

# LECTURE 36

## HTTP REQUESTS

MCS 275 Spring 2023

David Dumas

# LECTURE 36: HTTP REQUESTS

Reminders and announcements:

- Project 4 is due 6pm CDT Friday 28 April.
- Prepare for Wednesday: Install `beautifulsoup4` with

```
python3 -m pip install beautifulsoup4
```

# SWITCHING SIDES

Recently, we've talked a lot about making HTTP **servers** in Python (e.g. web applications).

This week we'll switch to talking about Python as an HTTP **client**, parsing HTML, and extracting data (scraping).

# URLS

A **Uniform Resource Locator** or URL specifies the location of a "resource", such as a document, a data file, or a coffee machine.

Basic structure is

```
protocol://hostname[:port]/path/filename?nam=val&nam2=val2
```

Everything after hostname is optional.

Sample URL:

```
https://www.dumas.io/teaching/2023/spring/mcs275/slides/lecture36
```

# DECODING A URL

`https://www.dumas.io/teaching/2023/spring/mcs275/slides/lecture36`

- **Protocol** is HTTPS (which is HTTP over an encrypted connection)
- **Hostname** is `www.dumas.io`
- **Path** is  
`/teaching/2023/spring/mcs275/slides/`
- **Filename** is `lecture36.html`
- No query parameters

# URLLIB

Module `urllib` can retrieve resources from URLs.

E.g., it can open a file if you give it a `file://` URL.

Most often it is used to make HTTP and HTTPS GET requests, to retrieve web pages from web servers and data from HTTP APIs.

`urllib.request.urlopen(url)` retrieves the resource and returns a file-like object

# HTTP RESPONSE

Response consists of a numeric **status code**, some **headers** (an associative array), then a **body** or **payload**.

E.g. GET a web page, the HTML will be in the body.

There are **lots of codes**; first digit gives category:

- 2xx — success
- 3xx — redirection; more action required (e.g. moved)
- 4xx — client error; your request has a problem
- 5xx — server error; cannot handle this valid request

Formal definition of the response structure is in [RFC 2616](#).

# PARTS OF A HTTP RESPONSE

Response to GET `http://example.com/`

```
HTTP/1.1 200 OK
Age: 309829
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Mon, 19 Apr 2021 03:40:44 GMT
Expires: Mon, 26 Apr 2021 03:40:44 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (ord/572F)
Vary: Accept-Encoding
Content-Length: 1256
```

```
<!doctype html>
<html>
<head>
  <title>Example Domain</title>
  .
  .
  .
```



# PARTS OF A HTTP RESPONSE

Response to GET `http://example.com/`

**Status line** with response code

```
HTTP/1.1 200 OK
```

```
Age: 309829
```

```
Cache-Control: max-age=604800
```

```
Content-Type: text/html; charset=UTF-8
```

```
Date: Mon, 19 Apr 2021 03:40:44 GMT
```

```
Expires: Mon, 26 Apr 2021 03:40:44 GMT
```

```
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
```

```
Server: ECS (ord/572F)
```

```
Vary: Accept-Encoding
```

```
Content-Length: 1256
```

**Headers**

"key: value", one per line

Required blank line

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
  <title>Example Domain</title>
```

```
  .  
  .  
  .
```

**Payload**

A sequence of bytes

# URLOPEN RETURN VALUE

`x = urllib.request.urlopen(URL)` returns an object that makes available:

- The status code as `x.status`
- The headers as `x.headers`
- The payload as `x.read()` (or use `x` where a file object is expected)

# HTTP BODY VS HTML BODY

An HTTP request has several parts, the last of which is the body/payload (an array of bytes).

Often, the body is an HTML document.

An HTML document has several parts, one of which is the body (contained in the tag `<body>`).

# GET DATA FROM AN API

Use the [Bored JSON API](#) to get a suggestion of an activity.

```
import json
from urllib.request import urlopen

with urlopen("https://www.boredapi.com/api/activity") as r:
    # treat payload as file, process as JSON
    data = json.load(r)
print("Maybe you could... ", data["activity"])
```

# GET A WEB PAGE

```
from urllib.request import urlopen  
  
with urlopen("https://example.com/") as r:  
    html_bytes = r.read()
```

This gives the body as a `bytes` object (an array of integers in the range 0...255).

If you want a string, you need to know the encoding.

And it might not be HTML! Can check `r.headers.get_content_type()` or `r.headers["content-type"]`.

# GET A WEB PAGE

```
from urllib.request import urlopen

with urlopen("https://example.com/") as r:
    html_bytes = r.read()
    # Determine encoding from Content-Type header
    # (recommended)
    charset = r.headers.get_content_charset()
    html = html_bytes.decode(charset)
```

The encoding is **usually** specified in the Content-Type header, but this is not actually required.

# GET A WEB PAGE

```
from urllib.request import urlopen

with urlopen("https://example.com/") as r:
    html_bytes = r.read()
    # Determine encoding, using utf-8 if the
    # server didn't give a Content-Type header
    charset = r.headers.get_content_charset(failobj="utf-8")
    html = html_bytes.decode(charset)
```

# GETTING DATA FROM THE WEB

HTML is a language for making documents, meant to be displayed to humans. Avoid having programs read HTML if at all possible.

Web pages often contain data that might be useful to a computer program.

The same data is often available in a structured format meant for consumption by programs, e.g. through an API that returns a JSON object.



What do you do if there is no API, and you need to extract information from an HTML document?

Sigh with exasperation, then...

# HTML PARSING

**Level 0:** Treat the HTML document as a string and use search operations (`str.find` or regexes) to locate something you care about, like `<title>`.

HTML is complicated, and this approach is very error-prone.

# HTML PARSING

**Level 1:** Use a **parser** that knows how to recognize start/end tags, attributes, etc., and tell it what to do when it finds them (e.g. call this function...)

`html.parser` is in the standard library.

This approach is event-based. You specify functions to handle things when they are found, but you don't get an overall picture of the entire document.

# HTML PARSING

**Level 2:** Use a higher-level HTML data extraction framework like [Beautiful Soup](#), [Scrapy](#), or [Selenium](#).

These frameworks create a data structure that represents the entire document, supporting various kinds of searching, traversal, and extraction.

# REFERENCES

- [The urllib documentation](#)
- [Examples of using urllib.request](#)
- [Beautiful Soup home page](#)
- [MCS 260 Fall 2020 Lecture 34 - Requesting URLs in Python](#)
- [A list of some public APIs](#)

# REVISION HISTORY

- 2022-04-20 Last year's lecture on this topic finalized
- 2023-04-17 Updated for 2023

