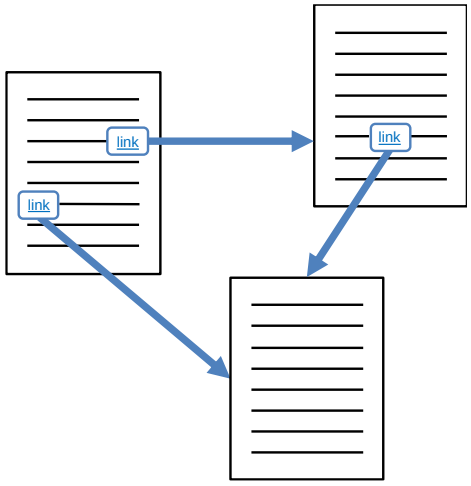
Web Applications

Name: _____ ( ) Class: _____ Date: _____

Lesson 1: Hypertext Transfer Protocol

The World Wide Web (WWW) and Hypertext Transfer Protocol (HTTP)

In 1990, a computer scientist named Tim-Berners Lee working at the European Organisation for Nuclear Research (also known as CERN) proposed a kind of application to facilitate the sharing and management of information. The application would be asymmetric with very different client and server programs. Users would be able to run the client program and request for documents from computers running the server program using one of several possible commands. The documents sent over would also be written in a new computer language to facilitate **hyperlinking**, which is the ability to reference one document from another in an easy-to-follow manner.



Tim-Berners Lee's proposal was tremendously successful, and today, we call this application the **World Wide Web (WWW)** or just the **web**. To use the web, we often use an all-in-one program called a **web browser** that serves as both a client for requesting documents from web servers as well as a viewer for the documents that are sent over. In particular, web browsers are designed to view documents written using the **Hypertext Markup Language (HTML)**. In turn, the protocol used to request for such documents is named the **Hypertext Transfer Protocol (HTTP)**.

As originally designed, requests and responses made using HTTP are not encrypted and can be read or modified by any party involved in routing that data across the Internet. To prevent this, web requests and responses can be encrypted using an extended version of HTTP named **HTTPS (HTTP Secure)**. However, HTTPS can be more difficult to set up compared to plain HTTP, so for learning purposes we will focus on plain HTTP only. Nevertheless, be aware that HTTPS is usually preferred and is in fact required for newer versions of HTTP.

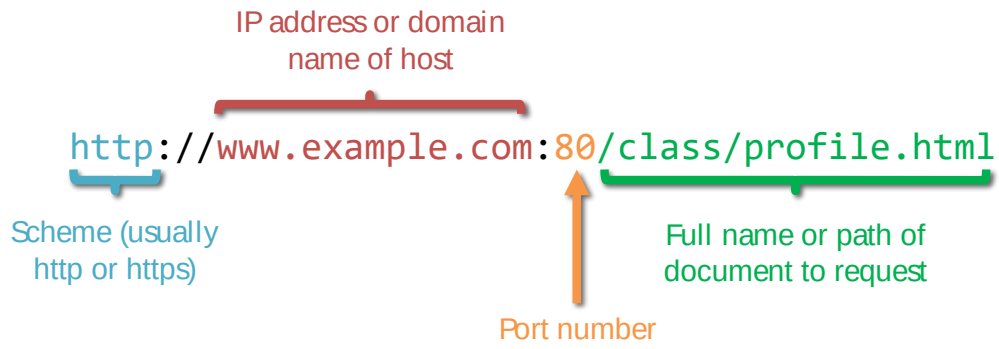
Web Applications

Locating Web Documents

To request for a web document, you need to give a web browser three things:

1. The IP address or domain name of the **host** server that has the document.
2. The port number that the web server program is listening on.
3. The full name of the document being requested. This is called the **path**.

All three pieces of information are supplied to the web browser using a single string called a **Universal Resource Locator (URL)**. For web documents, URLs typically start with a **scheme** that is either `http` or `https` depending on whether HTTP or HTTPS is being used, followed by a colon and two slashes, the host IP address or domain name, an optional colon and port number, then finally the path. In this context, the scheme is a string that specifies which protocol to use.



If the second colon and port number are left out, a default port number is used instead. The default port number is 80 for HTTP and 443 for HTTPS.

URLs must also meet other requirements that treat some **reserved characters** as special and allow only **unreserved characters** such as letters and numeric digits as well as the hyphen `-`, underscore `_`, period `.` and tilde `~` characters to be used.

To represent all other characters, we must use **percent-encoding** by replacing the character with a percent sign % followed by the character's ASCII code converted to a 2-digit hexadecimal number. Some percent-encodings of common characters are provided as examples in the following table:

Character	*(space)*	#	%	/	:	?
ASCII Code	32	35	37	47	58	63
Percent-Encoding	%20	%23	%25	%2F	%3A	%3F

1 Identify the scheme, host, port number and path components for each of the following URLs. If the port number component is missing, use an appropriate default value instead. If the path component is empty, use "*(empty)*" instead.

Example

`http://www.example.com:8080`

Scheme	**Host**	**Port No.**	**Path**
http	www.example.com	8080	*(empty)*

a) `http://www.example.com`

Scheme	**Host**	**Port No.**	**Path**

b) `https://secure.example.com:4430/`

Scheme	**Host**	**Port No.**	**Path**

c) `https://example.com:80/example.com`

Scheme	**Host**	**Port No.**
Path		

d) `https://example.org/example.com/example.html`

Scheme	Host	Port No.
Path		

e) `http://complex.example.org/complex.example.com`

Scheme	Host	Port No.
Path		

f) `https://www.moe.gov.sg/news/press-releases`

Scheme	Host	Port No.
Path		

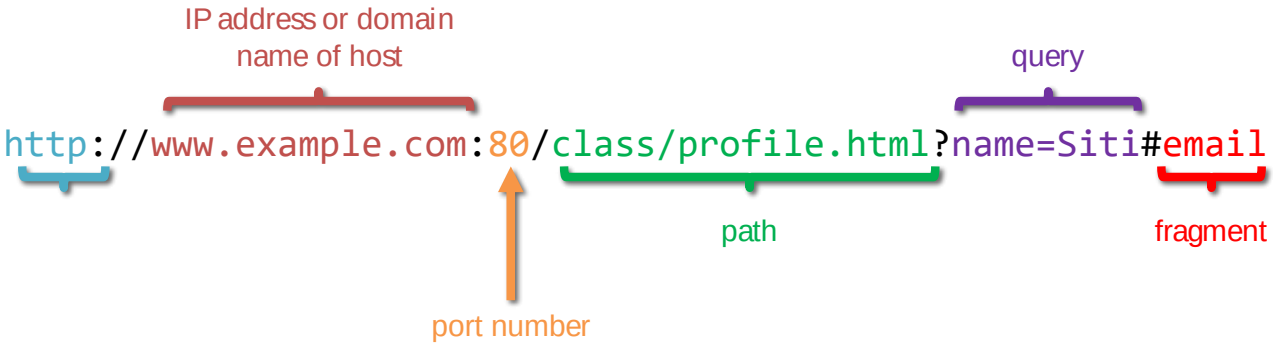
In addition to the scheme, host, port number and path components, a URL can also have two optional components.

After the path, there may be a question mark `?` followed by a **query** which provides additional information to the web server and may modify the server's response.

After the query (or path, if there is no query), the URL may also end with a hash character # followed by a **fragment**. Note that unlike queries, fragments are **NOT** sent to the web server and are only meant to control how a web browser displays the response. Changing the fragment component of a URL when making a HTTP request generally does not change the server's response.

Web Applications

The different parts of a URL are summarised below:



2 Identify the scheme, host, port number, path, query and fragment components for each of the following URLs. If the port number component is missing, use an appropriate default value instead. If any of the path, query or fragment components are empty, use "*(empty)*" instead.

a) `http://example.net:5000/com?hello=world`

Scheme	Host	Port No.	Path
Query		**Fragment**	

b) `https://sub.domain.example.org/9999/#hello-world`

Scheme	Host	Port No.	Path
Query		**Fragment**	

c) `https://example.org/res/search?keyword=computing#page-1`

Scheme	Host	Port No.	Path
Query		**Fragment**	

Web Applications

The URL for a web document is usually entered into the address bar of a web browser. For convenience, web browsers will usually let users omit the first portion of the URL before the host as it is assumed that most users intend to use HTTP. (For example, you can enter `example.com` as a shortcut for `http://example.com`.)

After a URL is entered, the web browser creates a socket and connects it to the specified host and port number (80 by default). After a connection is established, the web browser sends a request for the document using the path given by the URL. This is a **HTTP request**. If all goes well, the web server then responds with the requested document. This is a **HTTP response**. These requests and responses are encoded as bytes using rules defined by the HTTP protocol.

3 In HTTP, what is the difference between a “request” and a “response”?

A A HTTP request comes after a HTTP response.

B A HTTP request is sent from the client to the server while a HTTP response is sent from the server to the client.

C A HTTP request is sent from the server to the client while a HTTP response is sent from the client to the server.

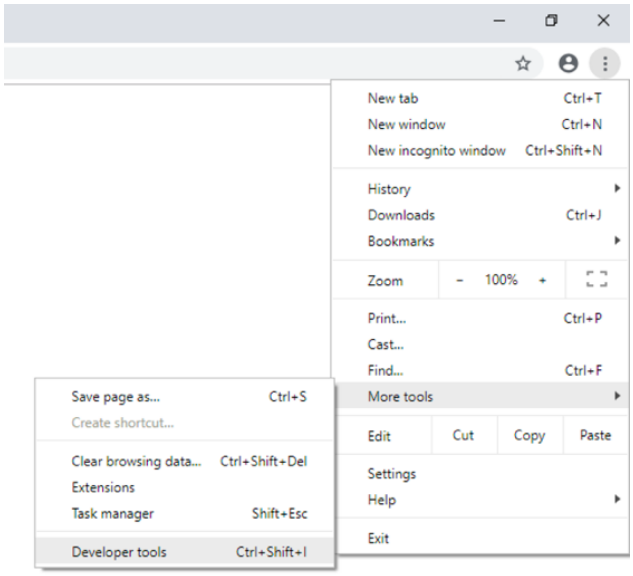
D A HTTP request must be shorter than a HTTP response.

( )

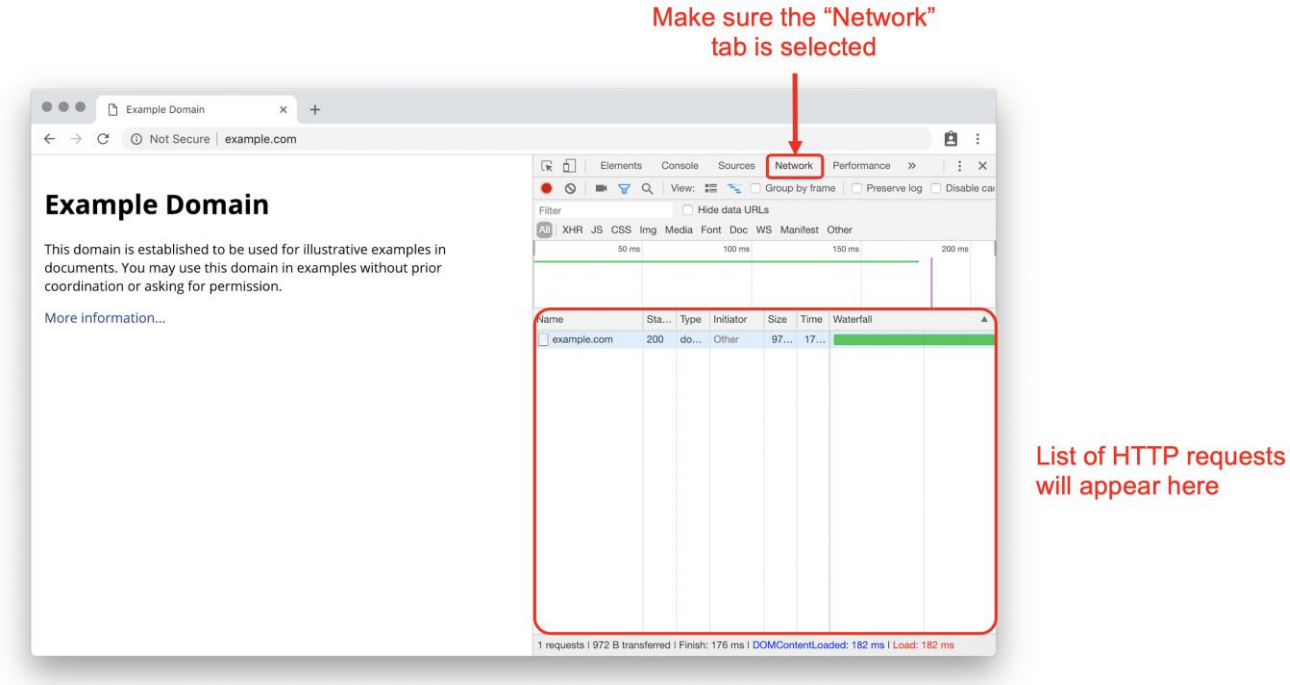
HTTP Requests

Some web browsers like Google Chrome and Microsoft Edge come with built-in “Developer Tools” that are useful for learning about HTTP. The instructions in the following tasks are customised for Google Chrome.

Launch Google Chrome and open Developer Tools from the ellipsis menu under the “More tools” item.

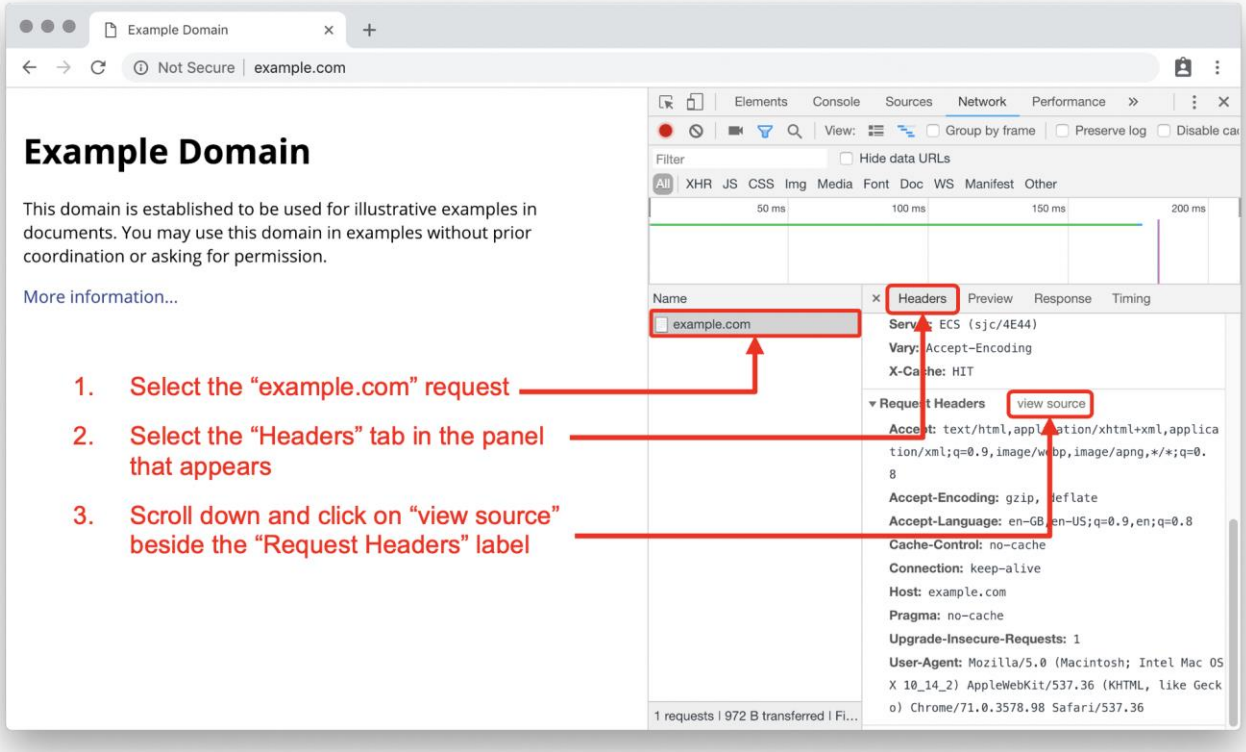


With the Developer Tools open, select the “Network” tab. Next, enter `http://example.com` into the address bar and press enter. After the page has loaded, the Network tab should list out all the HTTP requests made by the web browser. Select a request and you should be able to read more details about that request in a separate pane.



Scroll down in the details pane and you should see a section labelled “Request Headers” with a link labelled “view source” next to it. (If the “view source” link is missing, make sure you are using HTTP and not HTTPS. You may also need to refresh the page.) Click on the link and you will see a portion of the bytes that were sent by your web browser to the web server using a socket.

Web Applications



Unlike more complex protocols, HTTP 1.0 and 1.1 are text-based, so HTTP requests and responses are still readable even when they are encoded as bytes. Each request starts with a **request line**, followed by **header fields**, followed by an empty line and an optional **message body**.

request line

```
GET / HTTP/1.1
```

header fields

```
Host: example.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

empty line *(invisible)*

message body *(optional)*

missing as this example has no message body

Request Line

The request line contains three important pieces of information separated by spaces: the HTTP command (also called **method** or **verb**) that the client wants to perform, the path of the relevant document (together with the query portion of the URL, if any) and the version of HTTP used by the client.

The two most common HTTP methods are `GET` and `POST`. `GET` is used whenever we simply want to *get* or request for a web document without the intention to make any changes. For instance, when we enter `http://example.com/test.html` into a web browser's address bar, the web browser actually creates a socket to the `example.com` server and sends a request with the following request line:

```
GET /test.html HTTP/1.1
```

Notice that the path that is used in the request line comes directly from the URL that was entered in the address bar. An exception is if the URL's path is empty, such as for the URL `http://example.com` (without a slash). In this case, a path of `/` is used instead so `http://example.com` (without a slash) and `http://example.com/` (with a slash) result in the same HTTP request.

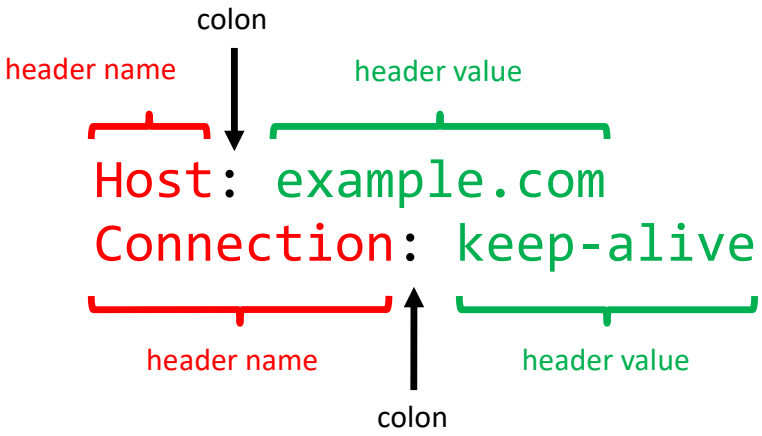
If the URL has a query component, it is included in the request line together with the question mark separator. For instance, when `http://example.com/hello?q=world` is entered into a web browser's address bar, the following request line is used:

```
GET /hello?q=world HTTP/1.1
```

On the other hand, `POST` is used when we intend to *post* or submit data and make changes to data on the web server. In this case, the request will usually include a message body containing the data that is being submitted. We will see examples of what a `POST` request looks like later in this practical task.

Header Fields

HTTP header fields provide additional information to the web server so it can customise its response accordingly. Each header field has its own line that consists of a name, followed by a colon, followed by a space and then the header field's value.



The only header field that is required by HTTP 1.1 is named `Host` and is copied directly from the host portion of the URL. This allows the same web server to provide different responses depending on which domain name is used to reach the server. For example, if `http://example.com/test.html` is entered into a web browser's address bar, the HTTP request will use the following header field:

```
Host: example.com
```

Some other important HTTP header fields are `Content-Type` (to specify how the message body should be interpreted) and `Content-Length` (to specify the message body's size in bytes). For a list of possible header fields and what they are used for, you may refer to https://en.wikipedia.org/wiki/List_of_HTTP_header_fields.

Message Body

After the header fields, a HTTP request is required to have an empty line followed by an optional message body.

For `GET` requests, the message body is ignored so this portion of the request is usually omitted. However, for `POST` requests, the message body typically contains the data that is being submitted. For instance, the following is a simplified `POST` request that a browser may make after a simple web form is submitted by the user:


```
POST /send HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

user=tim&msg=Hello%2C+World%21
```

You will learn what the above message body means and how it is related to the form shown in the next practical task.

4 Enter the following program:

```
import socket

my_socket = socket.socket()
my_socket.bind(('127.0.0.1', 8000))
my_socket.listen()
new_socket, address = my_socket.accept()
received_data = new_socket.recv(1024)
print(received_data.decode())
new_socket.close()
my_socket.close()
```

This server program simply listens for a connection request on port number 80, creates a connection to the client when a request is detected, and then prints out up to 1024 bytes of the request received from the client on the Python shell window. (Note that this server does NOT send any bytes back to the client – the browser.)

a) Restart the server and visit `http://127.0.0.1:8000/` using a browser. Copy the first line of what the browser sends to the server after connecting.

b) Restart the server and visit `http://127.0.0.1:8000/example` using a browser. Copy the first line of what the browser sends to the server after connecting.

c) Restart the server and visit `http://127.0.0.1:8000/example?q=query` using a browser. Copy the first line of what the browser sends to the server after connecting.

d) Restart the server and visit `http://127.0.0.1:8000/example#fragment` using a browser. Copy the first line of what the browser sends to the server after connecting.

5 The previous server program does not send any bytes back to the client before closing the socket. Which of the following most accurately describes what is displayed on the browser?

A The browser displays a blank page.

B The browser displays a loading animation.

C The browser displays a sequence of random bytes.

D The browser displays an error page.

( )

6 What three pieces of information must the first line of a HTTP request contain?

a) _____

b) _____

c) _____

7 When a URL is entered into the address bar of a web browser, a HTTP request is constructed and sent to the host that is specified in the URL. Predict the request line and host header field portions of the HTTP request for each of the following URLs. An example has been completed for you.

Example

`http://www.example.com/home?hello#world`

```
GET /home?hello HTTP/1.1
Host: www.example.com
```

a) `http://example.net:5000/com?hello=world`

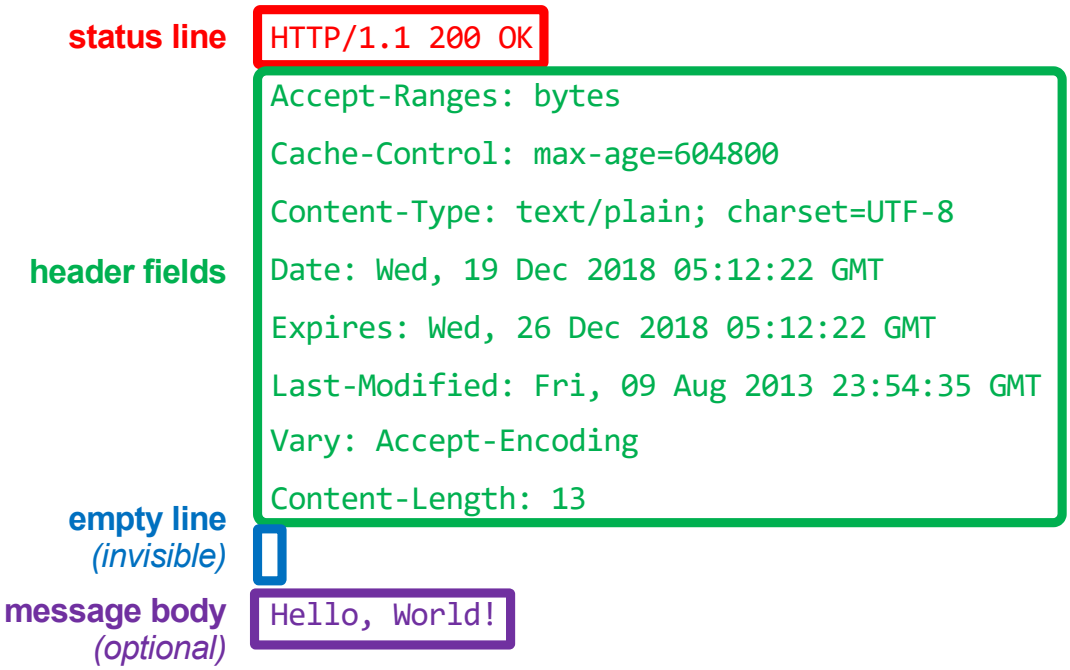
b) `http://sub.example.org/9999?keyword=computing#page-1`

c) `http://www.moe.gov.sg#footer`

HTTP Responses

After a web server receives a HTTP request, it sends back a HTTP response using the same socket. Open Google Chrome's Developer Tools, select the "Network" tab and visit `http://example.com` again. Above the section labelled “Request Headers” should be another section labelled “Response Headers”. Look for a “view source” link next to this label and click on it. (Once again, if the “view source” link is missing, make sure you are using HTTP and not HTTPS.)

HTTP responses have a similar format to HTTP requests. Each response starts with a **status line**, followed by **header fields**, followed by an empty line and an optional **message body**.



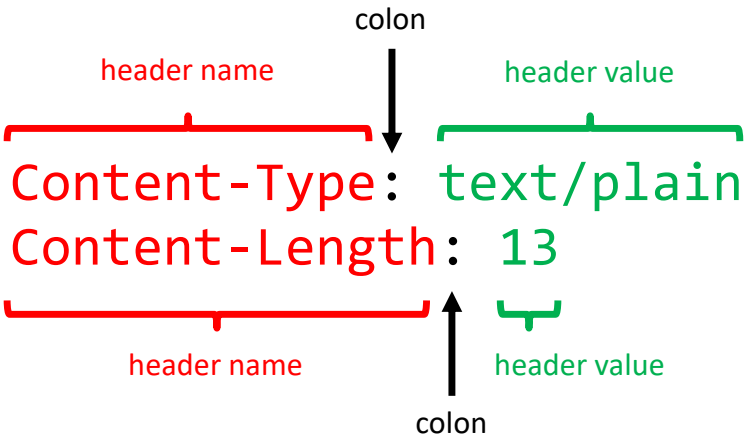
Status Line

The status line summarises if the web server was able to perform the request that was received from the client or web browser. It contains three important pieces of information separated by spaces: the version of HTTP that the server uses, a **status code** and a **reason phrase**. Each status code has a recommended corresponding reason phrase that helps to explain what the status code means. Three common status codes and their corresponding reason phrases are:

Status Code	200	404	500
Reason Phrase	OK	Not Found	Internal Server Error

Header Fields

Like HTTP requests, HTTP responses also have header fields that provide additional information. Each header field has its own line that consists of a name, followed by a colon, followed by a space and then the header field's value.



A typical HTTP response would have a `Content-Type` value to specify how the message body should be interpreted as well as a `Content-Length` value to specify the message body's size in bytes. Some typical values for `Content-Type` value are `text/plain`, `text/html`, `text/css`, `image/png`, `image/gif` and `image/jpg`.

Message Body

Like HTTP requests, the header fields and message body for a HTTP request are separated by an empty line.

The message body will typically contain a document in the format specified by the `Content-Type` header field. For instance, the following is a simplified HTTP response that a browser may make after receiving a request for a HTML document:

```
HTTP/1.1 200 OK                                          Status Line
Host: example.com
Content-Type: text/html                                  Header Fields
Content-Length: 110

<!DOCTYPE html>
<html lang="en">
<head><title>Hello, World!</title></head>                Message Body
<body>Hello, World!</body>
</html>
```

The message body in the above response is an example of a HTML document. You will learn more about HTML in the next practical task.

8 Enter the following program:

```
1  import socket
2
3  response = b'HTTP/1.1 200 OK\r\n'
4  response += b'Content-Type: text/plain\r\n'
5  response += b'Content-Length: 13\r\n'
6  response += b'\r\n'
7  response += b'Hello,\nWorld!'
8
9  my_socket = socket.socket()
10 my_socket.bind(('127.0.0.1', 8000))
11 my_socket.listen()
12 new_socket, address = my_socket.accept()
13 new_socket.sendall(response)
14 input('Press enter to quit: ')
15 new_socket.close()
16 my_socket.close()
```

Unlike the previous server program that did not send anything back to the browser, this server sends back a hard-coded HTTP response.

Notice from lines 3 to 7 that HTTP uses `'\r\n'` for line endings instead of `'\n'` such that every line that comes before the message body (including the empty line) ends with `'\r\n'` instead of `'\n'`. `'\r'` is called the carriage return character and `'\n'` is called the line feed character. Historically, when computers used printers for output, both characters were needed to position the print head correctly for the next line. Some systems such as HTTP still require use of these two characters to denote the end of a line.

However, note that the requirement to use `\r\n` only applies to the headers and does not affect the message body. HTTP transmits the message body without trying to interpret it, so it can use `'\n'` for line endings if desired.

Use a browser to visit `http://127.0.0.1:8000` and copy what is displayed on the browser.

9 What three pieces of information must the first line of a HTTP response contain?

a) _____

b) _____

c) _____