Intermediate Python

Unit 3: Introduction to Flask

Overview

- What is Flask?
- Why are we learning this?
- The Internet, aka "the web"
- Writing Flask applications

What is Flask?

Flask is a Python library.

Flask is a library that lets us create **web applications** and **websites/webpages**.

Flask is a web framework.

Flask helps us build web servers.

Flask helps us build web servers that power our web applications.

A note on terminology

The terms **web application**, **website**, and **webpage** are all interchangeable and refer to a website that is accessed with a web browser.

Why are we learning this?

Why are we learning this?

Much of our world is powered by the web.

Why are we learning this?

Even when we're not browsing *the web* on our *browsers*, we're likely on the web.

Why are we learning this?

Everything is connected to the web: your phone, your watch, even your fridge might even be connected to the web.

Why are we learning this?

But the primary use of the web is still the usage of webpages, and this is what we'll be learning about.

Why are we learning this?

Being able to create programs that rely on *the web* or *networking* is an important part of being a software engineer.

The Internet

What is The Internet?

The Internet is a global network of billions of computers and electronic devices that are able to talk to each other.

Talking to each other

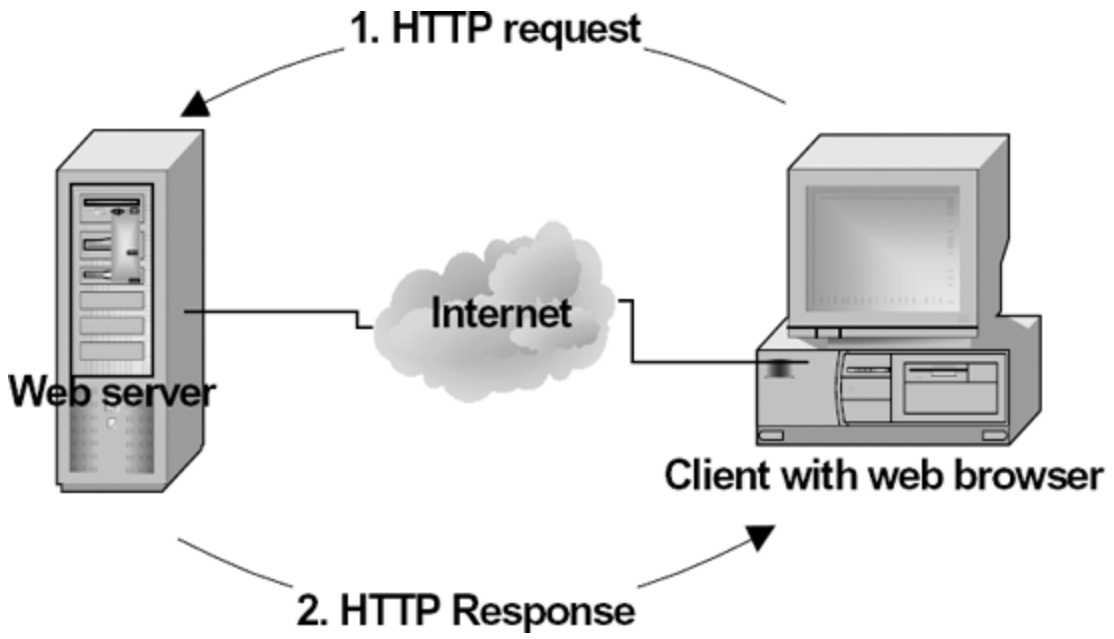
What is meant by "talking to each other" is simply the act of sending and receiving messages.

Talking to each other

The first computer sends a **request** for some data and the second computer **responds** to the request.

Terminology

- **Request**: a message sent by a computer, the sender, to another computer, the receiver.
- **Response**: a response to a message sent back from the receiver to the sender.


1. HTTP request
Internet
Web server
Client with web browser
2. HTTP Response

Let's jump into the code

Sample Flask application

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Let's break this down

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Imports

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Using imported code

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

__name__

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Creating an application

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Running an application

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Functions

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Whatever our function returns will be the response sent back to the client.

Whatever our function returns will be what is displayed in our browser.

Decorators

```
import flask

app = flask.Flask(__name__)

@app.get("/")
def index():
    return "Hello, world"

app.run()
```

Decorators

Decorators allow us to add functionality to out functions.

Templates

Routes can return HTML

```
@app.get("/")
def index():
    return """
        <!DOCTYPE html>
        <html>
            <head>
                <title>Project: Recipe book</title>
            </head>
            <body>
                <h1>Recipe Book</h1>
                <h2>Contents</h2>

                ...
    """
```

Routes can return HTML

But this can be cumbersome due to the length of the content.

Templates

Flask provides a function named `render_template` that lets us move our HTML code into separate files.

Contents of `templates/index.html`

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Contents of `application.py`

```
from flask import Flask, render_template

# ...

@app.get("/")
def index():
    return render_template("index.html")

# ...
```

Templates

This makes working with HTML easier because it's no longer a string in our Python code.

Templates

Flask will look for your templates in the `templates`.

Templates

Flask templates use a library called Jinja2.

Jinja2

Jinja2 offers functionality that lets you embed variables in your HTML code.

Embedding variables

Embedded variables must be wrapped in `{{ }}`.

For example, `{{ name }}`.

Contents of `templates/index.html`

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Hello {{ name }}!</h1>
    </body>
</html>
```

Contents of `application.py`

```
# ...

@app.get("/")
def index():
    return render_template("index.html", name="Marcos")

# ...
```

Keyword arguments

When you call a function in Python and pass an argument to it, you can specify the name of the argument.

Keyword arguments, an example

```
def print_greeting(name):
    print("Hello " + name)

print_greeting("Ahmed")
print_greeting(name="Cindy")

name_to_greet = "Janira"
print_greeting(name=name_to_greet)
```

Jinja2

Jinja2 also lets you embed code (that looks a lot like regular Python code) in your templates.

Embedding code

Embedding code must be wrapped in `{% %}`.

For example, `{% for driver in driver_scores %}`.

Contents of `application.py`

```
# ...

driver_scores = {
    "Max Verstappen": 454,
    "Charles Leclerc": 308,
    "Sergio Perez": 305,
    "George Russell": 275,
    "Carlos Sainz": 246
}

@app.route("/")
def index_route():
    return render_template("index.html", driver_scores=driver_scores)

# ...
```

Contents of `templates/index.html`

```
<body>
    {% for driver, score in driver_scores.items() %}
        {% if score > 300 %}
        <div class="winning">{{ driver }}: {{ score }}</div>
        {% else %}
        <div class="losing">{{ driver }}: {{ score }}</div>
        {% endif %}
    {% endfor %}
</body>
```

Embedded code samples

```
{% for name in student_list %}
  {{ name }}
{% endfor %}

{% if score > 100 %}
   Wow you're amazing!
{% elif score > 90 %}
   You're getting an A!
{% else %}
   Keep at it!
{% endif %}
```

URLs and routing

URLs and routing

Let's breakdown what URLs are, how they work, and how they are used to navigate **to** and **within** our Flask applications.

Let's start with definitions

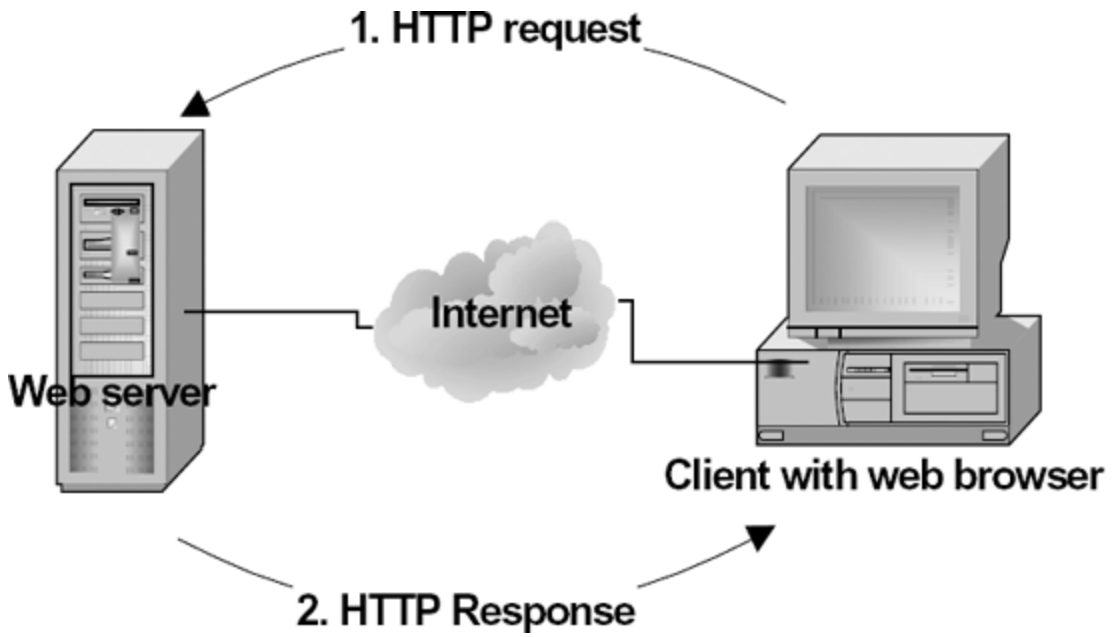
URL: an acronym for Uniform Resource Locator, URLs are the "address" of a resource (a webpage, a video, a photo, etc.) This resource can be in our own computer, or on another computer.

URLs are addresses

URLs are addresses and they help us navigate The Internet to find and access a resource.

URLs are addresses

By typing a URL into our browser's address bar, we send a request to the web server asking for what we need and it will response with the image/video/HTML/etc. that we asked for.

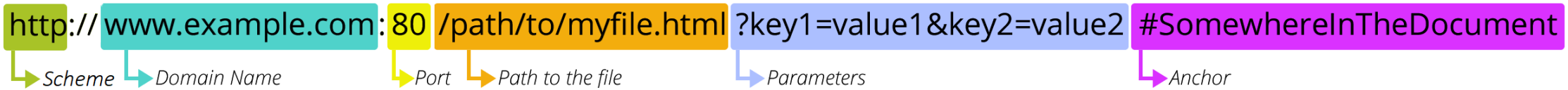

1. HTTP request
Internet
Web server
Client with web browser
2. HTTP Response

A web server is a program that is able to accept these requests and response appropriately.

Our Flask applications are web servers.

Example URLs

- https://www.google.com
- https://www.youtube.com/watch?v=Z1RJmh_OqeA
- https://en.wikipedia.org/wiki/Computer_programming#Programming_languages
- https://upload.wikimedia.org/wikipedia/commons/d/df/The_Fabs.JPG


http://
www.example.com
:80
/path/to/myfile.html
?key1=value1&key2=value2
#SomewhereInTheDocument
Scheme
Domain Name
Port
Path to the file
Parameters
Anchor

Scheme

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

---

The scheme indicates the protocol that must be used when talking to the server. This of a protocol as the "language" that must be used.

Domain

http://**www.example.com**:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

---

The domain is the address for the web server that we are trying to reach.

Domain (IP address)

http://**159.89.240.57**:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

Since a domain corresponds to an IP address, An IP address may be used in place of the domain.

Domain (local IP address)

http://**127.0.0.1**:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

---

`127.0.0.1` is the IP address for your local computer.

Domain (localhost)

http://**localhost**:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

---

`localhost` is a special domain that corresponds to your local computer as well.

Port

http://www.example.com:**80**/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument

---

The port indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. [1]

Path

http://www.example.com:80**/path/to/myfile.html**?key1=value1&key2=value2#SomewhereInTheDocument

---

The path corresponds to the path or route of the resource on the web server.

Path

http://www.example.com:80**/path/to/myfile.html**?key1=value1&key2=value2#SomewhereInTheDocument

---

In Flask, this is what we use `@app.route` for.

Path

http://www.example.com:80**/path/to/myfile.html**?key1=value1&key2=value2#SomewhereInTheDocument

http://www.example.com:80**/more**?key1=value1&key2=value2#SomewhereInTheDocument

---

Paths may have multiple parts, each separated by a forward slash.

Path

http://www.example.com:80/?key1=value1&key2=value2#SomewhereInTheDocument

---

`/` is the default path. When you see a URL without a path, it'll default to this. This path is referred to as the "index" path.

Parameters

http://www.example.com:80/path/to/myfile.html**?key1=value1&key2=value2**#SomewhereInTheDocument

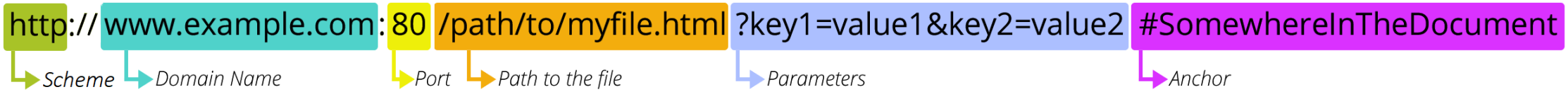
---

These are extra parameters (information) that is provided to the web server. These parameters are a list of key / value pairs (like a dictionary in Python) separated by `&`.

Anchor

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2**#SomewhereInTheDocument**

This is an anchor to a section in the webpage returned by the web server. This is used by browsers to scroll right to that section in the webpage.



- The **scheme** is used to determine which language to use when talking with the web server.
- The **domain name** is used to reach the web server.
- The **port** is used to pick the correct entry into the web server.
- The **path** and **parameters** are for the web server to use for whatever it wants.
- The **anchor** is used by the browser to scroll to the correct position.

How do URLs relate to Flask applications?

- The **scheme**, **domain name**, and **port** are used to reach the web server.
- The **path** is used by the web server (your Flask application) to determine what action it should perform and how it should respond.

Links

Links

In HTML, links are created with the Anchor tag (`a`) and have an `href` attribute with the URL we would like to link to.

```
<a href="https://google.com">Click here to go to Google.com</a>
```

Href attribute

You can use any valid URL in the `href` attribute.

```
<a href="https://www.google.com/search?q=Python">Click here to go to search for "Python"</a>
```

Absolute URLs

When a URL includes all of the usual parts (scheme, domain, port, path, etc.), it is referred to as an *absolute URL*.

`https://www.google.com/search?q=Python`

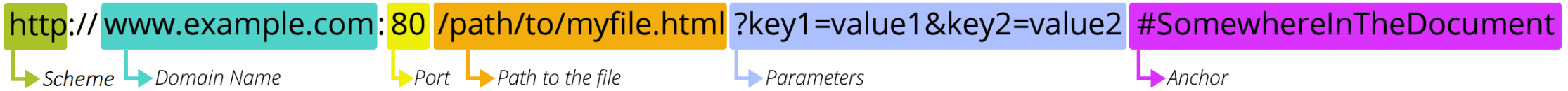
Relative URLs

URLs that only include the path, query, and anchor are referred to as *relative URLs*.

`/search?q=Python`

Relative URLs

/search?q=Python



Relative URLs

The parts of a relative URL that are left out (like the domain), are taken from the existing webpage that you are on.

Relative URLs

This means that when you are on `https://google.com`, when a user clicks on a link for `/search?q=Python` they will be taken to `https://google.com/search?q=Python`.

Absolute vs. relative

- Absolute: `https://www.google.com/search?q=Python`
- Relative: `/search?q=Python`

URLs in anchor tags

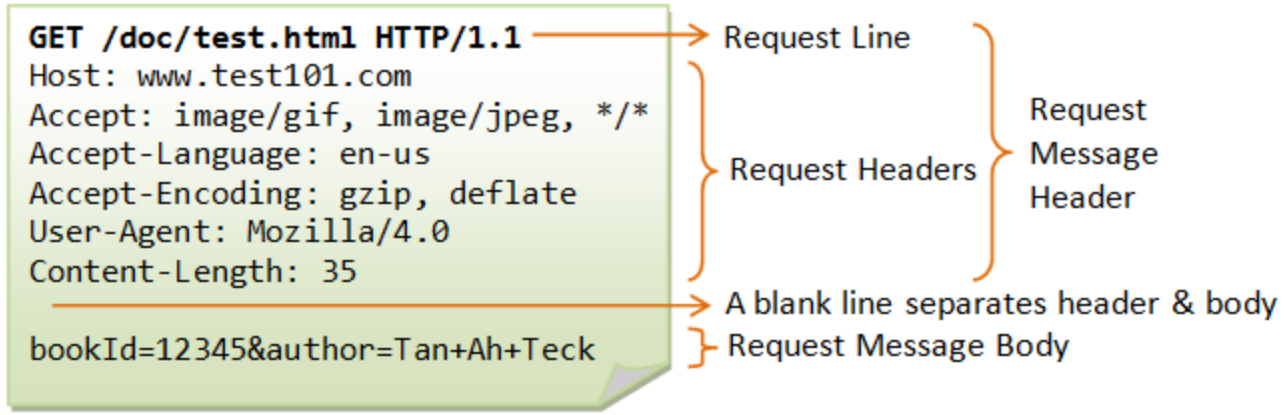
You can use both absolute and relative URLs in anchor tags.

```
<a href="https://www.google.com/search?q=Python">Search Google</a>

<a href="/search?q=Python">Search Google</a>
```

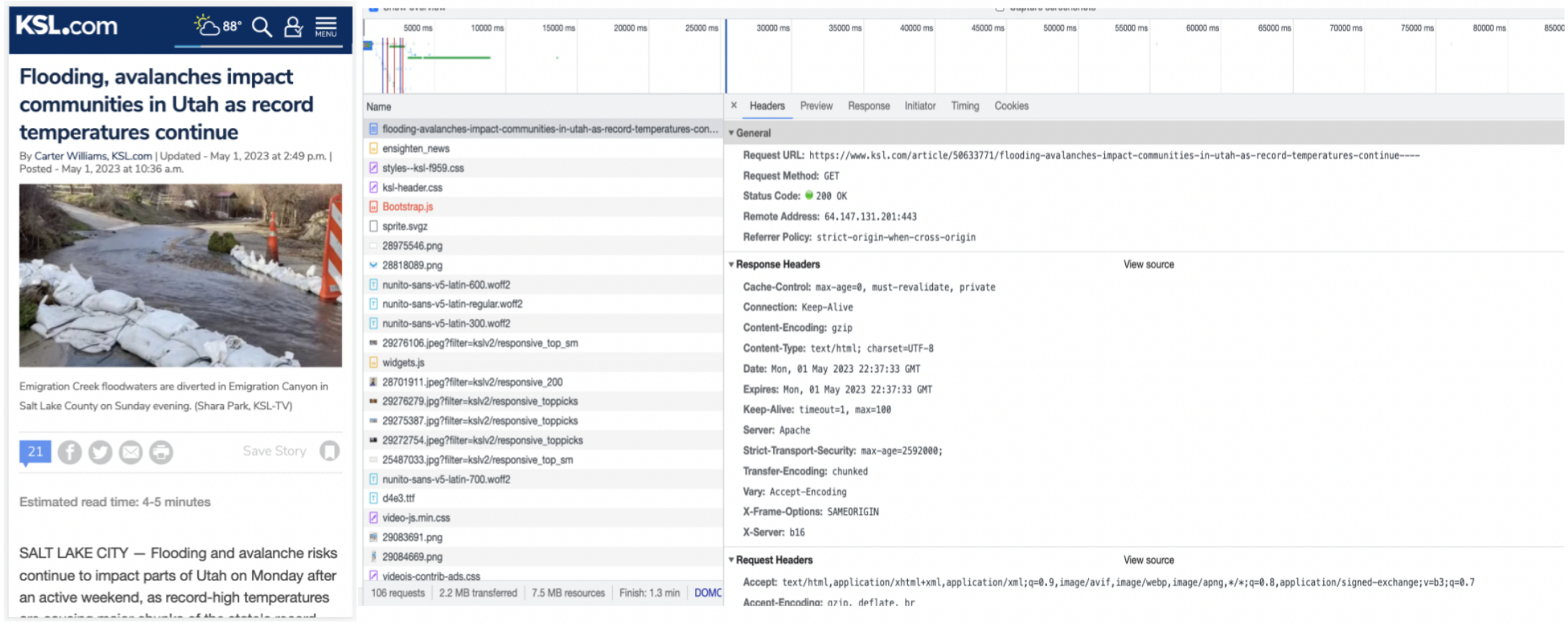
Query Parameters and Request Class In Flask

A Quick Review of URL's And Schemes

What Is HTTP/HTTPS?



Website Speak In HTTP



In Flask, We Access HTTP Requests With The `request` Class

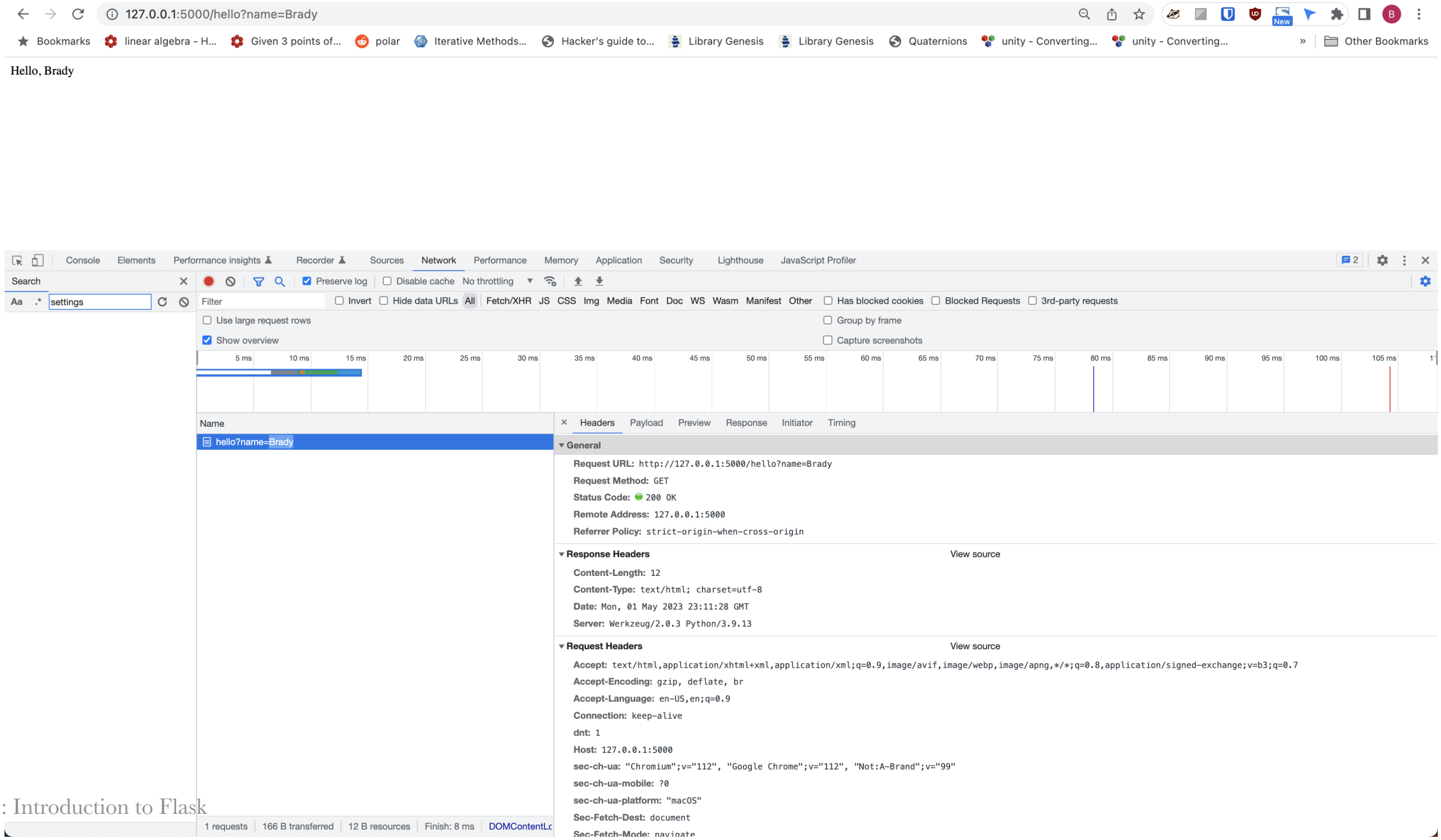
`http://127.0.0.1:5000/hello?name=Brady`

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/hello')
def hello():
        name = request.args.get('name')
        return "Hello, " + name

app.run()
```

Notice that request.args is a dictionary! Remember those?

```
my_dictionary = {"name": "Brady"}
```

Request looks like this



Working With Multiple Arguments

http://127.0.0.1:5000/hello?name=Brady&age=31&city=Salt+Lake+City

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/hello')
def hello():
        return "Here are all your args:" + str(request.args)

app.run()
```

Looks similar to

```
my_dictionary = {"name": "Brady", "age": 31, "city": "Salt Lake City"}
```

Handling User Input with Query Parameters

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/greet')
def greet():
        name = request.args.get('name')
        if name:
                return "Hello, " + {name}
        else:
                return 'Please enter your name.'

app.run()
```

Query parameters are Often Used to Filter, Sort or Search For Data

```
from flask import Flask, request

app = Flask(__name__)

# A list of data in JSON format
people = [
    {"name": "Brady", "age": 31},
    {"name": "Marcos", "age": 31},
    {"name": "Nephi", "age": "??"},
]

# A route that filters the data by a 'name' query parameter
@app.route('/people')
def filter_people():
    # Get the 'name' query parameter from the URL
    name = request.args.get('name')

    if name is None:
        return str(people)

    # Filter the data by name, if name is provided, using a loop
    filtered_people = []
    for person in people:
        if person['name'] == name:
            filtered_people.append(person)


    return str(filtered_people)

app.run(debug=True)

# http://localhost:5000/people?name=John
```