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.

Group name: uniform
Group members: Pradip Giri, Jalak Arvind Kumar Pansuriya, Madhu Rakhal Magar

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

```
<html>
<head>
        <title>Abusing the HTTP protocol - Example</title>
</head>
<body>
        <h1>Display data from the Server</h1>
        The following line changes on the servers command line
        input: <br>
        <span id="response" style="color:red">
                This will be replaced by messages from the server
        </span>
</body>
</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).

Anwser

Listing 1 server.py

```
#!/usr/bin/python

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

prev_message = 'Hello';
message = ""
class Server:
 """ Class describing a simple HTTP server objects."""

 def __init__(self, port = 8080):
     """ Constructor """
     self.host = ''   # <-- works on all avaivable network interfaces
     self.port = port
     self.www_dir = '.' # Directory where webpage files are stored

 def handle_ajax(self, conn):
     global message, prev_message
     message = input("Your Message Here ")
     if not message == '' and not message == prev_message:
        prev_message = message
     else:
        message = prev_message

     response_headers = self._gen_headers( 200)
     response_content = response_headers.encode()
     response_content += message.encode()
     print(response_content)
     conn.send(response_content)
     # conn.close()

 def activate_server(self):
     """ Attempts to aquire the socket and launch the server """
```

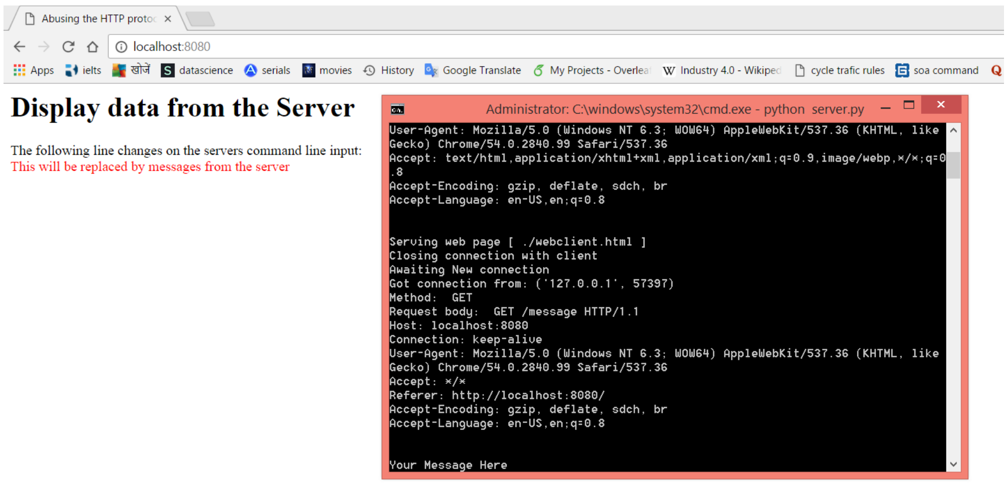
```
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try: # user provided in the __init__() port may be unavaivable
            print("Launching HTTP server on ", self.host, ":",self.port)
            self.socket.bind((self.host, self.port))

        except Exception as e:
            print ("Warning: Could not aquite 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, " an
                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 clo

    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'  # signal that the conection wil be closed afte

        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
                file_requested = file_requested.split('?')[0]  # disregard anything

                if file_requested == '/message':
                   self.handle_ajax(conn)
                else:
                    if (file_requested == '/'):  # in case no file is specified by t
                        file_requested = '/webclient.html' # load index.html by defa

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

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

```
                         response_content = file_handler.read() # read file conte
                     file_handler.close()

                     response_headers = self._gen_headers( 200)

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

                     if (request_method == 'GET'):
                        response_content = b"<html><body><p>Error 404: File not fo

                 server_response =  response_headers.encode() # return headers fo
                 if (request_method == 'GET'):
                     server_response +=  response_content  # return additional co
                 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(8080)  # construct server object
s.activate_server() # aquire the socket
```

Listing 2 webclient.html

```
<html>
<head>
  <title>Abusing the HTTP protocol - Example</title>
  <style>
    .alert {
      color: red;
    }
  </style>
</head>
<body>
  <div class="container">
```

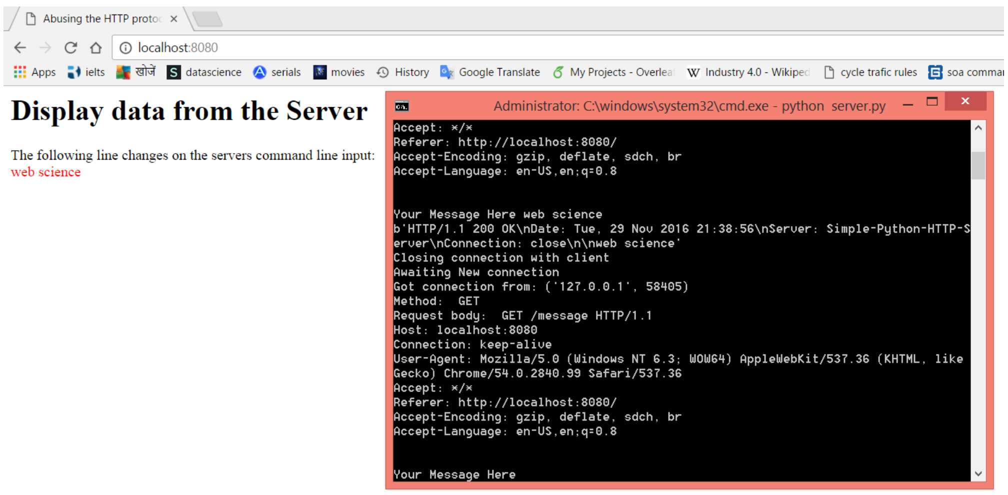
```
    <h1 class="h1">Display data from the Server</h1>
    The following line changes on the servers command line
    input: <br>
    <span id="response" class="alert">
      This will be replaced by messages from the server
    </span>
  </div>
  <script>
  var response = document.getElementById('response')
  function reqListener () {
    response.innerHTML = this.responseText;
  }
  function createRequest() {
    var oReq = new XMLHttpRequest();
    oReq.addEventListener("load", reqListener);
    oReq.open("GET", "http://localhost:8080/message");
    oReq.send();
  }
  setInterval(createRequest, 1000);
  </script>
</body>
</html>
```



Display data from the Server

The following line changes on the servers command line input:
This will be replaced by messages from the server

```
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/54.0.2840.99 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8


Serving web page [ ./webclient.html ]
Closing connection with client
Awaiting New connection
Got connection from: ('127.0.0.1', 57397)
Method:  GET
Request body:  GET /message HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/54.0.2840.99 Safari/537.36
Accept: */*
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8


Your Message Here
```



Display data from the Server

The following line changes on the servers command line input:
web science

```
Accept: */*
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8


Your Message Here web science
b'HTTP/1.1 200 OK\nDate: Tue, 29 Nov 2016 21:38:56\nServer: Simple-Python-HTTP-S
erver\nConnection: close\n\nweb science'
Closing connection with client
Awaiting New connection
Got connection from: ('127.0.0.1', 58405)
Method:  GET
Request body:  GET /message HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/54.0.2840.99 Safari/537.36
Accept: */*
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8


Your Message Here
```

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.

Answer:

Listing 3 crawlhelper.py

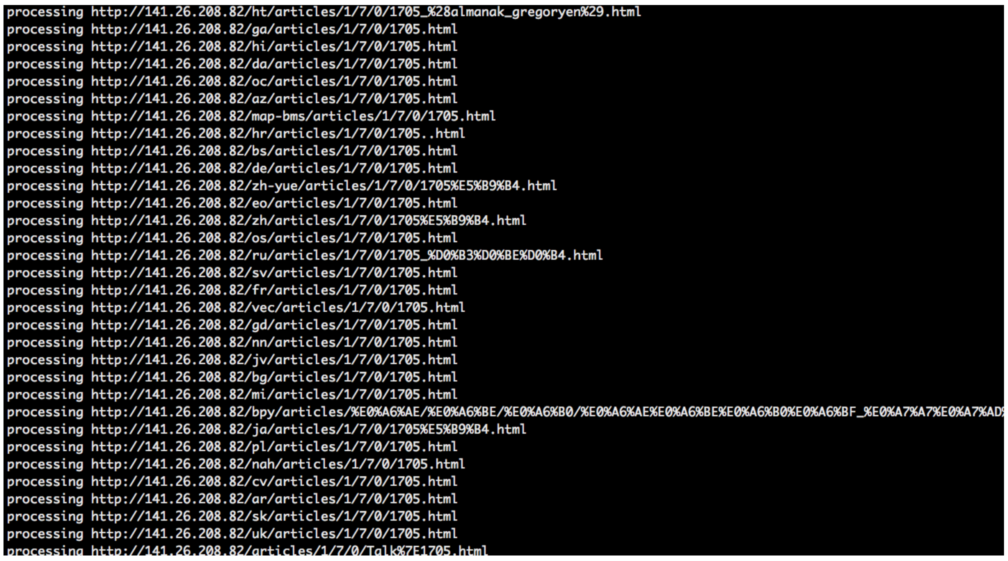
```python
import re
import os
from urllib.parse import urlparse

base_dir   = os.getcwd()
link_regex = re.compile(r'<a[^>]*>([^<]+)</a>')
href_regex = re.compile(r'href="(.*?)"')

def make_dir(directory):
    os.makedirs(directory, exist_ok=True)

def dir_exists(directory):
    return os.path.exists(directory)

def is_internal_link(domain, link):
    return link.startswith(domain) and link.find(".html") >= 0

def save_html(input_url, html):
    path = urlparse(input_url).path
    splitted_path = path.split("/")
    final_path = '/'.join(splitted_path[:-1])
    if not path.startswith('/'):
        final_path = '/' + final_path
    filename = splitted_path[-1]
    full_path = base_dir + final_path
    if not (dir_exists(full_path)):
        make_dir(full_path)
    with open(full_path + '/' + filename, 'w+') as f:
        f.write(html)

def build_url(domain, url):
    # ../../../../articles/g/o/o/Wikipedia%7EGood_articles_eaa4.html
    return domain + '/' + url.replace('../','')

def find_link_tag(html):
    return list(link_regex.finditer(html))

def url_path(link_tag):
    match = href_regex.search(link_tag)
    href = match.group(0)
    return href[href.find('"') + 1: -1]
```

Listing 4 linkqueue.py

```python
from queue import PriorityQueue

class LinkQueue(PriorityQueue):

  def __init__(self):
    PriorityQueue.__init__(self)
    self.counter = 0

  def put(self, item, priority):
    PriorityQueue.put(self, (priority, self.counter, item))
    self.counter += 1

  def get(self, *args, **kwargs):
    _, _, item = PriorityQueue.get(self, *args, **kwargs)
    return item
```

Listing 5 crawl.py

```
import re
import sys
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.parse import urlparse

from crawl_helper import find_link_tag
from crawl_helper import url_path
from crawl_helper import build_url
from crawl_helper import is_internal_link
from crawl_helper import save_html

from linkqueue import LinkQueue


# some global variables to track the information
base_url = None
dead_link_count = 0
internal_link_count = 0
external_link_count = 0
downloaded_links = set()
unique_external_links = set()
unique_internal_links = set()
dead_links = set()
pages = list()

current_iteration = None
links_to_visit = LinkQueue()


def clean_url(url):
    if url.startswith('http://'):
        return url
    return build_url(base_url, url)

def html_path(path):
    splitted_path = path.split("/")
    return '/'.join(splitted_path[:-1])

def download_html(url):
  try:
    html = urlopen(url)
  except HTTPError as e:
    return None
  return html

def process_url(url):
    global internal_link_count, external_link_count, unique_internal_links
    global dead_link_count, dead_links, unique_external_links, links_to_visit
    print('processing {}'.format(url))
    html = download_html(url)
    if html is None:
        dead_link_count += 1
        dead_links.add(url)
        return False
    html  = html.read().decode()
    save_html(url, html)

    links = list(map(lambda x: x.group(0), find_link_tag(html)))
    urls  = list(map(lambda x: url_path(x), links))
    urls  = list(map(lambda url: clean_url(url), urls))
    internal_links = list(filter(lambda url: is_internal_link(base_url, url), url
    internal_link_count += len(internal_links)
    iterable_links = (set(internal_links)).difference(unique_internal_links).diff

    unique_internal_links = unique_internal_links.union(set(internal_links))
    external_links = list(filter(lambda url: not is_internal_link(base_url, url),
    external_link_count += len(external_links)
    unique_external_links = unique_external_links.union(set(external_links))

    for item in iterable_links:
        links_to_visit.put(item, 1)


def main(url):
    global links_to_visit
    links_to_visit.put(url, 1)
    try:
        while not links_to_visit.empty():
            url = links_to_visit.get()
            process_url(url)
    except Exception as e:
        print(e)
    print("Total Dead Links is {}".format(dead_link_count))
    print("Total External Links is {}".format(external_link_count))
    print("Total Internal Links is {}".format(internal_link_count))
    print("Total Internal Unique Links is {}".format(len(unique_internal_links)))
    print("Total External Unique Links is {}".format(len(unique_external_links)))
    print("Total Dead Unique Links is {}".format(len(dead_links)))


if __name__ == "__main__":
  starting_url = 'http://141.26.208.82/articles/g/e/r/Germany.html' # sys.argv[1]
  # http://141.26.208.82/articles/g/e/n/Wikipedia%7EGeneral_disclaimer_3e44.html
  HOST, DOMAIN, PATH, _, PARAMETERS, FRAGMENTS = urlparse(starting_url)
  base_url = HOST + '://' + DOMAIN
  main(starting_url)
```



```
processing http://141.26.208.82/ht/articles/1/7/0/1705_%28almanak_gregoryen%29.html
processing http://141.26.208.82/ga/articles/1/7/0/1705.html
processing http://141.26.208.82/hi/articles/1/7/0/1705.html
processing http://141.26.208.82/da/articles/1/7/0/1705.html
processing http://141.26.208.82/oc/articles/1/7/0/1705.html
processing http://141.26.208.82/az/articles/1/7/0/1705.html
processing http://141.26.208.82/map-bms/articles/1/7/0/1705.html
processing http://141.26.208.82/hr/articles/1/7/0/1705..html
processing http://141.26.208.82/bs/articles/1/7/0/1705.html
processing http://141.26.208.82/de/articles/1/7/0/1705.html
processing http://141.26.208.82/zh-yue/articles/1/7/0/1705%E5%B9%B4.html
processing http://141.26.208.82/eo/articles/1/7/0/1705.html
processing http://141.26.208.82/zh/articles/1/7/0/1705%E5%B9%B4.html
processing http://141.26.208.82/os/articles/1/7/0/1705.html
processing http://141.26.208.82/ru/articles/1/7/0/1705_%D0%B3%D0%BE%D0%B4.html
processing http://141.26.208.82/sv/articles/1/7/0/1705.html
processing http://141.26.208.82/fr/articles/1/7/0/1705.html
processing http://141.26.208.82/vec/articles/1/7/0/1705.html
processing http://141.26.208.82/gd/articles/1/7/0/1705.html
processing http://141.26.208.82/nn/articles/1/7/0/1705.html
processing http://141.26.208.82/jv/articles/1/7/0/1705.html
processing http://141.26.208.82/bg/articles/1/7/0/1705.html
processing http://141.26.208.82/mi/articles/1/7/0/1705.html
processing http://141.26.208.82/bpy/articles/%E0%A6%AE/%E0%A6%BE/%E0%A6%B0/%E0%A6%AE%E0%A6%BE%E0%A6%B0%E0%A6%BF_%E0%A7%A7%E0%A7%AD
processing http://141.26.208.82/ja/articles/1/7/0/1705%E5%B9%B4.html
processing http://141.26.208.82/pl/articles/1/7/0/1705.html
processing http://141.26.208.82/nah/articles/1/7/0/1705.html
processing http://141.26.208.82/cv/articles/1/7/0/1705.html
processing http://141.26.208.82/ar/articles/1/7/0/1705.html
processing http://141.26.208.82/sk/articles/1/7/0/1705.html
processing http://141.26.208.82/uk/articles/1/7/0/1705.html
processing http://141.26.208.82/articles/1/7/0/Talk%7E1705.html
```

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
We only crawl 1000 pages for this question.
Phase I

1. We found total 1000 *webpages*.
2. We encountered 163796 links in the complete process of crawling.
3. Average number of links per page = 163796/1000 = 164
   Median number of links per page = 134.0

Listing 6 median
```
file_content = None
with open('pages.txt', 'r') as f:
    file_content = f.read()
pages = eval(file_content)
import numpy
median = numpy.median(numpy.array(pages))
```

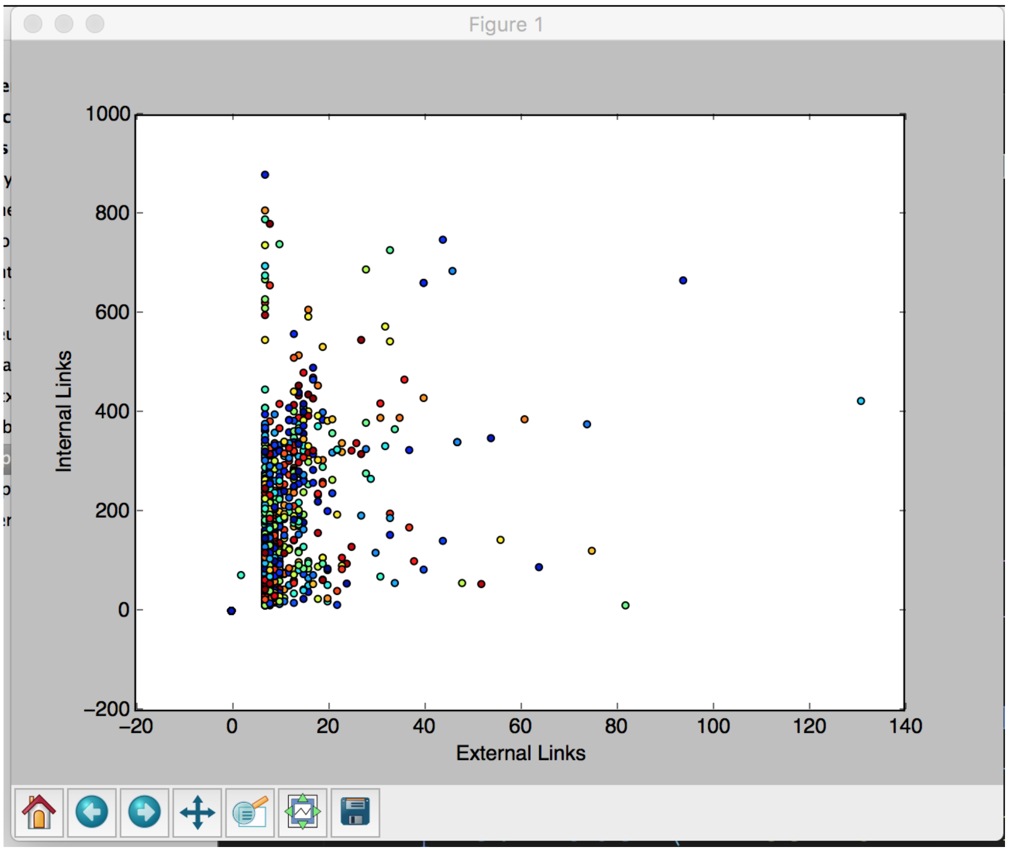
4. *Histogram*

Phase II

1. For every downloaded page, the number of internal links and external links.

Listing 7 Scatter Plot

```
import numpy as np
import matplotlib.pyplot as plt

def draw_scatter_plot():
    links = None
    with open('info.txt', 'r') as f:
        links = f.read()
    links = eval(links)
    internal = [x[1] for x in links]
    external = [x[0] for x in links]
    N = len(internal)
    colors = np.random.rand(N)
    plt.scatter(internal, external, s=15, c=colors, alpha=1)
    plt.xlabel('External Links')
    plt.ylabel('Internal Links')
    plt.show()

def main():
    draw_scatter_plot()

if __name__ == "__main__":
    main()
```



2. *scatter plot*

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`.