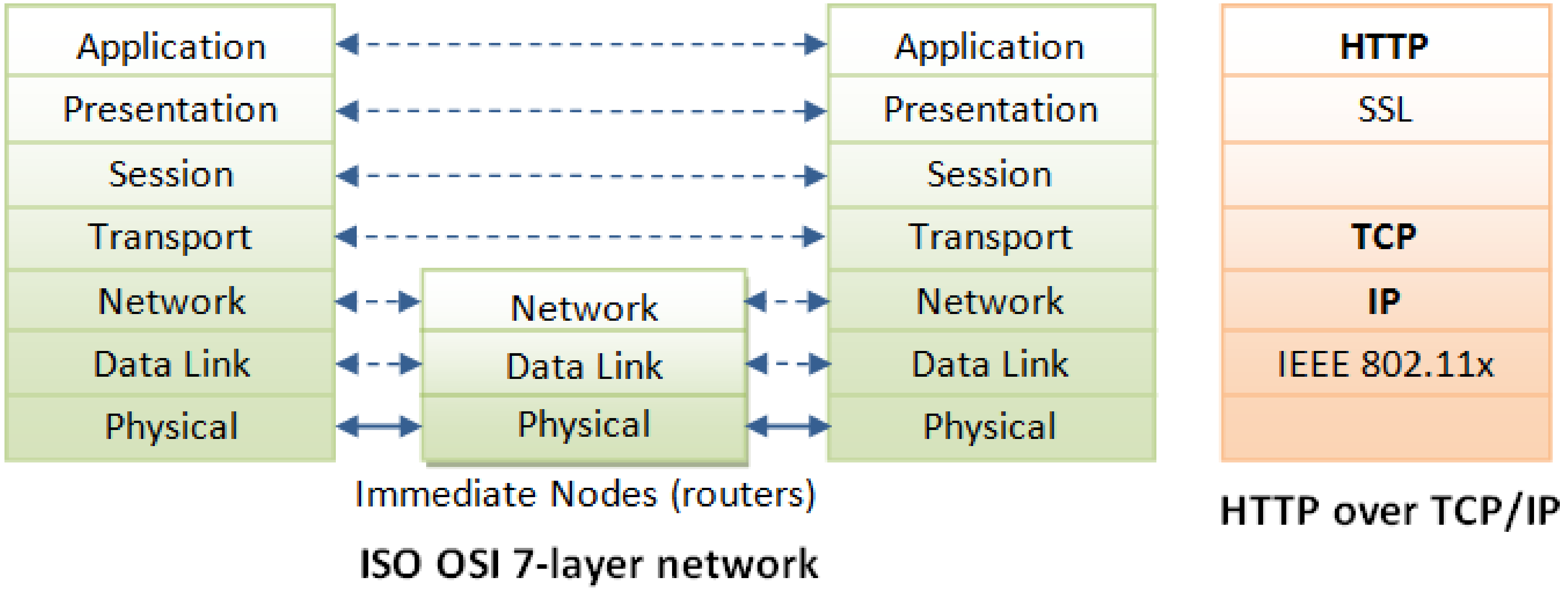
Python Object-Oriented Program with Libraries

Unit 3: Web Programming

CHAPTER 3: HTTP PROTOCOL AND URL

DR. ERIC CHOU

IEEE SENIOR MEMBER


Application
Presentation
Session
Transport
Network
Data Link
Physical
Network
Data Link
Physical
Application
Presentation
Session
Transport
Network
Data Link
Physical
HTTP
SSL
TCP
IP
IEEE 802.11x
Immediate Nodes (routers)
ISO OSI 7-layer network
HTTP over TCP/IP

HTTP

LECTURE 1

HTTP

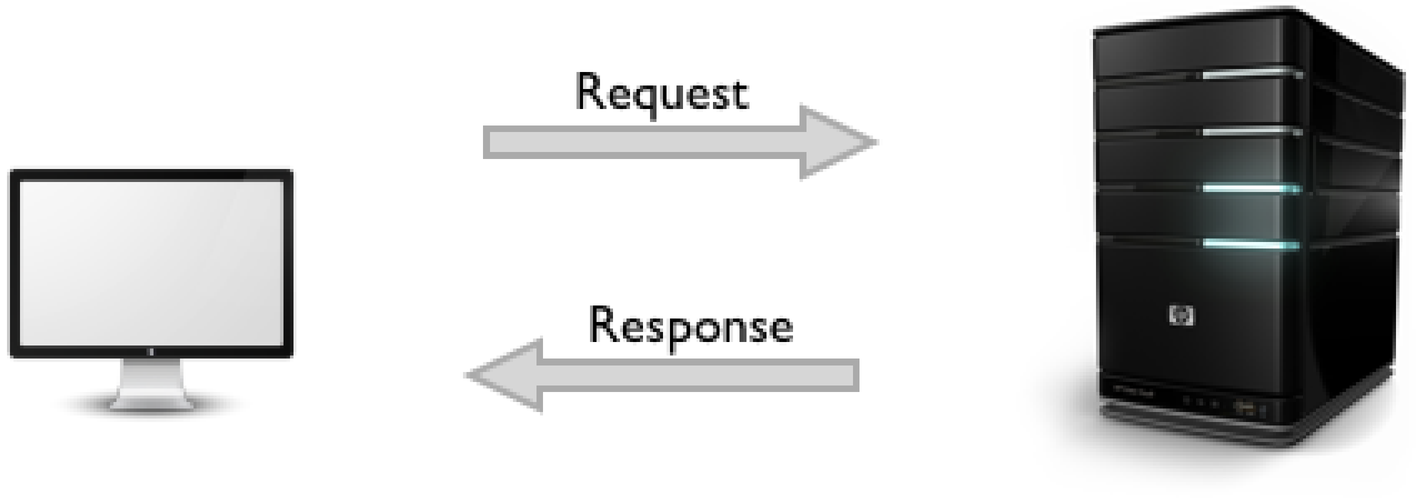
- HTTP stands for Hypertext Transfer Protocol.
- It's a stateless, application-layer protocol for communicating between distributed systems.
- It is the foundation of the modern web.

As a web developer, we all must have a strong understanding of this protocol.

HTTP Basics

- HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations.
- To make this possible, it assumes very little about a particular system, and does not keep state between different message exchanges.
- This makes HTTP a stateless protocol. The communication usually takes place over TCP/IP, but any reliable transport can be used. The default port for TCP/IP is 80, but other ports can also be used.

HTTP Basics



Client-Side

Server-Side

HTTP Basics

- Communication between a host and a client occurs, via a request/response pair.
- The client (program/browser) initiates an HTTP request message, which is serviced through a HTTP response message in return.
- The current version of the protocol is HTTP/1.1, which adds a few extra features to the previous 1.0 version.
- The most important of these, in my opinion, includes persistent connections, chunked transfer-coding and fine-grained caching headers.

HTTP

- **HTTP** (HyperText Transfer Protocol) is the protocol with which most web traffic on the Internet is transacted.
- Its latest version is **HTTP/2.0**, though it's still in the early stages of worldwide adoption; we'll stick with the more broadly-used (and easily-understood) **HTTP/1.1** for now.

HTTP Request-Response Protocol

- HTTP is a request-response protocol, which means that its conversations go something like this:
  - Client initiates connection to server
  - Server accepts connection
  - Client makes a request
  - Server sends a response

Single Request/Response Pair

After that single request and response, both sides close the connection.

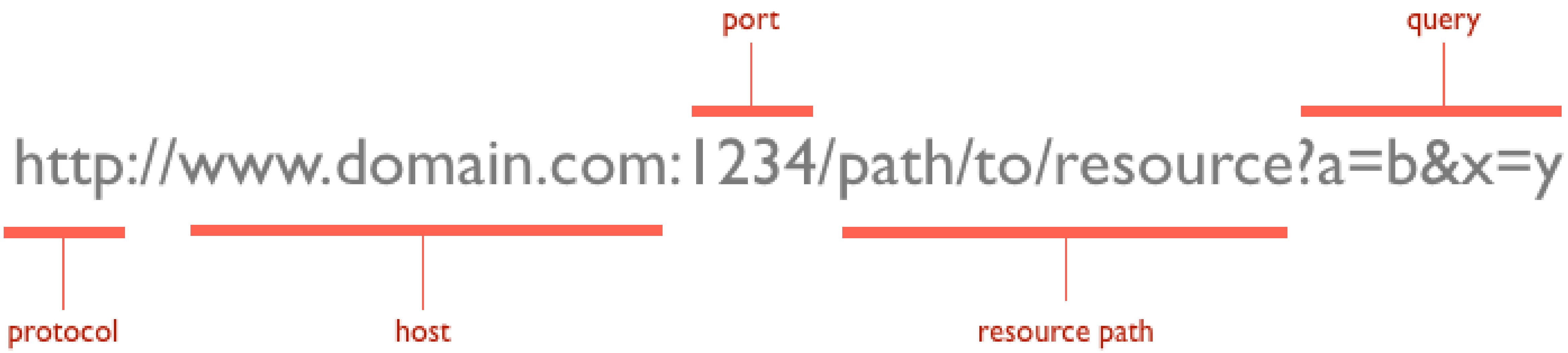
Note:

1. There are performance optimizations available that let a client specify that the connection should be kept open if, for example, the client knows that it needs not just a web page's text but also several images from the same server.
2. For our purposes, we'll stick with a single request and response per connection.

Uniform Resource Locator (URL)

LECTURE 2

Uniform Resource Locator

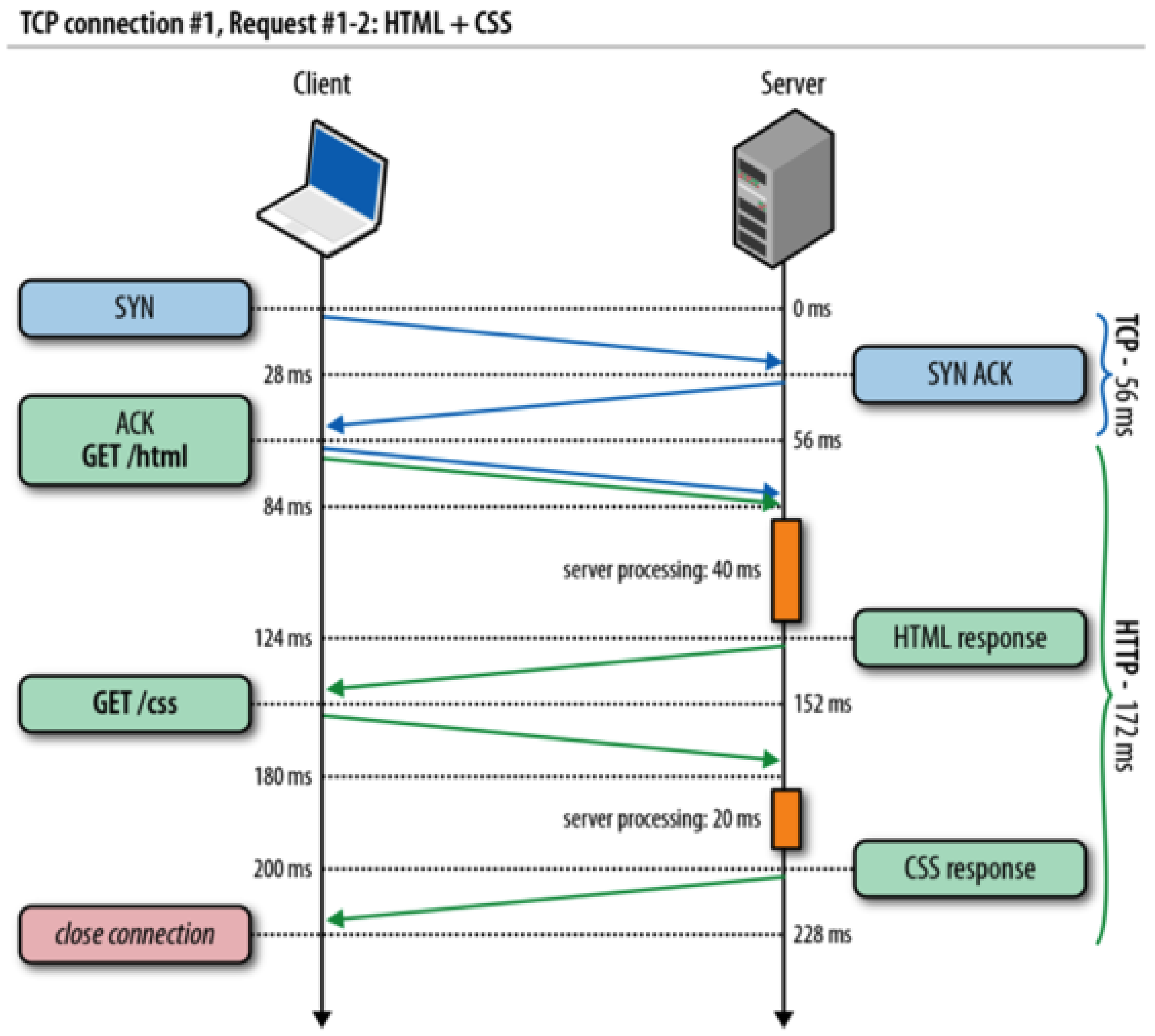


URL

- At the heart of web communications is the request message, which are sent via Uniform Resource Locators (URLs).
- The protocol is typically **http**, but it can also be **https** for secure communications.
- The default port is **80**, but one can be set explicitly, as illustrated in the above image.
- The resource path is the **local path** to the resource on the server.

HTTP Sessions

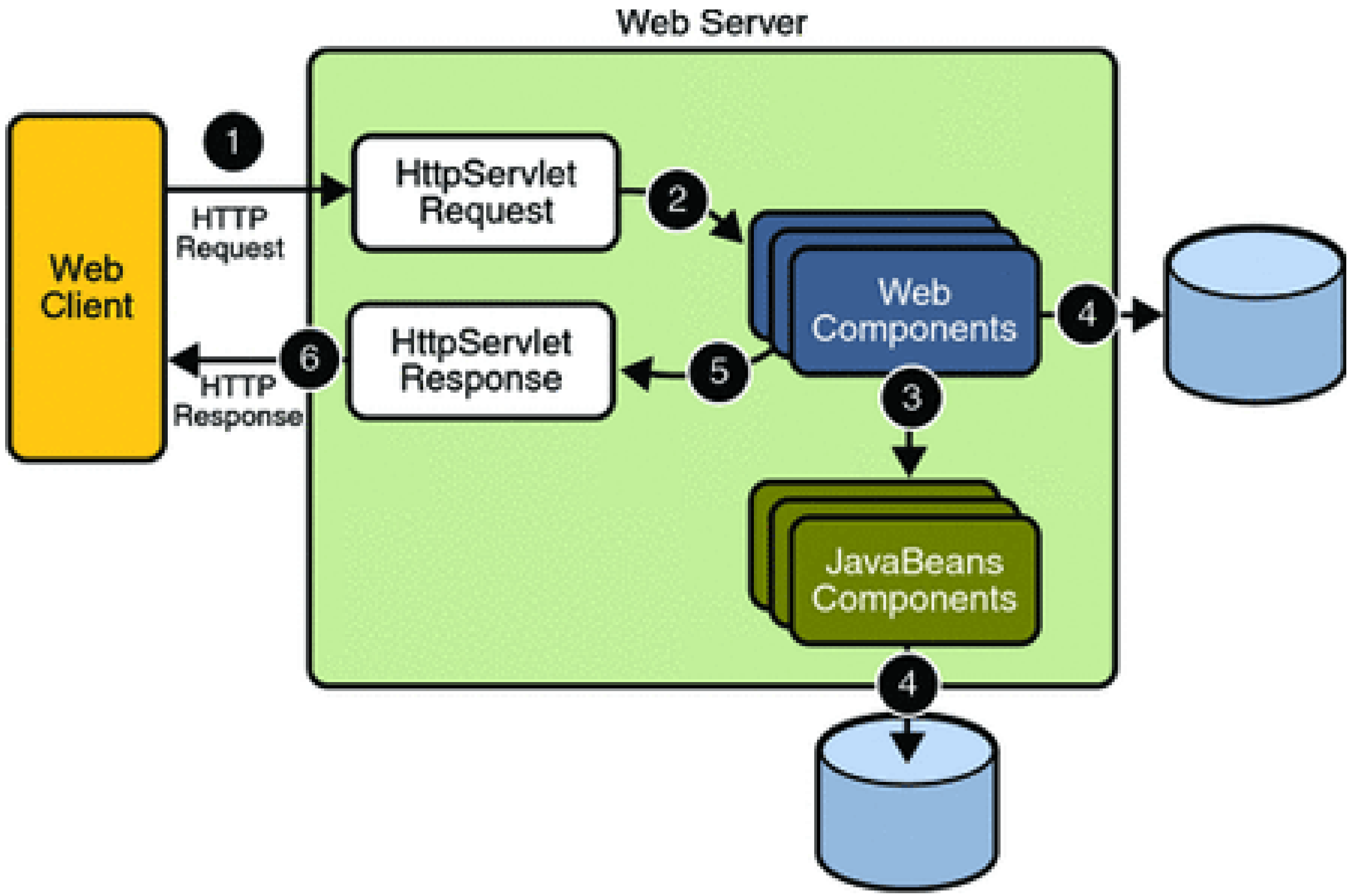
LECTURE 3



HTTP Sessions

WHEN YOU CLICK THE ADDRESS BAR FROM BROWSER, A HTTP SESSION IS CREATED.

Note: TCP for the sockets, HTTP for the contents.


Web Server
Web Client
1
HTTP Request
HttpServlet Request
2
Web Components
4
3
JavaBeans Components
4
5
HttpServlet Response
6
HTTP Response

HTTP Verbs

- URLs reveal the identity of the particular host with which we want to communicate, but the **action** that should be performed on the host is specified via **HTTP verbs**.
- Of course, there are several actions that a client would like the host to perform.
- HTTP has formalized on a few that capture the essentials that are universally applicable for all kinds of applications.

HTTP/2 Sample Object

Properties are backwards compatible to HTTP/1.1, supports:

- Same Methods (GET, POST, HEAD, etc)
- Same Request URI
- Same Headers (Host, etc)
- Same Encodings (mime/type)

HTTP Response
Cookie
ETag
Location
HTTP referer
DNT
X-Forwarded-For
Custom Fields
Body
OPTIONS
GET
PUT
HEAD
POST
PUT
DELETE
TRACE
CONNECT
PATCH

HTTP Verbs

These request verbs are:

- **GET**: fetch an existing resource. The **URL** contains all the necessary information the server needs to locate and return the resource.
- **POST**: create a new resource. **POST** requests usually carry a payload that specifies the data for the new resource.
- **PUT**: update an existing resource. The payload may contain the updated data for the resource.
- **DELETE**: delete an existing resource.

The above four verbs are the most popular, and most tools and frameworks explicitly expose these request verbs. **PUT** and **DELETE** are sometimes considered specialized versions of the **POST** verb, and they may be packaged as **POST** requests with the payload containing the exact action: create, update or delete.

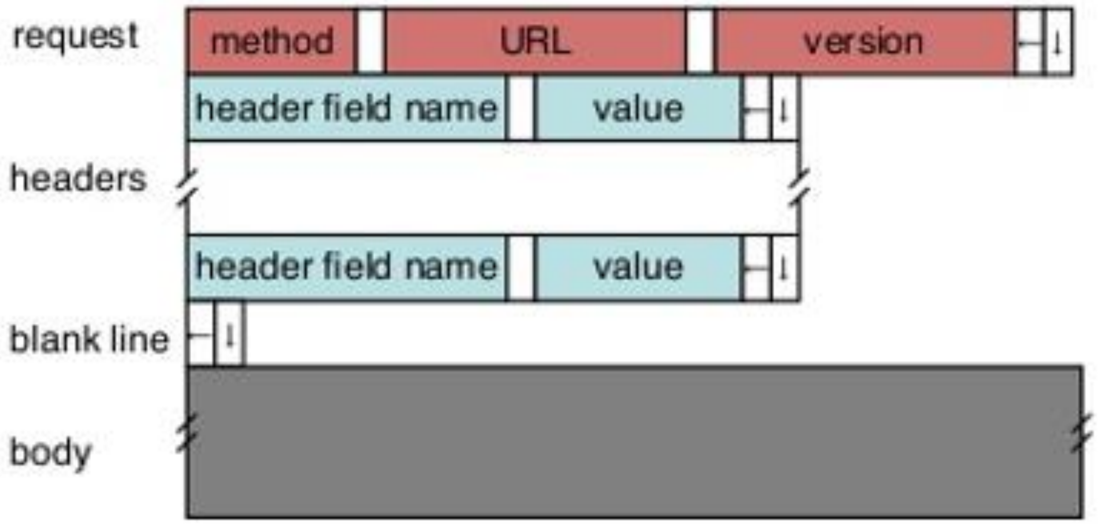
HTTP Verbs

There are some lesser used verbs that **HTTP** also supports:

- **HEAD**: this is similar to **GET**, but without the message body. It's used to retrieve the server headers for a particular resource, generally to check if the resource has changed, via timestamps.
- **TRACE**: used to retrieve the hops that a request takes to round trip from the server. Each intermediate proxy or gateway would inject its IP or DNS name into the Via header field. This can be used for diagnostic purposes.
- **OPTIONS**: used to retrieve the server capabilities. On the client-side, it can be used to modify the request based on what the server can support.

SAFE METHODS	GET	HTTP/1.1 MUST IMPLEMENT THIS METHOD
NO ACTION ON SERVER	HEAD	INSPECT RESOURCE HEADERS
MESSAGE WITH	PUT	DEPOSIT DATA ON SERVER – INVERSE OF GET
BODY	POST	SEND INPUT DATA FOR PROCESSING
SEND DATA TO SERVER	PATCH	PARTIALLY MODIFY A RESOURCE
	TRACE	ECHO BACK RECEIVED MESSAGE
	OPTIONS	SERVER CAPABILITIES
	DELETE	DELETE A RESOURCE – NOT GUARANTEED

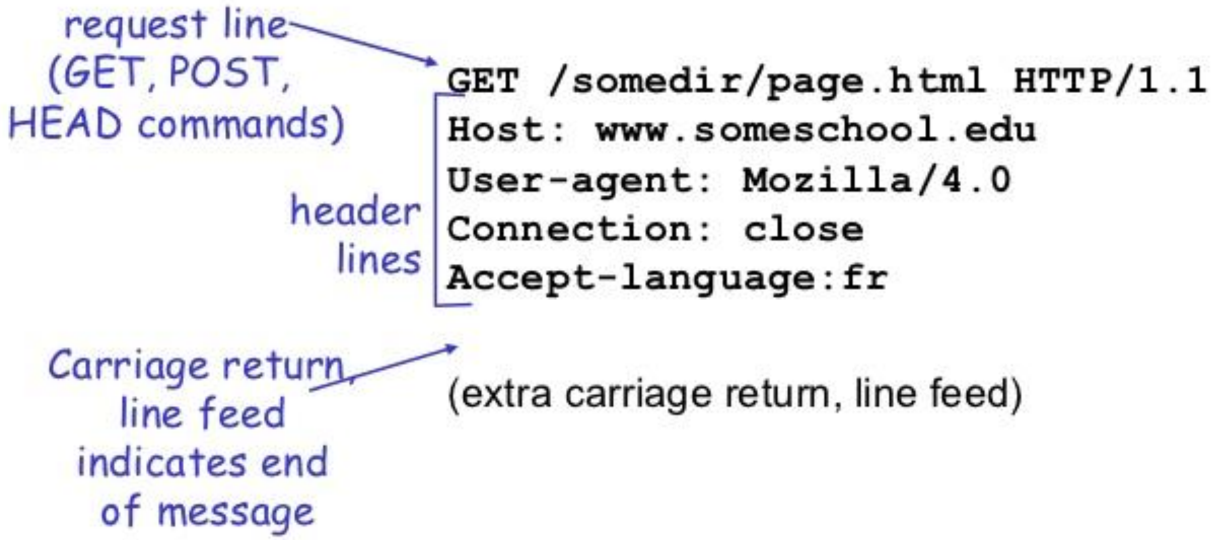
HTTP Request Format



- **Request types: GET, POST, HEAD, PUT, DELETE**
- **A URL given to browser:** `http://localhost:8000/`
- **Resulting request:** `GET / HTTP/1.1`
  - Someday, requests will contain the full URL not just path

HTTP request message

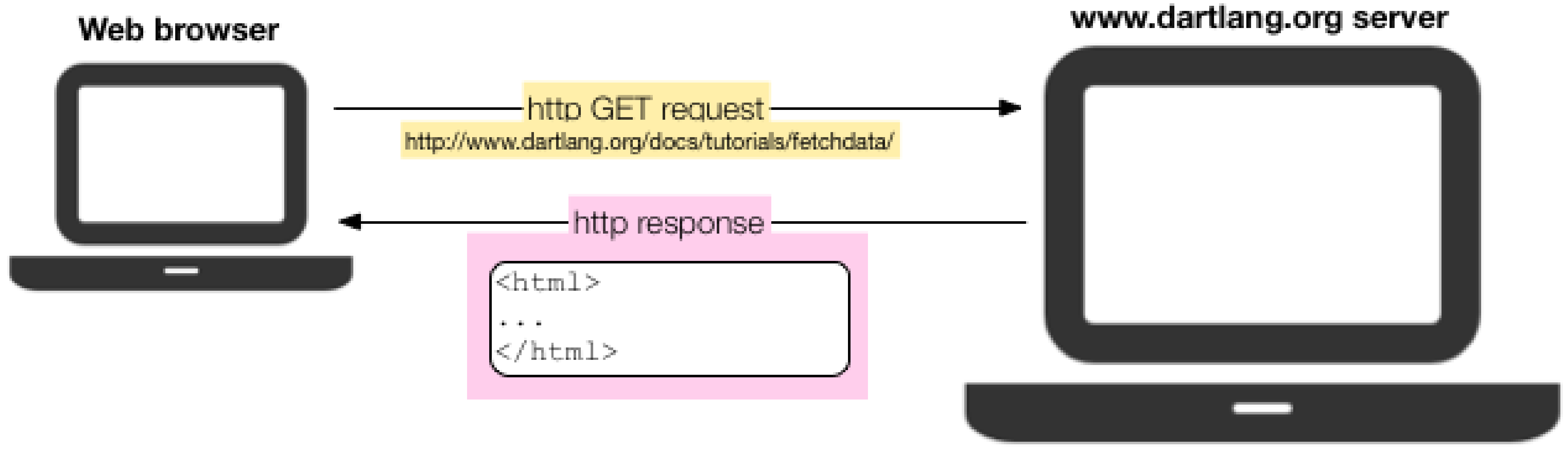
- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

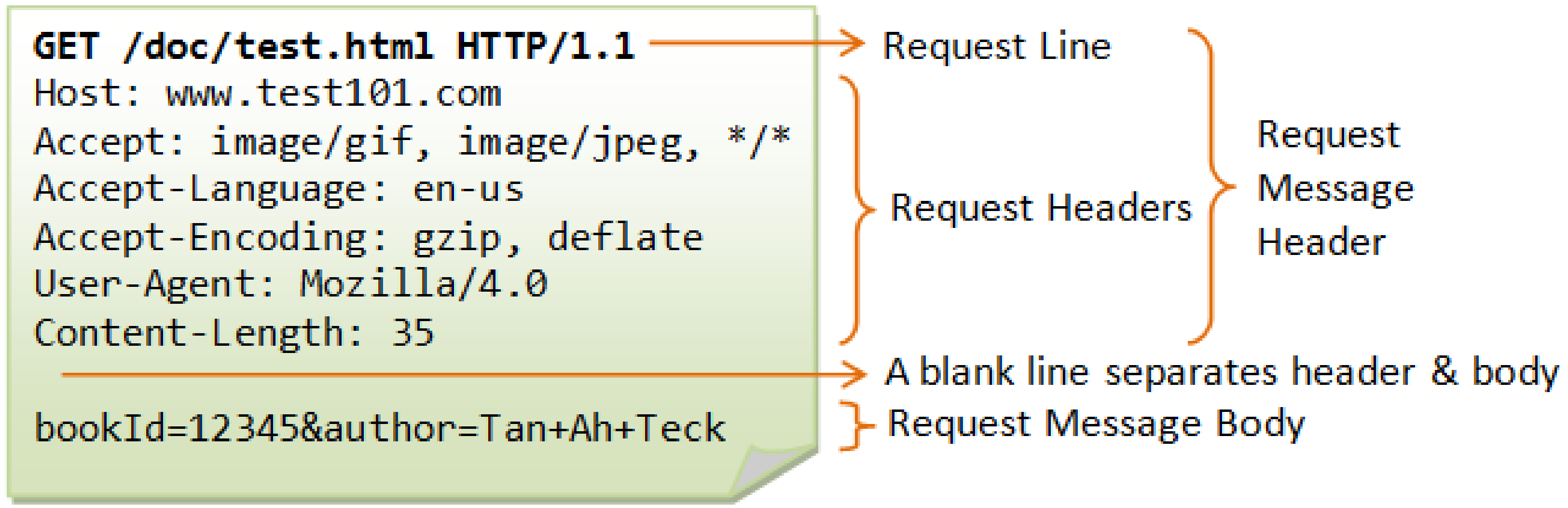

request line (GET, POST, HEAD commands)
header lines
Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

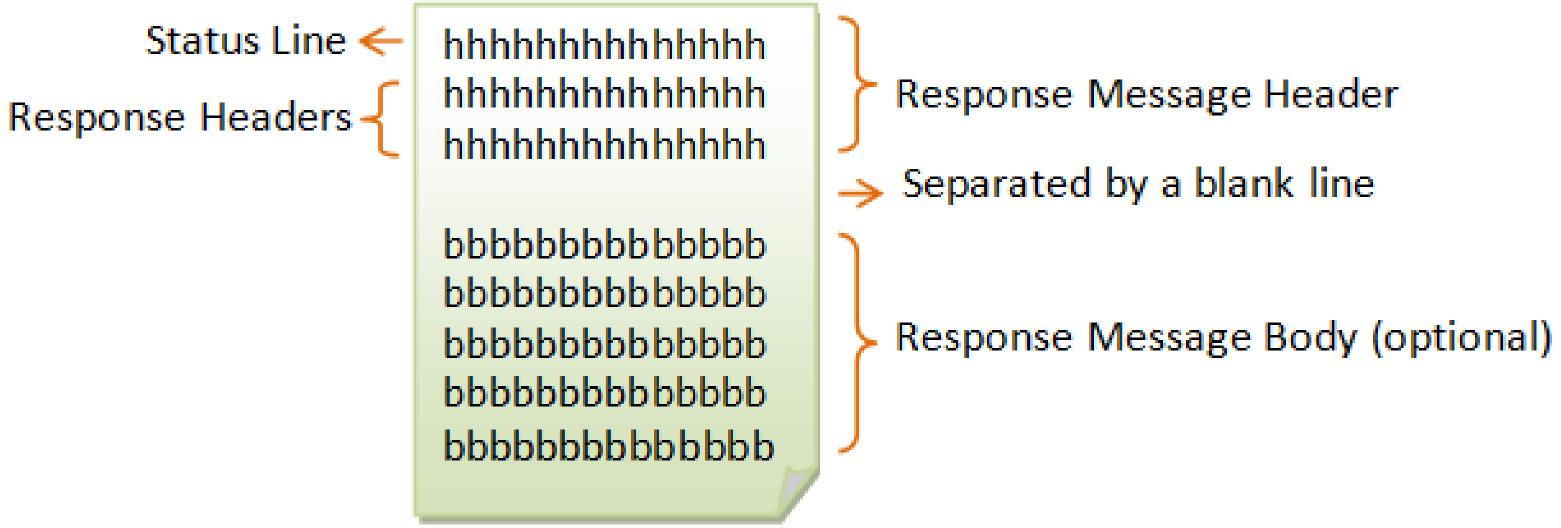
(extra carriage return, line feed)

HTTP GET

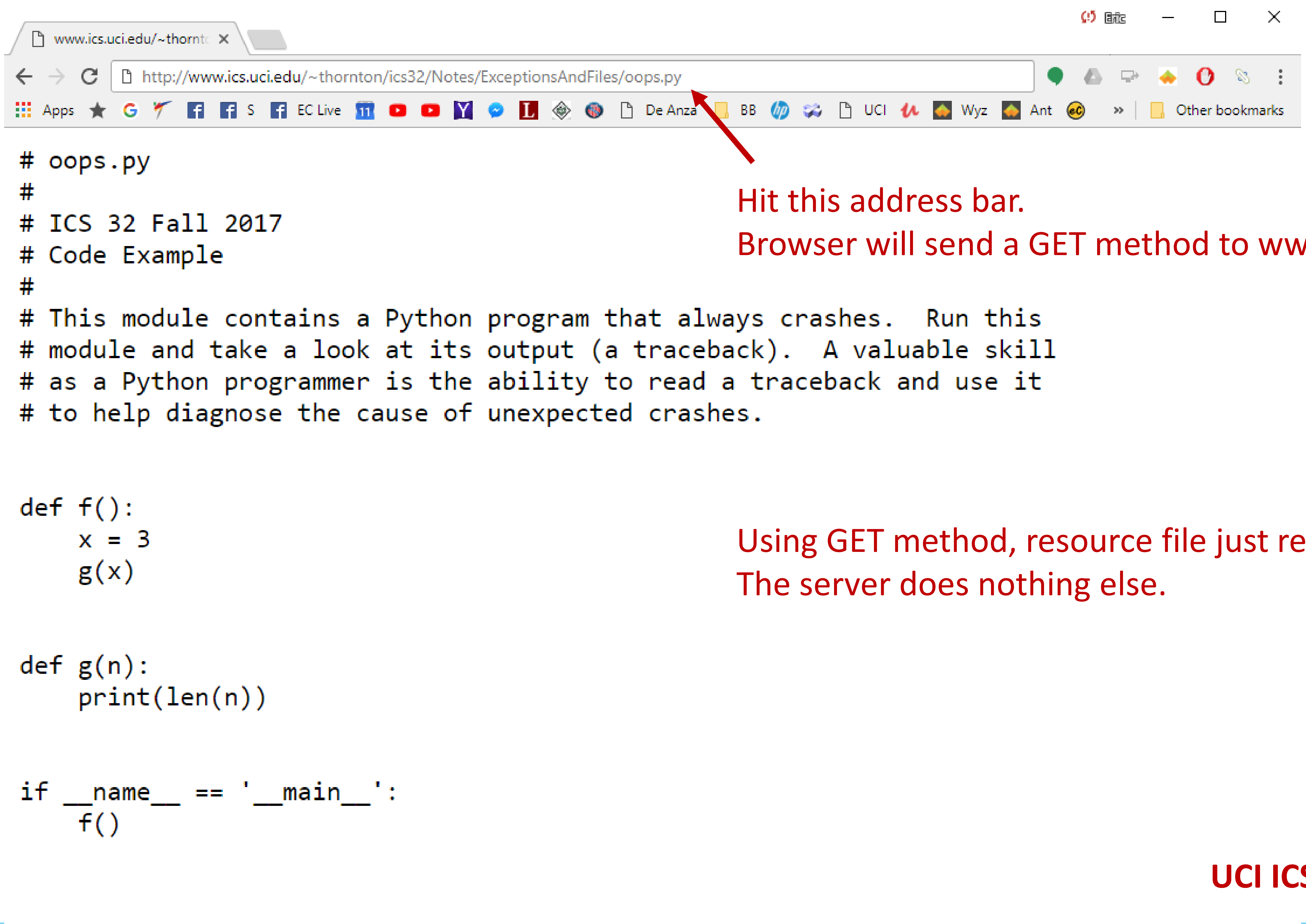

Web browser
www.dartlang.org server
http GET request
http://www.dartlang.org/docs/tutorials/fetchdata/
http response
<html>
...
</html>



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```



HTTP Response Message

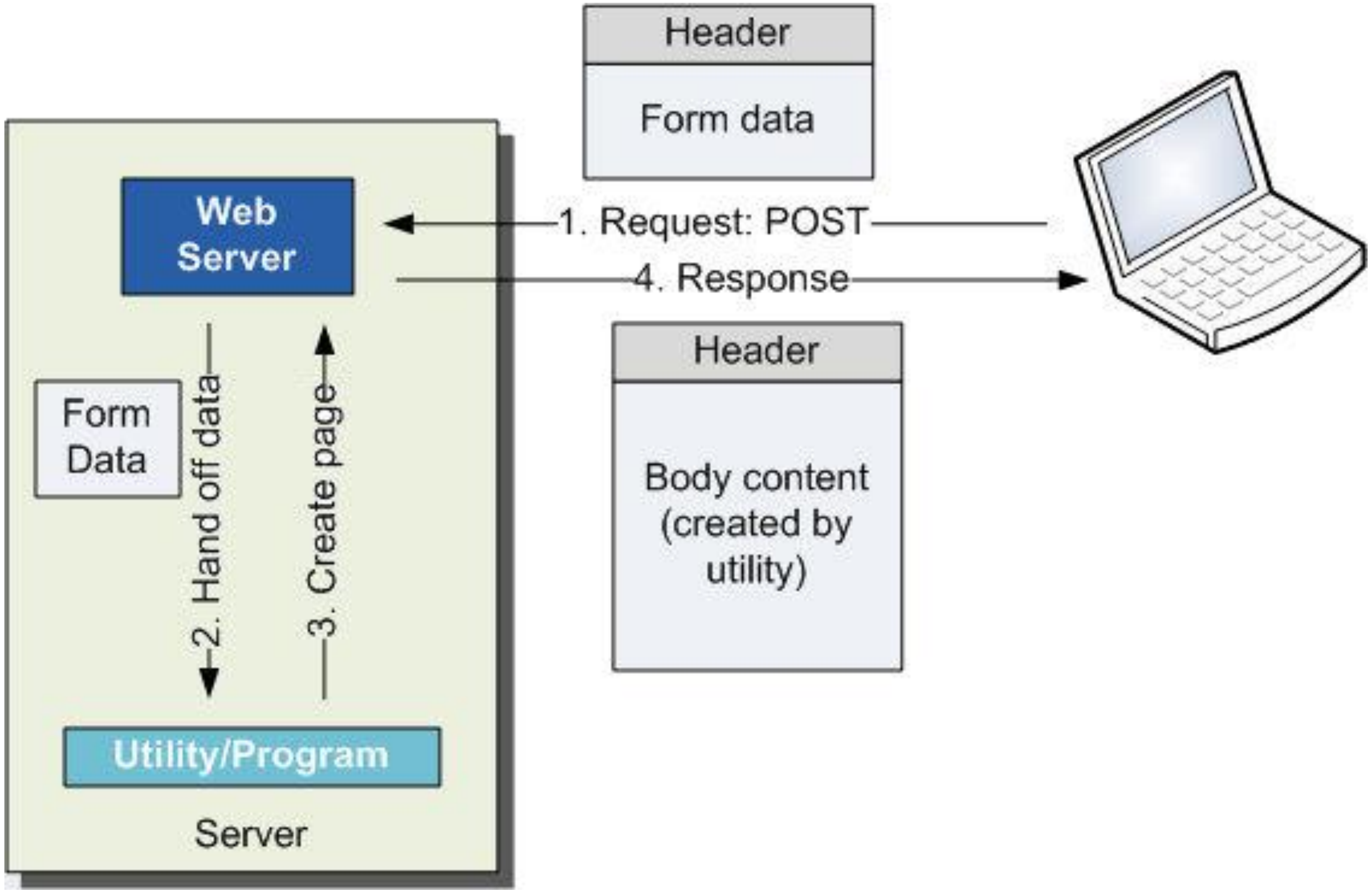

www.ics.uci.edu/~thornt
http://www.ics.uci.edu/~thornton/ics32/Notes/ExceptionsAndFiles/oops.py
Apps
EC Live
De Anza
BB
UCI
Wyz
Ant
Other bookmarks

```
# oops.py
#
# ICS 32 Fall 2017
# Code Example
#
# This module contains a Python program that always crashes.  Run this
# module and take a look at its output (a traceback).  A valuable skill
# as a Python programmer is the ability to read a traceback and use it
# to help diagnose the cause of unexpected crashes.


def f():
    x = 3
    g(x)


def g(n):
    print(len(n))


if __name__ == '__main__':
    f()
```

Hit this address bar.
Browser will send a GET method to www.ics.edu:80

Using GET method, resource file just returned.
The server does nothing else.

UCI ICS 32 Example


Header
Form data
Web Server
1. Request: POST
4. Response
Header
Body content (created by utility)
Form Data
2. Hand off data
3. Create page
Utility/Program
Server

Difference between GET & POST

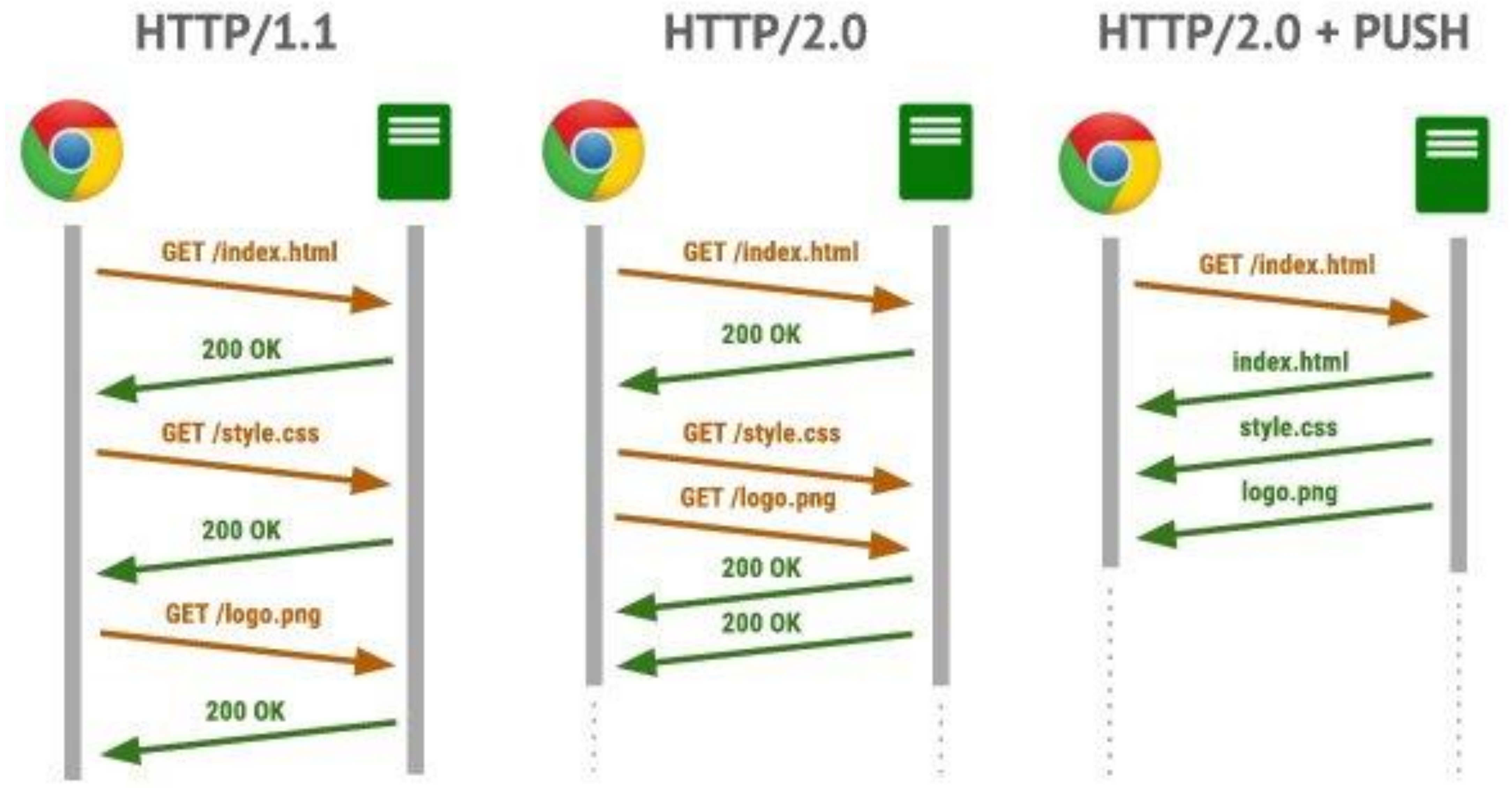
GET	POST
Can be cached	Never cached
Shown in URL of browser	Not shown in URL of browser

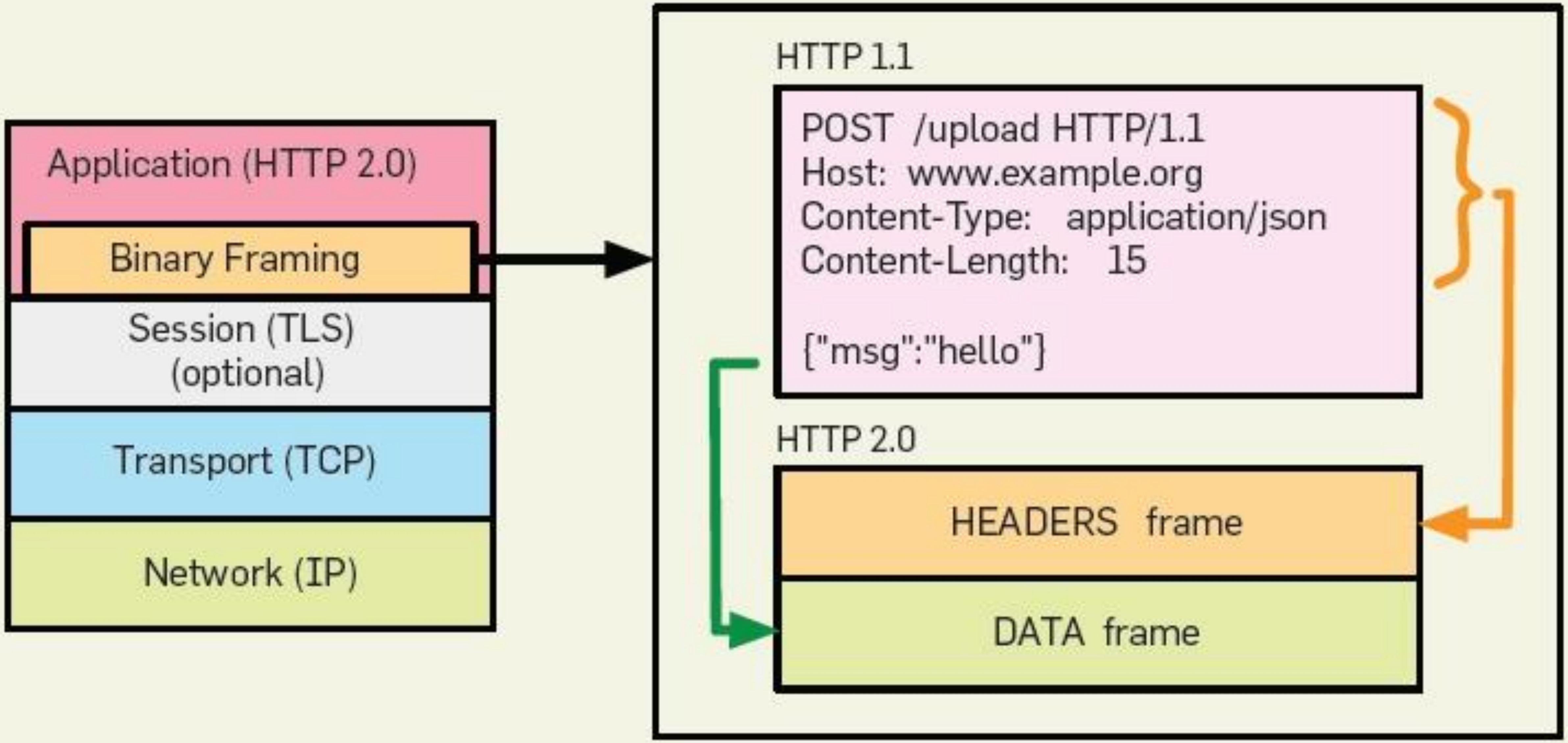
GET	POST
`GET is a safe method (idempotent)`	`POST is non-idempotent method`
`We can send limited data with GET method and it's sent in the header request URL`	`we can send large amount of data with POST because it's part of the body.`

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

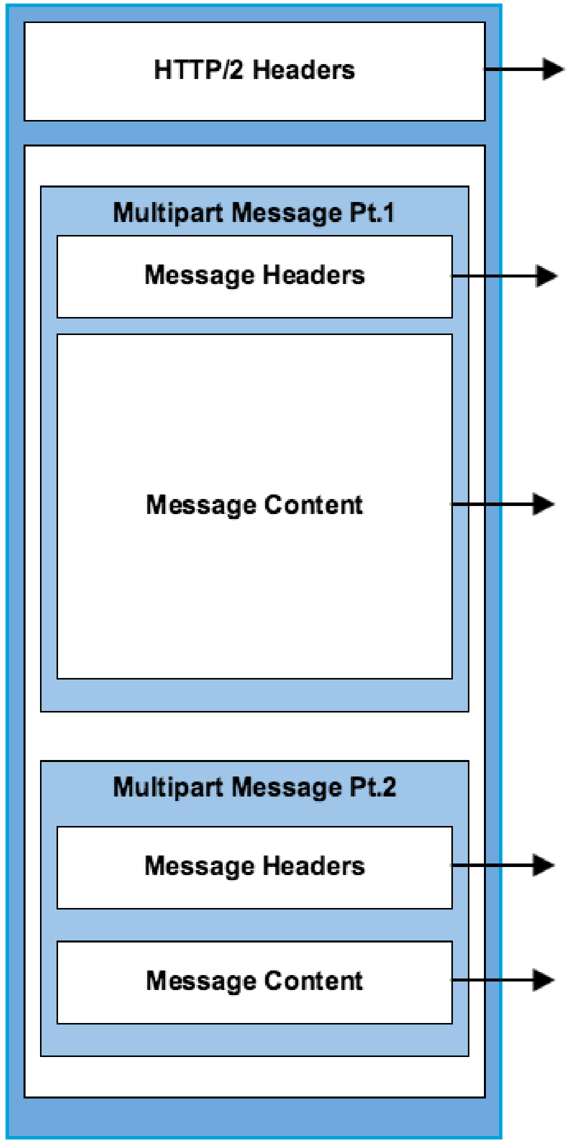
HTTP 2.0

LECTURE 4


HTTP/1.1
GET /index.html
200 OK
GET /style.css
200 OK
GET /logo.png
200 OK
HTTP/2.0
GET /index.html
200 OK
GET /style.css
GET /logo.png
200 OK
200 OK
HTTP/2.0 + PUSH
GET /index.html
index.html
style.css
logo.png



```
POST  /upload HTTP/1.1
Host:  www.example.org
Content-Type:   application/json
Content-Length:   15

{"msg":"hello"}
```



```
:method = POST
:scheme = https
:path:/{{API Version}}/events
authorization = Bearer foo-bar
content-type = multipart/form-data; boundary=this-
is-a-boundary
```

```
--this-is-a-boundary
Content-Disposition: form-data; name="metadata"
Content-Type: application/json; charset=UTF-08
```

```
{
   "event": {
      "header": {
         "namespace": "SpeechRecognizer",
         "name": "Recognize",
         "messageId": "message-123",
         "dialogRequestId": "dialogrequest-321"
      }
      "payload": {
         "profile": "CLOSE_TALK",
         "format": "AUDIO_L16_RATE_16000_CHANNELS_1"
      }
   }
}
```

```
--this-is-a-boundary
Content-Disposition: form-data; name="audio"
Content-Type: application/octet-stream
```

```
<<audio recorded via microphone>>
--this-is-a-boundary--
```

Python and HTTP

LECTURE 5

Python Program

- Python programs can make these requests and parse these responses, but that requires us to know a little bit about the format of each.
- A GET request in **HTTP/1.1** looks like this.

Request to download a resource

```
GET /~thornton/ics32/Notes/ExceptionsAndFiles/oops.py HTTP/1.1
Host: www.ics.uci.edu
```

Protocol

URL interpreted as

http://www.ics.uci.edu/~thornton/ics32/Notes/ExceptionsAndFiles/oops.py

Browser (or Python Program)

Given that information, a browser will know just what it needs to do:

1. Initiate a **socket** connection to port **80** on **www.ics.uci.edu**.
2. Use HTTP to request the page /~thornton/ics32/Notes/ExceptionsAndFiles/oops.py.
3. Parse the HTTP response and draw the page in the browser window.

First Line

- The first line of a GET request begins with the word GET, is followed by the web resource you want to download (the part of the URL that follows the protocol and host),
- and finally is followed by HTTP/1.1, as a way to indicate what protocol we expect to be using for the conversation.

Notice that there are spaces separating the word GET and the resource, and also between the resource and the HTTP/1.1. Because these spaces are part of the protocol — and because the presence of spaces elsewhere could make this more difficult for a server to handle — note that URLs are not permitted to contain spaces.

Second Line

- The second and subsequent lines contain what are called **headers**, which allow us to specify a variety of supplementary information that the server can use to figure out how to send us a response.
- In our case, we've included just one, a header called **Host:**, which specifies the **name** or **IP address** of the host we think we're connecting to; this is useful in the case that the same machine has multiple names, and is generally required in most **HTTP** requests.
- Additional headers include specifying what browser (and what version) is being used — so, for example, a server can send back different output for a small-sized screen like an iPhone than to a larger-sized screen like a laptop or desktop — or a variety of performance optimizations that are available, or security-related information (such as a password or an access token that grants access to a page that might otherwise be hidden).

A blank line following the last header informs the server that there are no more headers. At that point, the request is complete.

Response received from server.

```
HTTP/1.1 200 OK
Date: Sun, 29 Jan 2017 07:56:07 GMT
Server: Apache/2.2.15 (CentOS)
...
...
Content-Length: 435
Content-Type: text/plain; charset=UTF-8

# oops.py
#
# ICS 32 Winter 2017
# Code Example
...
...
if __name__ == '__main__':
    f()
```

The first line of the response indicates that the server agrees to have an **HTTP/1.1** conversation (that's the HTTP/1.1 part), followed by what's called a status code (in this case, **200**) and a reason phrase (in this case, OK). There are forty or so status codes that are defined as part of the HTTP/1.1 standard; the two most common ones are:

200 (OK), which means that everything went as planned, the server's way of saying "Okay, cool, here's the web page you asked for!"
404 (Not Found), which means that the server doesn't have the page that you asked to download. (If you've ever seen "404" show up in a browser during your travels around the web, this is why; it's an HTTP status code, "geekspeak" for a web page that doesn't exist.)

Format of Response Message

The first line of the response is followed by headers. The server determines what headers to send, and the details there are too numerous to list, but I've included a few of the more interesting ones in the example above:

- **Date** is the date/time at which the response was generated.
- **Server** specifies what type of server is being run and what version. As of this writing, the ICS web server is running version 2.2.15 of a server called Apache.
- **Content-Length** specifies the length, in bytes, of the content that will be sent back. This allows the client to know when the content has ended.
- **Content-Type** specifies what kind of content is being sent back. Browsers respond to the content type by deciding what to do with the content: web pages are shown in the browser, video is often displayed in a video plugin or an external media player, etc. If a browser isn't sure what to do with content, it generally just asks you if you want to save the file somewhere on your hard drive.

Top 10 Libraries for Web Interaction

- urllib
- urllib2
- urlparse
- httplib
- lxml
- rdflib
- json/simplejson
- mod_python, mod_wsgi
- bpython

Status Code

LECTURE 7

Status Codes (From Response - Header)

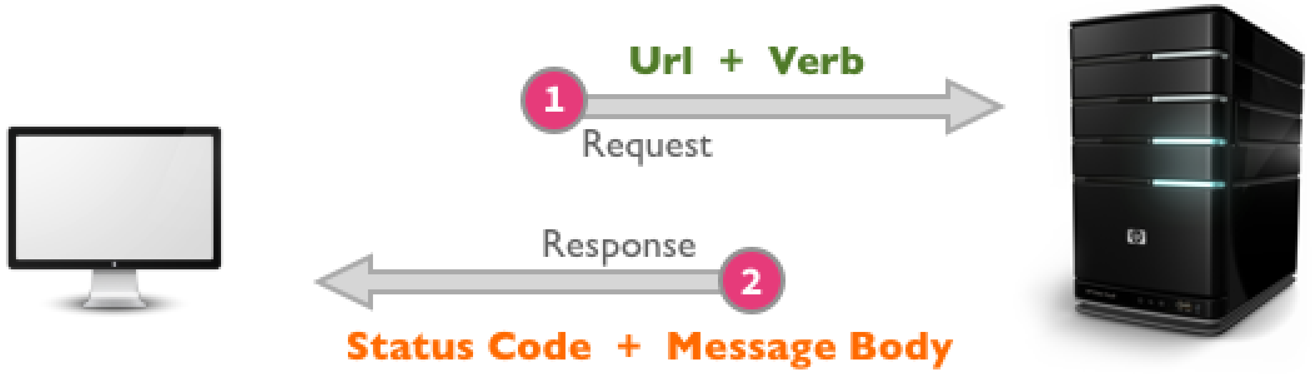
1xx: Informational Messages

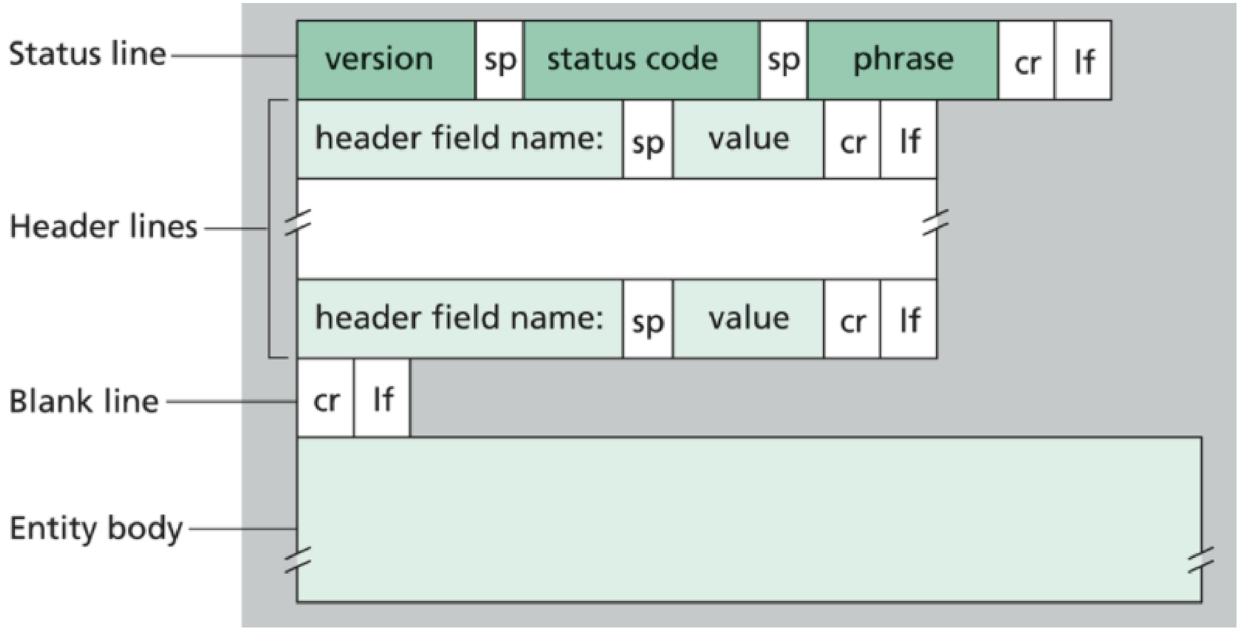
2xx: Successful

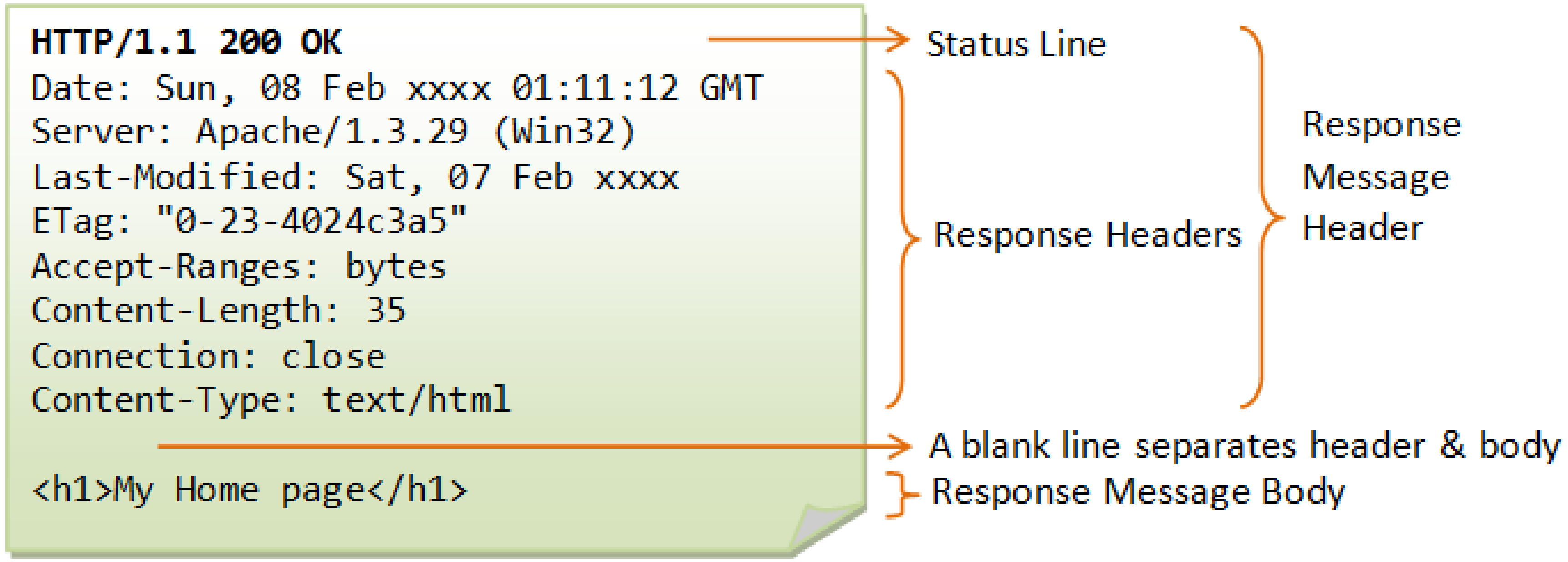
3xx: Redirection

4xx: Client Error

5xx: Server Error







```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

1xx: Informational Messages

All HTTP/1.1 clients are required to accept the Transfer-Encoding header.

This class of codes was introduced in HTTP/1.1 and is purely provisional.

The server can send a Expect: 100-continue message, telling the client to continue sending the remainder of the request, or ignore if it has already sent it.

HTTP/1.0 clients are supposed to ignore this header.

2xx: Successful

This tells the client that the request was successfully processed. The most common code is **200** OK. For a **GET** request, the server sends the resource in the message body. There are other less frequently used codes:

- **202 Accepted**: the request was accepted but may not include the resource in the response. This is useful for async processing on the server side. The server may choose to send information for monitoring.
- **204 No Content**: there is no message body in the response.
- **205 Reset Content**: indicates to the client to reset its document view.
- **206 Partial Content**: indicates that the response only contains partial content. Additional headers indicate the exact range and content expiration information.

3xx: Redirection

404 indicates that the resource is invalid and does not exist on the server.

This requires the client to take additional action. The most common use-case is to jump to a different URL in order to fetch the resource.

- **301 Moved Permanently**: the resource is now located at a new URL.
- **303 See Other**: the resource is temporarily located at a new URL. The Location response header contains the temporary URL.
- **304 Not Modified**: the server has determined that the resource has not changed and the client should use its cached copy. This relies on the fact that the client is sending ETag (Enttity Tag) information that is a hash of the content. The server compares this with its own computed ETag to check for modifications.

4xx: Client Error

These codes are used when the server thinks that the client is at fault, either by requesting an invalid resource or making a bad request. The most popular code in this class is **404 Not Found**, which I think everyone will identify with. **404** indicates that the resource is invalid and does not exist on the server. The other codes in this class include:

- **400 Bad Request**: the request was malformed.
- **401 Unauthorized**: request requires authentication. The client can repeat the request with the Authorization header. If the client already included the Authorization header, then the credentials were wrong.
- **403 Forbidden**: server has denied access to the resource.
- **405 Method Not Allowed**: invalid HTTP verb used in the request line, or the server does not support that verb.
- **409 Conflict**: the server could not complete the request because the client is trying to modify a resource that is newer than the client's timestamp. Conflicts arise mostly for PUT requests during collaborative edits on a resource.

5xx: Server Error

This class of codes are used to indicate a server failure while processing the request. The most commonly used error code is **500 Internal Server Error**. The others in this class are:

- **501 Not Implemented**: the server does not yet support the requested functionality.
- **503 Service Unavailable**: this could happen if an internal system on the server has failed or the server is overloaded. Typically, the server won't even respond and the request will timeout.

Python urllib module

LECTURE 7

urllib Module

- A high level module that allows clients to connect a variety of internet services
  - HTTP – http://
  - HTTPS – http://
  - FTP – ftp://
  - Local files – file://
- Works with typical URLs on the web...

urllib library

- High level standard library for retrieving data across the Web
- urllib libraray was split into several submodules in python 3:
  - urllib.request
  - urllib.parse
  - urllib.error
  - urllib.robotparser
- The **urllib.request.urlopen()** method is similar to the built-in function **open()** for opening files.

urllib protocols

- Supported protocols

```
u = urllib.urlopen("http://www.foo.com")
u = urllib.urlopen("https://www.foo.com/private")
u = urllib.urlopen("ftp://ftp.foo.com/README")
u = urllib.urlopen("file:///Users/beazley/blah.txt")
```

- Note: HTTPS only supported if Python configured with support for OpenSSL

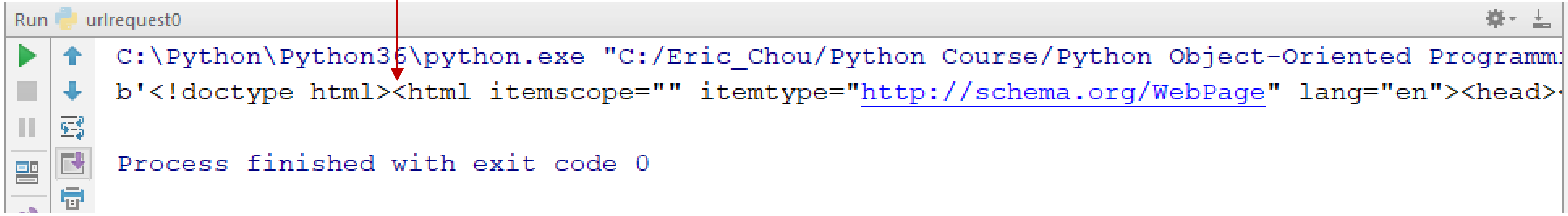
Demo Program: google.py

Go PyCharm!!!

google.py: (urlopen and close like files)

```
from urllib.request import *
req = urlopen("http://www.google.com")    # just like open a file
print(req.read())   # in encoded 'utf8 format'
req.close()
```

google.com's index page is downloaded. Data encoded in 'utf8' format.

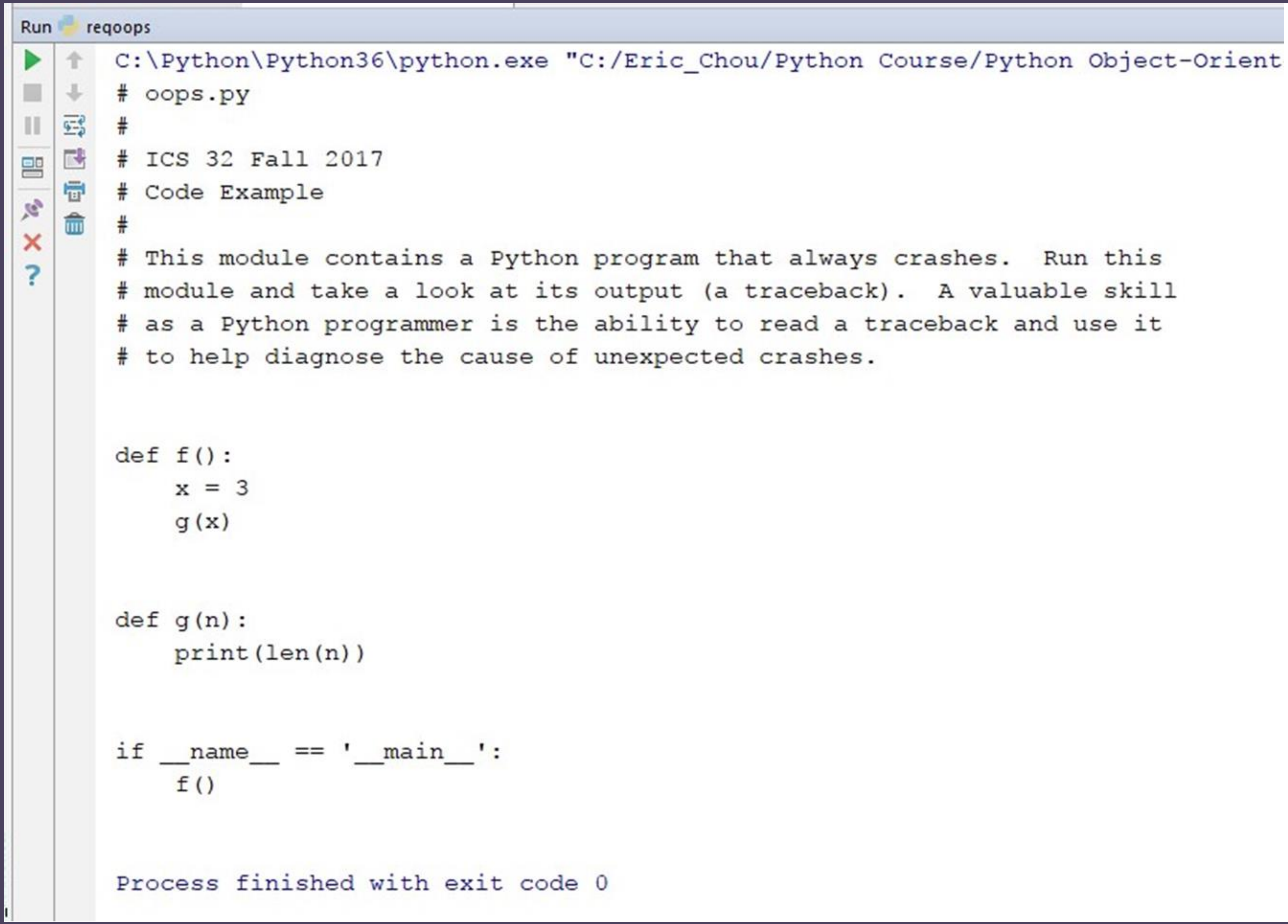


```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/Python Object-Oriented Programmi
b'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head>

Process finished with exit code 0
```

Demo Program: reqoops.py

```
from urllib.request import *
response = urlopen("http://www.ics.uci.edu/~thornton/ics32/Notes/ExceptionsAndFiles/oops.py")
r = response.read()
st = r.decode("utf8")
print(st)
response.close()
```

Decode the file in ‘utf8’bytes format back to Unicode format at client side.


Run reqoops

```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/Python Object-Orient
# oops.py
#
# ICS 32 Fall 2017
# Code Example
#
# This module contains a Python program that always crashes.  Run this
# module and take a look at its output (a traceback).  A valuable skill
# as a Python programmer is the ability to read a traceback and use it
# to help diagnose the cause of unexpected crashes.


def f():
    x = 3
    g(x)



def g(n):
    print(len(n))



if __name__ == '__main__':
    f()


Process finished with exit code 0
```

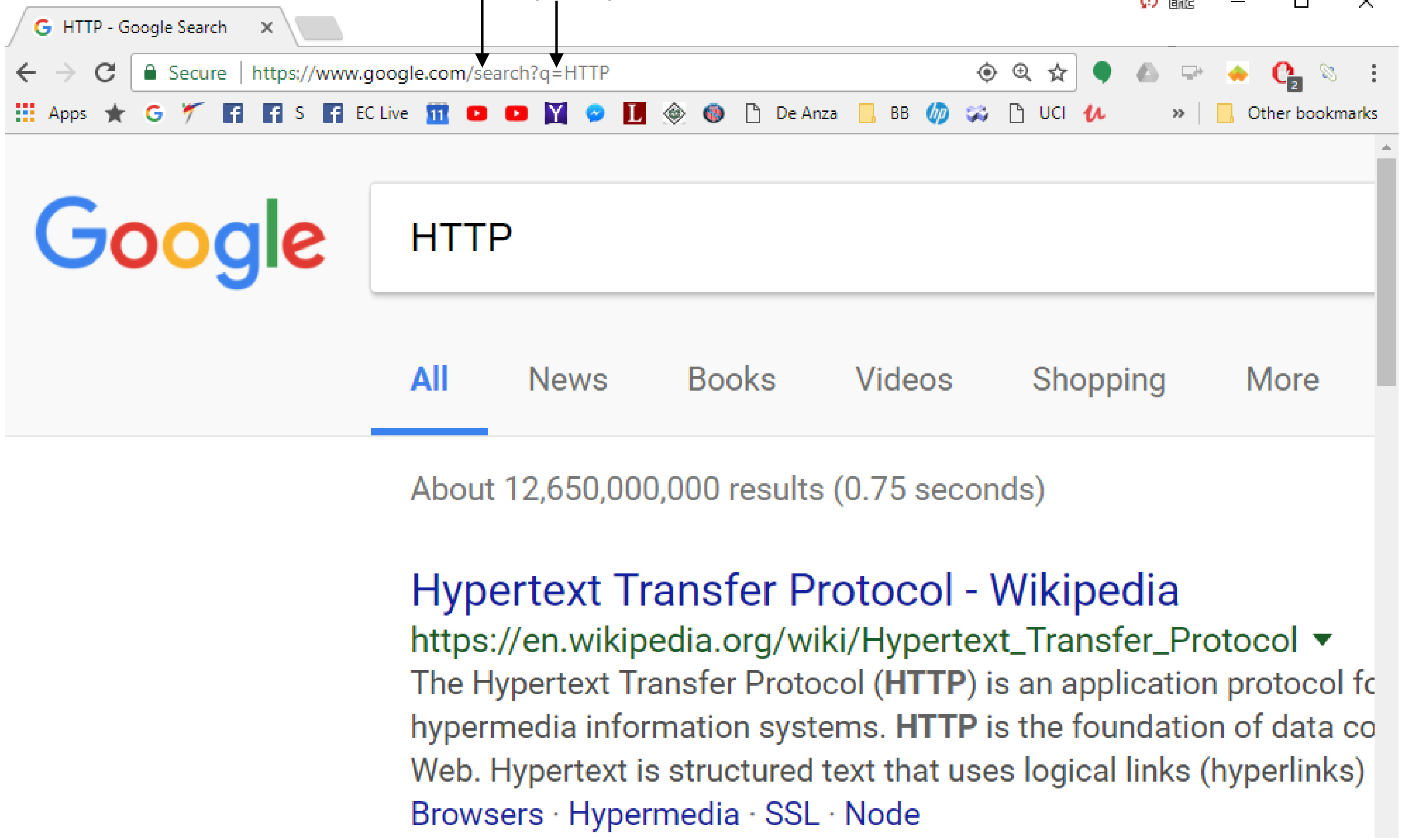
Python urllib Request

LECTURE 7

URL Request

Server: Google Search Engine

query HTTP

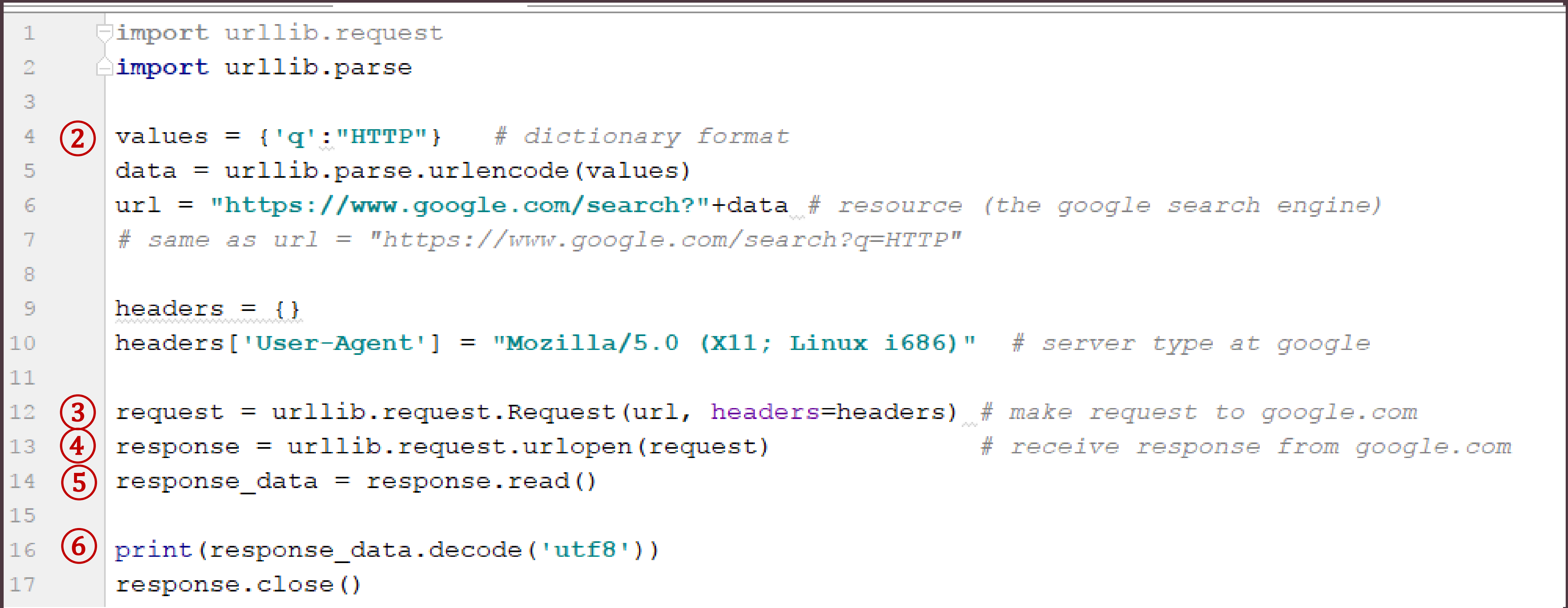


About 12,650,000,000 results (0.75 seconds)

Hypertext Transfer Protocol - Wikipedia
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
The Hypertext Transfer Protocol (**HTTP**) is an application protocol fc
hypermedia information systems. **HTTP** is the foundation of data co
Web. Hypertext is structured text that uses logical links (hyperlinks)
Browsers · Hypermedia · SSL · Node

Demo Program: makequery.py

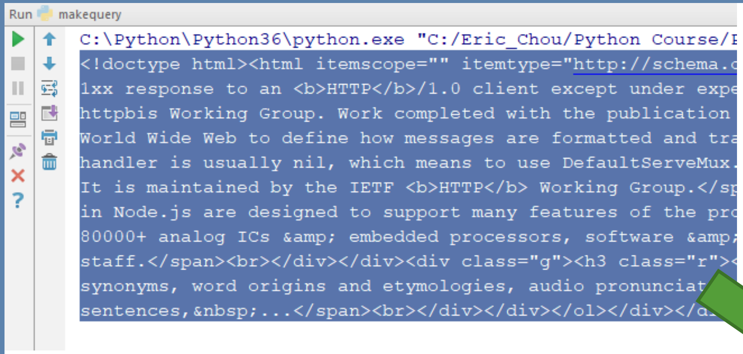
1. Using **POST** Method (Request-And-Response Pair)
2. Need to prepare URL and headers
3. Use **urllib.requestRequest(url, headers=headers)** to send request to server
4. Use **response_handler = urllib.urlopen(request_handler)** to receiver response from server
5. Use **response_data = response_handler.read()** to convert response to **utf8** string
6. Use **response_data.decode('utf8')** to convert the text string from **utf8** format to Unicode format
7. Grab the result and save in Brackets editor and post it on Chrome.



```
   import urllib.request
   import urllib.parse

②  values = {'q':"HTTP"}    # dictionary format
   data = urllib.parse.urlencode(values)
   url = "https://www.google.com/search?"+data # resource (the google search engine)
   # same as url = "https://www.google.com/search?q=HTTP"

   headers = {}
   headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)"  # server type at google

③  request = urllib.request.Request(url, headers=headers) # make request to google.com
④  response = urllib.request.urlopen(request)             # receive response from google.com
⑤  response_data = response.read()

⑥  print(response_data.decode('utf8'))
   response.close()
```

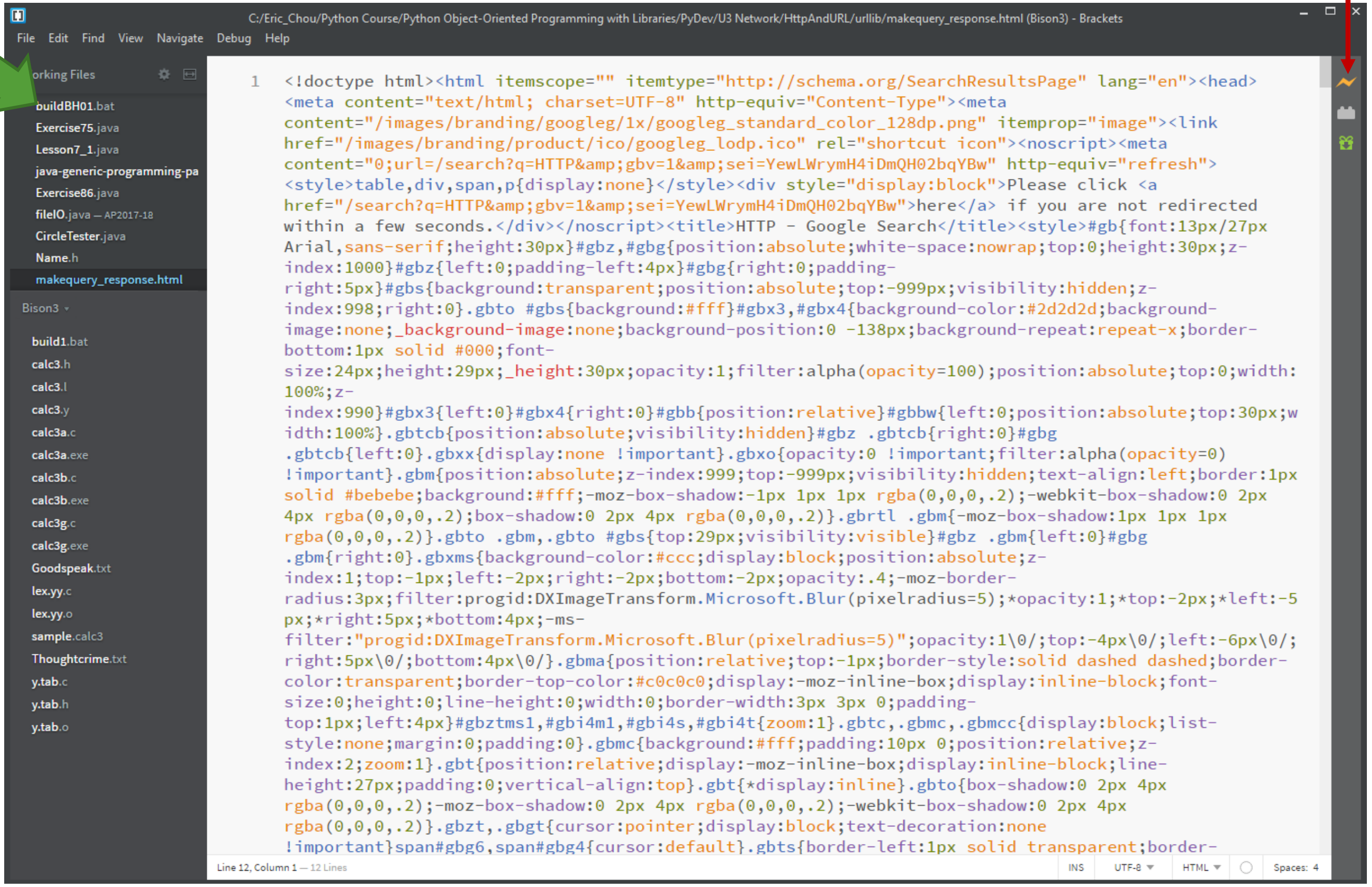
Note: Every service engine, every API may need different URL or headers settings.

Click to show in Chrome Browser



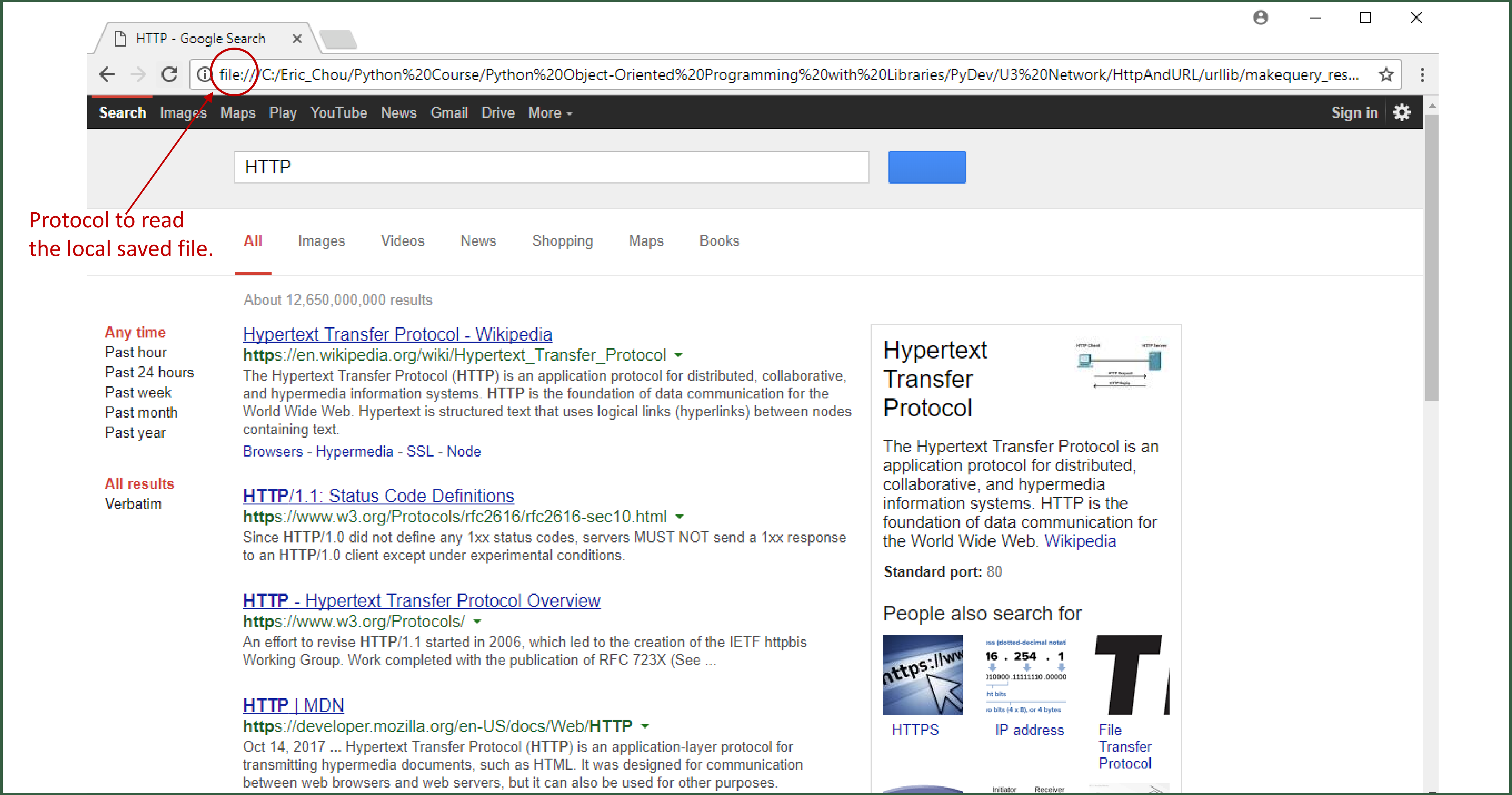
```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/P
<!doctype html><html itemscope="" itemtype="http://schema.o
1xx response to an <b>HTTP</b>/1.0 client except under expe
httpbis Working Group. Work completed with the publication
World Wide Web to define how messages are formatted and tra
handler is usually nil, which means to use DefaultServeMux.
It is maintained by the IETF <b>HTTP</b> Working Group.</sp
in Node.js are designed to support many features of the pro
80000+ analog ICs &amp; embedded processors, software &amp;
staff.</span><br></div></div><div class="g"><h3 class="r"><
synonyms, word origins and etymologies, audio pronunciat
sentences,&nbsp;...</span><br></div></div></ol></div></d
```

Grab and paste in brackets editor



```html
<!doctype html><html itemscope="" itemtype="http://schema.org/SearchResultsPage" lang="en"><head>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><link
href="/images/branding/product/ico/googleg_lodp.ico" rel="shortcut icon"><noscript><meta
content="0;url=/search?q=HTTP&amp;gbv=1&amp;sei=YewLWrymH4iDmQH02bqYBw" http-equiv="refresh">
<style>table,div,span,p{display:none}</style><div style="display:block">Please click <a
href="/search?q=HTTP&amp;gbv=1&amp;sei=YewLWrymH4iDmQH02bqYBw">here</a> if you are not redirected
within a few seconds.</div></noscript><title>HTTP - Google Search</title><style>#gb{font:13px/27px
Arial,sans-serif;height:30px}#gbz,#gbg{position:absolute;white-space:nowrap;top:0;height:30px;z-
index:1000}#gbz{left:0;padding-left:4px}#gbg{right:0;padding-
right:5px}#gbs{background:transparent;position:absolute;top:-999px;visibility:hidden;z-
index:998;right:0}.gbto #gbs{background:#fff}#gbx3,#gbx4{background-color:#2d2d2d;background-
image:none;_background-image:none;background-position:0 -138px;background-repeat:repeat-x;border-
bottom:1px solid #000;font-
size:24px;height:29px;_height:30px;opacity:1;filter:alpha(opacity=100);position:absolute;top:0;width:
100%;z-
index:990}#gbx3{left:0}#gbx4{right:0}#gbb{position:relative}#gbbw{left:0;position:absolute;top:30px;w
idth:100%}.gbtcb{position:absolute;visibility:hidden}#gbz .gbtcb{right:0}#gbg
.gbtcb{left:0}.gbxx{display:none !important}.gbxo{opacity:0 !important;filter:alpha(opacity=0)
!important}.gbm{position:absolute;z-index:999;top:-999px;visibility:hidden;text-align:left;border:1px
solid #bebebe;background:#fff;-moz-box-shadow:-1px 1px 1px rgba(0,0,0,.2);-webkit-box-shadow:0 2px
4px rgba(0,0,0,.2);box-shadow:0 2px 4px rgba(0,0,0,.2)}.gbrtl .gbm{-moz-box-shadow:1px 1px 1px
rgba(0,0,0,.2)}.gbto .gbm,.gbto #gbs{top:29px;visibility:visible}#gbz .gbm{left:0}#gbg
.gbm{right:0}.gbxms{background-color:#ccc;display:block;position:absolute;z-
index:1;top:-1px;left:-2px;right:-2px;bottom:-2px;opacity:.4;-moz-border-
radius:3px;filter:progid:DXImageTransform.Microsoft.Blur(pixelradius=5);*opacity:1;*top:-2px;*left:-5
px;*right:5px;*bottom:4px;-ms-
filter:"progid:DXImageTransform.Microsoft.Blur(pixelradius=5)";opacity:1\0/;top:-4px\0/;left:-6px\0/;
right:5px\0/;bottom:4px\0/}.gbma{position:relative;top:-1px;border-style:solid dashed dashed;border-
color:transparent;border-top-color:#c0c0c0;display:-moz-inline-box;display:inline-block;font-
size:0;height:0;line-height:0;width:0;border-width:3px 3px 0;padding-
top:1px;left:4px}#gbztms1,#gbi4m1,#gbi4s,#gbi4t{zoom:1}.gbtc,.gbmc,.gbmcc{display:block;list-
style:none;margin:0;padding:0}.gbmc{background:#fff;padding:10px 0;position:relative;z-
index:2;zoom:1}.gbt{position:relative;display:-moz-inline-box;display:inline-block;line-
height:27px;padding:0;vertical-align:top}.gbt{*display:inline}.gbto{box-shadow:0 2px 4px
rgba(0,0,0,.2);-moz-box-shadow:0 2px 4px rgba(0,0,0,.2);-webkit-box-shadow:0 2px 4px
rgba(0,0,0,.2)}.gbzt,.gbgt{cursor:pointer;display:block;text-decoration:none
!important}span#gbg6,span#gbg4{cursor:default}.gbts{border-left:1px solid transparent;border-
```

Saved local search result as the response from google.com/search?q=HTTP


HTTP - Google Search
file:///C:/Eric_Chou/Python%20Course/Python%20Object-Oriented%20Programming%20with%20Libraries/PyDev/U3%20Network/HttpAndURL/urllib/makequery_res...
Search Images Maps Play YouTube News Gmail Drive More
Sign in
HTTP
All Images Videos News Shopping Maps Books
Any time
Past hour
Past 24 hours
Past week
Past month
Past year
All results
Verbatim

Protocol to read the local saved file.

About 12,650,000,000 results

Hypertext Transfer Protocol - Wikipedia
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.
Browsers - Hypermedia - SSL - Node

HTTP/1.1: Status Code Definitions
https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

HTTP - Hypertext Transfer Protocol Overview
https://www.w3.org/Protocols/
An effort to revise HTTP/1.1 started in 2006, which led to the creation of the IETF httpbis Working Group. Work completed with the publication of RFC 723X (See ...

HTTP | MDN
https://developer.mozilla.org/en-US/docs/Web/HTTP
Oct 14, 2017 ... Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes.

Hypertext Transfer Protocol

The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Wikipedia

Standard port: 80

People also search for

HTTPS IP address File Transfer Protocol