# Lab: HTTP Protocol

1. **URL Decode**

You will receive an encoded URL. Decode the URL and print it on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| http://www.google.bg/search?q=C%23 | http://www.google.bg/search?q=C# |
| https://mysite.com/show?n%40m3= p3%24h0 | https://mysite/show?n@m3= p3$h0 |
| http://url-decoder.com/i%23de%25?id=23 | http://url-decoder.com/i#de%?id=23 |

### Hints

Import **parse** from **urllib** to decode the provided string

1. **Validate URL**

You will receive encoded URL. Decode the URL and validate it. If the URL is valid, print on the console the parts of the URL in the format:

"Protocol: {protocol}"

"Host: {host}"

"Port: {port}"

"Path: {path}"

"Query: {query string}" (if any)

"Fragment: {fragment}" (if any)

If the URL is invalid, print "Invalid URL".

A valid URL has the following parts:

|  |  |
| --- | --- |
| Protocol | **Required** |
| Host | **Required** |
| Port | **Required** (default value for http - **80**, for https - **443**) |
| Path | **Required** (default value: /) |
| Query Strings | Optional (multiple query strings are separated by &) |
| Fragment | Optional |

**Valid URLs**: http://mysite.com:80/demo/index.aspx, https://my-site.bg, https://mysite.bg/demo/search?id=22o#go.

**Invalid URLs**: https://mysite:80/demo/index.aspx, somesite.com:80/search?, https/mysite.bg?id=2.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| http://softuni.bg/ | Protocol: http  Host: softuni.bg  Port: 80  Path: / |
| https://softuni.bg:447/search?Query=pesho&Users=true#go | Protocol: https  Host: softuni.bg  Port: 447  Path: /search  Query: Query=pesho&Users=true  Fragment: go |
| http://google:443/ | Invalid URL |

### Hints

URL value can be encoded, so it is a good idea to use **parse** from **urllib** to decode the provided string to decode it.

## Request Parser

Write a console application that receives HTTP request and prints the HTTP response.

You will receive several lines with valid paths. The last part of the path will be the allowed method. Keep reading paths until you receive "END".

The input comes in the format "{path}/{method}".

After that you will receive a HTTP request. You will have to parse the request and return the corresponding response.

If the path of the request cannot be found in the received paths or the request method is not allowed for the path, the result will be "404 Not Found".

In any other cases the result will be "200 OK".

Write the result on the console in the following format:

**"HTTP/1.1 {status code}"**

**"Content-Length: {length of status text}"**

**"Content-Type: text/plain"**

***"\n"***

"{status text}"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| /register/get  /register/post  END  GET /register HTTP/1.1 | HTTP/1.1 200 OK  Content-Length: 2  Content-Type: text/plain  OK |
| /login/get  /register/get  END  POST /register HTTP/1.1 | HTTP/1.1 404 Not Found  Content-Length: 9  Content-Type: text/plain  Not Found |
| **/index/get**  **/index/post**  **END**  **POST /login HTTP/1.1** | **HTTP/1.1 404 Not Found**  **Content-Length: 9**  **Content-Type: text/plain**  **Not Found** |