Chapter 9: Concurrent Web Requests

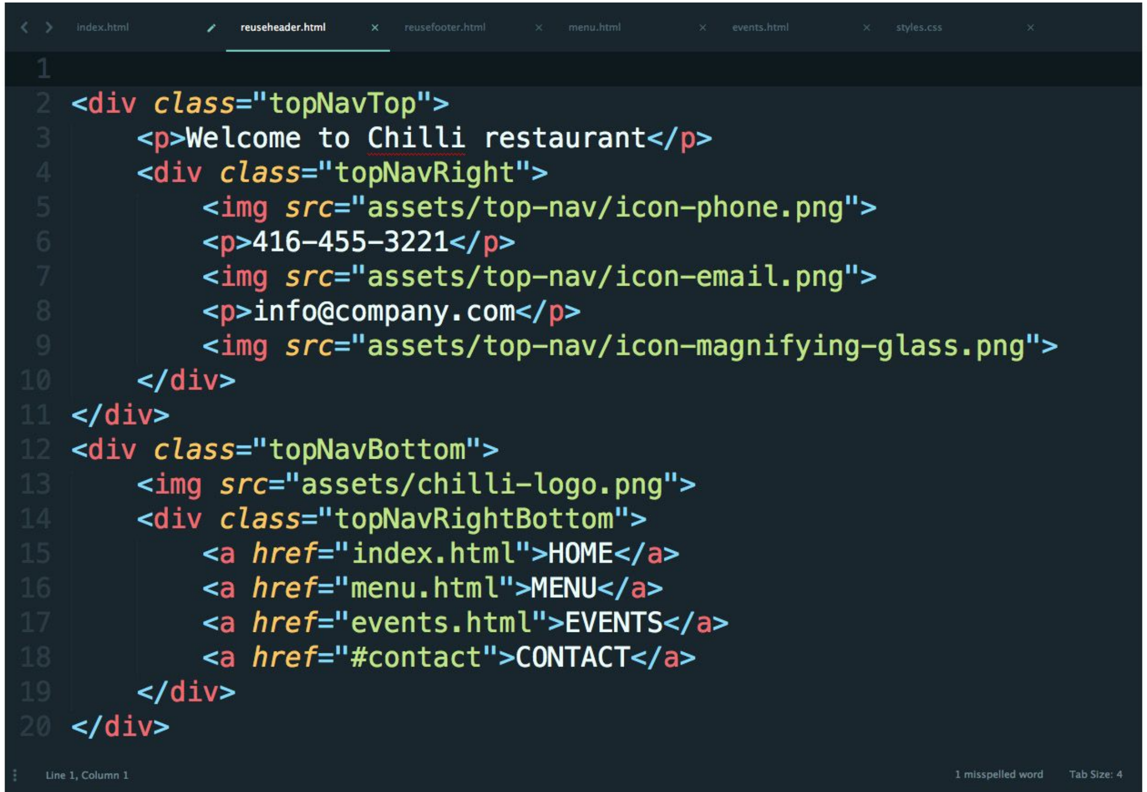
Hanyu Yang

Agenda

- The basics of web requests
- The requests module
- Concurrent web requests
- The problem with timeouts
- Good practices in making web requests
- Case study

The basics of web requests

- HTML
  - Hypertext Markup Language
  - Different tags
    - <p>, <img>, <i>
    - <div class>, <id>

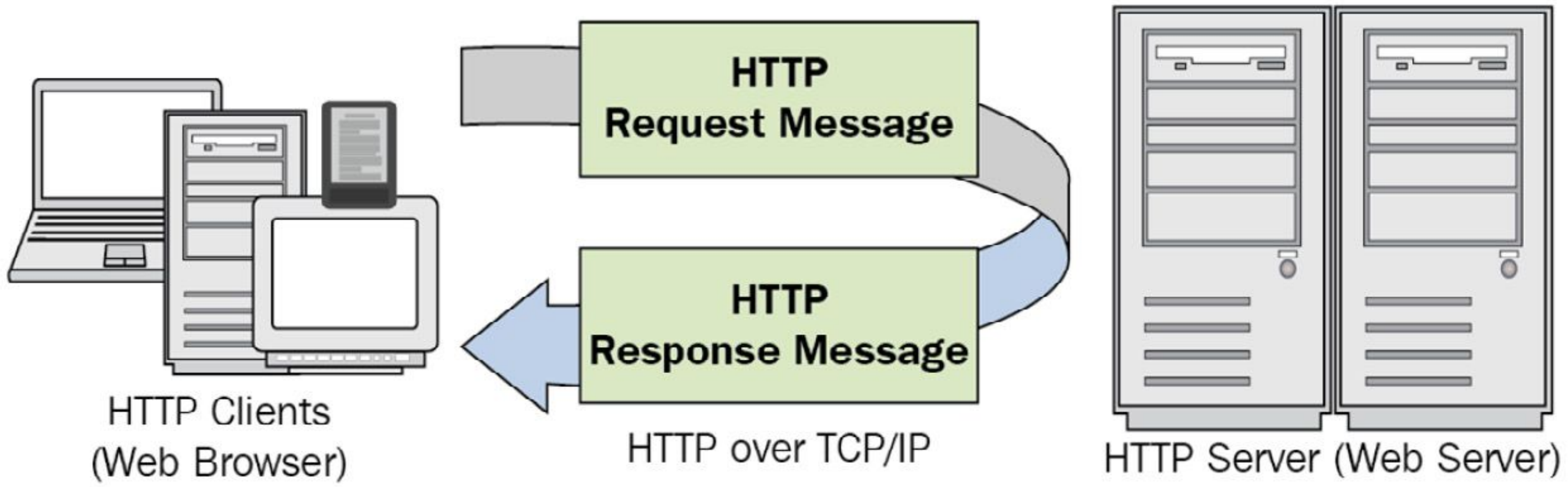


```html

<div class="topNavTop">
    <p>Welcome to Chilli restaurant</p>
    <div class="topNavRight">
        <img src="assets/top-nav/icon-phone.png">
        <p>416-455-3221</p>
        <img src="assets/top-nav/icon-email.png">
        <p>info@company.com</p>
        <img src="assets/top-nav/icon-magnifying-glass.png">
    </div>
</div>
<div class="topNavBottom">
    <img src="assets/chilli-logo.png">
    <div class="topNavRightBottom">
        <a href="index.html">HOME</a>
        <a href="menu.html">MENU</a>
        <a href="events.html">EVENTS</a>
        <a href="#contact">CONTACT</a>
    </div>
</div>
```

The basics of web requests

- HTTP requests
  - Via World Wide Web (WWW), utilizes Hypertext Transfer Protocol (HTTP) protocol
  - GET (headers and body), HEAD (headers), POST (create new resource, server assign URL, non-idempotent), PUT (update or create, client specify URL, idempotent), DELETE



The basics of web requests

- HTTP status code
  - 1xx (informational status code)
    - 100 - header received, waiting for body; 102 - request being processed
  - 2xx (successful status code):
    - 200 - successfully fulfilled; 202 - request accepted, but the processing not complete
  - 3xx (redirectional status code)
    - 300 - multiple options for the response to be processed; 301 - server moved
  - 4xx (error status code for the client)
    - 400 - client bad request; 404 - request not supported by server
  - 5xx (error status code for the server)
    - 500 - internal server error; 504 - gateway timeout

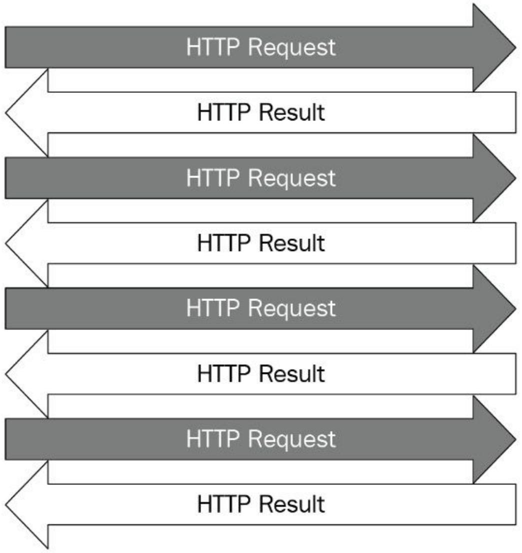
Internet Assigned Numbers Authority (IANA): https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

The requests module

- Making a request in Python
  - example1.py
- Running a ping test
  - example2.py
  - ping httpstat.us

Concurrent web requests

- Spawning multiple threads
  - example3.py
- Refactoring request logic
  - example4.py

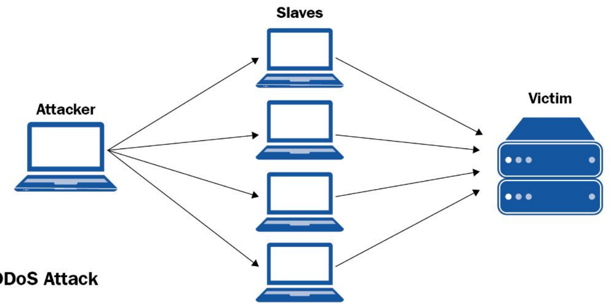


The problem with timeouts

- Support from httpstat.us and simulation in Python
  - example5.py
  - http://httpstat.us/200?sleep=10000
- Timeout specifications
  - example6.py
  - thread.setDaemon(True) run in background
  - otherwise block the main program like thread.join()

Good practices in making web requests

- Consider the terms of service and data-collecting policies
  - https://www.reddit.com/robots.txt
  - https://x.com/robots.txt
  - https://en.wikipedia.org/robots.txt
- Error handling
  - Use try…except block to avoid crashing, failed thread should not crash others
  - Not blind error-catching to cause error swallowing
- Update your program regularly
  - web scraping programs looking for specific tags, have backup, set up failures alert
- Avoid making a large number of requests
  - Distributed Denial of Service (DDoS)
  - Rate/Concurrency limiting



Case study

- Web Scraping tools
    - BeautifulSoup
        - Best for: Simple HTML parsing and small projects.
        - Pros: Lightweight, easy to learn, great for navigating complex HTML structures.
        - Cons: Not suitable for sites with JavaScript-heavy content (no JS execution).
    - Scrapy: https://github.com/scrapy/quotesbot
        - Best for: Large-scale scraping projects, crawling multiple pages, data pipelines.
        - Pros: Built-in support for multi-page crawling, robust data pipelines, and asynchronous requests.
        - Cons: Slightly higher learning curve, overkill for simple projects.
    - Selenium
        - Best for: Scraping JavaScript-heavy sites and interacting with dynamic content.
        - Pros: Simulates real user interaction, handles JavaScript.
        - Cons: Slower than other tools, can be resource-intensive.

Reference: https://chatgpt.com/share/671bbc90-97bc-8003-a5dc-ea08c1af45b0

Case study

- Anti-scraping measures
    - CAPTCHA
    - Rate Limiting
    - IP Blocking
    - User-Agent Filtering
    - JavaScript Rendering
    - Honeypot Traps
    - Session Validation
    - Dynamic Content Delivery
    - Obfuscation
    - Legal Notices

Reference: https://chatgpt.com/share/671bbc90-97bc-8003-a5dc-ea08c1af45b0

Case study

- Bypass Anti-scraping measures (for educational purpose)
    - CAPTCHA - ML tools or services that employ human workers
    - Rate Limiting - randomized request patterns
    - IP Blocking - using proxies like residential proxies or VPNs
    - User-Agent Filtering - Browser automation tools (like Puppeteer or Selenium)
    - JavaScript Rendering - Browser automation tools
    - Honeypot Traps - challenging and need careful testing
    - Session Validation - storing cookies and tokens used for authentication
    - Dynamic Content Delivery - Browser automation tools + Proxies with different location/IPs
    - Obfuscation - handle case by case
    - Legal Notices - seek permission from owner or use for research purpose

Reference: https://chatgpt.com/share/671bbc90-97bc-8003-a5dc-ea08c1af45b0