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

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).

Answer:

1. server.py

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

message_before = 'Intro Web Science';
message_after = ""

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 = '/home/jp/ass5' # Directory where webpage files are store


 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,
            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

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

    return h

def long_pooling(self, conn):
    global message_after, message_before
    message_after = input("Your Message Here ")
    if not message_after == '' and not message_after == message_before:
       message_before = message_after
    else:
       message_after = message_before

    response_headers = self._gen_headers( 200)
    response_content = response_headers.encode()
```

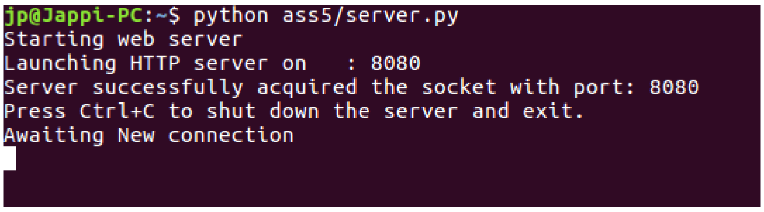
```
        response_content += message_after.encode()
        print(response_content)
        conn.send(response_content)

    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 anyth

                if file_requested == '/server_message':
                   self.long_pooling(conn)
                else:
                    if (file_requested == '/'):  # in case no file is specified
                        file_requested = '/webclient.html' # load index.html by

                    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
                            response_content = file_handler.read() # read file c
                        file_handler.close()

                        response_headers = self._gen_headers( 200)

                    except Exception as e: #in case file was not found, generate
                        print ("Warning, file not found. Serving response code 4
                        response_headers = self._gen_headers( 404)

                        if (request_method == 'GET'):
                           response_content = b"<html><body><p>Error 404: File n

                    server_response =  response_headers.encode() # return header
                    if (request_method == 'GET'):
                        server_response +=  response_content  # return additiona
                    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
```

2. webclient.html

```
<html>
<head>
        <title>Abusing the HTTP protocol - Example</title>
</head>
<body>
        <h1>Display data from the Server without page reload</h1>
        New message from the server : <br>
        <span id="response_message" style="color:blue">
                This will be replaced by messages from the server
        </span>
        <script>
          var response = document.getElementById('response_message')
```

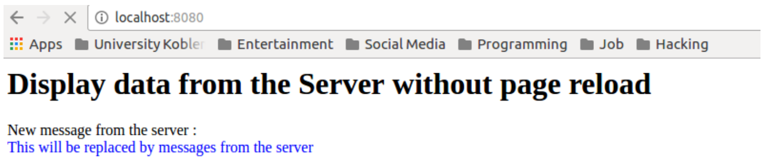
```
            function reqListener () {
              response.innerHTML = this.responseText;
            }
            function createRequest() {
              var oReq = new XMLHttpRequest();
              oReq.addEventListener("load", reqListener);
              oReq.open("GET", "http://localhost:8080/server_message");
              oReq.send();
            }
            setInterval(createRequest, 1000);
          </script>

</body>
</html>
```

3. output:

- 

```
jp@Jappi-PC:~$ python ass5/server.py
Starting web server
Launching HTTP server on    : 8080
Server successfully acquired the socket with port: 8080
Press Ctrl+C to shut down the server and exit.
Awaiting New connection
```

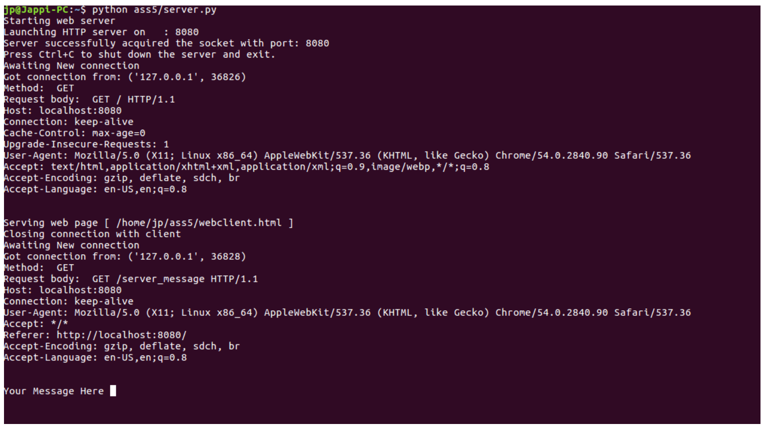
Figure 1: Server create socket

- 

Display data from the Server without page reload

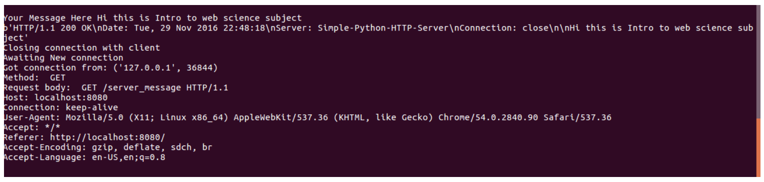
New message from the server :
This will be replaced by messages from the server

Figure 2: Client Connect

- 

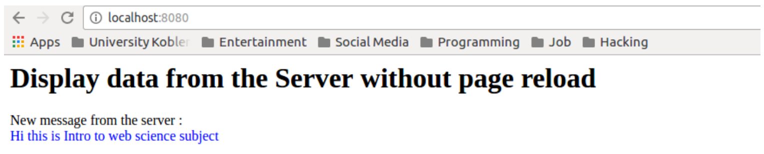
```
jp@Jappi-PC:~$ python ass5/server.py
Starting web server
Launching HTTP server on   : 8080
Server successfully acquired the socket with port: 8080
Press Ctrl+C to shut down the server and exit.
Awaiting New connection
Got connection from: ('127.0.0.1', 36826)
Method:  GET
Request body:  GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.90 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 [ /home/jp/ass5/webclient.html ]
Closing connection with client
Awaiting New connection
Got connection from: ('127.0.0.1', 36828)
Method:  GET
Request body:  GET /server_message HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.90 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 █
```

Figure 3: Server accept request

- 

```
Your Message Here Hi this is Intro to web science subject
b'HTTP/1.1 200 OK\nDate: Tue, 29 Nov 2016 22:48:18\nServer: Simple-Python-HTTP-Server\nConnection: close\n\nHi this is Intro to web science sub
ject'
Closing connection with client
Awaiting New connection
Got connection from: ('127.0.0.1', 36844)
Method:  GET
Request body:  GET /server_message HTTP/1.1
Host: localhost:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.90 Safari/537.36
Accept: */*
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8
```

Figure 4: Command line argument



Display data from the Server without page reload

New message from the server :
Hi this is Intro to web science subject

-

Figure 5: User get Notification without page reload

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.

1. We are using the code from the last assignment to crawle the webpage.
2. we are considering the webpages that returns 200 OK status.

Answer:

1. crawler.py

```
import http_client
import csv
from collections import deque
import sys
import re
#import downloadEverything
import http_client

queue = deque()
traversed_set = set()
external_urls = {}
internals_urls = {}
domain =  "http://141.26.208.82/"

def find_all_links(file_name):
  with open(file_name, 'r') as f:
    html = f.read()
  href_pattern = 'href=[\'"]?([^\'" >]+)'
  return re.findall(href_pattern, html)

def if_relative_url(url):
  parse_url = http_client.parse_url(url)
  if(parse_url.netloc):
    return False
  else:
    return True

def make_absolute_url(relative_url):
  relative_url = http_client.parse_url(relative_url).path
  return "".join([domain,relative_url] )

def add_value_to_queue(value):
  if value not in traversed_set:
    queue.append(value)
    traversed_set.add(value)

def process_queue():
  return queue.popleft()

def crawl(start_url):
  add_value_to_queue(start_url)
  i = 0
  while(len(queue)):
    if(i%25 == 0 ):
       print('crawling\t' + str(i) + ' th page')
    i = i +1

    url_to_be_crawled = process_queue()
    print("\ncarwaling url\n" + url_to_be_crawled)
    output = http_client.call_http_server(url_to_be_crawled, False)
    if output[0] == True:
      urls = find_all_links(output[1])
      for url in urls:
        if(if_relative_url(url)):
          absolute_url = make_absolute_url(url)
          url_list = internals_urls.get(url_to_be_crawled, [])
          add_value_to_queue(absolute_url)
          url_list.append(absolute_url)
          internals_urls[url_to_be_crawled] = url_list
        else:
          url_list = external_urls.get(url_to_be_crawled, [])
          url_list.append(url)
          external_urls[url_to_be_crawled] = url_list
  write = csv.writer(open("internal_urls.csv", "w"))
  for key, val in internals_urls.items():
    write.writerow([key, val])
  write = csv.writer(open("external_urls.csv", "w"))
  for key, val in internals_urls.items():
    write.writerow([key, val])


if __name__ == '__main__':
  start_url = "http://141.26.208.82/articles/g/e/r/Germany.html"
  crawl(start_url)
```

2. httpclient.py

```
import socket
from urllib.parse import urlparse
import sys
import os
__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

```

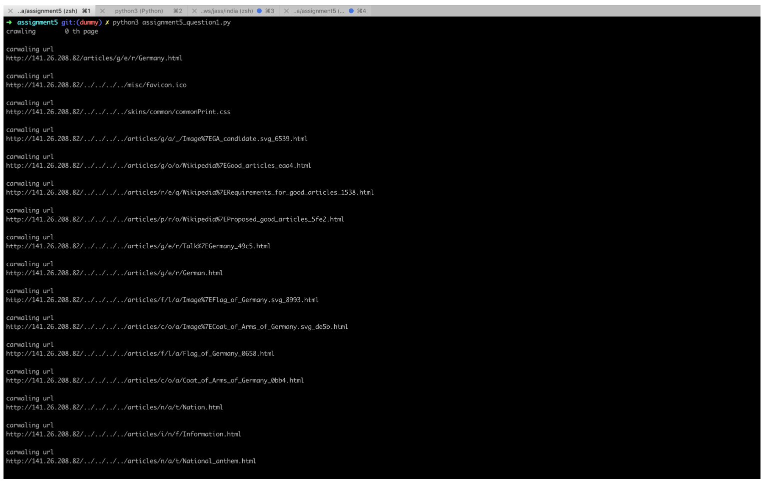
```
# Parses the URL into six components.i.e scheme, network location, path, para
# input:
#   url_name:String - takes the valid url passed
# output:
#   String - parsed Url into 6 components
def parse_url(url_name):
  return urlparse(url_name)

# User request for sending user input URL as HTTP request ending with two emp
# 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 t
def encode_http_request(path, method_type):
  protocol_type = 'HTTP/1.0'
  get_request = ''.join([method_type,' ', path,' ',protocol_type, '\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
#   http_request:String - takes the valid Http Request according to method ty
# output:
#   Bytes: returns the Http request made in bytes to send it further for comm
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
#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 a
  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)

    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
#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')
  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
#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.
#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 fi
#   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:
#   binary_data_array_to_be_written:List - List of Byte to be written on the
#   file_name: String- file name.
#output:
#    None
def save_binary_data(binary_data_array_to_be_written, file_name):
    #binary_data_to_be_written = str.encode(string_data_to_be_written)
    with open(file_name, 'bw') as file:
      for value in binary_data_array_to_be_written:
        file.write(value)



# 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
#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)
  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)
      tcp_socket.close()
      return (True, text_file_name)
    elif(find_content_type(decoded_message) == 'binary'):
      save_binary_data(http_data, file_name)
```

```
        tcp_socket.close()
        return (False, file_name)
    else:
        tcp_socket.close()
        return (False, 404)
if __name__ == "__main__":
    call_http_server(sys.argv[1])
```

3. output:

- 

```
..a/assignment5 (zsh) ⌘1    python3 (Python) ⌘2    ..ws/jass/india (zsh) ⌘3    ..a/assignment5 (... ⌘4
➜  assignment5 git:(dummy) ✗ python3 assignment5_question1.py
crawling        0 th page

carwaling url
http://141.26.208.82/articles/g/e/r/Germany.html

carwaling url
http://141.26.208.82/../../../../misc/favicon.ico

carwaling url
http://141.26.208.82/../../../../skins/common/commonPrint.css

carwaling url
http://141.26.208.82/../../../../articles/g/a/_/Image%7EGA_candidate.svg_6539.html

carwaling url
http://141.26.208.82/../../../../articles/g/o/o/Wikipedia%7EGood_articles_eaa4.html

carwaling url
http://141.26.208.82/../../../../articles/r/e/q/Wikipedia%7ERequirements_for_good_articles_1538.html

carwaling url
http://141.26.208.82/../../../../articles/p/r/o/Wikipedia%7EProposed_good_articles_5fe2.html

carwaling url
http://141.26.208.82/../../../../articles/g/e/r/Talk%7EGermany_49c5.html

carwaling url
http://141.26.208.82/../../../../articles/g/e/r/German.html

carwaling url
http://141.26.208.82/../../../../articles/f/l/a/Image%7EFlag_of_Germany.svg_8993.html

carwaling url
http://141.26.208.82/../../../../articles/c/o/a/Image%7ECoat_of_Arms_of_Germany.svg_de5b.html

carwaling url
http://141.26.208.82/../../../../articles/f/l/a/Flag_of_Germany_0658.html

carwaling url
http://141.26.208.82/../../../../articles/c/o/a/Coat_of_Arms_of_Germany_0bb4.html

carwaling url
http://141.26.208.82/../../../../articles/n/a/t/Nation.html

carwaling url
http://141.26.208.82/../../../../articles/i/n/f/Information.html

carwaling url
http://141.26.208.82/../../../../articles/n/a/t/National_anthem.html
```

Figure 6: crawl output

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.

1. for pages encountered we are considering all pages no duplicate removed but for url we are only considering unique 200 pages.
2. All statistical data is shared sing two csv files.They are pushed so while running they should be in home directory.

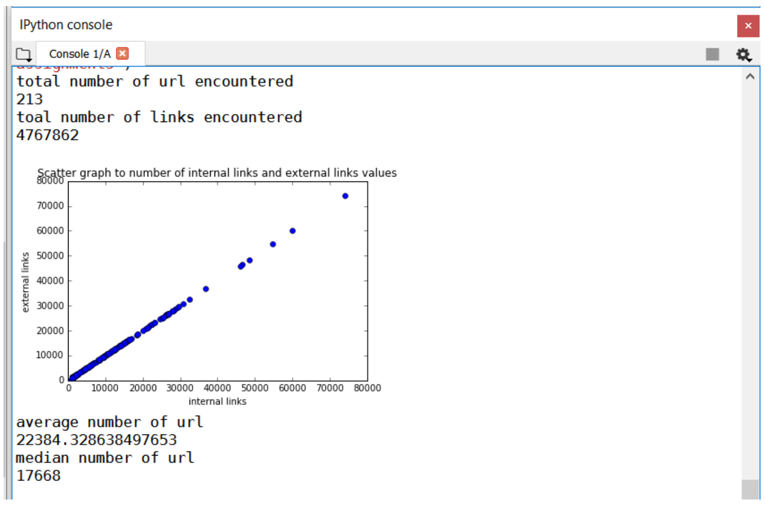
Answer:

1. statistics.py

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as hist
import csv
"""
Created on Tue Nov 29 19:17:08 2016

@author: Amani
"""

internal_urls = {}
```

```
external_urls = {}
def read_csv_as_dictionary(file_name):
    read_dictionary = {}
    for key, val in csv.reader(open(file_name)):
        read_dictionary[key] = val
    return read_dictionary

def counting_dictionary(url_dictionary):
    count_urls = dict()
    for key, value in url_dictionary.items():
        count_urls[key] = len(value)
    return count_urls
# sum of values and number of keys
def sum_link(count_dict):
    sum_of_values = 0
    number_of_keys = 0
    for key, value in count_dict.items():
        number_of_keys += 1
        sum_of_values += value
    return (number_of_keys, sum_of_values)


# page_link gives the urls with number of links
def mean(page_links):
   values = sum_link(page_links)
   mean_links = values[1] / values[0]
   return mean_links


#function for caluculating median
def median(page_link):
      sort_list= sorted(page_link.values())
      length = len(sort_list)
      median_pos = length // 2

      if length % 2 == 0:
          median = (sort_list[median_pos - 1] +
                      sort_list[ ( median_pos) ]) / 2
          return median
      else:
         median = sort_list[median_pos]
         return median
#Number of internal links
def add_data(data_dict1, data_dict2):
    added_dict  = {}
    for key, values in data_dict1.items():
        added_dict[key] = data_dict1[key] + data_dict2[key]
    return added_dict
def scaterring_graph(array_x,array_y):
    plt.plot(array_x,array_y, 'bo')
    plt.title('Scatter graph to number of internal links and external links v
    plt.xlabel('internal links')
    plt.ylabel('external links')
    return plt

if __name__ == '__main__':

#Reading the data from csv files assigned to string variables
      internal_link_file = 'internal_urls.csv'
      external_link_file = 'external_urls.csv'
#Assign the variables to read csv fiels
      internal_urls = read_csv_as_dictionary(internal_link_file)
      external_urls = read_csv_as_dictionary(external_link_file)
      external_urls_count = counting_dictionary(external_urls)
      internal_links_count = counting_dictionary(internal_urls)
      sum_of_urls = add_data(internal_links_count, external_urls_count)
#Printing the number of internal links
      print('total number of url encountered')
      total_urls_encountered = len(sum_of_urls.keys())
      print(sum_link(sum_of_urls)[0])
# Printing the number of externl links

#Printing total number of urls





      print('toal number of links encountered')
      print(sum_link(sum_of_urls)[1])

#Printing the number mean and median of the links per webpage
      internal_link_x = [value for (key, value) in sorted(internal_links_coun
      external_link_y = [value for (key, value) in sorted(external_urls_count
      plt1 = scaterring_graph(internal_link_x, external_link_y)
      plt1.show()
      print('average number of url')
      print(mean(sum_of_urls))
      print('median number of url')
      print(median(sum_of_urls))
#Ploting the scatterring graph


#printing the histogram
      data = list(sum_of_urls.values())
      d = plt.hist(data, bins = 30)
      plt.show()
```

2. output:

- 

```
total number of url encountered
213
toal number of links encountered
4767862
```

IPython console
Console 1/A
Scatter graph to number of internal links and external links values
external links
internal links

```
average number of url
22384.328638497653
median number of url
17668
```

Figure 7: Output for Statistical output

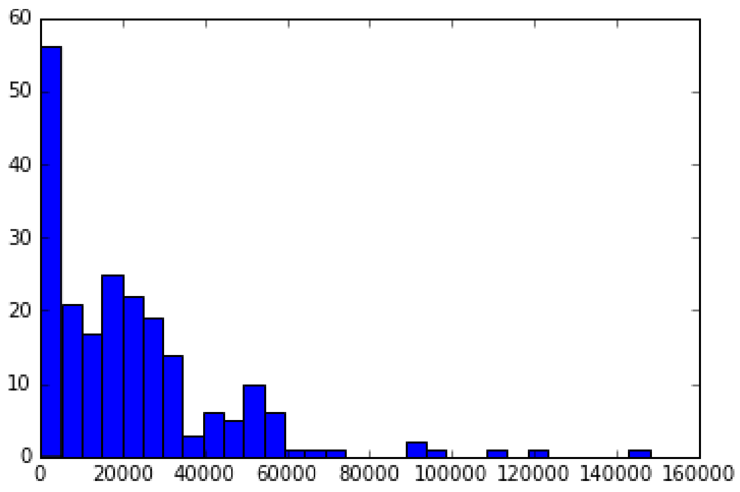
- 

Figure 8: Output for statistical output

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`.