COS 316 Precept #3: What is HTTP?
Overview of HTTP

- **HyperText Transfer Protocol**
  - Used to distribute hypertext over the Internet (i.e., HTML web pages)
  - Relies on a bidirectional stream protocol underneath → **TCP**!

- **Unit of operation: request+response pairs**
  - Establish a connection from client to server
  - Client: send HTTP request to server
  - Server: send HTTP response to client

- **Stateless protocol**
  - No mandatory state maintained beyond a request+response operation
  - Server & client can cooperate to maintain application state, e.g., through cookies

- **Standardized through a series of RFCs**
  → overview of applicable standards
URLs

- **Uniform Resource Locator**
  - uniquely identifies a given resource on the web
- **Syntax:**
  
  \[
  \text{scheme://authority/path?param=val#anchor}
  \]

  - **Scheme:**
    - Specifies *protocol* a client must use to interact with the resource.
    - E.g., *http* or *ftp*
  - **Authority:**
    - Indicates *location* of a given resource in terms of a service, e.g., offered by a server accepting TCP connections. Hostname and port (sometimes omitted).
    - E.g., *princeton.edu:80* or *google.com*
  - **Path:**
    - Indicates *location* of a resource within the scope of the service.
    - E.g., */precepts* or */courses/archive/fall19/cos316*
  - **Parameters:**
    - Encode additional information sent to the server. Behavior depends on the server.
    - E.g., *?mobile=true&lang=es*
  - **Anchor:**
    - Encode additional information for the client (not sent to server).
    - E.g., *#section-assignments*

**Examples:**

- http://xyz.org:8081/route/subroute
- mailto:ak18@cs.princeton.edu
- ftp://tug.ctan.org/pub
- rtsp://192.168.0.164/axis-media/media.amp
HTTP Example

1. Client requests URL:
   http://www.xyz.org:80/path/file

2. Client sends request message:
   GET URL HTTP 1.1
   Host: www.xyz.org:80
   ............

3. Server routes request to the appropriate handler/file

4. Server sends response message:
   HTTP 1.1 200 OK
   ............

5. Client processes response
HTTP Request and Response Messages

<table>
<thead>
<tr>
<th>Message Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank line</td>
</tr>
<tr>
<td>Message Body (optional)</td>
</tr>
</tbody>
</table>
HTTP Request Message

- **Request Message Header:**
  - Request Line
  - Request Headers

- **Blank line**

- **Request Message Body (optional)**


- **Request Line**
  - `[request-method-name] [request-URI] [HTTP-version]`
  - request-method-name: *HTTP verb*
    - GET, HEAD, POST, etc.
  - request-URI:
    - Name of resource (route) requested
  - HTTP-version:
    - HTTP/1.0, HTTP/1.1 or HTTP/2.0

- **Request Header**
  - Consists of name:value pairs
  - Multiple values, separated by commas
  - request-header-name: request-header-value1, request-header-value2, ...

- **Examples**
  
  ```
  Host: www.xyz.com
  Connection: Keep-Alive
  Accept: image/gif, image/jpeg, */*
  Accept-Language: us-en, fr, cn
  ```
HTTP Request Methods (*verbs*)

- **Common methods**
  - **GET**
    - retrieve a resource from the server
  - **HEAD**
    - return only the headers of GET response
  - **POST**
    - create a resource on the server (client sends resource in the request body)
- **Case Sensitive**
## HTTP Request Message

<table>
<thead>
<tr>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://registrar.princeton.edu/course-offerings/course-details?term=1202&amp;courseid=015166">https://registrar.princeton.edu/course-offerings/course-details?term=1202&amp;courseid=015166</a></td>
</tr>
</tbody>
</table>

## HTTP Request Message

```
GET /course-offerings/course-details?term=1202&courseid=015166 HTTP/1.1
Host: registrar.princeton.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-encoding: gzip, deflate, br
```
HTTP Response Message

- **Status Line**
  - [HTTP-version] [status-code] [reason-phrase]
    - HTTP-version: HTTP version used in this session e.g., HTTP/1.0, HTTP/1.1, HTTP/2.0
    - status-code: 3-digit response code
    - reason-phrase: short explanation for status code
    - Common status-code and reason-phrases are
      - "200 OK"
      - "404 Not Found"
  - **Examples**
    - HTTP/1.1 200 OK
    - HTTP/1.0 404 Not Found

- **Response Headers**
  - Multiple values, separated by commas
    - response-header-name: response-header-value1, response-header-value2, ...
  - **Examples**
    - Content-Type: text/html
    - Content-Length: 35
    - Keep-Alive: timeout=15, max=10

- **Response Message Body**
  - Data requested, e.g., HTML+CSS+JavaScript
HTTP/2

- Features
  - is binary, instead of textual
  - is fully *multiplexed*, instead of ordered and blocking
  - can therefore use one connection for parallelism
  - uses header compression to reduce overhead
  - allows servers to “push” responses proactively into client caches

- IETF Standard
  - https://httpwg.org/specs/rfc7540.html

- More on HTTP later in semester
Exercises

- Browser inspection
- CURL (-v)
Building Simple HTTP Servers in Go

1. Write a simple web server which only listens

2. Extend the web server to serve content

3. What’s in an `http.Request`?

4. How do we build a custom Mux?
1. Write a simple web server which only listens

```go
func ListenAndServe(addr string, handler Handler) error
```
2. Extend the web server to serve content

```go
func HandleFunc(pattern string, handler func(ResponseWriter, *Request))
```
3. What's in an `http.Request`?

https://pkg.go.dev/net/http#Request
4. How do we build a custom Mux?