

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: papa

Members: Brigitte Aznar

Bonasmitha Behura

Ilia Tugushi

1 Creative use of the Hypertext 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 served to the user the person controlling the server has the chance to make some input at its commandline. This input should then be sent 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 focus 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.

¹you 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).

Answer: For the sake of saving paper, we are only pasting here the relevant part of the code for the exercise.

```
1: while True:
2:     print ("Awaiting New connection")
3:     self.socket.listen(3) # maximum number of queued connections
4:
5:     conn, addr = self.socket.accept()
6:     # conn - socket to client
7:     # addr - clients address
8:
9:     data = conn.recv(1024) #receive data from client
10:    string = bytes.decode(data) #decode it to string
11:
12:    # split on space "GET /file.html" -into-> ('GET','file.html',...)
13:    file_requested = string.split(' ')
14:    file_requested = file_requested[1] # get 2nd element
15:
16:    # Check for URL arguments. Disregard them
17:    file_requested = file_requested.split('?')[0] # disregard anything after
18:
19:    if (file_requested == '/'): # in case no file is specified by the brows
20:        file_requested = '/index.html' # load index.html by default
21:
22:    file_requested = self.www_dir + file_requested
23:    file_requested = file_requested.strip("/")
24:    print("Serving web page [", file_requested, "]")
25:
26:    ## Load file content
27:    try:
28:        file_handler = open(file_requested,'rb')
29:        if (request_method == 'GET'): #only read the file when GET
30:            response_content = file_handler.read() # read file content
31:            file_handler.close()
32:
33:        response_headers = self._gen_headers( 200)
34:
```

```
35:         except Exception as e: #in case file was not found, generate 404 page
36:             if file_requested == "server":
37:                 inpt = input("Say somethin nice to the user: ")
38:                 conn.send(self._gen_headers( 200).encode() + inpt.encode())
39:                 conn.close()
40:                 continue
41:             else:
42:                 print ("Warning, file not found. Serving response code 404\n", e)
43:                 response_headers = self._gen_headers( 404)
44:
45:                 #if (request_method == 'GET'):
46:                 response_content = b"<html><body><p>Error 404: File not found</p>"
47:
48:
49:                 server_response = response_headers.encode() # return headers for GET and
50:                 #if (request_method == 'GET'):
51:                 server_response += response_content # return additional conten for GET
52:
53:                 conn.send(server_response)
54:                 print ("Closing connection with client")
55:                 conn.close()
```

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.

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.

Answer:

Code for parts II and III

```
1: import os
2: import re
3: import urllib
4: import numpy as npy
5: from collections import deque
6: import matplotlib.pyplot as plt
7: from urllib.parse import urljoin
8: import matplotlib.patches as mpatches
9:
10: total_num_links = []
11: num_external_links = []
12: num_internal_links = []
13:
14:
15: # def get_links(file_name, qurl):
16: def get_links(data, qurl):
17:     links = []
18:     links_to_return = []
19:     internal = 0
```

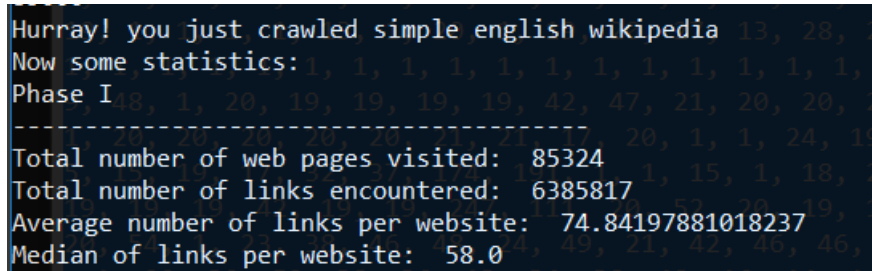
```
20:     external = 0
21:     domain = 'http://141.26.208.82'
22:
23:     # regex to check the a tag
24:     hrefs = re.compile(r'\<s*a [^>]*href="([^\"]+)"')
25:
26:     # with open(file_name, encoding="utf8") as file:
27:     #     for line in file:
28:     #         reg = hrefs.findall(line)
29:     #         if reg:
30:     #             links.extend(reg)
31:
32:     regex = hrefs.findall(data)
33:     if regex:
34:         links.extend(regex)
35:
36:     for link in links:
37:         if link.find(domain) > -1 or link.find('http') != 0:
38:             internal += 1
39:             #Dont include links that starts with #
40:             if link.find('#') != 0:
41:                 link = urljoin(qurl, link)
42:                 links_to_return.append(link)
43:             else:
44:                 external += 1
45:
46:     num_external_links.append(external)
47:     num_internal_links.append(internal)
48:     total_num_links.append(internal + external)
49:
50:     return links_to_return
51:
52: def plot():
53:     plt.scatter(num_internal_links, num_external_links)
54:
55:     # adds labels to the axis
56:     plt.xlabel('Internal Links')
57:     plt.ylabel('External Links')
58:
59:     # generates legend
60:     legend = mpatches.Patch(color='blue', label='Number of link in page')
61:     plt.legend(handles=[legend])
62:     plt.title("Number of Internal and External links in pages")
63:     plt.show()
64:     return plt
65:
66: def histogram(total_links):
67:     total_links = npy.transpose(total_links)
68:     plt.hist(total_links, bins=5, range=(0,150))
```

```
69:     plt.title("Histogram with 'auto' bins")
70:     return plt
71:
72: def main():
73:
74:     path_to_file = os.path.realpath('.') + '\\wikipedia_crawl'
75:     os.makedirs(path_to_file, exist_ok=True)
76:     url = 'http://141.26.208.82/articles/g/e/r/Germany.html'
77:     #url = 'http://localhost/wikipedia-simple-html/simple/articles/g/e/r/Germany.'
78:     queue = deque([url])
79:     visited = set()
80:     web_page = set()
81:
82:     while True:
83:         if queue:
84:             element = queue.popleft()
85:             file_name = element.split('/')[-1]
86:             if file_name.find(".html") == -1:
87:                 visited.add(element)
88:                 continue
89:             if element not in visited:
90:                 try:
91:                     response = urllib.request.urlopen(element)
92:                     data = response.read()
93:                     response.close()
94:                     visited.add(element)
95:                     web_page.add(element)
96:
97:                     with open(os.path.join(path_to_file, file_name), 'wb') as file_to_write:
98:                         file_to_write.write(data)
99:                         file_to_write.close()
100:
101:                     if len(web_page) % 1000 == 0:
102:                         print("pages downloaded: ", len(web_page))
103:                         print(element)
104:                         queue.extend(get_links(data.decode(), element))
105:                 except Exception:
106:                     visited.add(element)
107:                     #print("Couldn't download " + element)
108:                     pass
109:             else:
110:                 break
111:
112:     print("Hurray! you just crawled simple english wikipedia")
113:
114:     total_web_pages = len(web_page)
115:     total_visited = sum(total_num_links)
116:     avg = total_visited / total_web_pages
117:     median = npy.median(total_num_links)
```



```
118:
119:     print("Now some statistics:")
120:     print("Phase I")
121:     print("-----")
122:     print("Total number of web pages visited: ", total_web_pages)
123:     print("Total number of links encountered: ", total_visited)
124:     print("Average number of links per website: ", avg)
125:     print("Median of links per website: ", median)
126:     hist = histogram(total_num_links)
127:     hist.show()
128:     scatter = plot()
129:     scatter.show()
130:
131: if __name__ == "__main__":
132:     main()
```

Outputs:



```
Hurray! you just crawled simple english wikipedia 13, 28, 2
Now some statistics: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
Phase I 48, 1, 20, 19, 19, 19, 19, 42, 47, 21, 20, 20, 3
-----
Total number of web pages visited: 85324 20, 1, 1, 24, 19
Total number of links encountered: 6385817 1, 15, 1, 18, 2
Average number of links per website: 74.84197881018237 19, 1
Median of links per website: 58.0 44, 49, 21, 42, 46, 46,
```

Figure 1: Results of the statistics

Being: Total number of websites visited: Total Websites downloaded

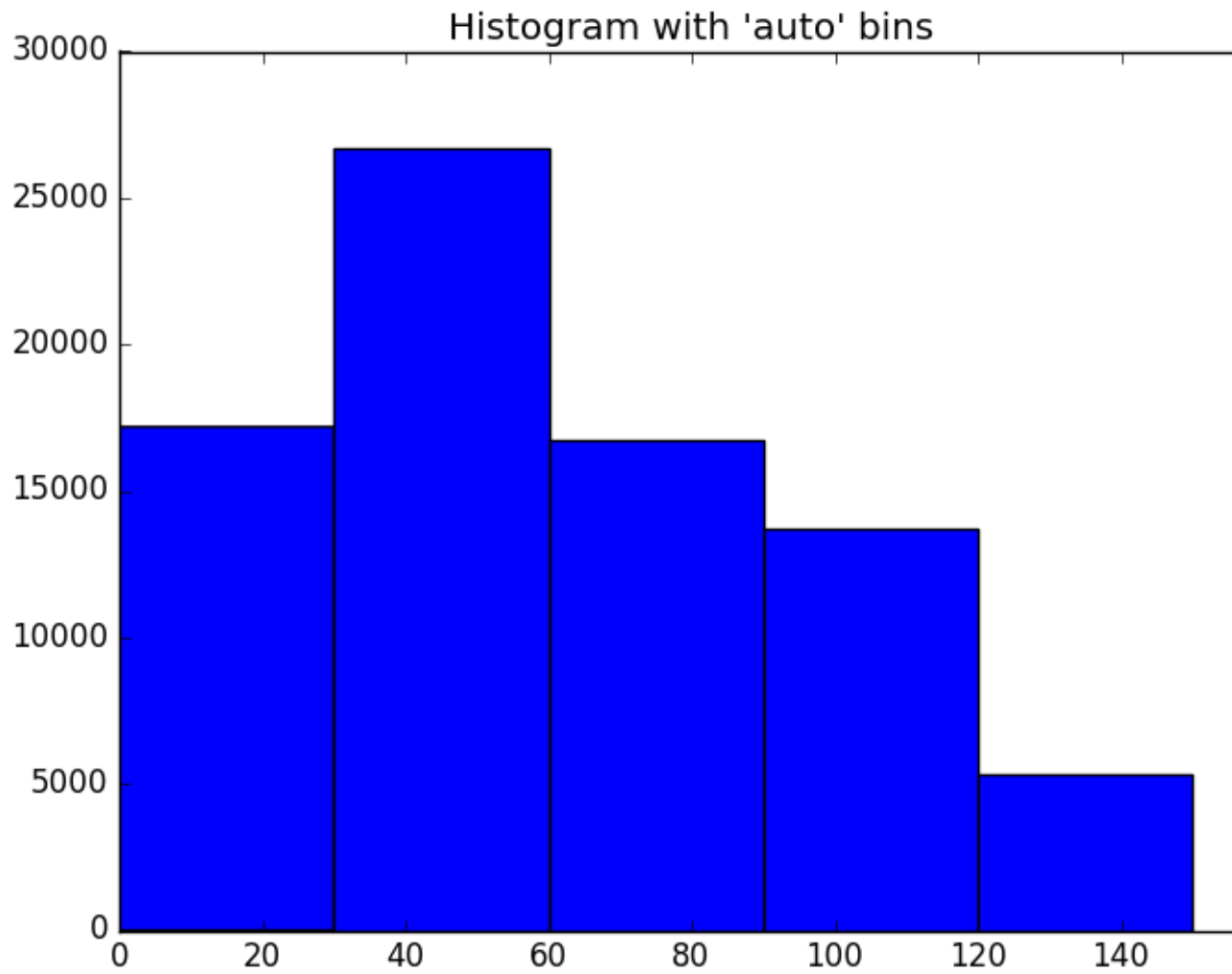


Figure 2: X axis number of websites, Y axis links encountered

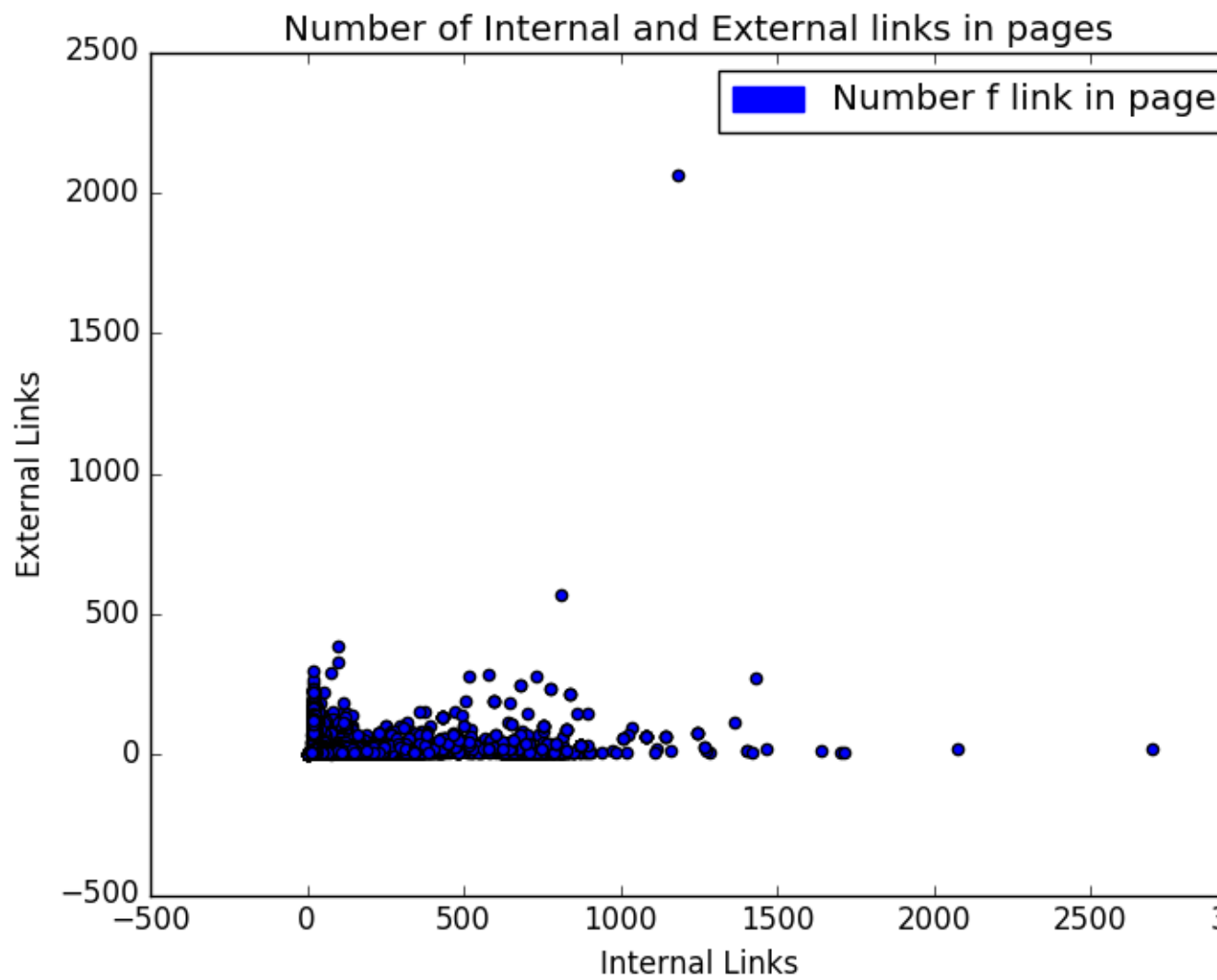


Figure 3: Internal Vs external links in pages

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 \LaTeX engine to LuaLaTeX.