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: Bravo - Shriharsh Ambhore, Kandhasamy Rajasekaran, Daniel Akbari

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 `t` 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).

```
#!/usr/bin/python

#Answer: Long polling is used. Server - runs the 'getting input from user' module
#in a separate thread. Although it is separate, the server blocks other client
# request :-).
#Server respond with index.html - which sends a request to server asking
#get_input_from_user.Server will keep looking for a new response from user
#(runs in a while loop forever and that is why it is blocked).
#Once when it obtains, it sends it back to client and after obtaining the response
#it puts the next request. It just repeats on and on.

import socket  # Networking support
import signal  # Signal support (server shutdown on signal receive)
import time    # Current time
import _thread # to run threading function

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
     self.userInput = ''


 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
```

```
40:          self.port = 8080
41:
42:          try:
43:              print("Launching HTTP server on ", self.host, ":",self.port)
44:              self.socket.bind((self.host, self.port))
45:
46:          except Exception as e:
47:              print("ERROR: Failed to acquire sockets for ports ", user_port, " and
48:              print("Try running the Server in a privileged user mode.")
49:              self.shutdown()
50:              import sys
51:              sys.exit(1)
52:
53:      print ("Server successfully acquired the socket with port:", self.port)
54:      print ("Press Ctrl+C to shut down the server and exit.")
55:      self._wait_for_connections()
56:
57:  def shutdown(self):
58:      """ Shut down the server """
59:      try:
60:          print("Shutting down the server")
61:          s.socket.shutdown(socket.SHUT_RDWR)
62:
63:      except Exception as e:
64:          print("Warning: could not shut down the socket. Maybe it was already clos
65:
66:  def _gen_headers(self,  code):
67:      """ Generates HTTP response Headers. Ommits the first line! """
68:
69:      # determine response code
70:      h = ''
71:      if (code == 200):
72:         h = 'HTTP/1.1 200 OK\n'
73:      elif(code == 404):
74:         h = 'HTTP/1.1 404 Not Found\n'
75:
76:      # write further headers
77:      current_date = time.strftime("%a, %d %b %Y %H:%M:%S", time.localtime())
78:      h += 'Date: ' + current_date +'\n'
79:      h += 'Server: Simple-Python-HTTP-Server\n'
80:      h += 'Connection: close\n\n'  # signal that the conection wil be closed after
81:
82:      return h
83:
84:  def _wait_for_connections(self):
85:      """ Main loop awaiting connections """
86:      while True:
87:          print ("Awaiting New connection")
88:          self.socket.listen(3) # maximum number of queued connections
```

```
89:
90:         conn, addr = self.socket.accept()
91:         # conn - socket to client
92:         # addr - clients address
93:
94:         print("Got connection from:", addr)
95:
96:         data = conn.recv(1024) #receive data from client
97:         string = bytes.decode(data) #decode it to string
98:
99:         #determine request method  (HEAD and GET are supported)
100:        request_method = string.split(' ')[0]
101:        #print ("Method: ", request_method)
102:        #print ("Request body: ", string)
103:
104:        #if string[0:3] == 'GET':
105:        if (request_method == 'GET') | (request_method == 'HEAD'):
106:            #file_requested = string[4:]
107:
108:            # split on space "GET /file.html" -into-> ('GET','file.html',...)
109:            file_requested = string.split(' ')
110:            file_requested = file_requested[1] # get 2nd element
111:
112:            #Check for URL arguments. Disregard them
113:            file_requested = file_requested.split('?')[0]  # disregard anything a
114:
115:            if (file_requested == '/'):  # in case no file is specified by the br
116:                file_requested = '/index.html' # load index.html by default
117:
118:            # if the client request for user data then wait untill the user input
119:            if(file_requested == '/get_user_input'):
120:                response_headers = self._gen_headers( 200)
121:                tmp = self.userInput
122:                while (tmp == self.userInput) :
123:                    time.sleep(1)
124:                response_content = bytes(self.userInput, 'utf-8')
125:            else:
126:                file_requested = self.www_dir + file_requested
127:                print ("Serving web page [",file_requested,"]")
128:
129:                ## Load file content
130:                try:
131:                    file_handler = open(file_requested,'rb')
132:                    if (request_method == 'GET'):  #only read the file when GET
133:                        response_content = file_handler.read() # read file conter
134:                    file_handler.close()
135:
136:                    response_headers = self._gen_headers( 200)
137:
```

```
138:                     except Exception as e: #in case file was not found, generate 404
139:                         print ("Warning, file not found. Serving response code 404\n"
140:                         response_headers = self._gen_headers( 404)
141:
142:                         if (request_method == 'GET'):
143:                            response_content = b"<html><body><p>Error 404: File not fo
144:
145:                 server_response =  response_headers.encode() # return headers for GET
146:                 if (request_method == 'GET'):
147:                     server_response +=  response_content  # return additional conten
148:
149:                 conn.send(server_response)
150:                 print ("Closing connection with client")
151:                 conn.close()
152:
153:             else:
154:                 print("Unknown HTTP request method:", request_method)
155:
156: def graceful_shutdown(sig, dummy):
157:     """ This function shuts down the server. It's triggered
158:     by SIGINT signal """
159:     s.shutdown() #shut down the server
160:     import sys
161:     sys.exit(1)
162:
163: def getInputFromUser(sockObj):
164:     while True:
165:         time.sleep(2)
166:         print('UserInput : ',end='')
167:         sockObj.userInput = input()
168:
169: ###############################################################
170: # shut down on ctrl+c
171: signal.signal(signal.SIGINT, graceful_shutdown)
172:
173: print ("Starting web server")
174: s = Server(80)  # construct server object
175:
176: # create a new thread and run it to get the input from the user
177: _thread.start_new_thread(getInputFromUser, (s,))
178:
179: s.activate_server() # aquire the socket
```

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.

```
1: # -*- coding: utf-8 -*-
2: """
3: Solution for Q2,Q3.1 and Q3.2
4: Created on Sun Nov 27 19:28:21 2016
5:
6: @author: Shriharsh Ambhore
7: @author: Kandhasamy Rajasekaran
8: @author: Daniel Akbari
9: web crawler
10:
11: Assumptions
12: links = number of internal & external links per page
13: webpage= a page which is physically downloaded
14:
15: """
16:
17: import logging
18: import urllib
19: import socket
20: import re
21: import os
22: import collections
```

```
23: from urllib import parse
24: from functions import *
25:
26:
27: logging.basicConfig(filename="info.log",level=logging.INFO)
28:
29:
30: counter=0
31: # total number of wepages encountered till the end
32: totalWebPageList=[]
33:
34: #dict data structure containing  parentWebPage and ListofLinks in parentWebPage
35: linksPerWebPage={}
36:
37: #this list containes the webpages that needs to be crawled
38: #toCrawlLinks=collections.deque()
39: toCrawlLinks=[]
40: #dict  where key= Parent Webpage, value = list[2]  list[0]=internalLinks,list[1]=
41: intExtWebPageCounter={}
42:
43: invalidHttpResponseCounter=[]
44: webpageCounter=0
45:
46:
47:
48:
49: def getResource(socClient, urlInput):
50:
51:     try:
52:         urlObj = urllib.parse.urlparse(urlInput)
53:         #print(urlObj)
54:         logging.info(urlObj)
55:     except:
56:         print('Invalid URL',urlInput)
57:         return None
58:     try:
59:
60:         urlScheme = urlObj[0] #http
61:         urlDomain = urlObj[1] #full domain name
62:         urlPath = urlObj[2]
63:         #resourceName = (urlPath[urlPath.rfind("/")+1:])
64:
65:         # Form the http GET request. Two \r\n at the end is very important
66:         httpRequest = 'GET ' + urlPath + ' ' + urlScheme+'/1.0\r\n'
67:         httpRequest += 'Host: '+urlDomain+ '\r\n\r\n'
68:
69:         socClient.send(httpRequest.encode('utf-8'))
70:         temp = socClient.recv(4096)
71:         data = bytearray()
```

```
72:         while (temp != b''):
73:             #print(temp)
74:             # Tried a lot to append in bulk
75:             # RIght now appending char by char
76:             for char in temp :
77:                 data.append(char)
78:             temp = socClient.recv(4096)
79:
80:         return [urlPath, data]
81:     except socket.error as msg:
82:         print("Error in creating a socket connection",msg)
83:
84:
85:
86: def extractHeaderAndResource(data):
87:     #The header and resource will be separated  by two \r\n's
88:     try:
89:         splitData = data.split(b'\r\n\r\n') #for actual testing
90:         #splitData = data.split(b'\n\n',1) # for local instance
91:         #print(len(splitData))
92:     except:
93:         print("exception")
94:         splitData = [None, None]
95:     return splitData
96:
97: def checkForRequest200(header):
98:     splittedHeader = header.split(b' ')
99:     if(splittedHeader):
100:         return splittedHeader[1] == b'200'
101:     return False
102:
103: def saveResource(data,iname):
104:     global webpageCounter
105:     try:
106:         #print("iname==",iname)
107:         directoryPath=iname[:iname.rfind("\\")]
108:         #print("Directory Path====",directoryPath)
109:         if not os.path.exists(directoryPath):
110:             #print("creating directory")
111:             os.makedirs(directoryPath,exist_ok=True)
112:
113:         fopen = open(iname,'wb')
114:         fopen.write(data)
115:         fopen.flush()
116:         fopen.close()
117:         webpageCounter=webpageCounter+1
118:     except IOError as io:
119:          pass
120:
```

```
121:
122: def downloadResource(socClient,receivedurl):
123: #    global invalidHttpResponseCounter
124:     #print("urlInput==========",receivedurl)
125:     [name, data] = getResource(socClient,receivedurl)
126:     name=parse.unquote(name)
127:     #print("name",name)
128:
129:
130:     localPath=os.getcwd()+name
131:     localPath=os.path.normpath(localPath)
132:
133:
134:     if(name == ''):
135:         name = 'index.html'
136:     if (data):
137:         try:
138:             #print("data is available!!!",data)
139:             header,resource = extractHeaderAndResource(data)
140:             #print("Resource====",len(resource))
141:             if (header):
142:                 #print("header is available")
143:                 if(checkForRequest200(header)) :
144:                     #print("sending this location for file===",localPath)
145:                     saveResource(resource,localPath)
146:                     #print('Resource is downloaded successfully !!!')
147: #                 else:
148: #                     #print('Invalid Http response !!!!')
149: #                     invalidHttpResponseCounter.append(name)
150:
151:             else:
152:                 print('Header and Resource extraction is invalid for url',urlInput
153:         except:
154:             print('Error in downloading the resource !!!')
155:     return localPath
156:
157:
158:
159: def extractLinkInformationUrl(path):
160:     #print("fileLocation",fileLocation)
161:
162:     try:
163:         hrefPattern=re.compile(r'<a [^>]*href="([^"]+)')
164:
165:         #print("file path received==",path)
166:         if os.path.isfile(path):
167:             #print("yes it is a file!!!")
168:             with  open(path,"r",encoding='utf8') as file:
169:                 con=file.read()
```

```
170:                 linkList=(hrefPattern.findall(con))
171:                 #filers out other domain links e.g wikitionary
172:                 internalLinks=list(filter(lambda x: x.find('articles')!=-1 ,linkL
173:                 externalLinks=list(filter(lambda y: y.find('http')!=-1 or y.find(
174:                 return [internalLinks,externalLinks]
175:
176:         else:
177:             #print("invalid file path")
178:             return[None,None]
179:             pass
180:     except IOError as e:
181:         print("Error in file handling",e)
182:
183:
184: def createFullUrl(orgUrl,loc):
185:     url = urllib.parse.urlparse(orgUrl)
186:     #print(url)
187:     path = url.path
188:     hostname = "http://"+url[1]
189:     if path == "":
190:         path = "/"
191:
192:     # absolut path
193:     if loc[0]=="/":
194:         url = hostname + loc
195:     #fully qualified domain name
196:     elif len(loc) > 7 and loc[0:7]=="http://":
197:         url = loc
198:     # relative path
199:     else:
200:         parentfolders = 0
201:         while len(loc)>3 and loc[0:3]=="../":
202:             loc = loc[3:]
203:             parentfolders= parentfolders + 1
204:             url = hostname + "/".join(path.split("/")[0:-(1+parentfolders)])+"/"+l
205:     return url
206:
207:
208:     ## starting point of the prog
209:
210:
211:
212: try :
213:
214:     socClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
215:     urlInput='http://141.26.208.82/articles/g/e/r/Germany.html'
216:     #urlInput = 'http://localhost/articles/g/e/r/Germany.html'
217:     urlInputObj = urllib.parse.urlparse(urlInput)
218:     socClient.connect((urlInputObj[1], 80))
```

```
219:     #print("Sending this url====>>",urlInput)
220:     url=urlInput
221:     file=downloadResource(socClient,url)
222:     inLinks,outLinks=extractLinkInformationUrl(file)
223:     toCrawlLinks=list(set(toCrawlLinks+inLinks))
224:
225:     #print("toCrawlList====",toCrawlLinks)
226:     counter=len(inLinks)
227:     linksPerWebPage[file]=len(inLinks)
228:     #print(linksPerWebPage)
229:     inoutList=[None]*2
230:     inoutList[0]=len(inLinks)
231:     #toCrawlLinks=toCrawlLinks.append(inLinks)
232:     inoutList[1]=len(outLinks)
233:     intExtWebPageCounter[file]=inoutList
234:     inoutList= None
235:     #print("Internal links and External Links counter per web page",intExtWebPage
236:     crawledLinks=collections.deque()
237:
238:
239:     toCrawlLinks=list(set(toCrawlLinks))
240:     #print("length=========",len(toCrawlLinks))
241:
242:
243:
244:     intj=0
245:     while len(toCrawlLinks)>0:
246: #         if intj>150:
247: #             print("breaking now!!!")
248: #             break
249:
250:         #print("Length of links==",len(toCrawlLinks))
251:         i=toCrawlLinks.pop(0)
252:         #logging.info("popped",i)
253:         try:
254:
255:             tempClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
256:             completeUrl=createFullUrl(url,i)
257:             tempClient.connect((urlInputObj[1], 80))
258:             tempFile=downloadResource(tempClient,completeUrl)
259:             if (tempFile):
260:                 tempInLinks,tempOutLinks=extractLinkInformationUrl(tempFile)
261:                 if (tempInLinks or tempOutLinks):
262:
263:                     #print("tempInLinks===",len(list(set(tempInLinks))))
264:                     #add the in links to the toCrawlLinks   list
265:                     toCrawlLinks=toCrawlLinks+list(set(tempInLinks))
266:
267:                     tempInOutList=[None]*2
```

```
268:                 #         # keep an record of links found per page
269:                         linksPerWebPage[tempFile]=(len(tempInLinks)+len(tempOutLinks))
270:                 #         # keep an record of number of internalLinks and externalLinks
271:                         tempInOutList[0]=len(tempInLinks)
272:                         tempInOutList[1]=len(tempOutLinks)
273:                         intExtWebPageCounter[tempFile]=tempInOutList
274:                 #         print("intExtWebPageCounter",intExtWebPageCounter)
275:                         tempInOutList= None
276:                 #         print("currently popped=====",i)
277:                         crawledLinks.append(i)
278:                        # toCrawlLinks.remove(i)
279:                         counter=counter+(len(tempInLinks)+len(tempOutLinks))
280:                     tempClient.close()
281:             except socket.error as e:
282:                 pass
283:             intj=intj+1
284:             if intj%100==0:
285:                 print("Counter:::",intj)
286:                 #print("Invalid Response Counter:::",len(invalidHttpResponseCounter))
287:                 print("toCrawlLinks length",len(toCrawlLinks))
288:
289:
290:
291:     print("-----****printing the stats****-----")
292:
293:     #print("length=========",len(toCrawlLinks))
294:     #logging.info(len(toCrawlLinks))
295:     #print("Invalid links==",invalidHttpResponseCounter)
296:     print("Total number of Links found===",counter)
297:     print("**************************")
298:     #logging.info("Total number of Links ===",counter)
299:     print("Total number of WebPages ===",webpageCounter)
300:     print("**************************")
301:     #logging.info("Total number of WebPages found===",len(crawledLinks))
302:     print("Internal and External Links per Webpage",intExtWebPageCounter)
303:     #logging.info("Internal and External Links per Webpage",intExtWebPageCounter)
304:     #print("Crawled Links====",crawledLinks)
305:     print("**************************")
306:     print("Links per web Page====",linksPerWebPage)
307:     #logging.info("Links per web Page====",linksPerWebPage)
308:
309:     print ('Average is:', Average(linksPerWebPage))
310:     print ('Median is:',Median(linksPerWebPage))
311:     print (Histogram(linksPerWebPage))
312:     print(Plot(intExtWebPageCounter))
313:
314:
315:
316:
```

```
317:     socClient.close()
318: except socket.error as msg:
319:     print('Error in socket connection',msg)
320: finally:
321:     socClient = None
322:     logging.shutdown()
323:
324:
325:
326: Supporting files
327:
328: # -*- coding: utf-8 -*-
329: """
330: Created on Mon Nov 28 08:14:25 2016
331:
332: @author: Daniel Akbari
333: @author: Shriharsh Ambhore
334: @author: Kandhasamy Rajasekaran
335: """
336:
337:
338: import statistics
339: import matplotlib.pyplot as plt
340: #import array
341:
342: #list=input('dict with key as web page and number of links per webpage as value')
343: def Average(linksPerWebPage):
344:
345:     valueList=list(linksPerWebPage.values())
346:     avg=(sum(valueList)/(len(valueList)))
347:     return avg
348:
349: def Median(linksPerWebPage):
350:     valueList=list(linksPerWebPage.values())
351:     return statistics.median(valueList)
352:
353: def Histogram(linksPerWebPage):
354:     plt.hist(list(linksPerWebPage.values()),bins=5)
355:     plt.title('Histogram')
356:     plt.xlim(0,150)
357:     plt.xlabel("Bins")
358:     plt.ylabel("Frequencey")
359:     plt.show()
360:
361: def Plot(intExtWebPageCounter):
362:     # Create a figure of size 8x6 inches, 80 dots per inch
363:     plt.figure(figsize=(8, 6), dpi=80)
364:     # Create a new subplot from a grid of 1x1
365:     plt.subplot(1, 1, 1)
```

```
366:
367:     data = intExtWebPageCounter
368:
369:     for coord in data.items():
370:         data["x"].append(coord[0])
371:         data["y"].append(coord[1])
372:
373:     plt.title('Internal & External Links')
374:     plt.xlabel('Internal Links')
375:     plt.ylabel('External Links')
376:     plt.grid(True)
377:
378:     plt.scatter(data["x"],data["y"],color="red")
379:     #plt.scatter(list,y, color="blue", label="External")
380:     plt.legend()
381:
382:     plt.show()
383: #
384: #list=[1,2,3,5,11,75]
385: #x=[2,6,5,9,15,17]
386: #y=[4,1,5,9,18,7]
387: #print ('Average is:', Average(list))
388: #print ('Median is:',Median(list))
389: ##print (Plot(x,y))
390: #print (Histogram(list))
```

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`.