'):
    pass
```

```
with open('webpage_results.txt', 'w'):
    pass

"""
WebPage class is used to store information about the webpage,
it inherits from Object class to be able to overide and use __eq__
    link : the url of the page
    valide_link : boolean holding information about link accessibility
    internal_links : a set of WebPage objects for internal links
    nbr_internal_links : hold the number of internal links
    external_links : a set of WebPage objects for external links
    nbr_external_links : number of external links
"""
class WebPage(object):
    # we initiliaze the default value and log that we are in a new webpage
    def __init__(self,url):
        self.link=url
        self.parsed_link = parse.urlparse(url)
        self.valid_link=True
        self.internal_links=set()
        self.nbr_internal_links= 0
        self.external_links=set()
        self.nbr_external_links= 0
        logging.info('New webpage added : %s'%url)

    # function used to store the external and internal links as well as
    # count them
    def stats(self,ext_links,int_links):
        self.internal_links=set(int_links)
        self.nbr_internal_links=len(self.internal_links)
        self.external_links=set(ext_links)
        self.nbr_external_links=len(self.external_links)
        logging.info('webpage {0} has {1} external links and {2} internal links'
                     .format(self.link,self.nbr_external_links,
                             self.nbr_internal_links))
        print('webpage {0} has {1} external links and {2} internal links'
                     .format(self.link,self.nbr_external_links,
                             self.nbr_internal_links))

        return self.nbr_external_links,self.nbr_internal_links
    # overiding __str__ to return link
    def __str__(self):
        return self.link
```

```
    # overiding __eq__ to compare using link
    def __eq__(self, other):
        return self.link == other.link
    # implementing hash function on the link
    def __hash__(self):
        return hash(self.link)

"""
our WebCrawler class, it initializes a number of global variables to keep
track of the crawled pages.
starting_link : the starting point for our crawler
root_dir : the current working directory
nbr_wpages : the number of webpages crawled (html files)
nbr_external_links : number of external links parsed so far
nbr_internal_links : number of internal links parsed so far
webpage_rslt : the result for each web pages, used later for ploting
"""
class WebCrawler:

    def __init__(self,link):
        self.starting_link = link
        self.parsed_link = parse.urlparse(link)
        self.root_dir = os.path.dirname(os.path.realpath(__file__))
        self.nbr_wpages=0
        self.nbr_external_links=0
        self.nbr_internal_links=0
        self.webpage_rslt=''

    # we reuse the function to complete the relative urls
    def create_full_url(self,link):
        url = link
        path = self.parsed_link.path
        hostname = "http://"+self.parsed_link.hostname
        if path == "":
            path = "/"
        # absolut path
        if link[0]=="/":
            url = hostname + link
        #fully qualified domain name
        elif len(link) > 7 and link[0:7]=="http://":
            url = link
        # relative path
        else:
            parentfolders = 0
```

```
            while len(link)>3 and link[0:3]=="../":
                link = link[3:]
                parentfolders= parentfolders + 1
                url = (hostname + "/".join(path.split("/")
                [0:-(1+parentfolders)])+"/"+link)
        return url

    # split_ext_int_links function allows to return a set of external links
    # and a set of internal links from an array of links
    def split_ext_int_links(self,links):
        ext_links=[]
        int_links=[]
        for l in links:
            # for each link we create a WebPage object
            wp = WebPage(l)
            # checking if it hase the same domain name or not
            if(wp.parsed_link.hostname == self.parsed_link.hostname):
                int_links.append(wp)
            else:
                ext_links.append(wp)
        return ext_links,int_links

    # main function for for crawling a giving page
    def crawl_page(self,web_page):
        # we determine the file_name (we decode the path for special characters)
        file_name=self.root_dir+parse.unquote(web_page.parsed_link.path)
        try:
            # attempting to open the url
            res = req.urlopen(web_page.link)
            # we store the headers received in this variable
            headers = res.info()
            # if the response code is 200
            if(res.getcode() == 200):
                logging.info('webpage %s returned status '\
                             'code 200'%web_page.link)
                # we extract the directory from the file name
                directory = os.path.dirname(file_name)
                # if the directory does not exist we create it
                if not os.path.exists(directory):
                    os.makedirs(directory)
                # we write the content of the filename as bynarie
                with open(file_name,'wb') as of:
                    of.write(res.read())
            # incase we receive a status code diffrent then 200
```

```
            else:
                web_page.valid_link=False
                logging.warning('webpage %s is not accessible, '\
                                'returned status code %s'\
                                %(web_page.link,res.getcode()))
                return
            # now we check if the content is an html file
            if(headers['Content-Type'] == 'text/html'):
                # we increment the number of webpages
                self.nbr_wpages+=1
                html=''
                # we extract the encoding used, if no encoding specified
                # we fall on utf8 as its the most common
                encoding_used=headers['charset'] if headers['charset'] else 'utf8'
                # we read the file (not optimal for looping, would be better to
                # store the html in a variable)
                with open(file_name,'rb') as f:
                    html=f.read().decode(encoding_used)
                # we extract all links in 'href' attributes
                links=re.findall(r'href=[\'"]?([^\'" >]+)',html)
                # creating full urls
                full_urls=[self.create_full_url(link) for link in links]
                # spliting the links into external and internal
                elinks,ilinks= self.split_ext_int_links(full_urls)
                # storing the links in the WebPage object
                nbr_elinks,nbr_ilinks=web_page.stats(elinks,ilinks)
                # appending the results in the webpage_reslt list
                # we use the ';' as seperator
                self.webpage_rslt+=(web_page.link+';'+\
                                    str(web_page.nbr_external_links)+\
                                    ';'+str(web_page.nbr_internal_links)+'\n')
                self.nbr_external_links+=nbr_elinks
                self.nbr_internal_links+=nbr_ilinks
                return web_page
        except error.URLError as e:
            logging.error('webpage %s not found'%web_page.link)
        except Exception as e:
            print(e)
            logging.error('webpage %s not accessilbe'%web_page.link)

    # save_results simply writes the final results in 2 files for later use
    def save_results(self):
        # we concatenate the results and use a ';'
        final_results=self.starting_link+';'+str(self.nbr_wpages)+';'\
```

```
            +str(self.nbr_external_links)+';'+str(self.nbr_internal_links)
        with open('final_results.txt','a') as f:
            f.write(final_results)
        with open('webpage_results.txt','a') as f:
            f.write(self.webpage_rslt)
        logging.info('Starting page: %s\n Nummber of webpages:%s\n Number '\
                     'of external links:%s\n Number of internal links:%s'\
                     %(self.starting_link,self.nbr_wpages,\
                       self.nbr_external_links,self.nbr_internal_links))

    # main  funtion to start the web crawler, it follows a breath first
    # search algorithm by using a set of visited links
    # and an array as of WebPage objects as a queue
    def crawl(self):
        # WebPage object as starting point
        starting_web_page=WebPage(self.starting_link)
        # we initialize the visited WebObject list and the queue
        # this is why we need to overide the __eq__ function
        # since we use mainly the link variable of WebPage for comparison
        visited, queue = set(), [starting_web_page]
        # while there are objects in queue
        while queue:
            # we pop the first element in queue
            wp = queue.pop(0)
            # if its not a visited WebPage we proceed
            if wp not in visited:
                # add the page to the visited links
                visited.add(wp)
                # crawl the page for internal links
                crawled_page=self.crawl_page(wp)
                # the crawled_page return a value if its an html webpage
                # not a media file or other
                if(crawled_page):
                    queue.extend(crawled_page.internal_links - visited)
            if(self.nbr_wpages == 5000):
                break
        # at the end of the crawling process we call the save_results function
        self.save_results()

# we start the program
logging.info('Starting web crawler')
# Creating a web crawler starting from the given link
web_crawler=WebCrawler('http://141.26.208.82/articles/g/e/r/Germany.html')
# calling the crawl function
```

```
web_crawler.crawl()
logging.info('Web crawler finished')
```

3 Web Crawl Statistics (10 Points)

If you have successfully completed the first exercise of this assignment, then please provide the following details. You may have to tweak your code in the above exercise for some of the results.

3.1 Phase I

1. Total Number of *webpages* you found.
2. Total number of links that you encountered in the complete process of crawling.
3. Average and median number of links per web page.
4. Create a *histogram* showing the distribution of links on the crawled web pages. You can use a bin size of 5 and scale the axis from 0-150.

3.2 Phase II

1. For every page that you have downloaded, count the number of internal links and external links.
2. Provide a *scatter plot* with number of internal links on the X axis and number of external links on the Y axis.

Screenshots:

```
Total number of webpages : 3136
Total number of links : 599502
Average number of links : 191.16772959183675
```

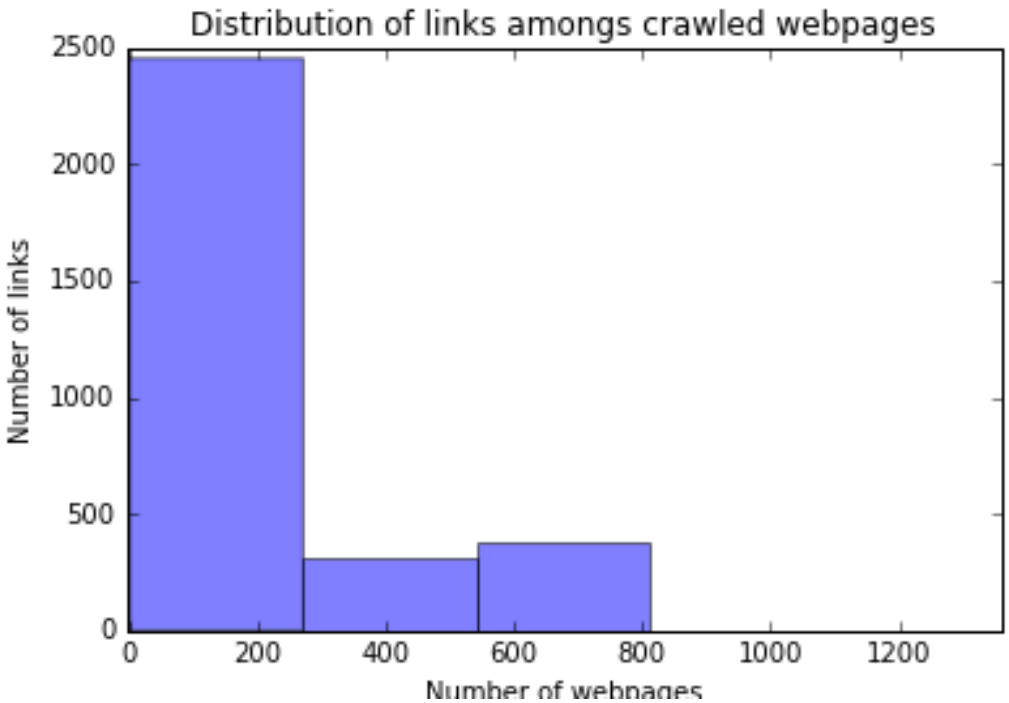

Figure 5: Histogram of the distribution of links

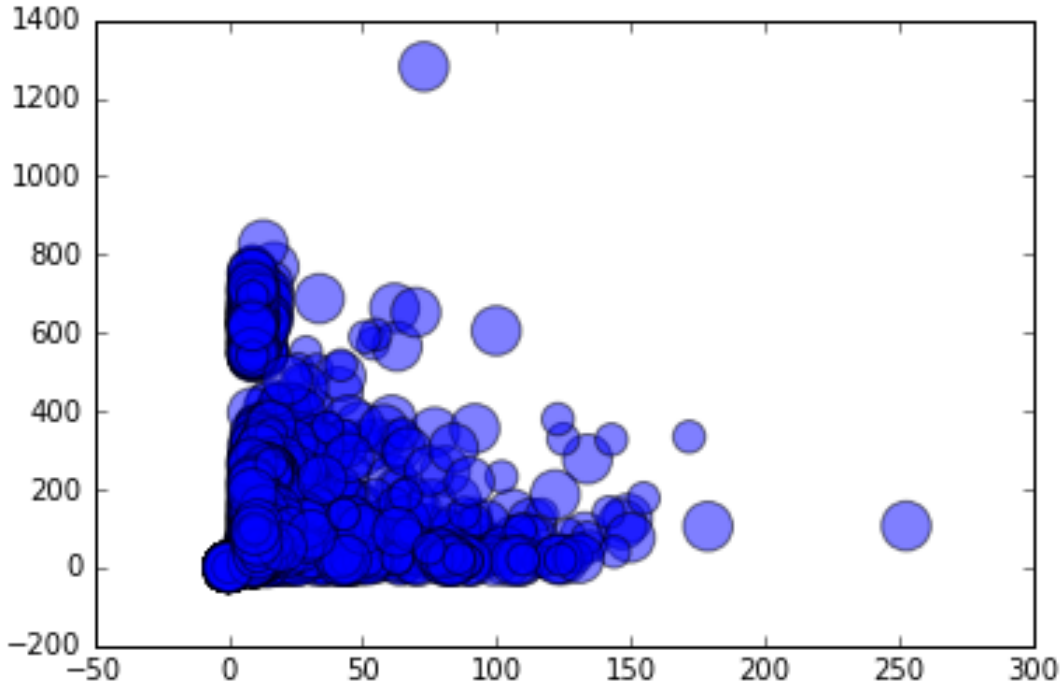

Figure 6: Scatter plot

stats.py:

```
# -*- coding: utf-8 -*-
"""
Introduction to Web Science
Assignment 5
Question 3
Team : golf

Script description :
    This scripts read the result file of the previous question and plots
    a histogram and a scatter plot
"""

import numpy as np
import matplotlib.pyplot as plt

# function used to calculate the median
def getmedian(lst):
    sortedLst = sorted(lst)
    lstLen = len(lst)
    index = (lstLen - 1) // 2
```

```
    if (lstLen % 2):
        return sortedLst[index]
    else:
        return (sortedLst[index] + sortedLst[index + 1])/2.0
# function used to draw a histogram
def draw_hist(data):
    plt.xlim([min(data)-5, max(data)+5])
    plt.hist(data, bins=5, alpha=0.5)
    plt.title('Distribution of links amongs crawled webpages')
    plt.xlabel('Number of webpages')
    plt.ylabel('Number of links')
    plt.show()
# function used to plot scatter
def draw_scatter(x,y):
    N = 2 # dimension
    area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radiuses
    plt.scatter(x, y, s=area,  alpha=0.5)
    plt.show()

# recovers our webpage results from the file
wps=[]
with open('webpage_results.txt', 'r') as f:
    wps = f.read().splitlines()
# splitting using the ';' seperator
# index 0 is the web page, 1 is the number of external links and
# 2 is the number of internal links
m =np.array(list(wp.split(';') for wp in wps))
# links
x=m[:,0]
# nbr external links per webpage
ey=list((int(k) for k in m[:,1]))
# nbr internal links per webpage
iy=list((int(k) for k in m[:,2]))
# total nbr of links per webpage
data=[a + b for a,b in zip(iy, ey)]
# total number of webpages
total=len(x)
# total umber of links
nbr_links=sum(data)
# average
average=nbr_links/total
# median
median=getmedian(data)
print('Total number of webpages : %s'%total)
```

```
print('Total number of links : %s'%nbr_links)
print('Average number of links : %s'%average)

draw_hist(data)
draw_scatter(iy,ey)
```

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment5/` 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`.