Practical Course "Web Technologies"

Exercises 2

A socket can operate in two modes:

- blocking

  By default, TCP sockets are in "blocking" mode: in case of a `recv`, control flow isn't returned until at least one byte of data is read from the remote site. The process of waiting for data to appear is referred to as "blocking". The same is true for `send`.

- non-blocking

  When placed in non-blocking mode, the process does not wait for an operation to complete. This is an invaluable tool if there is the need to switch between many different connected sockets, and want to ensure that none of them cause the program to "lock up."

  If calling `recv` in non-blocking mode, it will return any data that the system has in its read buffer for that socket. But, it won't wait for that data. If the read buffer is empty, the system will return from `recv` immediately and will throw an exception `socket.timeout`.

The operation `socket.settimeout(value)` sets a timeout on blocking socket operations. The value argument can be a nonnegative floating-point number expressing seconds, or None. If a non-zero value is given, subsequent socket operations will raise a timeout exception if the timeout period value has elapsed before the operation has completed. If zero is given, the socket is put in non-blocking mode. If None is given, the socket is put in blocking mode.

Exercise 2.1.

Implement a simple HTTP-client.

a. formulate a class `MyClient` in a file `client.py`:

   - attribute `commsock` – the communication socket
   - constructor
     prepare the socket for the communication to "httpbin.org" via port 80
   - method `doRequest`
     prepare a request

     1. the method of the request and the header (as key-value-pairs) shall be passed via parameters
     2. in the body create the request (SP means space, CRLF means carriage return + line feed, i.e. "\r\n")

        ```
        A Request-line

        Zero or more header (General|Request|Entity) fields followed by CRLF

        An empty line (i.e., a line with nothing preceding the CRLF)
        indicating the end of the header fields
        ```

        where the request is built according to

        ```
        Request-Line = Method SP Request-URI SP HTTP-Version CRLF
        ```

     3. then send the request via `commsock`
     4. wait for a response
     5. if a response has been received return the response, otherwise return None

b. formulate in a file main.py:

- a function `analyse` in which the response (without the entity) passed as parameter is analysed

```
A Status-line

Zero or more header (General|Response|Entity) fields followed by CRLF

An empty line (i.e., a line with nothing preceding the CRLF)
indicating the end of the header fields

Optionally a message-body
```

  fill a dictionary with the header elements, then print out

  1. the status code
  2. the header fields

- the creation of an object of type `MyClient`
- sending requests

  1. method HEAD, no header fields
  2. method HEAD with header field Host
  3. method HEAD with header fields Accept-Language, Accept-Encoding, User-Agent (and corresponding values)

  each response has to be analysed.

Exercise 2.2.

Requests is an elegant and simple HTTP library for Python. It allows to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your POST data. The following example shows how to send a GET-request with some header-values

```python
import requests

headers = {'user-agent': 'my-agent/1.0.1'}
response = requests.get('http://httpbin.org/', headers=headers)
print(response.request.headers)
```

Reformulate Exercise 2.1 using the Requests-library.

Timeouts:

By default, Requests uses blocking operations. To set a request's timeout, the timeout parameter of the request can be used. The timeout parameter can be an integer or float representing the number of seconds to wait on a response before timing out. If the request times out, then the function will raise a `requests.exception.Timeout` exception.